Control for Localization and Visibility Maintenance of an Independent Agent using Robotic Teams

Ethan A. Stump
University of Pennsylvania, estump@seas.upenn.edu

Control for Localization and Visibility Maintenance of an Independent Agent using Robotic Teams

Abstract
Given a non-cooperative agent, we seek to formulate a control strategy to enable a team of robots to localize and track the agent in a complex but known environment while maintaining a continuously optimized line-of-sight communication chain to a fixed base station. We focus on two aspects of the problem. First, we investigate the estimation of the agent's location by using nonlinear sensing modalities, in particular that of range-only sensing, and formulate a control strategy based on improving this estimation using one or more robots working to independently gather information. Second, we develop methods to plan and sequence robot deployments that will establish and maintain line-of-sight chains for communication between the independent agent and the fixed base station using a minimum number of robots. These methods will lead to feedback control laws that can realize this plan and ensure proper navigation and collision avoidance.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Mechanical Engineering & Applied Mechanics

First Advisor
Vijay Kumar

Keywords
multirobot system, control for localization, visibility, deployment, target-tracking

Subject Categories
Controls and Control Theory | Robotics

CONTROL FOR LOCALIZATION AND VISIBILITY MAINTENANCE OF AN INDEPENDENT AGENT USING ROBOTIC TEAMS

Ethan Stump

A DISSERTATION

in

Mechanical Engineering and Applied Mechanics

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2009

Acknowledgements

Finishing this dissertation would not have been possible without help from many people through the years, and I would like to acknowledge their contributions. I first wish to thank my advisor, Vijay Kumar, for his infectious enthusiasm that made me want to re-examine problems I had given up on, for his willingness to share his bountiful knowledge whenever asked, and especially for his sage-like patience.

I also wish to thank my committee for their time, attention, and suggestions. In particular, I thank Volkan Isler for his insight and contributions.

Many people were instrumental in helping me complete various projects for publication or personal research. For this, I wish to thank Megan Arenson, Christian Moore, Pedro Shiroma, Jon Fink, Dimitris Theodorakatos, and Dan Gomez-Ibanez. I reserve special thanks for two peers: Benjamin Grocholsky, for helping to shape my world-view through countless hours of work and discussion and for getting me to live a little; and Nathan Michael, for always listening to me and trying to understand me and imparting his wisdom unconditionally.

Finally, I wish to thank my family and non-academic friends for all of their support through the years, especially Kate and Jim Civello for being a second home, and my parents, Glenda and Thad, for always providing love and encouragement, even from 3000 miles away, and always when I needed it the most.

ABSTRACT

CONTROL FOR LOCALIZATION AND VISIBILITY MAINTENANCE OF AN INDEPENDENT AGENT USING ROBOTIC TEAMS

Ethan Stump

Vijay Kumar

Given a non-cooperative agent, we seek to formulate a control strategy to enable a team of robots to localize and track the agent in a complex but known environment while maintaining a continuously optimized line-of-sight communication chain to a fixed base station. We focus on two aspects of the problem. First, we investigate the estimation of the agent's location by using nonlinear sensing modalities, in particular that of range-only sensing, and formulate a control strategy based on improving this estimation using one or more robots working to independently gather information. Second, we develop methods to plan and sequence robot deployments that will establish and maintain line-of-sight chains for communication between the independent agent and the fixed base station using a minimum number of robots. These methods will lead to feedback control laws that can realize this plan and ensure proper navigation and collision avoidance.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis, we will approach the problem of using mobility as an enabling concept in the localization and communication-maintenance of deployments of robots. This is a reversal of traditional considerations: usually localization and communications are subproblems that are addressed or delegated in order to facilitate the main objective of getting robots to go somewhere of interest. However, this is part of a larger philosophical shift in using robots as a building block for information-gathering systems where we are not concerned with how the robots accomplish the gathering, just that they do so in a way that maximizes the quality of the information. Robots represent a unique way of accomplishing this because they provide a means to reduce the cost of information or scale the information-gathering to boost the quality.

In order to make this statement clear and motivate our problem, we will consider the use of robots in First Responder emergency teams [53]. Whenever a disaster occurs, an appropriate response is to contain the damage and search and extract survivors, and this search-and-rescue task is handled by a First Responders such as a unit of firefighters. The primary mission is to find survivors as quickly as possible, but this is complicated by the fact that environmental dangers such as gas leaks, unstable structures, and open flame all represent safety hazards for the rescue workers. Because of this, the underlying task for the rescue mission is *information*, not only of the survivor locations but also of the environmental conditions, but this information is costly because gathering it means putting people in harm's way. Robots provide a way to lower the cost of information because the loss of a robot is insignificant compared to the loss of a human life, and they provide a way to enhance the quality of information because they can be deployed in large enough numbers to explore quickly and establish networks for constant environmental monitoring. In this way, robots are instrumental in providing *situational awareness* for the emergency personnel. Accomplishing this requires: organizing robot teams sufficiently enough to provide communications, information integration, and relative positioning information; endowing them with sensors and algorithms for measuring and tracking environmental changes, creating and relaying maps, and searching for survivors.

In [53], three main challenges are identified. First, robots need a way to localize themselves and others since the tasks required (mapping, tracking, etc.) are impossible without some notion of location. Because of environmental conditions, cameras are not a reliable sensor, so the proposed solution is to deploy a network of radio beacons that provide distance information. Doing so introduces the need for algorithms that can perform localization using only range measurements, and the complexities of this will be discussed later. Second, we need a way to understand the flow of information in the system. This means understanding how to place sensors optimally in order to measure and track environment conditions such as temperature and also how to route the information that these sensors provide back to the emergency personnel who need to use it. Third, we need network control, or the ability to coordinate teams of robots to move through the environment while maintaining formations.

We will address parts of the first and third of these challenges but with some variations. In the challenge of localization, the major thrust is focused on how to make use of an established radio beacon network to perform the localization but misses the problem of placing this radio beacon network in the first place. If we insist on using the same ranging technology to localize as we traverse the environment before beacons are placed, then we can no longer use simple, memoryless approaches such as triangulation and instead must create and maintain state estimates that are shaped by measurements taken through motion. In this case, the localization is only possible with motion since static measurements do not have enough information to find a solution. Also overlooked in the presentation but inherent in all of the tasks is the challenge of communication. Without a way to communicate it back to emergency personnel, the information gathered by the robots is useless. With this in mind, we could imagine that the purpose of robot formations in the solution of the third challenge is specifically to address the needs of communication.

This returns us to the original goal of using mobility to enable localization and communications: for our application, localization is impossible without mobility, and we need mobility to establish formations for communication, so the benefit is clear.

1.1 Objective

We will focus on the single objective of localizing and tracking an agent in a complex but known environment while maintaining communications with a fixed base station. We will consider this agent to be an abstraction of a more general user participation in directing the information-gathering action; perhaps it could be a human being entering the environment to perform a rescue or another robot guided by an operator to locations of interest. In accordance with the earlier discussion, we split this problem into two parts: localization and communications.

First, we will estimate the agent's location by using nonlinear sensing modalities, in particular that of range-only sensing, and formulate a control strategy based on improving this estimation using one or more robots working to independently gather information.

Second, we will plan and sequence robot deployments that will establish and maintain line-of-sight chains between the independent agent and the fixed base station. We choose line-of-sight as a simplification of the communication process for reasons we will discuss later, but our approach could encompass a more general model. These methods will lead to feedback control laws that can realize this plan and ensure proper navigation and collision avoidance.

1.2 Literature Review

In order to justify our approach, we must investigate existing solutions to our stated problem. This is an ambitious undertaking because the problem set cuts across a wide swath of existing research. We will first consider how to use range measurements to solve the nonlinear estimation problem and how to choose control actions to improve estimate quality. Beyond this, our main focus will be on solving the task of deploying and moving formations of robots to maintain communications by maintaining line-of-sight. We will insist on line-of-sight after considering more general communications models. Two basic approaches will be investigated: viewing this is a formation-control problem, and viewing this as a pursuer-evader problem where we must keep a potentially adversarial target within our visibility region. Along the way, we will see some connections with research in active sensor network deployment and introduce Steiner trees as a tool from graph theory applied to our problem of finding formations.

1.2.1 Localization

Practical estimation tasks require us to deal with nonlinearities that are inherent in process dynamics and observation models. Common solutions to deal with such nonlinear state transition and measurement models require linearization of at least some portion of the problem. The concern is that linearization can lead to inconsistent error handling, removes the ability to directly represent ambiguous confidence sets, and cannot be applied when the underlying state is unobservable. As a targeted example, consider our use of range-only measurements as seen in Figure 1. The position of the target relative to the observer can be anywhere in an annulus around the observer, but using a linearized technique requires us to choose one particular position to linearize around. Any such position will necessitate throwing away most of the potential target locations, and so without good initialization, the filter will quickly become inconsistent.

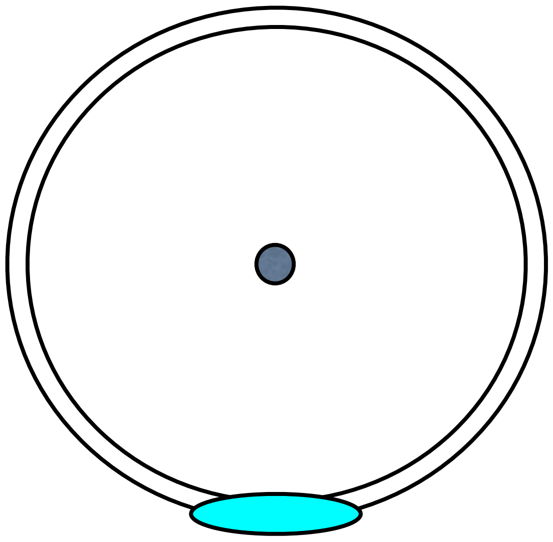


Figure 1: With only range information, a target could be located anywhere within an annulus around the observer. However, using a linearized technique would force us to choose a particular location to linearize about, discarding most of the potential target locations and quickly leading to inconsistencies unless we have good initialization.

Many techniques have been proposed to either avoid or delay linearization. Some of the most popular in recent years are sampling-based approaches such as Monte Carlo Localization [28]. They rely on estimating a probability distribution for the system state, but instead of maintaining a simple parameterized distribution, which may require linearization, the distribution is discretely sampled to allow for arbitrary densities. Further refinements have allowed the fusion of sampled representations with standard parameterized ones to solve more challenging problems such as simultaneous localization and mapping tasks [71].

The opposite approach is to delay linearization by simply storing all measurement and state updates until a later time when the larger data base allows for the use of consistency to improve the estimates. The GraphSLAM algorithm [101] is an example of such a technique; after several measurements have been taken, an estimate is formed by iteratively linearizing the state propagation and measurement equations and solving a least squares problem to maximize agreement with measurements and problem dynamics. Several different variations on this theme have been used [27, 49].

The disadvantage of such delayed measurement integration is that best estimates are not available during data acquisition, making it impossible to knowledgeably improve the collection process. As a compromise, the Sparse Extended Information Filter [101] has been proposed to proactively incorporate each measurement as it is taken while being able to explicitly manage the information links between different entities formed through measurements and motion. The final picture allows for intuitive identification of how features are related but still requires linearization.

For many applications, such linearization may prove to be acceptable, but not for our current application of localization using range-only measurements. In the most general form of this problem, there is no sense of direction and so any attempt to linearize a range measurement will likely result in crippling inconsistency after further measurements and motion. Compelling sampling-based estimation approaches to this problem have been demonstrated [23], but such implementations may require large numbers of samples making them computationally unattractive.

An important evolution in the methodology of Information-type filters was presented in [38]. The central idea is a nonlinear embedding, sometimes referred to as an over-parameterization, that maps the system states into a extended state space in a way such that the measurement equations become linear. The resulting framework lends itself to the application of the methodology introduced in [88] for the field of dynamic estimation under bounded noise. Successful application of such set-based estimation techniques to static localization tasks involving range-only measurements and relative bearing measurements are demonstrated in [11]. While set-based techniques have been investigated by other researchers [4, 65, 39, 90], the over-parameterization allows exact representations in the extended state space.

This over-parameterization technique does have other presentations in the literature. In [59], the Non-Minimal-State Kalman Filter is presented as a tool very similar to the extended state space. They continue to deal with Gaussian distributions, whereas this work adopts set-based conventions, but the basic idea of augmenting the state with extra nonlinear terms to make the system equations linear remains. In a related vein, by considering the underlying partial dynamic equations governing the evolution of probability distributions in dynamic systems, In [18], a class of Kalman-like filters are derived that use different sufficient statistics than the traditional mean and covariance in order to capture more exotic distributions that arise under nonlinear systems. The sufficient statistics that are found end up looking remarkably similar to the over-parameterized or augmented-state filters presented in this work and the others above.

In a system where we have control over some of the sources of estimation, it is natural to ask how we can apply control actions to maximize the acquisition of information about the unknown states. In [35], the *mutual information gain* of a particular control action was used as a metric to determine the desirability of that action. Maximizing this metric at each step resulted in rapid shrinking of the estimation uncertainty. Moreover, when the estimate state was shared between multiple sensing agents, the application of this local, greedy control action to each agent resulted in cooperative behavior with no explicit coordination. When it came time for each agent to make a decision, it would naturally try to move to provide information in directions that other agents were not able to.

1.2.2 Communications-Maintenance

Though wireless communications are ubiquitous in mobile robot applications, any radio communications methodology presents a host of difficulties that are rarely addressed and poorly understood. Though we care only about information transmission, the transmission capacity of a link is fundamentally tied to the strength of its wireless signal, and this signal strength is in turn affected by a variety of effects spanning the conceptually simple, such as range-based power loss, to the truly bizarre, such as multi-path cancellation and boosting [92]. The result is that signal strength can change dramatically when the receivers are moved by only several centimeters and practitioners prefer to understand such effects using statistical models such as Rayleigh fading [78]. It is important to note that these models are most applicable when the sender and receiver do not have direct line-of-sight. In the robotics community, usually these small-scale effects are abstracted away, but [63] devises a way to sample the Rayleigh distribution around a point of interest by driving a receiver through defined patterns. In [62], the problem is shifted slightly by working with a fixed trajectory but modulating the velocity along this trajectory to slow down during periods where the signal strength is strong in order to pass along information that was buffered during periods where the signal strength was too poor to allow meaningful reports.

One common theme is to consider communications as an unknown function, dependent on the positions of the agents involved, that must be optimized but can only be sampled. We can consider a generic example of representing this, given by [31], where an indicator function $R(i, j) \in \{0, 1\}$ signifies if there is a communications route between agents i and j and $Q(i, j)$ measures the quality of communication between them. The total communications utility of a deployment of robots is found by:

$$U = \sum_i \sum_j \left[\alpha R(i,j)Q(i,j) - \beta \neg R(i,j)\right]$$

and so the quality depends on the sum of the qualities of all point-to-point routes but is penalized whenever robots are unable to communicate. This problem is hard to optimize explicitly because the decision of whether a route exists and the quality of the route can depend on the states of agents deep within the network, leading to the need to search for solutions combinatorially. In [31], they appeal to research on distributed constraint optimization [64, 69] and auction-based techniques.

In [22], rather than treating the communications as a complete black-box, they assume a model where unknown point sources create localized signal degradation. In order to optimize the communications, they perform local function optimization based on Extremum-Seeking control [2], a technique that provides a control theory foundation for the question of how to choose controls to optimize an unknown function. They were specifically considering implementations on UAVs, so the exploration of functional values near a set point was accomplished automatically by the necessary orbiting that airplanes must perform for station-keeping, but they recognize that for ground robot applications, such constant motion could lead to excessive power drain.

Measuring an unknown spatial function is a common application for sensor networks. Their utility for the First Responder scenario has already been identified [47]. Though the literature on sensor networks is vast, we will focus on the direction of leveraging mobility to improve the function estimation. An early technique involved agents deploying to optimally cover a known function by balancing the integral of the function over the Voronoi region enclosing the robot [17]. Though the details were complicated, the concept was simple: in regions where the function was strong, more robots would be deployed to cover since the integral would be higher even though each robot's zone of influence (points closer to that robot than any other robot) was smaller. Such a technique was used experimentally in [85] to cause small Swarm-Bots to converge around a light source. In [84], the previous solution was recognized to be suboptimal if the function was multi-modal, but a “Ladybug” controller was proposed where control directions orthogonal to the desired direction were randomly applied in order to facilitate exploration of the functional space. This technique brings to mind the Extremum-Seeking controller of [2] with oscillatory perturbations applied by [22]. This controller could also address the problem of covering unknown functions, and this line of research was made more rigorous in [86], where the problem was cast as one of finding the parameters of a functional approximation using adaptive control while simultaneously optimizing the coverage.

Other solutions to the communication optimization problem tended towards even simpler communications models but applied to larger networks. A particular heuristic was one of characterizing shorter communication links as inherently better, such as that applied in [81], where each robot would move to the average location of all of its neighbors in an attempt to equalize all of the communication distances. In [33], a similar approach was used but the neighbors were weighted according to their communications capacity. Such a technique is very reminiscent of the formation control problem of rendezvous, where a team of robots is trying to all move to a single location while maintaining distance-based communication links with each other. The best results were obtained using a local controller that drove to circumcenter of all neighboring robots [16, 61]. These results lead to discussion on formation control involving making formal statements about global controllers using local controllers and understanding of the graph relationship of the robots to each other, but we will return to this later.

One particularly notable paper considers the problem of fault-tolerant communications networks [8] by requiring that the graph describing the available communications between agents be biconnected: that is, it remains connected even if a single vertex is removed. Their approach identifies which sets of agents have a biconnected communications graph already and then shifts these blocks towards each other in order to form new connections and create biconnectivity of the full system.

Another interesting direction is to consider the communications problem not as one of positioning but as one of routing, and in particular using mobility not to find better places to communicate from but to actually treat the robots as "active packets" that seek to drive from source to destination or perform shorter trips that pass the packets between several mobile robots [60]. In [7], this was implemented on a large scale by placing wireless communication nodes on city buses and then routing messages between distant pieces of wireless infrastructure by buffering them on buses whose routes passed through both regions.

An even further simplification is to consider the problem over discretized positions. In [50], a very large team of robots, on the order of one hundred, is used to first map and then cover an environment while maintaining communications. Their communication model is to require visibility and limiting the range between agents, and they accomplish this by enumerating all possible robot locations as positions along the Generalized Voronoi Graph (GVG) of the environment. The GVG is a 1-dimensional set of points in a planar environment that are equal distance from all the walls, as in Figure 2. It is readily computed for 2D and 3D environments [14, 74, 103] and it provides a convenient topological representation of the environment that is useful for a motion planning technique known as *retraction planning* [55]. With the GVG computed and broken up into equally-spaced "slots", each possible robot slot has its visibility and distance checked to all others, and the robots are deployed only in slots which are connected.

Such a discretization was also used in [100] in order to allow for the problem of communications-maintenance to be cast as a dynamic program. The authors deliberately provide no communications model but only assume that one could be applied to decide if communication were possible between two points of the discretization. They then solve two problems: first to move robots to allow an agent to remain

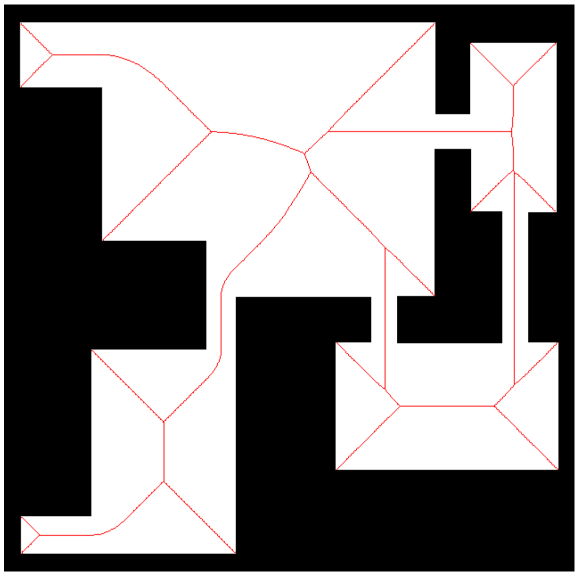


Figure 2: An example of the Generalized Voronoi Graph (GVG) for an environment. It is a 1-dimensional graph consisting of points whose set of closest wall points is size two or larger. The result provides a convenient topological representation of the space.

in communication at all times, and second to plan trajectories that maximize the escape time for an agent to leave communications, effectively bounding against the worst possible agent motions. What they have developed is essentially an approach for solving the pursuer-evader problem with a generic connectivity model; we will return to pursuer-evader problems later.

As we found by considering realistic radio models and applications involving the optimization of unknown communication-quality functions, communicating is difficult. We are best served by using a simple communications model because realistic models require a lot of tuning and sampling. Of the models typically used, range-based and visibility-based, only visibility-based is appropriate for the sorts of environments we are interested in since it is non-line-of-sight effects that create the problem. All of the research involving range-based models is taking place in open-field environments where the researchers can conveniently ignore this fact.

Formation Control

As was hinted at before, in recent years, there has been a great deal of research centered around developing controllers for groups of agents where inter-agent relationships form an explicit basis for action. The general theme of the research is to model the inter-agent relationships as edges in a graph structure describing the group and then make statements about stability of actions of the ensemble where control actions are chosen locally by considering only an agent's direct neighbors.

This methodical transition from local-to-global in the context of proving global properties from local relationships had some genesis in the study of coupled nonlinear oscillators [54] ("Kuramoto oscillators") and the observation that arbitrary interconnections between oscillators enable them to synchronize phase provided that the graph structure describing the connections is connected (see for example [45]). The consideration of phase led naturally to consideration of the heading of moving agents and so attention turned to "flocking" behaviors of agents seeking to synchronize their heading and velocity [44]. Further work established the properties of such control schemes for fixed and dynamic topologies [99, 97, 98] and for 3D heading control [70]. The fundamental message of this work is that formation stability can be obtained if each agent looks at the velocities of its neighbors and then steadily modifies its velocity to match theirs; the use of leaders that dictate the heading and velocity of the formation is natural since they are simply agents that do not employ a feedback law.

Though the provable transition from local controls to global properties is useful, it is desirable to make statements about goals more sophisticated than synchronizing headings. From this desire came the idea of what we shall refer to as *network graph control*, or control arising from optimization or maintenance of a property expressed over a graph structure. This is analogous to a traditional potential field or Lyapunov control approach but where the controls applied are local and we make use of the established theoretical machinery in order to make statements about their effect on the global property.

The standard setup is as follows: consider a set of robot agents, $\{V_i\} \subset \mathcal{V}$, a mapping, $m : \mathcal{V} \to \mathbb{R}^2$, that indicates where each agent is located in space, and then a rule that dictates whom each agent may interact with. A typical rule is a distance-based one:

$$A_{ij} = \begin{cases} 1 & \|m(V_i) - m(V_j)\| \leq R \\ 0 & \|m(V_i) - m(V_j)\| > R \end{cases}$$

where A_{ij} indicates the *adjacency* of robot i to robot j, with 1 meaning adjacent. R is an interaction radius. The interaction rule leads to an abstract graph structure, $G = (V, E)$, with interactions creating edges (E) between nodes (V) denoting agents.

A generic control strategy would be to first define some quantity $c_G(m_k)$ based on the graph structure and the current embedding m_k at time k. Then control actions u_k will modify m_k (and perhaps the structure) and we seek to either optimize c_G ($\max_{u_k} c_G(m_{k+1}(u_k))$) or constrain it (choose u_k s.t. $c_G(m_{k+1}(u_k)) \geq 0$). All of the following references use different choices of c_G or multiple versions of c_G simultaneously.

One option is to control the connectivity of the graph structure by considering powers of the adjacency matrix formed from the A_ij entries describing the interaction of agents. If the entries of A^k are all non-zero, then any two agents in the system interact through k or less intermediary interactions. Put another way, any two agents are connected through k or less hops. In this case, define $c_G = A^K$ and choose actions that constrain $A^k \succ 0$ (greater than 0 in all entries). In [106], the authors accomplish this approach by smoothing out the adjacency definitions with sigmoids to make the terms differentiable and then find control directions that prevent the system from losing connectivity.

Another option is to control the connectivity of the graph structure by controlling the eigenvalues of the graph Laplacian (specifically the 2nd-smallest eigenvalue λ_2, also known as the *Fiedler value*). The Graph Laplacian is defined as $L = D - A$, where D is a diagonal matrix of the row-sums of A. When $\lambda_2 = 0$, the graph is

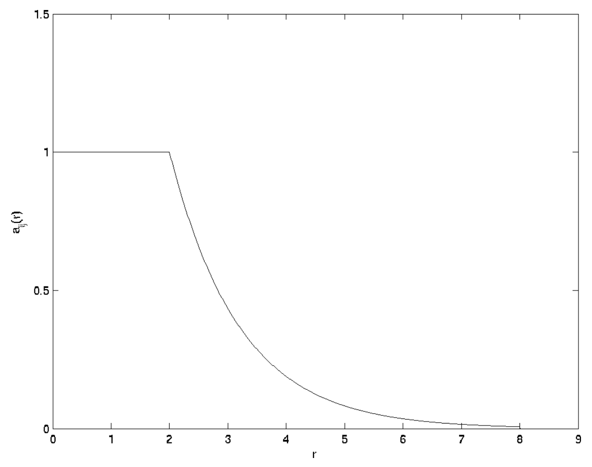


Figure 3: An interaction function based on distance between two agents.

not connected, and the magnitude of λ_2 indicates the connectedness of the graph in terms of number of connections and their distribution (bottlenecks reduce the Fiedler value, for example). However, if the adjacency matrix takes on values on a continuum between 0 and 1, then we have a weighted graph Laplacian, and then Fielder value $\lambda_2(L)$ now expresses not only the degree of connectivity of the graph but also the aggregate quality of the connection [32]. Such an approach allows for interaction *quality* to be incorporated; this could be useful for expressing that a certain radio communication channel has more throughput when the agents are closer together (value is 1) but becomes less useful as their distance increases, until no communication is possible beyond a certain cutoff radius (value is 0). With such a quality indication in place, we can ask questions about the quality of communication for the entire network of communication channels.

