# Local Distributed Algorithms for Multi-Robot Systems

by

Alejandro Cornejo

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012



Author .....
Department of Electrical Engineering and Computer Science
September 28, 2012

Certified by .....
Nancy Lynch
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie Kolodziejski
Chairman, Department Committee on Graduate Students

# Local Distributed Algorithms for Multi-Robot Systems

by

Alejandro Cornejo

Submitted to the Department of Electrical Engineering and Computer Science
on September 28, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

## Abstract

The field of swarm robotics focuses on controlling large populations of simple robots to accomplish tasks more effectively than what is possible using a single robot. This thesis develops distributed algorithms tailored for multi-robot systems with large populations. Specifically we focus on local distributed algorithms since their performance depends primarily on local parameters on the system and are guaranteed to scale with the number of robots in the system.

The first part of this thesis considers and solves the problem of finding a trajectory for each robot which is guaranteed to preserve the connectivity of the communication graph, and when feasible it also guarantees the robots advance towards a goal defined by an arbitrary motion planner. We also describe how to extend our proposed approach to preserve the $k$-connectivity of the communication graph. Finally, we show how our connectivity-preserving algorithm can be combined with standard averaging procedures to yield a provably correct flocking algorithm.

The second part of this thesis considers and solves the problem of having each robot localize an arbitrary subset of robots in a multi-robot system relying only on sensors at each robot that measure the angle, relative to the orientation of each robot, towards neighboring robots in the communication graph. We propose a distributed localization algorithm that computes the relative orientations and relative positions, up to scale, of an arbitrary subset of robots. For the case when the robots move in between rounds we show how to use odometry information to compute at each robot the relative positions complete with scale, of an arbitrary subset of robots. Finally we describe how to use the proposed localization algorithm to design a variety of multi-robot tasks.

Thesis Supervisor: Nancy Lynch
Title: Professor of Electrical Engineering and Computer Science

# Acknowledgments

First I would like to thank my advisor Nancy Lynch, for her support, advice, and encouragement, not only during the writing of this thesis, but overall during my time at MIT. From Nancy I have learned how to think as a researcher and how to communicate ideas with clarity and precision. I would also like to thank Erik Demaine, Leslie Kaelbling and Daniela Rus for sitting on my thesis committee and providing valuable feedback on this work.

A lot of the work on this thesis was a result of different collaborations with Fabian Kuhn, James McLurkin and Majid Khabbazian, whom I would like to thank as well.

I also thank Prasant Gopal and Mohsen Ghaffari, my current office mates, for making my time at the office more pleasant and for being an endless source of conversations and discussions, both work and mostly non-work related.

Finally, I would like to dedicate this thesis to my wife, Patty, who has endured with me this entire process and made me feel like its all worth it, and to my parents, who have always supported me and encouraged me to pursue my dreams.

# Contents

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

Many tasks that are suited for mobile robots can be accomplished far more effectively with multi-robot systems, in particular systems with large numbers of robots. Search and rescue, exploration and mapping, security and surveillance, and even construction are all ideal potential applications for large multi-robot systems. Disaster relief workers could use a swarm of robots to locate victims, biologists could use a swarm to study an ecosystem, and the military could use a swarm for surveillance and security.

These applications have challenging requirements: the robots must be highly mobile, they must maintain a communication network across a large geographical area, they must estimate their own physical configuration (as well as properties of the environment), and they must coordinate to make collective decisions. In order for algorithms to achieve these requirements they must operate at the intersection of physical mobility, communication networking, and distributed computation. In this work we aim to develop robust and practical algorithms for large multi-robot systems with a rigorous theoretical underpinning.

For an algorithm to become practical for multi-robot systems with large populations it needs to overcome a number of challenges. First, to leverage the size of the population, it becomes paramount to use decentralized strategies that allow robots to operate (mostly) independently while collectively making progress towards a global (task-dependent) goal. Second, since the expected number of failures grows together with the size of the system, it is crucial for algorithms to be robust to individual robot failures, and no single robot (or small group of robots) should be critical to task performance. With this in mind in this work we focus our attention on *local distributed algorithms*—distributed algorithms that run for a constant number of communication rounds—since their performance depends primarily on local

parameters and they are guaranteed to scale well with the size of the network.

This thesis will contain new algorithmic techniques for multi-robot systems that combine aspects of distributed algorithms, computational geometry, graph theory and robotics. We describe a model of computation at a level of abstraction that is suitable to design provably correct distributed algorithms, and at the same time accurately represents the physical world. Instead of designing algorithms for a particular application of multi-robot systems (i.e., search and rescue, construction, etc.), this work focuses on providing solutions to general problems that form the base for almost all canonical multi-robot system applications. In particular, we study the problems of *connectivity* and *localization*, which we briefly motivate and sketch in the paragraphs below.

**Connectivity.** The connectivity of the communication graph in a multi-robot system is the property that enables coordination. This means that to successfully execute a task that requires coordination, the robots must ensure the communication graph is connected. Therefore, algorithm designers for large multi-robot platforms often seek two seemingly contradicting properties. On one hand, to maximize the parallelism and enhance the performance of the system, when a robot is performing an individual task it should do so as independently from the rest of the multi-robot system as possible. On the other hand, to ensure that tasks which require coordination are not stalled indefinitely thereby preventing completion, at all times robots should make sure their motions do not disconnect the communication graph. This work seeks to alleviate this tension by providing a local distributed algorithm that mediates between the desired motion of the individual robots, and the requirement of preserving connectivity of the communication graph.

**Localization.** Most tasks that are well suited for multi-robot systems rely on the ability of the robots to control their motion in the environment and with respect to each other. Therefore, it should not come as a surprise that to perform even the most trivial of tasks, robots need some form of orientation and position information. In the localization problem each robots seeks to use its sensors to estimate some information about the orientation and position of close-by robots. The quality and difficulty of obtaining these estimates critically depends on the specific sensors available to the robots. For example, sensors such as dual antenna GPS can readily provide accurate absolute estimates for both position and orientation. However GPS is unavailable in many environments, including indoors, in urban canyons, under heavy foliage, underwater or on other planets. As an alternative one could endow

robots with LIDAR* (or other sophisticated sensors), which despite being unable to provide absolute positions, allow robots to recover accurate estimates of the relative orientation and position of close-by robots. Unfortunately, due to cost and size concerns, these sensors are not a real alternative for multi-robot systems with large populations. In this work we study a variant of the localization problem in which robots have minimal sensing capabilities, namely we assume that robots can only sense the angle (in their own private coordinate system) to neighboring robots. We describe local distributed algorithms that rely on these minimal sensing capabilities and allow a robot to recover the relative orientations and positions of other robots, where the relative positions match the ground truth up to a uniform-scaling factor.

The problems of connectivity and localization are fundamentally different, and therefore the techniques we will use to study them are also different in many respects. However, perhaps surprisingly, we will use the same general approach to solve both of these problems. Namely, we will first seek to cast them as graph theoretic and/or geometric problems, and we will use the insights learned by solving these problems to design efficient local distributed solutions.

In the next subsections we give a more detailed description of and motivation for the aforementioned problems, we outline our approach and contributions, and we briefly survey some related work. In the last subsection we list some additional results on algorithms for multi-robot systems, which despite not being directly related to the problems of connectivity or localization, are still relevant to our work.

## 1.1 Connectivity

**Motivation.** Designing efficient and robust algorithms for multi-robot systems can be difficult due to the distributed nature of problems, even when communication is performed through a infrastructure based (single-hop) wireless network. Relying on an ad hoc network for communication only complicates things further—in order to plan its trajectory a robot might need to communicate with other robots on the system, and at the same time the resulting motion might change the topology of the communication network. Since a connected communication graph is necessary to enable coordination, algorithms for multi-robot systems must reconcile the interaction between communication and motion planning in order to preserve a connected graph.

Most existing distributed algorithms for mobile ad hoc networks typically sidestep

*LIDAR stands for Light Detection and Ranging. In a typical LIDAR system a narrow laser beam is used to map physical features with very high resolution.

the issue of connectivity by assuming it is ensured by other means. For example, distributed algorithms for mobile ad hoc networks to solve routing [48, 75], leader election [60], and mutual exclusion [93] assume that the control laws that determine the motion of the robots are determined by a separate mobility layer. These algorithms deal with connectivity by assuming the mobility layer takes care of ensuring that every pair of nodes that need to exchange a message are connected at some instant or transitively through time; otherwise they work on each independent connected cluster. A similar situation is true in the control theory community. For example, work to solve the problems of flocking [77, 40], pattern formation [33], and leader following [13] provide control laws that determine how each robot will move. However, these works sidestep the problem of connectivity by assuming coordination runs atop a network layer that ensures it is always possible to exchange information between every pair of agents (i.e., an infrastructure based wireless network).

In our work we develop a local distributed algorithm which acts as a middleman between these two incompatible approaches, potentially allowing us to execute the flocking algorithm of [77] using the routing algorithm of [75], or running the leader follower algorithm of [13] using the leader election service of [60], with the formal guarantee that a connected communication graph is maintained throughout the execution. Aided by the proposed distributed algorithm, algorithm designers for multi-robot systems can focus on the problems which are specific to the application (i.e., search and rescue, demining fields, exploration, etc.) without having to deal with the additional problems that arise from the lack of a fixed communication infrastructure. We expect this modular approach to algorithm design will be instrumental to develop scalable and provably correct algorithms for multi-robot systems.

**Assumptions and Problem Statement.** We assume a collection of robots deployed in two-dimensional Euclidean space with arbitrary positions and orientations. Associated with the multi-robot system there is an underlying communication graph where two robots are considered neighbors if and only if they are at distance at most $r$ (in other words, the communication graph is a unit disk graph). We consider a synchronous model of computation, where time progresses in synchronous lock-step rounds and at each round robots that are neighbors in the communication graph can reliably exchange messages. Finally, we also assume that each robot is able to compute the relative orientation and the relative position of its neighbors in the communication graph, and can query an internal motion planning module for the immediate desired trajectory for the next time step.

Given the desired trajectory output by the motion planning module, the goal of the connectivity problem is to find an alternative trajectory which gets the robot

as close as possible to the destination of the original trajectory, while at the same time guaranteeing the communication graph remains connected. This allows us to run standard configuration control algorithms (which provide the motion planning module) together with our distributed connectivity algorithm. Since the connectivity problem does not require any information about the long term motion plan, it can be used together with online configuration control algorithms, where the trajectory of the robots is the result of the interaction with other robots and the environment. We describe in detail one such compositional result in Section 1.1.2.

**Approach and Contributions.** We split the connectivity problem in two subproblems. In the first subproblem each robot seeks to locally identify a subset of edges that when preserved guarantee the communication graph remains connected; we refer to such edges as connectivity-preserving edges. A conservative solution, which is far from optimal, would be to identify all edges as connectivity-preserving edges. Such a solution would prevent the multi-robot system from performing any task that requires them to spread apart. Furthermore, in the course of the execution two robots which were initially not neighbors might become neighbors, and if all edges are always preserved this could result in forcing all robots to cluster together. Therefore, we focus on solutions where robots (locally) find a "small" set of connectivity-preserving edges. The challenge here is that connectivity is a global property of the graph, and we seek solutions that rely only on local information. We show how to adapt several known sparse graph constructions in geometric graph theory to design simple local distributed algorithms that can identify a "small" (i.e., constant) set of connectivity-preserving edges. In the process of showing this we describe how to simplify several existing results in the field of topology control [19].

For the second subproblem, we assume each robot has identified a set of edges which it wants to preserve as well as a target position where it wants to move, and the goal is to find a trajectory that preserves the selected edges and simultaneously gets the robot "as close as possible" to its target position. For robustness purposes we focus on strategies that guarantee preserving the selected edges even if robots follow the prescribed trajectory at arbitrary speeds, or if robots suddenly halt after traveling only a fraction of the prescribed trajectory (for example, because the robot encountered an obstacle, it stopped to take a soil sample, its wheels slipped in a wet environment, etc.). We describe a local distributed algorithm to solve this problem, and we prove the trajectories produced by this algorithm are both *safe* (in that they guarantee to preserve the selected edges) and *robust* (in the sense described above).

Next we turn our attention to analyzing the progress guarantees of our proposed solution. Informally speaking, the *progress* of the algorithm measures how much

closer the robots get to their desired position. The progress is a function of both the task being run by the robots (which controls the desired motion plan) as well as the environment in which this task is executed (i.e., initial configuration of the robots, placement of obstacles, etc.). Therefore, to prove a progress bound that is independent of the task and/or the environment, we need to make assumptions on the resulting execution. Informally we assume that the robots do not want to perform a motion that requires disconnecting the graph or breaking a global cycle (observe that these assumptions are not necessary to prove the safety or robustness properties of the algorithm). Under these assumptions, we show that the algorithm guarantees progress at a rate of $\Omega(\min(d, r))$ units per round, where $r$ is the communication radius and $d$ is the smallest (non-zero) distance from a robot to its target. Furthermore, we exhibit executions where no local algorithm can do better than this bound, hence under these conditions the bound is tight and the algorithm is asymptotically optimal. Using this result we then show that all robots get $\varepsilon$-close to their desired target within $O(D/r + n^2 r/\varepsilon)$ rounds, where $n$ is the total number of robots, and $D$ is the total initial distance from the robots to the final target. An early version of this work appeared in [21] and [18].

**Related work.** The problem of preserving a connected communication graph while controlling the motion of the robots has been addressed before, mainly in the control theory community. Most proposed solutions are either centralized or are tailored to preserve a connected graph only while performing specific tasks, for example moving all robots so as to converge to a point. The work in [96] models connectivity as a constrained optimization problem, but as a result the solution is centralized and their approach cannot be translated easily to a distributed setting. Another centralized algorithm for second-order agents is proposed in [83], however it conservatively preserves all edges in the graph. The problem of gathering (rendezvous) all agents to a single point while preserving a connected communication graph is studied in [2, 35, 39, 84]. In [53, 59] the authors study the problem of connected deployment, however they evaluate the performance of their algorithms only through simulations, and they do not prove under which conditions their algorithms solve the problem. In contrast this work focuses on local distributed algorithms with rigorous mathematical proofs of their guarantees.

### 1.1.1 $k$-Connectivity

**Motivation.** As we argued before, the size of the population of a multi-robot system and the expected number of faults in the system grow at the same rate.

Therefore, fault-tolerance is an issue that needs to be tackled by any practical algorithm for large multi-robot systems. The connectivity of a graph is a good estimate of the fault-tolerance of the communication network, since higher connectivity means more robots can fail without disrupting the communication among the rest of the robots. Informally speaking, the connectivity of a graph $G$, denoted by $\kappa(G)$, is the size of the smallest set of vertices whose removal disconnects the graph. Although a complete graph on $n$ vertices cannot be disconnected by removing any subset of vertices, by convention its connectivity is defined to be $n-1$. We say a graph $G$ is $k$-connected if $\kappa(G) \geq k$.

In the previous subsection we considered the problem of designing a local distributed algorithm that allows a multi-robot system to perform an arbitrary task while guaranteeing the communication graph remains connected. Our proposed solution tolerates some inaccuracy in the motion of the individual robots, as well as unpredictable obstacles in the environment (as described by our *robustness* properties). However it does not provide any guarantees on the communication graph in the event that a robot fails and stops sending/forwarding messages. Here we turn our attention to the natural extension of this work to preserve a connected graph in spite of any set of $k$ robots failing. In other words, we describe a local distributed algorithm that allows a multi-robot system to perform an arbitrary task while guaranteeing the communication graph remains $k$-connected.

**Approach and Contributions.** In the same spirit as before, we split the problem in two parts. The second half of the problem is exactly the same as it was before, namely we assume each robot has identified a set of edges that it wants to preserve, as well as a target position where it wants to move, and the goal is to find a trajectory that preserves the selected edges and simultaneously gets the robot "as close as possible" to its target position. For this we can reuse the solution we developed for the original connectivity problem, obtaining the same safety and robustness guarantees.

The first part of the problem is slightly different. Specifically each robot seeks to locally identify a set of edges that when preserved guarantee the communication graph remains $k$-connected, we refer to such edges as $k$-connectivity-preserving edges. As before, we focus on solutions to this problem where the set of edges identified is "small". Unfortunately, existing sparse geometric graph constructions cannot be readily adapted to identify $k$-connectivity-preserving edges as we did for 1-connectivity. We describe To solve the problem of finding a "small" subset of edges that are guaranteed to be $k$-connectivity-preserving edges, we leverage some of our previous results on *local graph traits* [20]. Below we informally sketch what are local graph traits and what are the results which we extract from them.

By a standard locality argument it is possible to show that after running any distributed algorithm for $t$ communication rounds, the knowledge of a robot (aka. process) is limited to learning about all robots at distance at most $t$, as well as the communication links present between these robots (i.e., the $t$-neighborhood of a robot). If we do not restrict either the amount of local computation or the message size, it follows that after $O(diameter(G))$ rounds, every robot can acquire complete knowledge of the communication graph and can compute any predicate about $G$, for example all robots could determine if the graph is $k$-connected after running for $O(diameter(G))$ communication rounds. However, local distributed algorithms can only learn their local neighborhood and are therefore limited to observing local graph traits.

Previously [20] we showed that there does not exist a local graph trait that characterizes a $k$-connected graph. More precisely, we proved that for any constant $k > 0$ there does not exist a local graph trait $(t, T)$ such that a graph $G$ satisfies $(t, T)$ if and only if $G$ is $k$-connected. The same result holds even when considering 1-connected graphs, that is for any $k > 1$ there does not exist a local graph trait $(t, T)$ such that a 1-connected graph $G$ satisfies satisfies $(t, T)$ if and only if $G$ is $k$-connected. These results hold even in the case of unit disk graphs or weakly local graph traits.

Since traits that are satisfied if and only if a graph is $k$-connected do not exist, we instead look for traits that when satisfied imply $k$-connectivity. Specifically, in [20] we described different local graph traits which when fulfilled by a 1-connected graph imply that the graph is $k$-connected. We will use these results on local graph traits to derive local distributed algorithms which allow a robot to identify a "small" set of $k$-connectivity-preserving edges.

Finally, to preserve $k$-connectivity of a multi-robot system we describe how to stitch an algorithm for selecting $k$-connectivity-preserving edges with our previous connectivity preserving algorithms.

**Related Work.** Exploring the relationship between local and global graph properties has already been shown to be a fruitful research direction to prove upper and lower bounds for distributed algorithms on various problems. As an example, the seminal work of Linial [57] describes an elegant construction that uses properties of local $t$-neighborhood graphs to prove that any distributed algorithm that finds a maximal independent set in a cycle must take at at least $\Omega(\log^* n)$ rounds. The study of the relationship of local and global graph properties dates further back. In 1983, Wigderson [95] showed that if a graph is locally $k$-chromatic, then it has a chromatic number of $O(\sqrt{kn})$. Even earlier, in 1952, Dirac [26] proved that if $G$ has

at least three vertices and all nodes have degree at least $n/2$, then $G$ is Hamiltonian.

Most of the previous work on $k$-connectivity is in the field of topology control. Jorgic et al. [49] reported the experimental results of three different distributed algorithms to detect $k$-connectivity on random geometric graphs, but the work lacks any formal guarantees. Czumaj and Zhao [24] presented a greedy centralized algorithm to construct a $k$-connected $t$-spanner with runtime $\tilde{O}(nk)$. Thurimella [88] described a distributed algorithm to identify sparse $k$-connected subgraphs that runs in $O(diameter(G) + \sqrt{n})$ time. Jia et al. [47] described a centralized algorithm to approximate the minimum power assignment while preserving $k$-connectivity. Similarly, Li and Hou [54] describe a distributed algorithm that given a $k$-connected graph finds a $k$-connected spanner.

### 1.1.2 Applications

To validate the proposed solution to the connectivity problem, we study how to use it to simplify the development of multi-robot tasks with formal guarantees. As an example, we consider the problem of flocking.

Informally speaking, flocking (also known as swarming or schooling) is a form of collective behavior where a large number of interacting agents move as a cohesive group in the same general direction. This problem has received a lot of attention studied in the robotics community, particularly in the context of control theory, we refer the interested reader to [71] and references therein for a detailed survey of related work.

One of the main challenges in designing flocking algorithms, is to get the robots to agree on a direction of motion, while simultaneously guaranteeing cohesion. The problem of agreement in a network of agents (also known as consensus in the control theory community) has been studied extensively. For example the work of Jadbabaie et al. [45], Mureau [67] and Ren and Beard [78], deals with agreement in networks of agents with varying topology. Saber and Murray [80, 81] and Saber et al. [82] study the problem of consensus in networks with time delays and varying topology. We will show how these standard agreement procedures based on simple averaging [8], can be used together with our proposed solution to the connectivity problem, to yield flocking algorithms with the formal guarantee that both directional agreement and cohesion are achieved.

## 1.2 Localization

**Motivation.** Its not hard to motivate the localization problem in multi-robot systems, since few are the tasks which can be performed effectively by robots without position and orientation information. Accordingly, sensor-based robot localization has been recognized as one of the fundamental problems in mobile robotics [36]. Nevertheless, most existing work either addresses localization of a single robot, assumes landmarks/anchors[†] on the environment, or requires the use of complex and expensive sensors. Solutions relying on complex and expensive sensors are not feasible for large multi-robot systems, and many environments (more details below) prevent the use of approaches that rely on landmarks/anchors.

For slightly different reasons, localization is also particularly useful for sensor networks. Sensor networks are used to track objects and people, as well as to monitor a variety of ambient conditions, such as temperature, pressure, humidity, noise levels, and so on. Sensor networks are a challenging platform since they are typically very resource constrained, and are expected to operate for long stretches of time using a limited power supply (i.e., batteries). Due to the nature of the applications of sensor networks, the data collected is most useful when the sensor nodes are spatially aware. It is believed [32] that the development of local distributed algorithms for coordinate will enable sensor networks to revolutionize information gathering and processing, both in urban environments and in inhospitable terrain.

GPS (Global Positioning System) was developed precisely to aid in the localization problem, and is now widely used in many applications. However issues such as limited precision, increased power consumption (especially relevant in sensor networks), and availability (due to signal obstruction and multi-path effects), prevent GPS from being used in many relevant real-world applications (for example, any application which requires working indoors or under foliage).

We have argued that in multi-robot systems and sensor network platforms it is often crucial to learn the relative orientations and relative positions of other robots in the network. This allows, for example, for neighboring robots to align themselves or evenly spread out, or for robots (or sensor nodes) to route packets to their closest neighbor, amongst various other things. Note that to perform the operations just described (and many others) robots do not require absolute position and orientation information. In fact, upon closer inspection we can verify these operations do not

[†]Landmark-based localization requires the environment to have a collection of landmarks (either artificial or natural), with known positions and orientations, and which can be accurately sensed by the robots. Sometimes a group of robots endowed with GPS, labelled anchor robots, play the role of landmarks.

depend on the distances between robots, but just their ratio between them (i.e., robot $u$ and $v$ are at twice the distance than robot $u$ and $w$). In the same spirit as the work of O'Kane and LaValle [70, 69], this work describes a local distributed algorithm that allows robots with minimal sensing capabilities to recover such information. Specifically, we consider a very weak sensing platform where robots are only required to be able to measure the angle, with respect to their own orientation, to neighboring robots in the communication graph. Angle measurements can be provided by simple and inexpensive sensors, and are readily available in low cost multi-robot platforms [61, 43]. The cost of having such a weak sensing platform, is that it becomes impossible to recover the actual distances (angles are invariant to uniform-scaling), but still allows us to recover the relative orientations and the relative positions up to some constant scaling factor.

**Assumptions and Problem Statement.** As before, we assume a collection of robots deployed in two-dimensional Euclidean space with arbitrary (and unknown) positions and orientations. Associated with the multi-robot system there is an underlying communication graph which describes the neighbor relation between robots (this graph need not be a unit disk graph, and can be an arbitrary undirected graph). We consider a synchronous model of computation, where time progresses in synchronous lock-step rounds and at each round robots which are neighbors in the communication graph can reliably exchange messages. Finally, we also assume that robots are equipped with sensors that enable them to measure the angle (with respect to their own orientation) of every incoming message from a neighboring robots. We remark these assumptions are compatible with what is available in several low-cost multi-robot platforms [61, 43].

In this work we study a variation of the localization problem where each robot seeks to compute the relative orientations and relative positions of a subset of the robots (for example, its neighbors), where the relative positions are correct up to an unknown positive uniform-scaling factor. Although the positions and orientations recovered are not absolute, they are sufficient to compute many commonly used geometric structures (i.e., shortest path between two robots, minimum spanning tree, relative neighbor graphs, etc.). In Section 1.2.1 we outline some applications of our proposed solution to the localization problem.

**Approach and Contributions.** To tackle this problem we take our standard modular approach. First we study the problem from a graph-theoretic/geometric perspective, and then we use the solution to these problems to derive a local distributed algorithm.

We first develop an orientation agreement procedure which allows a robot to compute the relative heading of any other robot in the network from which it has received a message (either directly or indirectly). This procedure allows a robot to compute the orientation of any other robot at a hop distance at most $k$ using only $k$ communication rounds.

Next, we leverage the orientation agreement procedure to reduce the problem of computing the relative positions of all robots, to that of finding a satisfying *realization* of an *angle-constrained* graph. Informally speaking, an angle-constrained graph is an undirected graph which has associated with each of its edges an angle (or direction). A satisfying realization of an angle-constrained graph is a straight-line embedding of this graph to the Euclidean plane, where the angles of the edges in the embedding match those of the angle constraints. In the same spirit of the work in rigidity theory, we describe a simple algebraic characterization of angle-constrained graphs. We extend this characterization to consider satisfying realizations of angle-constrained graphs which only pertain a subset of the robots in the system.

We leverage the algebraic characterization of angle-constrained graphs to develop a local distributed algorithm that allows a robot to compute the relative positions (up to a positive uniform-scaling) of any subset of robots. Moreover, we prove that this algorithm is optimal in the sense that if it is unable to compute the relative positions of a subset of robots, then no other algorithm which runs in the same number of communication rounds can succeed.

Finally we discuss how, when available, odometry information at the robots can be incorporated into the framework to easily obtain the scale of the relative positions.

**Related Work.** Angle sensors have been used together with other sensors to solve the localization problem. For example, Basu et al. [6] studies the problem of localization assuming nodes have noisy length and angle measurements/constraints. The work of Dogancay [27] studies localization with a static observer and a moving target to which the observer can measure a angle, which is analogous to solving a triangulation. In a similar vein, Niculescu and Nath [68] consider a system where nodes can determine the angle to their neighbors, and a subset of the nodes have global positioning capabilities, which is also a variant of triangulation. Approaches relying only on distance information [12, 44, 1] to triangulate the positions have also been proposed. We remark that in all the works above an additional source of length or position information is required (on all or some of the nodes) for successful localization, and therefore the problems they consider are fundamentally different from the ones we study.

For a detailed history of the results in rigidity theory we refer the interested

reader to [94, 23, 3, 16] and references therein. The most relevant to our work is the seminal work of Whiteley [94], who studied rigidity with directional constraints (aka Parallel Drawings in the Plane) using the tools of matroid theory. Whiteley [94] arrives at a characterization of rigidity through the analysis of statics and stresses of bar frameworks, and then follows the more traditional rigidity theory approach of first-order kinematics, which culminates showing the equivalence of 2-rigidty and rigidity with directional constraints. Our work is complementary to this, and we offer an alternative derivation to tackle the localization problem. Concretely, we present a succinct and self-contained argument that precisely characterizes those graphs with angle-constraints for which it is possible to solve the localization problem. Our characterization relies only on simple observations regarding cycles on a graph and basic graph theoretic and geometric arguments. Another distinction with the traditional work in rigidity theory and localization, is that our end goal is to localize only a subset of robots (which is possible even when its impossible to localize all robots and the entire graph is not rigid).

### 1.2.1 Applications

In the motivation subsection we argued that most applications of multi-robot systems necessitate that each robot has some form of information about the positions and orientations of other robots in the system.

To validate our proposed solution to the localization problem, we describe various natural multi-robot applications which can be implemented when the localization information available to the robots is only correct up to a positive uniform-scaling. These applications include things as simple as having each robot determine which of its neighbors is closest to it, or as complex as having each robot compute the shape of the Voronoi cell associated with its position (assuming the Voronoi tessellation has been defined over the position of all the robots). Concretely, we describe how to implement various of the distributed algorithms to compute structures such as the Gabriel Graph, the Relative Neighbor Graph, the Cone-Based Topology Control Graph and the Local Minimum Spanning Graph.

For multi-robot applications that involve motion, we describe how to leverage the odometry information (when available) at each robot to compute the relative positions of any subset of robots using only angle measuring sensors. This allows us to implement any multi-robot task which requires the relative orientations and the relative positions of a subset of robots. These tasks include, amongst others, flocking and distributed coverage control.

## 1.3 Additional Related Work

In the previous subsections we mentioned a few works which were most directly related to the problems of connectivity and localization. In this section we give a wider overview of the previous work on large multi-robot systems, both from the perspective of distributed algorithms, and from the robotics community, from which we draw inspiration for this thesis.

### 1.3.1 Distributed Computing

During the last few years, the distributed computing community has proposed a number of computation models for multi-robot systems [87, 86, 34, 15]. The primary motivation has been to study the minimal set of capabilities required by a collection of distributed robots to solve a certain task, and as such the robots considered are relatively simple and weak. Specifically, the robots considered are assumed to be; *dimensionless*, modeled as points which do not obstruct each others movements or sensing; *oblivious*, unable to remember previous actions or store any state; *anonymous*, indistinguishable and unable to identify any other robots. Moreover, robots are assumed to have no explicit means of communication. Instead, robots communicate implicitly by observing the positions (in their private coordinate system) of all robots in their visibility range, and controlling their own motion. It is assumed that both the observation and motion are carried out with perfect precision. Each robot executes the same algorithm in look-compute-move cycles, where each robot first observes the positions of other robots within its visibility range, then it computes a target position based solely on the observed positions (recall robots are oblivious), and finally it moves to its target position.

Variants of this model have been considered which use different assumptions on the synchronization of the look-compute-move cycles of different robots [85, 86, 34] (namely, fully synchronous, semi synchronous or asynchronous), different visibility assumptions (i.e., unlimited visibility vs visibility graph) and geometric assumptions [34, 29, 76] (i.e., share a compass, share the notion of unit distance, share a coordinate system, etc.). The problems studied in these models include the formation of pre-agreed geometric patterns [86], gathering and convergence [87] (also known as rendezvous), following a pre-designated leader, the wakeup problem (where one initially awake robot must wake up all others), and partitioning [30, 29] (where robots must divide themselves into groups).

The models described above are especially well-suited to study the difficulties that arise when trying to break the symmetry of a system under various synchrony

and geometric assumptions. However, they are less than ideal to design practical distributed algorithms for real multi-robot systems. On one hand these models impose very stringent constraints on the robots, making it impossible for a robot to have a name or unique identifier, to keep a single bit of state, or to communicate anything besides its own position. On the other hand, on the most part, the algorithms developed in these models assume all robots are perfectly accurate and reliable, and therefore ignore the possibility of robot crashing, having a motion error (perhaps due to unknown obstacles or terrain features), or even having a limited sensing precision. As it is observed by [30], a problem which can be shown to be unsolvable in this model, can admit a trivial solution if we relax just one of these assumptions, for example by allowing robots to have unique identifiers.

### 1.3.2 Multi-Robot Systems

As noted by Parker [72], the field of distributed robotics had its origins in the late 1980's. Before then, robotics research had mostly concentrated on single robot systems. The term distributed robotics has been used to refer to a variety of problems that arise in robotics, which includes work on coordination of multiple manipulators, cellular/reconfigurable robot systems [37], etc. We refer the reader to [72] for a detailed (but outdated) survey of the work distributed robot systems and an application taxonomy. In this thesis we are only concerned with multi-robot systems which involve large number of autonomous agents operating in two-dimensional environments that communication through a wireless ad hoc network, and we are only concerned with applications which require some degree of cooperation between the robots.

Several encouraging experimental results with large multi-robot systems have been produced in recent years by the robotics community; the work in this thesis is inspired by them. Perhaps one the best examples is the work of McLurkin [62], who successfully built a swarm of 100+ robots (aka. SwarmBots) and demonstrated a wide range of behaviours in the lab. These behaviours included navigation, dispersion [64], follow-the-leader, gathering and physical sorting of the robots, amongst other things. Although the flavor of that work was mostly experimental, it relied extensively on strategies developed in the field of distributed computing to implement behaviours that would scale well to large populations of robots. More elaborate behaviours tested using the SwarmBot platform included a boundary detection algorithm [63], which was inspired by the work of Edelsbrunner et al. [28] in computational geometry.

Other examples of large multi-robot systems include the CentiBots developed

by Konolige et al. [52], the e-puck developed by Mondada et al. [65], the AmigaBot developed by Howard et al. [42], amongst others. Each of these platforms was tailored for different purposes, and each of the has been used to successfully demonstrate different multi-robot behaviors in the lab.

More than 10 years ago [11] observed that there were few real-world applications of cooperative multi-robot systems that had been reported, and supporting theory is was still its formative stages. We remark that, despite the encouraging experimental results mentioned above, and the fact that there is (and has been) a large pool of potential applications for large multi-robot systems, for the most part the observations made by Cao et al. [11] are still valid today. This fact attests to the difficulty of designing practical and robust algorithms for cooperative multi-robot systems.

Large cooperative multi-robot systems pose unique algorithmic and practical challenges, and we believe that in order to be able to use these systems for real-world applications, it will be necessary to develop a toolbox of robust algorithms for multi-robot systems with a rigorous theoretical underpinning.

# Chapter 2

# Model

This thesis is concerned with multi-robot systems with large populations of mobile robots that communicate via a decentralized wireless network without any preexisting infrastructure (known as a wireless ad hoc network). This chapter describes our assumptions on the communication and the computation capabilities of the robots, as well as our mathematical model for the multi-robot system. Specifically, the first two sections present graph theoretic and geometric definitions that will be used in later chapters. Whenever possible we adhered to the standard notation and terminology, so a reader familiar with these concepts can safely skip the first two sections of this chapter. The last section of this chapter describes the mathematical model we use to model multi-robot systems.

## 2.1 Graph Theory Preliminaries

An *undirected graph* is represented by a tuple $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges, which are two-element subsets of $V$. To simplify notation we will use $V_G$ to refer to the set of vertices of $G$, and $E_G$ to refer to the set of edges of $G$. Whenever it is clear from context that $v$ is a vertex and $e$ is an edge, we will simply use $v \in G$ and $e \in G$ to denote that $v$ is contained in the vertex set of $G$ and $e$ is contained in the edge set of $G$ respectively. The *neighbors* of a vertex $u$ in $G$ are the set of vertices that are connected to $u$ through an edge in $E_G$. We use $N_G(u) = \{v \mid \{u, v\} \in E_G\}$ to denote the neighbors of $u$ in $G$. The number of neighbors of a vertex $u$ in $G$ is its *degree*, denoted by $d_G(u) = |N_G(u)|$.

A graph $H$ is *spanning* of $G$ (alternatively we will sometimes say $H$ *spans* $G$) if $V_H = V_G$. If $H$ spans $G$ we define the intersection of $H$ and $G$, denoted by $H \cap G$, as the graph with vertex set $V_H = V_G$ and edge set $E_H \cap E_G$. Similarly, we define

the union of $H$ and $G$, denoted by $H \cup G$, as the graph with vertex set $V_H = V_G$ and edge set $E_H \cup E_G$. A graph $H$ is a *subgraph* of $G$ (alternatively we will sometimes say $G$ contains $H$) if $V_H \subseteq V_G$ and $E_H \subseteq E_G$, this is denoted by $H \subseteq G$. A subgraph $H$ of $G$ is an *induced subgraph* of $G$ if $\forall u, v \in V_H$ then $\{u, v\} \in E_H$ if and only if $\{u, v\} \in E_G$. We use $2^G$ to denote the set of all subgraphs of $G$.

A graph $P$ is a *path* if it has a vertex set $V_P = \{v_1, v_2, \ldots, v_k\}$ and edge set $E_P = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}\}$, where the $v_i$ are all distinct (in the literature that allows for paths with repeated vertices, our definition is sometimes referred to as a simple path). The vertices $v_1$ and $v_k$ are the *end points* of $P$, and the vertices $v_2, \ldots, v_{k-1}$ are the *inner* vertices of $P$. The number of edges in a path is its *length*. A *cycle* is a path graph with at least three vertices plus an edge between its end points.

A graph is *connected* if it contains a path between every pair of vertices, otherwise it is *disconnected*. A maximal connected subgraph of $G$ is called a *component* of $G$. The *distance* between $u$ and $v$ in $G$, denoted by $d_G(u, v)$, is defined as the length of the shortest path between $u$ and $v$ contained in $G$; if no such path exists then $d_G(u, v) = \infty$. The *diameter* of a graph $G$, denoted by $diam_G$, is the greatest distance between any two vertices in $G$.

A graph without cycles is a *forest*, and a connected forest is a *tree*.

A *vertex cut* of a graph is a set of vertices whose removal renders the graph disconnected. The *size* of a vertex cut is the number of vertices it contains. A vertex cut is a *minimum vertex cut* if it is a vertex cut of smallest size. The *connectivity* of a graph $G$, denoted by $\kappa_G$, is the size of a minimum vertex cut of $G$. A complete graph on $n$ vertices has no cuts at all, but by convention its connectivity is $n - 1$. We say a graph $G$ is *$k$-connected* if $\kappa_G \geq k$.

## 2.2 Geometry Preliminaries

We will be interested exclusively in Euclidean geometry, which is defined mathematically as a real vector space equipped with an inner product. For a positive integer $n$ we use $\mathbb{R}^n$ to denote the $n$-dimensional real vector space. We use bold lowercase letters to denote elements of a vector space, and lowercase letters to denote real scalars. A vector $\mathbf{p} \in \mathbb{R}^n$ is described by an $n$-tuple $(p_1, p_2, \ldots, p_n)$ of real numbers. We define the standard vector space operations and the inner product on $\mathbb{R}^n$:

$$\mathbf{p} \pm \mathbf{q} = (p_1 \pm q_1, p_2 \pm q_2, \ldots, p_n \pm q_n)$$
$$\alpha\mathbf{p} = (\alpha p_1, \alpha p_2, \ldots, \alpha p_n)$$
$$\mathbf{p} \cdot \mathbf{q} = \sum_{i=1}^{n} p_i q_i.$$

The inner product allows us to define the *norm* (or *length*) of a vector $\mathbf{p} \in \mathbb{R}^n$ as $\|\mathbf{p}\| = \sqrt{\mathbf{p} \cdot \mathbf{p}}$. We highlight that this norm satisfies the *triangle inequality*, namely $\forall \mathbf{p}, \mathbf{q} \in \mathbb{R}^n$ we have $\|\mathbf{p} + \mathbf{q}\| \leq \|\mathbf{p}\| + \|\mathbf{q}\|$.

Using the norm we can define the *distance* between two points $\mathbf{p}, \mathbf{q} \in \mathbb{R}^n$ as $d(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|$. We define the *interior angle* between three points $\mathbf{p}, \mathbf{q}, \mathbf{r} \in \mathbb{R}^n$ as $\angle \mathbf{pqr} = \cos^{-1}\left(\frac{(\mathbf{p}-\mathbf{q})\cdot(\mathbf{r}-\mathbf{q})}{\|\mathbf{p}-\mathbf{q}\|\|\mathbf{r}-\mathbf{q}\|}\right)$.

Consider two points $\mathbf{p}, \mathbf{q} \in \mathbb{R}^n$, the *line* that passes through these points is defined as $line(\mathbf{p}, \mathbf{q}) = \{(1-t)\mathbf{p} + t\mathbf{q} \mid t \in \mathbb{R}\}$, the *ray* with origin at $\mathbf{p}$ that passes through $\mathbf{q}$ is defined as $ray(\mathbf{p}, \mathbf{q}) = \{(1-t)\mathbf{p} + t\mathbf{q} \mid t \in \mathbb{R}_{\geq 0}\}$, and the *line segment* between $\mathbf{p}$ and $\mathbf{q}$ is defined as $seg(\mathbf{p}, \mathbf{q}) = \{(1-t)\mathbf{p} + t\mathbf{q} | t \in [0, 1]\}$.

A *hyperplane* is a flat subset of $\mathbb{R}^n$ of dimension $n-1$ that separates the space into two half-spaces. Specifically, the hyperplane with a *normal* $\mathbf{n} \in \mathbb{R}^n$ that contains a point $\mathbf{p} \in \mathbb{R}^n$ is defined as $hyperplane(\mathbf{n}, \mathbf{p}) = \{\mathbf{q} \in \mathbb{R}^n \mid \mathbf{n} \cdot (\mathbf{q} - \mathbf{p}) = 0\}$. The *reflection* of a point on a hyperplane maps the point to its "mirror image" in the hyperplane. Formally the reflection of a point $\mathbf{q} \in \mathbb{R}^n$ in $hyperplane(\mathbf{n}, \mathbf{p})$ is defined as $\text{Refl}_{\mathbf{n},\mathbf{p}}(\mathbf{q}) = \mathbf{q} - 2\frac{\mathbf{n}}{\|\mathbf{n}\|^2}(\mathbf{n} \cdot (\mathbf{q} - \mathbf{p}))$.

A *sphere* of radius $r$ centered at $\mathbf{q} \in \mathbb{R}^n$ is the set of points at distance $r$ from $\mathbf{q}$, denoted by $S_r(\mathbf{q}) = \{\mathbf{p} \in \mathbb{R}^n \mid \|\mathbf{p} - \mathbf{q}\| = r\}$. A ball is the space enclosed by a sphere, the ball is closed if it includes the sphere and the ball is open if it does not include the sphere. Formally, a *closed ball* of radius $r$ centered at $\mathbf{q} \in \mathbb{R}^n$ is the set of points at distance less or equal than $r$ from $\mathbf{q}$, denoted by $B_r[\mathbf{q}] = \{\mathbf{p} \in \mathbb{R}^n \mid \|\mathbf{p} - \mathbf{q}\| \leq r\}$. An *open ball* of radius $r$ centered at $\mathbf{q} \in \mathbb{R}^n$ is the set of points at distance less than $r$ from $\mathbf{q}$, denoted by $B_r(\mathbf{q}) = \{\mathbf{p} \in \mathbb{R}^n \mid \|\mathbf{p} - \mathbf{q}\| < r\}$.

For succinctness we use the term ball to mean a closed ball, and when denoting a unit sphere or a unit ball we omit the $r$ subscript. Moreover, we use the term *disk* to refer to a two-dimensional ball, and the term *circle* to refer to a two-dimensional sphere. Similarly a one-dimensional ball is simply a line segment, and a one-dimensional sphere is simply two points (i.e., the endpoints of a line segment).

The intersection (if any) of two $n$-dimensional spheres is an $(n-1)$-dimensional sphere (and therefore lies in a hyperplane in $\mathbb{R}^n$). The intersection (if any) of two $n$-dimensional balls is an $n$-dimensional *lens*. If two balls have the same radius

the lens produced by their intersection is a *symmetric lens*. The *base of the lens* produced by the intersection of two balls $B_r[\mathbf{p}]$ and $B_{r'}[\mathbf{q}]$ is the space enclosed by the intersection of the spheres $S_r(\mathbf{p})$ and $S_{r'}(\mathbf{q})$. Since by definition the intersection of two $n$-dimensional spheres is an $(n-1)$-dimensional sphere, and the space enclosed by a sphere is a ball, then it follows that the base of an $n$-dimensional lens is an $(n-1)$-dimensional ball.

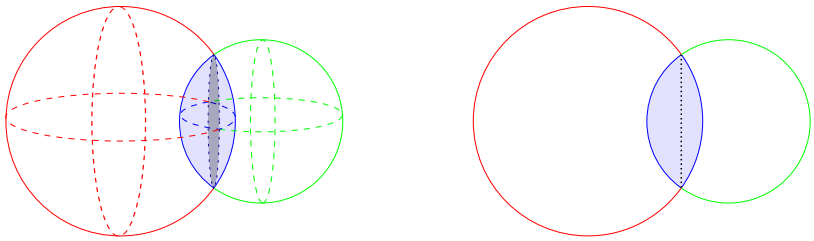


Figure 2-1: The lens produced by the intersection of the red ball and the green ball is colored in blue. The base of the lens is outlined by a black dotted line. If the balls that produced the lens are three-dimensional (left), the base of the lens is a two-dimensional ball (i.e., a disk). If the balls that produce the lens are two-dimensional (right), then the base of the lens is a one-dimensional ball (i.e., a line segment).

