# Multi-Robot Map-Merging-Free Connectivity-Based Positioning and Tethering in Unknown Environments

Somchaya Liemhetcharat and Manuela Veloso

February 16, 2012

**Abstract**

We consider a set of static towers out of communication range of each other, in an environment with no global coordinates. We address the problem of deploying mobile robots, initially not necessarily within range of each other or of the static towers, to serve as gateways to connect the towers. Our robot positioning algorithm consists of a heuristically controlled exploration with sharing of labeled relative positioning information, without the need to merge maps. After connectivity is achieved, we further address the problem of the robots locating and tethering to an agent using only measurements of signal strengths, without any communication between the agent and the team of robots. We contribute a two-step tethering algorithm that uses a data-driven multi-robot RSSI-distance model. We illustrate our connectivity algorithm in simulation and compare the efficiency of different proposed heuristics. We demonstrate the efficacy of the most promising heuristic in a variety of realistic indoor scenarios. We finally present results of the successful performance of our tethering algorithm in simulation and with real robots in an actual office environment.

**Keywords:** Multi-robot control; Multi-robot teamwork; Multi-robot WiFi-based communication

## 1 Introduction

We are interested in planning for multiple distributed robots to achieve a common positioning goal, without the need for map-merging. Concretely, we address the problem of using a set of mobile robots to ensure connectivity between a number of static communication towers sparsely deployed in an unknown environment and not within range of each other (Fig. 1a). The robots are themselves communication nodes and can communicate with the static towers and with one another, when within range. We assume that the robots have no knowledge of the environment, both in terms of the obstacles and the positioning of the static towers. The obstacles, such as walls, interfere with the signal propagation, and pose challenges in terms of modeling the signal propagation. In open space, models of wireless signal decay allow the signal strength to provide a good distance estimate [4], while in the presence of poorly modeled obstacles, signal strength provides multiple distance hypotheses, preventing the use of the network signal strength for localization.

After the positioning goal is accomplished, we are interested in using the deployed robot team to locate and tether to, i.e., follow, an autonomous agent. The agent does not need to communicate with the robot team, but each team member can measure the received signal strength indicator (RSSI) of connections to each other and to the agent. We assume that the agent is initially

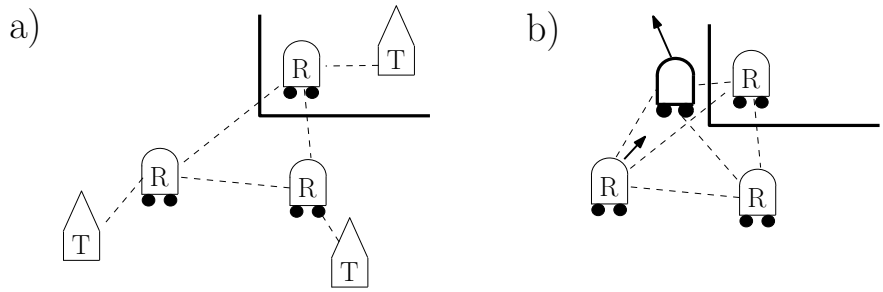



Figure 1: a) 3 mobile robots position themselves to connect 3 static towers. b) An autonomous agent (outlined in bold) enters the environment, and one of the robots tethers to it based on RSSI measurements. Bold lines indicate obstacles, dashed lines indicate wireless connectivity, and arrows indicate movement.

connected to a subset of the team, and remains connected to some member of the team as it moves. The goal is for one of the robots in the team to tether to the autonomous agent as the agent moves independently in the environment (Fig. 1b). In order to do so, based on the subset of robots connected to the target, our solution sets that all but one of the robots in the team become stationary and act as RSSI landmarks, and the remaining mobile robot, i.e., the seeker, tethers to the target. With our multi-robot algorithm, the robots infer the target's location and motion solely based on RSSI data. However, we note that, while we focus on RSSI and infer distance information from it, our algorithm is general and applicable to any form of sensing that provides distance information, even if it is highly noisy, e.g., a microphone that uses volume to estimate distance.

There are several real scenarios that are instances of the general problem we address. For example, emergency teams that need to assist in areas not fully covered with communication towers or where the connectivity is lost because of a disaster, can carry and drop small mobile robots to autonomously navigate and position themselves so that the connectivity is extended in the crisis area. More generally, this problem is not specific to the connectivity goal, and could be extended to other multi-robot positioning needs with other objectives. Once the robot team has been deployed, the robots can then be used to located and tether to a member of the emergency team, or to a victim of the disaster.

We consider the deployment of autonomous mobile robots in the environment with the goal of acting as communication gateways and landmarks. The robot navigation is "driven" by the communication signals. The robots can identify each other and the towers from communicated identifiers. In addition, the robots use only RSSI measurements to located and tether to the autonomous agent, and does not require the agent to communicate to the team, e.g., that it is moving or the direction of its motion.

Furthermore, our approach is targeted to be run on many small, low-cost robots indoors, where global positioning via GPS or wireless triangulation is unavailable. We do not use any assumptions about the nature of signal degradation in the environment, and instead build a data-based model of RSSI. Also, our approach does not require that the robots are homogeneous, or even know about the capabilities of the other robots — we find solutions to the problem readily without planning the full joint-actions of all the robots. A robot will never "instruct" another robot to head to a location that the latter has never visited, and so this ensures that the latter robot has the capabilities to

reach its target.

The organization of our article is as follows: in Section. 3, we describe the problem, our assumptions, and a general overview of our approach and contributions. In Section 4, we explain our algorithm for achieving connectivity and associated data structures in detail, as well as theoretical guarantees. We then discuss and analyze the planning heuristics used by the robots in Section 5. We describe the RSSI-distance model that we generate from real-world data in Section 6 and formally describe the second phase of our algorithm, to locate and tether to the autonomous agent, in Section 7. Sections 8 and 9 illustrate the results of running our algorithm in different scenarios for the connectivity and tethering goals respectively, and we summarize our contributions and discuss future work in Section 10.

## 2 Related Work

Previous work addressed the problem of dispersing a robotic swarm to provide coverage, using wireless signal intensity as a measure of distance between robots, assuming open space between the robots [12]. Robots can position themselves to optimize sensor readings from the environment, using Voronoi graphs [19]. Our goal is to provide connectivity between static towers, using the robots as gateways, and not to maximize coverage of an environment.

By deploying RFID tags as coordination points, robots build a joint map and can coordinate to explore an environment [21]. Similarly, a sensor network can also be deployed after mapping an unknown environment [5]. Our approach does not require any form of map-merging or common global reference frame, or leaving markers in the environment. Instead, the robots use position labels to refer to other robots' positions, without knowing where these positions are in the environment. Also, our goal is not the exploration of an unknown space, but to establish connectivity.

In an environment with unknown obstacles, a robot team that is initially connected can reason about connectivity maintenance, and constrain their mobility in order to avoid disconnecting the network [17]. Similarly, connected robot teams can reason about which links to delete while maintaining connectivity, through distributed consensus and market based auctions [14]. From an initial connected network, robots can achieve biconnectivity, i.e., every robot is connected to at least 2 other robots, to enable robustness in the network if any robot fails [2]. We address the case when mobile robots start from unknown and unconnected positions.

Connectivity of mobile robots can be achieved through coalescence, where robots that are connected coalesce into a cluster and stay connected as they explore the space together [15]. Our goal is to connect static towers using robots as gateways. These robots are capable of exploring the environment, while the towers remain fixed in their locations. In addition, our approach does not require that robots stay connected together; robots can disconnect from other robots and explore independently.

Using the received signal strength indicator (RSSI) to infer distance has been extensively studied. A data-driven approach has be used to fit a straight line between RSSI and $\log_{10}$(distance) in open space [18], and the RSSI to 2 known landmarks can be compared in order to localize a sensor in open space [11]. When the configuration of walls and obstacles are known, a model can be developed that provides an accurate measure of RSSI [3]. We use a data-driven approach to generate our RSSI-distance model, but we do not assume that the environment is open-space. Our model is general and does not require knowledge of the configurations of walls and obstacles in the environment, or any known landmarks.

In addition to RSSI, time-of-flight can be used to measure the distance between receivers and transmitters in a wireless sensor network [9, 16], but requires specialized hardware such as precise clocks and synchronization between sensor motes. Our approach does not require any specialized hardware on the robots; wireless communication hardware typically provides RSSI measurements which we use for locating and tethering to the target.

Localization of a robot can be performed using wireless (WiFi) signal strengths, by developing a WiFi signal map that a robot uses to localize in real time as it receives RSSI information [4, 1]. Similarly, range-only localization can be performed by placing known RFID tags in the environment [8]. We consider the case where the robots have no prior map of the environment, and there are no known static landmarks. Furthermore, the goal is not localization of the robot team, but for the seeker robot to remain tethered to the target.

Connections of a mobile robot team can be modeled as a binary relationship (connected/not connected) with a probabilistic relationship between distance and connectivity [6]. The robots are initially randomly positioned in the world, and the goal of the team is to estimate their relative positions. In our approach, the robots are randomly positioned as well, but we use RSSI to infer distance, and use information from the team of robots in order to locate and tether to a target robot.

A data-driven approach can be used to model RSSI and distance probabilistically, and the goal of the seeker robot is to locate and tether to the target robot [20]. We use a similar data-driven approach, but our model determines the minimum and maximum distance of a pair of robots given their RSSI. In addition, we consider the case where a *team* of robots is used to locate and tether to the target robot, instead of a single seeker robot. Furthermore, we do not assume that the target robot communicates with the team, or synchronizes its direction with the seeker, through a compass or other motions. We infer the motion of the target completely through RSSI observations among the robot team.

In order to track a mobile target, infrared transmitters and receivers can be used to determine the relative distance and bearing from a seeker robot to the target [10]. Range-only measurements can be used for a car-like robot to follow a mobile target [13]. We use only RSSI to obtain an estimate of the distance to the target robot, with no bearing information. Furthermore, we use a team of robots to provide additional information about the target's location and motion, in order for the seeker robot to tether to it effectively.

# 3 Challenges, Assumptions, and Approach

In this section, we formally describe the problem and identify its technical challenges. We present our assumptions and an overview of the solution we contribute.

## 3.1 Problem Statement and Challenges

A team of $n$ autonomous robots are deployed in an unexplored environment containing $m$ static (non-moving) communication towers. The first goal is to find a configuration of robots such that all the towers are connected. In Fig. 2a, towers T1 and T2 are not within direct communication range, and mobile robots R1 and R2 have positioned themselves such that T1 and T2 are connected in the communication network, by using R1 and R2 to relay network packets. Upon establishing a connected network, a subset of $n' \leq n$ of these robots will be used to locate and tether to,

i.e., follow, an autonomous agent (that is not part of the team and does not communicate with the team) that carries a radio whose signal strengths to the teams' radios can be measured.

The environment contains physical obstacles, such as walls, that impede the robots' movement as well as degrade signal propagation. As such, it is difficult for the robots to have an accurate model of signal propagation (due to signal degradation, reflection and interference) that will allow them to obtain an accurate estimate of the distance to the towers or other robots, as an accurate planning state. In Fig. 2b, R1 is connected to towers T1, T2, and T3, with equal signal strengths, due to the degradation of signal propagation in air and through the walls.

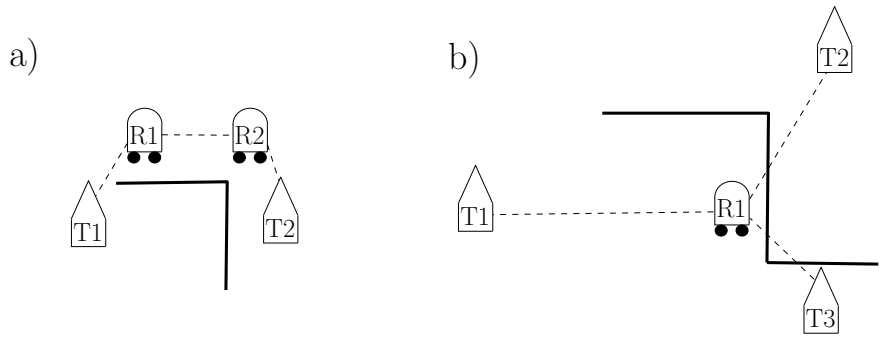



Figure 2: a) Connectivity example with 2 towers (T1 and T2) that are not within communication range, and 2 mobile robots (R1 and R2). b) R1 is connected to 3 towers, with equal signal strengths. Bold lines indicate obstacles (walls) that degrade signal propagation, and dashed lines indicate connectivity.

The robots do not have a map of the world, nor do they possess any form of global positioning. Thus, there is no global coordinate system, and coordinates used by each robot are relative to its starting position and orientation. Hence, robots cannot share coordinates with other robots as they do not have a common meaning.

Robots can only communicate when in range. In addition, other than communicating via network packets and measuring signal strengths, they are incapable of sharing information (e.g., by leaving physical markers in the world).