An example of an interaction function leading to a weighted Laplacian is seen in Fig. 3, used in [20]. With such a weighted function dictating the entries of the adjacency matrix, as λ_2 increases, either the amount of interconnectedness increases or the individual connections become higher quality. In [20], λ_2 is found as a continuous function of the entries of L (and hence the relative positions of the agents) and so is maximized directly with the added twist of being able to calculate the derivatives of the state-dependent λ_2 using distributed consensus [10].

An alternate formulation of a control based on maximizing λ_2 is in [48] where the authors use semi-definite programming at discrete intervals to solve for agent position updates to maximize the state-dependent λ_2. There is an important connection here with sensor networks because [48] is addressing the problem of placing the nodes of a sensor network to enhance the total communications capabilities by increasing the connectivity.

In [107], the authors take a middle approach: they are interested in λ_2 but only to make sure it is non-zero. To this end, they define the quantity $det(P^T LP)$, yielding the product of eigenvalues $\prod_{i=2}^{N} \lambda_i(L)$. The first-smallest eigenvalue of L is always zero and corresponds to the eigenvector of all ones, $\mathbf{1}$; because of this, the Laplacian has this spectral component removed by choosing P such that the columns are orthogonal to $\mathbf{1}$ and each other. Because $\lambda_2 = 0$ implies that $det(P^T LP) = 0$, they seek to minimize the potential function $\phi_c = 1/det(P^T LP)^a$ using a simple gradient control law; in this way they indirectly maximize λ_2 while ensuring that λ_2 never goes to zero because otherwise the potential function would go to infinity.

In [93], the precursor work to this thesis, a combination of these approaches is used, specifically using the criteria of both [20] and [106], maximizing λ_2 while constraining that the system remain connected. It was necessary to use both because loss of connection was observed using only a λ_2-maximizing controller in the context considered. The problem was to maintain a connected network between two leader agents, one that moves and one that stays in a fixed location. With the pure gradient control, the agents would prefer to just clump around the fixed node and make fully connected network, only to realize that the moving node was breaking connection after it was already too late to react. The problems were further complicated since the formulation was taking place inside a maze-like environment and the walls were providing further motion constraints. However, there was no explicit planning, and the greedy approach quickly failed when the agents would get stuck in local minima arising from the wall constraints. Such a sequence is seen in Fig. 4.

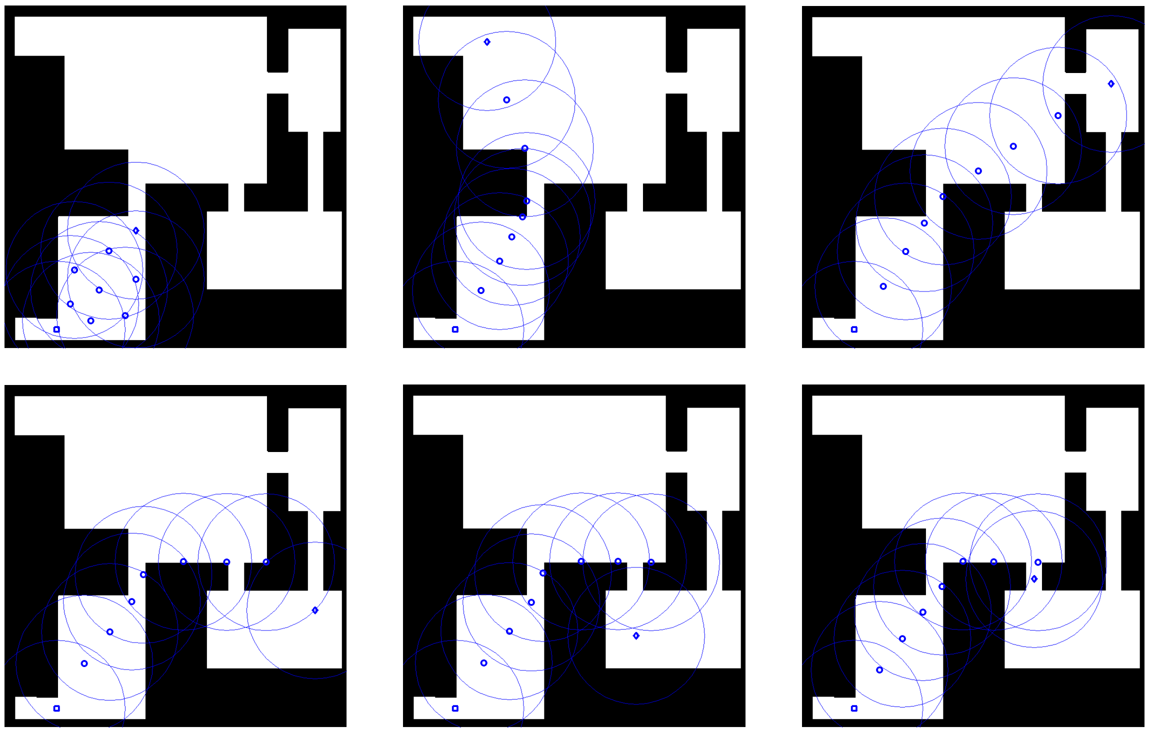


Figure 4: Six snapshots of the robot team motion explored in [93], starting from the initial configuration in the upper-left panel and proceeding from left-to-right in chronological order. Note that in panels 4 and 5, connectivity was lost because of the robots getting stuck against walls. The control is a combination of the λ_2-maximizing controller of [20] and the k-connectivity maintenance of [106], with additional motion constraints from the walls.

As a maturation of their previous work in [106, 107], the authors of [108] envisioned an explicit topology-controlled system. Given a graph that indicates the desired interactions between agents, each pairwise interaction is given a potential function based on the distance between the two agents:

$$\phi_{ij}(r_{ij}) = \frac{1}{\|r_{ij}\|^2} + \frac{1}{R^2 - \|r_{ij}\|^2}$$

and controls are chosen to follow the negative gradient of the total potential summed over all edges of the graph: $\phi = \sum_{(i,j)\in E} \phi_{ij}$. However, the key component is the ability for the system to modify its own topology and ensure connectivity. From the potential above, the value R places a maximum allowable distance between agents, but the authors define a new radius $\rho < R$ where new links are created between agents if they happen to get closer than ρ. Any links with length between ρ and R are candidates for deletion, and the network of agents uses an online auction procedure to decide one-by-one which links are safe to delete. The difference in distance between when links are added and when they are considered dead provides a hysteresis that prevents links from being continuously added and then negotiated away. The total control strategy splits nicely between two layers: a layer that chooses actions to maintain the current graph structure, and a layer that negotiates with the other agents to modify the graph structure.

Most importantly, this scheme was shown to work in experiments [68], overcoming the challenges of implementing a distributed auction with real communications delays.

The formation control literature forms an important foundation for addressing any problem involving control of teams of robots, but as a basis for implementing communication controls, there are two very fundamental pieces missing. First of all, the interaction models are entirely radial-based, and there is no attempt to expand beyond this simplification. A simple modification would be to replace the interaction function of 3 with one based on visibility, but this function does not enjoy all the same benefits of differentiability or even continuity. Second of all, most of the emphasis is on obtaining theoretical results of stability and similar qualities, and considerations of the effects of obstacles, other than the robots themselves, are not addressed. We have attempted [93] to rectify this deficiency but obviously an approach consisting of only local controls cannot always solve a global problem. We need to directly address the question of choosing motions to maintain visibility and also incorporate techniques of robot motion planning to handle obstacles.

Pursuit-Evasion and Visual Tracking

Planning robot motions to ensure visibility between different agents has close connections to the topic of pursuit-evasion problems [1, 37, 43, 95]. In such problems, a pursuer and evader are placed in a complex environment, often with velocity constraints, and the question is asked: Can the evader escape from line-of-sight of the pursuer? Different researchers pose the problem in different ways; sometimes the decision problem is tackled, other times constructive paths are desired, and sometimes we are more interested in what the pursuer must do to avoid escape.

One example of the decidability problem is presented in [72]. The authors consider a pursuer that is required to keep visibility to the evader and keep its distance within given lower and upper bounds. The problem is elegantly treated as one of trying to move a variable-length rod through a polygonal environment, a specialization of the classic Piano Movers' Problem. Solution techniques for the fixed-length rod problem were first presented in [87], and were integrated into the larger picture of robotic motion planning in [55]. Their techniques begin by representing the configuration space as the (x, y) position of one end together with the θ angle of the rod. Environment obstacles and walls create constraints on the configuration space, limiting the available angles when the rod is in certain positions or barring the position altogether if it lies within an obstacle. The key realization was that there are certain curve boundaries within the (x, y)-space where the angular constraints appear and disappear; these *critical curves* emanate from environment polygon segments and vertices in different combinations and were all classified by type. These curves separate the environment into *non-critical regions* where the topology of the boundary constraints remains fixed. By overlaying all of the critical curves of an environment and splitting it into all non-critical regions, the configuration space can be discretized in terms of these regions, and a graph can be constructed where the nodes are the pieces of configuration space and edges join pieces where it is possible for the fixed-length rod to move from one configuration to the other smoothly. The pursuer-evader problem is then reduced to determining if there is a “dead-end” in this region graph.

The authors of [72] were then able to fold in the concept of a variable-length rod by making the observation that the region graph only changes at a finite set of rod lengths corresponding to the times when new critical curves appeared and disappeared. By finding the region graphs for lengths between each of these special lengths, all of the graphs can be tied together into one larger graph that takes into account not only position and orientation, but also rod length. The decidability problem is then similarly answered by searching for “dead-ends” in this larger graph. The authors then go on to relate velocity constraints on the pursuer/evader to limits on the rate of angular change of the rod and tie these in to the question of escapability.

In [56], cases of evaders with known and unknown trajectories are covered. For known evader trajectories, the environment is discretized and the sequence of moves that the pursuer must take to maintain visibility is found using dynamic programming. For unknown trajectories, the authors assume that the evader’s motion at least has a probabilistic model. At each step, the pursuer seeks to maximize the probability that it will be able to see the evader at the next time step. An example is given of a simple bounded velocity model, where the motion model is a probability function of uniform density over a disc centered at the current evader position. The pursuer maximizes the probability of future detection by moving to maximize the area of this disc that is covered by its sensor visibility. We already saw that [100] was another version of this dynamic programming approach that considered a more generic communications-maintenance task.

However, this brought a new paradigm into use: working directly with the visibility-constrained sensor footprint of the observer. In [56] this footprint was used to cover as much of the evader’s potential motion as possible, but the authors suggested as future research the use of *escape time* as a possible metric to work with. This concept was explored more fully in [34]. The *escape time* of the evader is a measure of the shortest amount of time it would take the evader to leave the sensor footprint of the observer. For a constant-speed evader, this is usually interpreted as the shortest distance to a *gap* in the sensor footprint. In [34], the authors derive a simple derivative for this shortest distance and then use a gradient controller to continually increase the escape distance for the evader at each time step.

The gradient control of escape time is explored in more depth in [6]. The authors note that gaps in the sensor footprint always originate from a vertex in the environment, and that when the shortest escape path to a gap is found, the motion with respect to its corresponding vertex is of primary importance. There are two competing goals at play: by "swinging around" the vertex, the shortest path length is increased, but by moving closer to the vertex, any future "swinging" motions will increase the path length faster. The key contribution was a more carefully constructed controller that balances short-term payoffs of path length increases with long-term payoffs of better path length increases.

Finally, [9] is a culmination of the investigation of the dynamics of a pursuer and evader maneuvering around a corner. They provide a detailed decomposition of the space around a single corner and provide optimal strategies for both the pursuer and evader for each region and conclude that there is only one configuration where the evader can definitely escape. Though the problem is completely decided for a single corner, they recognize that doing such an analysis for multiple corners remains an open problem. However, researchers have already started looking to implement similar escape time techniques in 3D [5].

A random-sampling approach to the gradient schemes was presented in [73]. At each step, a number of candidate moves for the observer are generated according to a suitable motion model. For each of these moves, the new sensor footprints are found and the minimum escape time of the evader is calculated for each one. The candidate move with the highest such escape time is chosen for execution. Their approach was demonstrated on fairly large-scale environments with thousands of vertices but they managed real-time performance thanks to a number of pre-computation optimizations. In particular, they discretized the environment with a very fine grid and found the visibility polygons from each of these positions. However, they laid a larger grid over the environment, and only the environment features within the larger bin were used for the visibility polygon computation at each point, in order to reduce the complexity. Once each sample was generated, its corresponding visibility polygon was then simply found from the lookup table of pre-computed polygons.

Another reactive approach was suggested by [3], where they build upon their behavior-based robotic control architecture by adding behaviors that allow the robot to drive towards its goal only until it is about to break visibility with another agent. Such a technique will suffer the local minima problems of [93] because there is look-ahead capability to put agents in places that will benefit future visibility.

The decidability of the pursuer-evader problem was once again tackled in [72] but with the goal of finding complexity results rather than an explicit algorithm for decidability. The authors constructed a coarse model of the visibility within the region and then reduced the problem of deciding if there was a path that the pursuer could use to prevent evader escape into a *bottleneck traveling salesman problem*, which is known to be NP-complete.

Their environment decomposition is especially relevant to the topics of this thesis. They used a modified aspect decomposition of the environment, taken from literature on aspect graphs for object recognition [76]. What is striking is that the curves introduced to partition the environment are exactly those used as some of the critical curves for splitting up the configuration space of the fixed-rod motion planning problem [55]. In particular, both the type 3 and type 4 curves are used. In fact, the resulting partition is what would be used when solving a problem for the motion planning of a rod that had unbounded length. The authors of [72] make one addition by including the segments connecting the bitangent vertices of the type 3 curves.

With the partition in hand, the visibility problem is discretized by connecting regions that share *mutual visibility*. Further discussion of this topic is presented later in this thesis.

The resulting decomposition has a very important topological connection, especially in light of the research done on sensor footprints and escape distances to gaps. Every region in the aspect decomposition represents a fixed topology of obstacles and gaps in the sensor footprint. Perhaps the research that most exemplifies this line of thought is that on Gap-Navigation Trees [102]. The authors imagine a sensor that does a ranging sweep of the surrounding area and only reports on discontinuities in range: what is elsewhere considered gaps in the sensor footprint. They then go on to argue that exploration and mapping of the environment can be done entirely in terms of the topological description, i.e. number and ordering, of these gaps. Topological shifts in the gap sensing occur when gaps join and split, and these joins and splits occur precisely when the observer crosses aspect lines in the environment. Thus each region in the aspect decomposition occupies the same position in the Gap-Navigation Tree.

A related problem is that of the Art Gallery. Whereas in our case we are interested in placing robots to maintain mutual visibility and keep known targets in sight, the Art Gallery problem is generally a problem of placing guards in the environment to ensure that any point of the environment is visible to at least one guard. Further variants specify the nature of the environment and whether the guards are required to be able to see each other. The celebrated Art Gallery Theorem [15] says that for a polygonal environment with n vertices, then $\lfloor n/3 \rfloor$ guards are sufficient and sometimes necessary to cover the environment. For orthogonal environments, this bound can be reduced to $\lfloor n/4 \rfloor$ [46]. It is interesting to consider that an art gallery problem is like an optimal sensor network deployment problem where the sensor footprint is limited by visibility.

In the context of deploying robots while maintaining visibility, [30] is an important development. They take simple polygonal environments and devise a decentralized control scheme that drives agents to cover individual pieces of a star-shaped decomposition while maintaining line-of-sight, all without requiring the environment to be specified beforehand. The procedure relies on the inherent tree-topology of the simple environment, however, and they make no attempts to generalize to environments with holes. It would be possible to create tree-like decompositions of environments with holes if there were a mechanism for splitting the region with diagonals, but this raises questions of how to do this optimally. It may be possible to do this with a greedy online approach where splits are created automatically as soon as robots discover that they have closed a topological loop.

However, fundamentally, we do not have an Art Gallery problem because our target set is limited and mobile. Although a full Art Gallery deployment *would* solve the problem, a solution requiring less observers should be possible. For example, the *Maze* environment, seen in Figure 40 of Chapter 4, has 82 vertices, leading to a need for up to 27 robots to cover the visibility for the full environment according to the Art Gallery Theorem [15]. However, using the decomposition of [30] and calculating the mutual visibility graph, as we will discuss in Chapter 3, we find that the graph has a diameter of 6, implying that we can conservatively connect any two points in the environment with 4 intermediary robots. Since we are only interested in retaining visibility to a single agent at a time, using the Art Gallery deployment is too costly in general.

Though pursuit-evasion techniques and visual tracking have provided a more directed solution to the task of maintaining visibility, there are still deficiencies. The chief problem is that all of the presented research is concerned with keeping a single target in sight of a single robot, but does not evaluate the capabilities of these approaches to handle the case when this robot is itself being tracked by another robot or if we have even longer chains of tracking. Moreover, most of these approaches are still ultimately reactive, only performing next-best-move considerations or actions in a limited scope to deal with the current corner. Some, such as [100] begin to address the planning problem by considering the entire space but require additional modifications to handle problems like requiring that there be a chain of communications from the agent back to the base station rather than just ensuring that at least one robot can see the agent. However, the decompositions of [72] seem to offer a way to do both: use a discretization of the environment that is meaningful for visibility yet coarse enough to perform exhaustive search on.

1.2.3 Steiner Trees

Here we must introduce Steiner trees, a tool which will prove useful for finding visibility-maintaining formations in the discretized environment. From graph theory, a tree is any graph where there is only one path from any vertex to another, or equivalently a graph that has no cycles. Given a graph $G = (V, E)$ of vertices V and edges E connecting the vertices, and a set of terminals $K \subset V$, then a *Steiner tree* T is any subgraph of G that is a tree and whose vertices contain K. Usually we are interested in minimizing some quantity, so a *minimal Steiner tree* is one with a minimal edge weight sum. Any vertex of T that is not part of the terminal set K is called a *Steiner vertex* and can be thought of as helping to provide a link between the terminals of K. In this way we can think of Steiner tree as an extension of the well-known concept of a shortest path in a graph; in fact, the shortest path is the minimal Steiner tree for a terminal set of size two. Steiner trees are also used to solve the problem of how to draw the minimal-length lines to connect a set of points in the plane. The problem is referred to as a *Euclidean*, *rectilinear*, or *octilinear* Steiner tree if the lines are unconstrained, constrained to ninety-degree increments, or constrained to forty-five degree increments, respectively; these are also referred to as *geometric* Steiner tree problems. Excellent overviews of the Steiner tree problem and discussions of the algorithms and approximations for solving all of these versions of the problem are available in [42, 80]. We will discuss a few developments here.

The problem is known to be NP-complete. This may be somewhat surprising considering that the shortest path can be found efficiently, but since a tree can have a great variety of topologies, the combinatorial nature of the Steiner tree problem is clear. The Dreyfus-Wagner algorithm [24] is one technique for find the optimal Steiner tree and is based on dynamic programming. We will make use of this algorithm later, and it will be presented more fully in Chapter 3.

Many approximate algorithms have been developed to solve the problem. One early algorithm [96] builds a tree by starting with the shortest path between two terminals and then adding in the shortest path from other terminals one-by-one. Another approach [51] computes the *distance network* over the terminals: a complete graph where the length of each edge corresponds to the length of the shortest path between the corresponding terminals, essentially a compression of the geometry of the original graph into a graph that contains only information about the terminals of interest. Then a Minimum-Spanning Tree [52, 77] is computed over this network and the corresponding edges expanded back into the original graph and pruned to create a tree. A variation of this [66] uses a reduced distance network where we do not included edges in the network between terminals that are far apart in the original graph since they will not end up being connected in the final graph anyways. A more sophisticated approach similar to the full dynamic programming of [24] is found in [82] and uses hypergraphs, or graphs were edges are allowed to be between more than two vertices, in order to build up larger and larger terminal sets up until a fixed precision. In [79], randomization is used to speed up the process.

Some variations of the Steiner tree problem have been proposed and explored. For instance, the degree of a graph is the maximum number of edges connected to a single vertex, and finding the Steiner tree with minimum degree is explored in [29]. For solving the geometric versions, the problem of building a tree in the presence of obstacles was exhibited in [26]. Interestingly, it relies on performing a triangular decomposition of the space in order to help discretize the problem. Another early geometric algorithm is that of Melzak [67]. A notable feature is that it requires the topology to be specified beforehand and then the optimal tree for this topology can be quickly found. In fact, it can be found in linear time [41]. A similar version is also available for using dynamic programming to find fixed-topology Steiner trees over graphs [104] but the authors were pursuing a problem involving finding minimum-delay routes and added enough extra features to make their algorithm exponential. We will find later that the basic version of this problem has polynomial complexity.

There are a few classes of Steiner tree problems that are notably absent but will be useful for solving our deployment task. One variation is finding a *fixed-size Steiner tree*, where the number of vertices to be used is known. There is also no research into problems involving changing Steiner trees or solving for Steiner trees that are minimal with respect to movement from a prior tree. Both of these will be addressed by this thesis.

1.3 Approach

Our objective is to localize and track an agent in a complex but known environment while maintaining communications with a fixed base station. We split this problem into two parts: localization and communication.

For localizing the agent, we are limited to using range-only measurements and are unable to use memoryless approaches such as triangulation because we are dealing with limited number of sensors. We will make use of the over-parameterization techniques of [38] to modify the measurement equations to be linear using a new set of coordinates. This coordinate transformation has geometric significance and allows us to describe the system states as belonging to a coordinate manifold embedded within a higher-dimensional space of our own construction. By classifying measurement errors as hard-bounded, our measurement and update equations take on the character of operations with ellipsoids. As we add new sensing robots, it is simple to fuse the measurement contributions of multiple robots into a shared estimate.

Our contribution is in generating control directions based on the techniques of [35] but applied to this new filter. Since the representation of our estimates is ellipsoids in a linear space, we make analogies to Gaussian techniques and find simple gradient control laws that allow each independent agent to generate motion directions that will cause its future measurements to shrink the estimate ellipsoid as quickly as possible. Each agent can do this independently and achieve cooperation towards reducing estimate uncertainty but without performing explicit coordination to accomplish this.

For the communication problem, we begin by assuming that we have a map available with which to make visibility computations. We will use the decomposition of [72] to produce a discretization of the problem suited to expressions of visibility. We know that the complexity of this decomposition is bounded [91] and have establishd algorithms to compute it [13, 21, 25]. A mutual visibility graph is constructed offline and allows us to quickly establish whether two robots can see each other by determining if their current cell locations are mutually visible within the environment. Although this is a conservative calculation of visibility, it is a well-motivated discretization that provides a platform for building more sophisticated computations.

We can easily determine deployment locations for robots to form visibility chains by finding shortest paths within the mutual visibility graph. If we know which cells the independent agent and base station are located within, we can find the smallest set of cells that provides a visibility chain between them. Robots are assigned target cells, and we find paths for each robot that goes from their current location to the center of their target cell. We can extend this problem from maintaining visibility to a single target and instead consider tracking multiple targets by finding a Steiner trees to connect them.

We identify two unsolved problems of interest with regards to Steiner trees: the first is finding a Steiner tree with a specified number of vertices (*fixed-size Steiner tree*), and the second is finding a Steiner tree that is minimal with respect to the movement of the vertices from an earlier placement (*minimal movement Steiner tree*). Both of these address the practical problem of finding formations for deploying a team of robots by allowing us to generate a solution that uses all of the available robots and also move the formation as the target locations move without having to recompute the Steiner tree. We solve these problems using a dynamic programming approach similar to [100, 104].

With this, we have the machinery to generate and update robot formations based on network visibility and also incorporated a discretization that allows us to handle topological complexities in the creation of movement plans. Given the formations, we control the robots to travel to their goals while simultaneously avoiding collisions and maintaining visibility using the multi-robot motion planning technique of path coordination [57, 89]. First independent paths are planned for each robot using geodesics [55] computed using the vertex-visibility graph of the environment [58, 75]. Then, these paths are refined by dividing them into segments belonging to individual visibility cells or cell boundaries. By identifying sections in the coordination diagram between two paths where robots collide or lose visibility, we create obstacles in the joint path space and then plan a path there to avoid all regions that violate constraints. This translates into a scheduled execution of the original paths that respects collisions and maintains visibility.

1.4 Thesis Contributions

This thesis makes several novel contributions to solve the posed problem of localizing and maintaining communications with an independent agent.

Though the use of over-parameterization to solve nonlinear estimation problem has been established, we have, together with the prior work of [36, 94], extended previous work on control for localization using linear estimators [35] and provided experimental results that prove the approach.

Building upon the visibility-oriented decomposition of [72], we have applied Steiner trees to the problem of generating visibility-maintaining formations for the first time, guaranteeing to use a minimum number of robots for a conservative visibility model. In addition, we have framed two new problems related to Steiner trees: *fixed-size Steiner trees* and *minimal movement Steiner trees*. We present a dynamic programming algorithm to approximate solutions to both.

The coordination of multiple robots to avoid collisions by sequencing the execution of their individually-planned paths is an established technique, but including visibility constraints is a new consideration made possible by our use of the environment decomposition.

1.5 Thesis Outline

After this introduction, we will discuss the work in three parts, and present our conclusions. The division is as follows:

Chapter 2 discusses our solution of the localization problem. The nonlinear estimation framework is presented and specialized for the problem of range-only estimation, and the estimation equations are given and demonstrated in both simulation and experiment. The control methodology is discussed and applied to the nonlinear framework, and its efficacy is demonstrated with experiment.

Chapter 3 is devoted to the polygonal decomposition of the environment and computing Steiner trees in order to find robot deployments. We discuss the fixed-size Steiner tree and minimal movement Steiner tree problems and present the algorithm to solve them approximately.

Chapter 4 presents the realization of the deployments from chapter 3 and other implementation details. We investigate the complexities of our algorithms, both theoretic and in implementation. The creation of paths using geodesic and cell-adjacencies is explained, and then both techniques are combined to yield a hybrid approach that allows us to both avoid collisions and maintain visibility. The path sequencing is presented and demonstrated. We illustrate the use of our algorithms in simulation, and perform an experiment that demonstrates the benefit that visibility provides for communication.

Chapter 5 summarizes the contributions of the thesis once more and identifies areas of future work.