A *cone* in $\mathbb{R}^n$ is the union of all rays that originate at the *apex* $\in \mathbb{R}^n$ and pass through a *base* $\subseteq \mathbb{R}^n$. The *axis* of a cone is the ray (if any) that originates at the apex, goes through the base, and about which the cone has rotational symmetry. The *aperture* of a cone is the maximum angle between two rays of the cone which originate at its apex. A *right circular cone* in $\mathbb{R}^n$ is a cone whose base is an $(n-1)$-dimensional ball and whose axis passes at a right angle through the center of the base. All cones considered in this thesis are right circular, and in a slight abuse of notation we use the term cone to mean right circular cone.

A set of points is *convex* if and only if it contains the line segment between every pair of points inside it, formally a set $S \subseteq \mathbb{R}^n$ is convex if and only if $\forall \mathbf{p}, \mathbf{q} \in S, seg(\mathbf{p}, \mathbf{q}) \subseteq S$. A useful property of convex sets is that they are closed under intersection, in other words the intersection of an arbitrary collection of convex sets is also a convex set.

## 2.3 Modeling a Multi-Robot System

In this thesis we deal exclusively with mobile robots deployed in planar environments, and the model described in this section reflects this restriction. In practice this

assumption is reasonable for most multi-robot systems composed of ground vehicles. Nevertheless we remark that most of the algorithms and results presented in this thesis have natural extensions to three-dimensional space. In the pertinent sections we outline how each result can be extended to handle three-dimensional Euclidean space, or what are the obstacles that prevent a natural extension.

Loosely speaking, each robot is modeled by its state in the physical world and a program which controls its behavior. In the following paragraphs we formalize these concepts.

We denote by $R$ the set of *robot identifiers*. To model the program which controls the behavior of each robot we use the TIOA framework [51]. Specifically, we suppose there is a function *prog* which maps each robot identifier in $R$ to a timed I/O automaton which controls its behavior.

The *position* of robot $v \in R$ is described by the function $\varrho_v : \mathbb{R}_{\geq 0} \to \mathbb{R}^2$ which associates with robot $v$ a two-dimensional coordinate in a global coordinate system at every time point. We assume that two distinct robots are never mapped to the same position at the same time. The *orientation* (often called heading or bearing) of robot $v \in R$ is described by the function $\phi_v : \mathbb{R}_{\geq 0} \to [0, 2\pi)$ which associates with robot $v$ its counter-clockwise angle from the $x$-axis of the global coordinate system at each time point. Therefore a robot with an orientation of 0 radians points in the direction of the $x$-axis and a robot with an orientation of $\frac{\pi}{2}$ radians points in the direction of the $y$-axis. Finally the *pose* or *kinematic state* of a robot $v \in R$ at time $t \in \mathbb{R}_{\geq 0}$ is described by its position and orientation at time $t$, that is $pose_v(t) = [\varrho_v(t), \phi_v(t)]$.

The communication between the robots of the multi-robot system is modeled as an undirected graph where each vertex is occupied by a robot. Specifically the *communication graph* at time $t \in \mathbb{R}_{\geq 0}$ is denoted by $G_t$, and is an undirected graph with a vertex set $V$ and an edge set which (potentially) depends on $t$. When executing a *communication step* at time $t$, a robot $v \in V$ first broadcasts a message $m$ which is delivered to its neighbors in $G_t$ and then receives the messages which were broadcast by its neighbors in $G_t$. We remark that in reality the communication occurs via a wireless ad hoc network and, depending on the physical characteristics of the communication medium and the communication devices, directed links might exist between some robots (i.e., a robot $u$ can receive a message sent by robot $v$, but not vice versa). However for simplicity we assume no such links exist, which in practice can be accomplished by adding a preprocessing stage that detects and removes them.

We consider a *round based* model where time progresses in synchronous lock-step rounds of a fixed length. For simplicity and without loss of generality, we assume that rounds occur at integer time points $\{0, 1, 2, \ldots\}$. At the beginning of every round the following actions occur at each robot automaton instantaneously: i) An input action

is received from its sensors (if any). This action usually contains some information about the physical state (for example, the position of neighboring robots). ii) A constant number $\ell$ of communication steps are executed. iii) An output action is produced which includes a motion trajectory. In the remainder of each round, the state of the world is updated and the pose of each robot in the system is evolved using the trajectory output by its automaton.

In reality the actions executed by a robot automaton at the beginning of each round may not occur instantaneously. The amount of time required to execute them depends on the number $\ell$ of communication steps per round, as well as the processing speed and communication bandwidth available to the robots. Since the last two quantities are generally large when compared to the physical speed of the robots, even for moderately large values of $\ell$ it is reasonable to assume the robots remain static for the duration of a round. Moreover, even in systems where this is not the case (either due to severe communication or computational constraints, or due to arbitrarily large values of $\ell$), it is possible to artificially limit the physical speed of the robots so as to preserve the above guarantees.

When no confusion can arise, we will drop the reference to time from our notation and we use $G$ to denote the current communication graph of the multi-robot system, and for $v \in R$ we use $\varrho_v$, $\phi_v$ and $pose_v$ to refer to the current position, the current orientation and the current pose of robot $v$.

In the model presented so far we have not specified what is the initial knowledge of the robots, what sensors are available to the robots to perceive the world and each other, and what is the relationship between the edges present in the communication graph and the position of the robots. These details are specified in the chapters that require them, and for now we can consider that robots initially know nothing (not even their own identifiers or the size of the multi-robot system), they have no sensors, and the relationship between the communication graph and the position of the robots is arbitrary.

The model described above to be simple enough to allow us to state and prove correctness theorems about a multi-robot system, and at the same time general enough to capture the reality of multi-robot platforms for large populations [43, 61, 65].

# Part I

# Connectivity

Designing distributed configuration control algorithms for multi-robot systems is a difficult task, even when dealing with a single hop network. When communicating through a wireless ad hoc network the problem is further complicated by the need to ensure connectivity as the robots move and the communication graph changes. In this setting, there is a complex interplay between mobility and communication. On one hand the robots need to communicate in order to coordinate and decide where to move, and on the other hand every time a robot moves the communication network is subject to change (sometimes drastically). An additional consideration is that for an algorithm that controls the motion of the robots to be of any practical use, it must tolerate uncertainties in the motion of the robots. For example, differences in the actuators might cause the robots to travel at different speeds, unexpected obstacles might cause a robot to stop abruptly before reaching its target, etc. These difficulties might explain why most existing configuration control algorithms provide coordination but typically sidestep the challenge of ensuring the graph remains connected.

This first half of the thesis is devoted to tackling this problem. We propose a connectivity-preserving algorithm that can be used to maintain a connected (or, more generally, $k$-connected) communication graph while simultaneously advancing towards a goal defined by an arbitrary motion planner. The distributed algorithm we describe is *modular* in that it makes no assumptions about the motion-planning mechanism, *local* in that each robot communicates only with nearby robots and doesn't require any network routing infrastructure, and *memoryless* in that the output at each round does not depend on what happened on previous rounds.

**Problem Formulation.** At its core, the problem of moving while maintaining a connected graph deals with the interplay between the position of the robots and the connectivity of the communication graph. Therefore we first make the relationship between the positions of the robots and the presence of edges in the communication graph explicit. Concretely, we assume the communication graph is a *unit disk graph* of radius $r$, which means there is an edge between robot $u$ and robot $v$ if and only if $d(\varrho_u, \varrho_v) \leq r$. As their name suggests, traditionally unit disk graphs have a radius of one. We keep the radius $r$ as a variable in order to make the relationship between the communication radius and the other parameters of the system explicit. The algorithms we describe operate correctly in a slightly more general class of communication graph. In particular it suffices for the communication graph to *contain* a unit disk graph of radius $r$. In other words, if $d(\varrho_u, \varrho_v) \leq r$ then there is an edge between robot $u$ and $v$, but if $d(\varrho_u, \varrho_v) > r$ then the edge between robot $u$ and $v$ might or might not exist. The key property leveraged by the algorithms, is that if two robots

are at distance at most $r$ then they are guaranteed to be directly connected in the communication graph.

Informally, in the connectivity-preserving problem each robot starts with a desired trajectory (produced by a motion planner) and the goal is to find for each robot a new trajectory such that the following properties are satisfied: 1) *$\varepsilon$-progress*, individually each robot does not move away from its desired position and collectively the system moves $\varepsilon$ closer to its desired configuration, and 2) *robust safety*, the connectivity of the graph is preserved regardless of the speed at which each robot follows the trajectory prescribed to it. We highlight that the progress guarantees we will describe depend on various properties of the desired trajectories, while the safety guarantees will require no assumptions on the desired trajectories.

**Outline.** We take a two step approach to tackle this problem. In Chapter 3 we determine which edges in the communication graph are sufficient to guarantee connectivity (we extend this to $k$-connectivity in Chapter 5). In Chapter 4 we describe how to find a trajectory which preserves the selected edges while maximizing the progress towards a predefined target position. Our approach is modular enough to allow Chapters 3 and 4 to be read in either order.

# Chapter 3

# Selecting Edges

Ultimately our goal is to design a local distributed algorithm which allows each robot to move closer to its desired target position while guaranteeing that the communication graph remains connected. As a first step, this chapter asks and answers the following question (stated informally).

> If robots want to preserve the connectivity of the communication graph, to which of its neighbors should each robot remain connected?

Clearly it suffices for each robot to preserve connectivity to all its neighbors, but this would hinder progress since each robot will be very constrained when deciding where to move. This chapter describes various techniques that allow each robot to select a "good" subset of its neighbors to which to preserve connectivity. We guide this search by the following premises: 1) the fewer neighbors a robot is required to preserve, the less constraints it has when finding a trajectory that preserves them, and 2) when finding a trajectory that preserves connectivity to a set of robots, close-by robots represent lesser constraints than robots which are farther away.

**Roadmap.** Section 3.1 introduces some definitions that allow us to ask (and answer) the previous question formally. Section 3.2 surveys various existing techniques that can be easily adapted for our purposes. Finally Section 3.3 proves the optimality of one of the proposed techniques.

## 3.1 The Edge Selection Problem

Given a connected graph $G = (V, E)$ a set $S \subseteq E$ of edges is *connectivity-preserving* if and only if the graph $H = (V, S)$ is connected.

An edge-selection distributed algorithm locally selects at each robot $v$ a subset $S(v) \subseteq N_G(v)$ of its neighbors. We say an edge is *selected* if either of its end points locally selects the other; otherwise the edge is said to be *unselected*. We further partition the set of selected edges into *consistently selected edges*, those edges that are locally selected by their two endpoints; and *inconsistently selected edges*, those edges that are locally selected by exactly one of their endpoints.

We emphasize that at the expense of an additional communication step, any edge-selection distributed algorithm can be modified to guarantee that all edges which were originally inconsistently selected become unselected or consistently selected.

Therefore our goal is to design an edge-selection distributed algorithm that consistently selects a "good" set of connectivity-preserving edges, formalized next. First, since it is desirable to have fewer neighbors we are looking for the sparsest possible set of connectivity-preserving edges, or in other words a spanning tree of the communication graph. Second, since requiring being connected to close-by robots is preferable to requiring being connected to farther-away robots, then assuming the length of an edge is equal to the distance between its endpoints, we want a spanning tree that minimizes the maximum length of the edges (another reasonable goal would be a spanning tree that minimizes the total length of the edges). In other words, ideally we would like to design an edge-selection distributed algorithm that consistently selects the edges of a minimum spanning tree (defined next) and no other edges.

A *minimum spanning tree* of a weighted graph is defined as a spanning tree with minimum total weight. A minimum spanning tree can be shown to be a connected spanning subgraph that minimizes the maximum weight of the edges. In general, a weighted graph may have multiple distinct minimum spanning trees, but a graph with unique edge weights is guaranteed to have a unique minimum spanning tree. Unique edge weights are not a serious restriction since they can be simulated by leveraging the unique identifiers available to the robots. Namely, if a weighted graph associates with each edge $\{u, v\}$ the weight $w_{\{u,v\}}$, then we associate with each edge $\{u, v\}$ the weight tuple $(w_{\{u,v\}}, \min(id(u), id(v)), \max(id(u), id(v)))$. When computing the minimum spanning tree, instead of comparing the weights of two edges directly, we compare their weight tuples lexicographically. By definition, the weight tuples are unique, and the (unique) minimum spanning tree computed using these weight tuples corresponds to a minimum spanning tree using the original edge weights. Therefore, without loss of generality, in this chapter whenever we consider a minimum spanning

tree of a graph we assume it is unique. Specifically given a weighted graph $G$ we use $MST(G)$ to denote the unique minimum spanning tree of $G$, breaking ties as defined above.

However, with or without unique edge weights, and even when allowing messages of unbounded size and unbounded computational resources to each process, it is known that distributed computation of a minimum spanning tree (or any spanning tree for that matter) requires time proportional to the diameter of the graph [74]. In fact, it has been shown that finding any connected spanning subgraph that approximates the weight of the minimum spanning tree is a task that cannot be performed locally [31]. Hence, instead of trying to find a connected spanning subgraph that approximates the weight of the minimum spanning tree, our aim will be to find a subgraph that contains the minimum spanning tree. The fewer the edges in the subgraph, the better the solution.

**Definition 3.1.** *An edge-selection distributed algorithm solves the* MST-containing *problem if it consistently selects a subgraph which contains the minimum spanning tree.*

## 3.2 Sparse Connectivity-Preserving Sets of Edges

This section considers geometric graphs defined over a point set. In a *complete Euclidean graph* the vertex set is a point set in the Euclidean plane and there is an edge between every pair of points with an associated weight equal to the Euclidean distance between its end points. A geometric graph is simply a subgraph of the complete Euclidean graph.

The *unit disk graph* of a point set $P$, denoted by $UDG(P)$, is a subgraph of the complete Euclidean graph with only the edges of weight less or equal than one (or more generally less than some radius $r$). For a point set $P$ and a point $\mathbf{p} \in P$ we define $P[\mathbf{p}]$ as the set that consists of $\mathbf{p}$ and its neighbors in $UDG(P)$. Formally we let $P[\mathbf{p}] = P \cap B[\mathbf{p}]$ where $B[\mathbf{p}]$ denotes a closed unit ball around $\mathbf{p}$.

The *Euclidean minimum spanning tree* of a point set $P$, denoted by $EMST(P)$, is the minimum spanning tree of the complete Euclidean graph, where ties are broken using unique identifiers as described in Section 3.1.

We start this section by describing various geometric graphs. Specifically we describe the Gabriel graph, the Relative Neighbor graph, the Cone-Based Topology Control graph, and the Local Minimum Spanning graph. We also describe simple edge-selection distributed algorithms that consistently select the edges that belong to the intersection of each of these graphs with the unit disk graph. Moreover, we

show that the Local Minimum Spanning graph contains the Euclidean minimum spanning tree, and is therefore a solution to the MST-containing problem. Later, we leverage this result to show that the Gabriel graph, the Relative Neighbor graph, and the Cone-Based Topology Control graph are also solutions to the MST-containing problem, but the Local Minimum Spanning graph has the fewest edges.

Later in Section 3.3 we will show that no edge-selection distributed algorithm can solve the MST-containing problem and have fewer edges than the Local Minimum Spanning graph.

### 3.2.1 Gabriel graph

One of the earliest constructions for sparse connected spanning subgraphs is the Gabriel graph, which was described and proved to be connected by by K.R. Gabriel et al. [38] in 1969. The *GG-region* between two points $\mathbf{p}$ and $\mathbf{q}$ in the Euclidean plane is the closed disk with the line segment $seg(\mathbf{p}, \mathbf{q})$ as its diameter. The points $\mathbf{p}, \mathbf{q} \in P$ are *GG-neighbors* in $P$ if and only if the GG-region between $\mathbf{p}$ and $\mathbf{q}$ contains no other point in $P$ (see Figure 3-1).


Figure 3-1: (a) $\mathbf{p}$ and $\mathbf{q}$ are GG-neighbors since there is no other point in their GG-region. (b) $\mathbf{p}$ and $\mathbf{q}$ are *not* GG-neighbors since there is a point in their GG-region.

**Definition 3.2.** *The* Gabriel graph *of a point set $P$, denoted by $GG(P)$, has $P$ as its vertex set and all pairs of GG-neighbors in $P$ as its edge set.*

The next claim follows directly from the definition of a GG-neighbor/region.

**Claim 3.3.** *Fix a point set $P$, and two points $\mathbf{p} \in P$ and $\mathbf{q} \in P[\mathbf{p}] \setminus \{\mathbf{p}\}$. Then $\mathbf{p}, \mathbf{q}$ are GG-neighbors in $P$ if and only if $\mathbf{p}, \mathbf{q}$ are GG-neighbors in $P[\mathbf{p}]$.*

This claim allows us to use the following straightforward edge-selection distributed algorithm to consistently select the edges present in the unit disk graph and the Gabriel graph.

The following proposition is an immediate consequence of Claim 3.3.

**Algorithm 1** GG-EdgeSelect for robot at $\mathbf{p}$.

1: broadcast $\mathbf{p}$, and let $X = \{\mathbf{p}\} \cup \{\mathbf{q} \mid \mathbf{q} \text{ was received}\}$
2: locally select $\{\mathbf{q} \mid \mathbf{q} \in X \setminus \{\mathbf{p}\}$ and $\mathbf{p}, \mathbf{q}$ are GG-neighbors in $X\}$

**Proposition 3.4.** *Let $P$ be a point set and let $H(P)$ be the graph induced by the edges which are consistently selected by the* GG-EdgeSelect *algorithm. Then $H(P) = GG(P) \cap UDG(P)$.*

*Proof.* First we claim that $X = P[\mathbf{p}]$. This follows by the assumption that the communication graph is a unit disk graph and that (at line 1) each robot broadcasts its own position and subsequently receives the positions of all its unit disk graph neighbors.

Therefore, (at line 2) the robot at $\mathbf{p}$ locally selects a robot at $\mathbf{q}$ if and only if $\mathbf{q}$ is its unit disk graph neighbors and $\mathbf{p}, \mathbf{q}$ are GG-neighbors in $P[\mathbf{p}]$. Finally, by Claim 3.3 this is equivalent to selecting all its unit disk graph neighbors in $P$ that are also its GG-neighbors in $P$ and the statement follows. □

Moreover, observe that since the GG-neighbor relation is symmetric then it follows that no edge is inconsistently selected.

### 3.2.2 Relative Neighbor graph

In 1980 Godfried Toussaint [89] defined the Relative Neighbor graph as a sparser cousin of the Gabriel graph. The *RN-region* between two points $\mathbf{p}$ and $\mathbf{q}$ is the lens produced by the intersection of two open disks of radius $\|\mathbf{p} - \mathbf{q}\|$ centered at $\mathbf{p}$ and $\mathbf{q}$ respectively. The points $\mathbf{p}, \mathbf{q} \in P$ are *RN-neighbors* in $P$ if and only if the RN-region between $\mathbf{p}$ and $\mathbf{q}$ contains no other point in $P$ (see Figure 3-2).

**Definition 3.5.** *The* Relative Neighbor graph *of a point set $P$, denoted by $RNG(P)$, has $P$ as its vertex set and all pairs of RN-neighbors in $P$ as its edge set.*

Toussaint [89] showed the following relation between the Relative Neighbor graph and the Gabriel graph.

**Proposition 3.6.** *Let $P$ be a point set. Then $RNG(P) \subseteq GG(P)$.*

*Proof.* To prove the statement it suffices to show that if an edge is present in $RNG(P)$ then it is also present in $GG(P)$. We prove the contrapositive of this.

Let $e$ be an edge with endpoints $\mathbf{p}, \mathbf{q} \in P$ that is not present in $GG(P)$. Then by definition of the Gabriel graph there exists a point $\mathbf{w} \in P$ in the GG-region between


(a) (b)

Figure 3-2: (a) $\mathbf{p}$ and $\mathbf{q}$ are RN-neighbors since there is no other point in their RN-region. (b) $\mathbf{p}$ and $\mathbf{q}$ are *not* RN-neighbors since there is a point in their RN-region.

$\mathbf{p}$ and $\mathbf{q}$. This implies that $\mathbf{w}$ is also contained in the RN-region between $\mathbf{p}$ and $\mathbf{q}$, and by definition of the Relative Neighbor graph, $e$ is not present in $RNG(P)$. □

The next claim follows directly from the definition of a RN-neighbor/region.

**Claim 3.7.** *Fix a point set $P$, and two points $\mathbf{p} \in P$ and $\mathbf{q} \in P[\mathbf{p}] \setminus \{\mathbf{p}\}$. Then $\mathbf{p}, \mathbf{q}$ are RN-neighbors in $P$ if and only if $\mathbf{p}, \mathbf{q}$ are RN-neighbors in $P[\mathbf{p}]$.*

Essentially the same edge-selection distributed algorithm (and proof) we used for Gabriel graphs can be used to consistently select the edges present in the unit disk graph and the Relative Neighbor graph.

**Algorithm 2** RNG-EdgeSelect for robot at $\mathbf{p}$.

1: broadcast $\mathbf{p}$, and let $X = \{\mathbf{p}\} \cup \{\mathbf{q} \mid \mathbf{q}$ was received$\}$
2: locally select $\{\mathbf{q} \mid \mathbf{q} \in X \setminus \mathbf{p}$ and $\mathbf{p}, \mathbf{q}$ are RN-neighbors in $X\}$

**Proposition 3.8.** *Let $P$ be a point set and let $H(P)$ be the graph induced by the edges which are consistently selected by the* RNG-EdgeSelect *algorithm. Then $H(P) = RNG(P) \cap UDG(P)$.*

As before, since the RNG-neighbor relation is symmetric then it follows that no edge is inconsistently selected.

### 3.2.3 Cone-Based Topology Control graph

Topology control is a technique used in wireless networks to save energy by reducing the number of active links in the network. The construction described here was

originally proposed for topology control by Bahl et al. [5]. For a point set $P$, an $\alpha \in [0, 2\pi]$, and two points $\mathbf{p}, \mathbf{q} \in P$, we say $\mathbf{p}$ is *$\alpha$-safe* in $P$ with respect to $\mathbf{q}$ if and only if there exists a (right circular) cone with apex at $\mathbf{p}$ and aperture $\alpha$ that contains $\mathbf{q}$ and does not contain a point of $P$ that is closer to $\mathbf{p}$ than $\mathbf{q}$ (see Figure 3-3). The points $\mathbf{p}, \mathbf{q} \in P$ are *$\alpha$-neighbors* in $P$ if and only if $\mathbf{p}$ is $\alpha$-safe in $P$ with respect to $\mathbf{q}$ and vice versa.


Figure 3-3: (a) $\mathbf{p}$ is $\alpha$-safe with respect to $\mathbf{q}$ since there is a cone with apex at $\mathbf{p}$ and aperture $\alpha$ containing $\mathbf{q}$ that does not contain a point that is closer to $\mathbf{p}$ than $\mathbf{q}$. (b) $\mathbf{p}$ is *not* $\alpha$-safe with respect to $\mathbf{q}$ since there does not exist a cone with apex at $\mathbf{p}$ of aperture $\alpha$ containing $\mathbf{q}$ that does not contain another point closer to $\mathbf{p}$.

**Definition 3.9.** *The* Cone-Based Topology Control *graph of a point set $P$ and an angle $\alpha \in [0, 2\pi]$, denoted by $CBTC_\alpha(P)$, has $P$ as its vertex set and all pairs of $\alpha$-neighbors in $P$ as its edge set.*

The parameter $\alpha$ controls the number of edges in the resulting Cone-Based Topology Control graph. For example, consider a point set where no three points are collinear and all edges have distinct lengths. At one extreme if we let $\alpha \to 0$ the resulting Cone-Based Topology Control graph contains all edges. At the other extreme if we let $\alpha \to 2\pi$ then the edges included in the resulting Cone-Based Topology Control graph are a subset of the edges in the Euclidean minimum spanning tree.

Bahl et al. [5] proved that if $\alpha \leq 2\pi/3$ then $CBTC_\alpha(P)$ is connected. A shorter proof of this fact appeared in [19], which in addition showed that if $\alpha \leq 2\pi/3$ then $CBTC_\alpha(P)$ contains the minimum spanning tree. We generalize this in Theorem 3.17, using a proof that follows the same spirit as the proof in [19].

The next claim follows directly from the definition of $\alpha$-safe.

**Claim 3.10.** *Fix a point set $P$, and two points $\mathbf{p} \in P$ and $\mathbf{q} \in P[\mathbf{p}] \setminus \{\mathbf{p}\}$. Then $\mathbf{p}$ is $\alpha$-safe in $P$ with respect to $\mathbf{q}$ if and only if $\mathbf{p}$ is $\alpha$-safe in $P[\mathbf{p}]$ with respect to $\mathbf{q}$.*

We leverage this claim to design the following distributed algorithm.

**Algorithm 3** CBTC$_\alpha$-EdgeSelect for robot at $\mathbf{p}$.

1: broadcast $\mathbf{p}$, and let $X = \{\mathbf{p}\} \cup \{\mathbf{q} \mid \mathbf{q}$ was received$\}$
2: locally select $\{\mathbf{q} \mid \mathbf{q} \in X \setminus \{\mathbf{p}\}$ and $\mathbf{p}$ is $\alpha$-safe in $X$ with respect to $\mathbf{q}\}$

The following proposition is a consequence of Claim 3.10.

**Proposition 3.11.** *Let $P$ be a point set and let $H(P)$ be the graph induced by the edges which are consistently selected by the* CBTC$_\alpha$-EdgeSelect *algorithm. Then $H(P) = CBTC_\alpha(P) \cap UDG(P)$.*

We remark that since the $\alpha$-safe relation is not symmetric, the previous algorithm might select some edges inconsistently. However, as mentioned in the beginning of this chapter, at the expense of one additional communication step this algorithm can be easily modified to unselect all inconsistently selected edges. For completeness we illustrate this with the next algorithm.

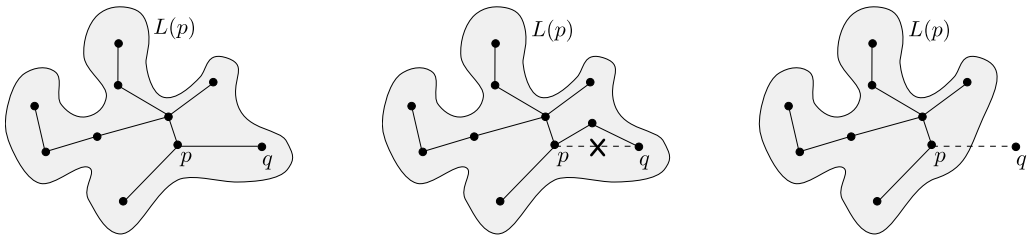
**Algorithm 4** Consistent CBTC$_\alpha$-EdgeSelect for robot at $\mathbf{p}$.

1: broadcast $\mathbf{p}$, and let $X = \{\mathbf{p}\} \cup \{\mathbf{q} \mid \mathbf{q}$ was received$\}$
2: let $Q = \{\mathbf{q} \mid \mathbf{q} \in X \setminus \{\mathbf{p}\}$ and $\mathbf{p}$ is $\alpha$-safe in $X$ with respect to $\mathbf{q}\}$
3: broadcast the set $Q$, and let $Q_q$ be the set received from $\mathbf{q}$
4: locally select $\{\mathbf{q} \mid \mathbf{q} \in Q$ and $\mathbf{p} \in Q_q\}$

### 3.2.4 Local Minimum Spanning graph

Here we describe a generalization of a topology control algorithm first described by Li et al. [55]. For a point set $P$, a function $L : P \to 2^P$ is a *local-region* function on $P$ if it maps every point in $P$ to a subset of $P$ which contains said point. Formally $L$ is a local-region function on $P$ if $\forall \mathbf{p} \in P$ we have $\mathbf{p} \in L(\mathbf{p})$ and $L(\mathbf{p}) \subseteq P$. Some examples of local-region functions, include $L(\mathbf{p}) = \{\mathbf{p}\}$, $L(\mathbf{p}) = P[\mathbf{p}]$ and $L(\mathbf{p}) = P$.

For a point set $P$, a local-region function $L$, and two points $\mathbf{p}, \mathbf{q} \in P$ we say $\mathbf{p}$ is *$L$-safe* in $P$ with respect to $\mathbf{q}$ if and only if either $\mathbf{q} \notin L(\mathbf{p})$ or the Euclidean minimum spanning tree of $L(\mathbf{p}) \cap P$ contains the edge between $\mathbf{p}$ and $\mathbf{q}$ (see Figure 3-4). Two points $\mathbf{p}, \mathbf{q} \in P$ are *$L$-neighbors* in $P$ if and only if $\mathbf{p}$ is L-safe in $P$ with respect to $\mathbf{q}$ and vice versa.




(a) (b) (c)

Figure 3-4: (a) **p** is L-safe with respect to **q** since the Euclidean minimum spanning tree of $L(\mathbf{p})$ includes the edge between **p** and **q**. (b) **p** is not L-safe with respect to **q** since the Euclidean minimum spanning tree of $L(\mathbf{p})$ *does not* include the edge between **p** and **q**. (c) **p** is L-safe with respect to **q** since **q** is not contained in $L(\mathbf{p})$.

**Definition 3.12.** *The* Local Minimum Spanning graph *of a point set $P$ and a local-region function $L$ on $P$, denoted by $LMSG_L(P)$, has $P$ as its vertex set and all pairs of $L$-neighbors in $P$ as its edge set.*

The local-region function controls the number of edges in the resulting Local Minimum Spanning graph. For example, if we take the local-region function to be $L(\mathbf{p}) = \{\mathbf{p}\}$ then the resulting Local Minimum Spanning graph contains all edges. At the other extreme, if we let $L(\mathbf{p}) = P$ then the resulting Local Minimum Spanning graph is exactly the Euclidean minimum spanning tree of $P$.

Next we show that regardless of what local-region function is used, the Local Minimum Spanning graph contains the Euclidean minimum spanning tree.

We start by proving the following lemma, which is a minor generalization of the well known cycle property of minimum spanning trees. We remark that this lemma holds for weighted graphs, and is not tied to Euclidean geometry in any way.

**Lemma 3.13** (Cycle property)**.** *Let $G$ be a graph with unique edge weights, and let $T$ be its minimum spanning tree. An edge $e$ is not present in $T$ if and only if $G$ contains a cycle where $e$ is the edge of maximum weight.*

*Proof.* We prove each direction separately.

- $\Leftarrow$ Suppose by contradiction that there is an edge $e$ in $T$ and $G$ contains a cycle $C$ where $e$ is the edge of maximum weight.

  The graph $H = T \setminus \{e\}$ produced by removing the edge $e = \{u, v\}$ from $T$ has exactly two connected components $S$ and $R$, where $u \in V_S$ and $v \in V_R$. The cycle $C$ minus the edge $e$ describes a path from $u$ to $v$. If we start at vertex

$u \in S$ we can follow this path and eventually we will use some edge $e' \neq e$ which crosses from component $S$ to component $R$. Let $T' = H \cup \{e'\}$ be the tree resulting from joining components $S$ and $R$ with edge $e'$. Since both edge $e$ and $e'$ belong to a cycle $C$, then by assumption the weight of $e$ is strictly greater than the weight of $e'$, and thus the weight of $T'$ is strictly smaller than the weight of $T$ – a contradiction.

- $\Rightarrow$ Suppose by contradiction that there is an edge $e$ not present in $T$ and $G$ has no cycle where $e$ is the edge of maximum weight.

  The graph $H = T \cup \{e\}$ produced by adding the edge $e$ to the tree $T$ contain a single cycle $C$, and this cycle uses edge $e$. Let $e'$ be the edge with the largest weight in $C$, and observe that by assumption $e \neq e'$. Let $T' = H \setminus e'$ be the tree that results from removing the edge $e'$ from the newly produced cycle $C$ in $H$. Since both edge $e$ and $e'$ belong to a cycle $C$, then by assumption the weight of $e$ is strictly smaller than the weight of $e'$, and thus the weight of $T'$ is strictly smaller than the weight of $T$ – a contradiction.

□

We can now use this lemma to show that the Local Minimum Spanning graph always contains the Euclidean minimum spanning tree.

**Theorem 3.14.** *Let $P$ be a point set and $L$ be a local-region function on $P$. Then $EMST(P) \subseteq LMSG_L(P)$.*

*Proof.* To prove the theorem it suffices to show that if an edge is not contained in the Local Minimum Spanning graph of $P$ then this edge is not contained in the Euclidean minimum spanning tree of $P$.

Suppose that the graph $LMSG_L(P)$ does not contain the edge between $\mathbf{p}$ and $\mathbf{q}$. By definition it follows that either $\mathbf{p}$ is not L-safe with respect to $\mathbf{q}$ or vice versa, without loss of generality assume $\mathbf{p}$ is not L-safe with respect to $\mathbf{q}$.

Since $\mathbf{p}$ is not L-safe with respect to $\mathbf{q}$ then $L(\mathbf{p})$ contains $\mathbf{p}$ and $\mathbf{q}$ but the Euclidean minimum spanning tree of $L(\mathbf{p})$ does not contain the edge between $\mathbf{p}$ and $\mathbf{q}$. Lemma 3.13 (the $\Rightarrow$ direction) implies that $L(\mathbf{p})$ contains a cycle $C$ where the edge between $\mathbf{p}$ and $\mathbf{q}$ has the largest weight. Since $L(\mathbf{p}) \subseteq P$ then the cycle $C$ is also present in $P$, and Lemma 3.13 (the $\Leftarrow$ direction) implies that the edge between $\mathbf{p}$ and $\mathbf{q}$ is not included in the Euclidean minimum spanning tree of $P$. □

### 3.2.5 Local Minimum Spanning Graphs With Few Edges

We have showed that regardless of the local-region function used the resulting Local Minimum Spanning graph contains the Euclidean minimum spanning tree. In this subsection we propose a local-region function which is appropriate for distributed computation, and we prove that its resulting Local Minimum Spanning graph has fewer edges than any of the other graphs we have considered so far.

When doing distributed computation on unit disk graphs, a natural choice for the local-region function of a point set is precisely the unit disk around each point. We define the *unit-local-region* function $L_{ud}$ of a point set $P$ as $L_{ud}(\mathbf{p}) = P[\mathbf{p}]$ for all $\mathbf{p} \in P$. The next claim follows from the definition of $L_{ud}$-safe.

**Claim 3.15.** *Fix a point set $P$ and two points $\mathbf{p} \in P$ and $\mathbf{q} \in P[\mathbf{p}] \setminus \{\mathbf{p}\}$. Then $\mathbf{p}$ is $L_{ud}$-safe in $P$ with respect to $\mathbf{q}$ if and only if $\mathbf{p}$ is $L_{ud}$-safe in $P[\mathbf{p}]$ with respect to $\mathbf{q}$.*

*Proof.* By definition of $L_{ud}$ we have $L_{ud}(\mathbf{p}) \cap P = L_{ud}(\mathbf{p}) = L_{ud}(\mathbf{p}) \cap P[\mathbf{p}]$ and the claim follows. □

We leverage Claim 3.15 claim to design the following distributed algorithm.

**Algorithm 5** $\text{LMSG}_{L_{ud}}$-EdgeSelect for robot at $\mathbf{p}$.

1: broadcast $\mathbf{p}$, and let $X = \{\mathbf{p}\} \cup \{\mathbf{q} \mid \mathbf{q} \text{ was received}\}$
2: locally select $\{\mathbf{q} \mid \mathbf{q} \in X \setminus \{\mathbf{p}\}$ and $\mathbf{p}$ is $L_{ud}$-safe in $X$ with respect to $\mathbf{q}\}$

The following proposition follows from the algorithm definition and Claim 3.15.

**Proposition 3.16.** *Let $P$ be a point set and let $H(P)$ be the graph induced by the edges which are consistently selected by the* $\text{LMSG}_{L_{ud}}$-EdgeSelect *algorithm. Then $H(P) = LMSG_{L_{ud}}(P) \cap UDG(P)$.*

Next we show that for the unit-local-region function the resulting graph is contained in the Gabriel Graph, the Relative Neighbor graph, and the Cone-Based Topology Control graph.

**Theorem 3.17.** *Let $P$ be a point set. Then $LMSG_{L_{ud}}(P) \cap UDG(P)$ is contained in $GG(P) \cap UDG(P)$, $RNG(P) \cap UDG(P)$ and $CBTC_{2\pi/3}(P)$.*

*Proof.* Fix an edge $e$ in $UDG(P)$ with endpoints $\mathbf{p}$ and $\mathbf{q}$.

We claim that if $e$ is not in $GG(P)$, or not in $RNG(P)$, or not in $CBTC_{\frac{2\pi}{3}}(P)$, there is a point $\mathbf{w}$ which is strictly closer to $\mathbf{p}$ than $\mathbf{q}$ and strictly closer to $\mathbf{q}$ than

$\mathbf{p}$. This claim implies that $L_{ud}(\mathbf{p})$ contains a cycle between $\mathbf{p}, \mathbf{q}$ and $\mathbf{w}$ where the weight of the edge between $\mathbf{p}$ and $\mathbf{q}$ is largest. This together with Lemma 3.13 (the $\Leftarrow$ direction) implies that $e$ is not in $LMSG_{L_{ud}}(P)$, which completes the theorem.

We prove the previous claim by cases.

1. If $e$ is not in $GG(P)$ then there is a point $\mathbf{w} \in P$ inside the GG-region between $\mathbf{p}$ and $\mathbf{q}$. However, by the definition of the GG-region this implies $\mathbf{w}$ is strictly closer to $\mathbf{p}$ than $\mathbf{q}$, and strictly closer to $\mathbf{q}$ than $\mathbf{p}$.

2. If $e$ is not in $RNG(P)$ then there is a point $\mathbf{w} \in P$ inside the RN-region between $\mathbf{p}$ and $\mathbf{q}$. However, by the definition of the RN-region this implies $\mathbf{w}$ is strictly closer to $\mathbf{p}$ than $\mathbf{q}$, and strictly closer to $\mathbf{q}$ than $\mathbf{p}$.

3. If $e$ is not in $CBTC_{\frac{2\pi}{3}}(P)$ then without loss of generality we can assume $\mathbf{p}$ is not $\frac{2\pi}{3}$-safe with respect to $\mathbf{q}$. This means that all cones with apex at $\mathbf{p}$ and aperture $\frac{2\pi}{3}$ that contain $\mathbf{q}$ also contain a point of $P$ which is strictly closer to $\mathbf{p}$ than $\mathbf{q}$. In particular the cone with apex at $\mathbf{p}$ and aperture $\frac{2\pi}{3}$ with its axis going through $\mathbf{q}$ contains a point $\mathbf{w} \in P$ which is strictly closer to $\mathbf{p}$ than $\mathbf{q}$. In other words we have shown that $\|p-w\| < \|p-q\|$, and to complete the claim it suffices to show that $\|q-w\| < \|p-q\|$.

   The existence of the previous cone implies there exists a (thinner) cone with apex at $\mathbf{p}$ and aperture at most $\frac{\pi}{3}$ which contains both $\mathbf{q}$ and $\mathbf{w}$. Applying the cosine law to the triangle formed by $\mathbf{p}, \mathbf{q}$ and $\mathbf{w}$ we have that $\|q-w\|^2 = \|p-q\|^2 + \|p-w\|^2 - 2\|p-q\|\|p-w\|\cos\theta$ where $\theta$ is the angle formed by $\angle\mathbf{wpq}$ and therefore $\theta \in [0, \frac{\pi}{3}]$. Finally, from this it follows that $\cos\theta \geq \frac{1}{2}$ and since we had already shown that $\|p-w\| < \|p-q\|$ we have:

$$\begin{aligned}\|q-w\|^2 &= \|p-q\|^2 + \|p-w\|^2 - 2\|p-q\|\|p-w\|\cos\theta \\ &< \|p-q\|^2 + \|p-w\|^2 - 2\|p-w\|^2\frac{1}{2} = \|p-q\|^2 \\ \|q-w\| &< \|p-q\|.\end{aligned}$$

□

This theorem implies that when using the unit-local-region function the resulting Local Minimum Spanning graph has fewer edges than any of the previously described graphs. Moreover, since Theorem 3.14 showed that the Local Minimum Spanning graph contains the Euclidean minimum spanning tree we have the following corollary.

**Corollary 3.18.** *The $GG/RNG/CBTC_{\frac{2\pi}{3}}/LMSG_{L_{ud}}$-edge-selection algorithms solve the MST-containing problem.*

Informally speaking, in the next section we show that there is no distributed solution to the MST-containing problem that has fewer edges than the graph produced using the Local Minimum Spanning graph strategy.

## 3.3 Optimal Local Minimum Spanning Graphs

The definition of a Local Minimum Spanning graph is not tied to the complete geometry of a point set (or on the unit disk graph assumption), and it can be generalized to any weighted graph (where the weights need not satisfy the triangle inequality). We state our optimality claims for this generalization of Local Minimum Spanning graphs.

For a weighted graph $G$ a function $L : V_G \to 2^G$ is a *local-region* function on $G$ if it maps every vertex $v$ in $G$ to a subgraph of $G$ which contains $v$. A vertex $u$ is *L-safe* with respect to $v$ if and only if $\{u, v\} \notin L(u)$ or the minimum spanning tree of $L(u)$ contains the edge $\{u, v\}$. Two neighbors $\{u, v\} \in E(G)$ are *L-neighbors* in $G$ if and only if $u$ is L-safe with respect to $v$ and vice versa.

**Definition 3.19.** *The* Local Minimum Spanning graph *of a weighted graph $G$ and local-region function $L$ on $G$, denoted by $LMSG_L(G)$, has $V_G$ as its vertex set and all pairs of L-neighbors in $G$ as its edge set.*

Given a point set $P$, Definition 3.12 is recovered from Definition 3.19 by considering $G$ to be a complete weighted graph over $P$ where the weight of each edge is equal to the Euclidean distance between its end points. The same proof used to demonstrate Theorem 3.14 can be used to yield the following generalization.

**Theorem 3.20.** *Let $G$ be a weighted graph and let $L$ a local-region function on $G$, then $MST(G) \subseteq LMSG_L(G)$.*

Before we can state the optimality result, we need some additional definitions. For a weighted graph $G$ and a positive integer $t \in \mathbb{Z}^+$ let $G^t[v]$ be the $t$-neighborhood of $v$ in $G$ (i.e., the largest weighted subgraph of $G$ that a robot $v$ can learn after $t$ communication steps of a full-information protocol). Observe that when dealing with geometric graphs, including unit disk graphs, $G^t[v]$ includes the positions of the reachable vertices. Moreover, regardless of $G$, $v$ or $t$, $v$ is included in $G^t[v]$ by definition.

The next theorem shows that if for a graph $G$ we let $L(v) = G^t[v]$, then it is impossible for a deterministic distributed algorithm that solves the MST-containing problem and runs for $t$ communication steps to consistently select fewer edges than those in $LMSG_L(G)$.

**Theorem 3.21.** *Let $\mathcal{A}$ be a deterministic edge-selection distributed algorithm that runs for $t$ communication steps and solves the MST-containing problem. If we let $L(v) = G^t[v]$ then the graph $LMSG_L(G)$ is contained in the edges consistently selected by $\mathcal{A}$ when run in $G$.*

The proof idea is very simple. First observe that if $G^t[v] = H^t[v]$ then by definition for a robot at $v$ the graphs $G$ and $H$ are indistinguishable when running for less than or equal to $t$ communication steps. To prove the theorem we show that given a graph $G$ and a vertex $v \in G$, it is possible to construct a graph $H$, where $G^t[v] = H^t[v]$, and the minimum spanning tree of $H$ is guaranteed to contain all the edges present in the minimum spanning tree of $H^t[v]$.

*Proof.* Let $\mathcal{A}$ be any edge-selection deterministic distributed algorithm that runs in $t$ communication steps and solves the MST-containing problem. Suppose by contradiction that there is a graph $G$ where $\mathcal{A}$ does not consistently select an edge $\{u, v\}$ and $\{u, v\} \in LMSG_L(G)$.