### 3.2 Assumptions

We list the assumptions of our approach, discuss the implications, and possible ways to overcome the assumptions:

**Assumption 1.** *The number and identification of the towers ($m$) are known.*

The identification of the towers can generally be retrieved via the network protocol. If $m$ is unknown, we can use an iterative deepening approach combined with time-limited exploration at each iteration, to prevent the algorithm from running infinitely.

**Assumption 2.** *The environment is bounded.*

**Assumption 3.** *There exists at least one configuration for $k \leq n$ robots that connects all $m$ towers.*

Assumptions 2 and 3 limit, in a very straightforward way, the amount of exploration that the robots need to entail in order to find a solution. More concretely, Assumption 2 ensures that the space that the robots need to explore in order to find the solution is somehow bounded. In practice, Assumption 2 is not limiting, as the space is limited by the finite number of towers. Assumption 3 is also quite reasonable in that it ensures that there is a solution. This, in turn, implies that the exploration algorithm does not go on forever and eventually terminates, if it is complete. Notice also that we do not require *all* robots to be part of the solution, which means that we do not need to know beforehand how many robots are necessary to attain a solution, as long as we have some upper bound on this number.

**Assumption 4.** *The exploration algorithm for the robots are such that, at any time $t$,*

$$P(\tau_C(t) < \infty) = 1,$$

*where $C$ denotes a general configuration of the robots in the environment and $\tau_C(T)$ is the first return time to $C$ after a given time $T$.*

This assumption states that each robot can revisit any configuration, in a finite amount of time, that may be relevant to find a solution. This assumption is used to guarantee that the relevant network information is passed between the robots and eventually propagates to all robots in the team. In practice, given the relatively large range within which the robots and towers can communicate, the solution configuration can be visited effectively. In general, the exploration algorithm can be designed so that each robot incrementally extends its area of exploration, cyclically returning to the areas already explored.

**Assumption 5.** *Communication is instantaneous, costless and error-free.*

In practice, communication is not instantaneous and is subject to error. Also, robots may come into and out of range of one another as messages are sent, causing messages to be lost. This assumption, however, only affects the efficiency of our algorithm. Our algorithm is completely asynchronous, and can use any communication protocol to make communication more reliable. Thus, we focus on achieving the goal, and abstract our problem from errors in communication. Also, while communication is error-free, this assumption does not exclude the varying of signal strengths of connections when robots are in range.

**Assumption 6.** *The communication devices of the robot team and the autonomous agent is known and identical.*

This assumption states that the characteristics of the radios of the robot team, as well as the autonomous agent that the team has to locate and tether to, are known *a priori*. This assumption allows some information to be obtained from measurements of signal strengths between radios, even if the signal strength measurements are noisy and vary based on the structure of the environment. Our approach is applicable to a relaxed form of this assumption — if the radios are not identical but known, then models of the different radios can be learned before applying our solution.

### 3.3 Overview of Approach and Contributions

In the first phase of the algorithm, the robots explore the environment, and collect information on connectivity as they do so. When robots meet (get within communication range), they share

their information, which allows them to readily find a solution configuration. Once a solution is found, the robots head to their solution positions and provide connectivity to all the towers in the environment, and prepare for the second phase of the algorithm.

The second phase begins when the autonomous agent comes within range of at least 1 of the robots in the team. At that point, the subset of the team that are connected to each other and to the agent begins locating and tethering to the agent. To do so, one of the robots move in a pattern and records signal strength measurements between robots of the team and to the autonomous agent. From these signal strengths, the relative location of the autonomous agent is found, and the robot uses a probabilistic model to track the motion of the agent and tethers to it.

We now describe some important features of our approach, outlining its contributions:

- Instead of sharing coordinates (which is impossible, since there is no global coordinate system and map-merging is not performed), the robots create position labels which they share with each other. A robot can reference another robot's position, without knowing where that position is in the actual environment (Sec. 4.1).

- Instead of sharing and merging maps, the robots build a more effective representation of connectivity - a network graph (Sec. 4.2), and share this information whenever they meet (Sec. 4.3). Merging a network graph involves just the union of vertices and edges. Furthermore, the network graph representation allows sharing of information that can be propagated across the team of robots efficiently.

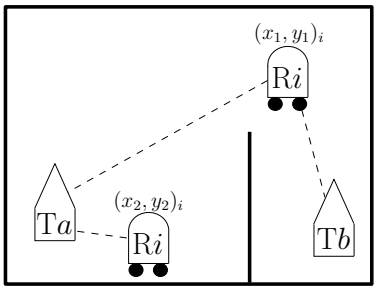
- In this network graph representation, determining whether the goal can be achieved with present knowledge equates to searching the graph for a connected solution, subject to certain constraints on the edges in the network graph. Searching for such a solution is an NP-complete problem, but we contribute a method that reduces the search space and runs efficiently in practice (Sec. 4.4).

- Once a solution is found, each robot simply has to travel to its solution position. Thus, the difficulty of the overall planning problem consists of effectively exploring the space for configurations that can be useful for the connectivity goal.

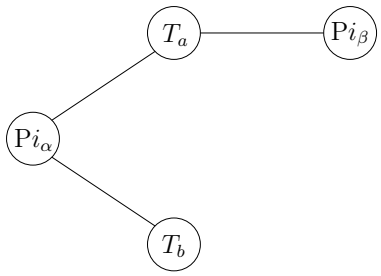
- We propose multiple planning heuristics to perform the exploration (Sec. 5), and present extensive simulation results in representative scenarios (Sec. 8).

- Although the received signal strength indicator (RSSI) data is noisy and varies depending on the environment, we contribute a RSSI-distance model that is learned from real-world data and can be used in new environments with unknown configurations of walls and obstacles (Sec. 6).

- We contribute an algorithm that is capable of locating and tethering to an autonomous agent, using only RSSI measurements between the robot team and the autonomous agent, without requiring the agent to communicate, e.g., that it is moving or the direction of its motion (Sec. 7), and present experiments both in simulation and on real robots in an actual office environment (Sec. 9).




(a) Spatial representation of robot R$i$ in two distinct positions, $(x, y)_i$ and $(x', y')_i$. The bold lines indicate obstacles, and dashed lines represent connectivity.




(b) Graphical representation of the same connections of robot R$i$ using position labels P$i_\alpha$ and P$i_\beta$.

Figure 3: Position labels and graphical representation of connectivity of robot R$i$.

# 4 Distributed Network Connectivity

Let $\mathcal{R} = \{\mathrm{R}1, \ldots, \mathrm{R}n\}$ be the robots deployed in the environment, and $\mathcal{T} = \{\mathrm{T}1, \ldots, \mathrm{T}m\}$ be the static towers.

Each robot R$i$ moves autonomously in the environment and the purpose of the robot team is to find a configuration $C$ such that all the towers in $\mathcal{T}$ are connected, as illustrated in Fig. 2, where a configuration is a vector of positions in the environment, one for each robot. Notice that, in any given environment, multiple such configurations may exist, and we make no requirements as to which one the robots should adopt. The goal is to find *any* such configuration.

## 4.1 Position Labels

In order for the robots to refer to positions in the world without using global coordinates, they use position labels:

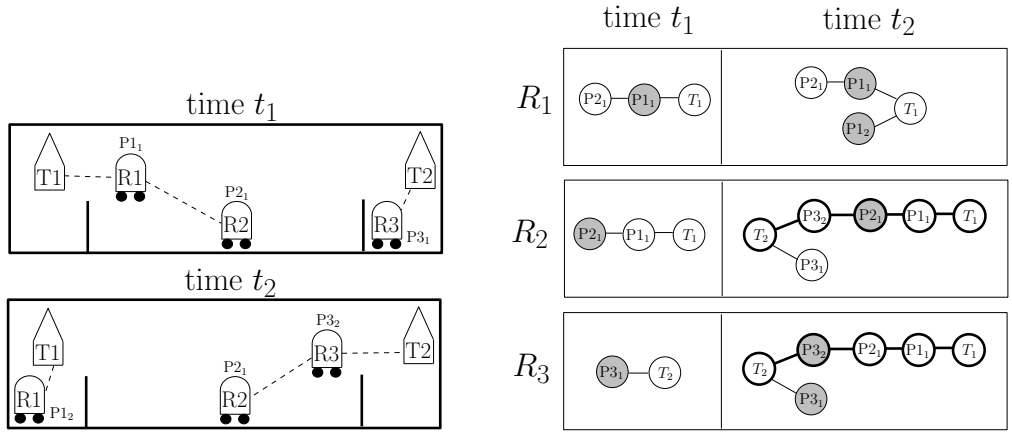
**Definition 4.1.** *Let $Ri \in \mathcal{R}$ be a robot. A **position label** $Pi_\alpha$ is a name that refers to a position (indexed by $\alpha$) of $Ri$.*

We illustrate the use of position labels through an example (see Fig. 3). Suppose that a given robot, R$i$, at some time $t_1$, is at coordinates $(x_1, y_1)_i$, where the subscript $i$ denotes the fact that the coordinates $(x_1, y_1)$ are expressed in terms of R$i$'s reference frame. Let R$i$ be connected to towers T$a$ and T$b$ in this position. At some other time $t_2$, R$i$ is at coordinates $(x_2, y_2)_i$, and is connected only to T$a$. The spatial positions and connections of R$i$ at $t_1$ and $t_2$ are shown in Fig. 3a.

The lack of a global coordinate system prevents robots other than R$i$ to assign any meaning to the coordinates $(x_1, y_1)_i$ and $(x_2, y_2)_i$ and as such, R$i$ assigns a label to each of the two positions, and stores a mapping of the position labels to the coordinates, e.g.,

$$\mathrm{P}i_\alpha = (x_1, y_1)_i; \qquad \mathrm{P}i_\beta = (x_2, y_2)_i$$

Each robot can convert position labels of its own positions into coordinates in its own reference frame, and these position labels can be shared readily among all the robots. For example, when R$i$ meets another robot R$j$, R$i$ can share that it is connected to T$a$ and T$b$ when at position P$i_\alpha$,




(a) Spatial representation of 3 mobile robots and 2 towers in an environment.

(b) Network graphs for the 3 robots shown in Fig. 4a. The shaded vertices indicate positions that the robots can convert into coordinates. Edges between vertices indicate connectivity. The solution found is outlined in bold.

Figure 4: Network graph representation shared between robots to find a solution

and is connected to T$a$ when at position P$i_\beta$. R$j$ does not need to know the coordinates of these positions; it only needs to know that R$i$ is capable of connecting to T$a$ and/or T$b$ at those positions, and that R$i$ can travel to the positions if need be. In particular, this connectivity information can be stored in the form of a graph, as shown in Fig. 3b. R$i$ merely has to share the graph shown in Fig. 3b to allow R$j$ to store the new connectivity information.

### 4.2 Macro Network Graph Representation for Connectivity Information

We developed a data representation, that we call a network graph, which allows robots to store, share and merge connectivity information readily.

**Definition 4.2.** *A **network graph** $G$ is an undirected graph $G = (V, E)$, where each vertex (or node) $v \in V$ is a position label (e.g., $Pi_\alpha$) or a tower (e.g., $Ta$), and each edge $e \in E$ is a pair $\{v_1, v_2\}$, where $v_1, v_2 \in V$. Edges represent connections between vertices (robots/towers) and the weights of the edges represent the signal strengths of the connections.*

To illustrate the usage and benefits of a network graph, consider Fig. 4.

At time $t_1$, the robots R1, R2 and R3 are at positions P1$_1$, P2$_1$, and P3$_1$ respectively. The physical positions of the robots and their connections are shown in Fig. 4a and the network graphs of the robots are shown in Fig. 4b. Note that the robots synchronize and merge their graphs when connected, which is why R1 and R2 have identical graphs.

At time $t_2$, R1 and R3 move to positions P1$_2$ and P3$_2$ respectively; R2 stays in position P2$_1$. At this time, R2 can share information regarding R1 with R3, even though R1 and R3 have never met. This allows both R2 and R3 to discover a solution where R1, R2 and R3 are at positions P1$_1$, P2$_1$, and P3$_2$ respectively. The network graphs of the robots are shown in Fig. 4b, and the solution found is outlined in bold.