Chapter 2

Nonlinear Estimation and Control for Localization

Here we consider the problem of localizing a target agent using range-only measurements from multiple mobile sources and choosing sensor movements to speed-up localization time and improve precision. With this application, it is difficult to use uninformed linearized approaches, so we present a novel nonlinear estimation framework and specialize it for the range-only case. From this we move on to the question of controlling the sensors involved so as to improve estimation quality, basing the design around minimizing the volume of the uncertainty set comprising our estimation. Both the estimation procedure and the control procedure are demonstrated experimentally and we highlight possible avenues for extending the estimation framework to problems with different structures than the range-only case.

2.1 Background

We will first formally establish the problem we wish to solve and the involved notation. This estimation framework will make extensive use of ellipsoids and so we summarize some of the relevant operations of Ellipsoid Calculus.

2.1.1 Problem Setup and Notation

We consider a mobile sensor network equipped with relative-range measurement capabilities with some capability of local sensing such as odometry or inertial measurements. This network consists of n standard nodes that have either unknown or only partially known positions and m anchor nodes that have fully known positions with respect to some global reference frame. These anchor nodes could be in motion; the only requirement is that their position is fully known.

Expressed in the global frame, the position of the ith standard node is a variable, $\boldsymbol{x}_i = [\ x_i \quad y_i\ ]^T$, and the position of the lth anchor node is $\boldsymbol{a}_l$. The total state of the network is $\mathbf{x} = [\ \boldsymbol{x}_1^T \quad \dots \quad \boldsymbol{x}_n^T\ ]^T$, and belongs to the space $\mathcal{S} = \mathbb{R}^{2n}$. A measurement between standard node i and standard node j has the form:

$$z_{ij} = h_{ij}(\mathbf{x}) + e = \|\boldsymbol{x}_i - \boldsymbol{x}_j\| + e \tag{1}$$

while a measurement between standard node i and anchor node l has the form:

$$z_i^l = h_i^l(\mathbf{x}) + e = \|\boldsymbol{x}_i - \boldsymbol{a}_l\| + e \tag{2}$$

These measurements have noise e; for our purposes we assume that this noise is bounded with constant bound ϵ. Thus $e \in [-\epsilon, \epsilon]$. These assumptions could be relaxed to include other models of bounded noise such as one with asymmetric bounds.

In a mobile sensor network, any of the nodes (standard or anchor) could be considered to be attached to mobile robots. We adopt, for simplicity, a point model:

$$\boldsymbol{x}_{i,k+1} = \boldsymbol{x}_{i,k} + \boldsymbol{u}_{i,k}, \quad i = 1,\ 2,\ \dots\ n \tag{3}$$

where $\boldsymbol{u}_i$ is the control input for the ith mobile node at time k. The state of the system evolves discretely, with the dynamic transition from step k to $k+1$ given by (3), and a set of inter-node measurements taken at each step k:

$$\mathbf{z}_k = \mathbf{h}_k(\mathbf{x}_k) + e_k$$

where $\mathbf{h}_k$ is a combination of the measurement types expressed in (1) and (2).

2.1.2 Ellipsoid Calculus

Because we rely extensively on the results in [83], we now summarize their notation and definitions. We will use x, x_0 to denote the state and z to denote observations without worrying about the previous notation in this section.

An ellipsoid can be defined by two quantities: a vector specifying the position of its center, and a symmetric positive semi-definite matrix that encodes the directions and lengths of its semi-axes as the eigenvectors and eigenvalues respectively. Given $x_0 \in \Re^n$ and $E \in \mathcal{S}^n_+$, an n-dimensional ellipsoid is defined by the set:

$$\varepsilon_n(x_0, E) = \left\{ \, x \; \middle| \; (x - x_0)^T E (x - x_0) \leq 1 \, \right\} \tag{4}$$

If E is singular, then the resulting ellipsoid is degenerate and possesses directions, corresponding to the eigenvectors of the zero eigenvalues, where x is unconstrained. The center in this case is actually only a single representative point of the affine set at the center of the ellipsoid.

Fusion

Analogous to the fusion operation in sensor fusion, we define fusion for set valued estimates to be the operation that takes two ellipsoids and finds an ellipsoid that tightly bounds their intersection. The minimum-volume bounding ellipsoid can be found using iterative algorithms such as that of [105], but, as noted there, the complexity of this procedure is an open problem. However, the suboptimal approach taken in [83], repeated here, involves the minimization of a convex function over a bounded interval and so is simple and fast. Given two n-dimensional ellipsoids, $B_1 = \varepsilon_n(x_1, E_1)$ and $B_2 = \varepsilon_n(x_2, E_2)$, a one-parameter family of fusing ellipsoids is

$\varepsilon_n^\lambda(x_0, E)$, $\lambda \in [0, 1]$, defined by:

$$\begin{aligned} X &= \lambda E_1 + (1-\lambda)E_2 \\ k &= 1 - \lambda(1-\lambda)(x_2 - x_1)^T E_2 X^{-1} E_1 (x_2 - x_1) \\ x_0 &= X^{-1}(\lambda E_1 x_1 + (1-\lambda)E_2 x_2) \\ E &= \frac{1}{k} X \end{aligned} \tag{5}$$

and the fused ellipsoid is taken as the ε_n^λ that has minimum volume. This amounts to either solving a bounded minimization problem over λ using the above, or finding the zero of the derivative of the volume as in Theorem 3 of [83]. This approximate intersection is denoted by $\tilde{\cap}$:

$$\varepsilon_n(x_0, E) \leftarrow \varepsilon_n(x_1, E_1) \,\tilde{\cap}\, \varepsilon_n(x_2, E_2)$$

Propagations

We have interest in two different ellipsoid propagations, both paralleling the state operations of our system given by Eqns. 1, 2, and 3. Theorem 1 of [83] provides the general operation that is specialized to these two special cases.

We consider first a 1-dimensional ellipsoid associated with a single measurement (also seen as an interval) $\varepsilon_1(z, 1/\epsilon^2)$. If this measurement is obtained with a linear observation model, $z = H_{1\times n}x$, its pre-image is the n-dimensional degenerate ellipsoid: $\varepsilon_n(H^\dagger z, (1/\epsilon^2)H^T H)$. $H^\dagger z$ can be any solution of the linear map, but we will use $H^\dagger$ as the pseudoinverse of H; note that the ellipsoid is rank 1 due to the form of its matrix. Thus there is an $n-1$ dimensional affine set of points that are consistent with this one dimensional observation.

Given an n-dimensional ellipsoid $\varepsilon_n(x_0, E)$, its image under the linear map $y = A_{n\times n}x + b_{n\times 1}$ is the n-dimensional ellipsoid: $\varepsilon_n(Ax_0 + b, AEA^T)$. This is the same expression encountered in propagation of Gaussian distributions in linear systems theory.

Slicing

It can be shown that the intersection of an n-dimensional ellipsoid $\varepsilon_n(x_0, E)$ with an m-dimensional $(m \leq n)$ affine set in $\Re^n$, $\mathcal{A} = \{\ y_0 + Yc \mid c \in \Re^m\ \}$, produces an ellipsoid $\varepsilon_m(\eta, F')$, where:

$$\begin{aligned} F =& Y^T E Y \\ \eta =& F^\dagger Y^T E(x_0 - y_0) \\ k =& 1 - (x_0 - y_0)^T E(x_0 - y_0) + \eta^T F \eta \\ F' =& \frac{1}{k} F \end{aligned} \tag{6}$$

If $k < 0$ then the affine set and the ellipsoid do not intersect.

Projection

An ellipsoid projection finds the "shadow" of an ellipsoid in some of its components. It can be shown that the projection of the n-dimensional ellipsoid $B = \varepsilon_n(x_0, E)$ onto its first m components is given by:

$$\mathcal{P}(B) = \varepsilon_m(x_{0,m}, E_{11} - E_{12}E_{22}^{-1}E_{12}^T) \tag{7}$$

where $x_{0,m}$ are the first m components of x_0, and $E = \begin{bmatrix} E_{11} & E_{12} \\ E_{12}^T & E_{22} \end{bmatrix}$, with E_{11} as an m-dimensional block.

Furthermore, if Y is a basis for any subspace of dimension m and Z is a basis for its null space of dimension $n - m$, then the ellipsoid $B = \varepsilon_n(x_0, E)$ projected onto this subspace is given by $B_Y = \varepsilon_m(Y^T x_0, E_Y)$, with E_Y given by:

$$E_Y = Y^T E\, Y - Y^T E\, Z\, (Z^T E\, Z)^{-1} Z^T E\, Y \tag{8}$$

The projection operation for ellipsoids is analogous to marginalization of multivariate Gaussians (see [101]).

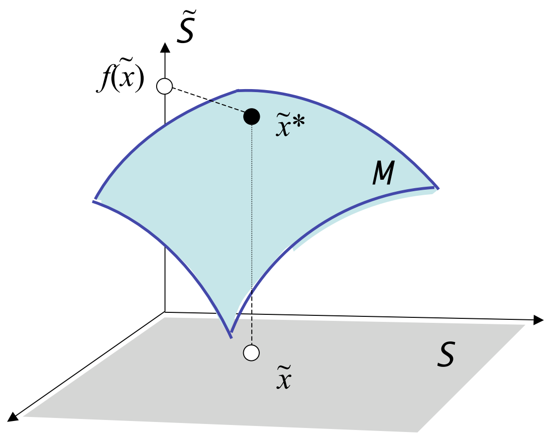


Figure 5: The state space $\mathcal{S}$ is extended by adding functionally dependent coordinates of $\mathcal{S}$ to create the extended state space $\mathcal{S}^\star = \mathcal{S} \oplus \tilde{\mathcal{S}}$.

2.2 Nonlinear Estimation Framework

We will now introduce the estimation framework and discuss the procedure for forming and updating a state estimate. Some techniques for making use of this state estimate will be given, and we will conclude by discussing ways that this framework could be extended to new problems beyond the range-only measurement problem we will develop in the next section.

2.2.1 Concept

Hanebeck [38] introduced a novel framework involving a nonlinear embedding that maps the system states into an extended state space in such a way that the measurement equations become linear. The basic idea is shown in Figure 5.

The *extended space* $\mathcal{S}^\star$ is formed by augmenting the *base space* $\mathcal{S}$ with additional dimensions. Define a smooth map $f : \mathcal{S} \to \tilde{\mathcal{S}}$ and let $\mathcal{S}^\star = \mathcal{S} \oplus \tilde{\mathcal{S}}$. Recall that $\mathbf{x}$ denotes elements of $\mathcal{S}$ and let $\mathbf{x}^\star$ denote elements of $\mathcal{S}^\star$. The map f defines a

$2n$-dimensional smooth sub-manifold in $\mathcal{S}^\star$ with coordinates $\mathbf{x}$:

$$\mathbf{x}^\star = g(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ f(\mathbf{x}) \end{bmatrix} \tag{9}$$

The Jacobian of g,

$$\frac{\partial g}{\partial \mathbf{x}} = \begin{bmatrix} I \\ \frac{\partial f}{\partial \mathbf{x}} \end{bmatrix}$$

is always full rank and so the resulting manifold M is diffeomorphic to $\mathcal{S}$. Our goal is to choose f and transform the system equations in such a way as to make them linear in the p-dimensional extended space $\mathcal{S}^\star$ ($p > 2n$). Note that we will ultimately be interested only in those points that lie on the manifold, *i.e.*, $\mathbf{x}^\star \in M$.

2.2.2 Incorporating New Measurements

An appropriate definition of the map f allows us to write each measurement equation at time step k in the form:

$$z_k^\star - w_k^\star = H_k^\star \mathbf{x}_k^\star$$

where $\mathbf{x}_k^\star \in \mathcal{S}^\star$ and $w_k^\star$ is the measurement noise, assumed to be contained in a known interval. By viewing the interval quantity on the left-hand side as a 1-dimensional ellipsoid, we apply the results on ellipsoid propagation under affine transformations, covered earlier in Section 2.1.2, to define $\mathcal{Z}_k = \varepsilon_P((H_k^\star)^\dagger z_k^\star, (1/(w_k^\star)^2)(H_k^\star)^T H_k^\star)$ as the feasibility ellipsoid in $\mathcal{S}^\star$ consistent with this measurement.

Each $\mathcal{Z}_k$ can be seen as a pair of bounding hyperplanes constraining the possible embedded states. However, since only the values of $\mathcal{S}^\star$ lying in M have meaning, the actual set described by $\mathcal{Z}_k$ can be interesting, as portrayed in Figure 6. As more measurements are included, more bounds need to be incorporated. However, rather than tracking an increasing number of hyperplanes, each new $\mathcal{Z}_k$ is incorporated into an aggregate state estimate ellipsoid $\mathcal{E}_k^\star$ using the ellipsoid fusion procedure discussed earlier in Section 2.1.2:

$$\mathcal{E}_k^\star \leftarrow \mathcal{E}_k^\star \,\tilde{\cap}\, \mathcal{Z}_k$$

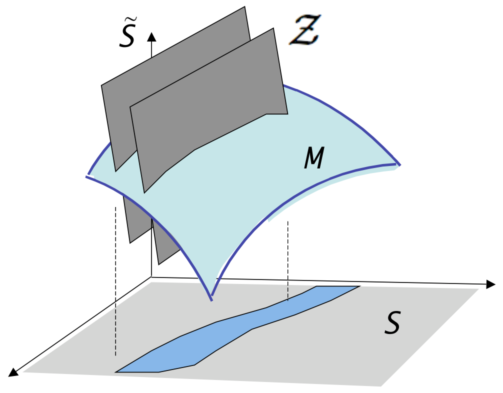


Figure 6: A measurement $\mathcal{Z}$, transformed into a set bounded by two hyperplanes, defines an interesting set in the base space $\mathcal{S}$ when intersected with M.

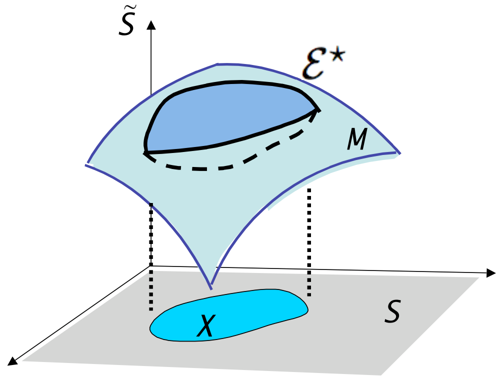


Figure 7: The feasibility set $\mathcal{X} \subset \mathcal{S}$ is found by intersecting an ellipsoid $\mathcal{E}^{\star} \subset \mathcal{S}^{\star}$ with the manifold M.

When the filtering is started, $\mathcal{E}_k^{\star}$ can be initialized with $\varepsilon_p(0,0)$, a fully degenerate ellipse, to reflect the fact that nothing is known about the state. This is analogous to the initialization of the Information form of the Kalman filter with zeroed values (see [101]). Unlike the case of an extended Kalman filter, tricky estimate initialization procedures are not necessary.

It was seen earlier in Sec. 2.1.2 that, when fusing new measurement ellipsoids into the estimate, there is a parameter λ that is solved for to choose how much of the measurement is incorporated into the updated estimate. $\lambda = 1$ indicates that the previous ellipsoid is left untouched and the measurement has no effect; conversely, $\lambda = 0$ indicates that the previous ellipsoid is lost and only the measurement ellipsoid is used. Values in-between give an indication of the innovation of the measurement. This fact will be exploited later on to understand the workings of the filter.

2.2.3 Making Use of Estimates

After performing filtering steps, the feasibility ellipsoid $\mathcal{E}_k^\star$ contains all $\mathbf{x}_k^\star$ consistent with measurements up to step k, but not all of these elements have physical meaning. Only the $\mathbf{x}_k^\star \in M$ actually represent the images of states in $\mathcal{S}$. This feasible set, $\mathcal{X}_k \subset \mathcal{S}$ is found by:

$$\mathcal{X}_k = \{\ \mathbf{x} \in \mathcal{S}\ \mid g(\mathbf{x}) \in \mathcal{E}_k^\star\ \}$$

The basic idea is shown in Figure 7.

This inversion can be carried out exactly using the implicit form (4) of $\mathcal{E}_k^\star = \varepsilon_p(x_0, E)$ together with g:

$$(g(\mathbf{x}_k) - x_0)^T E(g(\mathbf{x}_k) - x_0) \leq 1 \tag{10}$$

Any $\mathbf{x}_k$ satisfying this implicit nonlinear inequality belong to the true feasibility set.

The true size of the estimate set is only found by measuring the size after performing this inversion. However, this process can be quite difficult to perform in general, and so metrics on the ambient ellipsoid prove useful. The determinant of the ellipsoid moment matrix (which corresponds to the inverse of the volume) and the trace (which corresponds to the sum of the inverse squares of the semi-axis lengths) can both provide some notion of the size and shape of the ellipsoid. These metrics were employed to evaluate the filter performance during experimentation.

We suggest two approximate set inversion techniques: base projection and tangent slicing.

Base projection works by noticing that the form of the state augmentation in Eq. 9 has the nonlinear terms stacked on top of the normal state space. By projecting the higher-dimensional ellipsoid onto this linear portion of the state space using the results of Sec. 2.1.2, a conservative bound on the true set is found. Later, in Fig. 10 this projection is referred to as the "Linear Estimate Sub-space" because it corresponds to the linear terms of the state augmentation. It should be noted that

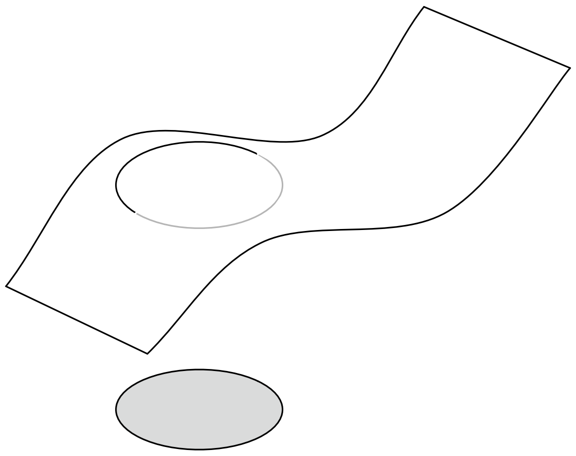


Figure 8: A base projection approximation of the estimate ellipsoid. The ellipsoid is projected onto the linear part of the coordinates, yielding a “shadow” which surrounds the true intersection.

this approximation completely ignores the contribution of the nonlinear terms in the embedding. Figure 8 gives an idea of the process.

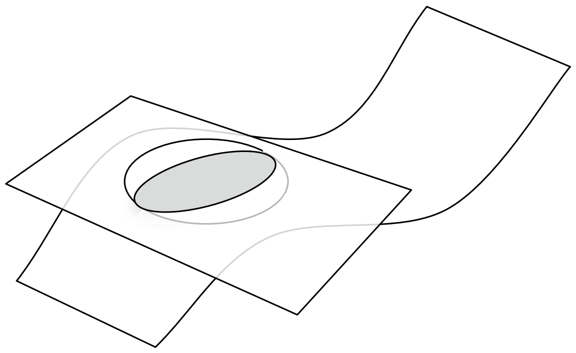


Figure 9: A tangent slice approximation of the estimate ellipsoid. A hyperplane tangent to the manifold near the ellipsoid is used and a lower-dimensional representation of the estimate is found in the coordinates of the base space.

Tangent slices, on the other hand, do take the contributions of the nonlinear transformation into account when approximating the true set inversion. By recognizing M as a manifold embedded into $\mathcal{S}^\star$ by g, the tangent space of M at a point

$\mathbf{x}^\star \in M$ can be found using the Jacobian of g. $\mathbf{x}^\star$ is taken to be the point M closest to the center of the estimate ellipsoid. Now $\frac{\partial g}{\partial \mathbf{x}}$ and $\mathbf{x}^\star$ define an affine set in $\mathcal{S}^\star$ with the same dimension as $\mathcal{S}$. By restricting $\mathcal{E}_k^\star$ to this set using the slicing operation of Sec. 2.1.2, an approximate representation is found in $\mathcal{S}$. This representation does not have the bounding properties that the base projection does, however. Figure 9 gives an idea of the process.

2.2.4 Finding New Embeddings

One good question to consider is: How do we choose f? The simplest choice is to do it by inspection, as advocated by the work of [59] on Non-Minimal-State Kalman Filters. As we have done here, by writing out the full nonlinear measurement equation a suitable split may present itself.

Another option, suggested by [38], is to use a Bernstein polynomial basis. The Stone-Weierstrass approximation theorem tells us that it is possible to approximate any continuous function over a real interval using a polynomial over the reals, and Bernstein polynomials are often used as a constructive proof of the theorem thanks to their properties. The $n + 1$ Bernstein basis polynomials of degree n are defined by:

$$b_{\nu,n}(x) = \begin{pmatrix} n \\ \nu \end{pmatrix} x^\nu (1-x)^{n-\nu}$$

where $\begin{pmatrix} n \\ \nu \end{pmatrix}$ is a binomial coefficient. Though this is a 1-dimensional basis, there are similar expressions for higher dimensions found by multiplying the bases together. There are many nice properties of the polynomial, including the ability to write a derivative as a combination of two polynomials of lower degree and the ability to write a polynomial as the combination of two polynomials of higher degree. Finding a state augmentation would involve taking the nonlinear system equations and finding their linear representations in this basis, either exactly (if they are polynomial) or approximately (if they are not).

A promising alternative is suggested by Daum in several places including [18]. He advocates using partial differential equations known as Fokker-Planck equations to express the evolution of the state estimate probability distribution over time. As long as the disturbance belong to an exponential family, there is a set of partial differential equations that, when solved, provide a set of terms encompassing a sufficient statistic for the distributions possible under the system's dynamics. From looking at the example of the radar tracking in [18], it is clear that there are direct connections between that work and the work presented here. This could lead to an automatic mechanism for producing augmentations.

2.3 Application to Range-only Measurements

As an illustration, begin with the range measurement equation between two standard nodes:

$$z_{ij} - e = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$$

and square both sides. The left hand side represents the interval $[z_{ij} - \epsilon, z_{ij} + \epsilon]$ which, when squared using interval arithmetic, becomes $[z_{ij}^2 - 2z_{ij}\epsilon + \epsilon^2, z_{ij}^2 + 2z_{ij}\epsilon + \epsilon^2]$. By letting $z_{ij}^\star = z_{ij}^2 + \epsilon^2$ and $w \in [-2z_{ij}\epsilon, 2z_{ij}\epsilon]$, the transformed measurement equation is:

$$z_{ij}^\star = \boldsymbol{x}_i \cdot \boldsymbol{x}_i + \boldsymbol{x}_j \cdot \boldsymbol{x}_j - 2\boldsymbol{x}_i \cdot \boldsymbol{x}_j + w$$

If a different bounded-noise model is used for e, such a transformation is still possible as long as care is taken to ensure that the modified estimate and bounds are conservative.

Notice that this equation is nonlinear in the system variables $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ but is

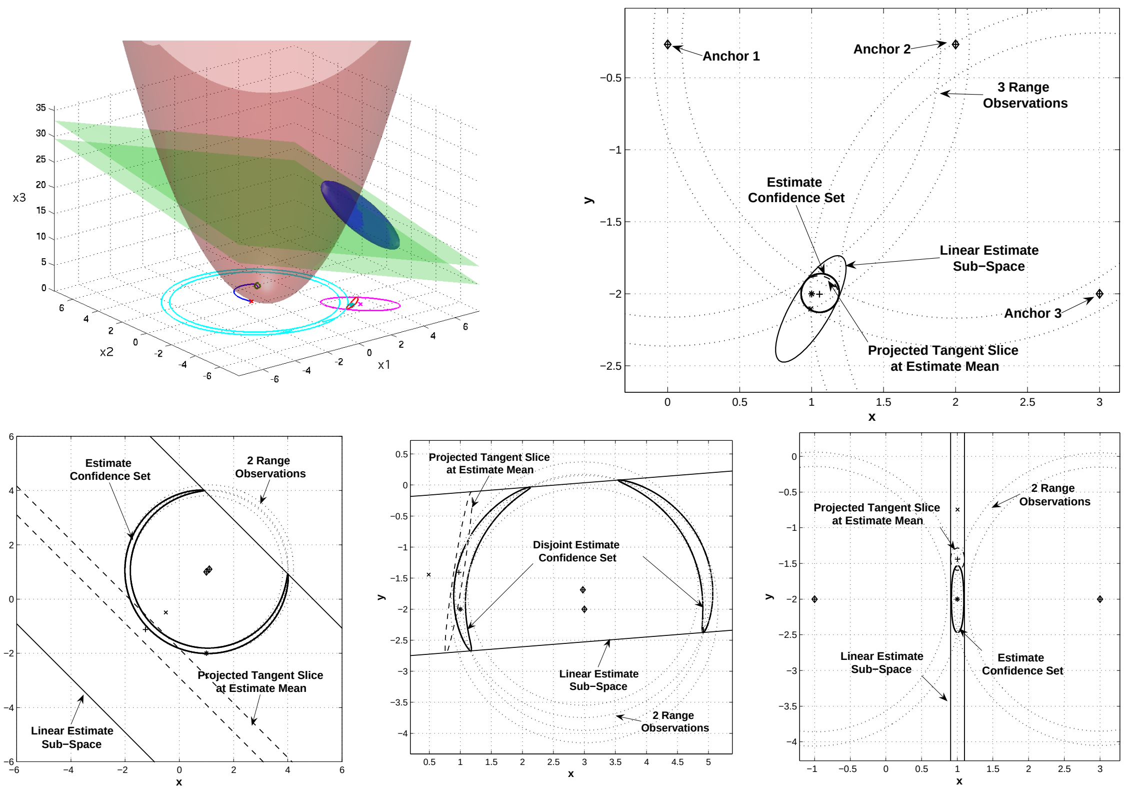


Figure 10: Visualization of the proposed embedding for a single robot range-only localization problem along with examples of possible estimate uncertainties represented by this formulation. In this case a closed form expression is available for the position estimate set. As more measurements are taken into account, the nature of the solution sets changes. Considering all possible (possibly degenerate) ellipsoids and their intersections with the state manifold, a paraboloid in this case, the resulting solution sets can take the form of an annulus, a simple set, or two disjoint simple sets.

linear in the variables $\boldsymbol{x}_i \cdot \boldsymbol{x}_i$, $\boldsymbol{x}_j \cdot \boldsymbol{x}_j$, and $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$:

$$z_{ij}^{\star} = \begin{bmatrix} 1 & 1 & -2 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_i \cdot \boldsymbol{x}_i \\ \boldsymbol{x}_j \cdot \boldsymbol{x}_j \\ \boldsymbol{x}_i \cdot \boldsymbol{x}_j \end{bmatrix} + w$$

Applying this process to the range measurement equation between a standard node and an anchor node leads to:

$$z_i^{l,\star} - \boldsymbol{a}_l \cdot \boldsymbol{a}_l = \begin{bmatrix} -2\boldsymbol{a}_l^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{x}_i \cdot \boldsymbol{x}_i \end{bmatrix} + w$$

Accordingly, we define f so that:

$$f(\mathbf{x}) = [\ldots,\ \boldsymbol{x}_i \cdot \boldsymbol{x}_i,\ \boldsymbol{x}_j \cdot \boldsymbol{x}_j,\ \boldsymbol{x}_i \cdot \boldsymbol{x}_j,\ \ldots]^T .$$

Thus $\mathcal{S}^{\star}$ is constructed by adding at most $n + {}_nC_2$ dimensions to $\mathcal{S}$, corresponding to all dot product combinations of the positions of the nodes that appear in the measurement equations. The measurement equations, after suitable modifications to the additive noise, are now linear in $\mathcal{S}^{\star}$ while having bounded noise. The resulting manifold and some examples of its applicability to the problem are seen in Figure 10.