Since $\{u, v\}$ is not consistently selected when running $\mathcal{A}$ in $G$, we can assume that when running $\mathcal{A}$ in $G$ the robot at $u$ does not locally select $v$.

Consider the communication graph $H = G^t[u]$ and observe that by the definition of $G^t[u] = H^t[u]$. Therefore, any deterministic procedure running at $u$ which runs for less or equal than $t$ communication steps will produce the same outcome in $G$ and $H$. Since by assumption robot $u$ does not locally select $v$ when running $\mathcal{A}$ in $G$, then it also does not locally select $v$ when running $\mathcal{A}$ in $H$. This implies the edge $\{u, v\}$ is not consistently selected by $\mathcal{A}$ when running in $H$.

However since $\{u, v\} \in LMSG_L(G)$ by assumption, then by the definition of the Local Minimum Spanning graph it must be that the edge $\{u, v\}$ is present in the minimum spanning tree of $G^t[u] = H^t[u] = H$. But since we have argued that the edge $\{u, v\}$ is not consistently selected when running $\mathcal{A}$ in $H$, this contradicts the assumption that $\mathcal{A}$ solves the MST-containing problem.

□

# Chapter 4

# Distributed Connectivity-Preserving Algorithm

Paraphrasing the informal description of the connectivity-preserving problem given at the beginning of Part I— at each round a motion planner produces a set of desired trajectories (one for each robot), and the goal of the connectivity-preserving problem is to produce another set of trajectories which satisfy the *$\varepsilon$-progress* and *robust safety* properties. The different components and their interactions are outlined in Figure 4-1, the detailed inputs and outputs of each component appear in Section 4.1.

Our approach to the connectivity-preserving problem is to subdivide the problem in two parts. In the first part, the goal is to find, for each robot, a set of neighbors such that preserving connectivity to these neighbors is sufficient for the communication graph to remain connected. The fewer neighbors a robot has to preserve, the greater freedom it has to compute a trajectory that remains connected to these neighbors. Moreover, remaining connected to close-by robots is less of a restriction than remaining connected to robots that are farther away. This problem was tackled in Chapter 3.

The second part of the problem is concerned with agreeing on a set of linear trajectories (one for each robot), so as to maximize the progress each robot makes with respect to its original trajectory (controlled by the motion planner). The only constraint when finding such trajectories is that each robot should remain robustly-connected to each of the neighbors which were identified as being sufficient for preserving connectivity in the first part. This is the problem tackled in the present chapter.

**Roadmap.** In Section 4.1 we introduce the definitions necessary to formalize the connectivity-preserving problem. In Section 4.2 we describe the connectivity-preserving algorithm CP-Alg . In Section 4.3 we prove that at each round the trajectories output by the algorithm CP-Alg satisfy the robust safety property. In Section 4.4 we argue that the algorithm CP-Alg guarantees that no robot will move away from its desired target position. We also demonstrate that, without additional assumptions, it is impossible for any algorithm to simultaneously guarantee that the graph remains connected and that the robots collectively move strictly closer to their desired target positions. We then show that under some reasonable assumptions on the trajectories produced by the motion planner module, the proposed algorithm guarantees that the robots collectively move closer to their desired target positions at each round. In Section 4.5 we describe how the progress arguments can be extended to analyze an execution through multiple communication rounds.

## 4.1 The Connectivity-Preserving Problem

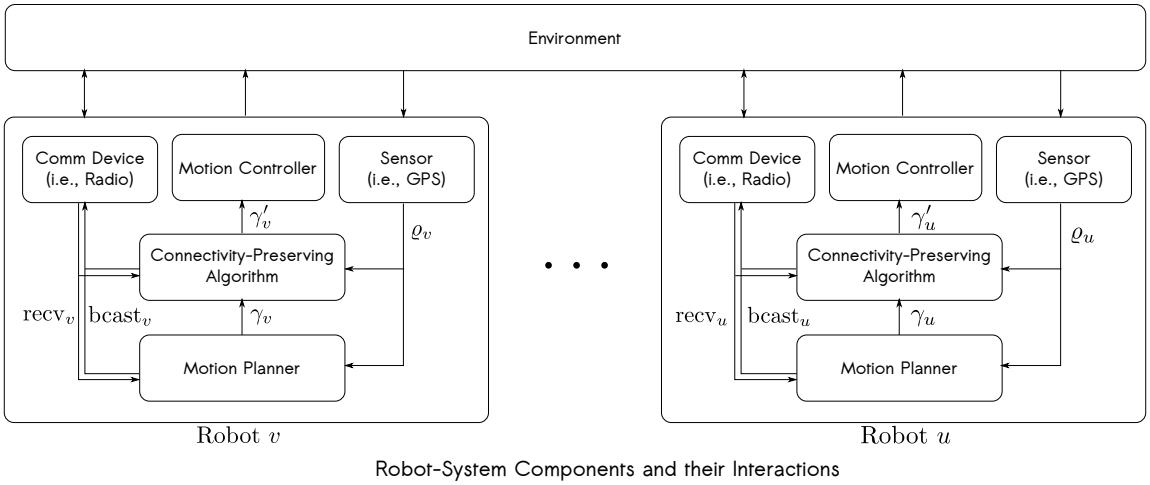



Figure 4-1: A robot $v$ can communicate by broadcasting a message $m$ to its neighbors through the $\text{bcast}(m)_v$ action, and receiving a message $m$ from a neighboring robot $u$ through the $\text{recv}(m,u)_v$ action. The sensors at robot $v$ provide it with its own position $\varrho_v$ (i.e., via GPS). The connectivity-preserving module at robot $v$ receives as input its own position $\varrho_v$ (output by its sensors) and a linear trajectory $\gamma_v$ (output by its motion planner module). The output of the connectivity-preserving module is a linear trajectory $\gamma'_v$, whose computation may require some number of communication steps. The motion controller receives as input the trajectory $\gamma'_v$ and controls the actuators of the robot to execute the trajectory in the physical world.

A *trajectory* is the path that a moving object follows through space as a function of time. In particular the linear trajectory between $\mathbf{a} \in \mathbb{R}^2$ and $\mathbf{b} \in \mathbb{R}^2$ is described by the function $f(t) = (1-t)\mathbf{a} + t\mathbf{b}$ where $t \in [0,1]$. We denote by $\gamma_v : [0,1] \to \mathbb{R}^2$ the trajectory produced by the motion planner module at robot $v$, and by $\gamma'_v : [0,1] \to \mathbb{R}^2$ the trajectory output by the connectivity-preserving module at robot $v$. For simplicity we restrict $\gamma_v$ and $\gamma'_v$ to be linear trajectories. We emphasize that this does not prevent the trajectory observed by a robot, considered over a series of rounds, from being non-linear. The starting point of a trajectory for a robot must be equal to the position of the robot at the beginning of the round. Therefore we have $\gamma_v(0) = \gamma'_v(0) = \varrho_v$, where $\varrho_v$ denotes the position of robot $v$ at the beginning of the round.

We define the *configuration* of the current round as the collection of robots $V$

together with their positions at the beginning of the round ($\varrho_v, v \in V$) and their trajectories produced by the motion planner at the beginning of the round ($\gamma_v, v \in V$). For a robot $v \in V$ in a configuration we define its *target vector* as $\gamma_v(1) - \gamma_v(0)$.

The progress made by a robot $v$ following trajectory $\gamma'_v$ with respect to the original trajectory $\gamma_v$, is measured by the distance robot $v$ advances towards the original destination $\gamma_v(1)$ when it reaches the new destination $\gamma'_v(1)$. Formally, the *progress* of robot $v$ following $\gamma'_v$ with respect to the original trajectory $\gamma_v$ is defined as $\delta_v = \|\gamma_v(0) - \gamma_v(1)\| - \|\gamma_v(1) - \gamma'_v(1)\|$. We highlight that if robot $v$ moves farther away from its desired target position $\gamma_v(1)$ when following the trajectory $\gamma'_v$, then its progress will be negative. Observe that the progress of a robot is a property of "what happens" during the round, but since our system model does not include any sources of uncertainty, the progress is simply a function of the configuration. The progress of a configuration is the sum of the progress of the individual robots, $\delta_V = \sum_{v \in V} \delta_v$.

Informally speaking, the $\varepsilon$-progress property guarantees that individually no robot moves farther away from its desired target, and collectively the robots get $\varepsilon$ closer to their desired positions.

**Definition 4.1.** *A configuration satisfies the* $\varepsilon$-progress *property if* $\delta_V \geq \varepsilon$ *and* $\delta_v \geq 0$ *for every* $v \in V$.

Informally speaking, we say the trajectories of a pair of robots are *robustly-connected* if, regardless of the speed at which the robots follow these trajectories, they remain within distance $r$ throughout the motion. This definition is motivated by the requirement for robots to remain connected, despite the fact that they might be traveling at different speeds, or that they might encounter unexpected obstacles which force them to halt in the middle of a trajectory.

**Definition 4.2.** *The trajectories* $\gamma_u$ *and* $\gamma_v$ *are* robustly-connected *if* $\|\gamma_u(s) - \gamma_v(t)\| \leq r$ *for* $\forall s, t \in [0, 1]$.

We also consider a weaker notion of connectivity which only requires the robots to be connected when both robots follow their respective trajectories using the same speed. More precisely, the trajectories of a pair of robots are *weakly-connected* if when both robots travel at the same speed, and each of them follows faithfully the trajectory prescribed to it from beginning to end, then at every instant they remain within distance $r$ of each other.

Despite the fact that the informal definition of weakly-connected is straightforward, the fact that different trajectories might have different lengths but their argument always goes from 0 to 1 introduces some technicalities which must be addressed in the formal definition. Specifically, since the argument $t$ of the trajectories does not

represent time, but a fraction of the distance, getting from $\gamma_u(0)$ to $\gamma_u(t)$ might require robot $u$ to travel a much greater (or smaller) distance than what robot $v$ needs to travel to get from $\gamma_v(0)$ to $\gamma_v(t)$, which explains why we can't compare $\gamma_u(t)$ and $\gamma_v(t)$ directly. Instead, we must compare the two trajectories at the points at which each robot has traveled the same distance, in other words we compare $\gamma_u(t/\ell_u)$ with $\gamma_v(t/\ell_v)$ where $\ell_u$ and $\ell_v$ correspond to the length of the trajectories $\gamma_u$ and $\gamma_v$ respectively. An additional caveat is that since the trajectories of the robots might have different lengths, and the weakly-connected property requires the robots to travel at the same speed, then it is possible for one robot to reach the destination of its trajectory while the other robot is still en route to its own destination. In this case we will require that once a robot has reached its destination, the other robot remains at distance $r$ while following its trajectory until it has reached its own destination. This is handled by the use of min in the formal definition below.

**Definition 4.3.** *The trajectories $\gamma_u$ and $\gamma_v$ are* weakly-connected *if $\|\gamma_u(s) - \gamma_v(t)\| \leq r$ for $s = \min(x/\|\gamma_u(0) - \gamma_u(1)\|, 1)$, $t = \min(x/\|\gamma_v(0) - \gamma_v(1)\|, 1)$ for all $x \in [0, \max(\|\gamma_u(0) - \gamma_u(1)\|, \|\gamma_v(0) - \gamma_v(1)\|)]$.*

A set of trajectories (one for each robot in $G$) satisfies *weak safety* if there exists a connected subgraph $H \subseteq G$ where every pair of adjacent robots in $H$ are assigned weakly-connected trajectories. Similarly, a set of trajectories satisfies *robust safety* if there exists a connected subgraph $H \subseteq G$ where every pair of adjacent robots in $H$ are assigned robustly-connected trajectories.

## 4.2 The Connectivity-Preserving Algorithm

At the beginning of each round and at each robot $v$, the connectivity-preserving algorithm receives as an input the communication radius $r$, its position $\varrho_v$ and a proposed trajectory $\gamma_v$ from its motion planner module. The output of the connectivity-preserving algorithm is a new trajectory $\gamma'_v$. We present a connectivity-preserving algorithm that requires only two communication steps per round. Specifically, CP-Alg is a three-phase algorithm that consists of a Selection phase, a Proposal phase, and an Adjustment phase (this last phase requires no communication). Below we give an informal description of CP-Alg , the pseudocode is in Algorithm 6.

In the Selection phase each robot learns its set of unit disk neighbors and their positions and locally selects a subset of them. The Selection Phase implements a distributed edge-selection algorithm that guarantees that a connectivity-preserving set of edges is consistently selected. The robots can accomplish this by running any of the distributed edge-selection algorithms that solve the MST-containing problem described in Chapter 3, but other implementations are possible. At each robot, the output of the Selection phase is the set of neighbors which are locally selected.

In the Proposal phase each robot "optimistically" chooses a target based on the neighbors locally selected in the Selection phase. Specifically, each robot $v$ proposes as its target the point $q$ defined as the point closest to its original target $\gamma_v(1)$ which is also within distance $r$ from each of the locally selected neighbors of $v$. The proposed target is optimistic in the sense that if robot $v$ follows a linear trajectory from $\varrho_v$ to $q$, and no other robot moved, then robot $v$ would remain at distance $r$ from its locally selected neighbors. The proposed target is then broadcast and the proposals of neighboring robots are recorded.

In the Adjustment phase each robot $v$ assumes that itself and its neighbors will move to their proposed targets, and it checks whether all of its locally selected neighbors would be within distance $r$ of itself. If every locally selected neighbor would be within distance $r$, then the robot chooses $q$ as the target position of its output trajectory, otherwise the robot chooses $q' = \frac{1}{2}(\varrho_v + q)$ as the target position of its output trajectory.

**Pseudo-code Description.** In the Selection phase robot $v$ runs a distributed edge-selection algorithm that consistently selects a connectivity-preserving set of edges while ignoring its non unit disk neighbors. This can be implemented using any of the strategies described in Chapter 3. In the Proposal phase, robot $v$ computes $R$ as the intersection of all the disks of radius $r$ centered at the positions of each of the neighbors of $v$ which were selected in the previous phase. Robot $v$ then computes $q$

**Algorithm 6** CP-ALG $(r, \varrho_v, \gamma_v)$ at robot $v$

▷ Initialization

$N \leftarrow \{\}, P \leftarrow \{\}, Q \leftarrow \{\}$

▷ Selection Phase

**bcast** $\varrho_v$
**if recv**$(\varrho_u, u)$ and $d(\varrho_u, \varrho_v) \leq r$ **then**
    $N \leftarrow N \cup \{u\}$, $P[u] \leftarrow \varrho_u$
LOCALLYSELECT a subset $S \subseteq N$

▷ Proposal Phase

$R \leftarrow \bigcap_{u \in S} B_r[P[u]]$
$q \leftarrow \text{argmin}_{p \in R}\, d(p, \gamma_v(1))$
**bcast** $q$
**if recv**$(q_u, u)$ and $d(\varrho_u, \varrho_v) \leq r$ **then**
    $Q[u] \leftarrow q_u$

▷ Adjustment Phase

$q' \leftarrow \frac{1}{2}(\varrho_v + q)$
**if** $\forall u \in S \quad d(q_v, Q[u]) \leq r$ **then**
    $\gamma'_v \leftarrow$ linear trajectory from $\varrho_v$ to $q$
**else**
    $\gamma'_v \leftarrow$ linear trajectory from $\varrho_v$ to $q'$
**return** $\gamma'_v$

as the point inside $R$ which is closest to its desired target. In the Adjustment phase robot $v$ first computes an adjusted proposal $q' = \frac{1}{2}(\varrho_v + q)$. Robot $v$ proceeds to check whether the distance between its own proposal and the proposals of its selected neighbors is at most $r$. If so, then the robot $v$ outputs a linear trajectory between its current position and its proposal, otherwise it outputs a linear trajectory between its current position and its adjusted proposal.

## 4.3 Safety

In this section we prove that the trajectories output by the algorithm CP-Alg guarantee the robust safety property. We start with two simple propositions. To disambiguate we use $S_v, R_v, q_v, q'_v$ to refer to the local variables $S, R, q, q'$ of robot $v$ respectively.

**Proposition 4.4.** *Fix $v \in V$. Then 1. $R_v$ is convex and contains $\varrho_v$, 2. $q_v, q'_v \in R_v$, and 3. $\gamma'_v$ is contained in $R_v$.*

*Proof.* We prove each property separately. 1. By construction $R_v$ is the intersection of a set of disks, each of which contains $\varrho_v$. 2. By construction $q_v \in R_v$. From 1 we have that $R_v$ is convex and contains $\varrho_v$, and the convexity of $R_v$ implies that $q'_v = \frac{1}{2}(q_v + \varrho_v) \in R_v$. 3. From 1 we have that $\varrho_v \in R_v$, and from 2 we have that $q_v, q'_v \in R_v$. Since $\gamma'_v$ is a linear trajectory from $\varrho_v$ and $q_v$ or $q'_v$, the convexity of $R_v$ implies that $\gamma'_v$ is contained in $R_v$. □

We say two robots are *consistent neighbors* if the edge between them is consistently selected. The next proposition captures a trivial but useful property about the regions $R_u$ and $R_v$ when $u$ and $v$ are consistent neighbors.

**Proposition 4.5.** *If $u$ and $v$ are consistent neighbors, then for any $p \in R_v$ we have $\|\varrho_u - p\| \leq r$.*

*Proof.* By construction $R_v \subseteq B_r[\varrho_u]$. □

As a corollary of these two propositions we have the following immediate result.

**Corollary 4.6.** *If robot $v$ follows the trajectory $\gamma'_v$ (at any speed) while all other robots remain stationary then all consistently selected edges are preserved.*

*Proof.* Since only robot $v$ is moving, we need only to show that it preserves all its consistent neighbors. Proposition 4.4 implies every point in the trajectory of $v$ are contained in $R_v$ and Proposition 4.5 implies that a point in $R_v$ is within distance $r$ from any of $v$'s consistent neighbors. □

In the remainder of this section we show that the trajectories output by the algorithm guarantee that all consistently selected edges are preserved, without having to require any subset of robots to remain stationary. We start by proving why the adjustment phase works.

**Lemma 4.7.** *If $u$ and $v$ are consistent neighbors, then $\|q'_u - q'_v\| \leq r$.*

*Proof.* Since by assumption $u$ and $v$ are consistent neighbors then Proposition 4.5 implies that $\|\varrho_u - q_v\| \leq r$ and $\|\varrho_v - q_u\| \leq r$. The statement follows from the triangle inequality and the definition of the adjusted proposals:

$$\begin{aligned}\|q'_u - q'_v\| &= \left\|\frac{1}{2}\varrho_u + \frac{1}{2}q_u - \frac{1}{2}\varrho_v - \frac{1}{2}q_v\right\| \\ &= \frac{1}{2}\|(\varrho_u - q_v) + (q_u - \varrho_v)\| \\ &\leq \frac{1}{2}\|\varrho_u - q_v\| + \frac{1}{2}\|\varrho_v - q_u\| \leq r\end{aligned}$$

□

We leverage this lemma to show that the endpoints of the trajectories of two consistent neighbors are connected.

**Lemma 4.8.** *If $u$ and $v$ are consistent neighbors, then $\|\gamma'_u(1) - \gamma'_v(1)\| \leq r$.*

*Proof.* If $\|q_u - q_v\| > r$ then both $u$ and $v$ adjust their trajectories and we have $\gamma'_u(1) = q'_u$ and $\gamma'_v(1) = q'_v$ and the statement follows by Lemma 4.7.

If $\|q_u - q_v\| \leq r$ and neither $u$ or $v$ adjust their trajectories, then $\gamma'_u(1) = q_u$ and $\gamma'_v(1) = q_v$ and the statement follows.

If $\|q_u - q_v\| \leq r$ and (without loss of generality) $u$ adjusts and $v$ doesn't adjust then $\gamma'_u(1) = q'_u = \frac{1}{2}(\varrho_u + q_u)$ and $\gamma'_v(1) = q_v$. From Proposition 4.5 it follows that $\|\varrho_u - q_v\| \leq r$ and since by assumption we have $\|q_u - q_v\| \leq r$, the rest follows by the triangle inequality:

$$\begin{aligned}\|q'_u - q_v\| &= \left\|\frac{1}{2}\varrho_u + \frac{1}{2}q_u - q_v\right\| = \left\|\frac{1}{2}(\varrho_u - q_v) + \frac{1}{2}(q_u - q_v)\right\| \\ &\leq \frac{1}{2}\|\varrho_u - q_v\| + \frac{1}{2}\|q_u - q_v\| \leq \frac{1}{2}r + \frac{1}{2}r = r.\end{aligned}$$

□

Lemma 4.8 showed that the endpoints of the trajectories of two consistent neighbors are connected. However this does not rule out the possibility that two robots

following these trajectories could become disconnected somewhere in between their initial positions and the target positions. The next theorem proves the *robustness* of the trajectories computed by the algorithm.

**Theorem 4.9.** *(The trajectories of consistent neighbors are robustly-connected.) If $u$ and $v$ are consistent neighbors, then $\|\gamma'_u(s) - \gamma'_v(t)\| \leq r$ for all $s, t \in [0, 1]$.*

*Proof.* Since $u$ and $v$ are neighbors in the communication graph, $\|\gamma'_u(0) - \gamma'_v(0)\| = \|\varrho_u - \varrho_v\| \leq r$; equivalently $\gamma'_v(0) \in B_r[\gamma'_u(0)]$. Since $\gamma'_v(1) \in R_u$ then Proposition 4.5 implies that $\|\varrho_u - \gamma'_v(1)\| = \|\gamma'_u(0) - \gamma'_v(1)\| \leq r$, and by symmetry $\|\gamma'_u(1) - \gamma'_v(0)\| \leq r$; equivalently $\gamma'_v(0) \in B_r[\gamma'_u(1)]$ and $\gamma'_v(1) \in B_r[\gamma'_u(0)]$. Moreover Lemma 4.8 implies that $\|\gamma'_u(1) - \gamma'_v(1)\| \leq r$; equivalently $\gamma'_v(1) \in B_r[\gamma'_u(1)]$.

Putting the above together we have $\gamma'_v(0), \gamma'_v(1) \in B_r[\gamma'_u(0)] \cap B_r[\gamma'_u(1)]$. The convexity of $B_r[\gamma'_u(0)] \cap B_r[\gamma'_u(1)]$ and the linearity of $\gamma'_v$ implies that $\gamma'_v(t) \in B_r[\gamma'_u(0)] \cap B_r[\gamma'_u(1)]$ for all $t \in [0, 1]$. This is equivalent to $\gamma'_u(0), \gamma'_u(1) \in B_r[\gamma'_v(t)]$ for every $t \in [0, 1]$. Finally, from the convexity of $B_r[\gamma'_v(t)]$ and the linearity of $\gamma'_u$ we have that $\gamma'_u(s) \in B_r[\gamma'_v(t)]$ for all $s, t \in [0, 1]$, which is equivalent to $\|\gamma'_u(s) - \gamma'_v(t)\| \leq r$ for all $s, t \in [0, 1]$. □

The result above holds for any pair of consistently selected neighbors, and since by assumption the distributed edge-selection algorithm used in the selection phase consistently selects a connectivity-preserving set of edges, then we have the following as an immediate corollary (where $H$ is the graph formed by the consistently selected edges).

**Theorem 4.10.** *(The algorithm satisfies robust safety.) There is a connected subgraph $H \subseteq G$ where for every pair of adjacent robots in $H$* CP-Alg *outputs robustly-connected trajectories.*

## 4.4 Progress

In Section 4.3 we showed that CP-Alg satisfies the robust safety property, and is therefore guaranteed to preserve the connectivity of the communication graph. However, for the algorithm to be of any use, it must also allow the robots to advance towards their destinations. We devote this entire section to this question.

**Roadmap.** Section 4.4.1 briefly argues that without additional assumptions no algorithm can unconditionally guarantee safety and simultaneously guarantee that collectively the robots get strictly closer to their desired positions. Sections 4.4.2 and 4.4.3 consider two different sets of assumptions under which CP-Alg guarantees that collectively the robots get strictly closer to their desired positions.

### 4.4.1 Unconditional Progress

It is easy to verify that, by construction, CP-Alg never outputs a trajectory that takes a robot farther away from its desired position. In other words, the trajectories output by CP-Alg guarantee that $\delta_v \geq 0$ for every $v \in V$; equivalently CP-Alg satisfies the 0-progress property. However, it remains to show that the trajectories output by CP-Alg allow the robots to get closer to their destinations, that is, that CP-Alg satisfies the $\varepsilon$-progress property for some $\varepsilon > 0$.

In what follows we argue that it is impossible for any algorithm, and in particular for a local distributed algorithm, to unconditionally guarantee that connectivity is preserved and simultaneously guarantee the robots advance towards the desired positions. For instance, the motion planner could instruct every robot to remain stationary (i.e., $\gamma_v(0) = \gamma_v(1)$ for every $v \in V$), in which case it is impossible to have $\delta_V > 0$ by definition. Therefore, the progress guaranteed must be a function of the amount of progress the robots "want" to make in the first place.

However even when robots "want" to make progress, there are other subtle conditions which might prevent them from doing so. Specifically, some trajectories might require breaking connectivity (i.e., violating safety) to make progress, while other trajectories might require global information about the system (i.e., violating locality) to make progress.

> Example 4.1: (Progress requires violating locality) Consider a configuration where the robots are arranged in a circle and their communication graph corresponds to a cycle graph. Two neighboring robots want to move apart (breaking the communication edge between them) and every other robot wants to remain stationary. Observe that locally it is impossible for any robot in the system to determine if the communication graph is a line graph or a cycle graph. Allowing the desired motion would result in the communication graph becoming a line graph, and would not violate safety. However, if the communication was initially a line graph, the motion *would* violate safety. Since these two initial conditions are indistinguishable to the individual robots, it follows that no local distributed algorithm can guarantee progress in this configuration without violating safety in another.

Given that CP-Alg is a local distributed algorithm which guarantees safety unconditionally, it follows that we can hope to show it makes progress only by making additional assumptions on the set of trajectories produced by the motion planner.

The next subsections consider two "reasonable" sets of assumptions, and show that under these assumptions CP-Alg satisfies the $\varepsilon$-progress property for some $\varepsilon > 0$.

### 4.4.2 Robust Progress

In Section 4.3 we showed that the trajectories output by the proposed algorithm are robustly-connected for consistent neighbors. It would be reasonable to hope that, if the trajectories output by the motion planner are robustly-connected for consistent neighbors, then the algorithm guarantees progress.

**Definition 4.11.** *A configuration satisfies the* robust assumption *if the trajectories produced by the motion planner are robustly-connected for consistent neighbors.*

Leveraging the previous assumption we can prove the following theorem.

**Theorem 4.12.** *Let $C$ be a configuration that satisfies the robust assumption. Then the progress of $C$ is $\sum_{v \in V} \|\gamma_v(0) - \gamma_v(1)\|$.*

In other words, if the configuration satisfies the robust assumption, then the trajectories output by CP-Alg allow each robot to reach its desired destination (i.e., the robots make full progress). To prove this theorem it suffices to prove the following lemma.

**Lemma 4.13.** *If a configuration satisfies the robust assumption then $\gamma'_v(1) = \gamma_v(1)$ for every $v \in V$.*

*Proof.* Fix a robot $v \in V$. The proof relies on two claims. First we claim that $q_v = \gamma_v(1)$. Second, we claim that $\gamma'_v(1) = q_v$ (i.e., $v$ does not adjust its trajectory). Together these claims imply $\gamma'_v(1) = q_v = \gamma_v(1)$, which completes our proof.

To prove the first claim it suffices to show $\gamma_v(1) \in R_v$, since by the choice of $q_v$ this implies $q_v = \gamma_v(1)$. From the robust assumption for every $u \in S_v$ we have $\|\gamma_v(1) - \gamma_u(0)\| \leq r$, or equivalently $\gamma_v(1) \in B_r[\varrho_u]$. Since $R_v$ is the intersection of the set of disks $B_r[\varrho_u]$ for $u \in S_v$ we have $\gamma_v(1) \in R_v$ which proves the first claim.

To prove the second claim, it suffices to show that for every $u \in S_v$ we have $\|q_v - q_u\| \leq r$. From the first claim we have $q_v = \gamma_v(1)$ and $q_u = \gamma_u(1)$; thus we have only to show $\|\gamma_v(1) - \gamma_u(1)\| \leq r$. Finally, since $u$ and $v$ are consistent neighbors, the robust assumption implies $\|\gamma_v(1) - \gamma_u(1)\| \leq r$. □

### 4.4.3 Weak Progress

In the same spirit of the assumptions considered in Section 4.4.2, this subsection studies the progress guaranteed by CP-Alg under the assumption that the trajectories produced by the motion planner are weakly connected for consistent neighbors and that the trajectories do not induce any *cyclic dependencies*, defined below.

**Definition 4.14.** *Given a configuration $C$, the trajectory of robot $v$* depends on *robot $u$ if and only if* CP-Alg *would output a different trajectory for robot $v$ in the configuration $C'$ which is identical to $C$ but without robot $u$.*

We define the *dependency graph* of a configuration as a simple (i.e., with no multi-edges or self-loops) directed graph $D = (V, A)$ where $A$ contains a directed edge from $v$ to $u$ (denoted by $(v, u) \in A$) if and only if the trajectory output by robot $v$ depends on robot $u$. We say that a configuration has a *cyclic dependency* if its dependency graph contains a simple directed cycle of three or more vertices.

**Definition 4.15.** *A configuration satisfies the* weak assumption *if the trajectories produced by the motion planner are weakly-connected for consistent neighbors and it has no cyclic dependencies.*

Our main result is the following theorem.

**Theorem 4.16.** *Let $C$ be a configuration that satisfies the weak assumption, and let $d = \min_{v \in V} \|\gamma_v(0) - \gamma_v(1)\|$. Then the progress of $C$ is at least $\Omega(\min(d, r))$.*

In other words, if the configuration satisfies the weak assumption then the progress of the system must be at least the minimum amount of progress than any robot "wants" to make.

To simplify the proof we will ignore the adjustment phase of the algorithm (for example, by assuming the algorithm was modified by replacing the condition at line 12 to ensure all robots to execute line 13 and do not adjust their trajectory). We claim that this simplification can be made at the expense of losing a factor of two in our progress lower bound. To see why, observe that if we modify the algorithm to force every robot to use their adjusted proposal, the progress of the system is at least half of what it would have been if we modify the algorithm so that every robot uses their unadjusted proposal. In other words, the difference between all robots adjusting their proposals, or no robot adjusting its proposal is at most two. This allows us to consider a simpler version of the algorithm, which may not preserve connectivity, but whose progress is at most twice the progress of the original algorithm.

The proof of Theorem 4.16 is presented in the following sections. Below we give a high level outline.

**Proof Outline.** The progress of each robot is a function of its initial trajectory and the trajectory output by the algorithm. The trajectory output by the algorithm at each robot is the solution of an optimization problem, in particular a quadratically constrained quadratic program (the quadratic constraints and quadratic objective function are defined in the proposal phase of Algorithm 6). This optimization problem depends on the initial and target position of the robot, as well as the initial positions of its locally selected neighbors.

Therefore, to prove a lower bound on the progress of a configuration requires characterizing (and analyzing) the worst case solution to a set of optimization problems whose definition depends on the initial and target positions of the robots, as well as the structure of the communication graph induced by the configuration.

To tackle this problem, we first consider the set of configurations that satisfy a series of properties (we defined them below), we refer to these configurations as the *worst-case configurations*. Worst-case configurations have a structure which is simple enough to allow us to determine a lower bound on their progress analytically. Next we show that among all the configurations that satisfy the weak assumption, the worst-case configurations have the smallest progress. To prove this, we consider the space of configurations that results when removing each of the properties satisfied by the worst case configurations, one at a time, proving after removing each property that the progress of the resulting space of configurations cannot be any smaller.

To formalize the properties satisfied by the worst-case configurations we need some additional notation and definitions. For a robot $v$ we use $\tau_v$ to denote the target position of $v$ (i.e., $\tau_v = \gamma_v(1)$). We refer to the vector between the initial position of $v$ and the target position of $v$ as the *target vector* of $v$ (i.e., the target vector of $v$ is $\tau_v - \varrho_v$). Robot $v$ is *$d$-bounded* if its target vector is of length $d$, i.e., if $\|\tau_v - \varrho_v\| = d$. Robot $v$ is *balanced* if either it has a single neighbor, or if it has at least one neighbor on each side of the line that passes through $\tau_v$ and $\varrho_v$. Robot $u$ and robot $v$ are *parallel* if their target vectors have the same length and direction (i.e., the points $\varrho_u, \varrho_v, \tau_u$ and $\tau_v$ form a parallelogram). Robot $u$ and robot $v$ are *separated* if they are at distance exactly $r$ from each other. Robot $v$ is *straight* if the positions of all its neighbors and itself are collinear.

Using the definitions above we can define formally what constitutes a worst-case configuration. We say a configuration is a *worst-case configuration* if it satisfies all of the following properties.

1. Line: The communication graph of the robots is a line graph.
2. $d$-Bounded: All robots are $d$-bounded.
3. Balanced: All robots are balanced.
4. Parallel: All adjacent robots are parallel.

5. Separated: All adjacent robots are separated.
6. Straight: All robots are straight.

The next subsection argues that in a worst-case configuration all the "interior" robots (i.e., all robots except for the endpoints of the line) are forced to remain stationary, and only the robots at the endpoints of the line are capable of moving. We show that the progress of a worst-case configuration is at least $d$. The remaining subsections generalize this progress lower bound to any configuration that satisfies the weak assumption.

#### 4.4.3.1 Worst-Case Configuration

This section proves a lower bound on the progress of configurations which satisfy property 1 through property 6. Specifically we line consider configurations with a set of robots $\{v_1, \ldots, v_n\}$ where robot $v_i$ is adjacent to robot $v_{i+1}$ for $i \in \{1, \ldots, n-1\}$, and each robot is $d$-bounded, balanced, parallel, separated and straight. A configuration which satisfies the properties above is a worst-case configuration.

Rigid transformations are an isometry in Euclidean space, i.e., they are distance-preserving transformations. Therefore, it follows that the progress of a configuration is invariant to these transformations. In other words any combination of translations, rotations or reflections of the global coordinate system does not affect the progress of a configuration.

Thus, from a progress standpoint, the only relevant parameters for a worst-case configuration are the number of robots $n$, the length $d$ of the target vectors, and the relative angle $\theta$ between the target vectors and the line formed by the positions of the robots. Regardless of these parameters, its not hard to see that in a worst-case configuration, the inner robots are 'pinned' down and do not move (cf. Figure 4-2). Formally, for $i \in \{2, \ldots, n-1\}$ robot $v_i$ has an intersection region $R_{v_i} = \{\varrho_{v_i}\}$, and this implies $\delta_{v_i} = 0$. Since $\delta_V = \sum_{i=1}^{n} \delta_{v_i}$ then we have $\delta_V = \delta_{v_1} + \delta_{v_n}$.

We are now ready to show the main theorem of this section.

**Theorem 4.17.** *Let $C$ be a worst-case configuration which is $d$-bounded and $d \leq r$. The progress of $C$ is at least $d$.*

*Proof.* Since $\delta_V = \delta_{v_1} + \delta_{v_n}$ we consider only the progress of robots $v_1$ and $v_n$. In particular, these robots are restricted either by no other robot (in which case they make full progress), or by a single neighboring robot (in which case their progress depends on the exact value of $\theta$).

By symmetry we can restrict to the case where the relative angle $\theta$ of the target vectors is in the quadrant $[0, \pi/2]$; if $\theta$ lies in any other quadrant the configuration is equivalent up to a reflection of the entire coordinate system.

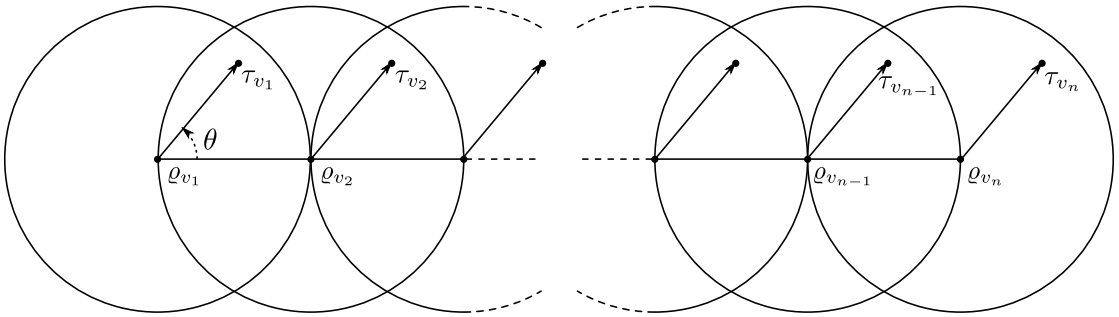



Figure 4-2: A worst-case configuration of $n$ robots. Except for robot $v_1$ and robot $v_n$ all other robots have an "empty" intersection region and have to remain stationary.

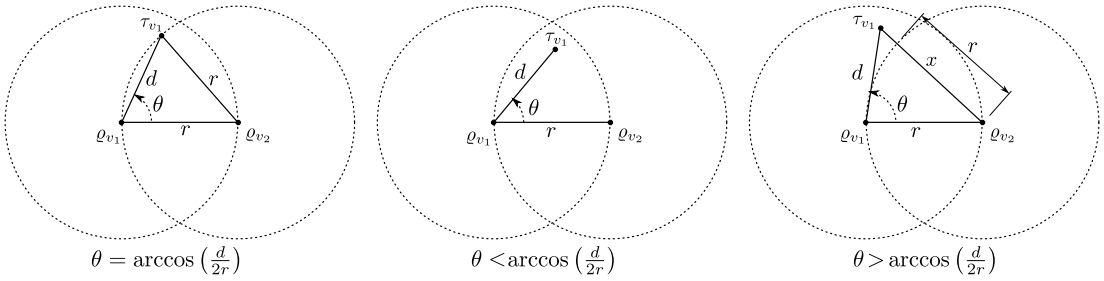



Figure 4-3: If $\theta = \arccos(d/2r)$ the target position of robot $v_1$ lies exactly at its region boundary (and robot $v_1$ makes full progress). If $\theta < \arccos(d/2r)$ the target position of robot $v_1$ is contained inside or at its region boundary (and robot $v_1$ makes full progress). If $\theta > \arccos(d/2r)$ the target position of robot $v_1$ is contained outside its region boundary (and the progress of $v_1$ depends on $\theta$).

If $\theta \in [0, \arccos\left(\frac{d}{2r}\right)]$ then the target of robot $v_1$ is contained inside the intersection region $R_{v_1} = B_r[\varrho_{v_2}]$ and the progress of robot $v_1$ depends on no other robot. Therefore $\delta_{v_1} = d$, which immediately implies $\delta_V \geq d$.

If $\theta \in (\arccos\left(\frac{d}{2r}\right), \frac{\pi}{2}]$ then the target of robot $v_1$ is outside the intersection region $R_{v_1} = B_r[\varrho_{v_2}]$ and the progress of robot $v_1$ depends on robot $v_2$. Therefore, the progress of robot $v_1$ is equal to $\delta_{v_1} = d - (x - r) = d + r - x$ where $x$ is the distance between $v_1$'s target and $v_2$'s position (i.e., $x = \|\varrho_{v_2} - \tau_{v_1}\|$). Using the cosine law (see Figure 4-3) we have that $x = \sqrt{r^2 + d^2 - 2rd\cos\theta}$ and thus $\delta_{v_1} = d + r - \sqrt{r^2 + d^2 - 2rd\cos\theta}$. The case of robot $v_n$ is symmetric. Namely, the target of robot $v_n$ is outside the intersection region $R_{v_n} = B_r[\varrho_{v_{n-1}}]$ (i.e., the progress of robot $v_n$ depends on robot $v_{n-1}$) and its progress is $\delta_{v_n} = d + r - \sqrt{r^2 + d^2 + 2rd\cos\theta}$.

Therefore, if $\theta \in (\arccos\left(\frac{d}{2r}\right), \frac{\pi}{2}]$ the progress of the configuration is $\delta_V = \delta_{v_1} + \delta_{v_n} = 2r + 2d - \sqrt{r^2 + d^2 - 2rd\cos\theta} - \sqrt{r^2 + d^2 + 2rd\cos\theta}$. Using Fermat's theorem we can confirm that this function attains its minimum at $\theta = \frac{\pi}{2}$ when $\delta_V = 2r + 2d - 2\sqrt{r^2 + d^2}$. Observe that this function is monotone increasing with respect to $r$, since its derivative with respect to $r$ is strictly positive. Finally, since $d \leq r$ by assumption, then we can let $r = d$ which results in $\delta_V \geq 2d + 2d - 2\sqrt{2d^2} = (4 - 2\sqrt{2})d \geq d$. □

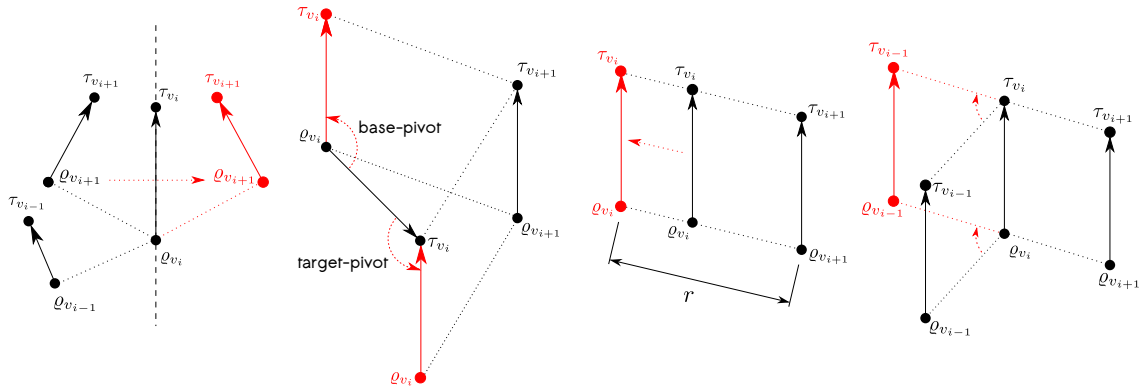
In the remaining subsections we will sequentially remove properties 1-6, showing that the progress of a configuration cannot be decreased by *not* satisfying these properties. To this end, we define a series of primitive operations that can be applied to the robots. We then use these operations to transform a configuration that does not satisfy a particular property, to a configuration that *does* satisfy it, showing that the transformation does not increase progress.

#### 4.4.3.2 Primitive Operations

We describe four basic operations that will be used in later sections. We highlight that some of this operations, by definition, can only be applied to configurations that satisfy certain properties. At its core, each operation relies on applying some rigid transformation (i.e., reflection, rotation or translation) to part of the configuration. Each of these operations satisfies the following proposition by construction.

**Proposition 4.18.** *Applying (when possible) one of the primitive operations to a robot in a line configuration that satisfies the weak assumption results in a configuration which also satisfies the weak assumption.*

In other words, these operations preserve the weak assumption.




(a) Reflect $v_i$ (b) Pivot $v_i$ (c) Stretch $v_i$ (d) Align $v_i$

Figure 4-4: The four primitive operations that can be applied to robot $v_i$. The configuration before the operation is depicted in black, and the configuration after the operation is depicted in red. To transform one configuration to another we will describe a choreographed sequences of these operations.

**Reflect.** The *reflect*$(v_i)$ operation reflects the robots the positions and the targets of the robots $v_{i+1}, \ldots, v_n$ on the line that goes through $\tau_{v_i}$ and $\varrho_{v_i}$. By definition applying the reflect operation on an unbalanced robot makes it balanced (see Figure 4-4a).

**Pivot.** If robots $v_i$ and $v_{i+1}$ are $d$-bounded, we define two pivot operations on robot $v_i$ that make the target vectors of robot $v_i$ and robot $v_{i+1}$ parallel. The *base-pivot*$(v_i)$ operation rotates the positions and the targets of the robots $v_1, \ldots, v_i$ around $\varrho_{v_i}$ until the target vectors of $v_i$ and $v_{i+1}$ have the same direction, i.e., until the vectors $\tau_{v_i} - \varrho_{v_i}$ and $\tau_{v_{i+1}} - \varrho_{v_{i+1}}$ have the same direction. Similarly, the *target-pivot*$(v_i)$ operation rotates the positions and the targets of the robots $v_1, \ldots, v_i$ around $\tau_{v_i}$ until the target vectors of $v_i$ and $v_{i+1}$ have the same direction. The *max-pivot*$(v_i)$ operation is equivalent to base-pivot$(v_i)$ if $\left\|\varrho_{v_i} - \varrho_{v_{i+1}}\right\| > \left\|\tau_{v_i} - \tau_{v_{i+1}}\right\|$, and otherwise it is equivalent to target-pivot$(v_i)$ (see Figure 4-4b).