The network graph representation offers multiple benefits. First of all, robots can readily share information. When two robots R$i$ and R$j$ meet, they can update their individual network

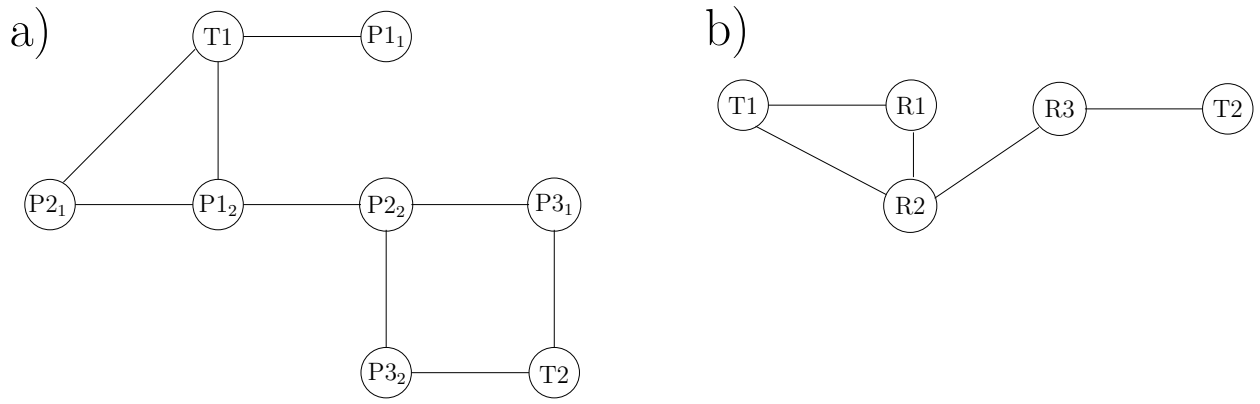



Figure 5: a) A network graph of 3 robots and 2 towers. b) The corresponding macro network graph of the same 3 robots and 2 towers. The position labels are not shown in the macro network graph, even though the information is embedded in the macro nodes.

graphs and unify their knowledge in all parts of the graph, independently of their current position. Furthermore, the updating of graphs is asynchronous in the sense that not all robots need to have the same network representation at all times.

In addition, a configuration that ensures connectivity of all towers can be obtained directly from the graph. Formally, a solution $s$ that connects all towers in a graph $G = (V, E)$ exists iff a sub-graph $G' = (V', E') \subseteq G$ exists such that all towers $\mathrm{T}a \in \mathcal{T}$ are connected, and each robot $\mathrm{R}i$ is in at most one position, i.e., $\forall i\ (\mathrm{P}i_\alpha, \mathrm{P}i_\beta \in V' \Leftrightarrow \alpha = \beta)$.

As the robots explore the environment, they create new position labels, which increases the size of the network graph. The decision of when to create a new position label is based on factors such as the granularity desired in discretizing the environment, and has a direct impact on the rate of growth of the network graph. In this article, we do not discuss when is best to create a position label, and assume that this decision is made by a provided function; in the experiments (described later), we use the discretization of the environment to create new position labels.

The size of the network graph increases as the robots explore the environment, and searching this graph becomes computationally expensive as more position labels as created over time. In order to cope with this growth, we consider a *macro network graph representation*, in which all nodes corresponding to each robot are collapsed into a single macro node. Fig. 5 shows a network graph and its corresponding macro network graph.

**Definition 4.3.** *A **macro node** $\mathbf{v}$ is an equivalence class defined over the set of vertices of the original network graph $G = (V, E)$, that corresponds to a single robot or tower, e.g., the macro node $Ri = \{Pi_\alpha : Pi_\alpha \in V, \forall\alpha\}$.*

**Definition 4.4.** *A **macro edge** $\mathbf{e} = \{\mathbf{v}_1, \mathbf{v}_2\}$ is an equivalence class defined over the set of edges in the original network graph $G = (V, E)$, that corresponds to all connections between the 2 macro nodes (i.e., $\mathbf{v}_1$ and $\mathbf{v}_2$), e.g., the macro edge $\{Ri, Rj\} = \{\{Pi_\alpha, Pj_\beta\} : \{Pi_\alpha, Pj_\beta\} \in E, \forall\alpha, \beta\}$.*

**Definition 4.5.** *A **macro network graph** is an undirected graph $H = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of all macro nodes, and $\mathcal{E}$ is the set of all macro edges.*

A network graph can be represented as a macro network graph, and vice versa. In a macro network graph (we henceforth drop the usage of the word *network* for brevity), $|\mathcal{V}| \leq m + n$, where

$m$ and $n$ are the number of towers and robots respectively, and $|\mathcal{E}| \leq \binom{m+n}{2}$. Any particular macro edge $\{\mathbf{v}_1, \mathbf{v}_2\} \in \mathcal{E}$ means that, in the original network graph, the robots/towers corresponding to nodes $\mathbf{v}_1$ and $\mathbf{v}_2$ share at least one connection. Each macro edge can also be seen as a set of *constraints* on the robots' positions. These constraints limit the possible robot positions in order to have the connection described by the macro edge.

Each macro edge $\mathbf{e} \in \mathcal{E}$ is associated with the corresponding equivalence class or constraint set that must store all edges in the original network graph and corresponding signal strength information. This means, in particular, that the macro graph representation is equivalent to the original network graph representation in terms of space-efficiency. However, the macro graph representation provides significant advantages when searching for a solution, which we will soon show.

We conclude by observing that a solution is a connected subgraph of $H$ that includes all macro nodes corresponding to towers in $\mathcal{T}$, and each robot R$i$ can be in a position P$i_\alpha$ such that all constraints are met in the solution subgraph. Further details are provided in Sec. 4.4.

### 4.3 Communication Phase

As the robots explore the world, they individually maintain a macro graph which they use to store connectivity information. Upon coming within communication range, robots share their corresponding macro graphs and update them to include the information coming from the other robots. This process can be decomposed into several steps which we now describe.

**Theorem 4.6.** *Suppose R*1, . . . , *Rn are connected, with macro graphs* $H_1, \ldots, H_n$ *respectively before the communication phase. After the communication phase, R*1, . . . , *Rn will have the same macro graph* $H$*, such that* $H \supseteq \bigcup_{i \in \{1,\ldots,n\}} H_i$*).*

*Proof.* Let the current positions of the R1, . . . , R$n$ be P$1_{\alpha_1}$, . . . , P$n_{\alpha_n}$.

Let the robots directly connected to robot R$i$ be $\mathcal{R}_i \subset \mathcal{R}$.

Each robot R$i$ first adds constraints to its macro graph $H_i$ (if the constraints do not already exist) regarding its direct connections to all robots in $\mathcal{R}_i$, i.e., $\left\{\text{P}i_{\alpha_i}, \text{P}j_{\alpha_j}\right\}, \forall \text{R}j \in \mathcal{R}_i$. After this step, R$i$'s new macro graph is $H_i' \supseteq H_i$.

Next, R$i$ shares its updated macro graph $H_i'$ with all the other robots (both directly and indirectly connected) in the following way: R$i$ sends $H_i'$ to its direct neighbors, who merge $H_i'$ with their macro graphs. The neighbors then share their updated macro graphs with their neighbors, and so on. This synchronization can take multiple rounds of communication until no new information is available to all the connected robots. Thus, R$i$ receives macro graph information from the other robots, and R$i$ incorporates the shared information. After this operation, R$i$'s new macro graph is:

$$\begin{aligned} H_i'' &= H_i' \cup (H_1' \cup \ldots H_{i-1}' \cup H_{i+1}' \cup \ldots \cup H_n') \\ &= \bigcup_{j \in \{1,\ldots,n\}} H_j' \end{aligned}$$

Therefore, after the communication phase, every connected robot has the same macro graph $H$, where:

$$H = \bigcup_{i \in \{1,\ldots,n\}} H'_i$$
$$H \supseteq \bigcup_{i \in \{1,\ldots,n\}} H_i \text{ (since } H'_i \supseteq H_i, \forall i \in \{1,\ldots,n\}\text{)}$$

□

### 4.4 Finding a Solution

The robots build a macro network graph and search it to find a solution configuration:

**Definition 4.7.** *A macro edge* $\mathbf{e} = \{Ri, \mathbf{v}\}$ *is* ***applicable*** *to a position label* $Pi_\alpha$ *if:*

$$\mathbf{v} = Rj \text{ and } \{Pi_\alpha, Pj_\beta\} \text{ is in the equivalence class of } \mathbf{e} \text{ for some } \beta$$
$$\text{or}$$
$$\mathbf{v} = Ta \text{ and } \{Pi_\alpha, Ta\} \text{ is in the equivalence class of } \mathbf{e}$$

**Definition 4.8.** *A* ***solution*** *s of a macro network graph H is a sub-graph* $H_s = (\mathcal{V}_s, \mathcal{E}_s) \subseteq H$ *such that all the towers in* $\mathcal{T}$ *are connected, and each robot can be at a single position, i.e.,* $\forall\ Ri \in \mathcal{V}_s, \exists\ Pi_\alpha$ *such that* $\forall \mathbf{e} \in \mathcal{E}_s,\ Ri \in \mathbf{e} \Rightarrow Pi_\alpha$ *is applicable to* $\mathbf{e}$.

To reduce the search space for a solution $s$ (or equivalently $H_s$), we add the restriction that $H_s$ is acyclic — if $H_s$ contains cycles, then macro edges from $H_s$ can be removed (eliminating the cycles) while still ensuring that all the towers are connected.

In order to find such a solution $H_s$, the search begins at one of the towers (e.g., T$a$), by inserting all macro edges connected to T$a$ into a queue (i.e., $Q = \{\mathbf{e} \in \mathcal{E}' : \mathbf{e} = \{\mathrm{R}i, \mathrm{T}a\}, \mathrm{R}i \in \mathcal{V}\}$), and running the function *find_solution* recursively (see Algo. 1 for the pseudocode).

The *find_solution* algorithm proceeds as follows: $P$ contains the possible positions that the robots can be in — initially, all robots can be in all positions. Given a queue $Q$ of macro edges and the first macro edge $\mathbf{e}$ in the queue, the algorithm attempts to use $\mathbf{e}$ and recurse, as well as not use $\mathbf{e}$ and recurse. This ensures that all combinations of using and not using macro edges are tested. In addition, as the algorithm recurses, $Q$, the queue of macro edges to consider, $\mathcal{V}_c$, the set of vertices (robots/towers) already connected, $\mathcal{E}_{used}$, the set of macro edges used, and $\mathcal{E}_{skip}$, the set of macro edges that were skipped, are updated.

Although it may seem that this search performs a complete search of the macro graph, the search tree is pruned quickly, due to the constraints in each of the macro edges. Thus, $P$ becomes more and more constrained, limiting the number of macro edges still available for use, and reducing the search space considerably.

#### 4.4.1 Updating Constraints

Given a set of possible positions $P = \{P_1, \ldots, P_n\}$, where $P_i$ refers to possible positions of R$i$, and a macro edge $\mathbf{e} = \{v_1, v_2\}$, we update $P$ such that the constraints of the macro edge $\mathbf{e}$ are satisfied. In order to do so, we take the intersection of the constraints of $\mathbf{e}$ and the relevant elements of $P$.

Next, we iterate through all $\mathbf{e}' \in \mathcal{V}_s$ and further constrain $P$ (since the reduced positions of R$i$ may further constrain positions of R$j$ through a previously-used macro edge $\mathbf{e}' = \{\mathrm{R}i, \mathrm{R}j\}$). After all the propagations have completed, if $\exists i$ s.t. $|P_i| = 0$, then $P$ becomes invalid. We have the following result:

**Theorem 4.9.** *For a problem with $m$ towers and $n$ robots verifying Assumptions 1 through 5, all robots will find at least one solution w.p.1.*

*Proof.* From Assumption 2, there is at least one configuration in which all towers are connected (a solution). The fact that the environment is bounded in the sense of Assumption 3 and that the exploration algorithm is thorough in the sense of Assumption 4 guarantees that at least one robot eventually determines a solution (the probability of this not happening goes to 0 as $t \to \infty$).

From Thm. 4.6, robots synchronize their macro graphs when they meet. Using Assumptions 4 and 5, this implies that the network structure eventually propagates to all robots. Thus, if a solution to the connectivity problem is found by some robot R$i$, then all the robots will find this solution w.p.1 in the limit, as $t \to \infty$.

As a result, the solution propagates to all robots in the limit, thus establishing the desired result. □

**Algorithm 1** find_solution$(Q, P, \mathcal{V}_c, \mathcal{E}_{used}, \mathcal{E}_{skip})$

```
if Q is empty then
    return false
end if
e = popQueue(Q)
P' = updateConstraints(e, P)
if P' is valid then
    Q' = updateQueue(e, Q)
    V'_c = addVertex(e, V_c)
    E'_used ← E_used ∪ {e}
    if T ⊆ V'_c then
        solution ← (V'_c, E'_used)
        return true
    end if
    if find_solution(Q', P', V'_c, E'_used, E_skip) = true then
        return true
    end if
end if
E'_skip ← E_skip ∪ {e}
if find_solution(Q, P, V_c, E_used, E'_skip) = true then
    return true
end if
return false
```

### 4.5 Algorithm for Distributed Network Connectivity

The robots run a fully-distributed algorithm that allows them to find and converge to a solution. A flow-chart of the algorithm is shown in Fig. 6. The fully-distributed algorithm shown in Algo. 2, which includes a state heuristic function. The robot can be in one of four states, namely *Explore, Share_Solution, Goto_Solution*, and *Stop*. Each robot R$i$ starts in the *Explore* state, with an empty macro graph. Fig. 7 shows the state transition diagram for the robots.