We are not limited to range-only sensors. Indeed, the measurement equations for bearing-only sensors can also be made linear with a similar embedding (see [11]).

2.4 Control for Localization

Though estimation is a challenging problem in its own right, the question of how to control systems to enable and improve estimation is an important one with many practical implications. In order to address this question, we make use of previous work on maximizing information gains in decentralized systems using a more traditional linear filter [35], summarized here, and then draw a correlation between the structures used in that approach and those in the current approach to develop a methodology.

2.4.1 Techniques Used with Gaussian filters

Though it involves more traditional linearized filtering techniques and seems somewhat tangential, we shall summarize the results of [35] in order to motivate our control strategy. This section is stand-alone in terms of notation used. $\mathbf{x}(k)$ is the vector state of the system at discrete time step k and $\mathbf{z}(k)$ is the vector of observations made at this time.

Start with the discrete time process and observation models given by:

$$\begin{aligned}\mathbf{x}(k) &= \mathbf{F}(k)\mathbf{x}(k-1) + \mathbf{G}(k)\mathbf{w}(k)\\ \mathbf{z}(k) &= \mathbf{h}(k, \mathbf{x}(k)) + \mathbf{v}(k)\end{aligned}$$

where the process noise $\mathbf{w}(k)$ and observation noise $\mathbf{v}(k)$ are associated with Gaussian models with zero mean and covariances of $\mathbf{P}$ and $\mathbf{R}$ respectively. The estimate of the underlying state $\mathbf{x}(k)$ can be performed using an Information Filter (sometimes called the "Information Form of the Kalman Filter") obtained by replacing the typical Kalman Filter state estimate $\hat{\mathbf{x}}$ and covariance $\mathbf{P}$ with the information state $\hat{\mathbf{y}}$ and Fisher information $\mathbf{Y}$. The notation $(i|j)$ indicates a value at time i conditioned on observations up until time j. The information state and Fisher information are given by the relations:

$$\hat{\mathbf{y}}(i|j) = \mathbf{P}^{-1}(i|j)\hat{\mathbf{x}}(i|j) \qquad (11)$$

$$\mathbf{Y}(i|j) = \mathbf{P}^{-1}(i|j) \qquad (12)$$

New observations are incorporated as information vectors and information matrices given by:

$$\begin{aligned}\mathbf{i}(k) &= \mathbf{H}^T(k)\mathbf{R}^{-1}(k)(\mathbf{z}(k) - \mathbf{h}(k, \hat{\mathbf{x}}(k|k-1))) + \mathbf{H}(k)\hat{\mathbf{x}}(k|k-1)\\ \mathbf{I}(k) &= \mathbf{H}^T(k)\mathbf{R}^{-1}(k)\mathbf{H}(k)\end{aligned}$$

where $\mathbf{H}^T(k)$ is the Jacobian $\nabla_{\mathbf{x}}\mathbf{h}(k, \hat{\mathbf{x}}(k|k-1))$. The filter splits into a prediction step governed by the equations:

$$\mathbf{Y}(k|k-1) = [\mathbf{F}(k)\mathbf{Y}^{-1}(k-1|k-1)\mathbf{F}^T(k) + \mathbf{Q}(k)]^{-1} \tag{13}$$

$$\hat{\mathbf{y}}(k|k-1) = \mathbf{Y}(k|k-1)\mathbf{F}(k)\mathbf{Y}^{-1}(k-1|k-1)\hat{\mathbf{y}}(k-1|k-1) \tag{14}$$

and a correction step given by the equations:

$$\mathbf{Y}(k|k) = \mathbf{Y}(k|k-1) + \Sigma_{i=1}^{N}\mathbf{I}_i(k) \tag{15}$$

$$\hat{\mathbf{y}}(k|k) = \hat{\mathbf{y}}(k|k-1) + \Sigma_{i=1}^{N}\mathbf{i}_i(k) \tag{16}$$

where $\mathbf{I}_i(k)$ and $\mathbf{i}_i(k)$ are the information matrix and vector associated with the ith sensor.

As those familiar with the technology will see, the information filter is simply an inverted form of the Kalman filter, drawn from the conversion of Eq. 11. By storing the state and estimate covariance in this form, the prediction step of Eq. 13 is made more complicated as a tradeoff for making the correction step of Eq. 15 very straight-forward. In [35] and earlier work, the additive form of Eq. 15 is exploited to realize Decentralized Data Fusion (DDF) systems that can incorporate observations from sensors and platforms that may not even be aware of each other but can communicate through a shared medium.

Derivation of Control Law Using an Information filter

However, the real contribution of this Information Filter work comes when addressing the problem of developing control strategies to maximize information gain. The basic idea is to start with a set of available inputs and a current state estimate, apply the inputs to the current estimate using the process model, hypothesize about what the future measurements would be for each input and incorporate them using the observation model, and finally quantify all of the potential future estimates to select the input that yields the estimate with the smallest uncertainty (has the most information). Though in general this would be a complex task, for linear systems the burden is substantially less.

The information value of the estimate is contained in the Fisher information matrix $\mathbf{Y}$ and is commonly quantified by taking the determinant $|\mathbf{Y}|$. The dual relationship to the standard Kalman filter is obvious here; for the Kalman filter, the uncertainty of the estimate is found by taking the determinant of the covariance, representing the volume of the uncertainty ellipsoid, and since $\mathbf{Y}$ is the inverse of $\mathbf{P}$ the information increases as the uncertainty decreases. We are interested in increasing $|\mathbf{Y}|$ and so we seek to maximize the *instantaneous mutual information rate*:

$$\mathcal{I}(t) = \frac{1}{2}\frac{d}{dt}\log\left(|\mathbf{Y}(t)|\right) = \frac{1}{2}trace\left(\mathbf{Y}^{-1}(t)\dot{\mathbf{Y}}(t)\right)$$

where we use the monotonic logarithm function in order to make use of a matrix identity and obtain the clean form on the right-hand side. The maximization of this rate gives rise to a gradient control law with zero look-ahead rather than planning actions over time.

With a stationary process model, the only influence on $\mathbf{Y}$ is coming from the inclusion of measurements as in Eq. 15:

$$\dot{\mathbf{Y}}(t) = \sum_{i=1}^{N}\mathbf{I}_i(t)$$

This gives rise to the very nice property that, given a common estimate $\mathbf{Y}$, the influence of each sensor on the total information gain is separable:

$$\mathcal{I}(t) = \frac{1}{2}trace\left(\mathbf{Y}^{-1}(t)\Sigma_{i=1}^{N}\mathbf{I}_i(t)\right) = \frac{1}{2}\sum_{i=1}^{N}trace\left(\mathbf{Y}^{-1}(t)\mathbf{I}_i(t)\right)$$

and so the optimization of information gain can occur in a distributed fashion since each element only needs to worry about its own contribution against the current estimate.

In [35], the setup was for a stationary process model (targets were stationary) while the observers were mobile with collective state $\mathbf{x}_R$. The problem is then to choose control directions for the observers to maximize the mutual information rate by following the gradient:

$$\nabla_{\mathbf{x}_R}\mathcal{I}(t) = \frac{1}{2}trace\left(\mathbf{Y}^{-1}(t)\nabla_{\mathbf{x}_R}\mathbf{I}(t)\right)$$

where the expression obviously splits according to separation above. This gradient algorithm is the desired control strategy.

2.4.2 Application to Set-based Filter

In our system, we have a stationary target and mobile robots equipped with range-only sensors. The position of the mobile robots is known, so they can be treated as anchors, but the target's position needs to be estimated. This setup exactly matches the setup of [35] with the chief difference being that we are performing estimation using sets over augmented states rather than a linearized filter. Using a point-motion model for the anchor (robot) positions, we can hypothesize future measurement ellipsoids based on potential inputs $u_i = (u_{x,i}, u_{y,i})$ and current estimates.

There is a strong structural analogy between ellipsoid-set filters and our nonlinear set filter. In particular, the moment matrix $E(t)$ and the information matrix $\mathbf{Y}(t)$ behave very similarly. In fact, the former could be treated as the 1-standard-deviation level set of the Gaussian described by the latter, and we carried through with this technical blurring and obtained encouraging results despite the theoretical challenges presented.

So by treating estimate and measurement ellipsoids as level sets of a Gaussian, an approximate mutual information rate for the motion of robot i can be found:

$$\tilde{\mathcal{I}}_i(u_i, t) = \frac{1}{2}trace\left(E^{-1}(t)H_i(u_i)^T R_i(u_i)H_i(u_i)\right) \tag{17}$$

where $H_i(u) = [-2(x_i + u_{x,i}),\ -2(y_i + u_{y,i}),\ 1]$ and $R = (2z_i(u_i)\epsilon)^{-1}$, with $z_i(u_i) = \sqrt{r_i(u_i)^2 + \epsilon^2}$ as the transformed version of the predicted measurement by robot i after its movement u_i and ϵ as the noise bound on the range measurement. $r_i(u_i)$ is the hypothesized range measurement that will be taken after performing input u_i; in this work it was assumed that the range measurement would be approximately the same as the last measurement and so it is the direction of the future measurements that is important in driving the gradient. The H vector is derived by taking the anchor version of the measurement equations in Sec. 2.3 and applying a point-motion model to the anchor position, as given in Eq. 3.

The control inputs to maximize $\tilde{\mathcal{I}}_i(u_i, t)$ are then found as $u_i(t) = \nabla_{u_i}\tilde{\mathcal{I}}_i(u_i, t)$. The length of $u_i(t)$ is ignored and only the direction is used in order to dictate the motion of the robot.

In essence, this control law is selecting input directions that are informative in directions that the current estimate $E(t)$ knows little about. In Eq. 17, the $H_i(u_i)^T R_i(u_i) H_i(u_i)$ term is describing the degenerate ellipsoid that would be obtained by a measurement taken after input u_i. It is constraining the state in the direction of the vector $H_i(u_i)$. The current estimate $E(t)$ could be decomposed into $E(t) = V(t)^T \Lambda(t) V(t)$ where $V(t)$ is an orthonormal matrix and $\Lambda(t)$ is a diagonal matrix. Viewing $E(t)$ as an inverse covariance (or information matrix), then $V(t)$ and $\Lambda(t)$ give the directions and inverse-squared lengths of the semi-axes of the 1-standard-deviation level set of the Gaussian ellipsoid. The directions of $V(t)$ dictate orthogonal directions in the estimate space, and the corresponding lengths dictate how much is "known" in these directions. So using the properties of the trace, we can see that we are calculating

$$\tilde{\mathcal{I}}_i(u_i, t) = \frac{1}{2} trace\left(\Lambda_i^{-1}(t) V(t) H_i(u_i)^T R_i(u_i) H_i(u_i) V(t)^T\right)$$

The informative directions of the new measurement, $H_i(u_i)$, are projected into the eigenspace of the current estimate, $V(t)$, and then weighted according to how much is known in these directions using the lengths of the semi-axes. Measurements providing more information in the longer directions of $E(t)$ will produce larger values of $\tilde{\mathcal{I}}_i$. Degenerate directions have infinite semi-axis lengths but these are treated numerically as some very large number to avoid singularities.

2.5 Experimental Results and Conclusions

In our experiments we make use of the Parrot hardware developed at CMU. The system establishes a peer-to-peer RF network and coordinates sonar pulses; ranges between individual nodes are calculated by time-of-flight and the negotiation procedure ensures that the source is identified. This sensor system is mounted on a pair of indoor ground robots and a station as shown in Fig. 11.

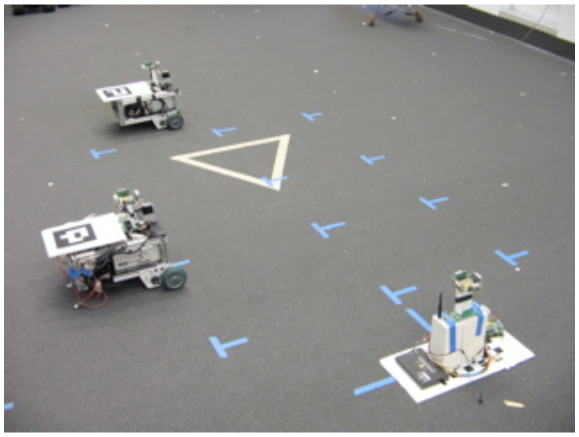


Figure 11: The experimental setup.

The task was for mobile robots to actively localize an unknown, static feature using range-only sensors. The location of the robots is assumed to be known and is provided by an overhead camera system using the ARToolkitPlus software, originally developed by researchers at the University of Washington and improved by researchers at the Graz University of Technology. Complete coverage of the experimental area is provided by four calibrated Dragonfly cameras manufactured by Point Grey, equipped with 6mm lenses, and mounted at a height of 19 feet above the floor. The largest errors in the system occur at the boundaries of each camera's field of view, and there is no graceful transition from one camera to the next. This limitation led to some position jumps during the experiment and highlighted some interesting features of the estimation technique, as seen and explained in Figs. 12 and 13.

In the experiment plots, the bold T indicates the target location and the cyan and green annuli show the current range measurements with bounded noise. Magenta ellipsoids correspond with the "shadow" of the ellipsoid projected onto the x, y plane (also seen in Fig. 10). Dotted blue ellipsoids are found using the tangent slice approximation and are essentially a linearized approximation of the ellipsoid near the surface of the paraboloid. Finally, the red shapes correspond with the true feasibility set, found by analytically solving for the intersection of the ellipsoid with the paraboloid.

We perfomed three sets of experiments:

1. A single robot was used to localize a static feature. The results are seen in Figs 12 and 13. An operator-controlled run was performed to attempt to quantify the effect of using control feedback, and these comparison results are seen in Fig. 14.

2. A second robot was added in but kept stationary in order to see how controls for the first robot were affected in an additional (stationary) information source, and the results are seen in Fig. 15 and 16.

3. Control inputs were applied to both robots in order to demonstrate a full cooperative mapping task, as seen in Figs. 17 and 18.

These experiments lead to the conclusion that the active control strategy is robust, has a definite beneficial effect, and leads to the strategy considered to be optimal.

Robustness

In Fig. 12, a single robot was commanded using the controller in Sec. 2.4. At some point during the experiment, the overhead localization system provided a flawed value, and an inconsistent measurement was introduced into the filter. This evidence for this is seen in the plot of $|E|$ in Fig. 13, where the value dips below zero in frame 4, indicating a measurement ellipsoid that does not intersect the previous estimate.

However, the estimate naturally converges to a set of feasible states as seen in the later frames.

What happened was that small λ values were used, effectively throwing away the current (bad) estimate after the faulty measurement, and the new measurements were trusted more completely. The magnitude of the determinant never fully recovered since at no point was the bad estimate completely thrown away, but it was enough recovery to prevent the target from being lost. This modulation between estimate and measurement happened automatically, but a more explicit system could help the recovery proceed faster.

Beneficial Effect

We measure the positive effect of the controller in two ways: the active control shrinks the estimate ellipsoid faster than otherwise, and adding robots alters the individual trajectories and improves the results. This suggests that multiple robots are providing benefit beyond that of simply running multiple copies of the same filter, one for each robot, and are in fact coordinating sensibly.

Fig. 14 shows a comparison of $|E|$ for a robot driven using the controller described in Sec. 2.4 and one driven at the same speed in a straight line by a human operator past the target. It can be clearly seen from the experimental trace that determinant is orders of magnitude higher using the closed loop controller than for the human driven case. Since the determinant is inversely proportional to the volume, the estimate ellipsoid in the extended space shrinks faster. Thus the true feasibility set, found by intersecting the estimate ellipsoid with a paraboloid as in Fig. 10, shrinks faster as well.

Figs. 15 and 16 shows an experiment run when a second robot was added but kept stationary. The addition of a new source of information alters the control trajectories from the single robot run, and the rate of determinant increase is comparable to the single robot case.

In Figs. 17 and 18, the experiment is carried out with two mobile robots, and the trajectories change yet again.

These trajectory changes arise naturally through the effect that each robot's measurement has on the estimate, similarly to [35]. It is important to note that there is not explicit coordination or planning taking place. The coordination is implicit and occurring through the shared information state of the target. At each instant, an agent can see the knowledge of the target as well as its uncertainty/information representation which encodes all previous measurements of the target made by all agents, in some meaningful way. For a one-step decision, it is more beneficial to take a movement and new measurement in a way that *other* agents have not. The fact that the shared information state gives us access to their previous decisions lets us completely ignore the details of who is providing information and how they are doing it.

This sort of implicit coordination does not reduce the complexity of building and maintaining a shared information state, but it *does* reduce the complexity of making decisions amongst several agents.

Hints of Optimal Strategy

In Figs. 15 and 17, the robot trajectories end up guiding the robot so that the measurement annuli of the individual robots lie at right angles to each other over top of the target location. This geometric configuration leads to superior quality measurements (consistent with intuition) but was not explicitly specified and instead arose from the information gradient considerations of Sec. 2.4. Similar results were observed in [35] with ground vehicles that had very accurate bearing measurements but very inaccurate range measurements.

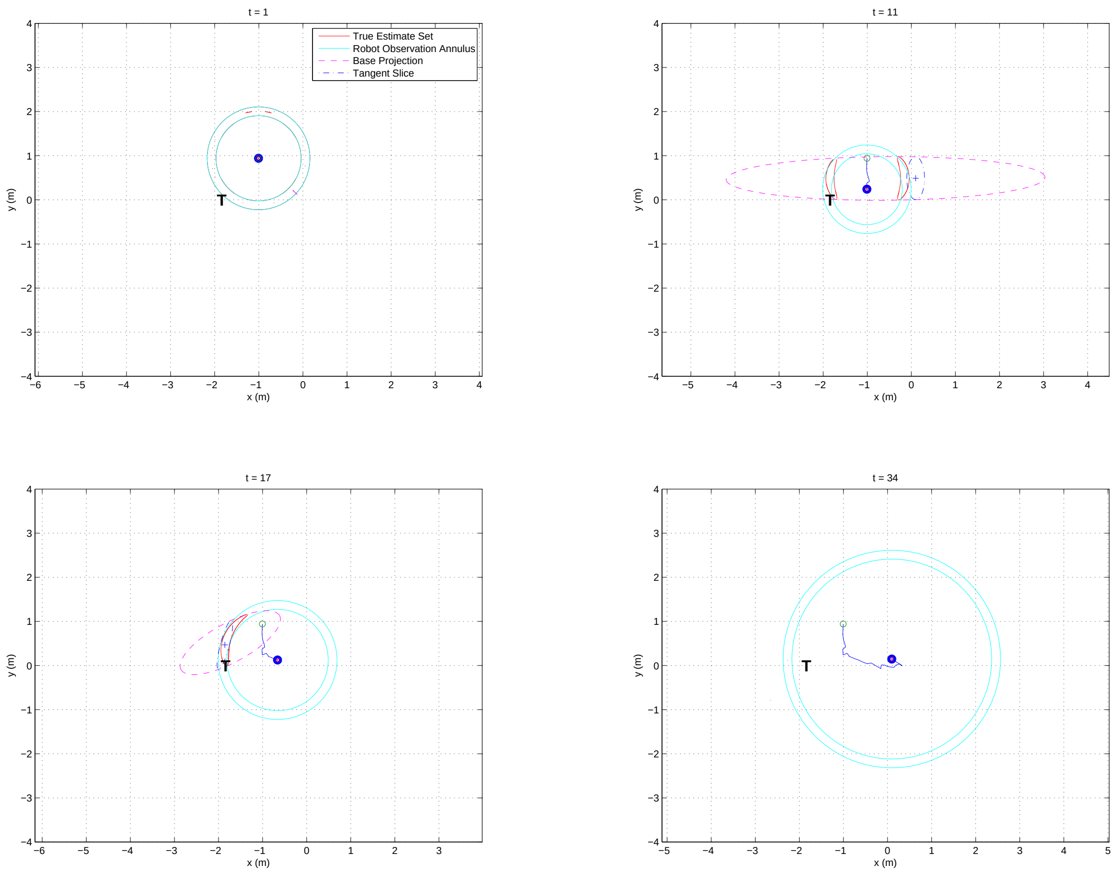
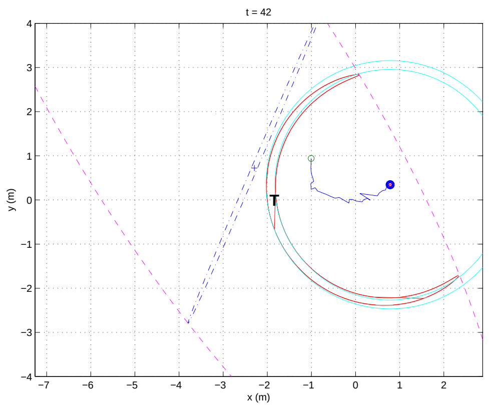


Figure 12: Six snapshots of a single robot mapping an unknown feature using the proposed controller. The plot elements are described in the text. With only one bounded range measurement, the feature estimate is an annulus. Further measurements reduce the annulus to two disjoint possible sets and finally a single set. In the fourth frame, an error in the ground truth scheme for the robot has made its measurement inconsistent and the filter estimate is destroyed. As further measurements are incorporated, the filter recovers and frames five and six show that the estimate qualitatively returns to normalcy.

2.6 Summary

In this chapter, we solved the problem of localizing an independent agent using range-only measurements from mobile sensors and then choosing sensor movements to speed-up localization time and improve precision. We did this with a novel estimation



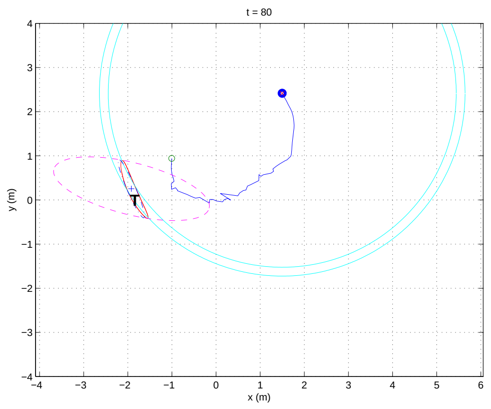


Figure 12: *(continued)* Six snapshots of a single robot mapping an unknown feature using the proposed controller. The plot elements are described in the text. With only one bounded range measurement, the feature estimate is an annulus. Further measurements reduce the annulus to two disjoint possible sets and finally a single set. In the fourth frame, an error in the ground truth scheme for the robot has made its measurement inconsistent and the filter estimate is destroyed. As further measurements are incorporated, the filter recovers and frames five and six show that the estimate qualitatively returns to normalcy.

framework based on using an extended state space to transform a nonlinear problem into a linear one and then using a standard linear filtering technique. In our case, we used a set-based technique with ellipsoid set representations.

From this estimation framework, we drew an analogy to existing techniques in control for information acquisition in probabilistic linear filters and derived equations for the new framework.

Both the estimation and control were implemented on a hardware platform and the technique was validated experimentally, with some interpretation and explanation of weaknesses of the set-based assumption and how the filter robustly recovered.

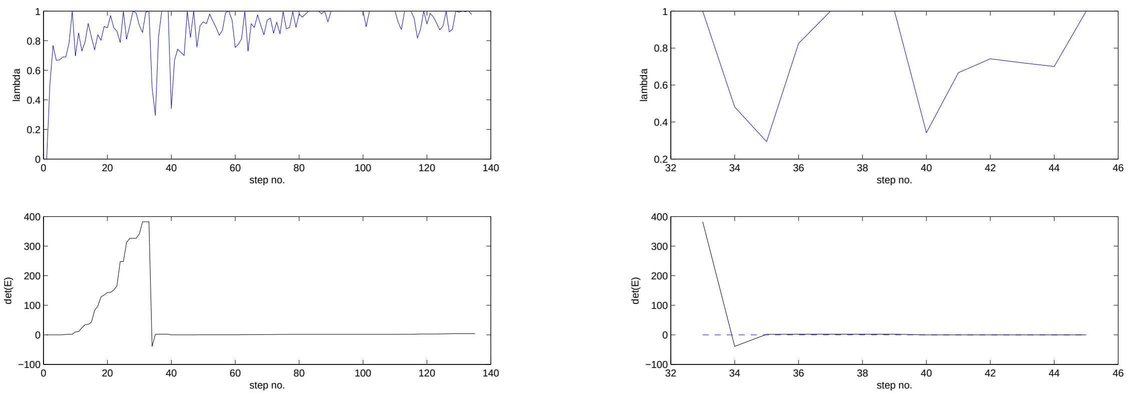


Figure 13: The fusion parameter and determinant of the moment matrix for the controlled single robot case. At time step 34, corresponding to the fourth frame in Fig. 12, a bad measurement results in the fused ellipsoid having a negative determinant, meaning that the prior estimate and measurement did not intersect and the fused ellipsoid is complex. As further measurements are incorporated, small λ values result in the complex estimate ellipsoid being thrown away and the measurements being used to bootstrap a new estimate. The second image is a close-up view of the recovery period.