**Stretch.** If robot $v_i$ and robot $v_{i+1}$ are $d$-bounded and parallel we define the *stretch*$(v_i)$ operation that makes robots $v_i$ and $v_{i+1}$ separated. Specifically, if the length of the horizontal sides of the parallelogram between $v_i$ and $v_{i+1}$ is $\ell$ (i.e., if $\ell = \left\|\varrho_{v_i} - \varrho_{v_{i+1}}\right\| = \left\|\tau_{v_i} - \tau_{v_{i+1}}\right\|$) then the stretch$(v_i)$ operation translates the positions and the targets of the robots $v_1, \ldots, v_i$ by the vector $(r/\ell - 1)(\varrho_{v_i} - \varrho_{v_{i+1}})$ (see

Figure 4-4c).

**Align.** If robot $v_i$ and robot $v_{i+1}$ are $d$-bounded, parallel and separated, we define the *align*$(v_i)$ operation that makes robot $v_i$ straight (i.e., makes $\varrho_{v_i}, \varrho_{v_{i+1}}$ and $\varrho_{v_{i+2}}$ collinear). Specifically, the align$(v_i)$ operation rotates the position of the robots $v_1, \ldots, v_i$ around the position of robot $v_{i+1}$ until $\varrho_{v_i}$, $\varrho_{v_{i+1}}$ and $\varrho_{v_{i+2}}$ are collinear, without changing the length or direction of the target vectors (see Figure 4-4d).

**Geometric Properties of the Operations.** The next sections will use these operations to transform one configuration into another, showing that the progress of the resulting configuration is no greater than the progress of the original configuration. Here we prove a series of lemmas that will be useful for that purpose.

First we prove that the max-pivot operation satisfies the following property.

**Lemma 4.19.** *Let $C$ be a line configuration that is $d$-bounded. The configuration that results from the max-pivot$(v_i)$ operation does not decrease any of the lengths of the segments of the quadrilateral between $\varrho_{v_i}, \varrho_{v_{i+1}}, \tau_{v_{i+1}}$ and $\tau_{v_i}$. (i.e., the new distances $\|\varrho_{v_i} - \tau_{v_i}\|$, $\left\|\varrho_{v_{i+1}} - \tau_{v_{i+1}}\right\|$, $\left\|\varrho_{v_i} - \varrho_{v_{i+1}}\right\|$, $\left\|\tau_{v_i} - \tau_{v_{i+1}}\right\|$, $\left\|\varrho_{v_i} - \tau_{v_{i+1}}\right\|$ and $\left\|\varrho_{v_{i+1}} - \tau_{v_i}\right\|$ are larger than or equal to the corresponding old distances.)*

To prove Lemma 4.19 we will use the quadrilateral law (stated below, see [50] for short a proof).

**Quadrilateral Law.** *Consider a quadrilateral on points $a$, $b$, $c$ and $d$ with side lengths $ab$, $bc$, $cd$ and $da$, and diagonals $ac$ and $bd$. If $x \geq 0$ is the distance between the midpoints of the diagonals then $ab^2 + bc^2 + cd^2 + da^2 = ac^2 + bd^2 + 4x$.*

A quadrilateral is a parallelogram if and only if the distance between the midpoints of the diagonals equals zero (i.e., if $x = 0$). In this case the quadrilateral law reduces to Euclid's parallelogram law. Furthermore, if the parallelogram is a rectangle the two diagonals are of equal length and the parallelogram law reduces to Pythagoras' theorem. We leverage the quadrilateral law to prove Lemma 4.19.

*Proof.* Consider the quadrilateral with vertices $\varrho_{v_i}$, $\varrho_{v_{i+1}}$, $\tau_{v_{i+1}}$ and $\tau_{v_i}$. By assumption $v_i$ and $v_{i+1}$ are $d$-bounded and therefore $\|\varrho_{v_i} - \tau_{v_i}\| = \left\|\varrho_{v_{i+1}} - \tau_{v_{i+1}}\right\| = d$. We refer to these opposing sides of the quadrilateral as its vertical sides. We call the remaining opposite sides of the quadrilateral its horizontal sides, and they have lengths $\left\|\varrho_{v_i} - \varrho_{v_{i+1}}\right\| = h_1$ and $\left\|\tau_{v_i} - \tau_{v_{i+1}}\right\| = h_2$. Finally the diagonals of the quadrilateral have length $\left\|\varrho_{v_i} - \tau_{v_{i+1}}\right\| = \ell_1$ and $\left\|\varrho_{v_{i+1}} - \tau_{v_i}\right\| = \ell_2$.

By construction, the pivot operations on $v_i$ do not change the lengths of the vertical sides of the quadrilateral, and thus the sides $\|\varrho_{v_i} - \tau_{v_i}\|$ and $\left\|\varrho_{v_{i+1}} - \tau_{v_{i+1}}\right\|$ are unaffected by max-pivot($v_i$). The base-pivot($v_i$) operation transforms the quadrilateral into a parallelogram with vertical sides of length $d$, horizontal sides of length $h_1$, and diagonals of length $\ell_1$ and $\ell'_2$. Similarly, the target-pivot($v_i$) operation transforms the quadrilateral into a parallelogram with vertical sides of length $d$, horizontal sides of length $h_2$, and diagonals of length $\ell'_1$ and $\ell_2$.

Therefore max-pivot($v_i$) produces a parallelogram with horizontal sides of length $\max(h_1, h_2)$ without changing the length of the vertical sides. To complete the proof it suffices to show that the diagonals in the parallelogram that results from max-pivot($v_i$) are no smaller than the diagonals of the original quadrilateral.

Without loss of generality assume that $\max(h_1, h_2) = h_1$ (the other case is symmetric); therefore max-pivot($v_i$) is simply base-pivot($v_i$). In this case, it follows that $\ell'_1 = \ell_1$ and we need only to show $\ell'_2 \geq \ell_2$. Applying the quadrilateral law to the resulting parallelogram we have

$$2d^2 + 2h_1^2 = \ell_1^2 + {\ell'_2}^2, \text{ that is } \quad {\ell'_2}^2 = 2d^2 + 2h_1^2 - \ell_1^2.$$

Similarly, applying the quadrilateral law (where $x$ is the distance between the midpoints of the diagonals) to the original quadrilateral we have

$$2d^2 + h_1^2 + h_2^2 = \ell_1^2 + \ell_2^2 - 4x, \text{ that is } \quad \ell_2^2 = 2d^2 + h_1^2 + h_2^2 - \ell_1^2 - 4x.$$

Finally since by assumption $2h_1^2 \geq h_1^2 + h_2^2$ and $x \geq 0$ we have $\ell'_2 \geq \ell_2$. □

The following lemma captures a property guaranteed by the reflect operation when applied to an unbalanced robot.

**Lemma 4.20.** *Let $C$ be a line configuration where robot $v_i$ is unbalanced, let $C'$ be the configuration that results from reflect($v_i$), and let $v'_{i+1}$ in $C'$ represent robot $v_{i+1}$ in $C$. Then there is a point $o$ which lies in the line between $\varrho_{v_i}$ and $\tau_{v_i}$ such that $\varrho_{v_{i+1}}$ is a rotation of $\varrho_{v_{i-1}}$ around $o$ of angle $\theta$, $\varrho_{v'_{i+1}}$ is a rotation of $\varrho_{v_{i-1}}$ around $o$ of angle $\theta'$ and $\theta' \geq \theta$.*

*Proof.* Let $T$ be the triangle formed by $\varrho_{v_{i-1}}$, $\varrho_{v_{i+1}}$ and $\varrho_{v'_{i+1}}$ (Figure 4-5). There is a unique circumcircle that passes through all the vertices of $T$, and the origin of this circumcircle lies at the point where all the perpendicular bisectors of the triangle's sides meet.

Let $o$ be the origin of the circumcircle defined by $T$. Since the perpendicular bisector between $\varrho_{v_{i+1}}$ and $\varrho_{v'_{i+1}}$ is the axis of reflection, it follows that $o$ lies in the axis of reflection (i.e., $o$ lies in the line between $\varrho_{v_i}$ and $\tau_{v_i}$).

Since $o$ is the origin of a circumcircle that contains $\varrho_{v_{i-1}}$, $\varrho_{v_{i+1}}$ and $\varrho_{v'_{i+1}}$, it follows that $\varrho_{v_{i+1}}$ and $\varrho_{v'_{i+1}}$ are rotations of $\varrho_{v_{i-1}}$ around $o$.

The fact that the angle $\angle\varrho_{v_{i-1}}o\varrho_{v'_{i+1}}$ is greater than the angle $\angle\varrho_{v_{i-1}}o\varrho_{v_{i+1}}$ follows from the assumption that robot $v_i$ is unbalanced in $C$ and therefore $\varrho_{v_{i-1}}$ and $\varrho_{v_{i+1}}$ lie on the same side of the axis of reflection that divides the circumcircle defined by $T$, while $\varrho_{v_{i-1}}$ and $\varrho_{v'_{i+1}}$ lie on opposite sides of the axis of reflection (see Figure 4-5).

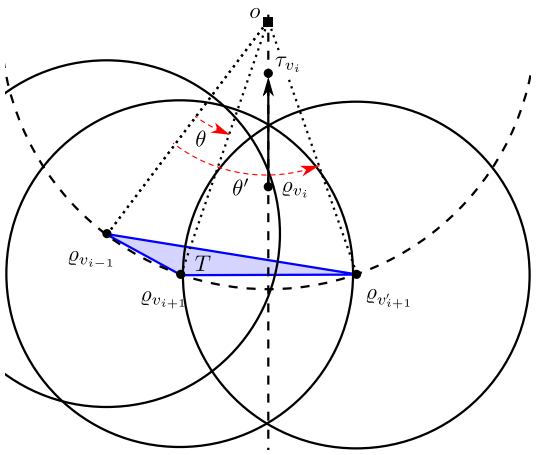



Figure 4-5: The target vector of robot $v_i$ is denoted by an arrow. The disks centered of radius $r$ centered at $\varrho_{v_{i-1}}$, $\varrho_{v_{i+1}}$ and $\varrho_{v'_{i+1}}$ are outlined with a black solid line. The triangle $T$ formed by $\varrho_{v_{i-1}}$, $\varrho_{v_{i+1}}$ and $\varrho_{v'_{i+1}}$ is denoted in blue. The circumcircle of $T$ is denoted with a dashed outline, and the origin of the circumcircle $o$ is depicted by a black square.

□

The following lemma shows that, as long as certain conditions are met, certain transformations are guaranteed not to increase the progress of some robots.

**Lemma 4.21.** *Let $C$ be a line configuration and let $v_i$ be a robot that does not depend on two neighbors in $C$. Let $C'$ be a line configuration where the robots $\{v_1, \ldots, v_i\}$ are exactly as in $C$ and the distance $\left\|\varrho_{v_{i+1}} - \tau_{v_i}\right\|$ is not decreased in $C'$. The progress of $v_i$ is not greater in $C'$ than it is in $C$.*

*Proof.* Assume robot $v_i$ depends on none of its neighbors in $C$. Since robot $v_i$ is not restricted by either neighbor in $C$ then it has maximum progress in $C$, and the progress of $v_i$ in $C'$ cannot be any greater.

Assume robot $v_i$ depends only on its neighbor $v_{i+1}$ in $C$. In this case the proposed target of robot $v_i$ in $C$ is the closest point to $\tau_{v_i}$ in $B_r[\varrho_{v_{i+1}}]$. Hence it follows that

by moving $\varrho_{v_{i+1}}$ further away from $\tau_{v_i}$ the progress of $v_i$ will decrease, and thus the progress of $v_i$ in $C'$ is not greater than the progress of $v_i$ in $C$.

Assume robot $v_i$ depends only on its neighbor $v_{i-1}$ in $C$. The restriction imposed by $v_{i-1}$ on $v_i$ in $C$ is the same as the restriction imposed by $v_{i-1}$ on $v_i$ in $C'$. Moving robot $v_{i+1}$ can only restrict the motion of $v_i$ further (since in $C$ robot $v_i$ does not depend on robot $v_{i+1}$ on any way). Therefore, the progress of $v_i$ in $C'$ is not greater than the progress of $v_i$ in $C$. □

Next, we leverage the previous lemma to prove a slightly different result.

**Lemma 4.22.** *Let $C$ be a line configuration and let $v_i$ be a robot that is balanced in $C$. Let $C'$ be a line configuration where the robots $\{v_1, \ldots, v_i\}$ are exactly as in $C$, the distance $\left\|\varrho_{v_{i+1}} - \tau_{v_i}\right\|$ is not decreased in $C'$, $v_i$ is balanced in $C'$, and $\left\|\varrho_{v_i} - \varrho_{v_{i+1}}\right\| = r$ in $C'$. The progress of $v_i$ is not greater in $C'$ than it is in $C$.*

*Proof.* First observe that Lemma 4.21 allows us to consider only the case when robot $v_i$ depends on both of its neighbors in $C$.

Let $q_{v_i}$ be the proposed target of $v_i$ in $C$. Since by assumption robot $v_i$ depends on both of its neighbors in $C$, then $q_{v_i}$ lies in the intersection of the two circles of radius $r$ centered at $\varrho_{v_{i-1}}$ and $\varrho_{v_{i+1}}$.

The rest of the proof follows by a geometric argument (see Figure 4-6). Let $\varrho'_{v_{i+1}}$ be the position of robot $v_i$ in $C'$ We argue that $\varrho_{v'_{i+1}}$ is the result of two rotations of $\varrho_{v_{i+1}}$, and the progress of $v_i$ is monotonically decreasing during each individual rotation. Specifically, first $\varrho_{v_{i+1}}$ is rotated counter-clockwise around $\tau_{v_i}$ until $\left\|\varrho_{v_i} - \varrho_{v_{i+1}}\right\| = r$ (see Figure 4-6). The progress of robot $v_i$ during this rotation is monotonically decreasing. Next $\varrho_{v_{i+1}}$ is rotated clockwise around $\varrho_{v_i}$, but without crossing the line between $\varrho_{v_i}$ and $\tau_{v_i}$, since by assumption robot $v_i$ is balanced (see Figure 4-6). As before, the progress of $v_i$ is monotonically decreasing during this second rotation.

□

#### 4.4.3.3 Straight Transformation

This section describes a transformation that uses the align operation to transform a $d$-bounded, balanced, parallel and separated line configuration to a $d$-bounded, balanced, parallel, separated and straight line configuration (i.e., a worst-case configuration). We prove that this transformation preserves the weak-assumption (trivial) and that the progress of the resulting configuration is no greater than the progress of the initial configuration.

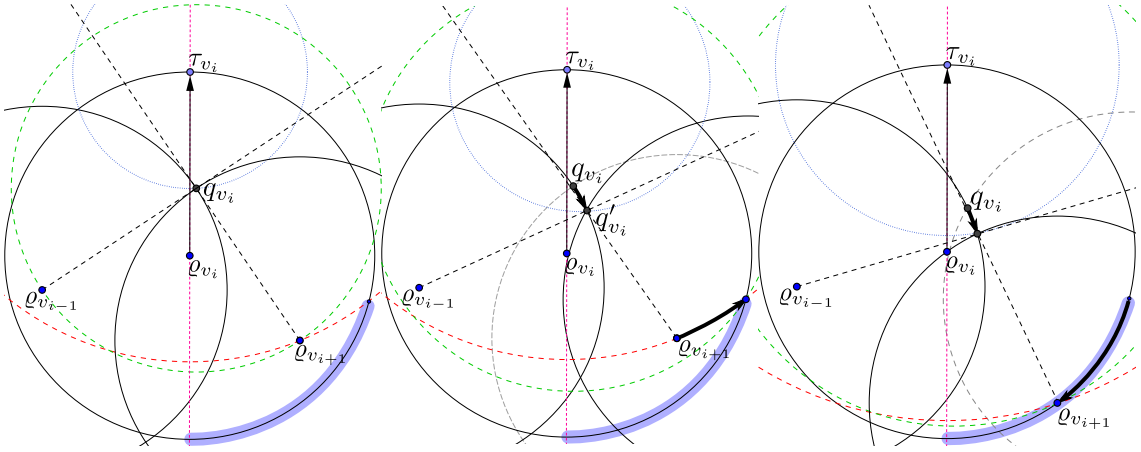



Figure 4-6: The first diagram shows a configuration where robot $v_i$ depends on both of its neighbors. The thick blue line represents the possible places occupied by $\varrho_{v'_{i+1}}$. The middle diagram shows a counter-clockwise rotation of $\varrho_{v_{i+1}}$ around $\tau_i$. The last diagram shows a clockwise rotation of $\varrho_{v_{i+1}}$ around $\varrho_{v_i}$.

First observe that in a $d$-bounded, balanced, parallel and separated line configuration, the only parameters relevant to determine the progress of a particular robot, are the relative angles (with respect to its target vector) to its neighbors. Specifically, given a line configuration which satisfies the properties above, its progress is determined by the number of robots $n$, the length $d$ of the target vectors, and $n-1$ angles $\theta_1, \ldots, \theta_{n-1}$, where $\theta_i \in [0, \pi]$ represents the angle between robot $v_i$'s target position and the position of its neighbor robot $v_{i+1}$ (see Figure 4-8).

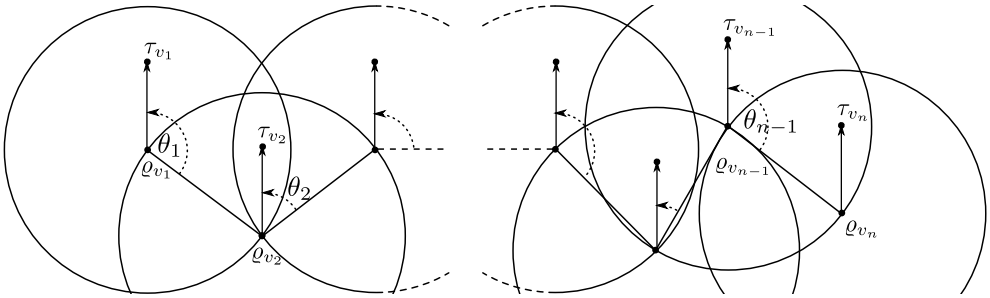



Figure 4-7: A $d$-bounded, balanced, parallel and separated line configuration.

This allows us to use trigonometry to derive analytically the progress of a robot in such a configuration as a function of the relative angle of its target vector to its neighbors; endpoint robots have only one neighbor, and therefore only one angle is

required.

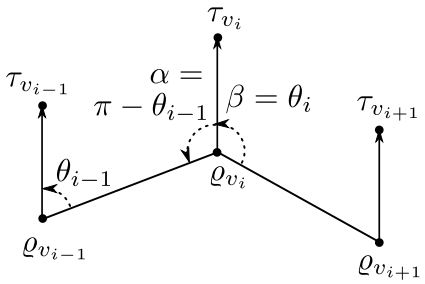



Figure 4-8: Inner robot $v_i$.

Consider an inner robot $v_i$, and let $\alpha = \pi - \theta_{i-1}$ be the counterclockwise angle to its neighbor $v_{i-1}$ and let $\beta = \theta_i$ be the clockwise angle to its neighbor $v_{i+1}$ (see Figure 4-8). To analyze the progress of robot $v_i$ we need only to do a case analysis depending on the relationship between $\alpha$ and $\beta$. In particular it suffices to consider five cases.

1. If $\alpha + \beta \geq \pi$ then robot $v_i$ is completely immobilized by its neighbors and its progress is 0.
2. If $\alpha \leq \arccos \frac{d}{2r}$ and $\beta \leq \arccos \frac{d}{2r}$ then robot $v_i$ is not restricted by either neighbor and its progress is $d$.
3. If $\alpha > \arccos \frac{d}{2r}$ and $\sin(\alpha + \beta) \geq \frac{d}{r} \sin \alpha$ the progress of robot $v_i$ depends only on the angle $\alpha$ to robot $v_{i-1}$, and its progress is given by the function $\Gamma_{single}(\alpha) = r + d - \sqrt{r^2 + d^2 - 2rd \cos \alpha}$.
4. If $\beta > \arccos \frac{d}{2r}$ and $\sin(\alpha + \beta) \geq \frac{d}{r} \sin \beta$ the progress of robot $v_i$ depends only on the angle $\beta$ to robot $v_{i+1}$ and its progress is given by the function $\Gamma_{single}(\beta)$.
5. If $\sin(\alpha + \beta) \leq \frac{d}{r} \min(\sin \alpha, \sin \beta)$ and $\alpha + \beta \leq \pi$ then the progress of robot $v_i$ depends on the angle to both of its neighbors and is given by the function $\Gamma_{both}(\alpha, \beta) = d - \sqrt{2r^2 + d^2 - 2rd(\cos \alpha + \cos \beta) + 2r^2 \cos(\alpha + \beta)}$.

Therefore the progress of an inner robot $v_i$ can be captured by a piecewise function $\Gamma(\alpha, \beta)$ described in Figure 4-10.

The progress of an endpoint robot $v_i$ can only be a function of the angle to its only neighbor, but the situation is still similar. Specifically, the progress of robot $v_1$ is a function of the angle $\theta_1$ to its only neighbor $v_2$, and is described by the function $\Gamma(0, \theta_1)$. Similarly, the progress of robot $v_n$ is a function of the angle $\theta_{n-1}$ to its only neighbor $v_{n-1}$ and is described by the function $\Gamma(\pi - \theta_{n-1}, 0)$.

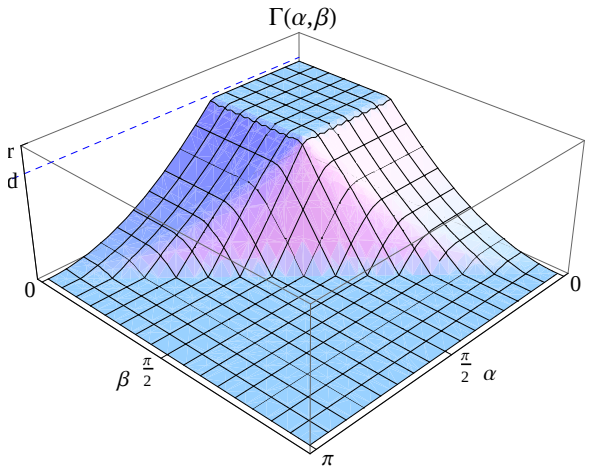
The next lemma shows that after applying each align operation, the resulting configuration is "closer" to being straight, and its progress is no greater than the

$$\Gamma_{single}(\alpha) = r + d - \sqrt{r^2 + d^2 - 2rd\cos\alpha}$$

$$\Gamma_{both}(\alpha, \beta) = d - \sqrt{2r^2 + d^2 - 2rd(\cos\alpha + \cos\beta) + 2r^2\cos(\alpha + \beta)}$$

$$\Gamma(\alpha, \beta) = \begin{cases} 0 & \alpha + \beta \geq \pi \\ d & \alpha \leq \arccos\frac{d}{2r} \text{ and } \beta \leq \arccos\frac{d}{2r} \\ \Gamma_{single}(\alpha) & \alpha \geq \arccos\frac{d}{2r} \text{ and } \sin(\alpha+\beta) \geq \frac{d}{r}\sin\alpha \\ \Gamma_{single}(\beta) & \beta \geq \arccos\frac{d}{2r} \text{ and } \sin(\alpha+\beta) \geq \frac{d}{r}\sin\beta \\ \Gamma_{both}(\alpha, \beta) & \alpha + \beta < \pi \text{ and } \sin(\alpha+\beta) < \frac{d}{r}\min(\sin\alpha, \sin\beta) \end{cases}$$

(a) Piecewise progress function




(a) Plot of piecewise progress function

Figure 4-10: For most of our proofs the exact shape of the progress function $\Gamma$ will not be important. Instead it will be sufficient for us to observe that $\Gamma$ is monotonically decreasing with respect to $\alpha$ and $\beta$.

progress of the initial configuration.

**Lemma 4.23.** *Fix $i \in \{2, \ldots, n-1\}$. Let $C$ be a line configuration that is $d$-bounded, balanced, separated, parallel and where the positions of the robots $v_1, \ldots, v_i$ are collinear. Let $C'$ be the configuration that results from the operation align$(v_i)$. Then $C'$ is a line configuration that is $d$-bounded, balanced, separated, parallel, where the positions of the robots $v_1, \ldots, v_{i+1}$ are collinear, and whose progress is no greater than the progress of $C$.*

*Proof.* The fact that $C'$ is $d$-bounded, balanced, separated and parallel follows from the fact that the align operation preserves all these properties. Moreover by definition of the align operation, it follows that the positions of the robots $v_1, \ldots, v_{i+1}$ are collinear. We need only to show that the progress of $C'$ is no greater than the progress of $C$.

Let $\theta_1, \ldots, \theta_{n-1}$ be the angles that define configuration $C$, and let $\theta'_1, \ldots, \theta'_{n-1}$ be the angles that define the resulting configuration $C'$. Since the robots $v_1, \ldots, v_i$ are collinear in $C$ it follows that $\theta_1 = \theta_2 = \ldots = \theta_{i-1}$ and the robots $v_2, \ldots, v_{i-1}$ are immobilized and have progress equal to zero in $C$. Similarly since the robots $v_1, \ldots, v_{i+1}$ are collinear in $C'$ it follows that $\theta'_1 = \theta'_2 = \ldots = \theta'_i$ and the robots $v_2, \ldots, v_i$ are immobilized and have progress equal to zero in $C'$. Also, by construction of the align operation it follows that $\theta_j = \theta'_j$ for $j \in \{i, \ldots, n-1\}$ so the progress of the robots $v_{i+1}, \ldots, v_n$ is the same in $C$ and $C'$.

Therefore, to prove the theorem we need only to show that the progress of robot $v_1$ in $C'$ is no greater than the combined progress of robots $v_1$ and $v_i$ in $C$. To prove this we proceed by a case analysis depending on the relationship of $\theta_{i-1}$ and $\theta_i$.

1. Robots $v_{i-1}, v_i$ and $v_{i+1}$ are collinear in $C$ ($\theta_{i-1} = \theta_i$). In this case $C$ is identical to $C'$ and this concludes our proof.

2. Robots $v_{i-1}$,$v_i$ and $v_{i+1}$ are 'convex' in $C$ ($\theta_{i-1} < \theta_i$). In this case robot $v_i$ is immobilized in $C$, so we need to show only that the progress of $v_1$ is not greater in $C'$ than in $C$. This follows from the fact that $\theta_1 = \theta_{i-1} < \theta_i = \theta'_1$ and the function $\Gamma_{single}(\theta)$ is monotonically decreasing with $\theta$ (recall the progress of $v_1$ in $C'$ is $\Gamma_{single}(\theta'_1)$ and the progress of $v_1$ in $C$ is $\Gamma_{single}(\theta_1)$).

3. Robots $v_{i-1}$, $v_i$ and $v_{i+1}$ are 'concave' in $C$ ($\theta_{i-1} > \theta_i$). We divide further the cases based on which neighbors the progress of robot $v_i$ depends on in $C$.

   3.1. Robot $v_i$ doesn't depend on robot $v_{i+1}$ in $C$. In this case the robots $v_1, \ldots, v_i$ behave in $C$ as if they were in a worst-case line configuration,

and we have already showed such configurations have progress at least $d$. Therefore the combined progress of $v_1$ and $v_i$ in $C$ is at least $d$, and since the progress of $v_1$ in $C'$ is at most $d$, this concludes our proof.

3.2. Robot $v_i$ depends only on robot $v_{i+1}$ in $C$. The angle between robot $v_i$ and robot $v_{i+1}$ in $C$ is the same as the angle between robot $v_1$ and $v_2$ in $C'$. Therefore robot $v_1$ depends only on robot $v_2$ in $C'$ and the progress of robot $v_1$ in $C'$ is the same as the progress of robot $v_i$ in $C$. Since the progress of robot $v_1$ in $C$ is at least zero this concludes our proof.

3.3. Robot $v_i$ depends on both robot $v_{i-1}$ and robot $v_{i+1}$ in $C$. Showing that the combined progress of $v_i$ and $v_1$ in $C$ is greater than the progress of $v_1$ in $C'$ is equivalent to showing $\Gamma_{both}(\pi - \theta_{i-1}, \theta_i) + \Gamma_{single}(\theta_{i-1}) - \Gamma_{single}(\theta_i) \geq 0$. For this we analytically evaluate the minimum of this function in the appropriate interval. This yields two minima; one at $\theta_i = \theta_{i-1} = \frac{\pi}{2}$ and the other at $\theta_{i-1} = \pi$. Since the function evaluates to zero at both minima this concludes our proof.

□

Finally we leverage Lemma 4.23 to prove our main theorem.

**Theorem 4.24.** *Let $C$ be a line configuration that is $d$-bounded, balanced, parallel, separated and satisfies the weak assumption. There is a line configuration $C'$ that is $d$-bounded, balanced, parallel, separated, straight, satisfies the weak assumption (i.e., a worst-case configuration) and whose progress is no greater than the progress of $C$.*

*Proof.* Let $C'$ be the configuration that results from applying inductively the align operation on robot $v_2, \ldots, v_{n-1}$ in $C$. From Lemma 4.23 it follows that $C'$ is $d$-bounded, balanced, parallel, separated, straight, satisfies the weak assumption and its progress is at most the progress of $C$. □

#### 4.4.3.4 Parallel-Separated Transformation

This section describes a transformation that uses the max-pivot, the stretch and the reflect operations to transform a $d$-bounded, balanced line configuration to a $d$-bounded, balanced, parallel and separated line configuration. We prove this transformation preserves the weak assumption and that the progress of the resulting configuration is no greater than the progress of the initial configuration.

We use Lemma 4.19 to prove that under some conditions a combination of the max-pivot($v_i$), the stretch($v_i$) and the reflect($v_i$) operations does not increase the progress of robot $v_i$ or robot $v_{i+1}$.

**Lemma 4.25.** *Let $C$ be a line configuration that is $d$-bounded, balanced and satisfies the weak assumption. Let $C'$ be the configuration that results from the max-pivot($v_i$), the stretch($v_i$) and (if needed to balance $C'$) the reflect($v_i$) and reflect($v_{i+1}$) operations. The progress of $C'$ is no greater than the progress of $C$.*

*Proof.* By construction the reflect($v_i$) and reflect($v_{i+1}$) operations affect only the progress of robot $v_i$ and $v_{i+1}$ respectively, while the max-pivot($v_i$) and the stretch($v_i$) operations also affect only the progress of robots $v_i$ and $v_{i+1}$. Therefore, given the combination of operations we consider, the progress of robot $v_j$ for $j \notin \{i, i+1\}$ is unchanged in $C$ and $C'$. Therefore to prove the lemma we need only to show that the combined progress of robot $v_i$ and robot $v_{i+1}$ is not greater in $C'$ than it is in $C$. First we prove the following claim.

**Claim 4.26.** *As a result of the max-pivot($v_i$), the stretch($v_i$), and (if needed) the reflect($v_i$) and reflect($v_{i+1}$) operations, robot $v_i$ and $v_{i+1}$ are balanced in $C'$ and the quadrilateral with vertices $\varrho_{v_i}, \varrho_{v_{i+1}}, \tau_{v_{i+1}}$ and $\tau_{v_i}$ is transformed to a parallelogram where the length of all the sides and diagonals are not decreased, and the length of the horizontals is $r$.*

*Proof of Claim 4.26.* The max-pivot($v_i$) operation is guaranteed to produce a parallelogram, and Lemma 4.19 implies that the lengths of the sides and diagonals of this parallelogram are larger than or equal to the length of the sides and diagonals of the original quadrilateral. Moreover the stretch($v_i$) operation only increases further the lengths of the sides and diagonals of the parallelogram and ensure the horizontal sides have length $r$. Finally if as a result of the max-pivot($v_i$) operation robot $v_i$ or $v_{i+1}$ became unbalanced, the reflect operations will make them balanced without changing any of the lengths of the sides and diagonals of the parallelogram. □

Next we use Claim 4.26 to apply Lemma 4.22 and show that the combined progress of robot $v_i$ and robot $v_{i+1}$ is not increased in the resulting configuration.

The only change observed by robot $v_i$ in $C'$ is the relative position of robot $v_{i+1}$. Claim 4.26 implies the distance $\left\|\varrho_{v_{i+1}} - \tau_{v_i}\right\|$ is not decreased in $C'$, robot $v_i$ is balanced in $C'$ and $\left\|\varrho_{v_{i+1}} - \varrho_{v_i}\right\| = r$ in $C'$. Therefore Lemma 4.22 implies the progress of $v_i$ is not greater in $C'$ than it is in $C$. The argument for robot $v_{i+1}$ is symmetric. Specifically, the only change observed by robot $v_{i+1}$ in $C'$ is the relative position of robot $v_i$. Claim 4.26 implies the distance $\left\|\varrho_{v_i} - \tau_{v_{i+1}}\right\|$ is not decreased in $C'$, robot $v_{i+1}$ is balanced in $C'$, and $\left\|\varrho_{v_{i+1}} - \varrho_{v_i}\right\| = r$ in $C'$. Therefore Lemma 4.22 implies the progress of $v_{i+1}$ is not greater in $C'$ than it is in $C$. □

Finally we leverage Lemma 4.25 to prove the main theorem of this subsection.

**Theorem 4.27.** *Let $C$ be a line configuration that is $d$-bounded, balanced and satisfies the weak assumption. There is a line configuration $C'$ that is $d$-bounded, balanced, parallel, separated, satisfies the weak assumption and whose progress is at most the progress of $C$.*

*Proof.* Let $C'$ be the configuration that results from applying inductively the max-pivot, the stretch, and (when needed to balance) the reflect operation to each robot in $C$ as in Lemma 4.25.

By construction it follows that $C'$ is $d$-bounded, balanced, parallel, separated and satisfies the weak assumption. Lemma 4.25 implies that the progress of $C'$ is at most the progress of $C$. □

#### 4.4.3.5 Balance Transformation

This section describes a transformation that uses the reflect operation to transform a $d$-bounded line configuration to a $d$-bounded and balanced line configuration. We prove that this transformation preserves the weak-assumption and that the progress of the resulting configuration is no greater than the progress of the initial configuration.

First we show that the reflect operation does not increase the progress of a robot.

**Lemma 4.28.** *Let $C$ be a line configuration and let $v_i$ be an unbalanced robot in $C$. Let $C'$ be the configuration that results from the reflect($v_i$) operation. The progress of $C'$ is no greater than the progress of $C$.*

*Proof.* By construction of the reflect($v_i$) operation it follows that the progress of robot $v_j$ for $j \neq i$ is unchanged in $C$ and $C'$. Therefore we need only to show that the progress of robot $v_i$ is not greater in $C'$ than it is in $C$.

Observe that the reflect($v_i$) operation does not change the relative position of robot $v_i$ and $v_{i-1}$, and while it does change the relative position between robot $v_i$ and $v_{i+1}$, it does not change the distance $\left\|\varrho_{v_{i+1}} - \tau_{v_i}\right\|$. This allows us to leverage Lemma 4.21 and consider only the case when robot $v_i$ depends on both $v_{i-1}$ and $v_{i+1}$ in $C$.

Let $v'_{i+1}$ represent robot $v_{i+1}$ after the reflection. Let $R_{v_i}$ represent the intersection of the closed disks of radius $r$ centered at $\varrho_{v_{i-1}}$ and $\varrho_{v_{i+1}}$. Similarly, let $R'_{v_i}$ represent the intersection of the closed disks of radius $r$ centered at $\varrho_{v_{i-1}}$ and $\varrho_{v'_{i+1}}$. The endpoint of the trajectory output by $v_i$ in $C$ is the point in $R_{v_i}$ closest to $\tau_{v_i}$. Similarly, the endpoint of the trajectory output by $v_i$ in $C'$ is the point in $R'_{v_i}$ closest to $\tau_{v_i}$. Thus, to prove that the progress of $v_i$ is not greater in $C'$ than in $C$ it suffices to show that $R'_{v_i} \subseteq R_{v_i}$.

Consider the points $\varrho_{v_{i-1}}$, $\varrho_{v_{i+1}}$ and $\varrho_{v'_{i+1}}$ around which the closed disks that produced $R_{v_i}$ and $R'_{v_i}$ are centered. Lemma 4.20 implies there exists a point $o$ which lies in the line between $\varrho_{v_i}$ and $\tau_{v_i}$ such that $\varrho_{v_{i+1}}$ is a rotation of $\varrho_{v_{i-1}}$ around $o$ of angle $\theta$, $\varrho_{v'_{i+1}}$ is a rotation of $\varrho_{v_{i-1}}$ around $o$ of angle $\theta'$, and $\theta' > \theta$. To prove that $R'_{v_i}$ is contained in $R_{v_i}$ it suffices to show that the point of rotation $o$ lies outside the closed disks that were used to produce these intersection regions. However, this follows from the assumption that $v_i$ depends on both of its neighbors in $C$, and that $v_i$ is unbalanced in $C$.

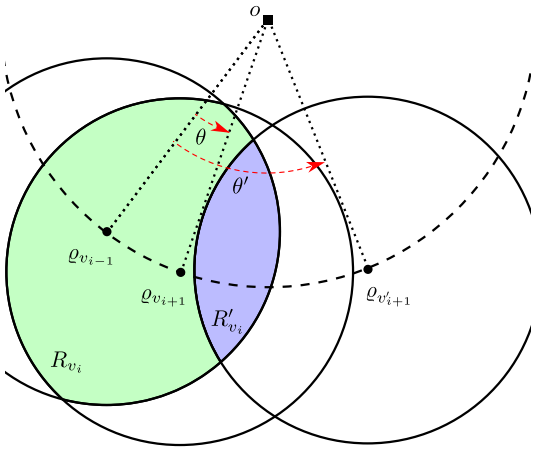



Figure 4-11: The origin $o$ is denoted by a black square. The disks centered of radius $r$ centered at $\varrho_{v_{i-1}}, \varrho_{v_{i+1}}$ and $\varrho_{v'_{i+1}}$ are outlined with a black solid line. The regions $R_{v_i}$ and $R'_{v_i}$ are shaded in light green and blue respectively.

□

Finally we can leverage Lemma 4.28 to prove the following theorem.

**Theorem 4.29.** *Let $C$ to be any line configuration that satisfies the weak assumption. There is a line configuration $C'$ that is balanced, satisfies the weak assumption and whose progress is no greater than the progress of $C$. Moreover, if $C$ is $d$-bounded then $C'$ is also $d$-bounded.*

*Proof.* Let $C'$ be the configuration that results from applying the reflection operation to every robot which is unbalanced in $C$.

By construction of the reflect operation it follows that $C'$ is balanced and satisfies the weak assumption. Similarly, the fact that the reflect operation doesn't change the length of any of the target vectors, implies that if $C$ is $d$-bounded then $C'$ is also

$d$-bounded. Finally, Lemma 4.28 implies the progress of $C'$ is at most the progress of $C$. □

#### 4.4.3.6 Bounded Transformation

This section shows how to transform a line configuration that satisfies the weak assumption, into a line configuration which is $d$-bounded (where $d = \min_{v \in V} \|\tau_v - \varrho_v\|$), satisfies the weak assumption, and has progress no greater than the original configuration.

**Theorem 4.30.** *Let $C$ be a line configuration that satisfies the weak assumption, and let $d \leq \min_{v \in V} \|\tau_v - \varrho_v\|$. There is a line configuration $C'$ that is $d$-bounded, satisfies the weak assumption, and whose progress is no greater than the progress of $C$.*

*Proof.* Let $\tau'_v$ be the position reached by robot $v \in V$ when following its trajectory in $C$ and stopping after having traveled exactly distance $d$. This is possible by our choice of $d$, since every robot in $C$ has a trajectory of length at least $d$. Let $C'$ be a configuration which is identical to $C$ except that every robot $v \in V$ has $\tau'_v$ as its target position instead of $\tau_v$.

To complete the theorem we only need to show that (1) $C'$ is $d$-bounded, (2) $C'$ satisfies the weak assumption, (3) the progress of $C'$ is at most the progress of $C$. We prove each of these properties separately.

1. By definition of $\tau'_v$ we have that $\|\varrho_v - \tau'_v\| = d$, so it follows that $C'$ is $d$-bounded.

2. By assumption $C$ satisfies the weak assumption. Since the configuration $C'$ results from having each robot travel at the same speed for a fraction of their trajectories in $C$, it follows that $C'$ also satisfies the weak assumption.

3. To prove that the progress of $C'$ is not larger than the progress of $C$, we show that the progress of each individual robot $v \in V$ is not larger in $C'$ than in $C$. First observe that by our choice of $d$ we have that $\tau'_v = \tau_v(t) = \varrho_v + t(\tau_v - \varrho_v)$ for some $t \in [0, 1]$.

   Let $R_v$ be the intersection region of robot $v$ in $C$ and $C'$, and let $q_v$ and $q_v(t)$ be the proposed targets by robot $v$ in $C$ and $C'$ respectively.

   By definition the progress of robot $v$ in $C$ is $\delta_v = \|\varrho_v - \tau_v\| - \|q_v - \tau_v\|$, and the progress of robot $v$ in $C'$ is $\delta'_v = \|\varrho_v - \tau_v(t)\| - \|q_v(t) - \tau_v(t)\|$. We will prove that $\delta_v \geq \delta'_v$.

From the definition of the algorithm, $q_v$ is the point in $R_v$ closest to $\tau_v$, and $q_v(t)$ is the point in $R_v$ closest to $\tau_v(t)$. Therefore, it immediately follows that $\|q_v - \tau_v\| \leq \|q_v(t) - \tau_v\|$. From the triangle inequality we have that $\|q_v(t) - \tau_v\| \leq \|q_v(t) - \tau_v(t)\| + \|\tau_v(t) - \tau_v\|$. By the definition of $\tau_v(t)$ we also have that $\|\tau_v(t) - \tau_v\| = (1-t)\,\|\varrho_v - \tau_v\|$. Putting these observations together we have $\|q_v - \tau_v\| \leq \|q_v(t) - \tau_v(t)\| + (1-t)\,\|\varrho_v - \tau_v\|$. Now the statement follows by algebraic manipulation.

$$\begin{aligned}
\delta_v &= \|\varrho_v - \tau_v\| - \|q_v - \tau_v\| \\
&\geq \|\varrho_v - \tau_v\| - \|q_v(t) - \tau_v(t)\| - (1-t)\,\|\varrho_v - \tau_v\| \\
&= t\,\|\varrho_v - \tau_v\| - \|q_v(t) - \tau_v(t)\| \\
&= \|\varrho_v - (\varrho_v + t(\tau_v - \varrho_v))\| - \|q_v(t) - \tau_v(t)\| \\
&= \|\varrho_v - \tau_v(t)\| - \|q_v(t) - \tau_v(t)\| = \delta'_v
\end{aligned}$$

□

#### 4.4.3.7 Line Transformation

This section shows that in any configuration without cyclic dependencies there is a subset of robots that are connected in a line and whose progress does not depend on any other robot. In other words, any configuration without cyclic dependencies contains a subset of robots that, from a progress standpoint, behave as if they were on a line configuration. This implies that a lower bound on the progress of line configurations is also a lower bound on the progress of configurations without cyclic dependencies.

The *out-neighbors* of a vertex $v \in V$ in a dependency graph $(V, A)$ are defined as $N^+(v) = \{u \mid (v,u) \in A\}$ and represents precisely the set of robots that robot $v$ depends on. The *out-degree* of a vertex $v \in V$ is denoted as $deg^+(v) = |N^+(v)|$ and represents the number of robots on which robot $v$ depends. We analogously define the *in-neighbors* and *in-degree* of a vertex $v \in V$ in a dependency graph, denoted by $N^-(v)$ and $deg^-(v)$ respectively.

A subset of vertices $V' \subseteq V$ is an *independent component* in a configuration if in the corresponding dependency graph there is no directed edge that originates in $V'$ and ends in $V \setminus V'$. Observe that by definition, if a subset of robots forms an independent component in a configuration, then their progress is not affected by any other robot in the system. This allows us to analyze the progress of an independent

component without considering the state of any robot outside the component. We will show that every configuration without cyclic dependencies has an independent component which corresponds to a line graph.