**Algorithm 2** Distributed Network Connectivity Algorithm

```
1: H ← {} // H is the macro graph
2: state ← Explore
3: loop
4:    (T_i, R_i) = getConnections()
5:    // Create position label
6:    Pi_α = getPositionLabel(current coordinates)
7:    // Update macro graph H with connections to towers
8:    for all Ta ∈ T_i do
9:       addConstraint( {Ta, Pi_α}, H)
10:   end for
11:   // Update macro graph H with connections to robots
12:   // Synchronize macro graphs
13:   performCommunicationPhase(R_i, H)
14:   // Check for solution and update state
15:   if graph_was_updated then
16:      s = checkForSolution(H)
17:      if s is valid and s ≠ current_solution then
18:         current_solution ← s
19:         sol_posn = getSolutionPosition(s)
20:         neighbors = getNeighborsToInform(s)
21:         state ← Share_Solution
22:      end if
23:   end if
24:   if state = Share_Solution & informed(neighbors) then
25:      state ← Goto_Solution
26:   else if state = Goto_Solution & Pi_α = sol_posn then
27:      state ← Stop
28:   end if
29:   // Plan the next action
30:   action = getNextAction(state)
31:   // Execute the action
32:   executeAction(action)
33: end loop
```

At each time step, the robot R$i$ generates a list of towers $\mathcal{T}_i \subseteq \mathcal{T}$, and a list of robots $\mathcal{R}_i \subseteq \mathcal{R}$ that are in range. For each tower T$a \in \mathcal{T}_i$, robot R$i$ adds a constraint into its macro graph as R$i$'s current position label (i.e., P$i_\alpha$), the tower's ID (i.e., T$a$), and the signal strength of the connection.

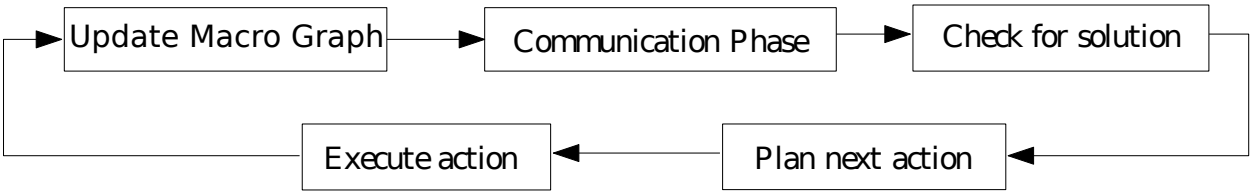



Figure 6: Flowchart of the fully-distributed algorithm

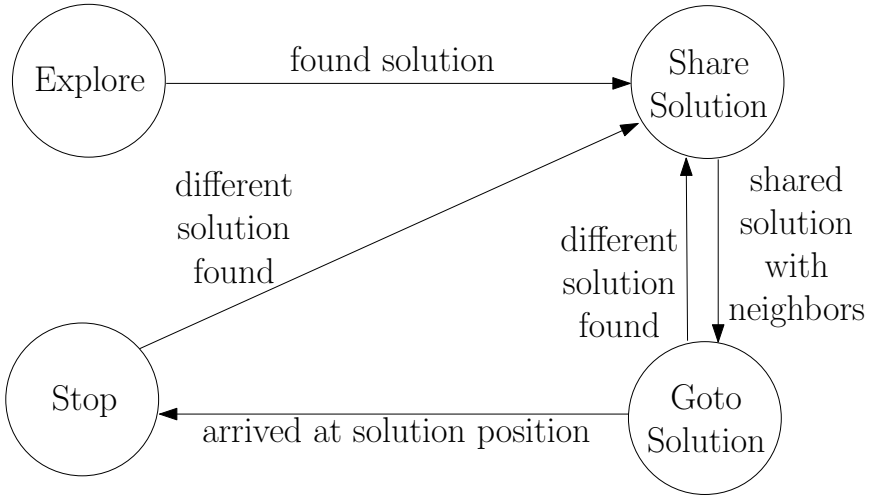



Figure 7: State transition diagram for each robot. The robots start in the *Explore* state. When all robots are in the *Stopped* state, the solution configuration has been achieved and all towers are connected.

R$i$ then enters a communication phase, where it adds constraints into its macro graph as R$i$'s current position label, the position labels of the robots R$i$ is directly connected to ($\mathcal{R}_i$), and the signal strengths of the connections. R$i$ then shares and synchronizes its macro graph with all the robots it is directly connected to ($\mathcal{R}_i$). After this phase, all robots that are connected will have the same macro graph.

Following the communication phase, R$i$ now searches the macro graph for a solution if the macro graph was updated. Lastly, R$i$ switches its internal states if necessary, based on whether a solution has been found.

### 4.5.1 Converging to a Solution

In Sec. 4.4, we have established that, with enough exploration, all robots eventually determine a solution. However, in environments where multiple solutions exist, it is possible that at any time step not all robots determine the same solution. Therefore, it is necessary to ensure that, in the presence of multiple solutions, all robots adopt and move to the same solution.

The process of ensuring consensus in a common solution arises from a common and deterministic solution selection mechanism. As seen before, the algorithm to find a solution (Algo. 1) is deterministic. Furthermore, since robots synchronize their macro graphs when they meet, connected robots will find the same solution.

Once a robot R$i$ has found a solution, it attempts to find its neighbors in the solution and synchronize its macro graph with them (*Share_Solution* state). After all its neighbors have synchronized their macro graphs, R$i$ heads to its solution position (*Goto_Solution* state). Finally, after arriving at the solution position, R$i$ will stop moving (*Stop* state). If at any time, a better solution is found (e.g., by receiving new information from other robots), R$i$ will restart its convergence process and attempt to find its neighbors again.

If a robot R$i$ finishes sharing its solution with its neighbors and moves to its final position while other robots are still negotiating, either the other robots settle in their positions corresponding to the solution adopted by robot R$i$ or some robot (that adopted a different solution) will not stop until it connects to robot R$i$. At this point, they synchronize their macro graphs and adopt the same solution. If the solution found is different, R$i$ restarts its sharing process. This means that, since the number of robots is finite, they all eventually settle in one solution and move to the corresponding position. This conclusion is stated in the following result:

**Theorem 4.10.** *For a problem with $m$ towers and $n$ robots verifying Assumptions 1 through 5, all robots converge to the same solution w.p.1.*

# 5 Planning an Action for Connectivity

As mentioned above, the robot can be in one of 4 states: *Explore, Share_Solution, Goto_Solution*, and *Stop* (see Fig. 7). In the *Explore* state, the goal of the planner is to traverse the world such that the robots will eventually find a solution configuration $s$ that connects all the towers. In the *Share_Solution* state, a solution $s$ has been found, and the goal is to communicate this solution $s$ to neighbor robots in $s$. In the *Goto_Solution* state, the planner has to find the shortest path from the current location to the robot's position in the solution $s$. Lastly, in the *Stop* state, the planner has no work to do and merely stops the robot in place.

We now describe a number of different heuristics that are used for exploration:

### Random Movement

The simplest heuristic was random movement, where a robot would choose an action randomly from the list of possible actions. There was no weighting of the actions, so if $k$ actions were available, they would each have a $\frac{1}{k}$ probability of being selected. This heuristic provides a baseline for comparison, since it is arguably the most naive form of exploration.

### Coverage of the Space

The next heuristic we considered was a coverage algorithm. We adapted the node-counting algorithm described in [7]. Each robot kept a counter $C_c$ of how many times it visited a cell $c$. Then, when choosing an action, it picks the adjacent cell $c'$ such that $C_{c'}$ is the minimum among all adjacent cells. In the case where more than one cell has the minimum value, it picks randomly among the minimum cells. All cells are initialized to have a counter of 0, so unexplored cells always have priority over explored cells.

**Weighted Exploration**

This heuristic was similar to the above coverage algorithm, except that the robot uses a weighted dice to decide among its adjacent cells, instead of choosing the least-visited cell (i.e., with the minimum value). We defined a ratio $\gamma$, that represents the exploration vs exploitation probabilities. Given $k_1$ unexplored adjacent cells, and $k_2$ explored adjacent cells, it chooses to explore with $\frac{k_1\gamma}{k_1\gamma+k_2(1-\gamma)}$ probability, and exploit otherwise. If it chooses to explore, it picks one of the unexplored cells randomly. Otherwise, if it chooses to exploit, it picks an explored cell, weighted on how many times it has previously visited that cell. For each explored cell $c_k$ and corresponding counter $C_{c_k}$, we define $p_k = 1 - \frac{C_{c_k} - \min_j C_{c_j}}{\max_j C_{c_j} - \min_j C_{c_j}} + \alpha$, where $\alpha$ is a weighting term such that the cell with the maximum counter will not have a 0 probability of being chosen. Given the $p_k$ of the adjacent cells, the robot picks a cell $c_k$ with a probability of $\frac{p_k}{\sum_j p_j}$.

For example, suppose the adjacent cells are $(c_1, 0), (c_2, 0), (c_3, 3), (c_4, 5), (c_5, 2)$, where each tuple represents an adjacent cell and its corresponding counter (where 0 means unexplored). The robot will choose to explore with a $\frac{2\gamma}{2\gamma+3(1-\gamma)}$ probability. If it decides to explore, it will pick either $c_1$ or $c_2$ with equal probability. If it decides to exploit, it will pick $c_3$ with a probability of $\frac{\frac{1}{2}+\alpha}{(\frac{1}{2}+\alpha)+(\alpha)+(1+\alpha)}$, $c_4$ with a probability of $\frac{\alpha}{(\frac{1}{2}+\alpha)+(\alpha)+(1+\alpha)}$ and $c_5$ with $\frac{1+\alpha}{(\frac{1}{2}+\alpha)+(\alpha)+(1+\alpha)}$.

**Stay at Towers**

In this heuristic, the robots had one of 2 roles: stay at an assigned tower, or avoid towers. A robot R$i$ is assigned the role of staying at tower T$a$ if in its macro graph, it has the most connections to T$a$. Otherwise, the robot R$i$ assumes the avoid towers role.

In the *stay at tower* role, if the robot R$i$ is not currently connected to its assigned tower T$a$, then it plans the shortest path (using breadth-first search) to the nearest cell that connects it to T$a$ (based on the map it built while exploring the world). If the robot is already connected to T$a$, then it decides to explore or exploit using a weighted dice, similar to the weighted exploration heuristic above. However, in this case, it ignores all adjacent explored cells that do not have a connection to T$a$. Thus, the weightage only occurs for unexplored cells, and explored cells that are known to have a connection to T$a$. In this way, R$i$ stays close to T$a$ and may lose connection only if it goes to an unexplored cell that is out of T$a$'s range.

In the *avoid towers* role, instead of choosing between explore and exploit, the robot chooses between explore, exploit, and visiting a tower, with probabilities $\alpha$, $\beta$, and $1 - \alpha - \beta$ respectively. The robot chooses between these 3 options based on the number of adjacent cells that match the requirement: i.e. if there are $k_1$ unexplored cells, $k_2$ explored cells that do not have a connection to a tower, and $k_3$ explored cells that have a connection to a tower, then the robot chooses to explore with $\frac{k_1\alpha}{k_1\alpha+k_2\beta+k_3(1-\alpha-\beta)}$ probability, exploit with $\frac{k_2\beta}{k_1\alpha+k_2\beta+k_3(1-\alpha-\beta)}$, and visits a tower cell otherwise. If it chooses to exploit or visit a tower, then the relevant cells are selected probabilistically based on their counter values, similar to the weighted exploration heuristic.

By using the heuristic, the robots that do not have an assigned tower tend to visit areas that have no connections to any tower, and explore new regions. This allows new towers to be discovered quickly, as well as connections to be found between towers. We experiment on this heuristic in detail in Sec. 8.

# 6 Modeling Distance Based on RSSI

After phase 1 of the algorithm is completed, the team of robots are placed in a configuration that connects the static wireless towers. An autonomous agent is also present in the environment in phase 2, and the goal of the robot team is now to locate and tether to the autonomous agent.

To do so, we use the received signal strength indicator (RSSI) between the robots and the autonomous agent to infer the distance between them. In an indoor environment, where there are walls, furniture and other obstacles, signal attenuation and multi-pathing makes it difficult to estimate distance accurately [3]. Furthermore, we are interested in the case where robots are placed in unknown environments, so they do not have a map that can be used to model multi-pathing and attenuation.

## 6.1 Collecting Real-World Data

We use a data-driven approach, similar to [20], in order to create an RSSI-distance model in a complex indoor environment. We used a pair of iRobot Creates, each with a Gumstix Verdex microprocessor and 2 attached radio antennas (Fig. 8). The antennas allow the robots to communicate wirelessly, and provide the RSSI of their connection (in integers of dBm). We then collected (RSSI, distance) pairs, over a range of 0 to 40m. The black dots in Fig. 9 shows a scatter plot of the collected data. Within the range of distances, there were many configurations of walls, furniture, and other obstacles. Thus, the data collected provided a large sample of possible configurations of the world, and not a model of open space.

Figure 8: One configuration of iRobot Creates used to collect real-world RSSI data. Many configurations of walls and obstacles between the robots were used in the data collection.