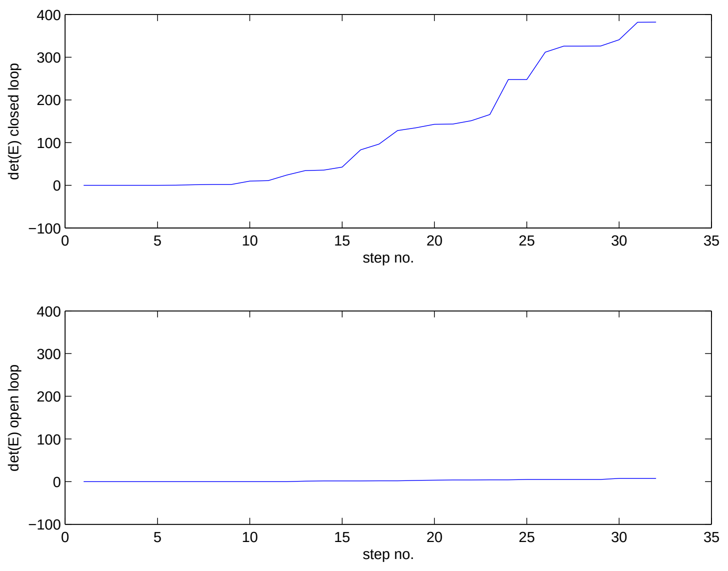


Figure 14: A comparison of the determinant of the ellipsoid moment matrices for two single robot runs: one driven using the closed-loop controller and one driven by an operator in a straight line past the target. The closed-loop data is from the run shown in Fig. 12 before the corrupting measurement is received. The determinant for the open-loop case is growing but is orders of magnitude smaller than the closed-loop case.

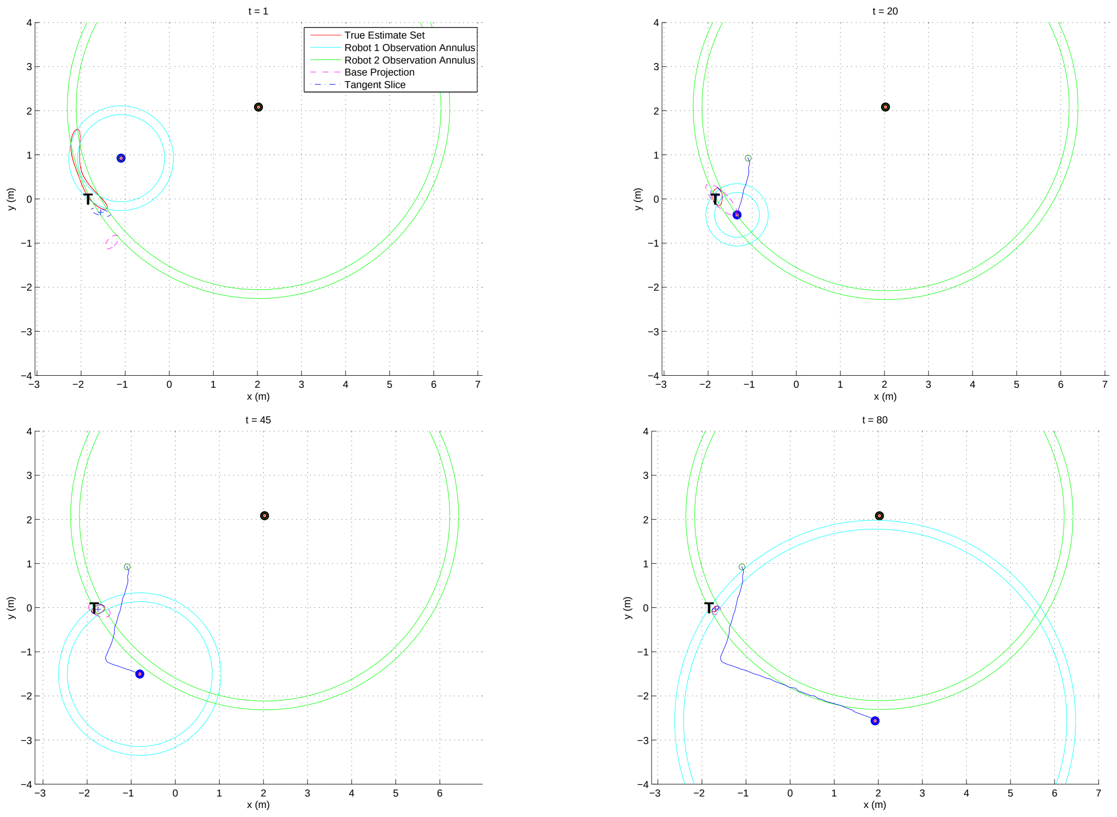


Figure 15: Four snapshots of a single controlled robot moving to map an unknown feature while another stationary robot provides additional measurements. The plot elements are described in the text. The resulting path is much different than the one achieved when only one robot is present (Fig. 12). Note that the final robot path naturally results in the two robots maintaining an approximately 90° separation around the target, a geometry which would be expected to produce superior estimates given the sensor capabilities.

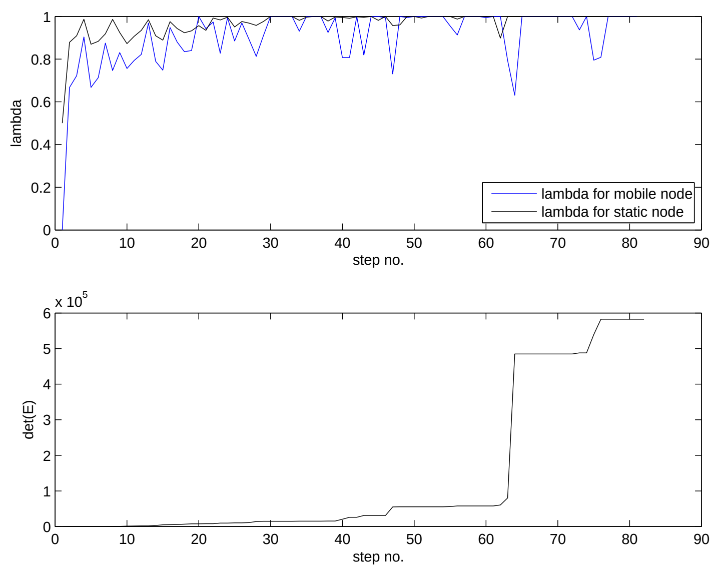


Figure 16: The fusion parameters and determinant of the moment matrix for the single mobile robot and single stationary robot case.

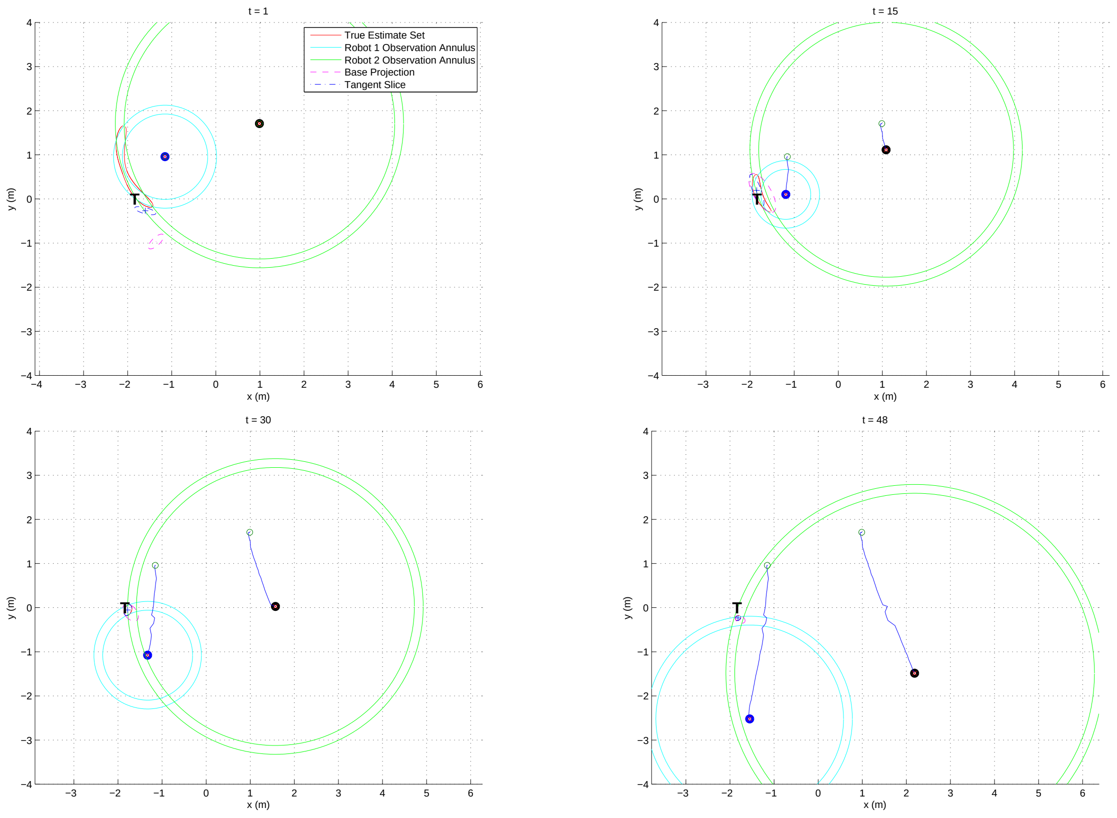


Figure 17: Four snapshots of two controlled robots moving to map an unknown feature. The plot elements are described in the text. The path taken by the first robot is different again from the paths taken in Figs. 12 and 15. Yet again, the paths result in measurements being taken from positions of 90° separation around the target. Neither robot is aware of the existence of the other; their effects on each other occur solely through their effects on the estimate ellipsoid.

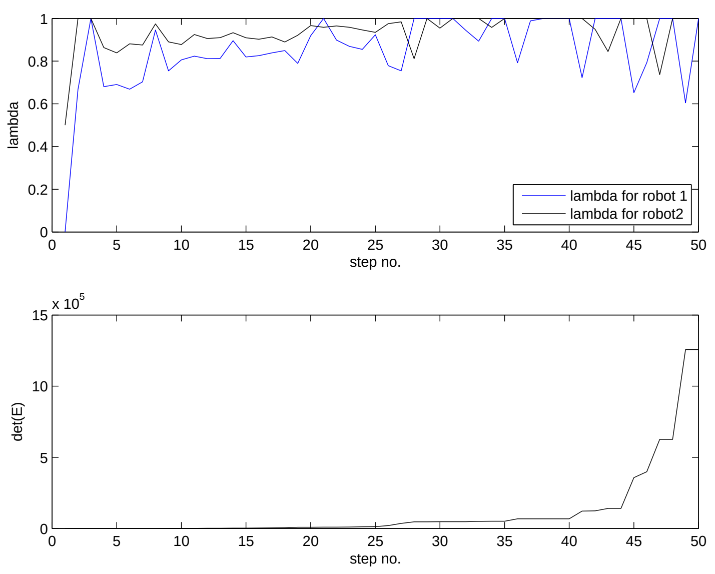


Figure 18: The fusion parameters and determinant of the moment matrix for the two mobile robot case.

Chapter 3

Visibility-Based Planning

In this chapter we will discuss the problem of deploying and moving a team of robots to maintain visibility amongst a set of independent agents. After discussion the definitions of these concepts, we will describe the special problem discretization used for addressing our computations. With this discretization in place, we will find both the number and positioning of robots in order to create visibility connections between agents, and discuss how we can tailor our definition of edge weights in order to establish preferences for these deployments. From the discussion of static deployments, we look at how to sequence these deployments and find sequences which minimize the total movement of the robot team. With these structures in place, we will discuss how to build controllers to achieve these deployments in the next chapter.

3.1 Background

Here we discuss the modeling of the two-dimensional environment and our notions of visibility and visibility chains.

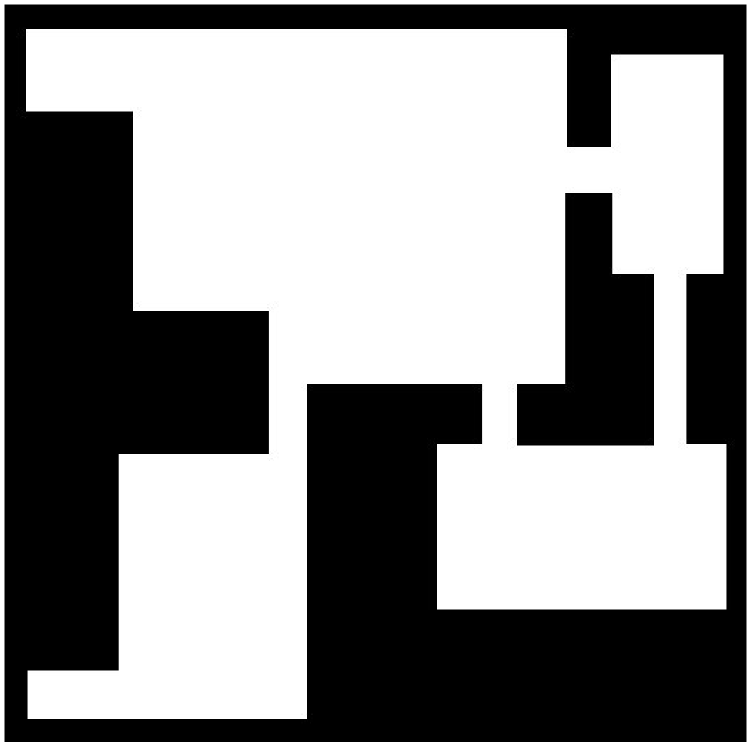


Figure 19: An example of a "complex" environment.

3.1.1 Environment Description

Open-field tracking problems are quite well-known, so we instead assume that all actions take place within a non-convex, multiply-connected environment such as that shown in Fig. 19 where line-of-sight and motion are severely restricted. Formally speaking, the environment is a set $\mathcal{E} \subset \mathcal{R}^2$. We will further require that the environment is polygonal.

By treating the agent and base station as equivalent to robots, we collect the positions of all the entities in the system into the indexed set $\mathcal{X}$. We use the notation that Roman indices range over the entire set, so $\boldsymbol{x}_i \in \mathcal{X}$ can correspond to any robot, the agent, or the base station. Greek indices correspond to distinguished entities: $\boldsymbol{x}_\alpha \in \mathcal{X}$ is the position of the agent and $\boldsymbol{x}_\beta \in \mathcal{X}$ is the position of the base station.

3.1.2 Notions of Visibility

Two points, $\boldsymbol{x}$ and $\boldsymbol{y}$, are considered to be in line-of-sight if the straight-line path connecting them is contained within the environment: $conv(\{\boldsymbol{x}, \boldsymbol{y}\}) \subset \mathcal{E}$. To this end we define the line-of-sight fuction:

$$\mathcal{L}_{\mathcal{E}}(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & conv(\{\boldsymbol{x}, \boldsymbol{y}\}) \subset \mathcal{E} \\ 0 & otherwise \end{cases}$$

The Visibility Matrix, A, for the entire system is given by computing the pair-wise visibility of members of $\mathcal{X}$. So the ij-th element is given by:

$$A_{ij}(\mathcal{X}) = \mathcal{L}(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{18}$$

In Graph Theory, our Visibility Matrix can be considered an Adjacency Matrix, and the Visibility Laplacian can be defined as:

$$L(\mathcal{X}) = D - A$$

where D is a diagonal matrix defined as the row-sum of V:

$$D_{ii} = \sum_{j \neq i} A_{ij}$$

Our requirement of having a “line-of-sight chain” from the agent to the base station can now be formalized as requiring that there is a path between the nodes α and β in the graph described by A. However, we now tighten this requirement by extending it to all robots in the system, motivated by the desire to not “lose” any robots. To this end, we require that the graph described by A is *connected*, so that all nodes $\boldsymbol{x}_i$ have a path back to the base station $\boldsymbol{x}_\beta$. This connectivity has two equivalent conditions: first that the second-smallest eigenvalue of L is non-zero, and second that for each $\{i, j\}$-pair there is some k such that the corresponding entry of the visibility matrix raised to this power is non-zero, $[A^k]_{ij} \neq 0$.

As discussed in Section 1.2.2, we are making use of a formation graph and we are defining our graph quantity of interest to be λ_2 and seeking positions that make the adjacency matrix of Eq. 18 lead to a constrained $\lambda_2 > 0$.

3.2 Polygonal Decomposition

Since visibility between points in $\mathcal{X}$ would need to be computed repeatedly in order to perform actions with A, it is helpful to relax the direct visibility computations and instead make use of *visibility cells.* A visibility cell is a polygonal subset, $V_i \subset \mathcal{E}$, whose visibility with respect to other cells can be expressed as a whole rather than on a point-to-point basis.

The visibility cells form a partition of $\mathcal{E}$, so:

$$\bigcup_i V_i = \mathcal{E}$$

and they only intersect in lines, so that $dim(V_i \bigcap V_j) = 1$.

3.2.1 Calculating Mutual Visibility

We use the term *mutual visibility* to describe that any point within one cell is visible to any point within another, thus the line-of-sight function can be extended to visibility cells:

$$\mathcal{L}_{\mathcal{E}}(V_i, V_j) = \begin{cases} 1 & conv(\{\boldsymbol{x}, \boldsymbol{y}\}) \subset \mathcal{E} \quad \forall \boldsymbol{x} \in V_i,\, \boldsymbol{y} \in V_j \\ 0 & otherwise \end{cases}$$

This gives rise to the *mutual visibility graph* described by the mutual visibility matrix, Q:

$$Q_{ij} = \mathcal{L}_{\mathcal{E}}(V_i, V_j)$$

We collect the visibility cells into an indexed set $\mathcal{V}$ with size $|\mathcal{V}| = M$ and note that M depends on the particular decomposition of $\mathcal{E}$.

3.2.2 Types of Decompositions

We make use of two different decompositions:

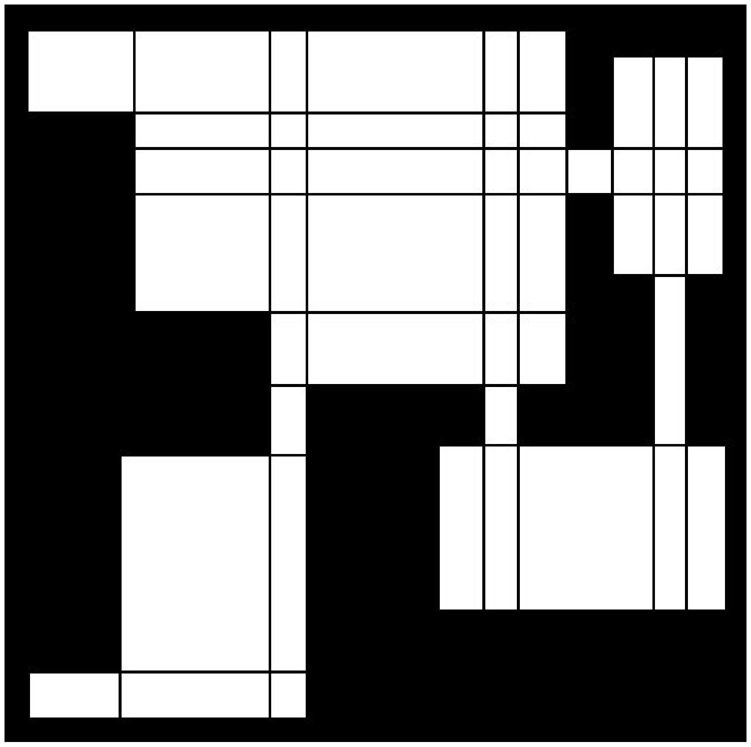


Figure 20: Inflection Decomposition

- *Inflection Decomposition*: An inflection vertex of the polygonal environment $\mathcal{E}$ is simply vertex whose internal angle is less than π. The inflection rays are the two rays that are tangent to the vertex coming from either direction, propagated outward until they hit the environment boundary; by adding the rays from all inflection vertices, the environment is subdivided into the convex cells desired. In an environment with only right-angles, this results in rectangular cells as shown in Fig. 20.

- *Aspect Decomposition*: If the inflection decomposition is further subdivided by including the bitangent lines connecting pairs of visible inflection vertices, the resulting set of visibility cells forms the aspect decomposition. It is named such because if the surrounding environment is scanned from within each cell, the

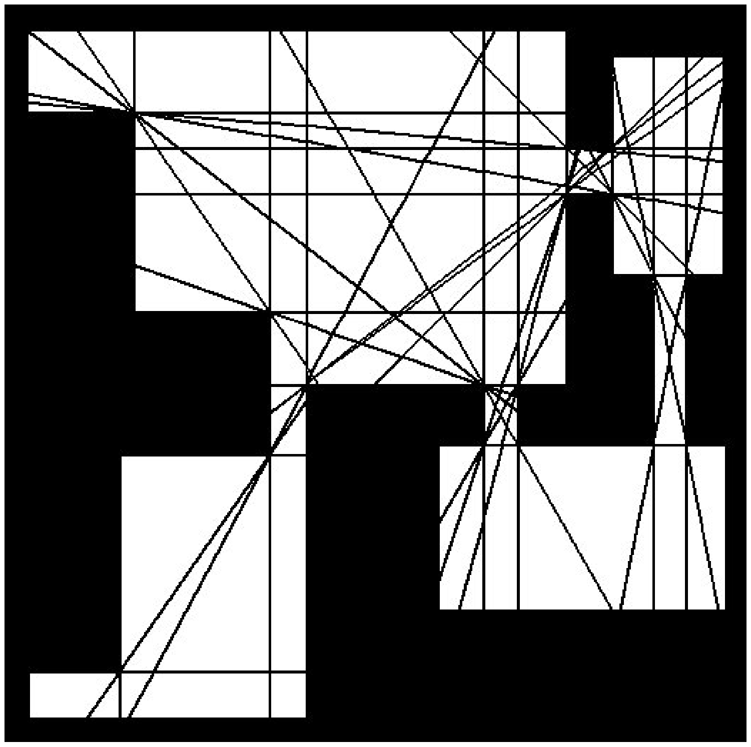


Figure 21: Aspect Decomposition

resulting topology of gaps is identical within each cell but changes incrementally between cells. A variation of this is the R-aspect decomposition, where inflection vertices are only connected if their Euclidean distance from each other is no greater than R. The aspect decomposition is a finer subdivision of the inflection decomposition, as seen in Fig. 21 compared to Fig. 20.

Two cells i and j are considered adjacent if $dim(V_i \bigcap V_j) = 1$, i.e. they share a boundary line but sharing a common point, such as a corner, does not make them adjacent. For the three decompositions above, we have a theorem to relate the adjacency of their cells with mutual visibility.

Theorem 3.2.1 *Adjacent cells are mutually visible under both the inflection decomposition and aspect decomposition.*

This theorem has implications on the mutual visibility graph:

Corollary 3.2.2 *If adjacent cells are mutually visible under Theorem 3.2.1, then connectivity of the environment $\mathcal{E}$ implies connectivity of the mutual visibility graph over its decomposition.*

Under these two results, then we can be assured that all regions of the environment can be seen from other regions of the environment and that we can discretize the problem of maintaining a line-of-sight chain to a base station into one of maintaining paths in the mutual visibility graph instead. It is important to note here that this is a conservative version of visibility; it is possibly to find two points who are visible to each other but whose cells are not mutually visible. However, our visibility model requires this sacrifice of strict visibility in order to make the problem tractable.

For the decompositions seen in Fig. 20 and Fig. 21, the corresponding mutual visibility graphs (MVGs) are seen in Fig. 22.

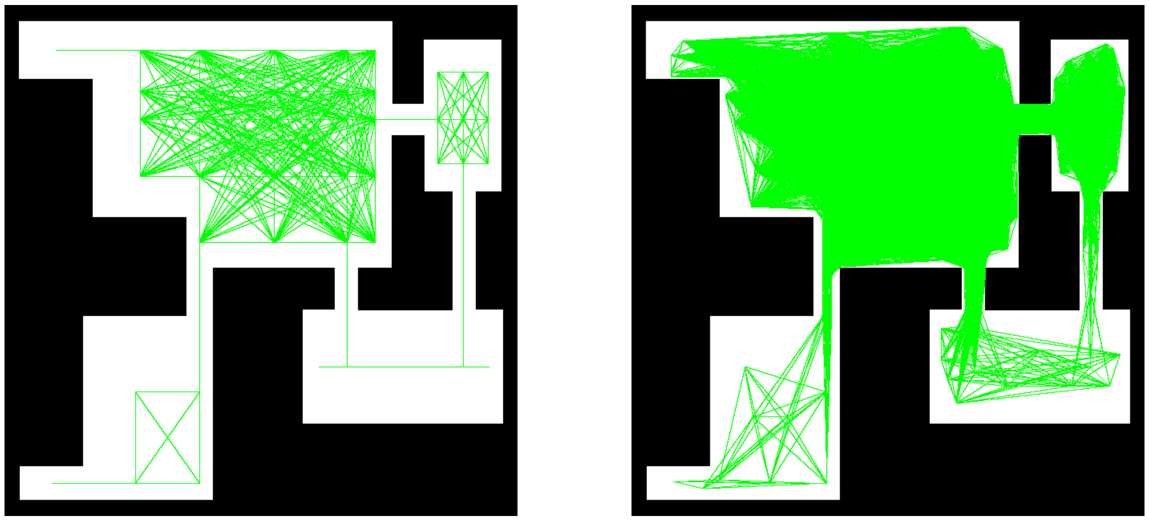


Figure 22: Mutual Visibility Graphs for the decompositions seen in Figs. 20 and 21.

3.2.3 Computation of Decompositions

Though polygon decomposition is usually handled with a Voronoi decomposition in order to obtain reasonably-shaped convex pieces, we are interested in building decompositions from arbitrary intersections of segments. To this end, we have developed a routine that methodically intersects segments and pieces together polygons. The idea is shown in Fig. 23 and summarized here.