First we show that every vertex in the dependency graph has an out-degree of at most two. Recall that by definition $q_v$ is the point inside $R_v$ which minimizes the distance to the target position $\tau_v$. Moreover, $R_v$ is defined as the intersection of a set of closed disks of radius $r$ centered at the initial positions of its locally selected neighbors $S_v$. Therefore the boundary of $R_v$ is described by a collection of circular arc segments, where each arc segment belongs to the boundary of a closed disk of radius $r$ centered at the initial position of one of the locally selected neighbors $S_v$. We leverage this observation in the next proposition.

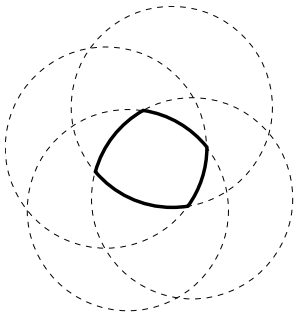


Figure 4-12: A thick line delineates the boundary of a region defined by the intersection of four disks denoted with dashed lines. A point contained in the region is either at the region boundary, or completely inside the region. A point in the region boundary lies either at the intersection of two circle arcs, or on the arc of a single circle.

**Proposition 4.31.** *For every $v \in V$ we have that $deg^+(v) \leq 2$.*

*Proof.* Fix a robot $v \in V$. We proceed by cases depending on where the target position $\tau_v$ is located with respect to $R_v$.

If $\tau_v$ is contained in $R_v$, then by construction $q_v = \tau_v$ and this implies the output of $v$ depends on no other robot (that is $deg^+(v) = 0$). On the other hand if $\tau_v$ lies outside $R_v$, then $q_v$ must lie on the boundary point of the region $R_v$ closest to $\tau_v$. If $q_v$ lies on the intersection point of two arc segments then the trajectory of robot $v$ can depend only on the two neighbors which are responsible for those arc segments (and thus $deg^+(v) = 2$). Otherwise, $q_v$ lies on a single arc segment on the boundary of $R_v$, in which case the trajectory of robot $v$ depends only on a single neighbor (and thus $deg^+(v) = 1$). □

We leverage Proposition 4.31 to show the main result of this subsection.

**Theorem 4.32.** *In configurations without cyclic dependencies, there is a subset of robots that form an independent component and are connected in a line.*

Before proving this theorem, we state and prove a well known property of directed acyclic graphs (or DAGs for shorthand notation). A sink vertex is defined as a vertex with no outgoing edges.

**Fact 1.** A finite DAG contains at least one sink vertex.

*Proof.* Consider a directed walk starting at any vertex of a finite DAG. As long as the walk doesn't hit a vertex with no outgoing edges (i.e., a sink vertex), the walk is continued by visiting one of the outgoing neighbors of the current vertex. If the directed walk is finite then the claim follows. If the directed walk is infinite it must visit at least one vertex more than once, which implies the graph contains a directed cycle – a contradiction. □

With this result in place, we are ready to prove Theorem 4.32.

*Proof.* A dependency graph with no cyclic dependencies might not be a DAG since it can contain simple cycles of two vertices. In what follows we describe how to transform a dependency graph with no cyclic dependencies to a finite DAG.

The transformation consists of a series of edge contractions between pairs of vertices that form a directed cycle of length two, until no such contractions are possible. Since by assumption there are no cyclic dependencies, it follows that the dependency graph has no simple directed cycles of length greater than two and the transformation results in a finite DAG. From the definition of the transformation, it also follows that any sink vertex in the resulting DAG corresponds to an independent component in the dependency graph (i.e., no outgoing edges).

We claim that any vertex in the resulting DAG represents a set of vertices in the dependency graph which are connected in a line graph. This claim implies the theorem since a sink vertex in the resulting DAG (which is guaranteed to exist by Fact 1) represents a set of vertices in the dependency graph which form an independent component and are connected in a line.

We prove this claim by an induction on the number of contractions a vertex in the resulting DAG was involved in. Moreover, we use the fact that each vertex in the dependency graph has at most two outgoing edges (shown in Proposition 4.31).

Let $v$ be any vertex in the DAG that corresponds to a set of $k$ vertices $v_1, \ldots, v_k$ in the dependency graph. Our inductive hypothesis is that the vertices $v_1, \ldots, v_k$ are connected in a line in the communication graph, and that the outgoing edges of $v$

correspond to (at most) one outgoing edge of $v_1$ and (at most) one outgoing edge of $v_k$.

In the inductive step we consider a vertex $v$ in the resulting DAG which corresponds to a set of $k+1$ vertices in the dependency graph. Vertex $v$ was the result of the contraction of a vertex $u$ and a vertex $w$, where $u$ corresponds to a set $u_1, \ldots, u_\ell$ of vertices and $w$ corresponds to a set $w_1, \ldots, w_m$ where $\ell \geq 1, m \geq 1$ and $\ell + m = k+1$. From our inductive hypothesis the directed edges between the cycle that was contracted to create vertex $v$ must correspond to directed edges between $u_\ell$ and $w_1$, or $u_\ell$ and $w_m$, or $u_1$ and $w_1$, or $u_1$ and $w_m$. In any case, it follows that $v$ represents a set of vertices $v_1, \ldots, v_{k+1}$ which are connected in a line, and any outgoing edges of $v$ correspond to (at most) one outgoing edge of $v_1$ and (at most) one outgoing edge of $v_{k+1}$. □

#### 4.4.3.8 Putting all the pieces together

Finally we leverage our lower bound on the progress of a worst-case configuration, as well as the transformations described in subsection 4.4.3.3 through 4.4.3.7, to prove a lower bound on the progress of any configuration that satisfies the weak assumption.

**Theorem 4.33.** *Let $C$ be a configuration that satisfies the weak assumption and let $d = \min_{v \in V} \|\zeta_v - \varrho_v\|$. Then $C$ has a progress of at least $\Omega(\min(d, r))$.*

*Proof.* Theorem 4.32 implies that if $C$ is not a line graph then it contains a subset of robots which are connected in a line, and whose progress does not depend on any other robot. Since a lower bound on the progress of these subset of robots (which are connected in a line and satisfy the weak assumption) is also a lower bound on the progress of $C$, then without loss of generality we can assume that $C$ is a line configuration that satisfies the weak assumption.

Theorems 4.30, 4.29, 4.27 and 4.24 imply there exists a worst-case line configuration $C'$ whose progress is strictly smaller than the progress of $C$. Finally Theorem 4.17 implies the progress of a worst-case configuration is at least $d$, and the theorem follows. □

## 4.5 Multi-Round Executions

Section 4.3 and Section 4.4 analyzed, respectively, the safety and progress properties of CP-Alg during a single communication round. This section extends the analysis to consider executions that span multiple rounds. Since at any round safety is preserved unconditionally, it immediately follows that safety is also preserved unconditionally in any multi-round execution. Therefore, we study the progress guarantees for multi-round executions.

We focus our efforts on a particular class of executions. Before we define them formally we provide a motivating example. Suppose that every robot wants to reach its docking station to recharge its batteries. For a particular robot, given its physical speed and the distance to its docking station, it might be impossible for its motion planner to output a trajectory that gets the robot to the docking station in a single communication round. Instead the motion planner at each robot can output, at every round, a trajectory which gets the robot as close as possible to its long term goal in that particular round (taking into consideration the maximum physical speed of the robot).

For simplicity in this section we assume that in the duration of a single round, each robot can travel distance $s$. This amounts to assuming that all robots have identical actuators. Moreover we also assume that $s \leq r$. This assumption almost always holds in practice, since real robots can rarely outrun their own communication range in a single communication round.

Formally, we say a *long term execution* is one where: 1) Each robot $v \in V$ has a long term desired target position $T_v$. 2) At each round in the execution, if robot $v$ has reached $T_v$ then it outputs a length 0 trajectory, otherwise robot $v$ outputs the linear trajectory of length at most $s$ with its endpoint closest to $T_v$.

We remark that the progress assumptions for configurations introduced in Section 4.4 can be extended naturally to multi-round executions. Namely, an execution satisfies the weak assumption (resp. the robust assumption) if the configuration at each round in the execution satisfies the weak assumption (resp. the robust assumption).

For a long term execution we define $d_v(\ell) = \|\varrho_v(r) - T_v\|$ as the distance from the position of robot $v$ at round $\ell$ to its long term target position. We denote with $D(\ell) = \sum_{v \in V} d_v(\ell)$ the sum over all robots of their distance at round $\ell$ to their long term target positions. In other words, $D(0)$ denotes the total (over all robots) distance that the robots need to travel at the beginning of the execution in order for every robot to reach its desired long term target. The *duration of a long term execution* is the earliest round $\ell$ such that $d_v(\ell) = 0$ for every $v \in V$.

Our goal is to bound, in terms of $D(0)$, the duration of any long term execution that satisfies the robust or weak assumption. For this we leverage Theorem 4.12 and Theorem 4.16 respectively. Not surprisingly, proving a bound on the duration of a long term execution that satisfies the robust assumption is significantly easier than it is to prove a bound on the duration of a long term execution that satisfies the weak assumption.

The next theorem captures the informal notion that the number of rounds required for all robots to reach their target should be proportional to the total distance the robots need to travel and inversely proportional to the distance the robots can travel at every round.

**Theorem 4.34.** *The duration of a long term execution that satisfies the robust assumption is at most $O(D(0)/s)$.*

*Proof.* Since by assumption the execution satisfies the robust assumption, then by definition the configuration at every round in the execution satisfies the robust assumption. By definition of a long term execution, if at a round $\ell$ a robot $v$ is such that $d_v(\ell) \geq s$, then the target vector of robot $v$ at round $\ell$ is of length $s$. Furthermore, Theorem 4.12 implies that in a round where at least one robot has a target vector of length $s$ the system makes progress at least $s$ in that round.

It follows that if at round $\ell$ there exists a robot $v$ such that $d_v(\ell) \geq s$ then we have that $D(\ell+1) \leq D(\ell) - s$. Hence, after at most $k = \lfloor D(0)/s \rfloor$ rounds $D(k) < s$ and hence every robot $v$ is such that $d_v(k) < s$.

Therefore at round $k = \lfloor D(0)/s \rfloor$ all robots have their long term trajectory within reach (i.e. within distance $s$), and Theorem 4.12 implies that every robot reaches its long term target by round $k+1$, completing the proof. □

The reason why this theorem cannot be easily extended to long term executions that satisfy the weak assumption, is that once a single robot has reached its target position (i.e., when there exists a round $r$ and a robot $v$ such that $d_v(r) = 0$), then Theorem 4.16 gives us the empty guarantee that the progress is at least $d = \min_{v \in V} \|\gamma_v(0) - \gamma_v(1)\| = 0$. To sidestep this problem we generalize the notion of a long term execution to one where robots are content with getting within a ball of radius $\varepsilon$ of their long term target positions, instead of reaching the exact long term target position.

Formally, a *long term $\varepsilon$-close execution* is one where: 1) Each robot $v \in V$ has a long term desired target position $T_v$. 2) At each round in the execution, if robot $v$ is within distance $\varepsilon$ from $T_v$ then it outputs a length 0 trajectory, otherwise robot $v$ outputs the linear trajectory of length at most $s$ which gets it closest to $T_v$. The

*duration of a long term $\varepsilon$-close execution* is the earliest round number $r$ such that $d_v(r) \leq \varepsilon$ for every $v \in V$.

The proof of our next result has a similar flavor to that of Theorem 4.16. In more detail, first we leverage the line transformation to restrict our attention to line configurations. Next we leverage the bounded, balance and parallel-separated transformations to further restrict the space of line configurations we consider. The resulting configurations are very similar to the ones considered in Section 4.4.3.3, but the differences prevent us from applying the straight transformation directly. However, these configurations share the same characterization of the progress of each robot in terms of the angles from its target vector to its neighbors. This allows us to prove a lower bound on the progress by induction on the number of robots in the configuration.

**Lemma 4.35.** *Fix a long term $\varepsilon$-close execution and a round $k$ in the execution. If there is a robot $v$ such that $d_v(k) \geq \varepsilon$, then the progress of the system at round $k$ is $\Omega(\varepsilon)$.*

*Proof.* Fix a long term $\varepsilon$-close execution, a round $k$ in the execution, and a robot $v$ such that $d_v(k) \geq \varepsilon$. We partition the robots into two sets $P$ and $Q$, such that for all $u \in P$ we have $d_u(k) \geq \varepsilon$ and for all $u \in Q$ we have $d_u(k) < \varepsilon$. By assumption $P$ is non-empty and the robots in $Q$ remain stationary at round $k$.

Let $P' \subseteq P$ be a maximal connected subset of robots in $P$. Theorem 4.32 applied to the configuration induced by the robots in $P'$ implies there is a subset $P'' \subseteq P'$ of robots which are connected in a line and which do not depend on any other robot in $P$. However, it is possible that each of the robots at the end points of the line formed by the robots in $P''$ might depend on (at most) one robot outside $P$ (i.e., in $Q$). The case where none of the robots in $P''$ depend on a robot outside $P$ is already handled by Theorem 4.16. In the rest of our progress argument we assume that both of the end point robots in $P''$ depend on a robot on $Q$. In other words we consider a line configuration where the robots at the end points are in $Q$ (and have a target vector of length 0) and the inner robots are in $P$ (and have a target vector of length at least $\varepsilon$). The case where only one of the end point robots is in $Q$ is analogous.

Since the inner robots are connected in a line and have a non-zero target vector we can apply the bounded, balance and parallel-separate transformations to assume, without loss of generality, that the inner robots are $\varepsilon$-bounded, balanced, parallel and separated. Furthermore, since the end point robots have zero target vectors, we can leverage Lemma 4.22 to assume, without loss of generality, that they are at distance exactly $r$ from their neighbors' position (i.e., separated), and at distance exactly $r$ from their neighbors' target position. Therefore, we consider a line configuration

where all robots are balanced and separated, the inner robots are $\varepsilon$-bounded and parallel, and the end point robots have a zero length target vector and are at distance exactly $r$ from their neighbors' target position.

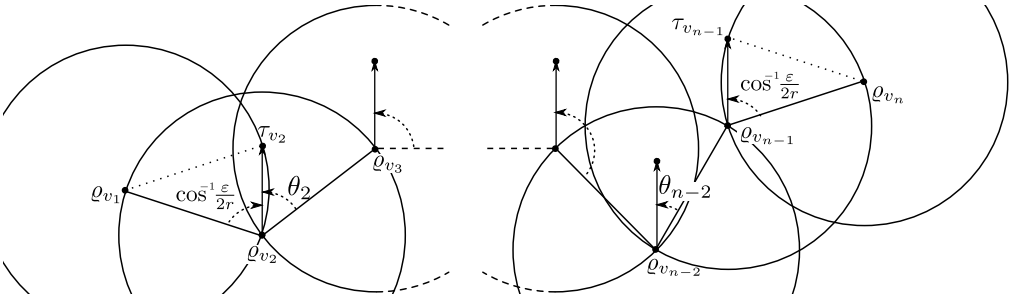



Figure 4-13: A line configuration of $n$ robots where all robots are balanced and separated, the inner robots are $\varepsilon$-bounded and parallel, and the end point robots have a zero target vector and are at distance $r$ from their neighbors' target position.

We highlight that this situation is very similar to the one considered in the straight transformation. Namely, the progress of such a line configuration with $n$ robots is a function of $\varepsilon$ and a series of $n-1$ angles $\theta_1, \ldots, \theta_{n-1}$. Furthermore, since the end point robots are at distance exactly $r$ from their neighbors' target position (in addition to being separated) then the angles $\theta_1$ and $\theta_{n-1}$ are fixed (see Figure 4-13).

Despite these similarities, we cannot apply the straight transformation directly. Instead, in the next claim, we prove a lower bound of $\Omega(\varepsilon)$ on the progress of the configuration by an induction on the number of robots.

*Claim.* Fix a line configuration with $n \geq 3$ robots that satisfies the weak assumption and is balanced, separated, the inner robots are $\varepsilon$-bounded and parallel, and the two end point robots have a target vector of length zero and are at distance exactly $r$ from their neighbors' target position. The progress of this configuration is $\geq \varepsilon/3$.

Base Case: If $n = 3$ then there is a single inner robot, and by definition its target vector is contained in the ball of radius $r$ around both of its neighbors. This implies the inner robot is unrestricted by its neighbors and its progress is exactly $\varepsilon$.

Inductive Step: If $n \geq 4$ then the progress of the configuration can be written as $\chi + \Gamma(\alpha, \beta) + \Gamma(\pi - \beta, \arccos\frac{\varepsilon}{2r})$ where $\chi \geq 0$ represents the progress of the first $n-4$ inner robots and $\Gamma(\alpha, \beta) + \Gamma(\pi - \beta, \arccos\frac{\varepsilon}{2r})$ represents the progress of the last two inner robots. In the previous expression $\alpha$ and $\beta$ represent the angles from the target vector of the one before last inner robot to its neighbors, and $\arccos\frac{\varepsilon}{2r}$ is the angle from the last inner robot to the end point robot (this angle is fixed since these two robots are separated and the distance from the end point robot to its neighbors' target is exactly $r$). The rest of the proof follows by some algebraic manipulation.

If $\alpha \leq \frac{\pi}{2}$ then the minimum of $\Gamma(\alpha, \beta) + \Gamma(\pi - \beta, \arccos \frac{\varepsilon}{2r})$ is at $\alpha = \beta = \frac{\pi}{2}$ which is greater than $\varepsilon/3$ and completes our inductive step.

If $\alpha > \frac{\pi}{2}$ then we leverage the inductive hypothesis for a configuration with $n-1$ robots. In particular by inductive hypothesis we have that $\chi + \Gamma(\alpha, \arccos \frac{\varepsilon}{2r}) \geq \varepsilon/3$. Therefore we need only to show that $\Gamma(\alpha, \beta) + \Gamma(\pi - \beta, \arccos \frac{\varepsilon}{2r}) - \Gamma(\alpha, \frac{\varepsilon}{2r}) \geq 0$, which can be confirmed by finding the minimum of this function in the region $\alpha > \frac{\pi}{2}$. □

With this lemma in place, we are ready to prove our main theorem for long term $\varepsilon$-close executions that satisfy the weak assumption.

**Theorem 4.36.** *The duration of a long term $\varepsilon$-close execution that satisfies the weak assumption is at most $O(D(0)/s + n^2 r/\varepsilon)$.*

*Proof.* Since by assumption the execution satisfies the weak assumption, then by definition the configuration at every round in the execution satisfies the weak assumption. By definition of a long term $\varepsilon$-close execution, if at a round $\ell$ a robot $v$ is such that $d_v(\ell) \geq s \geq \varepsilon$, then the target vector of robot $v$ at round $\ell$ is of length $s$. Furthermore, Theorem 4.16 implies that in a round where all robots have a target vector of length $s$ (i.e., an $s$-bounded configuration) the system makes progress at least $s$ in that round.

It follows that if at round $\ell$ every robot $v \in V$ is such that $d_v(\ell) \geq s$ then we have that $D(\ell + 1) \leq D(\ell) - s$. Hence after at most $k = \lfloor D(0)/s \rfloor$ rounds there is (at least) one robot $v$ such that $d_v(k) \leq s$.

The weak assumption requires the trajectories of adjacent robots to be weakly connected at every round. This implies that the difference in the length of the target vectors of two adjacent robots is at most $2r$. Together with the fact that at round $k$ there is a robot $v$ such that $d_v(k) \leq s \leq r$, this implies that $D(k) \leq r + (r + 2r) + (r + 2r + 2r) + \ldots = 2r \cdot \sum_{i=1}^{n} i - nr = rn^2$. For any round $\ell > k$ if there exists a robot $v$ such that $d_v(\ell) \geq \varepsilon$ then Lemma 4.35 implies the progress at round $\ell$ is $\Omega(\varepsilon)$. Otherwise the execution has terminated (since for every robot $v$ we have $d_v(\ell) \leq \varepsilon$). This shows the duration of the execution is $O(D(0)/s + n^2 r/\varepsilon)$, which completes the proof. □

# Chapter 5

# Preserving a $k$-Connected Graph

Chapters 3 and 4 tackled the problem of preserving the connectivity of the communication graph while allowing each robot to get closer to its desired target position. This chapter describes how to generalize this approach to preserve a $k$-connected graph.

Paraphrasing the formal definition given in Section 2.1, the connectivity of a graph $G$, denoted $\kappa_G$, is the size of the smallest set of vertices whose removal disconnects the graph. A complete graph on $n$ vertices cannot be disconnected by removing vertices, but by convention its connectivity is $n-1$. A graph $G$ is $k$-connected if $\kappa_G \geq k$. As we argued before, the connectivity of a graph is a good estimate of the fault-tolerance of the communication network, since higher connectivity means more robots can fail without disrupting the communication among the rest of the robots.

**Roadmap.** As we did for the problem of preserving a simply (i.e., $k = 1$) connected communication graph, the problem we consider can be subdivided into two parts. In the first part, the goal is to find, for each robot, a set of neighbors such that preserving connectivity to these neighbors is sufficient for the communication graph to remain $k$-connected. The fewer neighbors a robot has to preserve, the greater freedom it has to compute a trajectory that remains connected to these neighbors. Moreover, remaining connected to close-by robots is less of a restriction than remaining connected to robots that are farther away.

The second part of the problem is concerned with agreeing on a set of linear trajectories (one for each robot), so as to maximize the progress each robot makes with respect to its original trajectory (controlled by the motion planner). The only constraint when finding such trajectories is that each robot should remain robustly-connected to each of the neighbors which were identified as being sufficient for pre-

serving $k$-connectivity in the first part.

For $k = 1$ these problems were tackled in Chapters 3 and 4 respectively. Instead of solving these two problems for general $k$ from scratch, this chapter describes how to leverage the solutions derived in Chapters 3 and 4 for $k = 1$ and apply them for any $k \geq 1$. To this end we explore the relationship between preserving the connectivity of a communication graph using "small" edges, and preserving the $k$-connectivity of a graph using "normal" edges.

## 5.1 From 1-Connected to $k$-Connected

A graph with $n$ vertices has, by definition, connectivity of at most $n - 1$. Since we are concerned with guaranteeing that a graph remains $k$-connected, all graphs we consider will have at least $n \geq k + 1$ vertices unless specifically stated.

The main technical result in this section is a theorem that says that if a connected spanning subgraph of a unit disk graph uses only "small" edges then the unit disk graph has "high" connectivity. This is captured by the next theorem.

**Theorem 5.1.** *Let $G$ be a unit disk graph of radius $r$ of $n \geq k+1$ points. If $G$ has a connected spanning subgraph using edges of length at most $\frac{r}{k}$ then $G$ is $k$-connected.*

*Proof.* Let $G$ be a unit disk graph of radius $r$ of $n \geq k+1$ points that has a connected spanning subgraph $H$ which uses edges of length at most $\frac{r}{k}$. Recall from section 2.1 that a vertex cut is a set of vertices whose removal disconnects the graph.

If $G$ *does not* have a vertex cut of size $k - 1$ or smaller, then $G$ is $k$-connected and the theorem follows. Therefore, let us assume by way of contradiction that $G$ has a vertex cut $C$ of size $|C| \leq k - 1$. Let $P$ and $Q$ be two connected components produced by removing the vertices of the cut $C$ from $G$ (i.e., the cut $C$ separates $P$ and $Q$ in $G$). Fix any vertex $p \in P$ and any vertex $q \in Q$. Since $H$ is a connected spanning subgraph of $G$ there must exists a path $H_{pq} \subseteq H$ that connects $p$ and $q$ using edges of length at most $\frac{r}{k}$.

We define a *gap* in the path $H_{pq}$ as a maximal set of vertices of $C$ which are contiguous in the path $H_{pq}$. For a gap $g$ in $H_{pq}$ we use $pred(g)$ and $succ(g)$ to denote the vertices in the path $H_{pq}$ immediately before and after the gap. The size of a gap $g$, denoted $|g|$, is the number of vertices it contains. The length of a gap $g$, denoted $length(g)$, is the Euclidean distance between the vertices $pred(g)$ and $succ(g)$. Since the path $H_{pq}$ uses only edges of length at most $r/k$ then for any gap $g$ in $H_{pq}$ we have $length(g) \leq (|g|+1)r/k$. Moreover for any gap $g$ in $H_{pq}$ we have that $|g| \leq |C| \leq k-1$ by definition, and therefore $length(g) \leq (|C| + 1|)r/k \leq (k)r/k = r$.

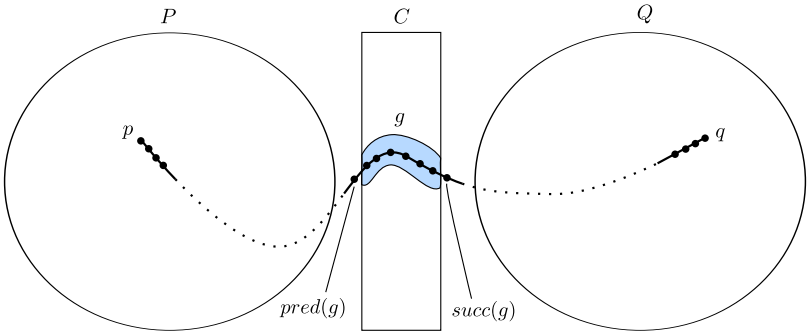



Figure 5-1: A path from $p \in P$ to $q \in Q$ with a single gap $g$ of 6 vertices.

Since by assumption $G$ is a unit disk graph of radius $r$ the above implies that for any gap $g$ in $H_{pq}$ the graph $G$ contains an edge between $pred(g)$ and $succ(g)$. Therefore $G$ contains a path from $p$ to $q$ without using vertices of $C$, which contradicts the assumption that $C$ is a vertex cut of size $|C| \leq k-1$ that separates $P$ and $Q$. □

Let $UDG_r(P)$ denote the unit disk graph of radius $r$ on a point set $P$. Theorem 5.1 can be rephrased to yield the following corollary.

**Corollary 5.2.** *For any point set $P$ where $|P| \geq k+1$, if $UDG_{r/k}(P)$ is connected then $UDG_r(P)$ is $k$-connected.*

The next section describes how we can leverage this corollary to design an algorithm that preserves a $k$-connected graph.

## 5.2 Preserving a $k$-Connected Graph

For shorthand notation, we say a graph is $(r,k)$-connected if it has a connected subgraph using edges of length at most $r/k$. Corollary 5.2 implies that if the communication graph is a $k$-connected unit disk graph, then to preserve $k$-connectivity it suffices to preserve a $(r,k)$-connected graph.

Naturally, to preserve a $(r,k)$-connected graph, the graph must initially be $(r,k)$-connected. We highlight that the requirement that the graph is initially $(r,k)$-connected is slightly stronger than requiring that it is $k$-connected. This follows from the fact that Corollary 5.2 implies that an $(r,k)$-connected graph is $k$-connected, but the converse is not always true. For instance, the unit disk graph of radius $r$ that results from three nodes arranged in an equilateral triangle with side length $r$ is a complete graph of 3 vertices. This graph is 2-connected by definition, but since it does not have a single edge of length $r/2$ it is not $(r,2)$-connected.

Theorem 4.10 showed that when CP-Alg received as a parameter the communication radius $r$ it robustly preserves a connected subgraph with edges of length at most $r$. Therefore we immediately have as a corollary that if CP-Alg is used with a communication radius of $r/k$ then it robustly preserves a connected subgraph with edges of length at most $r/k$. This is captured by the following theorem.

**Theorem 5.3.** *If the communication graph is initially $(r, k)$-connected, then running* CP-Alg *with a communication radius of $r/k$ robustly preserves an $(r, k)$-connected subgraph.*

Corollary 5.2 implies that an $(r, k)$-connected graph is always $k$-connected. This, together with the previous theorem yields the following corollary.

**Corollary 5.4.** *If the communication graph is initially $(r, k)$-connected, then running* CP-Alg *with a communication radius of $r/k$ robustly preserves $k$-connectivity.*

The robust and weak progress theorems 4.12 and 4.16 apply unchanged to this setting. Therefore the cost of preserving $k$-connectivity (as opposed to preserving simple connectivity) is that the robust and weak assumptions need to be satisfied with the more stringent communication radius of $r/k$.

# Chapter 6

# Applications of Connectivity

The purpose of this chapter is to argue that the connectivity-preserving algorithm can be used to facilitate the design of higher level behavior for multi-robot swarms. Concretely we consider the problem of flocking. Informally, flocking describes an emergent behavior of a collection of agents with no central coordination that move cohesively despite having no common a priori sense of direction. This chapter describes how to combine a connectivity-preserving algorithm with standard agreement procedures [8, 46] to implement a provably correct flocking behavior.

## 6.1 What is flocking?

The word flocking is typically used to describe the behavior exhibited by a group of birds (i.e., a flock) during flight. A similar behavior is observed in schools of fish, swarms of insects, herds of hoofed animals (i.e., zebra, sheep, etc.), colonies of bacteria, etc. In computer science the word flocking is used to refer to the collective motion of a large number of collaborating but independent entities. Flocking is an often cited example of emergent behavior that does not require central coordination and arises from a collection simple rules followed by individuals. Flocking has several applications to multi-robot systems, and one of the most often cited applications [7] is to control the behavior of Unmanned Aerial Vehicles (UAVs).

The Boids computer program of Reynolds [79] in 1986 is the first example of simulated flocking. The Boids program simulates simple agents that move according to a set of basic local rules. Visually the result is akin to the behavior of a flock of birds in flight.

One of the first works to study flocking behavior from a biological perspective, is that of Partridge [73] in 1982. Patridge studied the coordination of fish schools,

in particular the response of fish schools to predators. Vicsek et al. [92] studied flocking from a physics perspective through simulations. The work of Vicsek et al. focused on the emergence of alignment in self-driven particle systems. Flocking has also been studied from a control theoretic perspective, for example in the work of Olfati-Saber [71] and Jadbabaie et al. [46], where the emphasis is on the robustness of the eventual alignment process despite the local and unpredictable nature of the communication.

Despite numerous works devoted to studying flocking, different research groups have used different definitions for flocking, and there does not exist a unique formal definition of what constitutes flocking behavior. Nevertheless, most (if not all) works agree that in order for the behavior of a collection of entities to be considered flocking it must satisfy certain *connectivity* and *alignment* properties. Informally, the connectivity property ensures the group remains cohesive and the agents do not become dispersed in the environment and the alignment property ensures that the flock moves in the same general direction. There is an inherent synergy between these two properties: without agreement on the direction of motion it is hard for the group to remain cohesive, and without cohesion it is hard for the group to agree on a direction of motion. Therefore, the flocking behavior embodies the challenges that our connectivity preserving algorithm was designed to overcome.

Formally, we say a configuration is *connected* if its communication graph is connected, and we say it is *aligned* if every robot has the same target vector (i.e. they are moving in the same direction and with the same speed). A configuration is *flocking* if it is both connected and aligned. The goal of this chapter is to describe a local distributed algorithm that allows a multi-robot system to converge to a flocking configuration.

**Roadmap.** Section 6.2 presents a brief summary of various agreement procedures available in the literature. It also outlines how alignment can be achieved by carrying out an agreement procedure on the components of the target vectors. Section 6.3 describes how to combine the connectivity preserving algorithm of Chapter 4 with the agreement procedures of Section 6.2 to design a provably correct flocking algorithm.

## 6.2 Alignment and Agreement

In the 1980s Bertsekas and Tsitsiklis [8, 91] studied various models of "distributed asynchronous iterations". They were motivated mainly by distributed signal processing, distributed optimization and parallel computation. An important part of this

work was an agreement algorithm whereby a set of agents converge to the same value, by performing a series of weighted averages of their own value and their neighbors (possibly outdated) values.

In this respect, their main result was to show that simple repeated averaging procedures (proposed earlier by De Groot [25]) allow agents to converge to the same common value under very mild assumptions on the communication delays and the structure of the time-varying neighbor relations. In particular, they considered directed (as opposed to undirected) communication between neighbors, and a semi-asynchronous model of computation with unknown but bounded time delays. This section presents only a special case of their results for a synchronous round-based model of computation and an undirected model of communication; we refer the interested reader to [8] for the more general version of the results.

The agreement algorithm considered by Bertsekas and Tsitsiklis has every node compute the weighted arithmetic mean of the scalar values of its neighbors. The weighting coefficients used by each node should be convex (in other words, they should add up to one) and they should also be positive and bounded away from zero.

Recall that $G_t = (V, E_t)$ denotes the communication graph at time $t$. Let $N_t[u] = \{u\} \cup \{v \mid \{u, v\} \in E_t\}$ represent the set of closed neighbors of robot $u$ at time $t$. Suppose each robot $u$ starts with a scalar value $x_0(u)$. The precise agreement algorithm is described in Algorithm 7.

**Algorithm 7** AgreementUpdate at robot $u$

$$x_{t+1}(u) = \sum_{w \in N_t[u]} \alpha_t(w, u) x_t(w) \tag{6.1}$$

The weight coefficients $\alpha_t(w, u)$ should satisfy the following for some $\alpha_0 > 0$:

$$\forall t \in \mathbb{N}, \forall v \in V, \forall w \in N_t[v] \qquad \alpha_t(w, v) \geq \alpha_0 \tag{6.2}$$

$$\forall t \in \mathbb{N}, \forall v \in V \qquad \sum_{w \in N_t[v]} \alpha_t(w, v) = 1 \tag{6.3}$$

The following theorem is a special case of the main theorem in Bertsekas and Tsitsiklis [8, Section 7.3].

**Theorem 6.1.** *If at every time $t$ the graph $G_t$ is connected, then there is a set of non-negative coefficients $\{\phi(w)\}_{w \in V}$ (that depend only on the sequence of graphs)*

*such that*

$$\lim_{t\to\infty} x_t(v) = \sum_{w\in V} \phi(w)x_0(w) \quad \forall v \in V,$$

*and convergence takes place at the rate of a geometric progression.*

In other words, as long as the communication graph remains connected and every robot updates its own value using a convex combination of the values of its neighbors, eventually every robot will converge to the same value. The proof of this theorem relies on a result on the convergence of products of stochastic matrices from Markov chain theory.

**From Scalar Agreement to Vector Agreement.** The target vector of each robot can be decomposed into two scalars, for example the vector's magnitude and direction. Each robot can then update its target vector using the agreement algorithm on each of the scalar components of its target vector. Provided the conditions of Theorem 6.1 are met, this would result in every robot converging to the same target vector (i.e., achieving alignment). This is summarized by the next corollary.

**Corollary 6.2.** *If at every time $t$ the graph $G_t$ is connected, and at every round each robot updates its target vector to match the arithmetic mean of the magnitude and direction of its neighbors' target vector, then all robots converge to the same target vector.*

The connection between these simple averaging procedures and the emergent alignment property exhibited during flocking was first studied through simulations by Vicsek et al. [92]. Jadbabaie et al. [46] later provided a rigorous analysis showing the convergence of the averaging procedures used in Vicsek's work under certain connectivity assumptions. The main convergence result of [46] can be interpreted as an instance of Theorem 6.1 for the case when $\alpha_t(u, v) = 1/\left|N_t[v]\right|$ for $\forall t \in \mathbb{N},\ \forall v \in V,\ \forall u \in N_t[u]$. However, as observed in [9], more general convergence results had appeared earlier in the work of Bertsekas and Tsitsiklis [8, 91], although not in the context of flocking.

However, as noted in the work on Olfati-Saber [71], on their own, averaging update rules do not enforce group cohesion and lead to flocking behavior only for a "very restricted set of initial states". In particular, [71] showed through simulations that in configurations of 10 or more robots with a set of random initial states, relying solely on an averaging rule to achieve alignment most likely leads to a fragmentation and therefore does not achieve flocking. This fragmentation problem was also observed,

although not dealt with, by the earlier work of Vicsek et al. [92] and Jadbabaie et al. [46].

To alleviate the fragmentation problem and achieve successful flocking, Olfati-Saber proposed an approach that relied on using a "$\gamma$-agent" to provide a moving rendezvous point that prevents fragmentation. This approach is non-local since it requires all robots to follow the global trajectories of this "$\gamma$-agent". In contrast, the next section describes how to achieve flocking behavior by combining the aforementioned averaging procedure with the connectivity-preserving algorithm of Chapter 4 to prevent the flock from dispersing, all this using only local interactions.

## 6.3 Flocking Algorithm

This subsection describes how to combine the agreement algorithm described in Section 6.2 with the connectivity-preserving algorithm of Chapter 4 to produce an algorithm that steers the multi-robot system to a flocking configuration. In particular we describe an "alignment motion planning module" and we argue that, when combined with the connectivity-preserving algorithm of Chapter 4, it guarantees that the system will converge to a flocking configuration.

The only state kept by the alignment motion planner module is the current direction and magnitude of the target vector. Initially these are set to arbitrary values. At each round, the motion planner uses one step of the agreement algorithm described in Section 6.2 to update the direction and magnitude of the target vectors. The psuedo-code appears in Algorithm 8.

**Algorithm 8** AlignmentMotionPlanner at robot $u$

▷ Initial state

$dir_u \leftarrow$ arbitrary(0,2$\pi$), $mag_u \leftarrow$ arbitrary(0,1)

▷ At each round

**bcast** $\langle dir_u, mag_u \rangle$
$D \leftarrow \{dir_u\}, M \leftarrow \{mag_u\}$
**add** every $\langle dir_w, mag_w \rangle$ **recv'd** to $D$ and $M$
$dir_u \leftarrow \text{mean}(D)$, $mag_u \leftarrow \text{mean}(M)$

**return** linear trajectory $\gamma_u$ from $\varrho_u$ with direction $dir_u$ and magnitude $mag_u$

We refer to the composition of the alignment motion planner module and the connectivity-preserving algorithm as the *flocking algorithm*. At each round the flocking algorithm feeds the trajectory output by the alignment motion planner to the

connectivity-preserving algorithm. This allows the robots to try to move closer to the goal dictated by the alignment motion planner, while guaranteeing that the communication graph of the configuration remains connected. This prevents the fragmentation observed in the algorithm considered by by Vicsek et al. [92] and Jadbabaie et al. [46], which instead feeds the trajectory output by the alignment motion planner to the robot's actuators.

We refer the reader back to Figure 4-1 for a diagram of the system components and the interactions between the motion planner, the connectivity-preserving algorithm and the actuators. We highlight that the modified trajectories produced by the connectivity-preserving algorithm are fed directly to the actuators and have no effect on the state variables of the motion planner module. Specifically, the direction and magnitude of the target vector computed by the alignment motion planning module at the end of round $r-1$ is used as input to the arithmetic mean used to update these values at round $r$. To prevent the communication graph from becoming disconnected, we do not use the output of the alignment motion planning module to control the motion of the robots directly, and instead we feed this to the connectivity-preserving algorithm. In particular at round $r$ the connectivity-preserving algorithm uses as its input the target vector defined by the direction and magnitude values computed by the motion planner module at round $r$. The (possibly) modified target vector output by the connectivity-preserving algorithm is used to control the resulting motion of the robots. Since this (possibly) modified vector *has no effect* on the computations carried out by the alignment motion planner, we can leverage Corollary 6.2 to guarantee the convergence of the target vectors as computed by the alignment motion planning module.

Algorithm 9 describes in pseudo-code the high-level interactions between the different components of the system at each round.

**Algorithm 9** FlockAlgorithm: Component Interactions at robot $u$ in a round

$\varrho_u \leftarrow$position-sensors()
$\gamma_u \leftarrow$alignment-motion-planner($\varrho_u$)
$\gamma'_u \leftarrow$CP-Alg $(r, \varrho_u, \gamma_u)$
updatePosition($\gamma'_u$)

We are ready to prove the main theorem of this chapter.

**Theorem 6.3.** *Starting from any connected configuration, the flocking algorithm guarantees the system will converge to a flocking configuration.*

*Proof.* From Theorem 4.10 it follows that the connectivity-preserving algorithm will guarantee the configuration remains connected, regardless of what target vectors

are used by the robots. In other words the configuration is always connected. Theorem 6.1 implies that since the configuration is always connected the alignment motion planning module will converge to the same target vector (same magnitude and same direction) for every robot. In other words, the alignment motion planning module guarantees the system converges to an aligned configuration. Since by definition a configuration is flocking if it is connected and aligned, the theorem follows.

□

By construction if all target vectors are equal at some round $r$, the alignment motion planning module will not modify the target at any round $r' \geq r$. This, together with the fact that the connectivity-preserving algorithm unconditionally guarantees the communication graph remains connected, implies that if the system reaches a flocking configuration it will remain in a flocking configuration thereafter. This is captured by the next corollary.

**Corollary 6.4.** *Starting from a flocking configuration, the flocking algorithm guarantees the system will remain in a flocking configuration.*

We remark that since the target vectors of the robots are initially arbitrary, the connectivity-preserving algorithm cannot guarantee progress. Nevertheless, once the system has reached a flocking configuration, we can translate some of the progress results of Section 4.4 to a flocking configuration. Specifically, we can prove the following theorem.

**Theorem 6.5.** *Let $C$ be a flocking configuration with no cyclic dependencies, and let $d$ be the length of the target vectors of $C$. The progress of $C$ is at least $\Omega(\min(d, r))$.*

*Proof.* Since $C$ is flocking, then by definition it is aligned and therefore all robots have the same target vectors (of length $d$). Therefore by definition it follows that all neighboring robots have weakly-connected trajectories. Moreover since $C$ by assumption has no cyclic dependencies, then Definition 4.15 implies $C$ satisfies the weak assumption. Finally from Theorem 4.16 it follows that the progress of $C$ is at least $\Omega(\min(d, r))$. □

# Part II

# Localization

Large populations of robots can solve many challenging problems such as mapping, exploration, search-and-rescue, and surveillance. All these applications require robots to have at least some information about the *network geometry*: the positions and orientations of other robots relative to their own. Different approaches to computing network geometry have trade-offs between the amount of information provided, the complexity and cost of the sensors required, and the amount of communications used. For instance, a GPS system provides each robot with a global position, which can be used to easily derive complete network geometry information. However, GPS is not available in many environments: under foliage, indoors, underwater, or on other planets. Vision- and SLAM*-based approaches to build a global map and extract a shared global reference frame do not suffer from the environmental restrictions of GPS. However, the cost and complexity of these solutions make them unsuitable for large populations of simple robots. Ideally, a localization system for large populations of simple robots must rely on simple and low cost sensors and allow the robots to operate in as many environments as possible.

The second part of this thesis is devoted to studying coordinate systems that can be obtained from simple angle measuring sensors. Angle measuring sensors are appropriate for low-cost simple robots, and are readily available in multi-robot systems intended for large populations [61]. We use the term *scale-free coordinates* to describe the maximum amount of network geometry which can be recovered when using only angle measuring sensors. We propose a distributed algorithm that can be used to compute the scale-free coordinates of any subset of robots using the minimum number of communication rounds possible. To do so, we first develop a precise mathematical characterization of the computability of scale-free coordinates using only angle measuring sensors.

**Problem Formulation.** We consider a simple sensing model in which each robot can only measure the angle, relative to its own orientation, to neighboring robots in the communication graph. We highlight that, in contrast to Part I, we assume no relationship between the distance between two robots and the presence of an edge in the communication graph. We require only that the communication graph is connected (or alternatively, we consider the problem only for robots in the same connected component). For simplicity, we assume that when a robot $u$ receives a message from a robot $v$, it records the identifier $id(v)$ of the sender and measures the angle $\theta(u, v)$, with respect to $u$'s orientation, from which the message originated. In

*SLAM is an acronym for simultaneous localization and mapping. It is a widely used [36] technique in robotics that allows robots to build a map within an unknown environment while simultaneously keeping track of their current location in that environment.

practice the mechanism for measuring angles between adjacent robots needs not be coupled with message reception, but this abstraction simplifies our model.

The goal of each robot is to compute the relative orientation and the relative position up to scale (i.e., scale-free coordinates) of some specific subset of robots in the system. To accomplish this, robots will collaborate and leverage the angle measurements available in the network. Informally, scale-free coordinates provide complete network geometry information up to a fixed but unknown scaling factor. In other words, each robot can recover the "shape" of the network, but not its scale. Formally, the scale-free coordinates of a set of robots $S$ is described by a set of tuples $\{(i, x_i, y_i, \theta_i) \mid i \in S\}$, where each tuple represents the relative orientation and the relative position (up to some unknown positive scaling constant) of a robot in $S$. Of particular interest to us are the *local scale-free coordinates* of a robot, which are the scale-free coordinates of itself and its neighbors.