## 6.2 Creating the RSSI-Distance Model

The black and gray lines in Fig. 9 show the maximum and minimum boundaries of the model created, based on the data collected. The intuition behind the lines is that the maximum and minimum RSSIs should decrease monotonically as the distance increases. In this manner, the maximum RSSI-distance and minimum RSSI-distance lines are composed of horizontal segments and slopes of negative gradient. Our RSSI-distance model differs from [20] in that we construct

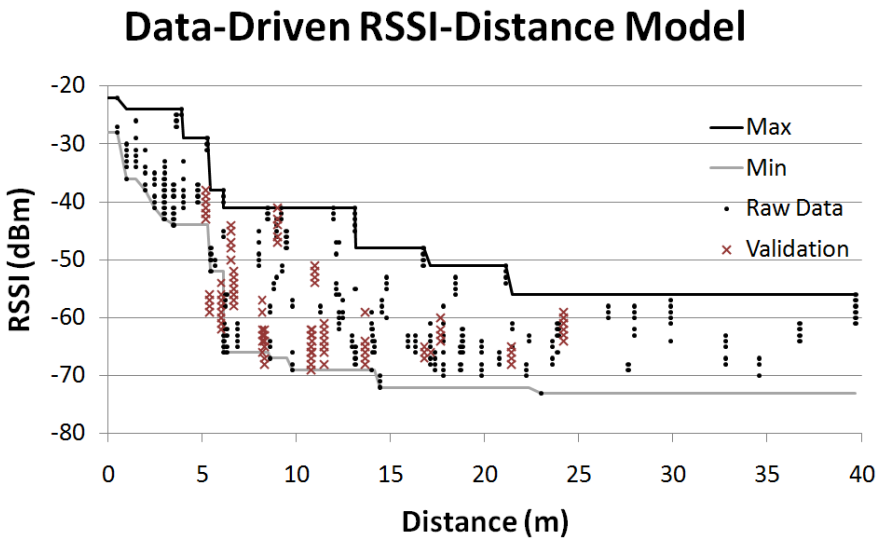



Figure 9: The RSSI-Distance model created from real-world data. The black and gray lines show the maximum and minimum boundaries of the model, and the black dots show the data used to generate the model. The red crosses show data collected from a different part of the building.

the minimum and maximum bounds instead of a trapezoid, because our data did not show lower likelihood near the boundaries.

Using the RSSI-distance model $M$, we can determine the minimum and maximum RSSI for a given distance, i.e., $(dist_{min}, dist_{max}) = M_{rssi}(RSSI)$, and the minimum and maximum distance for a given RSSI, i.e., $(RSSI_{min}, RSSI_{max}) = M_{distance}(dist)$.

From the function $M_{distance}$, we also define a likelihood function $M_{likelihood}$, such that there is a constant prior within the upper and lower bounds of RSSI, i.e., $M_{likelihood}(dist, RSSI) = 1$ if $RSSI_{min} \leq RSSI \leq RSSI_{max}$ and 0 otherwise.

While the model we create does not explicitly capture the number of walls and obstacles in the environment, it implicitly handles all possible configurations of walls and obstacles that were present in the data-collection phase. Thus, with sufficient data, the model generated can handle all possible configurations of walls (of different thickness and materials) and obstacles. We collected real-world data from one floor of the building to create the model, and used data collected from a different floor of the building to validate the model. The red crosses in Fig. 9 show the validation data. While not all the crosses fall within the model, the distance between the cross and the boundary of the model is very small ($< 1$m). Thus, the model is sufficiently accurate and is applicable for use in situations when robots have no map of the environment, but have representative data used to create the model.

## 7 Tethering to a Target with Multiple Robots

In the second phase of our algorithm, an autonomous agent enters the environment, and the goal of the robot team is to locate and tether to this autonomous agent. We define $R \subseteq \mathcal{R}$, where $|R| = n' \leq n$, to be the subset of the robot team that is connected to each other and to the autonomous agent, and we define $R_{n'+1}$ to be the autonomous agent.

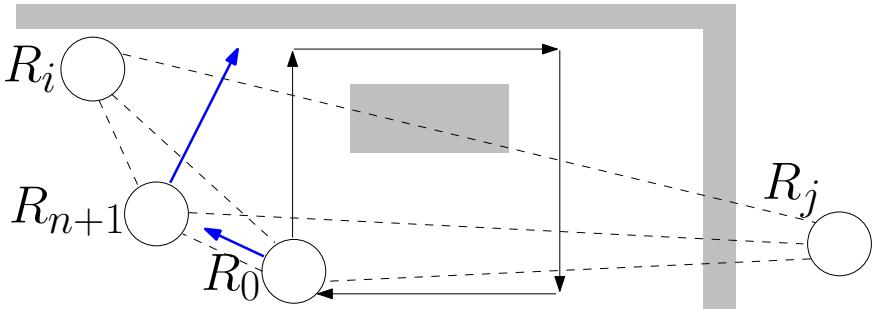



Figure 10: An example scenario: dotted lines and gray areas indicate connections and obstacles respectively. $R_0$ first moves to determine the target $R_{n'+1}$'s location (black arrows), and when $R_{n'+1}$ begins moving, $R_0$ tethers to it (blue arrows).

**Definition 7.1.** *The **multi-robot tethering domain** is a tuple $\{L, R, X, O, f_O\}$, where:*

- *$L$ is the set of locations in the world;*
- *$R = \{R_0, \ldots, R_{n'+1}\}$ is the set of robots, where $R_0$ is the seeker, $R_1, \ldots, R_{n'}$ are $n'$ cooperative teammates, and $R_{n'+1}$ is the autonomous agent;*
- *$X$ is the set of states of the world, i.e., $x^{(t)} \in X$ contains the locations of the robots, walls and obstacles at time $t$, and $x_i^{(t)} \in x^{(t)}$ is the location of $R_i$ at time $t$;*
- *$O$ is the set of possible observations, i.e., connections among robots and the corresponding RSSI;*
- *$f_O : X \to O$ is the observation generation function, which determines signal strengths of connections among the robots, given the state of the world.*

The configuration of the walls and obstacles in the world are initially unknown to the robots, and they are initially placed in a random configuration. In addition, the observation function $f_O$ is unknown to the robots, although they have a model $M$ of RSSI and distance (Sec. 6). In a real-world environment, $f_O$ would include the physical attributes of signal propagation and attenuation, multi-pathing and other factors, while $M$ provides a minimum and maximum distance given RSSI and vice versa.

The **multi-robot tethering problem** is for one of the robots in the team, which we call the seeker $R_0$, to tether to the target $R_{n'+1}$, i.e., minimize $\sum_{t \in T} |x_0^{(t)} - x_{n'+1}^{(t)}|$, as $R_{n'+1}$ moves in the environment, using information from the multi-robot team $R_0, \ldots, R_{n'}$. However, $R_{n'+1}$ is not part of the multi-robot team and does not communicate or inform the team that it is moving; the team must infer $R_{n'+1}$'s location and motion through RSSI measurements only.

In this article, we focus on the case where $R_1, \ldots, R_{n'}$ remain stationary in the world, and communicate their connections and RSSI to $R_0$, and we do not discuss how $R_0$ is selected. In the first step of our algorithm, we assume that $R_{n'+1}$ is stationary, and the multi-robot team determines $R_{n'+1}$'s initial location $x_{n'+1}^{(0)}$. In the second step of the algorithm, after $R_{n'+1}$'s location has been determined, $R_{n'+1}$ begins moving and $R_0$ remains tethered to it. Fig. 10 shows an example scenario of the problem and our algorithm.

### 7.1 Locating the Target

At each time step $t$, an observation $o^{(t)}$ is generated by the observation generation function $f_O$, and each robot $R_i$ observes $o_i^{(t)} \subseteq o^{(t)}$. $R_0, \ldots, R_{n'}$ communicate and merge their individual observations. Even though the target robot $R_{n'+1}$ does not communicate, its connections can be inferred from the team's observations, e.g., if $R_{n'+1}$ and $R_i$ are connected, then that connection is shared by $R_i$. Thus, $o^{(t)} = \cup_{i=0}^{n'} o_i^{(t)}$, and $o^{(t)}$ is available to the multi-robot team.

The robots are initially randomly positioned in the world. However, from the RSSI of their connections, they can infer the maximum and minimum distances between them, using the RSSI-distance model $M$. In addition, all the robots except $R_0$ are stationary in this step, and we assume that $R_0$ is capable of keeping track of its location relative to its initial location, i.e., it has good odometry and sensor feedback in order to compensate for dead-reckoning errors. Thus, $x_0^{(t)}$ is known to $R_0$ in its own coordinate frame.

#### Creating Constraints from Observations

For each observation $o^{(t)} \in O$, we define a function $CreateConstraints(o^{(t)})$ that creates constraints $C$, based on the RSSI of connections $c_{i,j} \in o^{(t)}$ (Algo. 3).

The RSSI-distance model $M$ is used to infer the maximum and minimum distance ($d_{min}$ and $d_{max}$) of 2 robots ($R_i$ and $R_j$) by using the RSSI of their connection. Since only $R_0$ moves in this step of the algorithm, $x_i^{(t)} = x_i^{(0)} \; \forall \; i \neq 0$. Thus, $CreateConstraints(o^{(t)})$ returns constraints of the forms $d_{min} \leq |x_i^{(0)} - x_j^{(0)}| \leq d_{max}$ and $d_{min} \leq |x_0^{(t)} - x_i^{(0)}| \leq d_{max}$.

**Algorithm 3** Creating Constraints from an Observation

CreateConstraints($o^{(t)}$)
  $C \leftarrow \{\}$
  **for all** $(R_0, R_i, s_{0,i}) \in o^{(t)}$ **do**
    $(d_{min}, d_{max}) \leftarrow M_{rssi}(s_{0,i})$
    $addConstraint(C, d_{min} \leq |x_0^{(t)} - x_i^{(0)}| \leq d_{max})$
  **end for**
  **for all** $(R_i, R_j, s_{i,j}) \in o^{(t)}$ s.t. $i, j \neq 0$ **do**
    $(d'_{min}, d'_{max}) \leftarrow M_{rssi}(s_{i,k})$
    $addConstraint(C, d'_{min} \leq |x_i^{(0)} - x_j^{(0)}| \leq d'_{max})$
  **end for**
  **return** $C$

#### Moving the Seeker and Merging Constraints

In order to locate the target more effectively, the seeker moves along a pre-determined path, and creates new constraints as it does so. $x_0^{(t)}$ is known to $R_0$ in its own coordinate frame, and so these constraints form rings of possible locations. For example, if $x_0^{(t)} = (0, 0), d_{min} = 5, d_{max} = 10$, then $x_i^{(0)}$ is a ring of radius 5 to 10 centered about the origin. Thus, intersections of these rings over time provide a good estimate of the robots' initial locations.

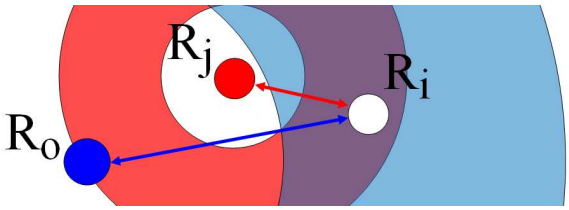



Figure 11: Constraints on $R_i$'s position generated from connections to $R_0$ (in blue), and $R_j$ (in red).

However, as Fig. 9 shows, for a given RSSI, the range of possible distances can be very large, e.g., $d_{min}$ and $d_{max}$ for -60 dBm is 5.5m to 39.7m respectively. As such, the intersections of rings from constraints involving $R_0$ help to narrow a robot's position (Fig. 11), but not exactly locate it unless $R_0$ travels a large distance. Pairwise distance constraints between other robots are necessary for more accurate positions of the robots.

**Finding Possible Joint Locations of Robots**

All the constraints while the seeker is moving can be merged to form a set $C$. Possible joint locations of all the robots, i.e., $x_1^{(0)}, \ldots, x_{n'+1}^{(0)}$, can be inferred from this set of constraints $C$. However, it is infeasible to simply consider all possible combinations of locations and checking if $C$ is satisfied, as the size of the joint location space grows exponentially as $n'$ increases.

As such, we developed an algorithm to generate possible joint locations of the robots, as shown in Algo. 4. The algorithm first initializes the set of possible locations of each robot as $L$, the set of locations in the domain. Next, by considering the constraints involving $R_0$, this set of individual locations is reduced in size. The next step of the algorithm considers joint-locations of 2 robots, and eliminates combinations that do not satisfy $C$. The algorithm then continues iterating on the number of robots to consider in the joint-space, while eliminating combinations that do not satisfy $C$. In this way, the size of possibilities does not grow unbounded as $C$ eliminates unsatisfiable joint locations, and converges to a small number of complete joint-locations of all robots, given sufficient movement of the seeker.

## 7.2 Tethering to the Target

After the first step is completed, the robot team has a good estimate of the target's location, and the seeker $R_0$ is to remain tethered to the target robot $R_{n'+1}$ as the target moves in the world. The target does not communicate with the other robots, and the team has to infer its movement through observations. The goal is to minimize the distance between the seeker's location $x_0^{(t)}$ and the target's location $x_{n'+1}^{(t)}$ over a time period.