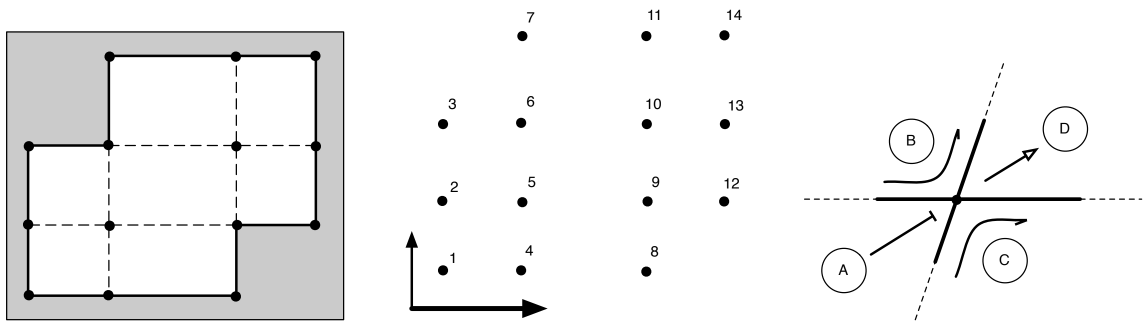


Figure 23: Steps in the decomposition of an environment into polygons. The environment and all dividing segments are added into a large set of segments that are then collectively intersected. The intersections are ordered from left-to-right and then bottom-to-top. The intersections are processed in order, and each one allows for the closure of an existing partial polygon (region A), the addition of a new bounding segment to an existing partial polygon (regions B and C), or the formation of a new partial polygon (region D).

The environment is already represented as a set of line segments. To this set are added all of the desired inflection and aspect lines emanating from reflex vertices. The entire set of segments is intersected against itself to produce a set of all of the intersections, seen in the left-most panel of Fig. 23. These intersections are ordered from left-to-right and bottom-to-top, as seen in the middle panel of Fig. 23. We traverse through this list of intersections, and as seen in the right-most panel of Fig. 23, we recognize each intersection as either the completion of an existing partial polygon (region A), the addition of a new bounding segment to an existing partial polygon (regions B and C), or the start of a new polygon (region D). By keeping a list of partial polygons and then adding, modifying, or removing finished entries to be placed on a list of finished polygons, we can systematically build all polygons formed from the arbitrary intersection of a set of line segments. The total runtime complexity will be $O(L^2 log(I) + I)$ for L segments and I intersections.

It is appropriate to note here that this problem is also known as finding the faces of an arrangement of line segments in the plane. The presented algorithm is one of two common algorithms used to solve the problem [25], but there is actually a more efficient algorithm [13] that solves the problem in $O(n \log n + k)$ for n line segments and k intersections (not known beforehand). The complexity improvement over our approach is due to more efficient line segment intersection. Another useful solution is a data structure that can do point queries and determine in what face a point lies in $O(\log n)$ time [21], though the faces are not explicitly computed.

3.3 Mutually-Visible Deployments

We define our Mutual Visibility Graph as $G = (C, S)$, where C is the set of visibility cells as nodes and S are the line-of-sight relationships as edges between the nodes.

A *deployment* of agents is a graph structure $D = (V, E, m)$, where V are the agents as nodes of the graph, $E \subset V \times V$ defines a relationship between the agents, and $m : V \to C$ maps the nodes into the visibility cells they are located within. Note that this ties into our synopsis of the unifying structure behind the formation control literature reviewed in Section 1.2.2.

Our deployment is considered a *mutually-visible deployment* if the graph structure (V, E) is a connected graph, and the edges correspond to line-of-sight relationships in the MVG. This means that $\forall(e_1, e_2) \in E$, $(m(e_1), m(e_2)) \in S$.

Defining deployments in this way naturally splits the structure of the robot formation, encoded in (V, E), from their physical locations, given by mapping m. As an example, consider Fig. 24. The robot formation on the left consists of nodes related to each other by visibility, and then this maps into physical locations on the right, where the visibility lines are evident.

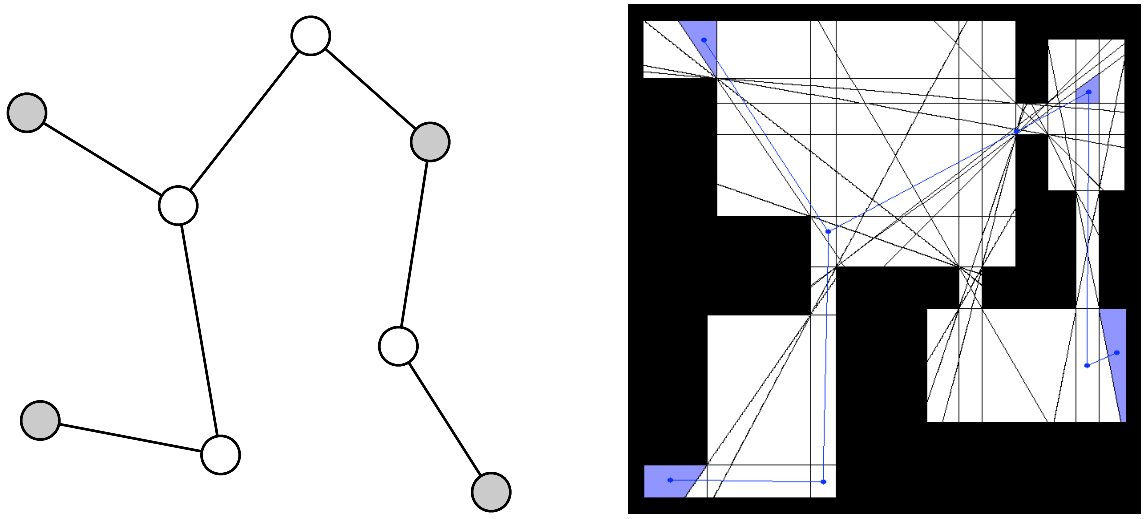


Figure 24: A deployment $D = (V, E, m)$ separates the structure of the robot formation, encoded in (V, E), from its physical location, given by the mapping m. The relationship of the robots between each other, visibility in this case, is shown on the left, and then this formation is mapped into physical locations as shown on the right. The shaded nodes in the graph correspond to the shaded cells in the decomposition.

We are interested in solving for mutually-visible deployments where some of the nodes are agents we are tracking or the fixed ground station, with their locations given. The additional nodes correspond to robots we need to deploy to maintain visibility between all agents.

Finding mutually-visible deployments proceeds in two steps: first, we solve for shortest paths connecting just two agents, and then extend this for multiple agents.

3.3.1 Virtual Edge Weights

The most critical piece of solving for mutually-visible deployments is designing the edge weights that give rise to the cost of the shortest paths and Steiner trees. These weights give us the opportunity to define what we consider to be desirable configurations of robots. We will focus on three separate considerations: keeping a tight deployment, making use of larger visibility cells whenever possible, and using as few robots as possible.

To this end, we define our virtual edge weight as follows, with each of the three costs to be discussed next:

$$W_v(i,j) = c_a(i,j) + c_b(i,j) + c_c(i,j) \tag{19}$$

Distance Cost

For keeping a tight deployment, we would like to prefer that given two pairs of visible cells, the pair that are closer together are given preference. To this end, we will define the first cost term as the distance between the centroids of the visibility cells involved:

$$c_a(i,j) = k_a \| cent(C_i) - cent(C_j) \|_2 \tag{20}$$

where k_a is a coefficient used to weight this first cost term versus the others.

Area Cost

Since we would like to prefer pairs of mutually visible cells where one or both of the cells are larger, we need a cost term that scales inversely with cell area. Two costs that achieve this goal are a simple inverse area and a negative exponential:

$$c_b(i,j) = k_b \left(\frac{1}{area(C_i)} + \frac{1}{area(C_j)} \right) \tag{21}$$

$$c_b(i,j) = k_b \left(exp(-area(C_i)) + exp(-area(C_j)) \right) \tag{22}$$

where k_b is again a coefficient to weight this cost term relative to the others.

To compare these two, Fig. 25 presents a histogram of the inverse-area cost, while Fig. 26 presents a histogram of the negative-exponential cost, both taken over all mutually-visible cell pairs in the aspect decomposition of Fig. 21.

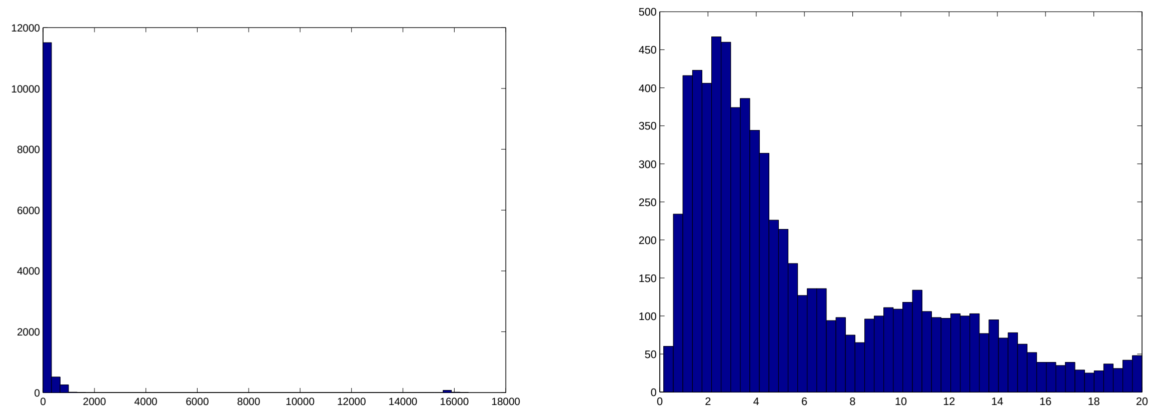


Figure 25: Histogram of the inverse area measure for the MVG of the decomposed environment seen in Fig. 21. The view on the right is a truncated version since the distribution has a heavy tail. The mean is 191.02 and the standard deviation is 1.383e3.

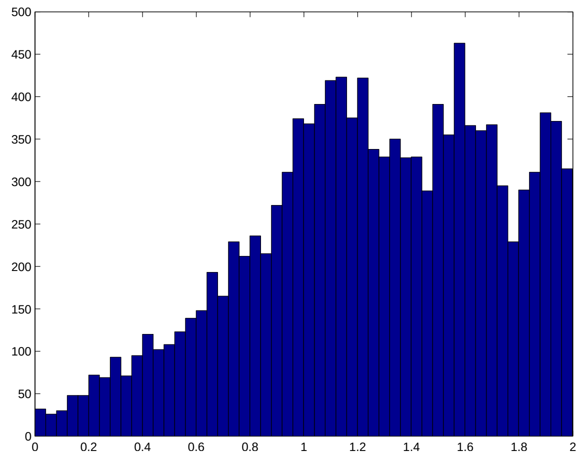


Figure 26: Histogram of the negative exponential measure (d_e) for the MVG of the decomposed environment seen in Fig. 21. The mean is 1.25 and the standard deviation is 0.46. Note that by definition the measure cannot exceed 2.

The inverse area cost results in very explicit separation between visibility pairs with nominal areas and those with very small areas, as evidenced by the heavy tail on the distribution. The negative exponential cost, by comparison, is very smooth and has a hard limit of 2 by definition.

We will make use of the negative exponential cost. The relative weighting between the negative exponential and centroid distance costs will be discussed shortly.

Minimum Deployment

The preference to minimize the robot count is realized by simply adding a large penalty term to each edge. The need for this is seen in Fig. 27. Without a penalty term, many small hops are preferred to a few large hops because the closer centroid distances result in a shorter path. Though including costs related to the area can mitigate this to some extent, we want the preference to be strong. To this end, we define the deployment cost as:

$$c_c(i,j) = Z, \tag{23}$$

$$Z > \sup_{i,j}(c_a(i,j) + c_b(i,j)) \tag{24}$$

By ensuring that is Z is larger than the other costs combined, we are guaranteed that at no point will it be preferable to use two visibility hops where one will suffice.

Combining Costs

Consider the histograms of the centroid distance and negative exponential area costs in Fig. 28. We need to combine the two to make the edge weights prefer both tight deployments and the use of larger visibility cells. However, the relative scaling between the two is important since weighting one too heavily will make the other meaningless. Set $k_a = 1$ and consider just the changing of k_b. The combined cost $W_v(i,j) = \|cent(C_i) - cent(C_j)\|_2 + k_b(exp(-area(C_i)) + exp(-area(C_j)))$ for various values of k_b is seen in Fig. 29. As k_b is increased, the distribution changes from

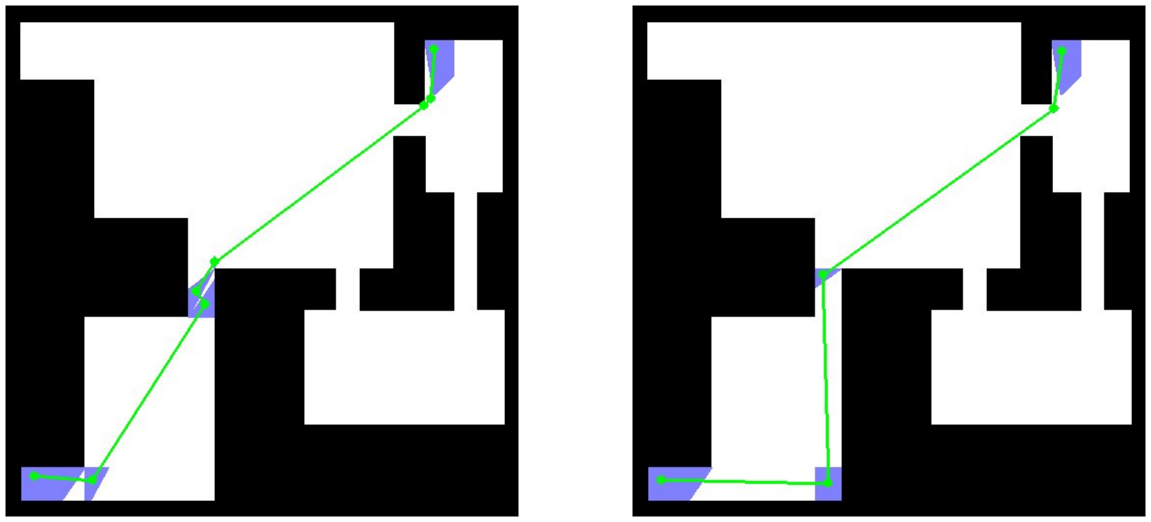


Figure 27: Using only centroid distances as the edge weighting, the resulting shortest path can involve taking many small hops instead of fewer large hops. By adding a large penalty term to each edge, in this case the length of the diagonal of the entire area, it will always be preferable to use the smallest number of hops.

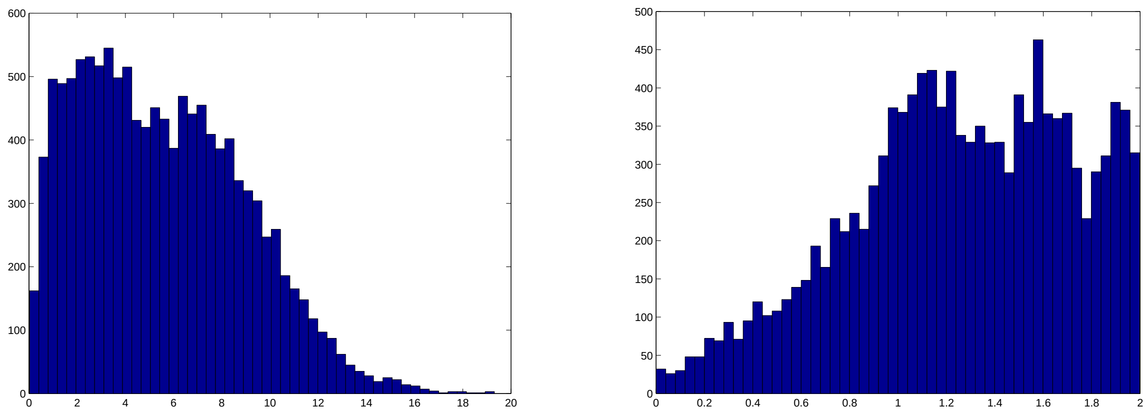


Figure 28: The statistics for the centroid distance (left) and negative exponential area (right) costs from the MVG for the decomposition of Fig. 21. We need to mix the two to obtain a cost that stresses both tight deployments and the use of larger visibility cells.

that of the centroid distance alone to that of the negative-exponential area alone. Expressing the preferences of both requires some in-between histogram.

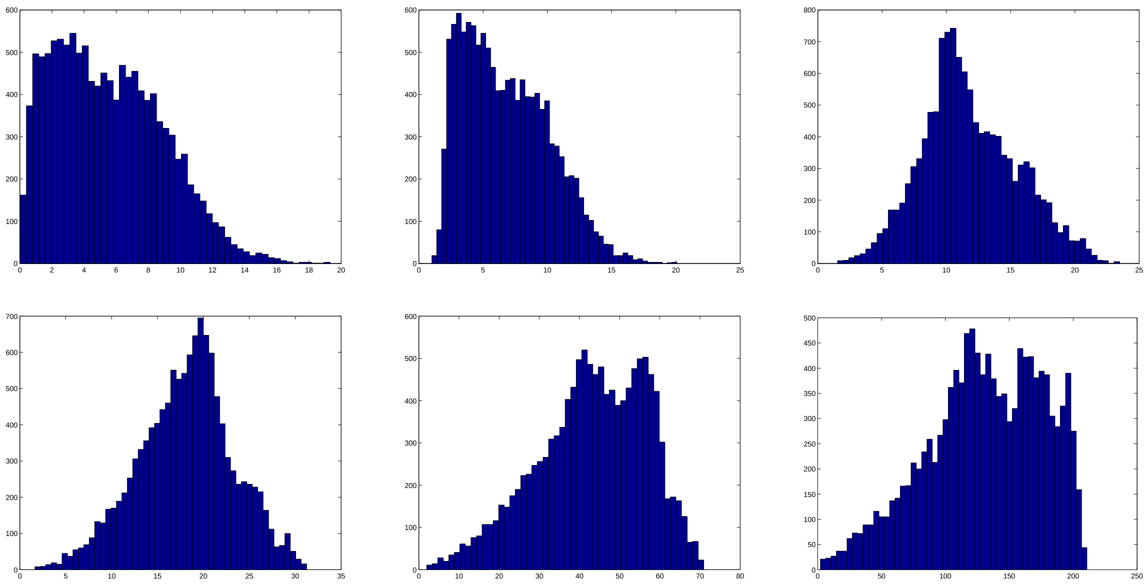


Figure 29: The statistics using the measure $W_v(i,j) = \|cent(C_i) - cent(C_j)\|_2 + k_b(exp(-area(C_i)) + exp(-area(C_j)))$ for a range of k_b values. Across from top left to bottom right, the values are $k_b = 0$, 1, 5, 10, 30, and 100. Note how the distributions go from that of the left histogram of Fig. 28 to the right histogram as k_b becomes larger and the negative-exponential-area measure is more represented (albeit scaled).

The ultimate goal is to manage the relative weighting so that both costs are of the same magnitude on average. One way to do this is to divide each cost by its maximum value to normalize it, and then they can be averaged together on equal scaling. This approach results in the histogram of Fig. 30. Note that the result looks similar to a Gaussian.

The normalized approach results in the following virtual edge weight:

$$W_v(i,j) = \frac{1}{2}\left(\frac{\|cent(C_i) - cent(C_j)\|_2}{\max_{i,j}\|cent(C_i) - cent(C_j)\|_2}\dots \right. \tag{25}$$

$$\left. + \frac{exp(-area(C_i)) + exp(-area(C_j))}{\max_{i,j}(exp(-area(C_i)) + exp(-area(C_j)))}\right) + 1.01 \tag{26}$$

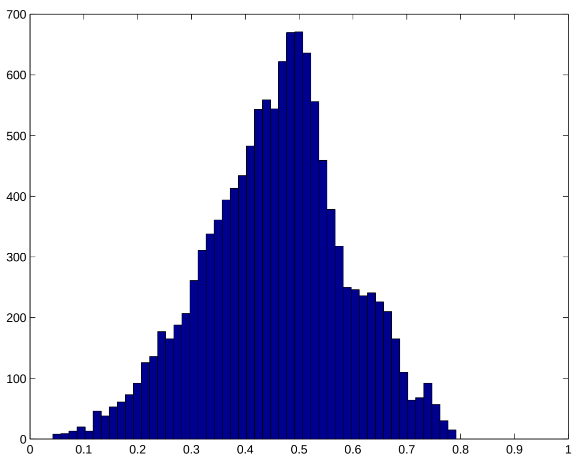


Figure 30: Each cost (c_a, c_b) was normalized by dividing by its max and then the total measure was taken as the average of the two, resulting in this almost Gaussian distribution.

combining all of the desired preferences into a weight that balances between tighter deployments while preferring larger cells and minimizing the number of deployed robots.

3.3.2 Shortest Paths

Given the locations of the base station and independent agent, reduced to their corresponding cell locations, we can deploy robots to create a visibility chain by choosing placing them in cells creating a path in the MVG. We will restrict ourselves to using the shortest path calculated with the virtual edge weights discussed in Sec. 3.3.1. Such a path will fulfill the priorities outlined, namely a geographically compact deployment using the fewest number of robots and preferring cells with larger areas.

We therefore treat the shortest path as the complete solution to the deployment problem in the case of a single robot requiring connection to a fixed base station. Examples can be seen Fig. 31.

The shortest path from one cell to all other cells can be computed using the widely-known Dijkstra Algorithm. However, since we will need to compute shortest

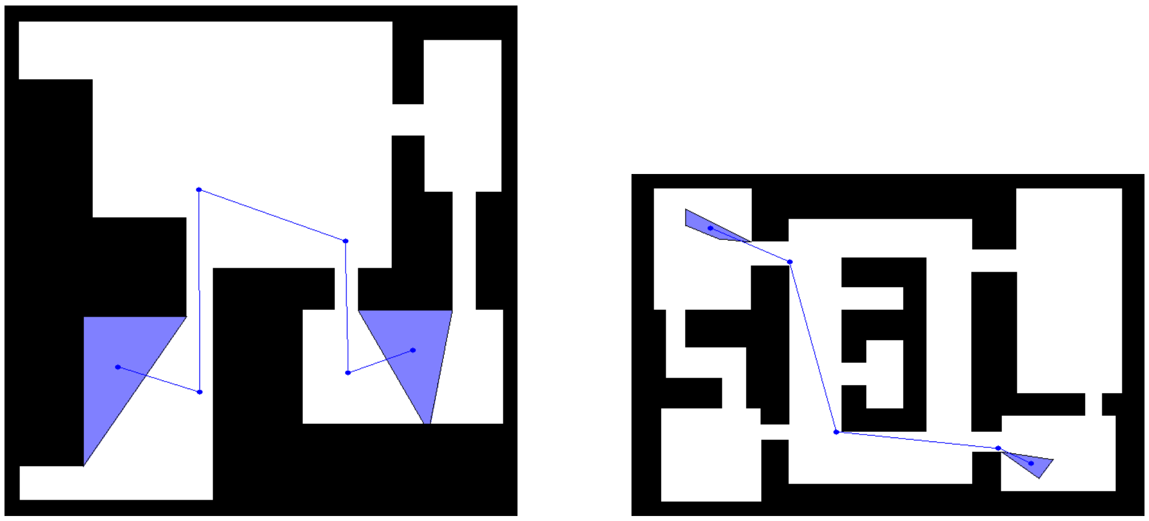


Figure 31: Examples of shortest paths in two environments using the virtual edge weights of Sec. 3.3.1. A shortest path completely solves the problem of deploying robots to maintain a visibility chain between a single agent and the base station.

paths repeatedly for algorithms that follow, it is useful to perform the calculation of the shortest paths between all cells at once, handled by the Floyd-Warshall algorithm or by repeatedly using the Dijkstra algorithm beginning from different starting cells. Both have a complexity of $O(|C|^3)$ with the possibility of some reduction using more sophisticated data structures.

3.3.3 Steiner Trees

Steiner Trees are an extension of the shortest path problem: given a set of terminals, find the minimum-weight tree to connect all of them. If the set consists of only two terminals, the problem reduces to the shortest path problem. Any non-terminal nodes that are included are known as *Steiner nodes* and can be considered as intermediaries that help maintain connectivity between the terminals.

If the terminal locations are considered to be a set of target agents together with the base station, then the Steiner Tree over this terminal set solves the problem of deploying the robot team to cover multiple targets.

Using the virtual weight W_v on the edges of G, our Steiner Tree is a mutually-visible deployment $T^\star = (V, E, m)$ where:

$$T^\star = \min_{(V,E,m)} \sum_{(e_1,e_2)\in E} W_v(m(e_1), m(e_2)) \tag{27}$$

This is a hard problem because not only are we optimizing over the placement of the nodes, m, but also over topologies of the nodes, (V, E). It will be shown that if we pick a topology, the problem can be solved efficiently (though the final result may not be the true Steiner tree if the topology was not the optimal one). Before that, we will solve the Steiner tree problem exactly.

Finding the Optimal Steiner Tree

To find the optimal topology and mapping, we use the Dreyfus-Wagner algorithm, a dynamic programming approach first given in [24] and reviewed in [42, 80]. We already know that the optimal tree for two terminals is just the shortest path between them. The optimal tree for three terminals can be constructed from the optimal trees for each of the pairs of terminals using one of a few simple operations. Then we construct the optimal tree for four terminals using the three-terminal solution, and so on, until we have the optimal tree for all terminals.

Following the notation of [80], we begin with the set of nodes of the entire graph, V, and define a terminal set $K \subseteq V$. Take any subset of these terminals, $X \subseteq K$, and some external node $v \in V \backslash X$, then we define two optimal tree costs: $s(X \cup \{v\})$ is the cost of the minimum Steiner tree for the set of terminals $X \cup \{v\}$, and $s_v(X \cup \{v\})$ is the cost of the minimum Steiner tree for the terminal set $X \cup \{v\}$ where we require that v have a degree of at least two in the tree constructed.