**Outline.** We tackle the problem of computing the relative orientations and computing the relative positions (up to scale) separately.

Chapter 7 describes a centralized procedure that allows each robot to compute the relative orientation of every other robot. We use this procedure to translate the angle measurements of any connected subset of robots to the same reference orientation. These "translated" angle measurements can then be used as an angle-constraint on the graph.

Chapter 8 provides a precise mathematical characterization of the communication graphs and angle-constraints in which it is possible to compute the relative positions (up to scale) of every robot in the system. Later we generalize this characterization to the case of computing the relative positions of a specific subset of robots in the system (even if it is not possible to compute them for all robots).

Chapter 9 leverages the previous two results to design a distributed protocol, with an optimal round complexity, which efficiently computes the scale-free coordinates of any subset of robots in the system. Finally Chapter 10 describes how to perform a variety of multi-robot tasks using scale-free coordinates.

# Chapter 7

# Relative Orientations

This chapter tackles the problem of computing the relative orientations of all robots in the system. Informally, the relative orientation between two robots is simply the difference between their orientations. (Recall from Section 2.3 that the orientation of a robot is simply the counter-clockwise angle between the $x$-axis of the global coordinate system and the robots' heading.) A robot does not know its own orientation, but we will show that this does not prevent it from computing the relative orientation of other robots using the angle measurements in the communication graph.

**Roadmap.** Section 7.1 defines the concept of an angle-constrained graph. Briefly, an angle-constraint on a graph is simply a collection of angle measurements on the graph which are taken with respect to the same orientation. The additional structure present in angle-constraints makes them easier to work with than the "raw" angle measurements.

Section 7.2 describes, and proves the correctness of, a simple procedure that allows us to compute, with respect to a fixed robot, the relative orientations of every other robot in the system. At the same time, this procedure allows us to transform a collection of angle measurements on a graph to an angle-constraint on the same graph.

These results allow us to focus in Chapter 8 on the problem of finding the relative positions (up to scale) of the robots given an angle-constrained graph.

## 7.1 Defining Angle-Constraints

To simplify some definitions it will be useful to consider for each undirected edge in the communication graph its two directed counterparts. Specifically let $\overleftrightarrow{E}_G = \{(u,v) \mid \{u,v\} \in E_G\}$ denote the directed edges present in the undirected graph $G$.

Recall from Chapter 2 that $\phi_v$ represents the orientation of robot $v$, defined as the counter-clockwise angle from the $x$-axis of the global coordinate system to $v$'s heading. We use this to define the relative orientation between two robots.

**Definition 7.1.** *The* relative orientation *of robot $v$ with respect to robot $u$ is defined as* $h_{uv} = \phi_v - \phi_u \pmod{2\pi}$.

Next we define formally what is the angle measurement between two robots.

**Definition 7.2.** *The* angle measurement *from $u$ to $v$, denoted by $\theta(u,v)$, is the counter-clockwise angle between the heading vector of $u$ and the vector from $u$ to $v$ (see Figure 7-1).*

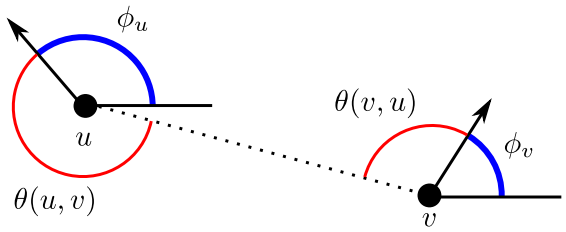



Figure 7-1: Each robot is represented by a dot denoting its position and an arrow denoting its heading. Thin (black) dotted lines connect neighboring robots, thick (blue) arcs represent the orientations of the robots and thin (red) arcs represent the angle measurements between neighboring robots.

We let $\Theta_G : \overleftrightarrow{E}_G \to [0, 2\pi)$ be a function that associates an angle measurement to each directed edge in $G$, so that $\Theta_G(v,w) = \theta(v,w)$ for every edge $(v,w) \in \overleftrightarrow{E}_G$.

For a pair of robots $u$ and $v$ the following proposition relates the angle measurements $\theta(u,v)$ and $\theta(v,u)$.

**Proposition 7.3.** $\theta(u,v) + \phi_u = \theta(v,u) + \phi_v + \pi \pmod{2\pi}$.

*Proof.* The term $\theta(u,v) + \phi_u$ represents the counter-clockwise angle from the $x$-axis of the global coordinate system to the vector from $u$ to $v$ while the term $\theta(v,u) + \phi_v$ represents the counter-clockwise angle from the $x$-axis of the global coordinate system

to the vector from $v$ to $u$. The difference of $\pi$ between these two terms accounts the fact that the vector from $u$ to $v$ and the vector from $v$ to $u$ have opposite directions. □

By looking at Proposition 7.3 more closely we can observe that it allows us to express the relative orientation between robot $u$ and $v$ in terms of the angle measurements from $u$ to $v$ and from $v$ to $u$.

As a warm up, first observe that if $\theta(u, v) = \theta(v, u) = 0$ then the robots $u$ and $v$ are "pointing" at each other (see Figure 7-2 left). Generalizing this slightly, it is not hard to see that as long as $\theta(u, v) = \theta(v, u)$ then the heading of robot $u$ must the opposite of the heading of robot $v$ (see Figure 7-2 middle and right). In other words if $\theta(u, v) = \theta(v, u)$ then it follows that $h_{uv} = \pi$.

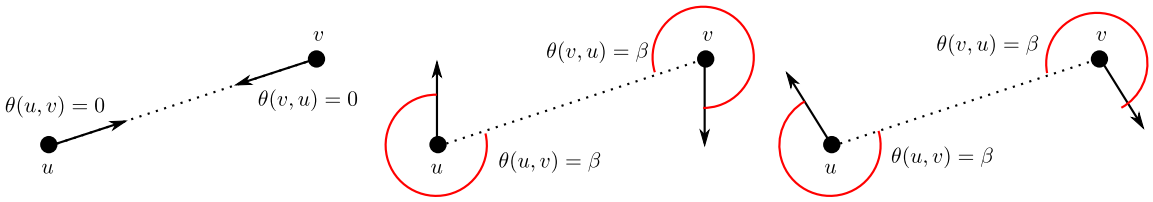



Figure 7-2: Robots are represented by a dot denoting their position and an arrow denoting their orientation. A thin dotted line connects robot $u$ and $v$, and the angle measurements $\theta(u, v)$ and $\theta(v, u)$ are denoted with dashed lines. The figure depicts three cases when $\theta(u, v) = \theta(v, u)$.

In general, the relationship between $h_{uv}$ and $\theta(u, v)$ and $\theta(v, u)$ is captured by the next proposition.

**Proposition 7.4.** $h_{uv} = \theta(u, v) - \theta(v, u) - \pi \pmod{2\pi}$.

This proposition follows immediately from Proposition 7.3 and Definition 7.1 and requires no proof.

By definition, the angle measurement $\theta(v, w)$ from robot $v$ to robot $w$ is with respect to the orientation of robot $v$. Informally speaking, if a third robot $u$ wanted to interpret the angle measurement $\theta(v, w)$ we would expect robot $u$ would need to "translate" this measurement to its own orientation. This informal notion of "translating" angle measurements is captured formally by the next definition.

**Definition 7.5.** *An* angle measurement with respect to $u$ *from* $v$ *to* $w$ *is defined as* $\theta^u(v, w) = \theta(v, w) + h_{uv} \pmod{2\pi}$.

We let $\Theta^u_G : \overleftrightarrow{E}_G \to [0, 2\pi)$ be a function that associates an angle measurement to each directed edge in $G$, so that $\Theta^u_G(v, w) = \theta^u(v, w)$ for every edge $(v, w) \in \overleftrightarrow{E}_G$.

For a pair of robots $v$ and $w$ the following proposition relates the angle measurements $\theta^u(v, w)$ and $\theta^u(w, v)$ with respect to a robot $u$. The proof follows by unraveling the definitions and applying Proposition 7.3.

**Proposition 7.6.** $\theta^u(v, w) = \theta^u(w, v) + \pi \pmod{2\pi}$

*Proof.* Applying Proposition 7.3 to robots $v$ and $w$ we have that,

$$\begin{aligned}
\theta(v, w) + \phi_v &= \theta(w, v) + \phi_w + \pi \pmod{2\pi} && \\
\theta(v, w) + \phi_v - \phi_u &= \theta(w, v) + \phi_w - \phi_u + \pi \pmod{2\pi} && \text{subtracting } \phi_u \text{ on both sides} \\
\theta(v, v) + h_{uv} &= \theta(w, v) + h_{uw} + \pi \pmod{2\pi} && \text{by Definition 7.1} \\
\theta^u(v, w) &= \theta^u(w, v) + \pi \pmod{2\pi} && \text{by Definition 7.5}
\end{aligned}$$

which concludes our proof. □

Finally we define the concept of an angle-constraint on a graph.

**Definition 7.7.** *A function $\omega : \overleftrightarrow{E}_G \to [0, 2\pi)$ that associates an angle to each directed edge of $G$ is an* angle-constraint *on $G$ iff $\omega(u, v) = \omega(v, u) + \pi \pmod{2\pi}$ for every edge $(u, v) \in \overleftrightarrow{E}_G$.*

From this definition it is easy to verify that $\Theta^u_G$ is an angle-constraint on $G$ for any $u \in V_G$ (this follows immediately from Proposition 7.6). We refer to the tuple $(G, \omega)$ as an angle-constrained graph.

**Definition 7.8.** *A function $\ell : \overleftrightarrow{E}_G \to \mathbb{R}^+$ that associates a length to each directed edge of $G$ is A a* length-constraint *on $G$ iff $\ell(u, v) = \ell(v, u)$ for every edge $(u, v) \in \overleftrightarrow{E}_G$.*

Observe that in contrast with angle-constraints, length-constraints assign the same length to both directions of an undirected edge. We refer to the tuple $(G, \ell)$ as a length-constrained graph.

## 7.2 Computing an Angle-Constraint

The main contribution of this section is to describe and prove the correctness of the centralized ComputeAngleConstraint procedure. This procedure allows us to

use the angle measurements available in $\Theta_G$ to compute the angle-constraint $\Theta_G^u$ on $G$ for some $u \in V_G$, as well as the relative orientations of all robots in $G$ with respect to robot $u$.

First observe that from Definition 7.5 it follows that if we are given $\Theta_G$ then to compute $\Theta_G^u$ it suffices to compute $h_{uv}$ for every $v \in V_G$. Therefore the problem of computing $\Theta_G^u$ from $\Theta_G$ can be reduced to the problem of computing $h_{uv}$ for every $v \in V_G$.

Moreover, for the case when robots $u$ and $v$ are neighbors, the angle measurements $\theta(u, v)$ and $\theta(v, u)$ are available in $\Theta_G$. This allows us to use Proposition 7.4 directly to compute $h_{uv}$. Subsequently from Definition 7.5 it follows that we can use $h_{uv}$ together with $\Theta_G$ to compute $\theta^u(v, w)$ for every $w \in N_G(v)$.

However, in the case when robots $u$ and $v$ are not neighbors then the angle measurements $\theta(u, v)$ and $\theta(v, u)$ are not available in $\Theta_G$. This complicates things slightly, since we are unable to leverage Proposition 7.4 to compute $h_{uv}$ directly.

The centralized ComputeAngleConstraint procedure essentially leverages Proposition 7.4 repeatedly to compute $h_{uv}$ for the case when the robots $u$ and $v$ are not necessarily neighbors, but are connected through a path. Informally the ComputeAngleConstraint procedure first computes a spanning tree rooted at $u$ and then "pushes" the orientation of the root robot down the levels of the tree. To push the orientation from one level to the next the algorithm relies on Proposition 7.4.

The ComputeAngleConstraint procedure receives as input an undirected graph $G$, the angle measurements $\Theta_G$, and a distinguished node $u \in V_G$. It computes the relative orientations $h_{uv}$ for all $v \in V_G$, and the angle-constraint $\Theta_G^u$ on $G$.

**Algorithm 10** ComputeAngleConstraint $(G, \Theta_G, u)$

$T \leftarrow$ spanning tree of $G$ rooted at $u$
$H(u, u) \leftarrow 0$
**for each** $z \in N_G(u)$ **do**
    $L(u, z) \leftarrow \theta(u, z)$
**for** $k := 1$ to height of $T$ **do**
    **for each** $v$ at level $k$ in $T$ **do**
        $w \leftarrow$ parent of $v$ in $T$
        $H(u, v) \leftarrow L(w, v) - \theta(v, w) - \pi \pmod{2\pi}$
        **for each** $z \in N_G(v)$ **do**
            $L(v, z) \leftarrow \theta(v, z) + H(u, v) \pmod{2\pi}$

**Theorem 7.9.** *After executing* ComputeAngleConstraint$(G, \Theta_G, u)$ *we have that* $H(u,w) = h_{uw}$ *for every* $w \in V_G$*, and* $L(v,w) = \theta^u(v,w)$ *for every* $(v,w) \in \overleftrightarrow{E}_G$.

This theorem follows from an inductive application of Proposition 7.4.

*Proof.* Let $parent(w)$ and $level(w)$ denote the parent of node $w$ and the level of node $w$ in the spanning tree used by the algorithm. Since the spanning tree is rooted at $u$ then $level(u) = 0$ and $parent(u)$ is undefined.

Fix a node $v \in V_G$. To prove the theorem it suffices to show that the algorithm computes the relative orientation $H(u,v) = h_{uv}$ and the angle measurements $L(v,z) = \theta^u(v,z)$ for every $z \in N_G(v)$.

We prove this by induction on the level of node $v$. The base case is trivial, since $level(v) = 0$ implies $v \equiv u$ and we have $H(u,u) = 0 = h_{uu}$ and $L(u,z) = \theta(u,z) = \theta^u(u,z)$ for every $z \in N_G(u)$, as required.

For the inductive step we suppose $v$ has $level(v) = k+1$ and we let $w = parent(v)$. Since $level(w) = k$ then the inductive hypothesis implies that $H(u,w) = h_{uw}$ and $L(w,z) = \theta^u(w,z)$ for every $z \in N_G(w)$. In particular this implies that $L(w,v) = \theta^u(w,v) = \theta(w,v) + h_{uw}$.

By construction of the algorithm we have that $H(u,v) = L(w,v) - \theta(v,w) - \pi$, and since $L(w,v) = \theta(w,v) + h_{uw}$ we have that $H(u,v) = \theta(w,v) + h_{uw} - \theta(v,w) - \pi$. Moreover, Proposition 7.4 implies that $h_{wv} = \theta(w,v) - \theta(v,w) - \pi$ and replacing this in $H(u,v)$ we have that

$$H(u,v) = h_{uw} + h_{wv} = \phi_w - \phi_u + \phi_v - \phi_w = \phi_v - \phi_u = h_{uv}.$$

Finally since by construction the algorithm sets $L(v,z) = \theta(v,z) + H(u,v) = \theta(v,z) + h_{uv} = \theta^u(v,z)$ for every $z \in N_G(v)$, the rest of the statement follows. □

# Chapter 8

# Relative Positions

Building upon the results in Chapter 7, in this chapter we avoid dealing with the angle measurements and instead we assume we are given a graph $G$ and an angle-constraint $\omega$ on $G$. The main contribution of this chapter is to describe a centralized procedure that given an angle-constrained graph $(G, \omega)$ and a subset of robots $S$, computes an assignment of positions to the robots in $S$ which satisfies the angle-constraint $\omega$.

**Roadmap.** Section 8.1 defines graph realizations, which are simply embeddings of the vertices of the graph in the Euclidean plane. Informally speaking, we say a realization satisfies an angle-constrained graph, if for adjacent vertices in the graph the angles between the vertices in the embedding match the corresponding angles in the angle-constraint.

Section 8.2 describes a simple mathematical framework that relates the problem of finding a realization that satisfies an angle-constrained graph, to the problem of computing the null space basis of a matrix which can be derived efficiently from an angle-constrained graph. As mentioned in the introduction, Section 8.2 presents an alternative derivation to that descried in the earlier and seminal results of Whiteley [94], who considered a very similar problem following the traditional graph rigidity approach. Chapter 9 argues how this characterization can be used to compute (if possible) relative positions for *all* the robots.

Section 8.3 extends this framework to the problem of finding a realization of an angle-constrained graph for only a particular subset of robots. This might be possible even when finding a realization for all the robots is impossible.

## 8.1 Defining Unique Realizations

A *realization* of a set $V$ is a function $p : V \to \mathbb{R}^2$ that maps each element of $V$ to a point in the Euclidean plane. A *realization of a graph* is a realization of its vertices. We refer to realizations which map all elements to the same point as trivial realizations.

Let $\mathsf{S}^2 := \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1\}$ be the space of two-dimensional unit vectors. We define the function $\psi : [0, 2\pi) \to \mathsf{S}^2$ that maps an angle to a two-dimensional unit vector, concretely we let $\psi(z) := \begin{bmatrix} \cos z & \sin z \end{bmatrix}^T$. Since $\psi$ is invertible via the atan2 function we can define the function $\psi^{-1} : \mathsf{S}^2 \to [0, 2\pi)$ which maps a unit vector $\hat{n}$ to an angle $z = \psi^{-1}(\hat{n})$ such that $\psi(z) = \hat{n}$. We define the *angle of a vector* as the counter-clockwise angle from the $x$-axis of the coordinate system to the vector.

Recall from Definition 7.7 in Chapter 7 that a function $\omega : \vec{E} \to [0, 2\pi)$ that associates with every directed edge of $G$ an angle is an angle-constraint on $G$ if for every edge $\{u, v\} \in E_G$ we have that $\omega(u, v) = \omega(v, u) + \pi \pmod{2\pi}$ and therefore $\psi(\omega(u, v)) = -\psi(\omega(v, u))$. Similarly from Definition 7.8 we have that a function $\ell : \vec{E} \to \mathbb{R}^+$ that associates with every directed edge of $G$ a length is a length-constraint on $G$ if for every edge $\{u, v\} \in E_G$ we have that $\ell(u, v) = \ell(v, u)$.

Next we define what it means for a realization to satisfy an angle-constraint.

**Definition 8.1.** *Fix a graph $G$, a realization $p$ of $G$ and an angle-constraint $\omega$ on $G$. We say $p$ is a* satisfying realization *of $(G, \omega)$ iff every edge $(u, v) \in \overleftrightarrow{E}_G$ satisfies* $p(v) - p(u) = \psi(\omega(u, v)) \, \|p(v) - p(u)\|$ [*].

Analogously we define what it means for a realization to satisfy a length-constraint.

**Definition 8.2.** *Fix a graph $G$, a realization $p$ of $G$ and a length-constraint $\ell$ on $G$. We say $p$ is a* satisfying realization *of $(G, \ell)$ iff every edge $(u, v) \in \overleftrightarrow{E}_G$ satisfies* $\|p(v) - p(u)\| = \ell(u, v)$.

Next, we introduce some basic geometry definitions. If $p$ is a realization, we use $p(v)^T$ to denote a column vector representing the point $p(v)$. We say a realization $p'$ is a *translation* of $p$ if for some $t \in \mathbb{R}^2$ it holds that $p'(v) = p(v) + t$ for every $v \in V$. We say a realization $p'$ is a *positive uniform-scaling* of $p$ if for some $s \in \mathbb{R}^+$ it holds that $p'(v) = sp(v)$. We say a realization $p'$ is a *rotation* of $\theta \in [0, 2\pi)$ of $p$ if it holds that $p'(v)^T = R(\theta)p(v)^T$ where $R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$. We say a realization $p'$

[*]This definition allows any angle-constraint to be satisfied by a trivial realization that maps every vertex to the same point. A more substantive discussion about the meaning of trivial realizations appears at the end of Section 8.2.

is a *reflection* of $p$ if it holds that $p'(v)^T = Qp(v)^T$ where $Q = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ (in other words we consider only reflections on the $x$-axis, since any other reflection can be expressed as a reflection on the $x$-axis followed by a rotation and a translation).

To analyze the realizations that satisfy a particular angle-constraint or length-constraint, it will be useful to define two different equivalence relations that induce two different partitions of the set of realizations.

**Definition 8.3.** *Fix realizations $p$ and $p'$ of $V$. We say $p$ is* angle-equivalent *to $p'$ if $p'$ can be obtained from $p$ by a positive uniform-scaling followed by a translation.*

Observe that the effect of performing a translation followed by a scaling can also be achieved by first performing a scaling and then a translation (the same scaling will be used, but the translation is adjusted). Also observe that the effect of any sequence of translations and scalings can be obtained by a single scaling followed by a single translation. These remarks imply that the angle-equivalence relation is an equivalence relation.

The following proposition shows that two realizations are angle-equivalent if and only if there exists a positive scalar $s$ such that for any pair of vertices the vector between them in one realization is the same as in the other realization but multiplied by $s$.

**Proposition 8.4.** *The realizations $p$ and $p'$ of $V$ are angle-equivalent if and only if there exists $s \in \mathbb{R}^+$ such that for every $u, v \in V$ it follows that $p'(v) - p'(u) = s(p(v) - p(u))$.*

*Proof.* First assume that $p$ and $p'$ are angle-equivalent. From this it follows that there exists $s_0 \in \mathbb{R}^+$ and $t_0 \in \mathbb{R}^2$ such that $p'(v) = s_0 p(v) + t_0$ for every $v \in V$. From this it follows that for every pair of elements $u, v \in V$ we have $p'(v) - p'(u) = s_0(p(v) - p(u))$ as needed.

Next, assume that there exists $s \in \mathbb{R}^+$ such that for every $u, v \in V$ it holds that $p'(v) - p'(u) = s(p(v) - p(u))$. It suffices to define $t_0 \in \mathbb{R}^2$ such that $p'(v) = sp(v) + t_0$ for every $v \in V$. Let $t_0 = p'(w) - sp(w)$ for an arbitrary $w \in V$. We need to show only that for any other $v \in V$ it holds that $p'(v) = sp(v) + t_0$ or equivalently that $t_0 = p'(v) - sp(v)$. Observe that by assumption we have that $p'(v) - p'(w) = s(p(v) - p(w))$ and rearranging the terms we obtain $p'(w) - sp(w) = t_0 = p'(v) - sp(v)$ as needed. □

In fact, it turns out that for non-degenerate point sets, that is, point sets for which no three points are collinear, the following proposition holds (observe that the order of the quantifiers is reversed).

**Proposition 8.5.** *The realizations $p$ and $p'$ of $V$ are angle-equivalent if and only if for every $u, v \in V$ there exists $s \in \mathbb{R}^+$ such that $p'(v) - p'(u) = s(p(v) - p(u))$.*

Next we define the concept of length-equivalence, that is in many ways analogous to angle-equivalence but using lengths instead of angles.

**Definition 8.6.** *Fix realizations $p$ and $p'$ of $V$. We say $p$ is* length-equivalent *to $p'$ if for all pairs of elements $u, v \in V$ we have that $\|p'(u) - p'(v)\| = \|p'(u) - p'(v)\|$.*

As before, it is easy to verify that the length-equivalence relation is symmetric, reflexive and transitive, and is therefore an equivalence relation. The following proposition characterizes the type of transformations that relate two length-equivalent realizations.

**Proposition 8.7.** *Fix realizations $p$ and $p'$ of $V$. Then $p$ is length-equivalent to $p'$ if and only if $p$ can be obtained from $p'$ by a reflection, followed by a rotation, followed by a translation.*

*Proof.* In Euclidean geometry two point sets that have the same pairwise distances are said to be congruent, and one can be obtained from the other by an isometry [14]. Moreover, it is known that a transformation is an isometry if and only if it can be decomposed into a translation, a rotation and a reflection.

Therefore the proposition follows by definition of the length-equivalence relation. □

Finally, we say an angle-constrained graph $(G, \omega)$ has a *unique realization* if and only if it has at least one non-trivial satisfying realization and all its non-trivial satisfying realizations are angle-equivalent. Analogously, we say a length-constrained graph $(G, \ell)$ has a *unique realization* if and only if it has at least one satisfying realization and all its satisfying realizations are length-equivalent.

**Discussion.** A satisfying realization of $(G, \omega)$ is by definition a mapping that associates with each robot in $V_G$ a position in the Euclidean plane where for every pair of adjacent robots in $G$, the angle of the vector between their associated positions matches the corresponding angle in $\omega$. Therefore if $(G, \omega)$ has no non-trivial satisfying realization then, except for the trivial assignment that maps all robots to the same point, there is no assignment of positions to robots which is consistent with the angle-constraint $\omega$.

EXAMPLE 8.1: Let $G$ be a cycle graph on three vertices $\{u, v, w\}$. Consider the angle-constraint $\omega$ on $G$ that assigns $\omega(u, v) = 0$, $\omega(v, w) =$

$\pi/8$ and $\omega(w,u) = 0$. Except for the trivial realizations that map $u, v$ and $w$ to the same point, there is no realization that satisfies this angle-constrained graph $(G, \omega)$. One way to see this, is that in any non-trivial realization of $u, v$ and $w$, if the angle-constraint in $\omega$ is satisfied then the interior angles of the triangle formed by $u, v$ and $w$ cannot sum to $\pi$ (which is a contradiction since no such triangle can exist).

On the other hand if $(G, \omega)$ has multiple satisfying realizations which are not angle-equivalent, then there are several assignments of positions to robots which are consistent with the angle-constrained graph, and these assignments are *not* just translations and positive uniform-scalings of each other (for example, see Figure 8-1).

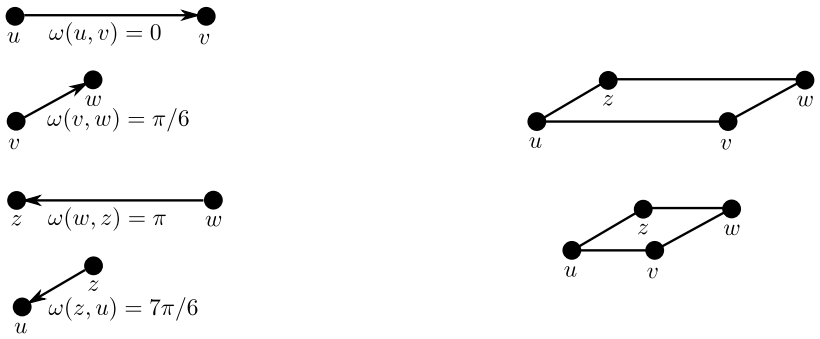



Figure 8-1: Let $G$ be a cycle graph on four vertices $\{u, v, w, z\}$. The left part of the figure depicts the direction of the vectors between $\{u, v\}$, $\{v, w\}$, $\{w, z\}$ and $\{z, u\}$ for an angle-constraint that assigns $\omega(u, v) = 0$, $\omega(v, w) = \pi/6$, $\omega(w, z) = \pi$ and $\omega(z, u) = 7\pi/6$. The right part of the figure shows two non-trivial satisfying realizations of the angle-constrained graph where one *cannot* be obtained from the other by a translation and a uniform-scaling operation (i.e. these realizations are not angle-equivalent).

However, if $(G, \omega)$ has a unique satisfying realization, then it implies that there is at least one non-trivial assignment of positions to robots that is consistent with the angle-constraint, and all non-trivial assignments of positions to robots which are consistent with the angle-constraint are translations and uniform-scalings of each other.

## 8.2 Satisfying Realizations

Given an angle-constrained graph $(G, \omega)$, the questions that we are concerned with in this section are: Does the angle-constrained graph have any satisfying non-trivial

realizations? If it does, what are they? Are they all angle-equivalent?

**Roadmap.** Section 8.2.1 states some simple facts about realizations of graphs. Section 8.2.2 studies satisfying realizations of angle-constrained trees. Section 8.2.3 describes the restrictions on a satisfying realization imposed by cycles. Section 8.2.4 describes a relationship between satisfying realizations and the restrictions imposed by all the cycles in a graph. Section 8.2.5 describes an explicit method for computing the satisfying realizations of an angle-constrained graph.

### 8.2.1 Basic Facts

First observe that a realization $p$ of $G$ defines a length-constraint and an angle-constraint which are simultaneously satisfied by $p$. This is captured in the following algorithm and accompanying lemma.

**Algorithm 11** RealizationToConstraints$(G, p)$

**for each** $(u, v) \in \overleftrightarrow{E}_G$ **do**
  $\ell(u, v) \leftarrow \|p(v) - p(u)\|$
  **if** $\ell(u, v) \geq 0$ **then**
    $\omega(u, v) \leftarrow \psi^{-1}((p(v) - p(u))/\|p(v) - p(u)\|$
  **else**
    $\omega(u, v) \leftarrow 0$
**return** $\omega, \ell$

**Lemma 8.8.** *Fix a graph $G$ and a realization $p$ of $G$. The angle-constraint and length-constraint returned by* RealizationToConstraints$(G, p)$ *are simultaneously satisfied by $p$.*

*Proof.* The fact that the length-constraint and angle-constraint returned by RealizationToConstraints$(G, p)$ are simultaneously satisfied by $p$ follows immediately by construction. We note that any variation to the lengths of the length-constraint will result in a length-constraint which is no longer satisfied by $p$. Moreover, except for the edges of zero length, any variation to the angles of an angle-constraint will also result in an angle-constraint which is not satisfied by $p$. □

On the other hand, there exist length- and angle-constraints that cannot be satisfied simultaneously by any realization. For instance, if a length-constraint does not

satisfy the triangle inequality then no realization can satisfy it (much less satisfy it simultaneously with an angle-constraint). This implies the converse of Lemma 8.8 does not hold in general.

### 8.2.2 Trees

The next lemma shows that if we restrict the graph to be a tree, then a statement that is analogous to the converse of Lemma 8.8 holds.

**Algorithm 12** TreeConstraintToRealization$(T, \omega, \ell)$

$v_0 \leftarrow$arbitrary vertex in $V_T$
**for each** $v \in V_T$ **do**
  $P(v_0, v) \leftarrow$ unique directed path from $v_0$ to $v$ in $T$
  $p(v) \leftarrow \sum_{e \in P(v_0,v)} \ell(e)\psi(\omega(e))$
**return** $p$

**Lemma 8.9.** *Fix a tree $T$, an angle-constraint $\omega$ on $T$ and a length-constraint $\ell$ on $T$. The realization returned by* TreeConstraintToRealization$(T, \omega, \ell)$ *simultaneously satisfies $\omega$ and $\ell$, and is unique up to translations.*

*Proof.* It is not hard to verify that by construction the realization output by the algorithm is such that every edge $e \in E_T$ in the tree has length $\ell(e)$ and angle $\omega(e)$ in $p$. Therefore $p$ simultaneously satisfies both $\ell$ and $\omega$. It remains only to show that $p$ is unique up to translations.

Let $p'$ be any angle- and length-satisfying realization of $T$, we will show that $p$ is a translation of $p'$. Specifically we define $r_0 = p(v_0) - p'(v_0)$ for an arbitrary vertex $v_0 \in V_T$ and we show that $\forall v \in V_T$ we have $p(v) = p'(v) + r_0$. We proceed by induction on the number of edges in the directed path $P(v_0, v)$ from $v_0$ to $v$ in $T$.

The base case is trivial, since $|P(v_0, v)| = 0$ implies $v = v_0$, and by definition of $r_0$ it holds that $p(v) = p'(v) + r_0$.

For the inductive step, consider a vertex $v$ such that $|P(v_0, v)| = k$ for some $k > 0$. There must exist a vertex $w \in V_T$ such that $|P(v_0, w)| = k - 1$ and $(w, v) \in E_T$.

Consider the edge $(w, v)$. Since by assumption both $p$ and $p'$ simultaneously satisfy $\omega$ and $\ell$, then $p(w) - p(v) = \ell(v, w)\psi(\omega(v, w)) = p'(w) - p'(v)$. Moreover, since $P(v_0, w) = k - 1$ then by the inductive hypothesis we have $p(w) = p'(w) + r_0$. We can combine these two equations to yield that $p'(w) + r_0 - p(v) = p'(w) - p'(v)$. Canceling and rearranging terms we have $p(v) = p'(v) + r_0$, which completes the proof. □

The following corollary is a direct consequence of the previous result.

**Corollary 8.10.** *An angle-constrained tree always has a satisfying realization, but never a unique realization.*

Additionally, we can leverage Lemma 8.9 to prove the following result for general graphs.

**Algorithm 13** ConstraintToRealization$(G, \omega, \ell)$

$T \leftarrow$ spanning tree of $G$
$\omega_T \leftarrow \omega$ projected to the edges of $T$
$\ell_T \leftarrow \ell$ projected to the edges of $T$
**return** TreeConstraintToRealization$(T, \omega_T, \ell_T)$

**Lemma 8.11.** *Let $G$ be a graph and let $\omega$ and $\ell$ be an angle-constraint and a length-constraint that can be simultaneously satisfied by $G$. Then the realization returned by* ConstraintToRealization$(G, \omega, \ell)$ *(a) simultaneously satisfies $\omega$ and $\ell$, and (b) is unique up to translations.*

*Proof.* Let $p$ be the realization returned by ConstraintToRealization$(G, \omega, \ell)$, and let $T$, $\omega_T$ and $\ell_T$ be the values taken by the corresponding internal variables of this procedure.

Lemma 8.9 implies that $p$ simultaneously satisfies $\omega_T$ and $\ell_T$, and it also implies that $p$ is unique up to translations and therefore any other realization that satisfies $\omega_T$ and $\ell_T$ must be a translation of $p$. Fix $p'$ to be *any* satisfying realization of $\omega$ and $\ell$; we know at least one such realization exists by assumption. Observe that $p'$ must also satisfy $\omega_T$ and $\ell_T$, and therefore $p'$ must be a translation of $p$.

We claim that this completes the lemma, since by showing that $p$ is a translation of $p'$ we have shown that (a) $p$ simultaneously satisfies $\omega$ and $\ell$, since $p'$ satisfied $\omega$ and $\ell$ by assumption and $p'$ is just a translation of $p$, and (b) $p$ is unique up to translations, since $p'$, which was chosen to be any realization that simultaneously satisfies $\omega$ and $\ell$, is a translation of $p$.

□

The results we have proved so far reveal a close correspondence between the set of satisfying realizations of an angle-constrained graph $(G, \omega)$, and the set of length-constraints that can be satisfied simultaneously with the angle-constraint $\omega$. Namely Lemma 8.8 allows us to translate a realization that satisfies an angle-constraint graph $(G, \omega)$ to a length-constraint that can be satisfied simultaneously with $\omega$. On

the other hand, Lemma 8.11 allows us to translate a length-constraint that can be simultaneously be satisfied with an angle-constraint $\omega$ to a realization that satisfies $(G, \omega)$.

### 8.2.3 Facts about Cycles

Corollary 8.10 suggests that the restrictions that make satisfying realizations unique (or impossible to construct) must be encoded in the cycles of a graph (defined formally below). This subsection describe how a cycle yields a pair of equations that must be satisfied by any satisfying realization.

A *cycle* is a set of edges that can be arranged end-to-end to describe a path from a vertex to itself. The *traversal of a cycle* is a consistent orientation of the edges of the cycle that allows us to start at any vertex and follow the directions of the edges to eventually return to the starting vertex. Observe that any cycle has only two possible traversals. For a cycle $C$ we let $\vec{C}$ and $\overleftarrow{C}$ denote the two possible traversals.

For any realization $p$ and an undirected cycle $C$ defined on the vertices of $p$, we have the following proposition (which follows from the fact that after traversing a cycle we return to the starting vertex).

**Proposition 8.12.** *Let $G$ be a graph and let $p$ be a realization of $G$. For every cycle $C$ in $G$ the following equation must hold.*

$$\sum_{(u,v)\in E_{\vec{C}}} (p(v) - p(u)) = \sum_{(u,v)\in E_{\overleftarrow{C}}} (p(v) - p(u)) = \mathbf{0} \tag{8.1}$$

Since the two possible traversals of a cycle produce exactly the same equation, from here on we can consider only one of them. Given an angle-constraint $\omega$ and a length-constraint $\ell$ that can be satisfied simultaneously by a realization, we can rewrite Proposition 8.12 as follows.

**Proposition 8.13.** *Let $G$ be a graph and let $\omega$ and $\ell$ be an angle-constraint and length-constraint on $G$ that can be satisfied simultaneously. For every cycle $C$ in $G$ the following equation must hold.*

$$\sum_{e\in E_{\vec{C}}} \ell(e)\psi(\omega(e)) = \mathbf{0} \tag{8.2}$$

Since the terms of the equation are two-dimensional vectors, every cycle generates two scalar equations, one for the $x$-component and one for the $y$-component.

### 8.2.4 Graphs with Cycles

This subsection establishes a relationship between the set of satisfying realizations of an angle-constrained graph and the solutions to a system of equations derived from the cycles present in the graph. Concretely, through Equation 8.2, a cycle in $G$ represents two restrictions that must be satisfied by a realization of $G$.

Our goal is to construct a system of equations that captures the restrictions encoded by all the cycles in the graph. However, the number of cycles in a graph can be exponential on the number of edges in the graph, and therefore considering the equations generated by every cycle in the graph may lead to a very large system of equations that would later yield very inefficient algorithms. Instead we will construct a reduced set of equations that express the same constraints. The observation that allows us to do this, is that a large number of cycles must share a large number of edges, and this implies that the constraints imposed by these cycles are not all independent.

In particular we will show that it is enough to consider only the fundamental cycles of a spanning tree of $G$ (defined below).

**Definition 8.14.** *Let $\mathcal{T}$ be a spanning tree of $G$. Every non-tree edge $\{u, v\}$ defines a* fundamental cycle *of $\mathcal{T}$, which is the cycle formed by the union of the edge $\{u, v\}$ and the edges in the unique path between $u$ and $v$ in $\mathcal{T}$.*

We refer to a set $\mathcal{C}$ of cycles as a *fundamental cycle basis* of $G$, if there exists a spanning tree $T$ of $G$ such that $\mathcal{C}$ is equal to the set of all fundamental cycles of $T$.

Now we are ready to state and prove the main theorem of this subsection.

**Theorem 8.15.** *Let $G$ be a graph, let $\omega$ be an angle-constraint on $G$, let $\ell$ be a length-constraint on $G$, and let $\mathcal{C}$ be a fundamental cycle basis of $G$. There is a realization that simultaneously satisfies $\omega$ and $\ell$ if and only if $\omega$ and $\ell$ satisfy Equation 8.2 for every cycle in $\mathcal{C}$.*

*Proof.* We prove each direction separately.

**Case $\Rightarrow$:** Suppose there is a realization $p$ that simultaneously satisfies $\omega$ and $\ell$. From Proposition 8.13 it follows that $p$ satisfies Equation 8.2 for every cycle in $G$, and this implies that $p$ satisfies Equation 8.2 for every cycle in $\mathcal{C}$.

Proving the other direction is more involved, since we need to show that all the satisfiability restrictions which are implicit in the graph are captured by considering Equation 8.2 for the cycles in $\mathcal{C}$.

**Case $\Leftarrow$:** Suppose $\omega$ and $\ell$ satisfy Equation 8.2 for every cycle in $\mathcal{C}$. Let $T$ be the spanning tree of $G$ used to generate the fundamental cycles in $\mathcal{C}$, and let $\omega_T$ and

$\ell_T$ correspond to $\omega$ and $\ell$ projected to the edges of the spanning tree $T$ respectively. Lemma 8.9 implies there is a realization $p$ of $T$ that simultaneously satisfies $\omega_T$ and $\ell_T$ and is unique up to translations.

Since realizations are mappings of vertices (and not edges), it follows that $p$ is also a realization of $G$. To complete our proof it suffices to show that $p$ simultaneously satisfies $\omega$ and $\ell$. Moreover, since by construction $p$ satisfies $\omega_T$ and $\ell_T$, we need only to show that $p$ satisfies $\omega$ and $\ell$ for the edges in $F$, where $F$ is the set of (directed) edges that are not covered by the spanning tree $T$.

Fix an edge $(u, v) \in F$; we show that $p$ satisfies $\omega$ and $\ell$ on $(u, v)$. The edge $(u, v) \in F$, together with the edges on the unique path between $u$ and $v$ contained in $T$ describe a fundamental cycle $C$ of $G$. Since by assumption Equation 8.2 is satisfied on every fundamental cycle of $T$ it follows that Equation 8.2 is satisfied for $C$. Moreover, except for the edge $(u, v) \in F$, by construction of $p$, individually all other edges in the cycle satisfy $\omega$ and $\ell$, and since Equation 8.2 is satisfied in $C$ it follows that $\omega$ and $\ell$ are also satisfied on the edge $(u, v)$. □

Theorem 8.15 establishes a close correspondence between the length-constraints that can be satisfied simultaneously with an angle-constraint, and the solution set of a system of equations defined over the fundamental cycles of a spanning tree of the graph. Moreover, Lemma 8.11 allows us to translate these length-constraints to satisfying realizations of the angle-constraint graph.

**A Remark on Theorem 8.15.** Theorem 8.15 can be generalized by considering any cycle basis (defined below) of $G$ as opposed to a fundamental cycle basis of $G$ (which is just a particular kind of cycle basis). Here we briefly discuss this generalization.

In a graph $G$ with an edge set $\{e_1, \ldots, e_m\}$ a simple cycle can be represented by a $\{0, 1\}$ edge incidence vector $[\lambda_1, \ldots, \lambda_m]$ where for every $i \in \{1, \ldots, m\}$ we have $\lambda_i = 1$ if the cycle includes edge $e_i$, and $\lambda_i = 0$ otherwise. Next we define a vector space associated with the cycles in the graph.

**Definition 8.16.** *The* cycle space *of a graph $G$ is the vector space over $\{0, 1\}$ generated by the incidence vectors associated with the directed cycles in $G$,*

In particular, it turns out that it suffices to consider only the restrictions imposed by the cycles in a *cycle basis* of the cycle space of the graph, which we define below.

**Definition 8.17.** *A* cycle basis *of a graph $G$ is a set of simple cycles in $G$ that have linearly independent edge incidence vectors and that span the cycle space of $G$.*

For a more in depth discussion about cycle bases and their properties, we refer the interested reader to [41, 56]. For our purposes it suffices to know that every connected graph on $n$ vertices and $m$ edges has a cycle basis with $m - n + 1$ cycles. Moreover, it is known [41] that the fundamental cycles of any spanning tree of $G$ define a cycle basis of $G$.

With minor modifications, the proof of Theorem 8.15 can be extended to prove the same result holds for any cycle basis of $G$, without requiring it to be a fundamental cycle basis.

### 8.2.5 Computing Satisfying Realizations

Lemma 8.11 showed that computing a satisfying realization of an angle-constrained graph $(G, \omega)$ is equivalent to computing a length-constraint that can be satisfied simultaneously with the angle-constraint $\omega$. Theorem 8.15 showed that computing a length-constraint that can be satisfied simultaneously with an angle-constraint $\omega$ is equivalent to computing a length-constraint that satisfies a system of equations defined over a fundamental cycle basis of $G$. This subsection describes how to use standard linear algebra tools to solve the previously defined system of equations and compute the set of length-constraints on $G$ that can be satisfied simultaneously with an angle-constraint $\omega$ on $G$.