To maintain an accurate estimate of $x_{n'+1}^{(t)}$, we use a probabilistic approach, and apply Bayesian updates to the observations. From Def. 7.1, $x^{(t)} \in X$ is the state of the world at time $t$, where $x_i^{(t)} \in x^{(t)}$; we denote $x^{(t_1:t_2)} = x^{(t_1)}, \ldots, x^{(t_2)}$. Similarly, $o^{(t)}$ and $u^{(t)}$ are all the observations and control inputs at time $t$ respectively.

We define $\overline{bel}_i(x_i^{(t)}) = P(x_i^{(t)}|o^{(1:t-1)}, u^{(1:t)})$, which is the belief of $R_i$'s position at time $t$, given observations up to time $t-1$ and $bel_i(x_i^{(t)}) = P(x_i^{(t)}|o^{(1:t)}, u^{(1:t)})$ as the belief of the robot's

**Algorithm 4** Generating possible joint locations

GetLoc($x_0^{(0)} \ldots, x_0^{(T)}, o^{(0)}, \ldots, o^{(T)}$)
  // Generate all constraints
  $C \leftarrow \cup_{t=0}^{T} CreateConstraints(o^{(t)})$
  // Initialize individual robot locations
  **for** $i = 1, \ldots, n' + 1$ **do**
    $X_i \leftarrow L$ // $L$ is the set of all possible locations
  **end for**
  // Constrain individual locations using $R_0$'s connections
  **for all** $(d_{min} \leq |x_0^{(t)} - x_i^{(0)}| \leq d_{max}) \in C, x_i \in X_i$ **do**
    **if** $|x_0^{(t)} - x_i| < d_{min}$ or $|x_0^{(t)} - x_i| > d_{max}$ **then**
      $X_i \leftarrow X_i \backslash \{x_i\}$
    **end if**
  **end for**
  // Generate joint locations
  $X \leftarrow X_1$
  **for** $i = 2, \ldots, n' + 1$ **do**
    $X' \leftarrow X$; $X \leftarrow \{\}$
    **for all** $x_i \in X_i, (x_1, \ldots, x_{i-1}) \in X'$ **do**
      **if** $SatisfyConstraints(C, (x_1, \ldots, x_i))$ **then**
        $X \leftarrow X \cup \{(x_1, \ldots, x_i)\}$
      **end if**
    **end for**
  **end for**
  **return** $X$

position after incorporating the latest observation. From the Markovian assumption,

$$bel_i(x_i^{(t)}) \propto \sum_{x^{(t)} s.t. x_i^{(t)}} P(o^{(t)}|x^{(t)}) \prod_{j=0}^{n'+1} \overline{bel}_j(x_j^{(t)}) \tag{1}$$

Eqn. 1 is intractable to compute, since the combinations of $x^{(t)}$ grows exponentially as $n$ increases, and $P(o^{(t)}|x^{(t)})$ depends on the observation generation function $f_O$, which is unknown to the robots. However, by using the RSSI-distance model $M$ (Sec. 6), we model that the RSSI between any 2 robots depends only on their distance, and not on the locations of other robots or obstacles. Thus, we can simplify $P(o^{(t)}|x^{(t)})$:

$$P(o^{(t)}|x^{(t)}) = \prod_{c_{(i,j)} \in o^{(t)}} P(c_{(i,j)}|x^{(t)}) \tag{2}$$

$$= \prod_{c_{(i,j)} \in o^{(t)}} P(c_{(i,j)}|x_i^{(t)}, x_j^{(t)}) \tag{3}$$

$$= \prod_{c_{(i,j)} \in o^{(t)}} P(c_{(i,j)}|dist_{i,j}), \text{where } dist_{i,j} = |x_i^{(t)} - x_j^{(t)}| \tag{4}$$

**Algorithm 5** Calculating summed likelihoods

*GetLikelihood*$(M, o^{(t)}, \overline{bel})$

// $sum_{c_{(i,j)},i}$ stores $\sum_{x_i^{(t)}, x_j^{(t)}} P(c|x_i^{(t)}, x_j^{(t)})\overline{bel}_i(x_i^{(t)})$

**for all** $c_{(i,j)} = (R_i, R_j, s_{i,j}) \in o^{(t)}$ **do**

$sum_{c_{(i,j)},i} \leftarrow 0$; $sum_{c_{(i,j)},j} \leftarrow 0$

**end for**

**for all** $L_1, L_2 \in L$ **do**

**for all** $c_{(i,j)} = (R_i, R_j, s_{i,j}) \in o^{(t)}$ **do**

$sum_{c_{(i,j)},i} += M_{likelihood}(s_{i,j}, |L_1 - L_2|)\overline{bel}_i(L_1)$

$sum_{c_{(i,j)},j} += M_{likelihood}(s_{i,j}, |L_1 - L_2|)\overline{bel}_j(L_2)$

**end for**

**end for**

**return** $(sum_{c_{(i,j)},i}, sum_{c_{(i,j)},j}) \forall c_{(i,j)} \in o^{(t)}$

Also, we make an approximation that for each robot $R_i$, $P(o^{(t)}|x^{(t)})$ is only dependent on its connection to $R_0$, and the connections between $R_i$ and other robots. This allows a simplification of Eqn. 1:

$$bel_i(x_i^{(t)}) \propto \sum_{x^{(t)} s.t. x_i^{(t)}} P(o^{(t)}|x^{(t)}) \prod_{j=0}^{n'+1} \overline{bel}_j(x_j^{(t)}) \tag{5}$$

$$\propto \sum_{x^{(t)} s.t. x_i^{(t)}} P(c_{(0,i)}, c_{(i,1)}, \ldots, c_{(i,n'+1)}|x^{(t)}) \prod_{j=0}^{n'+1} \overline{bel}_j(x_j^{(t)}) \tag{6}$$

$$\propto \overline{bel}_i(x_i^{(t)}) P(c_{(0,i)}|x_0^{(t)}, x_i^{(t)}) \sum_{x^{(t)} s.t. x_0^{(t)}, x_i^{(t)}} \prod_{c_{(i,j)} \in o^{(t)}} P(c_{(i,j)}|x_i^{(t)}, x_j^{(t)}) \overline{bel}_j(x_j^{(t)}) \tag{7}$$

$$\propto \overline{bel}_i(x_i^{(t)}) P(c_{(0,i)}|dist_{0,i}) \prod_{c_{(i,j)} \in o^{(t)}} \sum_{x_j^{(t)}} P(c_{(i,j)}|dist_{i,j}) \overline{bel}_j(x_j^{(t)}) \tag{8}$$

Eqn. 7 uses the fact that $x_0^{(t)}$ is known and $x_i^{(t)}$ is given as a parameter, and that the probabilities of the connections between $R_i$ and other robots are independent (using the model $M$). Eqn. 8 reverses the order of the sum and product, since the combinations of $x_j^{(t)}$ are independent and can be factorized in that way.

In particular, since $L$, the set of locations, is common for all robots, $\sum_{x_j^{(t)}} P(c_{(i,j)}|dist_{i,j})\overline{bel}_j(x_j^{(t)})$ can be computed in $O(|L|)$ steps simultaneously. Furthermore, $x_i^{(t)} \in L$, so by looping across all combination $L_1, L_2 \in L$, the beliefs of all robots can be computed in quadratic time, instead of being exponential in the number of robots. We incrementally update the beliefs of each robot at every timestep in this manner. Although each connection in $o^{(t)}$ can be treated independently, in practice this is also a quadratic operation, since it involves a summation across combinations of 2 robot locations. Hence, our approach provides higher accuracy while maintaining the same com-

putational cost. In particular, Algo. 5 computes $sum_{c_{(i,j)},i} = \sum_{x_i^{(t)},x_j^{(t)}} P(c_{(i,j)}|x_i^{(t)}, x_j^{(t)})\overline{bel}_i(x_i^{(t)})$ and $sum_{c_{(i,j)},j} = \sum_{x_i^{(t)},x_j^{(t)}} P(c_{(i,j)}|x_i^{(t)}, x_j^{(t)})\overline{bel}_j(x_j^{(t)})$ for all $c_{(i,j)} \in o^{(t)}$ simultaneously.

The estimated location of the target is updated as part of this incremental belief update step, and the seeker then plans the shortest path (around obstacles if necessary) to the target's estimated location, in order to minimize the tether distance. As the seeker encounters obstacles, it updates its local map and replans the path to the target.

# 8 Experiments in Establishing Connectivity

In this section, we describe the extensive experiments that we performed in simulation on the first phase of our algorithm — for the robot team to establish connectivity among the static wireless towers.

## 8.1 Setup

We created a simulator that models a discrete 2D world, which allows horizontal and vertical walls (in between the discrete cells) to be placed anywhere in the world. The simulator calculates signal strength between any two cells in the world, based on an exponential decay rate, as well as degradation from the walls. We did not simulate interference between robots, or reflection of signals from the walls; the underlying algorithm would not be significantly affected even if the signal strength calculations were different. In this 2D world, each robot had 4 possible actions, namely to move north, south, east, or west. In the event that an action would cause the robot to hit a wall, the action would fail and the robot would stay where it originally was; otherwise the robot would move in the direction specified.

We implemented our algorithm as described in Sec. 4, as well as the different heuristics described in Sec. 5. In addition, we created 3 different scenarios: an Office World, a Corridor World, and a Lobby World (see Fig. 12).

### 8.1.1 Office World

The Office World was $20 \times 24$ cells in size, and contained 2 small offices on the top, and a larger office at the bottom left (see Fig. 12a). An L-shaped corridor ran in between the top offices and the bottom. A tower was placed inside each office, at a distance such that they could not communicate directly.

Due to the small size of the Office World, we fixed the number of robots to 5 and selected initial starting positions of the robots. This world was designed as a proof-of-concept of the algorithm, as well as to compare the performance of the different heuristics given a fixed initial state.

### 8.1.2 Corridor World

The Corridor World was $40 \times 24$ cells in size, and contained 8 offices. 4 offices were arranged horizontally on the top of the world, and the other 4 were arranged horizontally at the bottom (see Fig. 12b). A long corridor ran in between the top row of offices and the bottom, and 4 towers were placed in a zigzag fashion in the offices.

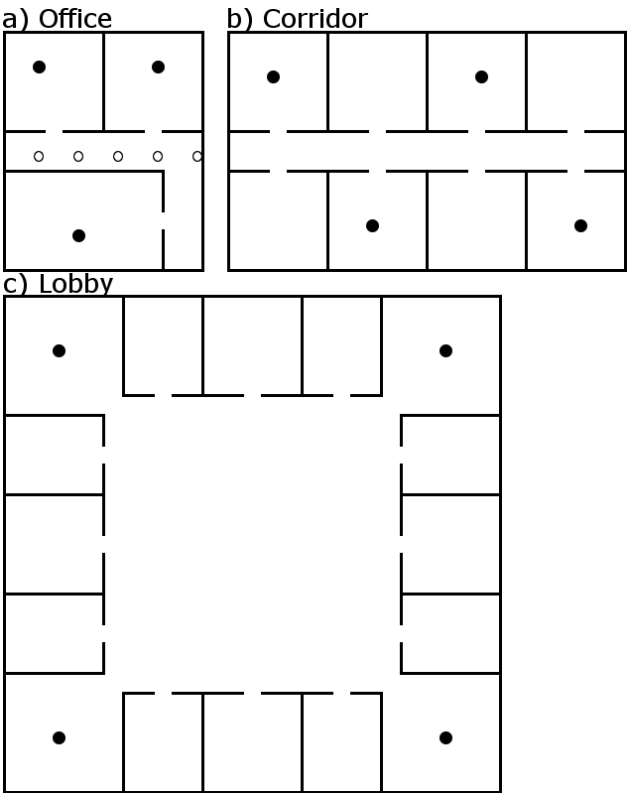



Figure 12: Representative worlds that were experimented on: a) Office World b) Corridor World, c) Lobby World. Black lines indicate walls/obstacles, filled circles represent towers, and hollow circles (only in Office World) show the fixed initial configuration of robots. Corridor and Lobby Worlds had random initial configurations of robots.

The Corridor World provided a realistic depiction of many corridors in university hallways which are flanked by offices on both sides.

### 8.1.3 Lobby World

The Lobby World was $50 \times 50$ cells in size, and contained a large lobby in the middle ($30 \times 30$). Around the lobby were 12 small offices - 3 on each side, as well as 4 slightly larger offices located at each corner of the lobby (see Fig. 12c). We placed 4 towers in this world, 1 in each of the corner offices.

The Lobby World provided a depiction of a large lobby area, that is connected to many small rooms. This world was designed not only to simulate real-world situations, but also to provide a worst-case scenario for our algorithm. By having a large open area, the robots would be able to move around freely and create a dense macro graph, where each robot vertex would have every other robot vertex as a neighbor. This could cause the search for solution to take extremely long, and so we wanted to test the effectiveness of our algorithm in such a scenario.