The reason for defining a cost such as s_v is that we can write it as the sum of the costs of two smaller trees, as seen in Fig. 32. Since v is of degree at least two, then its removal would break the tree over X into two or more subtrees. If we state that the tree over $X \cup v]$ is optimal, then these two subtrees must also be optimal by the

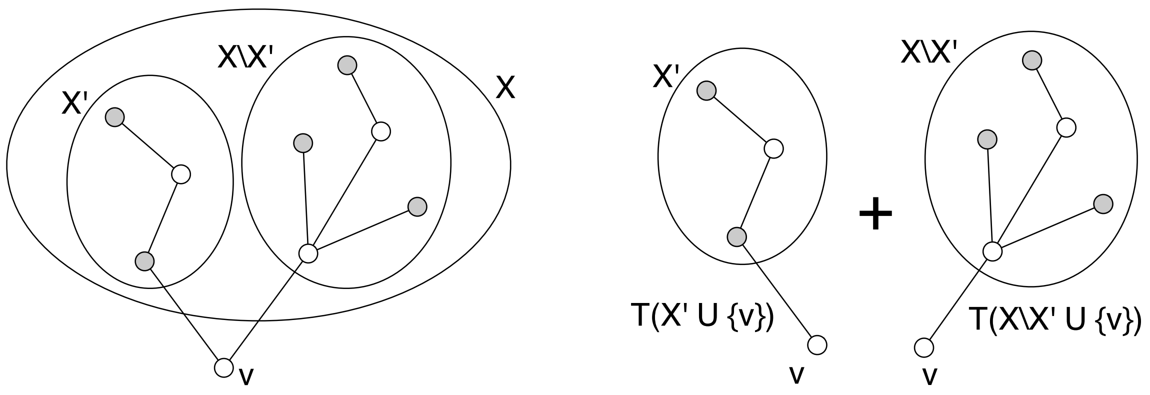


Figure 32: The split-tree-cost $s_v(X \cup \{v\})$ is composed of two costs. Since v is defined to be of degree at least two, then it links two smaller subtrees X' and $X\backslash X'$. We find the cost by summing the optimal costs of both of these subtrees.

principle of optimality. So suppose we knew which terminals belonged to each tree: $X' \subset X$ are the terminals of one half of the split, and $X\backslash X'$ are the other. We can find the split-tree-cost $s_v(X \cup \{v\})$ in terms of the optimal costs of both of these subtrees:

$$s_v(X \cup \{v\}) = s(X' \cup \{v\}) + s(X\backslash X' \cup \{v\}) \tag{28}$$

However, we do not know the optimal split beforehand, so we simply try every possible split of $X' \subset X$ to find which one gives the smallest cost. Note that we require $X' \neq \emptyset$ and X' to be a strict subset of X, so we can state that $|X' \cup \{v\}| < |X \cup \{v\}|$ and $|X\backslash X' \cup \{v\}| < |X \cup \{v\}$. Because of this, we can see that we are finding the cost of the tree over a given terminal set in terms of the costs of trees over smaller terminal sets. Since we are building this computation from small terminal sets to larger, the optimal costs over these smaller terminal sets is already available.

Though we can find the optimal cost for $X \cup \{v\}$ while requiring v to have degree greater than two, we need an expression for the optimal cost over $X \cup \{v\}$ without conditions. We obtain this by recognizing that we can connect v into the optimal tree over X in three ways, as shown in Fig. 33. Either v connects into one of the terminals $w \in X$, v connects into some other node $w \notin X$ belonging to the optimal

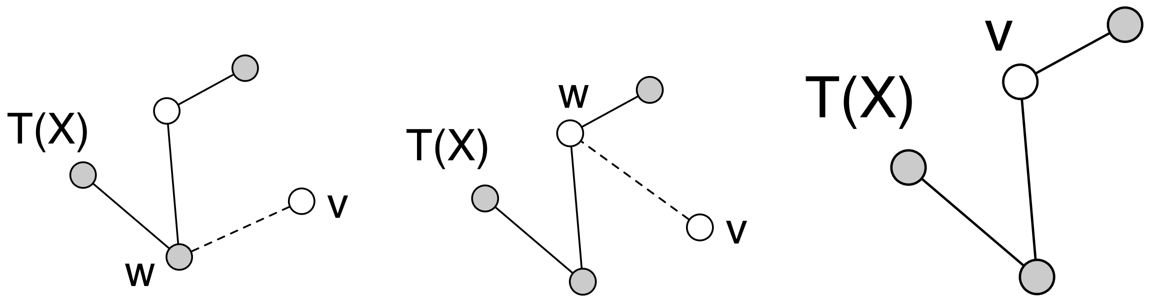


Figure 33: The optimal tree over the set $X \cup \{v\}$ relates to the optimal tree over X in one of three ways. Either v connects to one of the terminals $w \in X$, v connects to a node $w \notin X$ that has degree at least two in the tree over X, or v was already part of the optimal tree over X. Note that the third case is really just a special case of the second where we let $w = v$.

tree over X, or v is already part of the optimal tree over X. The third case is really just the second case where we let $w = v$. It may happen that there is no direct connection between v and w, in which case we connect v into the tree using the shortest path instead since it represents the lowest cost connection possible. It is important to note that in the second case, the node w must necessarily have degree at least two in the optimal tree otherwise it would be a leaf and thereby one of the original terminals and covered under the first case. Because of this, we can write the cost of this type of tree in terms of the split-tree-cost s_v defined earlier. We find the optimal tree over $X \cup \{v\}$ by considering all the possible ways to connect v into the tree over X and then taking the one with least cost.

Let $p(a, b)$ be the cost of the shortest path between nodes a and b, perhaps found using Dijkstra's algorithm. We can iterate through all nodes $w \in V$, find their cost associated with a tree where v connects to w depending on whether w is part of X or not, and then take the minimum over all of them:

$$s(X \cup \{v\}) = \min_{w} \begin{cases} p(v, w) + s(X), & w \in X \\ p(v, w) + s_w(X \cup \{w\}), & w \notin X \end{cases} \tag{29}$$

The full Dreyfus-Wagner algorithm, as given in [80] is reproduced in Algorithm

1.

Algorithm 1 DreyfusWagner(V, E, W_v, K)

Require: A graph (V, E) with edge costs W_v, and terminal set $K \subset V$

Ensure: The cost $s(K)$ of the minimum Steiner tree over K.

```
1: for $a, b \in V$ do
2:     compute p(a,b)
3: end for
4: for $\{x, y\} \subseteq K$ do
5:     $s(\{x, y\}) = p(x, y)$
6: end for
7: for $i = 2$ to $|K| - 1$ do
8:     for $X \subset K$ with $|X| = i$ do
9:         for $v \in V \backslash X$ do
10:            $s_v(X \cup \{v\}) = \min_{\emptyset \neq X' \subset X} [s(X' \cup \{v\}) + s((X \backslash X') \cup \{v\})]$
11:        end for
12:    end for
13:    for $X \subset K$ with $|X| = i$ do
14:        for $v \in V \backslash X$ do
15:            $s(X \cup \{v\}) = \min_w \begin{cases} p(v, w) + s(X), & w \in X \\ p(v, w) + s_w(X \cup \{w\}), & w \notin X \end{cases}$
16:        end for
17:    end for
18: end for
19: return $s(K)$
```

In order to produce a tree using DreyfusWagner, a backtracking system is added that records which tree-joining case gave the optimal cost and which nodes were connected. From this information, the optimal Steiner tree can be constructed.

Finding an Approximate Steiner Tree

We will make use of two algorithms for finding approximate Steiner trees. In each case, we are guaranteed that the cost of the approximation is no worse than twice that of the true optimal [80].

Minimum-Spanning-Tree Approximation

The minimum-spanning-tree approximation was described by many researchers independently, see [80]. The idea is to find the minimum-spanning-tree within the distance network taken over the terminal set, expand the results into the union of the corresponding paths, and then clean up the resulting tree.

A distance network is a construct where we build an abstract graph over a weighted graph where all nodes are connected to all other nodes with an edge whose weight is the length of the shortest path between the two nodes. For instance, Fig. 34 shows a simple weighted graph and the complete graph forming the distance network over its nodes with some of the edge weights given. All of the weights correspond to the length of the shortest path between the two adjacent nodes. A distance network can be defined for any graph and for any subset of nodes of a graph, all that is required is a way to find shortest paths between nodes.

There is a direct correspondence between edges in the distance network and shortest paths in the graph, as seen in Fig. 35. Any edge can be expanded into its corresponding shortest path.

The other construction needed is a *minimum spanning tree* over the graph (V, E): the minimum weight subgraph that connects all of the nodes of V. This can be found using standard efficient algorithms such as Prim's algorithm [77] or Kruskal's algorithm [52].

With these two computations, we can perform the Minimum-Spanning-Tree-Steiner (MST-Steiner) algorithm, shown in Algorithm 2.

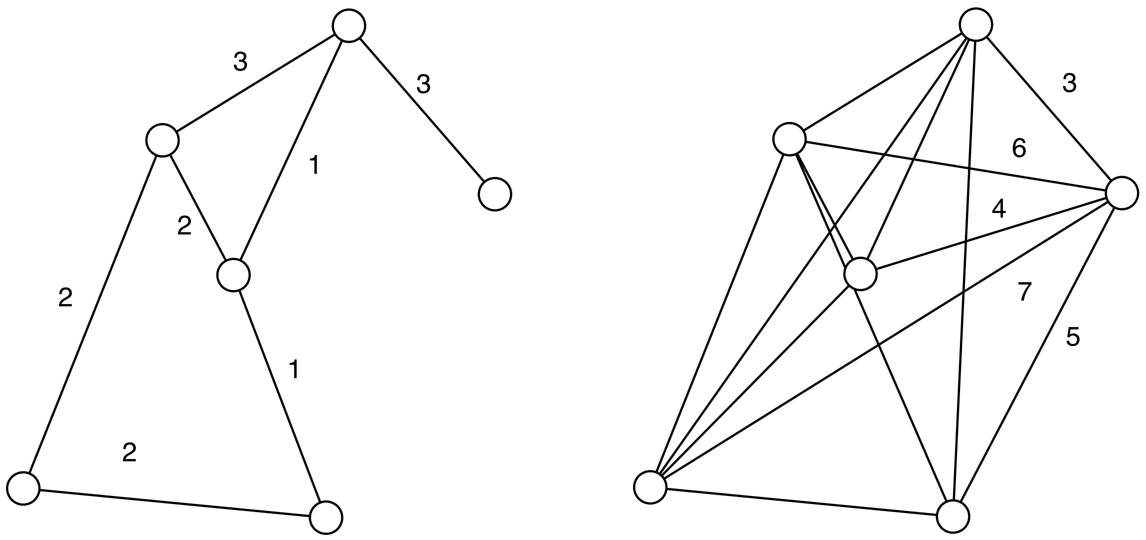


Figure 34: Given the weighted graph on the left, the distance network over its nodes is shown on the right with some of the edge weights labeled. A distance network is a complete graph over the chosen nodes where the edge weights correspond to the length of the shortest path between the adjacent nodes, but constructed in the original graph.

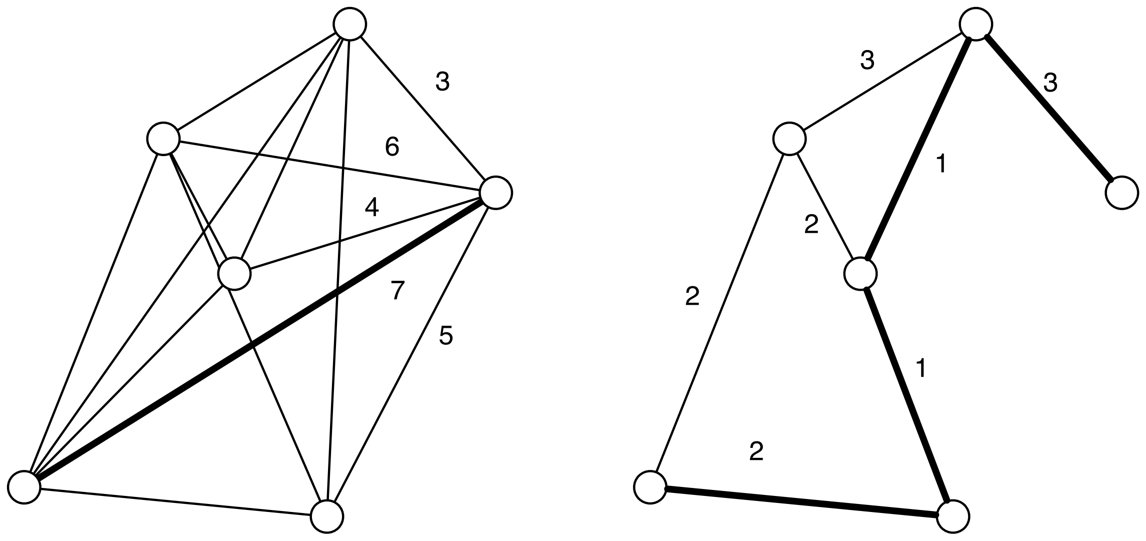


Figure 35: There is a direct correspondence between edges of the distance network and shortest paths in the original graph. The highlighted edge on the left can be expanded into the shortest path shown on the right.

Algorithm 2 MST-Steiner(V, E, W_v, K)

```
Require: A graph $(V, E)$ with edge costs $W_v$, and terminal set $K \subset V$
Ensure: A Steiner tree over the terminal set $K$ with approximately optimal cost.
1: Compute the distance network $N_D$ over $K$ in $V$.
2: Compute the minimum spanning tree $T_D$ within $N_D$.
3: Transform the edges of $T_D$ into their corresponding shortest paths in $(V, E)$ to
   form a subnetwork of $(V, E)$.
4: Compute a minimum spanning tree $T$ over this subnetwork.
5: Remove all leaves of $T$ that are not terminals.
```

Takahashi-Matsuyama Approximation

Another simple approximation was described by Takahashi and Matsuyama [96]. The basic idea is to incrementally build an approximate minimal Steiner tree by starting with the shortest path between two terminals and then incrementally connecting each other terminal into the growing tree using the shortest path to any of the nodes of the tree. The basic idea is shown in Fig. 36. Each node is successively connected by its shortest path.

The algorithm was compactly described in [80] and is repeated here in Algorithm 3.

3.3.4 Finding Deployments with Known Topology

As suggested in Sec. 3.3.3, if the topology of a deployment is known, then finding the placements is efficient. Here we develop a dynamic programming approach for finding the m that minimizes the total cost of the deployment $D = (V, E, m)$.

Since we assume that the topology (V, E) is given, the cost of the deployment (V, E, m) is:

$$C(D) = \sum_{(e_1, e_2) \in E} W_v(m(e_1), m(e_2))$$

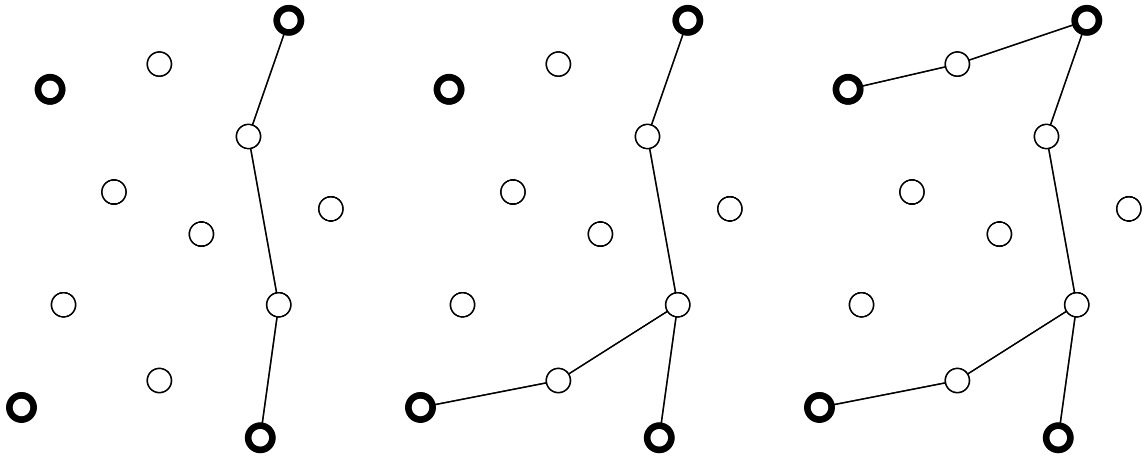


Figure 36: An approximate minimum Steiner tree is constructed using the Takahashi-Matsuyama algorithm by successively connecting each terminal into a growing tree by the shortest path to any of the nodes of the tree. Here, two of the highlighted nodes are connected by their shortest path, and then the remaining highlighted nodes are attached one-by-one.

Algorithm 3 TakahashiMatsuyama(V, E, W_v, K)

```
Require: A graph $(V,E)$ with edge costs $W_v$, and terminal set $K \subset V$
Ensure: A Steiner tree $S_K$ over the terminal set $K$ with approximately optimal
    cost.
1: Choose any $v \in K$ and set $S_K = \emptyset$, $X = \{v\}$, $L = K \backslash \{v\}$
2: while $L \neq \emptyset$ do
3:     Find the $w \in L$ whose shortest path $P_w$ to some node of $X$ is smallest.
4:     Shrink $L$ to $L \backslash \{w\}$, then expand $X$ to include the vertices of $P_w$ and add
       $P_w$ to $S_K$.
5: end while
```

or, recognizing m as the only variable:

$$C_{(V,E)}(m) = \sum_{(e_1,e_2)\in E} W_v(m(e_1), m(e_2)) \tag{30}$$

We will suppress the (V, E) subscript since the topology is fixed in all that follows.

We will assume that the virtual cost of an edge between two nodes placed in the same cell is zero, and also that the given topology is a tree $T = (V, E)$ where the leaves are all target agents whose positions are fixed. These fixed locations provide the constraints needed to solve for the tree uniquely.

We will assume, without loss of generality, that in nodes are indexed from 1 to n and so we will denote the mapping m by its action on the various indices: $m_i = m(i)$. We have the constraint that for node indices corresponding to target locations, $\mathcal{T}$, the mapping is already known, which we will write as $m_{\mathcal{T}} = \{M_{\mathcal{T}}\}$ to cover the assignment of all the mapping entries of the set. The optimal cost is then the minimization over all possible mapping assignments under this constraint:

$$C^{\star}(T) = \min_{m|m_{\mathcal{T}}=\{M_{\mathcal{T}}\}} C(m) \tag{31}$$

We will suppress the $m_{\mathcal{T}} = \{M_{\mathcal{T}}\}$ notation with the assumption that all further cost calculations are done under this constraint.

Because the topology is a tree, removing any edge results in the formation of two trees. If the edge is removed between two adjacent nodes i and j, then we obtain two trees: $T^{i\backslash j}$ and $T^{j\backslash i}$ to signify the tree including node i (resp. j) formed by removing the edge to node j (resp. i). This is illustrated in Fig. 37. We can write the optimal cost of the tree in terms of the optimal cost of the two subtrees together with the cost of the edge we removed:

$$C^{\star}(T) = \min_{m} \left(C^{\star}(T^{i\backslash j}) + C^{\star}(T^{j\backslash i}) + W_v(m_i, m_j) \right) \tag{32}$$

This leads to the idea of conditionally optimal costs: the cost associated with choosing the optimal mapping with conditions, written as $C^{\star}_{i\to m_i}(T)$ to indicate the

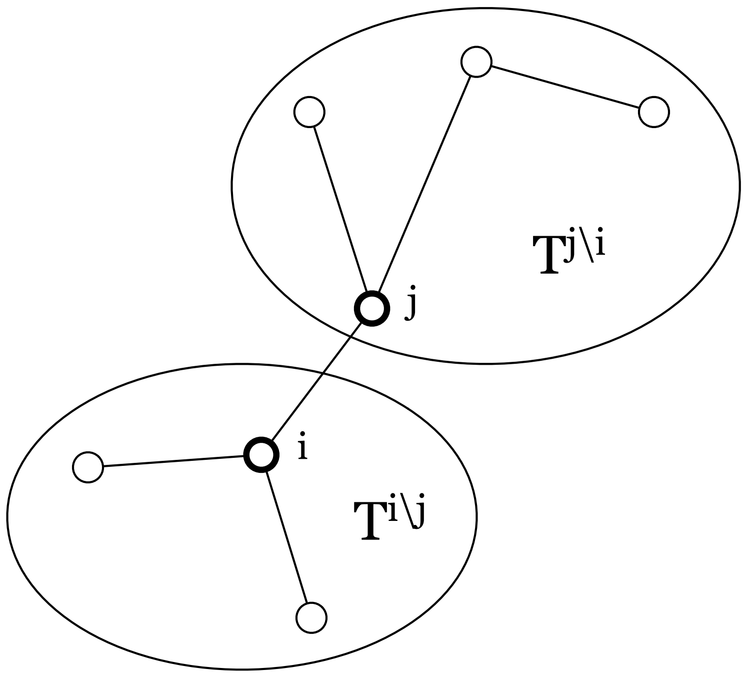


Figure 37: By removing an edge between two nodes i and j, the tree T is split into two subtrees we call $T^{i\backslash j}$ and $T^{j\backslash i}$ to indicate the tree formed by taking node i (resp. j) and removing the edge connecting it to node j (resp. i). The cost of the total tree can be expressed as the sum of the costs of the two subtrees plus the cost of the removed edge, and we make use of this to derive a recursive formulation of the cost.

optimal cost of tree T with the requirement that node i mapped into cell m_i. For example, we may decide that node i is mapped into cell number 42 of the visibility decomposition and then want the cost associated with choosing all other mappings to be optimal. If we were to take a particular node and then find the conditionally optimal costs for all possible assignments of that node, we can take the minimum one and recover the optimal cost:

$$C^{\star}(T) = \min_{m_i} C^{\star}_{i \to m_i}(T) \tag{33}$$

Using this idea, we can rewrite Eq. 32 as a minimization over the choices of the mappings for nodes i and j:

$$C^\star(T) = \min_{m_i,m_j} \left(C^\star_{i\to m_i}(T^{i\backslash j}) + C^\star_{j\to m_j}(T^{j\backslash i}) + W_v(m_i,m_j) \right) \tag{34}$$

If we look from the point of view of node i, then we could also write this as:

$$C^\star(T) = \min_{m_i} \left(C^\star_{i\to m_i}(T^{i\backslash j}) + \min_{m_j} \left(C^\star_{j\to m_j}(T^{j\backslash i}) + W_v(m_i,m_j) \right) \right) \tag{35}$$

The conditionally optimal cost of $T^{i\backslash j}$ does not depend on m_j since $T^{i\backslash j}$ does not contain node j. Finally, we are finding the optimal cost by minimizing over all possible m_i, but if we pick a particular m_i, we are simply finding the conditionally optimal cost for this selection. Thus we have:

$$C^\star_{i\to m_i}(T) = C^\star_{i\to m_i}(T^{i\backslash j}) + \min_{m_j} \left(C^\star_{j\to m_j}(T^{j\backslash i}) + W_v(m_i,m_j) \right) \tag{36}$$

This leads to a recursion because Eq. 36 can be used again with $T^{i\backslash j}$ becoming T since $T^{i\backslash j}$ is just another tree.

The final piece of the recursion is the fact that some of the mappings are already fixed because of the motion of the targets, so they have no conditionally optimal cost. Mathematically speaking, we can set to infinite the conditionally optimal cost of them going to any spot other than the one that they actually did go to. This handles the $m_{\mathcal{T}} = \{M_{\mathcal{T}}\}$ from earlier.

If the environment decomposition has N cells, so that each node has a number of assignments of $O(N)$, then for each of the $O(N)$ possible assignments we are checking against $O(N)$ assignments of each neighbor. For k nodes, there are $k-1$ links, so the total complexity will be $O(kN^2)$ for the full computation.

We will build upon this procedure to include motion constraints and costs in Section 3.5.1. The link is that we can recover the fixed-topology Steiner tree calculation by letting the movement costs go to zero.

3.4 Deployment Sequences and Costs

Given the mutual visibility graph, $G = (C, S)$, computed for a given environment decomposition, we define a mutually visible tree $T = (V, m, E)$ as a connected set of nodes, V, with labels, $m : V \to C$, identifying which cell of C they belong to, and an edge set E connecting each node with other mutually visible nodes, as identified by the LoS restrictions of S, in such a way that the resulting graph is a tree.

A tree sequence $\mathcal{T} = T_1, T_2, \ldots, T_n$ is an indexed set of these mutually visible trees defined over G.

Given a weighting $W_v : S \to \mathbb{R}$ on the edges of G, a weight $W_t : E \to \mathbb{R}$ is induced by mapping the adjacent vertices of an edge to their corresponding cells in C and using the weight of the connecting edge in S. This leads to a *virtual* cost of a tree associated with the "virtual" cost of visibility that we have created to influence agent deployments into cells with better size or proximity. This virtual tree cost is defined as:

$$C_v(T) = \sum_{e \in E} W_t(e) = \sum_{(e_1, e_2) \in E} W_v((m(e_1), m(e_2))) \tag{37}$$

Given a physical weighting function $W_p : C \times C \to \mathbb{R}$ relating the cost of physical relocation between cells (discussed shortly in Sec. 3.4.1), we can define the *physical* cost of transition between two trees by looking at the costs for the nodes of one tree to move from their corresponding cells to the cells of the nodes of another. What this entails is figuring out how to match the nodes of one tree to the other in such a way as to minimize the total cost of moving and then find this cost. Assuming that the tree nodes are all zero-indexed, then this really amounts to finding an optimal permutation amongst all permutations of n integers, P_n. We call this an identification function, $I : V_i \to V_j$, identifying the nodes of one tree T_i with another tree T_j. All possible identifications (permutations) belong to the set $\mathcal{I}$. Using this in conjunction with the cell labels and physical weighting function, we define the physical cost as:

$$C_p(T_i, T_j) = \min_{I \in \mathcal{I}} \sum_{v \in V_i} W_p(m_i(v), m_j(I(v))) \tag{38}$$

Though this definition is defined in terms of a minimization that would seem to be exponential because of the need to look at permutations of the n nodes, this is in fact an *assignment problem*, also known as the *bipartite matching problem* because you are choosing the edges to match the nodes of the two halves of a bipartite graph such that the total cost is minimized. In our case, the cost is simply the physical cost of moving from the associated cell of a node in the original tree to the associated cell of a node in the new tree. This can be solved in $O(n^3)$ time using a variant of the Hungarian algorithm. Note that solving this also requires finding the assignment, which is useful in actually commanding the mobile agents to transition between the two trees.