To borrow the tools of linear algebra it will be useful to represent a length-constraint as a real vector. To simplify our discussion, we introduce some additional definitions. Let $\mathcal{E}_G = \{e_1, \ldots, e_m\}$ be a subset of the directed edges $\overleftrightarrow{E}_G$ such that every undirected edge in $E_G$ has one of its directed counterparts (but not both) present in $\mathcal{E}_G$. Let $\mathbf{x}$ be an $m \times 1$ column vector whose $i^{\text{th}}$ entry represents the length of the directed edge $e_i \in \mathcal{E}_G$. In an angle-constrained graph $(G, \omega)$, every cycle in $G$ represents, through Equation 8.2, a linear restriction on any length-constraint which can be satisfied simultaneously with $\omega$. The following matrix equation captures the linear restrictions imposed by Equation 8.2 on every cycle of a fundamental cycle

basis $\mathcal{C} = \{C_1, \ldots, C_q\}$ of $G$.

$$\begin{array}{c} \\ C_1 \\ \vdots \\ C_q \end{array} \underbrace{\begin{array}{c} \begin{array}{ccc} e_1 & \cdots & e_m \end{array} \\ \begin{bmatrix} a_{11} & \ldots & a_{1m} \\ b_{11} & \ldots & b_{1m} \\ \vdots & \ddots & \vdots \\ a_{q1} & \ldots & a_{qm} \\ b_{q1} & \ldots & b_{qm} \end{bmatrix} \end{array}}_{A_{(\mathcal{C},\omega)}} \underbrace{\begin{bmatrix} \ell(e_1) \\ \vdots \\ \ell(e_m) \end{bmatrix}}_{\mathbf{x}} = \mathbf{0} \tag{8.3}$$

Since Equation 8.2 is a vector equation then every cycle in $\mathcal{C}$ corresponds to two scalar rows in $A_{(\mathcal{C},\omega)}$; in other words $A_{(\mathcal{C},\omega)}$ is $2q \times m$ matrix. In more detail, for $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, m\}$ we have that $[a_{ij}, b_{ij}]^T = \psi(\omega(e_j))$ if cycle $C_i$ uses edge $e_j$, $[a_{ij}, b_{ij}]^T = -\psi(\omega(e_j))$ if cycle $C_i$ uses edge $\overleftarrow{e}_j$, and $[a_{ij}, b_{ij}]^T = [0, 0]^T$ otherwise. We highlight that by virtue of $\mathcal{C}$ being a fundamental cycle basis, no cycle in $\mathcal{C}$ will use both directions of an edge.

Therefore each length-constraint that satisfies Equation 8.2 for every cycle in $\mathcal{C}$ corresponds to an assignment of $\mathbf{x}$ where $A_{(\mathcal{C},\omega)} \cdot \mathbf{x} = \mathbf{0}$. Since Equation 8.3 is a homogeneous system, the solution space is exactly the null space of $A_{(\mathcal{C},\omega)}$, denoted by $\mathsf{null}(A_{(\mathcal{C},\omega)})$. In a slight abuse of notation for a length-constraint $\ell$ on $G$ we say $\ell \in \mathsf{null}(A_{(\mathcal{C},\omega)})$ if the vector representation of $\ell$ is in the null space of $A_{(\mathcal{C},\omega)}$. Theorem 8.15 has the following immediate corollary.

**Theorem 8.18.** *Let $G$ be a graph, let $\omega$ be an angle-constraint on $G$, let $\ell$ be a length-constraint on $G$, and let $\mathcal{C}$ be a fundamental cycle basis of $G$. There is a realization that simultaneously satisfies $\omega$ and $\ell$ if and only if $\ell \in \mathsf{null}(A_{(\mathcal{C},\omega)})$.*

**Null Space Discussion.** Next we list a few properties of the null space of a matrix and we briefly discuss what are their implications for the space of satisfying realizations of an angle-constrained graph.

First, observe that the null space of a matrix always contains the zero vector. This zero vector solution corresponds to a trivial satisfying realization of an angle-constraint that maps all vertices to the same point. To see why, observe that if the length-constraint of an edge is zero, then any realization that satisfies that length-constraint must map both endpoints of the edge to the same point (regardless of the angle-constraint is for that edge).

Second, if the null space of a matrix contains the vector $\mathbf{x}$, then it also contains the vector $c \cdot \mathbf{x}$ for any positive constant $c \in \mathbb{R}^+$. This corresponds to the fact that

satisfying realizations of angle-constrained graphs are equivalent up to a positive uniform-scaling. Therefore if a particular realization satisfies $(G, \omega)$ then a positive uniform-scaling of the realization also satisfies $(G, \omega)$.

Third, recall that each column of $A_{(\mathcal{C},\omega)}$ corresponds to an edge in $G$. If a column $i$ is all zeros, then the null space of $A_{(\mathcal{C},\omega)}$ allows any length assignment for edge $e_i$. However, column $i$ is all zeros if and only if edge $e_i$ is not used by any cycle in $\mathcal{C}$, and this happens only when there is a partition of the graph in to two components where $e_i$ is the only edge between them. Therefore, the fact that the null space allows any assignment for edge $e_i$ corresponds to the fact that these two connected components can be shifted and scaled independently of each other, while respecting the angle-constraint on $e_i$.

In principle the null space of a matrix can contain an infinite number of vectors. Therefore, in order to be able to compute it we first need a concise way of representing it, which we introduce below.

**Definition 8.19.** *A* null space basis *of a matrix is a set of linearly independent column vectors that span the null space of the matrix.*

In other words, every vector in the null space of a matrix can be expressed as a linear combination of the columns in a null space basis of the same matrix. The *nullity* of a matrix is the number of dimensions of its null space, or equivalently the number of columns in a null space basis of the matrix.

Let $\mathcal{N}$ be a null space basis of $A_{(\mathcal{C},\omega)}$. By definition each column of $\mathcal{N}$ represents a length-constraint that can be satisfied simultaneously with $\omega$ (for every column of $\mathcal{N}$, the value of row $i$ represents the length of edge $e_i$). Since the columns of the null space basis are linearly independent then the length-constraints represented by two different columns are not scalings of each other. This, together with Theorem 8.18 imply the following proposition.

**Proposition 8.20.** *The columns of a null space basis of $A_{(\mathcal{C},\omega)}$ correspond to non-zero length-constraints of $G$ which can satisfied simultaneously with $\omega$ and that are not scalings of each other.*

Theorem 8.18 has the following immediate corollary relating a null space basis of $A_{(\mathcal{C},\omega)}$ and the space of satisfying realizations of $(G, \omega)$.

**Corollary 8.21.** *Fix an angle-constrained graph $(G, \omega)$ and a fundamental cycle basis $\mathcal{C}$ of $G$:*
**1)** *$(G, \omega)$ has no non-trivial satisfying realization iff the nullity of $A_{(\mathcal{C},\omega)}$ is zero.*
**2)** *$(G, \omega)$ has a unique satisfying realization iff the nullity of $A_{(\mathcal{C},\omega)}$ is one.*

**3)** *$(G,\omega)$ has several distinct non-trivial satisfying realizations iff the nullity of $A_{(\mathcal{C},\omega)}$ is greater than one.*

Therefore, in particular, to determine whether $(G,\omega)$ has a unique satisfying realization, and to compute this realization, it suffices to compute a null space basis of $A_{(\mathcal{C},\omega)}$ where $\mathcal{C}$ is a fundamental cycle basis of $G$. Specifically, if the nullity of $A_{(\mathcal{C},\omega)}$ is one then the unique column vector in a null space basis of $A_{(\mathcal{C},\omega)}$ corresponds to the lengths of the edges of a non-trivial satisfying realization of $(G,\omega)$.

We conclude this section by describing the ComputeRealizations procedure, which is the algorithmic counterpart of Proposition 8.20 and Corollary 8.21. This procedure receives as input a graph $G$ and an angle-constraint $\omega$ on $G$. It produces as an output a set of non-trivial satisfying realizations of $(G,\omega)$.

ComputeRealizations starts by computing a fundamental cycle basis $\mathcal{C}$ of the graph $G$. The fundamental cycle basis $\mathcal{C}$ is then used together with the angle-constraint $\omega$ to produce the matrix $A_{(\mathcal{C},\omega)}$. Next, the algorithm proceeds to compute a null space basis $N$ of $A_{(\mathcal{C},\omega)}$. Finally, each column of the null space basis $N$ (which from Proposition 8.20 correspond to a non-zero length-constraint that can be satisfied simultaneously with $\omega$) is transformed to a non-trivial satisfying realization of $(G,\omega)$ using the ConstraintToRealization procedure described in Section 8.2.2.

**Algorithm 14** ComputeRealizations$(G,\omega)$

$\mathcal{C} \leftarrow$ fundamental cycle basis of $G$
$A_{(\mathcal{C},\omega)} \leftarrow$ matrix of Equation 8.3 using $\mathcal{C}$ and $\omega$
$N \leftarrow$ null space basis of $A_{(\mathcal{C},\omega)}$
$k \leftarrow$number of columns in $N$
$Q \leftarrow \{\}$
**for** $i := 1$ to $k$ **do**
    $\ell_i \leftarrow$ length constraint from column $i$ in $N$
    $p_i \leftarrow$ConstraintToRealization$(G,\omega,\ell_i)$
    $Q \leftarrow Q \cup \{p_i\}$
**return** $Q$

The next theorem states the properties of the ComputeRealizations procedure. Its proof follows directly from Proposition 8.20, Corollary 8.21 and the correctness of the ConstraintToRealization procedure.

**Theorem 8.22.** *Let $Q$ be the output of* ComputeRealizations$(G,\omega)$. *Each element in $Q$ is a non-trivial satisfying realization of $(G,\omega)$.*

1. *$|Q| = 0$ iff $(G,\omega)$ does not have a non-trivial satisfying realization.*

2. $|Q| = 1$ *iff* $(G, \omega)$ *has a unique satisfying realization.*
3. $|Q| \geq 2$ *iff* $(G, \omega)$ *has several non-trivial satisfying realizations which are not angle-equivalent.*

## 8.3 Unique Subset Realizations

Chapter 9 describes a distributed algorithm that leverages our results on the set of satisfying realizations of an angle-constrained graph to compute relative positions (up to scale) for every robot of a multi-robot system. It turns out that it is possible to compute the relative positions (up to scale) of every robot in the multi-robot system if and only if a particular angle-constrained graph has a unique satisfying realization.

However, in our general problem formulation we are interested in computing the relative positions for only a subset of the robots. Furthermore, it is possible (and in practice it will frequently be the case) that it is feasible to compute the relative positions for a subset of robots, but the entire associated angle-constrained graph *does not* have a unique satisfying realization. For this purpose, in this section we introduce a concept which is analogous to a unique realization but is defined for a particular subset of robots.

We start by restricting our notion of a graph, a realization, an angle-constraint and a length-constraint to a subset of vertices. For a subset $S$ of the vertices in $G$, let $G[S]$ be the subgraph of $G$ induced by $S$, formally $V_{G[S]} = S$ and $E_{G[S]} = \{\{u, v\} \mid \{u, v\} \in E_G \wedge u \in S \wedge v \in S\}$. If $p$ is a realization of $G$ then we let $p[S] : S \to \mathbb{R}^2$ be the restriction of $p$ to the vertices in $S$, where for every $u \in S$ we have that $p(u) = p[S](u)$. If $\omega$ is an angle-constraint on $G$ then we define $\omega[S] : \overleftrightarrow{E}_{G[S]} \to [0, 2\pi)$ as the restriction of $\omega$ to $G[S]$ where for every edge $(u, v) \in \overleftrightarrow{E}_{G[S]}$ we have $\omega[S](u, v) = \omega(u, v)$. Similarly, if $\ell$ is a length-constraint on $G$ then we define $\ell[S] : \overleftrightarrow{E}_{G[S]} \to \mathbb{R}^+$ as the restriction of $\ell$ to $G[S]$ where for every edge $(u, v) \in \overleftrightarrow{E}_{G[S]}$ we have $\ell[S](u, v) = \ell(u, v)$.

The next lemma shows that, as natural generalizations of the definitions we presented in Section 8.1, the previous definitions satisfy a number of properties. (The proof of the lemma follows immediately by unravelling the definitions.)

**Lemma 8.23.** *Let $G$ be a graph, let $S \subseteq V_G$ be a subset of vertices of $G$, let $p$ be a realization of $G$, let $\omega$ be an angle-constraint on $G$, and let $\ell$ be a length-constraint on $G$. We have that:*

1. *$\omega[S]$ is an angle-constraint on the graph $G[S]$.*
2. *$\ell[S]$ is a length-constraint on the graph $G[S]$*

*3. $p[S]$ is a realization of the graph $G[S]$.*
*4. if $p$ satisfies $\omega$ then $p[S]$ satisfies $\omega[S]$.*
*5. if $p$ satisfies $\ell$ then $p[S]$ satisfies $\ell[S]$.*

With these definitions in place, we are ready to introduce a concept analogous to unique realizations for a subset of vertices.

**Definition 8.24.** *The angle-constrained graph $(G, \omega)$ is* $S$-unique *if and only if $(G, \omega)$ has at least one satisfying realization $p$ such that $p[S]$ is non-trivial, and every satisfying realization $p'$ of $(G, \omega)$ such that $p'[S]$ is non-trivial is angle-equivalent to $p[S]$.*

This definition gives rise to natural generalizations of the questions we asked at the beginning of Section 8.2, namely: Does an angle-constrained graph have any satisfying realizations that are non-trivial when restricted to the vertices in $S$? If it does, what are they? Is the angle-constrained graph $S$-unique?

It turns out that we have already developed all the tools necessary to address these questions. Below we describe the ComputeSubsetRealizations procedure, which is the natural generalization of the ComputeRealizations procedure for subsets of vertices. The procedure receives as input a graph $G$, an angle-constraint $\omega$ on $G$, and a subset of vertices $S \subseteq V_G$. It produces as an output a set of non-trivial satisfying realizations of $(G[S], \omega[S])$ that are restrictions to the vertices in $S$ of satisfying realizations of $(G, \omega)$.

**Algorithm 15** ComputeSubsetRealizations$(G, \omega, S)$

```
C ← fundamental cycle basis of G
A_(C,ω) ← matrix of Equation 8.3 using C and ω
N ← null space basis of A_(C,ω)
k ←number of columns in N
Q ← {}
for i := 1 to k do
    ℓ_i ← length constraint from column i in N
    p_i ←ConstraintToRealization(G, ω, ℓ_i)
    if p_i[S] is non-trivial and is not angle-equivalent to a realization in Q then
        Q ← Q ∪ {p_i[S]}
return Q
```

The ComputeSubsetRealizations procedure is almost identical to the ComputeRealizations procedure. The only difference is that before inserting the

satisfying realizations of $(G, \omega)$ into the output set $Q$, they are first restricted to the vertices in $S$ and it is verified that they are non-trivial and they are not angle-equivalent to any realization previously inserted into the output set. This modification guarantees that only realizations which are non-trivial when restricted to the vertices in $S$ are returned, and that if two or more realizations are returned, then they are not angle-equivalent. The following theorem states the properties satisfied by the algorithm, which follow by construction and the correctness of the ComputeRealizations procedure.

**Theorem 8.25.** *Let $Q$ be the output of* ComputeSubsetRealizations$(G, \omega, S)$. *Each element in $Q$ is a non-trivial realization of $(G[S], \omega[S])$ that is the restriction to the vertices in $S$ of a satisfying realization of $(G, \omega)$.*

1. *$|Q| = 0$ iff $(G, \omega)$ does not have a satisfying realization $p$ such that $p[S]$ is non-trivial.*
2. *$|Q| = 1$ iff $(G, \omega)$ is $S$-unique.*
3. *$|Q| \geq 2$ iff $(G, \omega)$ has at least two satisfying realizations $p$ and $p'$ such that $p[S]$ and $p'[S]$ are both non-trivial and $p[S]$ is not angle-equivalent to $p'[S]$.*

# Chapter 9

# Distributed Localization Algorithm

Informally, in the localization problem each robot seeks to compute the relative orientations and the relative positions (up to scale) of an arbitrary subset of robots in the system. The specific subset of robots that each robot seeks to localize is application dependent. For instance, in many applications it suffices for each robot to localize only its neighboring robots. However one can also envision applications where each robot wants to localize the robots in the system that have a particular capability, regardless of whether these robots are neighbors or not.

In particular, we consider the localization problem in a very simple sensing model where each robot can measure only the angle, relative to its own orientation, to neighboring robots in the communication graph.

We tackle the localization problem with a two step approach. In the first part each robot computes the relative orientations of the desired subset of robots and converts all the angle measurements available to the same reference frame. This first part of the problem was tackled with a centralized procedure in Chapter 7. For the second part of the problem, the converted angle measurements obtained in the first part are used to compute the relative positions (up to scale) of the desired subset of robots. This second part was tackled with a centralized procedure in Chapter 8.

This chapter describes how to leverage the centralized procedures described in Chapter 7 and Chapter 8 to design the distributed algorithm SubsetLocalize$_k$, which solves a $k$-hop variant of the localization problem. Here $k \in \mathbb{N}$ is a positive integer that determines the number of communication steps used by the distributed algorithm. The specifics of the multi-robot platform and the application determine the ideal value of this parameter. Larger values of $k$ allow each robot to localize robots

that are farther away, but also results in a distributed algorithm that requires more communication steps and is therefore more demanding on the speed and bandwidth of the communication infrastructure of the multi-robot system. We also highlight that larger values of $k$ will rarely prove useful in applications that require localizing only neighboring robots. We argue that the proposed distributed algorithm is optimal in the following sense: if a subset of robots can be successfully localized by any algorithm using $k$ communication steps, then they will be successfully localized by SubsetLocalize$_k$.

Towards the end of the chapter we consider what happens when the proposed distributed localization algorithm is run in two consecutive rounds, and the robots move in between rounds. We argue that by using odometry (i.e., each robot keeps track of how much it moved between rounds) it is possible to recover the scale of the relative coordinates provided by the distributed localization algorithm

**Roadmap.** Section 9.1 introduces the definitions necessary to formalize the distributed localization problem. Section 9.2 contains a detailed description of the proposed distributed localization algorithm. Section 9.3 proves the correctness and optimality claims of the proposed algorithm. Finally, in Section 9.4 we describe how to leverage odometry and the distributed localization algorithm to obtain relative positions with scale.

## 9.1 Problem Statement

To define formally the distributed localization problem we first describe formally the local coordinate system of each robot. Specifically, the local coordinate system of robot $u$ has its position $\varrho_u$ at the origin, and has its heading aligned with the $x$-axis. In other words, in its own local coordinate system, each robots sits at the origin and has orientation 0; therefore to transform from a robot's local coordinate system to the global coordinate system it suffices to perform a rotation followed by a translation (see Figure 9-1). We highlight that since each robot $u$ is not aware of its own position $\varrho_u$ or its own orientation $\phi_u$, it cannot transform between local and global coordinates.

Informally, we say that a robot $u$ has learned the relative orientation and relative position of robot $w$, if robot $u$ knows in its local coordinate system the orientation and position of robot $w$. Formally, for robot $u$ the *relative orientation* and the *relative position* of robot $w$ are defined as $\phi_w^u = \phi_w - \phi_u \pmod{2\pi}$ and $\varrho_w^u = R_{-\phi_u}(\varrho_w - \varrho_u)$, respectively, where $R_\theta$ is a rotation matrix with angle $\theta$ around the origin. (This

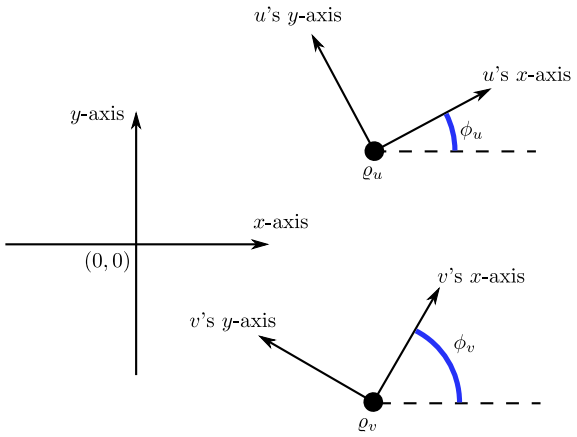



Figure 9-1: Local coordinate system of robots $u$ and $v$.

definition of relative orientation matches our previous Definition 7.1.)

Intuitively, a distributed algorithm solves the localization problem if each robot $u$ outputs the set of relative orientations and relative positions that correspond to the robots that need to be localized by robot $u$. To formalize this intuitive definition we must address some important subtleties. First, recall that in our system model a robot can only send and receive messages to neighbors in the communication graph and measure the angle (relative to their own orientation) of neighboring robots. In this model the execution of an algorithm can depend only on the communication graph $G$ and the angle measurements $\Theta_G$ (recall from Section 7.1 that $\Theta_G$ associates an angle measurement to each directed edge in $\overleftrightarrow{E}_G$). Second, to determine if the output of a distributed localization algorithm is successful, it is necessary to compare the output against the assignment of orientations $\phi$ and positions $\varrho$ to robots. Therefore a formal definition of a distributed localization algorithm must account for the fact that there are multiple assignments of positions and orientations to robots which produce exactly the same angle measurements (and these will be indistinguishable to the algorithm).

For succinctness, we use $\langle \varrho, \varphi, G \rangle$ to denote an environment where $\varrho$ is the assignment of positions to robots, $\varphi$ is the assignment of orientations to robots, and $G$ is the communication graph of the robots.

**Definition 9.1.** *Consider an execution of a distributed localization algorithm $\mathcal{A}$ on $\langle \varrho, \varphi, G \rangle$ where the subset $S_u \subseteq V_G \setminus \{u\}$ occurs as an input to every robot $u \in V_G$. We say:*

1. $\mathcal{A}$ succeeds *at $u$ if its output at $u$ is a set $\{(w, o_w, t_w) \mid w \in S_u\}$ where $o_w \in [0, 2\pi)$ and $t_w \in \mathbb{R}^2$ such that for all $w \in S_u$ it holds that $o_w = \phi_w^u$ and*

$t_w = \alpha_u \varrho_w^u$ *for some* $\alpha_u \in \mathbb{R}^+$.

2. $\mathcal{A}$ aborts *at* $u$ *if its output at* $u$ *is* $\perp$.
3. $\mathcal{A}$ fails *at* $u$ *otherwise.*

Observe that each robot recovers, in its local coordinate system, the coordinates of the other robots up to a fixed but unknown scaling factor. Moreover, the coordinates recovered by different robots might have different scales. To have all robots agree on the same scaling factor would require $O(diameter(G))$ communication steps. More importantly, even without agreement on the scaling factor, the information recovered is sufficient to perform a number of multi-robot tasks. Concretely Chapter 10 describes some simple applications using these coordinates.

Since the distributed algorithm we will consider in the next section runs for $k$ communication steps, then for simplicity we will impose the requirement that the subset of robots $S_u$ that occurs as an input at each robot $u$ must be contained in the $k$-neighborhood $u$. (It is not hard to show that it is impossible for a robot at $u$ to learn anything about a robot $w$ outside its $k$-neighborhood, including the relative position or relative orientation of $w$, using $k$ or fewer communication rounds).

Definition 9.1 allows a distributed localization algorithm to succeed at some (but not all) of the robots. The distributed localization algorithm described in the next section succeeds in as many cases as possible and never fails in any environment at any robot.

## 9.2 Algorithm

From a very high level, the distributed algorithm to solve the localization problem we describe in this section has the same basic structure as the distributed algorithm for the edge-selection problem described in Chapter 3. Namely, each robot first runs a data collection phase for $k$ communication rounds, and then applies the centralized procedures of Chapter 7 and Chapter 8 on the data collected by each robot to produce the desired output. The pseudo-code is presented in Algorithm 16.

In more detail, to collect the maximum amount of information, first each robot runs a full-information protocol for $k$ communication steps. This allows each robot $u$ to learn its $k$-neighborhood $G^k[u]$, which is by definition the largest subgraph of $G$ which can be learned by robot $u$ after $k$ communication steps. In addition, robot $u$ also learns all the angle measurements associated with the directed edges in $\overleftrightarrow{E}_{G^k[u]}$, denoted $\Theta_{G^k[u]}$.

An input at each robot $u$ provides a subset $S_u$ of robots in the $k$-neighborhood of $u$ to be localized. Robot $u$ then runs the centralized procedure ComputeAn-

**Algorithm 16** SUBSETLOCALIZE$_k$ at robot $u$

$G' \leftarrow$ empty graph
$\Theta' \leftarrow$ empty angle measurements
**for** $k$ communication steps **do**
    run full-information protocol
    update $G'$ and $\Theta'$
Let $S$ in $V_{G'}$ be the input that occurs at robot $u$
$H, L \leftarrow$COMPUTEANGLECONSTRAINT $(G', \Theta', u)$
$Q \leftarrow$COMPUTESUBSETREALIZATIONS$(G', L, S \cup \{u\})$
**if** $|Q| = 1$ **then**
    let $p$ be the only realization in $Q$
    **return** $\{(w, H(w), p(w) - p(u)) \mid w \in S\}$
**return** $\perp$

GLECONSTRAINT described in Chapter 7, using as input the graph $G^k[u]$ and the angle measurements $\Theta_{G^k[u]}$. This procedure outputs the relative orientations of all robots in $G^k[u]$ with respect to robot $u$, as well as the angle measurements $\Theta^u_{G^k[u]}$ with respect to robot $u$. Recall from Definition 7.5 that the angle measurements in $\Theta^u_{G^k[u]}$ correspond to the angle measurements that would have resulted if every robot in $G^k[u]$ had $u$'s orientation. Moreover from Definition 7.7 it follows that $\Theta^u_{G^k[u]}$ is an angle-constraint on $G^k[u]$.

Finally, each robot $u$ runs the centralized procedure COMPUTESUBSETREALIZATIONS described in Chapter 8, using as input the graph $G^k[u]$, the angle-constraint $\Theta^u_{G^k[u]}$, and the subset of robots $S_u \cup \{u\}$. The output of this procedure is a set $Q$, and from Theorem 8.25 it contains non-trivial realizations of $S_u \cup \{u\}$, which are restrictions of satisfying realizations of the angle-constrained graph $(G^k[u], \Theta^u_{G^k[u]})$. If $Q$ contains exactly one realization, then robot $u$ uses this realization to compute the relative positions (up to scale) of the robots in $S_u$ after translating the realization to ensure that robot $u$ is at the origin. If $Q$ does not contain exactly one realization then robot $u$ fails to compute the relative orientations and relative positions of the robots in $S_u$ and returns $\perp$.

By construction, it immediately follows that the communication step complexity of the SUBSETLOCALIZE$_k$ algorithm is $k$. The computational complexity is dominated by the computation of a null space basis, used in the COMPUTESUBSETREALIZATIONS procedure. Since the dimensions of the matrix are $2q \times m$ where $q < m$ and $m$ is the number of edges in $G^k[u]$, then if implemented using Gaussian elimination, the null space basis can be computed using $O(m^3)$ arithmetic operations. However,

in practice due to numerical stability issues it is usually preferable to compute the null space basis through costlier methods such as Singular Value Decomposition or QR decomposition [90].

## 9.3 Correctness and Optimality

In this section we prove the correctness of the SubsetLocalize$_k$ distributed algorithm, and we state and prove our optimality claims about the algorithm.

First we show that whenever the algorithm outputs a non-empty set at a robot $u$, it succeeds in localizing the subset of robots input to the algorithm by robot $u$.

**Theorem 9.2.** *Consider an execution of* SubsetLocalize$_k$ *on* $\langle \varrho, \varphi, G \rangle$ *where the subset* $S_u \subseteq V_G \setminus \{u\}$ *occurs as an input at every robot* $u \in V_G$*. If the algorithm does not abort at robot* $u$*, then the algorithm succeeds at robot* $u$*.*

*Proof.* Let $G'$ and $\Theta'$ be the subgraph of $G$ and the associated angle measurements learned by robot $u$ after running the $k$ communication steps of the full-information protocol (i.e., $G' = G^k[u]$ and $\Theta' = \Theta_{G^k[u]}$). Let $S' = S_u \cup \{u\}$ be a subset of robots contained in $G'$.

If the algorithm does not abort at robot $u$, then by definition it does not output $\perp$ at robot $u$ (line 13). Then it follows that the set of realizations $Q$ returned by ComputeSubsetRealizations contains exactly one realization $p$ (line 10). Moreover, the set output by $u$ contains exactly one tuple $(w, H(w), p(w) - p(u))$ for each robot $w \in S_u$ (line 12).

To complete the proof we need only to show that for every $w \in S_u$ it holds that $H(w) = \phi^u_w$ and for some $\alpha_u \in \mathbb{R}^+$ we have $p(w) - p(u) = \alpha_u \varrho^u_w$. The fact that $H(w) = \phi^u_w$ for every $w \in S_u$ follows immediately from Theorem 7.9, thus we need only to show that for every $w \in S_u$ it holds that $p(w) = \alpha_u \varrho^u_w$ for some $\alpha_u \in \mathbb{R}^+$.

From Theorem 7.9 it also follows that $L$ contains the angle-constraint $\Theta^u_{G'}$ on $G'$, and from Definition 7.5 this angle-constraint corresponds to the angle measurements that would result if every robot had $u$'s orientation. Therefore it follows that the realization $p'$ defined as $p'(v) = R_{-\phi_u}(\varrho_v)$ for every $v \in V_{G'}$ satisfies the angle-constraint $\Theta^u_{G'}$. Moreover, since a position assignment never maps two robots to the same position, it follows that $p'$, and therefore any restriction of $p'$, is also a non-trivial realization.

Since the set $Q$ returned by ComputeSubsetRealizations contains exactly one realization by assumption, Theorem 8.25 implies that the angle-constrained graph $(G', \Theta')$ is $S'$-unique. This means that for any realization $p$ that satisfies

$(G', \Theta')$ where $p[S]$ is non-trivial (such as the realization returned in $Q$), $p[S']$ is angle-equivalent to $p'[S']$.

Finally if we let $p$ be the realization returned in $Q$, then by the fact that it is $S$-unique and the definition of angle-equivalence it follows that there exists an $\alpha_u \in \mathbb{R}^+$ and a $t_0 \in \mathbb{R}^2$ such that $p(v) = \alpha_u p'(v) + t_0$ for every $v \in S'$. The rest of the proof follows from some simple algebraic manipulation:

$$\begin{aligned} p(w) - p(u) &= \alpha_u(p'(w) - p'(u)) \\ &= \alpha_u(R_{-\phi_u}\varrho_w - R_{-\phi_u}\varrho_u) \\ &= \alpha_u R_{-\phi_u}(\varrho_w - \varrho_u) \\ &= \alpha_u \varrho_w^u. \end{aligned}$$

□

Theorem 9.2 implies that the SubsetLocalize$_k$ algorithm never fails in any environment at any robot, it only succeeds or aborts. To complement Theorem 9.2, we show that if there is a distributed algorithm that runs in $k$ or fewer communication rounds, that succeeds at robot $u$ in some environment, and does not fail at robot $u$ in other environments, then SubsetLocalize$_k$ also succeeds at robot $u$ in the same environment.

**Theorem 9.3.** *If there is a distributed algorithm that runs for $k$ or fewer communication steps, that succeeds at robot $u$ on $\langle \varrho, \varphi, G \rangle$ and does not fail at robot $u$ in any other environment, then* SubsetLocalize$_k$ *succeeds at robot $u$ on $\langle \varrho, \varphi, G \rangle$.*

*Proof.* Let $G'$ and $\Theta'$ be the subgraph of $G$ and the associated angle measurements learned by robot $u$ after running the $k$ communication steps of the full-information protocol (i.e., $G' = G^k[u]$ and $\Theta' = \Theta_{G^k[u]}$). Let $S' = S_u \cup \{u\}$ be a subset of robots contained in $G'$, where $S_u$ is the set of robots to localize that occurred as an input at robot $u$.

To prove the theorem it suffices to show that if SubsetLocalize$_k$ aborts at robot $u$ on $\langle \varrho, \varphi, G \rangle$, then any other distributed algorithm that runs for $k$ or fewer communication rounds must either fail at $u$ in some enviroment or also abort at $u$ on $\langle \varrho, \varphi, G \rangle$.

Suppose that SubsetLocalize$_k$ aborts at robot $u$ on $\langle \varrho, \varphi, G \rangle$, that is, that it outputs $\perp$ at robot $u$ (line 13). Then it follows that the set of realizations $Q$ returned by ComputeSubsetRealizations does not contain exactly one realization $p$ (line 10). There are two possible cases: the set $Q$ of realizations returned by ComputeSubsetRealizations either (a) contains zero realizations, or (b) contains two or more realizations. We consider these two cases separately.

(a) Theorem 7.9 implies that the $L$ returned by ComputeAngleConstraint contains the angle-constraint $\Theta^u_{G'}$ on $G'$, and from Definition 7.5 this angle-constraint corresponds to the angle measurements that would result if every robot had $u$'s orientation. Therefore it follows that the realization $p'$ defined as $p'(v) = R_{-\phi_u}\varrho_v$ for every $v \in V_{G'}$ satisfies the angle-constraint $\Theta^u_{G'}$.

However, this implies that case (a) cannot occur, since there is at least one realization of $G'$ that satisfies $\Theta^u_{G'}$ and which is non-trivial when restricted to $S'$ (the non-triviality follows from the fact that any robot assignment $\varrho$ will not assign two distinct robots to the same position), and Theorem 8.25 implies this realization is contained in $Q$.

(b) Suppose $Q$ has two realizations. Theorem 8.25 implies there are two realizations $p$ and $p'$ of $G$ that satisfies $\Theta^u_{G'}$ and which are not angle-equivalent, even when restricted to $S'$. This implies there is an environment $\langle \varrho', \varphi, G\rangle$ where $\varrho'$ is not a translation and scaling of $\varrho$, even when restricted to the robots in $S'$. Since these two environments are indistinguishable at $u$ for any algorithm that runs in $k$ or fewer communication steps, then any algorithm that runs for $k$ or fewer communication steps must produce the same output at $u$ in both environments. Therefore if an algorithm that runs in $k$ or fewer communication steps succeeds at robot $u$ in environment $\langle \varrho, \varphi, G\rangle$ then it must fail at robot $u$ in environment $\langle \varrho', \varphi, G\rangle$, which concludes our proof. □

## 9.4 Recovering Scale Through Odometry

The SubsetLocalize$_k$ algorithm described in Section 9.2 uses $k$ communication steps but runs in a single round, and as such it does not require the robots to move. This property makes the algorithm suitable for platforms that are not capable of motion, such as sensor networks. In this section we argue that, in applications where the robots are required to move, it is possible to use odometry (i.e., have each robot keep track of the motion it performs between rounds) to recover the scale (e.g., to have the distance in meters between robots) of the relative positions provided by the proposed localization algorithm.

Concretely, we assume that at each communication round every robot computes the relative orientations and relative positions, up to scale, of a subset of robots in the communication graph (for instance, via the SubsetLocalize$_k$ algorithm). In between rounds each robot moves by following the trajectory prescribed by an arbitrary motion planner, and robots use odometry to keep track of the motion performed. Our goal is then to leverage the information provided by the odometry to recover the scale of the relative positions. Using odometry as a source of "distance"

information allows us to combine our existing localization approach with the standard triangulation-based techniques for localization [27, 68].

As before we use $\varrho_u, \phi_u$ and $N(u)$ to denote the position, orientation and neighbors of robot $u$ at the current round, and in addition, we use $\varrho'_u, \phi'_u$ and $N'(u)$ to denote the position, orientation and neighbors of robot $u$ at the previous round. For convenience at the first round (i.e. when there is no "previous" round) we let $\varrho'_u = \varrho_u, \phi'_u = \phi_u$ and $N'(u) = N(u)$. The *orientation change* at the current round for robot $u$ is denoted by $\mathcal{O}_u = \phi_u - \phi'_u$, and the *translation change* at the current round for robot $u$ is denoted by $\mathcal{T}_u = \varrho_u - \varrho'_u$. Therefore by definition, in the first round the orientation change and translation change of every robot is zero.

Formally, our additional odometry assumption amounts to assuming that at the beginning of each round, every robot $u$ knows its orientation change $\mathcal{O}_u$ and its *local translation change* $\mathcal{LT}_u = R_{-\phi_u}\mathcal{T}_u$. In other words, each robot knows how many degrees it turned, how many meters it moved, and the direction of the movement in its local coordinate system; however it does not know the direction of the movement in the global coordinate system.

This additional knowledge has some minor but important side effects on the SubsetLocalize$_k$ algorithm described in Section 9.2. Namely, after running the full-information protocol for $k$ communication steps, each robot $u$ learns not only its $k$-neighborhood $G^k[u]$ and the associated angle measurements $\Theta_{G^k[u]}$, but also learns the orientation change $\mathcal{O}_v$ and translation change $\mathcal{LT}_v$ for every robot $v$ in its $k$-neighborhood. For convenience in this section we assume that together with the relative orientations and relative positions of the robots being localized, SubsetLocalize$_k$ also returns the local orientation change and the local translation change of the robots being localized.

However, this additional knowledge cannot always be used to recover the scale of the coordinates, for instance, if the robots remain stationary (i.e., $\mathcal{O}_u = 0$ and $\mathcal{T}_u = 0$ for every robot $u$) or all robots perform the same motion at every round (i.e., $\mathcal{O}_u = \mathcal{O}_v$ and $\mathcal{T}_u = \mathcal{T}_v$ for every robot $u$ and $v$). The following definition captures what constitutes a "useful" motion for a pair of robots.

**Definition 9.4.** *Robots $u$ and $v$ have a* useful motion *if the points $\varrho'_u$, $\varrho'_v$ and $\varrho'_u + \mathcal{T}_u - \mathcal{T}_v$ are not collinear.*

To see that this definition is symmetric for $u$ and $v$, observe that the points $\varrho'_u$, $\varrho'_v$ and $\varrho'_u + (\mathcal{T}_u - \mathcal{T}_v)$ are collinear iff the vector $\varrho'_v - \varrho'_u$ is parallel to the vector $\mathcal{T}_u - \mathcal{T}_v$, and this last condition is symmetric for $u$ and $v$.

For two vectors $a, b \in \mathbb{R}^2$ the *two-dimensional cross product* is defined as $a \otimes b = a_x b_y - a_y b_x$. In contrast with the three-dimensional cross product, the two-

dimensional cross product returns a scalar. However, the two-dimensional cross product satisfies a number of properties also satisfied by its three-dimensional counterpart. The next proposition lists a number of properties satisfied by the two-dimensional cross product (see [10] for a more extensive list of properties, together with their proofs).

**Proposition 9.5.** *Let $a, b \in \mathbb{R}^2$ be two-dimensional vectors, let $r \in \mathbb{R}$ be a scalar, and let $R$ be a two-dimensional rotation matrix. Then:*

1. *Anti-commutative: $a \otimes b = -b \otimes a$.*
2. *Distributive over addition: $a \otimes (b + c) = a \times b + a \otimes c$.*
3. *Compatible with scalar multiplication: $(ra) \otimes b = a \otimes (rb) = r(a \otimes b)$.*
4. *Invariant to rotations: $(Ra) \otimes (Rb) = a \otimes b$.*

The following proposition, which appears in [10], provides an efficient way of determining if three points are collinear through the two-dimensional cross product.

**Proposition 9.6.** *The points $a, b, c \in \mathbb{R}^2$ are collinear iff $(b - a) \otimes (c - a) = 0$.*

The information required to recover the scale of the relative coordinates at a robot $u$ is captured by the next definition. Informally speaking, robot $u$ needs two sets of tuples (one for the previous round and the other for the current round), where each tuple contains the identifier of a robot $v$, the relative orientation of robot $v$, the relative position (up to scale) of robot $v$, the orientation change of robot $v$, and the translation change of robot $v$. Specifically, the tuples in the first set have the relative positions scaled by a positive constant $a \in \mathbb{R}^+$, and the tuples in the second set have the relative positions scaled by a positive constant $b \in \mathbb{R}^+$. The goal is to use the information contained in these two sets of tuples, together with the known motion of the robots, to recover both $a$ and $b$.

**Definition 9.7.** *Let $X'$ and $X$ be sets of tuples of the form $(v, o, t, O, T)$ where $v$ is a robot identifier, $o \in [0, 2\pi)$ is an angle, $t \in \mathbb{R}^2$ is a vector, $O \in [0, 2\pi)$ is an angle and $T \in \mathbb{R}^2$ is a vector. For $a, b \in \mathbb{R}^+$ we say $X'$ and $X$ are* $(a, b)$-valid *for robot $u$ at the current round iff the following two conditions hold:*

1. *If $(v, o'_v, t'_v, O'_v, T'_v) \in X'$ then in the* previous *round robot $v$ is such that $o'_v = \phi^u_v$, $t'_v = a\varrho^u_v$, $O'_v = \mathcal{O}_v$, $T'_v = \mathcal{LT}_v$.*
2. *If $(v, o_v, t_v, O_v, T_v) \in X$ then in the* current *round robot $v$ is such that $o_v = \phi^u_v$, $t_v = b\varrho^u_v$, $O_v = \mathcal{O}_v$, $T_v = \mathcal{LT}_v$.*

The main contribution of this section is the RecoverScale procedure. It receives as input two sets of tuples that are $(a, b)$-valid for robot $u$ at the current round, and it returns $(a, b)$ if it succeeds and $\perp$ if it fails.

The idea behind the procedure is straightforward. Robot $u$ searches through $X'$ and $X$ looking for a robot $v$ which is present in both sets, and with which it has a useful motion according to Definition 9.4. If no such robot is found, then robot $u$ fails and returns $\perp$. If such a robot $v$ is found, then robot $u$ uses the relative positions and the translation change of $u$ and $v$ to construct a non-degenerate triangle (the fact that the triangle is non-degenerate follows from the non-collinear requirement of Definition 9.4). Specifically, two sides of the triangle correspond to the relative coordinates of robot $v$ at robot $u$ obtained at the previous and the current round. The lengths of these sides are known up to a positive scalar $a$ and $b$ respectively. The third side of the triangle is the subtraction of the translation change of $v$ from the translation change of $u$; the length of this third side is known exactly. The properties of this triangle are instrumental in our correctness proof, and a diagram of it appears in Figure 9-2. Robot $u$ uses the dot product to compute the inner angles of the triangle, and finally using the sine law it recovers and returns the scaling factors $a$ and $b$. The detailed pseudo-code appears in Algorithm 17.

**Algorithm 17** RecoverScale$(X', X)$ at robot $u$

1: **for each** $(v, o'_v, t'_v, O'_v, T'_v) \in X'$ **do**
2: **for each** $(v, o_v, t_v, O_v, T_v) \in X$ **do**
3: $A \leftarrow (R_{-\mathcal{O}_u} t'_v)$, $B \leftarrow t_v$, $C \leftarrow (\mathcal{LT}_u - R_{o_v} T_v)$
4: **if** $A \otimes C \neq 0$ **then**
5: $\alpha \leftarrow \arccos\left(\frac{B \cdot C}{\|B\|\|C\|}\right)$ $\beta \leftarrow \arccos\left(\frac{A \cdot C}{\|A\|\|C\|}\right)$, $\gamma \leftarrow \arccos\left(\frac{A \cdot B}{\|A\|\|B\|}\right)$
6: **return** $(\|A\| \sin\gamma / \|C\| \sin\alpha, \|B\| \sin\gamma / \|C\| \sin\beta)$
7: **return** $\perp$

The following theorem proves the correctness of this procedure.

**Theorem 9.8.** *Let $X'$ and $X$ be $(a, b)$-valid for robot $u$ at the current round. If there is a robot $v$ in $X$ and $X'$, with which robot $u$ has a useful motion, then running* RecoverScale $(X', X)$ *at robot $u$ returns $(a, b)$, otherwise it returns $\perp$.*

The next auxiliary proposition states a few properties of rotations.

**Proposition 9.9. (1)** *Rotation additivity: $R_x R_y = R_{x+y}$ where $R_x$ is a rotation of angle $x$, $R_y$ is a rotation of angle $y$ and $R_{x+y}$ is a rotation of angle $x + y$.* **(2)** *Dot-Product Rotation Invariance: $(Ra) \cdot (Rb) = a \cdot b$ where $R$ is a rotation and $a, b \in \mathbb{R}^2$ are two-dimensional vectors.* **(3)** *Norm Rotation Invariance: $\|Ra\| = \|a\|$ where $R$ is a rotation and $a \in \mathbb{R}^2$ is a two-dimensional vector.*

*Proof of Theorem 9.8.* We organize the proof as a series of simple claims regarding the values of the variables computed at different steps in the algorithm.

The following three claims prove properties about the values of the variables computed at line 3 in the algorithm.

Claim 1. $A = aR_{-\phi_u}(\varrho'_v - \varrho'_u)$.

*Proof.* By definition $\mathcal{O}_u = \phi_u - \phi'_u$. By the assumption that $X'$ and $X$ are an $(a, b)$-valid input it follows that $t'_v = aR_{-\phi'_u}(\varrho'_v - \varrho'_u)$.

Since by construction we have that $A = R_{-\mathcal{O}_u} t'_v$, then by the additivity of rotations $A = R_{-\phi_u+\phi'_u} aR_{-\phi'_u}(\varrho'_v - \varrho'_u) = aR_{-\phi_u}(\varrho'_v - \varrho'_u)$. □

Claim 2. $B = bR_{-\phi_u}(\varrho_v - \varrho_u)$.