### 8.2 Comparing the Heuristics

We ran the different heuristics in the Office World scenario (which had 5 robots in a fixed initial configuration), with 100 trials per heuristic. In each trial, we measured how much time the algorithm took in order to find a solution configuration. We then compared the percentage of trials that found a solution in $t$ seconds on less; Fig. 13a shows the comparison of the different heuristics in the Office World. All the heuristics performed well in this scenario, with the *stay-at-towers* heuristic performing slightly better than the others. It found 100% of the solutions within 12s of execution, compared to 23s of the *weighted exploration* heuristic. The *coverage* and *random* heuristics found 99% of the solutions within 36s and 69s respectively.

We believed that all the heuristics performed relatively well in the Office World scenario due to the fact that many solution configurations existed, and that the robots began in a configuration that was close to many solutions. Therefore, we performed similar experiments on the Corridor and Lobby Worlds, with 5 robots in each case, and a random initial configuration for the robots in each trial.

As shown in Figs. 13b and 13c, the *stay-at-towers* heuristic outperformed all other heuristics by a large margin, in terms of the time taken to find a solution, as well as the percentage of solutions found given a fixed amount of computation time. It is interesting to note that the *coverage* and *weighted exploration* heuristics performed more poorly than the *random* heuristic. This is because in a large world, a large emphasis on exploration reduces the chance that robots will meet in a "useful" configuration (where useful refers to a partial configuration that can later be used as part of a solution), since they rarely return to previously visited cells. As such, it is difficult for the robot team to find a solution configuration, even after they have individually explored the entire environment.

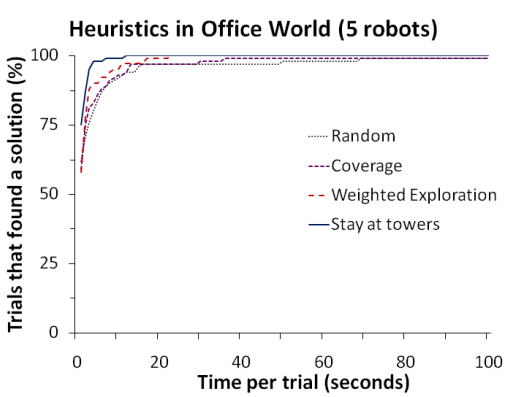
After these experiments, we concluded that the *stay-at-tower* heuristic was the most promising, as it performed well in all the scenarios. We then tested this heuristic extensively in the Corridor and Lobby Worlds, as described below.

### 8.3 Further Experiments for Corridor and Lobby

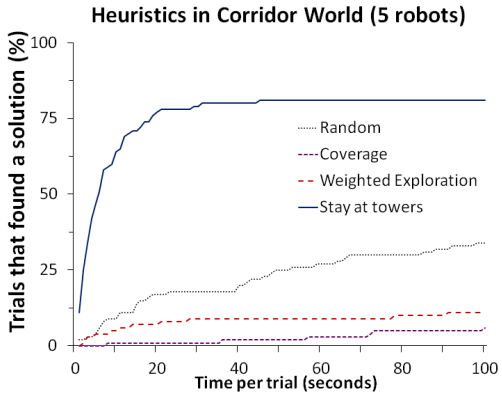
We used the *stay-at-towers* heuristic exclusively for all the experiments described below, as it was the most promising heuristic (see Sec. 8.2). In each experiment, we fixed the number of robots and ran 1000 trials. The initial configuration of the robots was randomly selected in each trial, and we measured how long the algorithm took to find a solution. Since our algorithm is distributed, but we were running a simulation where all the robots performed their computations, we divided the total amount of time elapsed by the number of robots being simulated. We then compared the percentage of trials that found a solution in a limited amount of time per robot. To be consistent with the previous experiments comparing the heuristics (which ran for 100s for 5 robots), the upper limit for each trial was set to 20s per robot, i.e., 100s for 5 robots, 300s for 15 robots, etc.

In Fig. 14a, we show the results in the Corridor World, with the number of robots varying from 5 to 50. Similarly, in Fig. 14b, we show the results in the Lobby World, also with the number of robots varying from 5 to 50. In Fig. 15a, we show some random initial configurations of 5 robots and the corresponding solutions found in the Corridor World; in Fig. 15b, we do the same for the Lobby World.

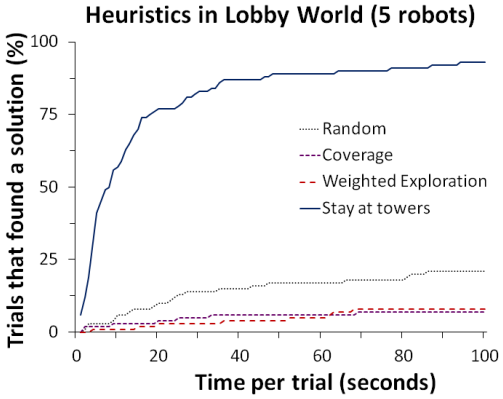
In both the Corridor World (Fig. 14a) and Lobby World (Fig. 14b), all the graphs trend upwards towards 100%, the graphs are not flat at the end of 20s, and would reach 100% if the algorithm was




(a) Office World

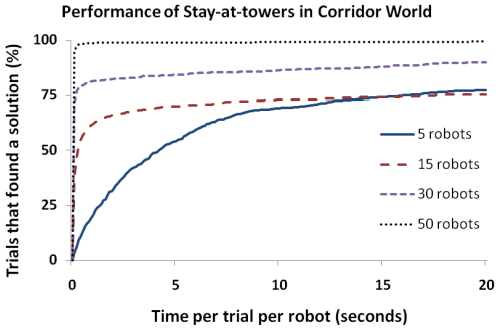



(b) Corridor World

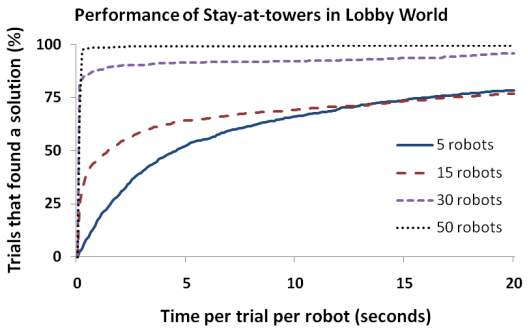



(c) Lobby World

Figure 13: Percentage of trials that found a solution in $t$ seconds or less with 5 robots in the different scenarios.



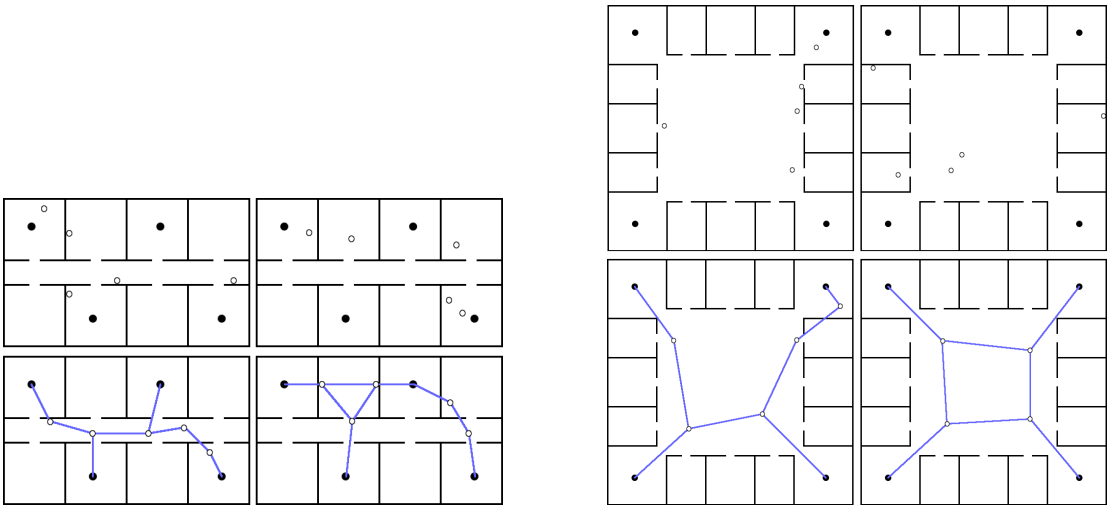

(a) Corridor World




(b) Lobby World

Figure 14: Percentage of trials that found a solution with varying number of robots running *stay-at-towers* heuristic.

run to infinity. An interesting observation is that the graph for 5 robots crosses that of 15 robots in both scenarios, which occurs because searching a network graph of 5 robots is much quicker than searching one with 15 robots — at each time step, $O(n^2)$ new constraints are added, where $n$ is



(a) Corridor World (b) Lobby World

Figure 15: The top row shows random initial configurations of 5 robots (hollow circles) and 4 towers (filled circles); the bottom row shows the corresponding solutions found. The black lines represent walls/obstacles, and the blue lines indicate connections between robots and towers.

| | Number of steps | |
|---|---|---|
| Number of robots | Corridor World | Lobby World |
| 5 | $7597 \pm 5544$ | $4304 \pm 3925$ |
| 15 | $131 \pm 205$ | $155 \pm 233$ |
| 30 | $6 \pm 12$ | $8 \pm 30$ |
| 50 | $0.3 \pm 1.6$ | $0.2 \pm 2.7$ |

Table 1: Number of steps to find a solution

the number of robots, and thus, while the size of the graph grows polynomially with the increase in the number of robots, searching for a solution in the graph takes exponentially longer. As the number of robots increases beyond 15, the number of possible solution configurations increases dramatically, and solutions are found much more quickly.

Table 1 shows the average number of steps to find a solution in Corridor and Lobby worlds. As the number of robots increases, the number of steps decreases substantially, which reflects the increase in the number of possible solution configurations. Also, when 50 robots are randomly placed in the world, there is a high probability that they already are in a solution configuration, which is why the average number of steps taken is close to 0.

## 9 Tethering Experiments

We implemented the second phase of our algorithm, to locate and tether to an autonomous agent, on the iRobot Creates and in simulation, and describe our experiments below.

## 9.1 Experimenting in Simulated Environments

We created a 2D discrete simulator, that allows walls to be placed in between cells of the world. For our experiments, we used a $30 \times 30$ world, with rooms and corridors to simulate an office environment. Each cell is 1m $\times$ 1m, and thus $L$, the set of locations in the world, is a set of 900 discrete cells.

The robots in the simulator can move in the 4 cardinal directions. Multiple robots are able to occupy the same cell, as we are simulating the iRobot Creates, and their physical dimensions are small enough for at least 4 of them to occupy a 1m $\times$ 1m area. The robots' actions move them exactly 1m in the desired direction, unless a wall blocks the path. We simulate perfect odometry on the robots, as imperfect odometry can be compensated by other sensory feedback. The robots build occupancy grids that allow path-planning.

To simulate connections between robots, we used the RSSI-distance model $M$ described in Sec. 6. For a given distance $d$, $M_{distance}(d)$ returns the maximum and minimum RSSI ($RSSI_{min}$ and $RSSI_{max}$ respectively) that can be achieved at the distance. We then return an RSSI value sampled uniformly between $RSSI_{min}$ and $RSSI_{max}$.

At each time step, the current positions of the robots are used to generate connections and RSSIs. Each robot receives a list of direct connections and their RSSIs. The team of cooperative robots, $R_0, \ldots, R_n$, communicate and merge their received information, as described in Sec. 7.

### 9.1.1 Locating the Target Successfully

In the first phase of our experiments, we initialized $R_0$ in the middle of the world, and randomly placed the other robots. $R_0$ was the only mobile robot, and it moved in a square of length 10m, i.e., North, then East, then South, then West, and collected observations, as described in Sec. 7.1. We varied $n$, the number of cooperative teammates, from 0 to 5, and measured the error in the estimated location of $R_{n+1}$ using the L2 metric. The estimated location of $R_{n+1}$ was calculated by considering all valid joint-location hypotheses of the robots, and taking an average of $R_{n+1}$'s location in each hypothesis. For each value of $n$, we ran 500 experiments.

Fig. 16 shows the error of the estimation as the number of teammates vary. When there are no teammates (only the seeker and target exist), the error in the estimated position of the target is $3.89 \pm 3.88$m. As the number of teammates increase, this error decreases monotonically, to $1.35 \pm 1.23$m when there are 5 teammates. Thus, having more teammates increases the accuracy of estimation, as more information is collected at each time step. Also, teammates provide additional distance constraints, which aid in narrowing down possible hypotheses in the target's location.

### 9.1.2 Tethering to the Target Effectively

In the second phase of our experiments, the target robot moves, and the seeker minimizes its distance to the target. We simulated the target's movement by first flipping a balanced coin to decide if the target takes an action. If an action is to be taken, then the target flips a weighted coin to decide if it maintains its current direction (70%) or heads in a new direction (30%) with equal probability for each direction.

We initialized the seeker at the center of the world, and placed the target robot in the cell north of the seeker, to simulate that the seeker moved to the target prior to the target's motion. The other robots in the team were randomly positioned in the world. The target robot moved autonomously without communicating its actions. Using the algorithm explained in Sec. 7.2, the

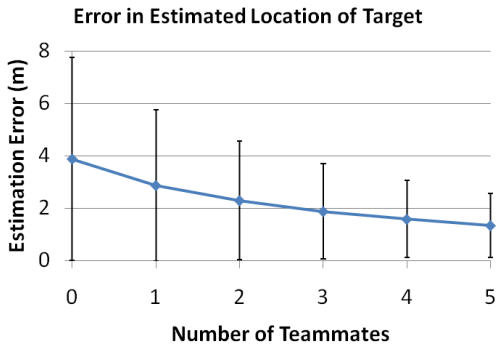



Figure 16: The effect of teammates in estimating the target's location. For each number of teammates, 500 trials were run with random initial positions of teammates and target.

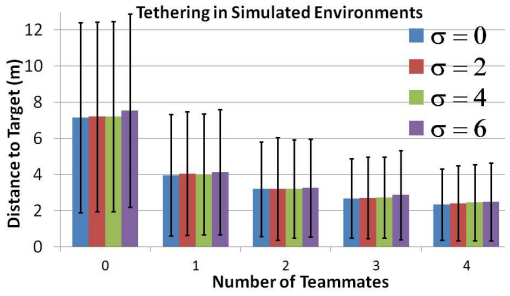
seeker updated a probabilistic model of the target's location. However, since the seeker is unaware of if and when the target moves, as well as the direction of motion, at each time step, the seeker performs a blur on the probabilistic map of the target's location. To do so, the seeker assumes that the target's action $u_{n+1}^{(t)} = move$, such that:

$$\begin{aligned} P(L_2|L_1, move) &= \begin{cases} \alpha & \text{if } L_1 \text{ is adjacent to } L_2 \\ 0 & \text{otherwise} \end{cases} \\ P(L_1|L_1, move) &= 1 - \sum_{L_2 \neq L_1} P(L_2|L_1, move) \end{aligned}$$

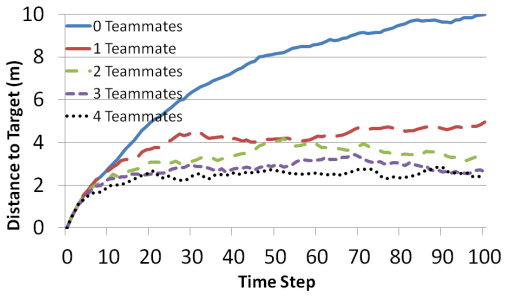
We set $\alpha = 0.2$, varied the number of teammates from 0 to 4, and measured the distance between the seeker and target in the L1 metric. We used the L1 metric because the robots moved in the 4 cardinal directions, so the L1 metric calculates the minimum time steps required for the seeker to reach the target. Fig. 17a shows the average distance between the seeker and target, over 100 time steps, averaged across 250 runs. In each run, the teammates' positions were generated randomly. We also varied the standard deviation of the initial estimate of the robots' locations, from $\sigma = 0$m to $\sigma = 6$m.

The results show that increasing the number of teammates reduces the average tethered distance, even when the initial estimate of the locations is imperfect. Thus, the seeker is capable of following the target as it moves in the world, even though the target does not inform the other robots when it moves, and the direction that it moves in. Further, the probabilities used by the target to decide its motion are also unknown to the seeker; the seeker uses a general blur to capture all possible motions of the target in each time step.

Fig. 17b shows the average distance between the seeker and target as a function of time steps, when $\sigma = 0m$. The target moves randomly in the world and the seeker follows it. When there are no teammates in the world, the distance between the seeker and target quickly rises, and reaches a steady-state of $9.7 \pm 6.2$m after 84 time steps. As the number of teammates increases, the seeker achieves a steady-state more quickly and at a closer distance (e.g., $2.5 \pm 1.9$m after 42 time steps with 4 teammates). Thus, increasing the number of teammates enables faster convergence and better tethering.



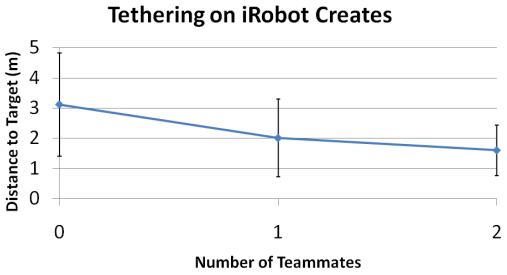

(a) The average tethered distance to the target robot in simulated environments, over 250 trials of 100 time steps each, with $\sigma$ varying from 0m to 6m.




(b) Tethered distance between the seeker and the target over time in simulation.

Figure 17: Tethered distance between the seeker and the target in simulation.

(a) Multiple iRobot Creates deployed in a real office.




(b) The average tethered distance between the seeker and target robots in a real office environment, over a period of 10 minutes for each trial.

Figure 18: Tethering experiments with real robots.

## 9.2 Measuring Real Robot Performance

We implemented the tethering algorithm on the iRobot Creates, and evaluated the performance in a real office (Fig. 18a). The seeker and target robots were placed 1.5m apart, and the seeker's initial estimates of the other robots had a standard deviation $\sigma = 2.5m$. The target robot traveled in a straight line, while the seeker estimated its location and tethered to it. We marked the location of the seeker and target robots every minute for a total of 10 minutes in each trial. Due to the limited memory and CPU on the robots, the experiments were limited to a 10m $\times$ 10m area.

We varied the number of teammates from 0 to 2, and ran 5 trials for each configuration of the robots; there were 2 possible configurations of a single teammate. Fig. 18b shows the average distance between the seeker and target. The results clearly show that as the number of teammates increases, the average distance decreases, which is similar to the results in simulation. The average distance is a smaller number compared to the simulation results due to the environment's size — the real robot experiments were performed in a 10m $\times$ 10m world while the simulations were 30m $\times$ 30m.

# 10 Conclusion

We addressed the challenging problem of coordinating multiple robots to position themselves in an unknown environment to enable connectivity of a static set of communication nodes. One of the main technical contributions of the algorithm is its fully distributed nature. Robots explore the environment simultaneously and separately, each maintaining a record of the connectivity information associated with its own visited positions in its own frame of reference. When within range, robots share the connectivity information corresponding to their positions with no need for map merging. The robots can separately determine that there is a solution at some visited configuration. Another technical contribution is the network graph and network macro graph data structures that are used by the algorithm to share information and represent solutions. The macro network graph abstracts the physical details of the world, while keeping enough information such that a solution can be found efficiently without map-merging.

The fact that our algorithm is fully distributed and requires no map-merging is, in our view, very interesting. It allows the robots to use different granularity and even different representations of the environment, i.e., one robot may use a grid-based representation, while another may use some topological map. Any additional information shared among the robots, if common references are available, can easily be accommodated by the algorithm and could be used in the distributed exploration. It is worth mentioning that the process by which the network structure is tracked and maintained allows the robots to completely separate the problem of navigation from the problem of tracking the network. Therefore the existence of unknown obstacles in the environment has no impact on the performance guarantees of the algorithm.

We contribute an RSSI-distance model that is created from real-world RSSI data. The model provides the minimum and maximum distance given RSSI, as well as the minimum and maximum RSSI given distance. The model does not require any prior knowledge of the environment, i.e., the configuration and types of walls and obstacles, nor does it assume open-space between the robots. The model is capable of functioning in unknown environments, as long as the data used to create the model generalizes to the new environment.

We formally defined the multi-robot tethering domain and problem, and contribute our 2-step algorithm to locate and tether to the target. The seeker robot first moves in a pre-determined path in the environment, collecting RSSI information from the multi-robot team. After the motion is completed, all the information is collected and constraints on the joint-locations of the robots are created. By parsing through the constraints iteratively, valid hypotheses of the joint locations of all the robots, including the target robot, are created efficiently. Once the target robot's location is found, the algorithm uses a probabilistic approach to update its location based on the RSSI information of the multi-robot team. The seeker robot then heads towards the estimated position of the target robot, and remains tethered to it.

Extensive experiments were performed in simulation to demonstrate the efficacy of our distributed algorithm to achieve connectivity. We varied the exploration heuristics and the number of robots and showed that the *stay-at-towers* heuristic achieved good results. We also studied the effect of teammates on the tethering problem, through extensive experiments in simulation, and with real robots in an actual office environment. We showed that as the number of teammates increases, the estimate of the target robot's initial location becomes more accurate, the seeker robot remains more closely tethered to the target, and the time required for the tethering distance to reach a steady state decreases.

## Acknowledgments

This work was partially supported by the Lockheed Martin, Inc. under subcontract 8100001629/ 1041062, the Air Force Research Laboratory under grant no. FA87501020165, and the Agency for Science, Technology, and Research (A*STAR), Singapore. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity. The authors thank Francisco Melo and Daniel Borrajo for contributions of the initial part of this work. This article is published in *Progress In Artificial Intelligence*, 2012, DOI: 10.1007/s13748-011-0007-1 — http://www.springerlink.com/content/p1147gk8qvp3572g/. The original publication is available at www.springerlink.com.

## References

[1] J. Biswas and M. Veloso. WiFi Localization and Navigation for Autonomous Indoor Mobile Robots. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 4379–4384, 2010.

[2] J. Butterfield, K. Dantu, B. Gerkey, O. Jenkins, and G. Sukhatme. Autonomous biconnected networks of mobile robots. In *Proc. 6th Int. Symp. Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops*, pages 640–646, 2008.

[3] H. Hashemi. The Indoor Radio Propagation Channel. In *Proc. IEEE*, volume 81, pages 943–968, 1993.

[4] A. Howard, S. Siddiqi, and G. Sukhatme. An Experimental Study of Localization using Wireless Ethernet. In *Proc. 4th Int. Conf. Field and Service Robotics*, pages 142–153, 2003.

[5] A. Howard, L. Parker, and G. Sukhatme. Experiments with a large heterogeneous mobile robot team: Exploration, mapping, deployment and detection. *Int. Journal of Robotics Research*, 25 (5-6):431–447, 2006.

[6] T. Jung, M. Ahmadi, and P. Stone. Connectivity-based Localization in Robot Networks. In *Proc. Int. Workshop Robotic Wireless Sensor Networks*, 2009.

[7] S. Koenig and B. Szymanski. Value-Update Rules for Real-Time Search. In *Proc. 16th Int. Conf. Artifical Intelligence*, pages 718–724, 1999.

[8] D. Kurth, G. Kantor, and S. Singh. Experimental Results in Range-Only Localization with Radio. In *Proc. Int. Conf. Intelligent Robots and Systems*, pages 974–979, 2003.

[9] S. Lanzisera, D. Lin, and K. Pister. RF Time of Flight Ranging for Wireless Sensor Network Localization. In *Proc. Int. Workshop Intelligent Solutions in Embedded Systems*, pages 1–12, 2006.

[10] T. Li, S. Chang, and W. Tong. Fuzzy Target Tracking Control of Autonomous Mobile Robots by Using Infrared Sensors. *IEEE Transactions on Fuzzy Systems*, 12(4):491–501, 2004.

[11] C. Liu, K. Wu, and T. He. Sensor localization with Ring Overlapping based on Comparison of Received Signal Strength Indicator. In *Proc. IEEE Int. Conf. Mobile Ad-hoc and Sensor Systems*, pages 516–518, 2004.

[12] L. Ludwig and M. Gini. Robotic swarm dispersion using wireless intensity signals. In *Proc. Int. Symp. Distributed Autonomous Robotic Systems*, pages 135–144, 2006.

[13] A. Matveev, H. Teimoori, and A. Savkin. The Problem of Target Following Based on Ranged-Only Measurements for Car-like Robots. In *Proc. IEEE Conf. Decision and Control*, pages 8537–8542, 2009.

[14] N. Michael, M. Zavlanos, V. Kumar, and G. Pappas. Maintaining Connectivity in Mobile Robot Networks. In *Proc. Int. Symp. on Experimental Robotics*, pages 117–126, 2008.

[15] S. Poduri and G. Sukhatme. Achieving Connectivity through Coalescence in Mobile Robot Networks. In *Proc. Int. Conf. on Robot Communication and Coordination*, pages 4:1–4:6, 2007.

[16] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket location-support system. In *Proc. 6th ACM Conf. Mobile Computing and Networking*, pages 32–43, 2000.

[17] J. Reich, V. Misra, D. Rubenstein, and G. Zussman. Spreadable connected autonomic networks (SCAN). Technical Report TR CUCS-016-08, CS Department, Columbia University, 2008.

[18] M. Saxena, P. Gupta, and B. Jain. Experimental Analysis of RSSI-based Location Estimation in Wireless Sensor Networks. In *Proc. Int. Conf. Communication System Software and Middleware*, pages 503–510, 2008.

[19] M. Schwager, J. McLurkin, and D. Rus. Distributed coverage control with sensory feedback for networked robots. In *Proc. Robotics: Science and Systems*, pages 49–56, 2006.

[20] S. Zickler and M. Veloso. RSS-Based Relative Localization and Tethering for Moving Robots in Unknown Environments. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 5466–5471, 2010.

[21] V. Ziparo, A. Kleiner, B. Nebel, and D. Nardi. RFID-based exploration for large robot teams. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 4606–4613, 2007.