*Proof.* By the assumption that $X'$ and $X$ are an $(a, b)$-valid input it follows that $t_v = bR_{-\phi_u}(\varrho_v - \varrho_u)$. The claim now follows by construction since $B = t_v$. □

Claim 3. $C = R_{-\phi_u}(\mathcal{T}_u - \mathcal{T}_v)$.

*Proof.* By construction $C = \mathcal{LT}_u - R_{o_v} T_v$ and by definition $\mathcal{LT}_u = R_{-\phi_u}\mathcal{T}_u$, thus we need only to show that $R_{o_v} T_v = R_{-\phi_u}\mathcal{T}_v$.

By the assumption that $X'$ and $X$ are an $(a, b)$-valid input it follows that $o_v = \phi^u_v = \phi_v - \phi_u$ and $T_v = R_{-\phi_v}\mathcal{T}_v$, and thus by the additivity of rotations $R_{o_v} T_v = R_{\phi_v-\phi_u} R_{-\phi_v}\mathcal{T}_v = R_{-\phi_u}\mathcal{T}_v$. □

Claim 4. The points $\varrho'_u$, $\varrho'_v$ and $\varrho'_u + \mathcal{T}_u - \mathcal{T}_v$ are collinear iff $A \otimes C = 0$

*Proof.* From Proposition 9.6 it follows that $\varrho'_u$, $\varrho'_v$ and $\varrho'_u + \mathcal{T}_u - \mathcal{T}_v$ are collinear iff $(\varrho'_v - \varrho'_u) \otimes (\mathcal{T}_u - \mathcal{T}_v) = 0$. Since $a \in \mathbb{R}^+$ by assumption, then this is equivalent to $a(\varrho'_v - \varrho'_u) \otimes (\mathcal{T}_u - \mathcal{T}_v) = 0$. From the rotational invariance of the cross product this is equivalent to $(aR_{-\phi_u}(\varrho'_v - \varrho'_u)) \otimes (R_{-\phi_u}(\mathcal{T}_u - \mathcal{T}_v))$. The claim now follows from Claim 1 and 3. □

Observe that by construction of the algorithm and Claim 4 it already follows that if there is a robot $v$ present in $X'$ and $X$, with which robot $u$ has a useful motion, then RecoverScale $(X', X)$ returns a pair (at line 6), otherwise it returns $\perp$.

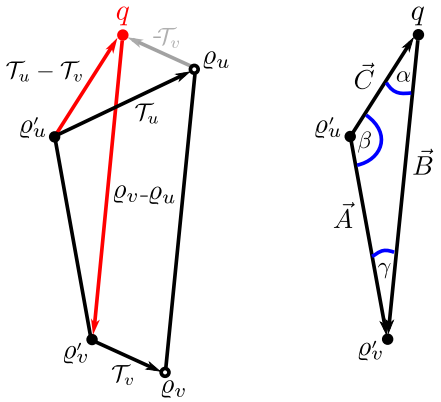



Figure 9-2: Left: The motion vectors of robot $u$ and robot $v$ are denoted with black arrows. The gray arrow denotes the motion vector $-\mathcal{T}_v$. Red arrows denote the vectors $\mathcal{T}_u - \mathcal{T}_v$ and $\varrho_v - \varrho_u$. Right: The inner angles of the triangle formed by $\varrho'_v$, $\varrho'_u$ and $\varrho_{uv}$ are denoted in blue. The vectors outlining the sides of the triangle are labeled $\vec{A}$, $\vec{B}$ and $\vec{C}$, and the opposite angles are labeled $\alpha$, $\beta$ and $\gamma$, respectively.

To prove the theorem it remains only to show that the pair returned (at line 6) corresponds to $(a, b)$. Before we prove the remaining claims we introduce some additional definitions. Namely, let $q = \varrho'_u + \mathcal{T}_u - \mathcal{T}_v$ and consider the triangle between $\varrho'_u$, $\varrho'_v$ and $q$ with sides $\vec{A} = \varrho'_v - \varrho'_u$, $\vec{B} = q - \varrho'_v$ and $\vec{C} = q - \varrho'_u$ (see Figure 9-2).

Observe that Claim 4 implies that this triangle is non-degenerate, and therefore the length of its three sides is non-zero and each of the inner angles of the triangle is greater than zero and smaller than $\pi$.

Claim 5. $A = aR_{-\phi_u}\vec{A}$, $B = bR_{-\phi_u}\vec{B}$ and $C = R_{-\phi_u}\vec{C}$.

*Proof.* From Claim 1 it follows that $A = aR_{-\phi}\vec{A}$.

From Claim 2 it follows that to prove that $B = bR_{-\phi_u}\vec{B}$ we need only to show that $q - \varrho'_v = \varrho_u - \varrho_v$. This follows by unraveling the definitions, $q - \varrho'_v = \varrho'_u + \mathcal{T}_u - \mathcal{T}_v - \varrho'_v = \varrho'_u + \varrho_u - \varrho'_u - \varrho_v + \varrho'_v - \varrho'_v = \varrho_u - \varrho_v$.

From Claim 3 it follows that $C = R_{-\phi}(\mathcal{T}_u - \mathcal{T}_v)$ and by definition $\vec{C} = q - \varrho'_u = \mathcal{T}_u - \mathcal{T}_v$, and thus $C = R_{-\phi}\vec{C}$. □

Claim 6. The inner angles of the triangle opposite to the sides $\vec{A}$, $\vec{B}$ and $\vec{C}$ correspond to $\alpha$, $\beta$ and $\gamma$, respectively.

*Proof.* We prove this claim for $\gamma$, the case for $\alpha$ and $\beta$ is analogous.

Proving that $\gamma$ is the inner angle of the triangle opposite to the side $\vec{C}$ is equivalent to showing that $\gamma$ is the angle between the vectors $\vec{A}$ and $\vec{B}$. By definition of the dot product, this is equivalent to showing that $\gamma = \arccos(\vec{A} \cdot \vec{B} / \left\|\vec{A}\right\| \left\|\vec{B}\right\|)$, this is what we show next.

By construction we have that $\gamma = \arccos(A \cdot B / \|A\| \|B\|)$, and from Claim 5 this implies that $\alpha = \arccos\left((aR_{-\phi_u}\vec{A}) \cdot (bR_{-\phi_u}\vec{B}) / \left\|aR_{-\phi_u}\vec{B}\right\| \left\|bR_{-\phi_u}\vec{B}\right\|\right)$.

From the rotational invariance of the dot product it follows that $\gamma = \arccos\left((a\vec{A}) \cdot (b\vec{B}) / \left\|a\vec{A}\right\| \left\|b\vec{B}\right\|\right) = \arccos\left(ab\vec{A} \cdot \vec{B} / ab \left\|\vec{A}\right\| \left\|\vec{B}\right\|\right) = \arccos(\vec{A} \cdot \vec{B} / \left\|\vec{A}\right\| \left\|\vec{B}\right\|)$. □

We are now ready to conclude our proof. From Claim 6 and the sine law, we have that $\left\|\vec{A}\right\| / \sin\alpha = \left\|\vec{B}\right\| / \sin\beta = \left\|\vec{C}\right\| / \sin\gamma$. Therefore $\left\|\vec{A}\right\| = \left\|\vec{C}\right\| \sin\alpha / \sin\gamma$ and $\left\|\vec{B}\right\| = \left\|\vec{C}\right\| \sin\beta / \sin\gamma$.

From Claim 5 and the rotational invariance of the dot product we have that $\|A\| = a\left\|\vec{A}\right\|$, $\|B\| = b\left\|\vec{B}\right\|$ and $\|C\| = \left\|\vec{C}\right\|$. Thus $a = \|A\| / \left\|\vec{A}\right\| = \|A\| \sin\gamma / \|C\| \sin\alpha$ and $b = \|B\| / \left\|\vec{B}\right\| = \|B\| \sin\gamma / \|C\| \sin\beta$. Therefore the pair returned (at line 6) corresponds to $(a, b)$, which concludes our proof. □

Given the definition of an $(a, b)$-valid input (Definition 9.7), and the definition of the output of a localization algorithm (Definition 9.1), then Theorem 9.2 implies that the successful output of SubsetLocalize$_k$ for the previous round and the current round constitutes an $(a, b)$-valid input for the RecoverScale procedure.

Therefore, as a corollary of Theorem 9.2 and Theorem 9.8 we have the following.

**Theorem 9.10.** *Let $X'$ and $X$ be the output at robot $u$ of successful runs of the* SubsetLocalize$_k$ *algorithm for the previous and the current round, respectively. By definition $\exists a, b \in \mathbb{R}^+$ such that all the relative positions in $X'$ and $X$ are scaled by $a$ and $b$ respectively. If there is a robot $v$ present in $X'$ and $X$ with which robot $u$ has a useful motion then the output of* RecoverScale $(X', X)$ *is the tuple $(a, b)$.*

In Chapter 10 we leverage this theorem to show how various multi-robot motion control tasks can be implemented by combining the RecoverScale procedure, together with the SubsetLocalize$_k$ distributed algorithm, and a task-dependent motion planner.

# Chapter 10

# Applications of Localization

The contributions of Chapter 9 can be summarized by the SubsetLocalize$_k$ distributed algorithm and the RecoverScale procedure, together with their respective correctness proofs. The SubsetLocalize$_k$ distributed algorithm enables robots to compute the relative orientations and relative positions, up to scale, of other robots in the multi-robot system. This algorithm runs in a single round, and requires only that each robot can measure the angle, with respect to its own orientation, towards neighboring robots. The RecoverScale procedure allows a robot to use the information provided by the SubsetLocalize$_k$ algorithm, over the course of two consecutive rounds, to recover relative positions complete with scale. This procedure requires the additional assumption that robots move between rounds and use odometry to track the motion they perform.

The purpose of this chapter is simply to argue that the information provided by these two procedures is sufficient to implement a variety of multi-robot tasks. We provide examples of tasks which do not involve motion (and thus where RecoverScale is not applicable), and of tasks which do involve motion (and therefore where RecoverScale is applicable). For tasks that do not require motion we discuss how the distributed edge-selection algorithms described in Chapter 3 can be implemented using only the information provided by SubsetLocalize$_k$. For tasks that require motion, we argue how SubsetLocalize$_k$ and RecoverScale can be combined to implement motion control algorithms that require the knowledge of the relative positions (for instance the flocking algorithm in Chapter 6).

**Roadmap.** Section 10.1 briefly sketches how the information obtained by the SubsetLocalize$_k$ distributed algorithm can be used to implement the edge-selection procedures described in Chapter 3. Section 10.2 describes how to combine the infor-

mation provided by the SubsetLocalize$_k$ distributed algorithm and the RecoverScale procedure to implement any distributed motion control algorithm that requires the relative positions of its neighbors.

## 10.1 Localization for Static Applications

This section describes how to use the information provided by the distributed localization algorithm of Chapter 9, which is correct only up to scale, to implement some multi-robot tasks that do not require motion. Concretely, we describe how this information can be used to implement the distributed edge-selection algorithms which were described in Chapter 3.

Paraphrasing the formal definitions in Section 3.1, a subset of edges of a connected graph $G$ is a *connectivity-preserving set of edges* if it contains a connected subgraph that spans $G$. An *edge-selection algorithm* is a distributed algorithm in which each robot locally selects a subset of its neighbors. An edge $\{u, v\}$ is *consistently selected* by an edge-selection algorithm if the robot at $u$ locally selects the robot at $v$ and vice versa.

Chapter 3 described various distributed edge-selection algorithms that consistently select a small connectivity-preserving set of edges of the communication graph. Specifically Chapter 3 described edge-selection algorithms that consistently select the Gabriel graph, the Relative Neighbor graph, the Cone-Based Topology Control graph, and the Local Minimum Spanning graph.

All the edge-selection algorithms in Chapter 3 were described for a model where the following two conditions are satisfied: (a) the underlying communication graph is a unit disk graph, and (b) each robot is knows its own position in a global coordinate system. We argue that the second requirement can be replaced by endowing each robot with the capability of computing the relative coordinates, up to scale, of its neighbors.

The edge-selection algorithms described in Chapter 3 share the same outline. First each robot broadcasts its own position and receives the position of its neighbors in the communication graph. Next, to determine which of its neighbors to locally select, each robot evaluates a predicate that depends on the positions of itself and its neighbors. Algorithm 18 describes this with pseudo-code.

**Algorithm 18** GenericEdgeSelect at robot $u$.

1: broadcast $(u, \varrho_u)$, and let $X = \{(u, \varrho_u)\} \cup \{(v, \varrho_v) \mid (v, \varrho_v)$ was received$\}$
2: locally select $\{v \mid v \neq u$ and predicate$(u, v, X)\}$

For instance, in the edge-selection algorithm that consistently selects the edges in the Gabriel graph, each robot $u$ locally selects its neighboring robot $v \in N(u)$ if and only if there is no other neighboring robot $w \in N(u)$ which lies inside the closed disk with the line segment $seg(\mathbf{p}, \mathbf{q})$ as its diameter, where $\mathbf{p}$ and $\mathbf{q}$ are the position of robot $u$ and robot $v$ respectively. Similarly, in the edge-selection algorithm to consistently select the Relative Neighbor graph, each robot $u$ locally selects its neighboring robot $v \in N(u)$ if and only if there is no other neighboring robot $w \in N(u)$ which lies inside the lens produced by the intersection of two open disks of radius $\|\mathbf{p} - \mathbf{q}\|$ centered at $\mathbf{p}$ and $\mathbf{q}$, where $\mathbf{p}$ and $\mathbf{q}$ are the positions of robot $u$ and robot $v$ respectively.

It is not hard to verify that the previous two predicates, as well as the predicates used by the other edge-selection algorithms described in Chapter 3, are invariant to applying a translation, a rotation and a uniform-scaling to the positions of the robots. In other words, one of the predicates used for edge-selection evaluates to true when applied to the actual positions of the robots if and only if the same predicate evaluates to to true when applied to the positions of the robots after having applied a translation, a rotation and a uniform scaling. We can leverage this fact to implement the edge-selection algorithms described in Chapter 3 in a system where robots do not have access to their absolute position, but instead can compute the relative coordinates, up to scale, of its neighbors.

Concretely, Algorithm 19 presents the pseudo-code of an edge-selection algorithm to select the Gabriel graph. Instead of having each robot broadcast its own position and receive the position of its neighbors, each robot relies on SubsetLocalize$_k$ to compute the relative coordinates, up to scale, of neighboring robots. Next, each robot applies exactly the same predicate for Gabriel graphs described in Chapter 3, but using as input the relative positions of its neighbors (as opposed to the absolute positions). Since the predicate is invariant to rotations, translations and positive uniform-scalings of the positions, it follows that this edge-selection algorithm consistently selects the edges in the Gabriel graph.

**Algorithm 19** ScaleGabrielGraphEdgeSelect at robot $u$.

1: $X \leftarrow$ SubsetLocalize$_k$ using neighbors as input subset
2: locally select $\{v \mid v \neq u$ and $(u, o_u, t_u), (v, o_v, t_v) \in X$ and $\forall (w, o_w, t_w) \in X$
3: where $w \notin \{u, v\}$ then $t_w$ is not in the circle with $seg(t_u, t_v)$ as its diameter$\}$

The remaining edge-selection algorithms in Chapter 3 can be implemented in a similar fashion, changing only the predicate to locally select neighbors as appropriate.

## 10.2 Localization for Motion Control Applications

This section describes how multi-robot tasks that involve motion and where each robot requires the relative positions can be implemented by combining the information provided by the SubsetLocalize$_k$ distributed algorithm with the RecoverScale procedure and a motion planner.

Concretely, let $\mathcal{A}$ represent a multi-robot distributed algorithm that controls the motion of the robots and that requires that each robot knows the relative orientations and relative positions (complete with scale) of its neighbors. The purpose of this section is to describe how the SubsetLocalize$_k$ algorithm can be combined with the RecoverScale procedure to allow robots to run $\mathcal{A}$ in a model where each robot can only track its own motion and measure the angle, relative to its own orientation, to its neighbors.

In what follows we describe the ScaledLocalization distributed algorithm, which implements $\mathcal{A}$ at each robot using the information by the SubsetLocalize$_k$ algorithm and the RecoverScale procedure. The key insight behind ScaledLocalization, is that once the relative coordinates have been computed (complete with scale) using the RecoverScale procedure, in subsequent rounds that information can be used together with the odometry information to compute the scale of the relative coordinates returned by the SubsetLocalize$_k$ algorithm, without resorting again to the RecoverScale procedure.

The high level outline of the ScaledLocalization distributed algorithm is straightforward. During the first two rounds every robot goes through a bootstrapping phase that uses the SubsetLocalize$_k$ distributed algorithm and the RecoverScale procedure to obtain the scale of the relative positions. In subsequent rounds, the motion information and the relative positions (complete with scale) of the previous round are used to compute the scale of the relative positions of the current round without resorting to the RecoverScale procedure.

In more detail, in the first and second round each robot $u$ uses the SubsetLocalize$_k$ algorithm to obtain the relative orientations, the relative positions (up to scale), the orientation change and the translation change of each of its neighbors. Additionally each robot moves in a arbitrary direction between the first and second round. Each robot uses the information collected by the SubsetLocalize$_k$ algorithm in these two rounds, together with the RecoverScale procedure, to obtain the relative positions (complete with scale) of its neighbors.

From round three onwards, each robot starts by using the SubsetLocalize$_k$ algorithm to localize (up to scale) each of its neighbors. Next, each robot $u$ searches for a neighboring robot $v$ which was also its neighbor in the previous round. If such

a robot $v$ is found, its current relative position (with scale) is computed by updating the previous known relative position of $v$ (computed in the previous round) with the orientation and translation change of $u$ and $v$. Since all the relative positions returned by SubsetLocalize$_k$ at robot $u$ are scaled by the same positive constant, then robot $u$ uses the computed relative position of robot $v$ to update the relative positions of the rest of its neighbors to have the correct scale. Finally, each robot $u$ uses the relative positions (with scale) of its neighbors to run the $\mathcal{A}$ algorithm to determine its position for the next round.

The pseudo-code for the ScaledLocalization distributed algorithm appears in Algorithm 20. The pseudo-code assumes that the contents of the variable $X$ is stored in the state of each robot and carried over from one round to the next.

**Algorithm 20** ScaledLocalization at robot $u$

▷ Bootstrap Phase

$X \leftarrow$ SubsetLocalize$_k$ using neighbors as input
move in a arbitrary direction
$X' \leftarrow X$, $X \leftarrow$ SubsetLocalize$_k$ using neighbors as input
$scale \leftarrow$ RecoverScale$(X', X)$
**if** $scale \neq \perp$ **then**
  let $(a, b) \leftarrow scale$
  multiply the relative positions in $X'$ by $1/a$
  multiply the relative positions in $X$ by $1/b$
  run $\mathcal{A}$ using the relative positions in $X$
**else**
  fail

▷ Subsequent Rounds

$X' \leftarrow X$, $X \leftarrow$ SubsetLocalize$_k$ using neighbors as input
**if** $\exists (v, o'_v, t'_v, O'_v, T'_v) \in X', (v, o_v, t_v, O_v, T_v) \in X$ **then**
  let $(v, o'_v, t'_v, O'_v, T'_v) \in X'$, $(v, o_v, t_v, O_v, T_v) \in X$
  $\alpha \leftarrow \|t_v\| \,/\, \|t'_v - (\mathcal{L}\mathcal{T}_u - R_{-O_v} T_v)\|$
  multiply the relative positions in $X$ by $1/\alpha$
  run $\mathcal{A}$ using the relative positions in $X$
**else**
  fail

If in the bootstrap phase there is a useful motion for robot $u$ and one of its neighbors then Theorem 9.10 implies that the ScaledLocalization algorithm correctly implements $\mathcal{A}$ at robot $u$. For subsequent, as long as there is a neighbor of $u$ which

was also a neighbor of $u$ in the previous rounds, then the fact that SubsetLocalize$_k$ uses the same positive scaling factor for all the relative positions implies that ScaledLocalization algorithm correctly implements $\mathcal{A}$ at a robot $u$.

We highlight that there are a large number of distributed algorithms that can be implemented using the aforementioned scheme. For example, the flocking algorithm described in Chapter 6 can be implemented using only relative positions. Another example is the distributed coverage control algorithm proposed in [22].

Informally, the goal of coverage control is to control the position of the robots within the environment to optimize the "performance" of a particular task. The meaning of performance depends on the specific task being considered. The uses of coverage control include environmental monitoring and clean up, automatic reconnaissance of buildings, monitoring human activity, providing wireless coverage to soldiers in a battlefield, and a wide variety of applications that can be cast as a location optimization problem. It has been shown [22] that, under reasonable assumptions on the communication radius of the robots, if each robot knows the relative positions of its neighbors in the communication graph then it is possible to implement standard coverage control algorithms (such as Lloyd's [58] algorithm).

# Chapter 11

# Conclusion

To conclude this thesis we present a brief summary of our results, we discuss their implications, and describe some future work and open problems suggested by this thesis. We treat the two parts of the thesis separately.

## 11.1 Connectivity

The first part of this thesis considered the problem of preserving a connected communication graph while allowing the robots to execute an arbitrary motion control algorithm. The following subsections outline our contributions and possible future work.

### 11.1.1 Summary of Contributions

Chapter 3 suggested various edge-selection distributed algorithms to select a spanning subgraph of the communication graph, such that if the communication graph is connected then the selected spanning subgraph is also connected. In particular, we proposed the Local Minimum Spanning Graph algorithm, and proved that this algorithm is optimal in the sense that no distributed algorithm with the same run time that selects a connected spanning subgraph of the communication graph can select a spanning subgraph with fewer edges.

Chapter 4 described a distributed connectivity-preserving algorithm. At each round an arbitrary motion planner produces a set of desired trajectories, one trajectory at each robot. These trajectories are used as an input to the connectivity-preserving algorithm, which outputs at each robot a trajectory that is guaranteed to preserve a selected spanning subgraph (to select the spanning spanning subgraph

we can use one of the edge-selection algorithms described in Chapter 3, but other implementations are possible).

We proved that our connectivity-preserving algorithm guarantees the communication graph remains connected regardless of the input trajectories and irrespective of the speed used by each robot to follow the output trajectory (even if each robot travels only a fraction of its prescribed trajectory). We also showed that under reasonable assumptions the algorithm guarantees that collectively the robots move closer to their desired positions.

Chapter 5 extended our connectivity-preserving algorithm to preserve a $k$-connected communication graph. Specifically, we showed how to leverage the algorithm described in Chapter 4 to preserve a $k$-connected communication graph by simulating a communication radius of $r/k$ instead of $r$ (i.e., by trying to keep neighboring robots within distance $r/k$ instead of distance $r$).

Chapter 6 described how to use the connectivity-preserving algorithm to facilitate the design of higher level multi-robot behaviors. Specifically we considered flocking which, informally, is the behavior of a collection of agents with no central coordinate that move as cohesively despite having no common a priori sense of direction. In particular, we described how standard averaging procedures can be combined with our connectivity-preserving algorithm to yield a provably correct flocking behavior.

### 11.1.2 Future Work and Open Questions

**Using Edge-Selection for Topology Control.** We believe that the edge-selection procedures described in Chapter 3 could be used for other purposes other than our connectivity-preserving algorithm. For instance, one immediate application could be to the field of topology control. Topology control is a technique used in wireless networks to save energy by reducing the number of active links in the network [19, 5]. It is not unreasonable to expect that, at least for reasonable power models, our optimality results on the Local Minimum Spanning graph as an edge-selection procedure could be used to design optimal topology control algorithms. An in-depth investigation of the details of these applications is future work.

**Preserving Connectivity in Three Dimensions.** The connectivity-preserving algorithm described in Chapter 4 assumes the robots are in the Euclidean plane and the communication graph is a unit disk graph of radius $r$. However, the algorithm can easily be generalized to consider robots in three-dimensional space, where the communication graph is a "unit ball graph" of radius $r$ (i.e., two robots are connected iff they are distance less or equal than $r$). The proof of the safety properties of the

original algorithm apply to its three-dimensional generalization. However, the proofs for the progress properties do not hold in this three-dimensional setting. Specifically, it remains as an open question weather the weak progress property holds in the three-dimensional version of the problem.

**Preserving Connectivity with Noisy Sensors.** An interesting direction for future work, would be to study the connectivity-preserving problem in a model where the localization information available to the robots is noisy. We believe that it is possible to modify the connectivity-preserving algorithm proposed Chapter 4 to adopt more conservative trajectories and reserve connectivity in this setting. Working out the details of the required modifications and the implications on the correctness of the algorithm remains as future work.

**Optimal $k$-Connected Edge-Selection.** A natural extension to the $k$-connectivity-preserving algorithm described in Chapter 5 would be to study separately the problem of selecting a "small" spanning subgraph which is $k$-connected. We describe some preliminary results in this direction in [20], but the optimality of the edge-selection procedures for $k$-connected subgraphs still remains as an open question.

**Repairing ($k$-)Connectivity.** A question which arises when studying the problem of preserving the connectivity of a communication graph in a multi-robot swarm, is that of *repairing* the connectivity or $k$-connectivity of the communication graph. In particular we would like to design distributed algorithms that control the motion of the robots so as to repair poorly connected areas of the communication graph while minimizing the energy spent (or the motion required). Different versions of this problem have already been studied in the literature [17, 4]. While in general optimal solutions to this problem are NP-hard even in a centralized setting, we believe that it is possible to provide distributed solutions that approximate the optimal solution.

## 11.2 Localization

The second part of this thesis studied the problem of computing the relative orientations and the relative positions (up to scale) of an arbitrary subset of robots in a system where each robot is equipped with only with a sensor that measures the angle, relative to its own orientation, to neighboring robots in the communication graph. In the following two subsections we outline our contributions and describe future work.

### 11.2.1 Summary of Contributions

Chapter 7 described a centralized procedure that given a subgraph of the communication graph together with the angle measurements associated with the robots in that subgraph, can be used to compute the relative orientations of all the robots in the subgraph. This procedure also allows a robot to "translate" the angle measurements available in the communication graph to its own orientation, thereby defining an *angle-constraint* on the graph.

Chapter 8 provided a precise mathematical characterization of the communication graphs and angle-constraints in which it is possible to compute the relative positions (up to scale) of every robot in the system. In the same chapter we generalized this characterization to the case of computing the relative positions of a specific subset of robots in the system (even if it is not possible to compute them for all robots). The key of this characterization is leveraging the satisfiability restrictions encoded by the cycles in the communication graph.

Chapter 9 leveraged the results of Chapter 7 and Chapter 8 to design a distributed localization algorithm which runs in $k$ communication steps, and efficiently computes the relative positions and relative orientations of an arbitrary subset of robots in the system. We proved that this algorithm is optimal in the sense that no other distributed localization algorithm that runs in $k$ communication steps can succeed in a setting where the proposed algorithm fails. For the case when odometry information is available, we described how standard triangulation procedures can be used to recover the relative positions, complete with scale, of an arbitrary subset of robots.

Chapter 10 argued that the information provided by the proposed distributed localization algorithm is sufficient to implement a variety of multi-robot tasks. In particular we discussed how the distributed edge-selection algorithms described in Chapter 3 can be implemented using only relative orientations and relative positions (up to scale). For tasks that require motion, we described how the extra information provided by odometry can be used to implement any task that requires relative orientations and relative positions (for instance, the flocking algorithm in Chapter 6).

### 11.2.2 Future Work and Open Questions

**Localization in Higher Dimensions.** All our results concerning the localization problem deal exclusively with two dimensional Euclidean space. However, we believe there is no fundamental problem that prevents our techniques from being extended to higher dimensional spaces. For example, a point in three-dimensional space can be described in spherical coordinates by a distance $\ell$ and two angles $\theta \in [0, \pi)$ and $\phi \in [0, 2\pi)$. Therefore in three-dimensional space an angle measurement between two

robots would be a pair $(\theta, \phi)$ of angles. Accordingly, we could define an invertible function $\psi(\theta, \phi) = \begin{bmatrix} \sin\theta\cos\phi & \sin\theta\sin\phi & \cos\theta \end{bmatrix}^T$ that maps an angle measurement to a point in the unit sphere. The rest of our definitions and results can be extended naturally to three-dimensions. As future work, we would like to explore the consequences of the generalization of our results to higher dimensional spaces.

**Localization with Imperfect Sensors.** To solve the localization problem we assume each robot has available simple angle measurement sensors that provide only a minimal amount of information about the configuration. Nevertheless, we assume that the information that is provided by these sensors is error-free, an assumption that cannot hold in practice. It remains an open problem to study the robustness of our proposed methods to noisy angle measurements. In what follows we discuss specific techniques that we believe could be used applied in different components of the algorithm to mitigate the impact of noisy sensor measurements.

**Rooted Trees and Relative Orientations.** The procedure used to compute the relative orientations of the robots relies on constructing a rooted spanning tree of the communication graph and then the relative orientation of different robots is computed by exploring iteratively the spanning tree, starting at the root and concluding at the leafs. When the angle measurements are error-free then any rooted spanning tree will produce the same relative orientations. However, when dealing with noisy angle measurements the choice of the spanning tree used will affect the precision of the resulting relative orientations. In particular, the number of angle measurements used to compute the relative orientation of a robot $u$ depends on the length of the directed path in the spanning tree from the root robot to robot $u$. Therefore, by using a breadth-first search tree we can minimize the number of angle-measurements used to compute the relative orientations of each robot.

**Cycle Bases and Relative Positions.** The procedure used to compute the relative positions of the robots relies on first finding a cycle basis of the communication graph. In a setting with perfect angle measurements the choice of cycle basis is irrelevant, since they all produce the same relative positions. An interesting open problem is to define a suitable noise model (for instance, additive Gaussian noise) and then compute the cycle basis that minimizes the error of the resulting relative positions.

Another key step in the process of computing relative positions is computing the null space basis of a matrix. Noisy measurements could easily result in a null

space basis is of dimension zero. Fortunately, most of the existing techniques used to compute the null space basis of a matrix (including Singular Value Decomposition) are already robust to errors. However, it remains an open question to analyze the properties of the resulting relative positions computed using existing techniques with respect to the amount of noise in the angle measurements.

## 11.3 Other Future Work

Ultimately our goal is to construct a toolbox of distributed algorithms that facilitate the task of designing high-level behaviors on multi-robot systems. In this vein, other problems that we would like to look at include distributed task allocation, distributed state estimation and distributed environment exploration. The challenge here is two-fold; to provide formal problem definitions which have a general enough interface to allow us to compose solutions to different problems to design high-level multi-robot behaviors, and to describe distributed algorithms that solve these problems. Some of these challenges have been described in more detail in [66].

# Index

# Bibliography


[1] H. Akcan, V. Kriakov, H. Bronnimann, and A. Delis. GPS-Free node localization in mobile wireless sensor networks. In *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, pages 35–42. ACM, 2006. ISBN 1595934367.

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

[3] L. Asimow and B. Roth. The rigidity of graphs, II. *Journal of Mathematical Analysis and Applications*, 68(1):171–190, 1979.

[4] N. Atay and B. Bayazit. Mobile wireless sensor network connectivity repair with k-redundancy. *Algorithmic Foundation of Robotics VIII*, pages 35–49, 2009.

[5] P. Bahl, J. Y. Halpern, Li L., Y-M. Wang, and R. Wattenhofer. Analysis of a Cone-Based Distributed Topology Control Algorithm for Wireless Multi-hop Networks. *Principles of Distributed Computing*, pages 264–273, 2001. doi: {10.1145/383962.384043}.

[6] A. Basu, J. Gao, J.S.B. Mitchell, and G. Sabhnani. Distributed localization using noisy distance and angle information. In *Proc. 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 262–273, 2006.

[7] P. Basu, J. Redi, and V. Shurbanov. Coordinated flocking of uavs for improved connectivity of mobile ground nodes. In *Military Communications Conference (MILICOM)*, volume 3, pages 1628–1634, 2004.

[8] D.P. Bertsekas and J.N. Tsitsiklis. Parallel and distributed computation. 1989.

[9] D.P. Bertsekas and J.N. Tsitsiklis. Comments on âĂIJcoordination of groups of mobile autonomous agents using nearest neighbor rulesâĂİ. *IEE Transactions on Automatic Control*, 52(5):968–969, 2007.

[10] C. A. Bishop. Using vector products in two dimensions. *The Mathematical Gazette*, pages 296–298, 1978.

[11] Y.U. Cao, A.S. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous robots*, 4(1):7–27, 1997.

[12] S. Capkun, M. Hamdi, and J.P. Hubaux. GPS-free positioning in mobile ad hoc networks. *Cluster Computing*, 5(2):157–167, 2002. ISSN 1386-7857.

[13] S. Carpin and L. E. Parker. Cooperative Leader Following in a Distributed Multi-Robot System. *ICRA*, 2002.

[14] H.S.M. Coexeter. *Introduction to Geometry, 2nd Edition*. Wiley, Boston, MA, 1989.

[15] R. Cohen and D. Peleg. Robot convergence via center-of-gravity algorithms. *Structural Information and Communication Complexity*, pages 79–88, 2004.

[16] R. Connelly. Rigidity. *Handbook of Convex Geometry, volume A*, pages 223–271, 1993.

[17] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3602–3608. IEEE, 2004.

[18] Alejandro Cornejo and Nancy Lynch. Connectivity Service for Mobile Ad-Hoc Networks. *IEEE Self-Adaptive and Self-Organizing Systems (SASO 08): Spatial Computing Workshop*, 2008.

[19] Alejandro Cornejo and Nancy Lynch. Minimum Spanning Trees and Cone-Based Topology Control. *Proc. of the 28th ACM Symposium on Principles of Distributed Computing (PODC 2009)*, 2009.

[20] Alejandro Cornejo and Nancy Lynch. Reliably Detecting Connectivity Using Local Graph Traits. *International Conference On Principles Of Distributed Systems (OPODIS 2010)*, 2010.

[21] Alejandro Cornejo, Ruy Ley-Wild, Fabian Kuhn, and Nancy Lynch. Keeping Mobile Robot Swarms Connected. *23rd International Symposium on Distributed Computing (DISC 2009)*, 2009.

[22] Jorge Cortes, Sonia Martinez, and Francesco Bullo. Spatially-distributed coverage optimization and control with limited-range interactions. *ESAIM: Control, Optimisation and Calculus Variations*, 2004.

[23] H. Crapo. Structural rigidity. *Structural Topology*, 1(1), 1979.

[24] A. Czumaj and H. Zhao. Fault-tolerant geometric spanners. *Discrete and Computational Geometry*, 32(2):207–230, 2004.

[25] M.H. DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, pages 118–121, 1974.

[26] G. A. Dirac. Some theorems on abstract graphs. *Proc. London Mathematical Society*, 2, 1952.

[27] K. Dogancay. Bearings-only target localization using total least squares. *Signal processing*, 85(9):1695–1710, 2005.

[28] Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 1983.

[29] A. Efrima and D. Peleg. Algorithms for partitioning swarms of autonomous mobile robots. Technical report, Technical Report MCS06-08, The weizmann Institute of Science, 2006.

[30] Asaf Efrima and David Peleg. Distributed models and algorithms for mobile robot systems. In *Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science*, pages 70–87. Springer-Verlag, 2007.

[31] M. Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proc. of the 36th Annual Symposium on Theory of Computing (STOC)*, pages 331–340, 2004.

[32] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. *Proc. 5th International Conference on Mobile Computing and Networking (MobiCom)*, 1999.

[33] R. Fierro and A.K. Das. A modular architecture for formation control. *Robot Motion and Control, 2002. RoMoCo '02. Proceedings of the Third International Workshop on*, pages 285–290, Nov. 2002.

[34] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed coordination of a set of autonomous mobile robots. In *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, pages 480–485. IEEE, 2000.

[35] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Gathering of asynchronous robots with limited visibility. *Theor. Comput. Sci.*, 337(1-3):147–168, 2005.

[36] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A Probabilistic Approach to Collaborative Multi-Robot Localization. *Autonomous Robots*, 8, 2000.

[37] T. Fukuda and S. Nakagawa. Dynamically reconfigurable robotic system. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, 1988.

[38] K.R. Gabriel and R.R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278, 1969. doi: {10.2307/2412323}.

[39] Anurag Ganguli, Jorge Cortés, and Francesco Bullo. Multirobot rendezvous with visibility sensors in nonconvex environments. *CoRR*, abs/cs/0611022, 2006.

[40] A.T. Hayes and P. Dormiani-Tabatabaei. Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots. In *ICRA*, 2002.

[41] J. D. Horton. A Polynomial-Time algorithm to find the shortest cycle basis of a graph. *SIAM Journal on Computing*, 1987.

[42] A. Howard, L.E. Parker, and G.S. Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *The International Journal of Robotics Research*, 25(5-6):431, 2006.

[43] iRobot. Swarmbot. *www.irobot.com*, 2002.

[44] R. Iyengar and B. Sikdar. Scalable and distributed GPS free positioning for sensor networks. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 1, pages 338–342. IEEE, 2003. ISBN 0780378024.

[45] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 2003.

[46] A. Jadbabaie, J. Lin, and A.S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. In *IEEE Transactions on Automatic Control*, volume 48, pages 988–1001, 2003.

[47] X. Jia, D. Kim, S. Makki, P.J. Wan, and C.W. Yi. Power assignment for k-connectivity in wireless ad hoc networks. *Journal of Combinatorial Optimization*, 9(2):213–222, 2005.

[48] D. Johnson and D. Maltz. Dynamic source routing in ad-hoc wireless networks. *Computer Communications Review - SIGCOMM*, 1996.

[49] M. Jorgic, N. Goel, K. Kalaichevan, A. Nayak, and I. Stojmenovic. Localized detection of k-connectivity in wireless ad hoc, actuator and sensor networks. *Proc. 16th ICCCN*, 2007.

[50] Geoffrey A. Kandall. Euler's theorem for generalized quadrilaterals. *The College Mathematics Journal*, 3(5):403–404, 2002.

[51] D.K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. The theory of timed i/o automata. *Synthesis Lectures on Distributed Computing Theory*, 1(1):1–137, 2010.

[52] K. Konolige, D. Fox, C. Ortiz, A. Agno, M. Eriksen, B. Limketkai, J. Ko, B. Morisset, D. Schulz, B. Stewart, et al. Centibots: Very large scale distributed robotic teams. *Experimental Robotics IX*, pages 131–140, 2006.

[53] Geunho Lee, Nak Young Chong, and X. Defago. Robust Self-Deployment for a Swarm of Autonomous Mobile Robots with Limited Visibility Range. In *Robot and Human interactive Communication*, 2007.

[54] N. Li and J.C. Hou. FLSS: a fault-tolerant topology control algorithm for wireless networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 275–286. ACM New York, NY, USA, 2004.

[55] N. Li, J. C. Hou, and L. Sha. Design and analysis of an MST-based topology control algorithm. *INFOCOM*, 3:1702–1712, 2003.

[56] Christian Liebchen and Romeo Rizzi. Classes of cycle bases. *Discrete Applied Mathematics*, 155(3):337 – 355, 2007.

[57] N. Linial. Distributive graph algorithms Global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science, 1987.*, pages 331–335, 1987.

[58] S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[59] Andrew Howard Maja, Andrew Howard, Maja J MatariÄĞ, and Gaurav S Sukhatme. An Incremental Self-Deployment Algorithm for Mobile Sensor Networks. *Autonomous Robots, Special Issue on Intelligent Embedded Systems*, 13: 113–126, 2001.

[60] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad-hoc networks. *DIAL-M: Workshop in Discrete Algorithms and Methods for Mobile Computing and Communications*, 2000.

[61] J. McLurkin, A. Lynch, S. Rixner, T. Barr, A. Chou, K. Foster, and S. Bilstein. A Low-Cost Multi-Robot System for Research, Teaching, and Outreach. *Proc. 10th Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2010.

[62] James McLurkin. Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots. Master's thesis, Massachusetts Institute of Technology, Cambridge MA, 2004.

[63] James McLurkin and Erik D. Demaine. A distributed boundary detection algorithm for multi-robot systems. In *International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[64] James McLurkin and Jennifer Smith. Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In *Distributed Autonomous Robotic Systems (DARS)*, pages 399–408. 2007.

[65] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65, 2009.

[66] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2436–2441. IEEE, 2003.

[67] L. Mureau. Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, 2005.

[68] D. Niculescu and B. Nath. Ad hoc positioning system (APS) using AOA. In *Proc. 22nd IEEE Computer and Communications*, volume 3, pages 1734–1743, 2003.

[69] J.M. O'Kane and S.M. LaValle. Localization with limited sensing. *Robotics, IEEE Transactions on*, 23(4):704–716, 2007.

[70] J.M. O'Kane and S.M. LaValle. Comparing the power of robots. *The International Journal of Robotics Research*, 27(1):5–23, 2008.

[71] R. Olfati-Saber. Flocking for multi-agent dynamic systems: Algorithms and theory. In *IEEE Transactions on Automatic Control*, volume 51, pages 401–420, 2006.

[72] L.E. Parker. Current state of the art in distributed autonomous mobile robotics. *Distributed autonomous robotic systems (DARS)*, 2000.

[73] B.L. Partridge. The structure and function of fish schools. In *Scientific American*, volume 246, pages 114–123, 1982.

[74] D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed mst construction. In *Proc. of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 253–261, 1999.

[75] C. Perkins and E. Royer. Ad hoc on-demand distance vector routing. *Workshop on Mobile Computing Systems and Applications*, 1999.

[76] G. Prencipe. Corda: Distributed coordination of a set of autonomous mobile robots. 2002.

[77] A. Regmi, R. Sandoval, R. Byrne, H. Tanner, and CT Abdallah. Experimental Implementation of Flocking Algorithms in Wheeled Mobile Robots. In *American Control Conference, 2005. Proceedings of the 2005*, pages 4917–4922, 2005.

[78] W. Ren and R. W. Beard. Consensus seeking in multiagent systems under dynamically changing topologies. *IEEE Transactions on Automatic Control*, 2005.

[79] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34, 1987.

[80] R. Saber and R. M. Murray. Consensus protocols for networks of dynamic agents. *Proc. of the American Control Conference (ACC)*, 2003.

[81] R. Saber and R. M. Murray. Consensus protocols for networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 2004.

[82] R. Saber, A. Fax, and R. M. Murray. Consensus and cooperation in multi-agent networked systems. *Procedings of IEEE*, 2007.

[83] K. Savla, G. Notarstefano, and F. Bullo. Maintaining limited-range connectivity among second-order agents. *SIAM Journal on Control and Optimization*, 2007.

[84] Samia Souissi, Xavier Défago, and Masafumi Yamashita. Using eventually consistent compasses to gather oblivious mobile robots with limited visibility. In *SSS*, pages 484–500, 2006.

[85] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotic Systems*, 13(3):127–139, 1996.

[86] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation and agreement problems? In *Structure, information and communication complexity: 3rd Colloquium, SIROCCO'96, Certosa di Pontignano, Siena, June 1996: proceedings*, page 313. McGill-Queen's University Press, 1997.

[87] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: formation of geometric patterns. *SIAM J COMPUT*, 28(4):1347–1363, 1999.

[88] R. Thurimella. Sub-linear distributed algorithms for sparse certificates and biconnected components. In *PODC*, pages 28–37. ACM New York, NY, USA, 1995.

[89] G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12(4):261–268, 1980.

[90] L.N. Trefethen and D. Bau. *Numerical linear algebra*. Number 50. Society for Industrial Mathematics, 1997.

[91] J. Tsitsiklis, D. Bertsekas, and M. Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *Transactions on Automatic Control*, 31(9):803–812, 1986.

[92] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet. Novel type of phase transition in a system of self-driven pinproceedingss. In *Physical Review Letters*, volume 75, pages 1226–1229, 1995.

[93] J. Walter, J. Welch, and N. Vaidya. A mutual exclusion algorithm for ad-hoc mobile networks. *Wireless Networks*, 2001.

[94] W. Whiteley. Matroids from Discrete Geometry. *AMS Contemporary Mathematics*, 197:171–312, 1996.

[95] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM (JACM)*, 30(4):735, 1983.

[96] M. M. Zavlanos and G. J. Pappas. Controlling Connectivity of Dynamic Graphs. *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 6388–6393, 2005.