To apply these cost functions to the entire tree sequence, they are simply applied to each element or sequential pair of elements:

$$C_v(\mathcal{T}) = \sum_{i=1}^{N} C_v(T_i) \tag{39}$$

$$C_p(\mathcal{T}) = \sum_{i=1}^{N-1} C_p(T_i, T_{i+1}) \tag{40}$$

These two costs must be combined in order to give a total sequence cost:

$$C(\mathcal{T}) = C_p(\mathcal{T}) + \mu C_v(\mathcal{T}) \tag{41}$$

where the non-negative parameter μ defines the relative weighting between the two costs. High values of μ leads to a preference (lower cost) for graph sequences with trees that consist of agents in cells with better proximity, size, etc, as defined by our virtual visibility weights W_v. Low values of μ leads to a preference (lower cost) for graph sequences where the node transitions are not far according to the physical costs W_p. Note that this is a fundamental tradeoff; tuning this parameter gives us a way to decide how greedy we are going to be with deployment selection.

In practice, we are more interested in designing a sequence to minimize costs. Given a tree T_i, we can find the subsequent tree T_{i+1} that minimizes the cost by considering just the transition cost and the cost of the next tree since the cost of T_i is already fixed:

$$T_{i+1} = \arg\min_{T \in \mathcal{T}} \left[C_p(T_i, T) + \mu C_v(T) \right] \tag{42}$$

Also, assuming the physical cost W_p is symmetric, this process could be applied in either direction along the tree sequence finding either the previous or subsequent tree to minimize costs.

3.4.1 Physical Edge Weights

Analogous to the virtual edge weights described in Sec. 3.3.1, physical edge weights allow us to specify preferences between cell-pair choices, but now we are indicating how unfavorable it is to require the robot to move from one cell to another cell. The most logical choice is to make use of Euclidean distance: if the centroids of two cells are far apart, it will take longer for the robot to traverse between the two and hence has higher cost. The corresponding weight would be $W_p(i, j) = \|centroid(C_i) - centroid(C_j)\|_2$.

However, this weight is less meaningful if the cells are not directly visible since the robot would need to navigate around obstacles to move between the two. In this case, it would make more sense to use the shortest path between the centroids and then take the length of that path as the physical cost. In Sec. 4.2, both the geodesic and adjacency path are calculated. The geodesic is the true shortest path between two points that respects the environment boundary, while the adjacency path is the shortest path between two cells formed by only traversing between adjacent cells and its length is taken as the sum of the length of the segments joining the centroids of subsequent cells. Either length will give some notion of the cost of traversing between cells that do not have direct line-of-sight.

3.5 Deployment Sequence Optimization

Given the definitions of a deployment sequence and its associated cost, we seek to design a deployment sequence that minimizes the total cost. This problem has practical significance: we know the starting locations of our agents and intermediary robots and what the final deployment should look like based on the locations of the agents, but we need to create the sequence of mutually-visible deployments in-between the two. Doing so will ensure that our system is mutually-visible at each intervening step that we calculate.

We can make these time horizons as small as we want by using smaller transitions for the agents, addressing the problem that we may not know the final destination of the agents and therefore the final deployment tree needed. We can settle for incremental deployments, computing deployment sequences between small agent shifts.

As noted in the calculation of deployments with fixed topology, this problem has two components: picking a topology and deciding where the nodes are placed. With the topology fixed, the placement problem is easy, and we extend the results of Sec. 3.3.4 to handle the physical costs associated with moving the nodes. We will find the optimal deployment assuming that one is possible with the given topology. If there is no deployment possible using the fixed topology, some heuristic approach for changing the topology is needed. This is discussed in Sec. 3.6.

Incorporating movement costs into the Steiner tree calculation represents an unexplored problem that we call finding a *minimal movement Steiner tree*. The main structural difference is that a traditional minimal Steiner tree problem considers edge costs alone but for consideration of movement we must incorporate per-node costs as well. Another way to think of it is that given a set of target locations, you can compute a Steiner tree to find the optimal way to connect them, but you may be willing to trade-off the optimality of the final tree for a reduced cost of moving robots to form the tree. Our approach to performing this tradeoff is to introduce a parameter μ that scales the edge-weight contribution to the final cost and then perform a dynamic programming step to solve for the optimal placements with a fixed topology. Using Sec. 3.6, we can explore the space of possible topologies to optimize the cost even more. But as will be seen later, searching all topologies requires exponential time and so we will make a heuristic simplification.

3.5.1 One-step Sequence Optimization

Equation 42 states the optimization problem that needs to be solved to perform the 1-step tree design. Though it seems difficult, if the structure of the new tree is known, we can solve this problem using dynamic programming. This approach can incorporate both the virtual and physical costs associated with the new tree and can work with constraints on cell location of the nodes of the new tree. This is a direct extension of the technique of Sec. 3.3.4 where we consider minimizing the edge costs of the new deployment but now we have an additional cost in the form of the physical cost for a node to move from its existing position to a new one.

We find this by declaring that the conditionally optimal cost of a single-node tree is simply the physical cost of the conditioned move of node i from its current position m_i^0 to its new one:

$$C^\star(T_{i\to m_i}) = W_p(m_i^0, m_i) \tag{43}$$

From Sec. 3.3.4, we have the recursive formula for the conditionally optimal cost of a tree in terms of the two trees formed by removing an edge between node i and j:

$$C^\star_{i\to m_i}(T) = C^\star_{i\to m_i}(T^{i\setminus j}) + \min_{m_j}\Big(C^\star_{j\to m_j}(T^{j\setminus i}) + W_v(m_i, m_j)\Big) \tag{44}$$

where we can add Eq. 43 as the endpoint of the recursion once we've split the tree down into single-node trees.

Though we have defined the optimal costs in terms of arbitrary splits of the tree, it is most useful to split the tree one node at a time since we can directly write the optimal conditional cost of a single node with Eq. 43. Moreover, if a node has several adjacent edges, the contributions of the trees formed by removing each of these edges are independent and can be added together. With these modifications in mind, we can write the final recursion as:

$$C^\star(T_{i\to m_i}) = W_p(m_i^0, m_i) + \sum_{j\in adj(i)} \min_{m_j} \left[C^\star(T^{j\setminus i}_{j\to m_j}) + \mu W_v(m_i, m_j) \right] \tag{45}$$

where $adj(i)$ are the nodes adjacent to i in tree T. We find the conditioned cost $C^\star(T^{j\setminus i}_{j\to m_j})$ of $T^{j\setminus i}$ by using this same Eq. 45 replacing T with $T^{j\setminus i}$.

This is demonstrated in Fig. 38. We pick some arbitrary node i and then write it's conditionally optimal cost in terms of the sum of the conditionally optimal costs of each child node j. However, we first need to compute this conditionally optimal cost so we recurse into the subtree and repeat the procedure with the child node j now treated as the base node i and using the conditionally optimal costs of this node's children j's. We keep recursing down the tree and querying for the conditionally optimal cost until we are left with a single-node tree, whose conditionally optimal cost we know from Eq. 43 as just the physical cost. With this in hand, we can unwind back through the tree and compute the conditionally optimal costs with minimizations over the children costs until we end up back at the original base node. Its minimum conditional cost is the minimum cost for the entire tree.

A note on design: constraints on the problem can be introduced through design of the cost functions W_p, W_v. If a pair of cells are not visible to each other (adjacent in the mutual visibility graph), then set their virtual cost W_v to infinite. This automatically removes them from consideration as viable choices for adjacent cells in T, unless there is no realization of T possible, in which case the optimal cost becomes infinite. If a movement constraint is desired so that certain cell choices are infeasible because of range or possibly nonholonomic conditions, then the physical cost W_p can be made infinite for that particular cell.

The algorithms are given in Algs. 4, 5, and 6 and described here.

Algorithm 4 InitializeRedeployment($\boldsymbol{m}^0, \boldsymbol{m}^c, W_p$)

Require: A set of current node placements $\boldsymbol{m}^0$, a set of constrained locations $\boldsymbol{m}^c$ where the targets are to be placed, and physical weights W_p between placement locations.

Ensure: A cost structure C of physical costs alone.

```
1: $C \leftarrow \{\}$
2: for $i = 1$ to $k$ do
3:     if $i$ is a constrained target in $\boldsymbol{m}^c$ then
4:         $C(i,j) = \infty,\ \forall j \neq m_i^c$
5:         $C(i, m_i^c) = 0$
6:     else
7:         $C(i,j) = W_v(m_i^0, j)$
8:     end if
9: end for
10: return $C$
```

Algorithm 5 Redeployment(C, W_v, T, μ, v, a)

Require: A cost structure C incorporating physical costs, virtual edge weights W_v, a tree topology T, a mixing parameter μ, the current node v, and the ancestor node a.

Ensure: The cost structure C is correct for the subtree $T^{v\backslash a}$.

```
1: for $j \in \mathcal{N}_T(v),\ j \neq a$ do
2:     Redeployment($C, W_v, T, \mu, j, v$)
3:     for $m_v = 1$ to $N$ do
4:         $C(v, m_v) \leftarrow C(v, m_v) + \min_{m_j} [C(j, m_j) + \mu W_v(m_i, m_j)]$
5:     end for
6: end for
```

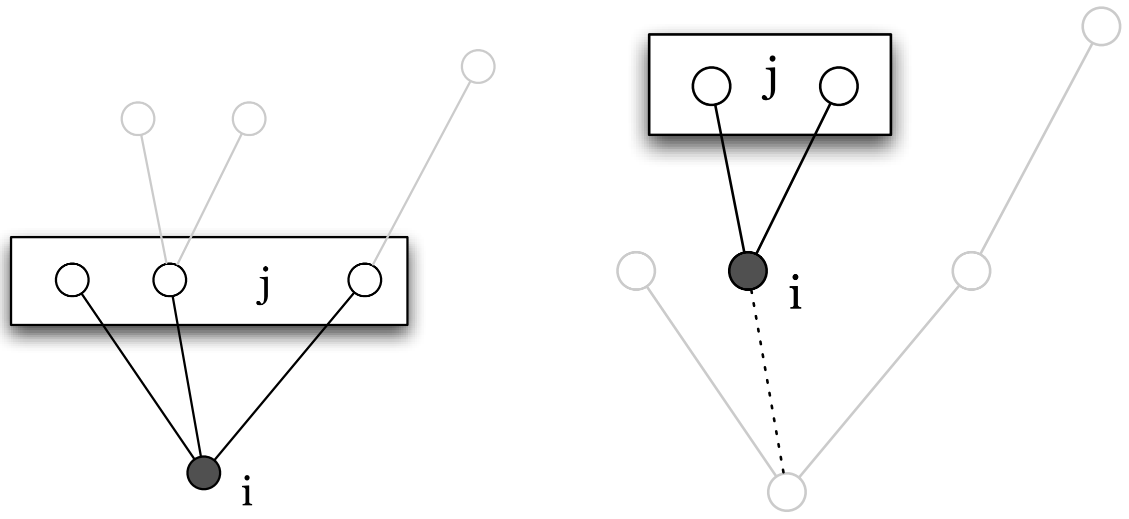


Figure 38: Beginning with some arbitrary node i, it's conditionally optimal cost depends on the sum of minimizations over the conditionally optimal cost of each child node j. We recurse into this node and repeat the procedure, treating it as a base node i querying the conditionally optimal cost of all of its children. This continues until we have obtained a single-node tree, whose conditionally optimal cost is found using just its physical cost as in Eq. 43. This lets us complete the computations back up through the tree.

First the conditionally optimal cost structure is initialized with InitializeRedeployment and all of the physical costs are filled in. For the targets whose movements are fixed, we set the physical cost of their new position to zero and set all other positions to infinity. For all other agents, we set their costs according to the established physical cost function.

With the physical costs entered into the cost structure, we use the recursive function Redeployment to solve for the costs. We can start from any node v of the graph and initially set the ancestor a as empty. The algorithm starts by iterating over each neighbor j of v and ensuring that its conditionally optimal cost is computed. We then minimize over this cost and add it into the conditionally optimal cost of v. When iterating over the neighbors of v, we skip over a in order to establish that we are optimizing the subtree $T^{v \backslash a}$.

Algorithm 6 RedeploymentBacktrack$(m, C, W_v, T, \mu, v, a)$

```
Require: Structure of optimal locations $m$, the conditionally optimal costs $C$, virtual edge weights $W_v$, a tree topology $T$, a mixing parameter $\mu$, the current node $v$, and the ancestor node $a$.
Ensure: The optimal locations $m$ are correct for the subtree $T^{v\backslash a}$.
1: if $a$ is $\emptyset$ then
2:     $m_v = \arg\min_{m_v} C(v, m_v)$
3: else
4:     $m_v = \arg\min_{m_v} [C(v, m_v) + \mu W_v(m_a, m_v)]$
5: end if
6: for $j \in \mathcal{N}_T(v),\ j \neq a$ do
7:     RedeploymentBacktrack$(m, C, W_v, T, \mu, j, v)$
8: end for
```

After the conditionally optimal cost is computed for our chosen start node, we can compute the positions that give rise to this cost by using Redeployment-Backtrack. First the lowest cost position is used for the current node v, and then all of its neighbors are updated, again skipping over the ancestor a to ensure that we are optimizing the subtree $T^{v\backslash a}$.

3.6 Topology Modification

Similarly to [108], we consider the use of topology modification in order to find solutions to the visibility-maintenance problem in the case where the current topology is suboptimal or infeasible for keeping the targets mutually-visible.

Our approach is very simple and is demonstrated in Figure 39: take the current tree topology and remove a single edge, producing two trees. Then connect these two trees by adding an edge between every pair of vertices except the original. For a tree $T = (V, E)$, the set of trees produced by removing edge (i, j) is denoted $\bar{T}_{(i,j)}$

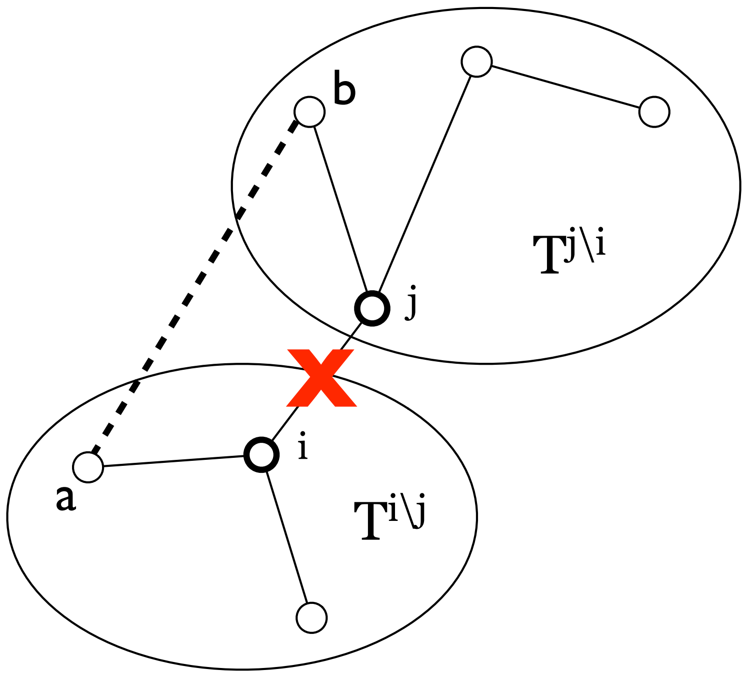


Figure 39: We systematically create new tree topologies by removing an edge (i, j) from the original tree and then adding edges between all vertices of the two trees produced by the deletion.

and is found by:

$$\bar{T}_{(i,j)} = \left\{ (V, E - (i,j) + (\alpha,\beta)) \;\middle|\; \forall(\alpha,\beta) \neq (i,j),\ \alpha \in V(T^{i\backslash j}),\ \beta \in V(T^{j\backslash i}) \right\} \quad (46)$$

The full set of single-edge modifications of T is found by:

$$\tilde{T} = \bigcup_{(i,j)\in E} \bar{T}_{(i,j)} \quad (47)$$

and by checking $\tilde{T}$ and T, we have broadened our search for ways to fulfill the mutual visibility requirement. The OneStepTopologyModification procedure is shown in Algorithm 7.

This modification of the redeployment is interesting because it is a step towards a solution to the problem of finding a *fixed-size Steiner tree*, or an optimal Steiner Tree with a specified number of vertices. This is a completely new problem that has not

Algorithm 7 OneStepTopologyModification(T)

Require: A tree $T = (V, E)$

Ensure: A set of trees $\tilde{T}$ found from T by a single edge-modification.

```
1: $\tilde{T} \leftarrow \emptyset$
2: for $(i, j) \in E$ do
3:     for $(m, n) \in V \times V,\ (m, n) \neq (i, j)$ do
4:         $\tilde{T} \leftarrow \tilde{T} \cup (V, E - (i, j) + (m, n))$
5:     end for
6: end for
7: return $\tilde{T}$
```

been previously explored. This approach to solving it falls short, of course, because a complete solution would check all possible tree topologies for the given robot count, and we have only checked those which can be found by one edge-shifting operation away from a given topology. From [12] we know that there are n^{n-2} possible trees that can be constructed over n vertices, so an exhaustive search would be exponential.

3.7 Summary

In this chapter we investigated the problem of deploying a team of robots to maintain a visibility chain with moving targets in a non-convex, multiply-connected environment. We accomplished this by discretizing the environment into polygonal cells and pre-computing visibility information between them. From this, we cast all of our deployment problems as graph computations and used standard algorithms to find the associated graph structures.

We introduced the idea of deployment topologies where the required visibility relationships were separated from the physical robot locations within the environment, and we saw that if the topology is given, finding the optimal robot locations is easy.

This led to an algorithm for computing robot motions to accommodate changing target locations while keeping the deployment topology fixed and devised a heuristic way to check multiple topologies to see if better solutions could be found.

One important building block was devising edge weights for the visibility links within the environment, and we constructed them to express preferences for minimizing robot count, making the deployment as compact as possible, and utilizing larger visibility cells as much as possible.

Now that we have decomposed the visibility-based deployment into a sequence of deployment topology snapshots, we must implement a control strategy that will move the robots between snapshots while preserving visibility between connected robots and avoiding collisions between the agents and both the environment and each other.

Chapter 4

Implementation and Experimentation

In this chapter we will discuss solving the practical realization of the deployments planned in Chapter 3 as well as characterizing the complexity of the algorithms. Although the necessary robot positions have been found to ensure visibility at discrete times during the deployment, the actual paths and control actions have not been found.

There are two goals for the paths and their execution: to maintain mutual visibility during the motion and to avoid collisions between the robots and both the environment and each other. We will realize this goal by having each robot plan a path independently and then sequence the execution of these paths. Two alternate approaches, each satisfying one of the goals, will be presented, and then they will be combined into one unified approach.

The intent is to eventually validate this planning and execution technique through implementation on a team of robots in an uncontrolled environment. As a preliminary step, we will perform simulated deployments to follow a user-controlled agent through an environment. Before this, we will perform a simple experiment to motivate the requirement of visibility for communication.

4.1 Complexity

We will investigate two aspects of the complexity of the problem: first, we will create decompositions and graphs for several example environments in order to investigate the effect that the environment specifics such as vertex count have on the complexity of the resulting graphs. Second, we will perform the various procedures of Chapter 3 and compare the timing against theoretical complexity results.

4.1.1 Environment Complexity

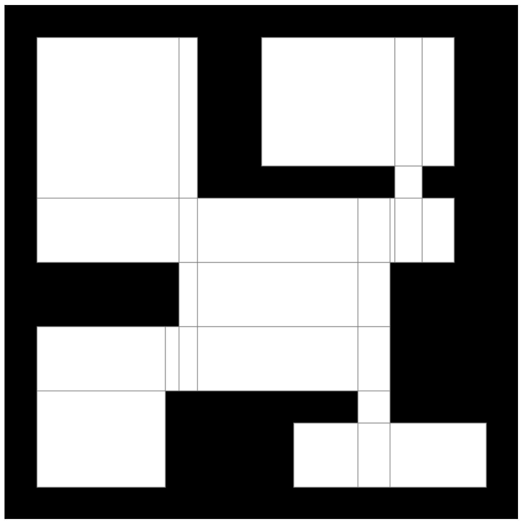
The reflex and aspect decompositions have been performed on several environments shown in Fig. 40 and the resulting metrics are shown in Table 1.

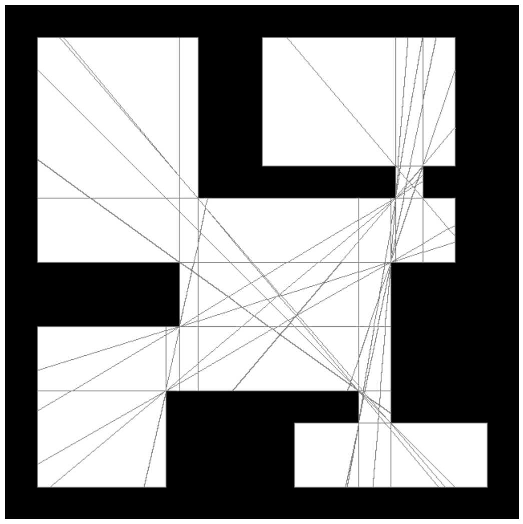
There is a theoretical upper-bound on the number of polygons created using this procedure. Though we are actually computing the faces of an arrangement of line segments, we can bound this by considering the number of faces of an arrangement of lines, and it is well-known that for n lines in the plane, there are $O(n^2)$ faces formed [91]. Trying to specifically bound the complexity of an arrangement of line segments is much more complex [25] because the faces may no longer be convex. In our decomposition, the lines emanate from the reflex vertices; for r reflex vertices, there will be $2r$ lines extending out tangent to them, but $O(r^2)$ bitangent lines joining them, so in total there will be $O(r^2)$ lines. Because of this, we will have no more than $O(r^4)$ faces in our final decomposition, though the actual number is usually much less because many of the reflex vertices will not be visible to each other, leading to no bitangent being placed between them.

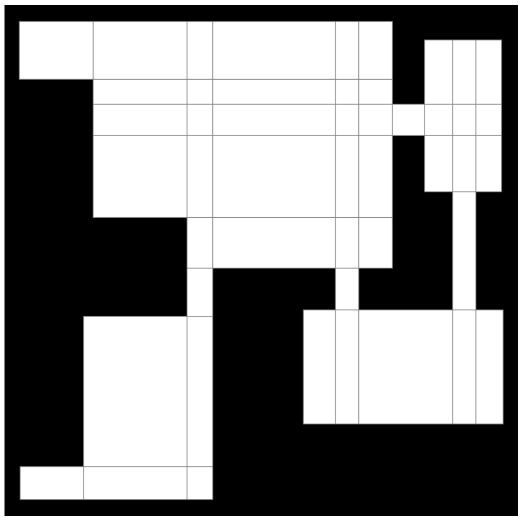
The complexity of computing the decomposition is fairly low. The steps are:

1. Perform an intersection between all segment pairs. This can be slightly optimized by only intersecting pairs whose bounding boxes overlap. For a total segment count of L, this is up to $L(L-1)/2$ checks.

2. Order the intersections lexically by their corresponding point, and process each


(a) *Room1 w/o bitangents*


(b) *Room1 w/ bitangents*


(c) *Room2 w/o bitangents*

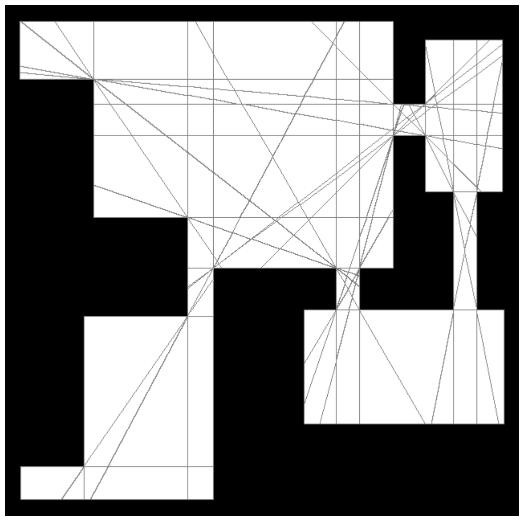
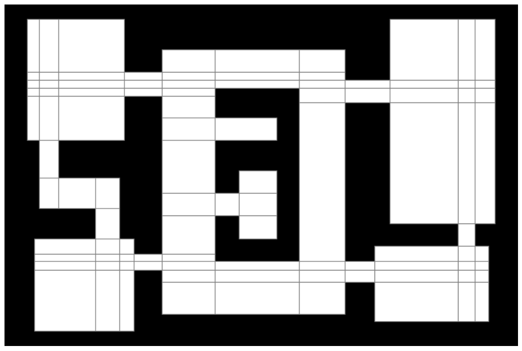
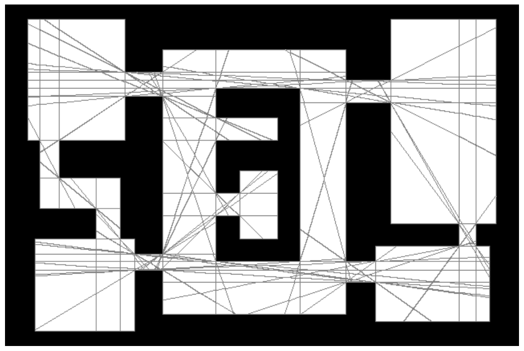

(d) *Room2 w/ bitangents*

Figure 40: The decompositions of sample environments used for testing. Versions with and without bitangents included are shown, but some of the environments were too complex to include bitangents. The stats of the environments are summarized in Table 1.


(e) *Room3 w/o bitangents*


(f) *Room3 w/ bitangents*

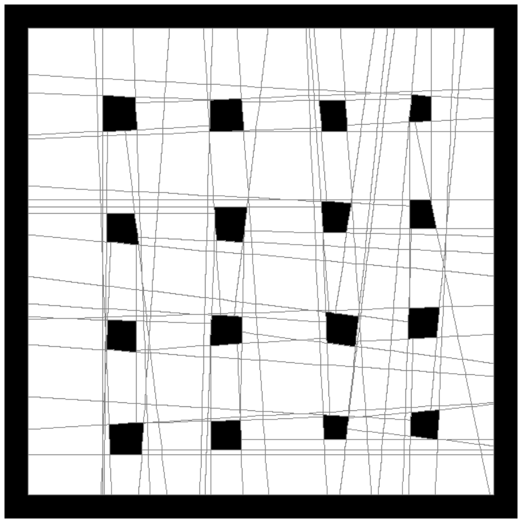
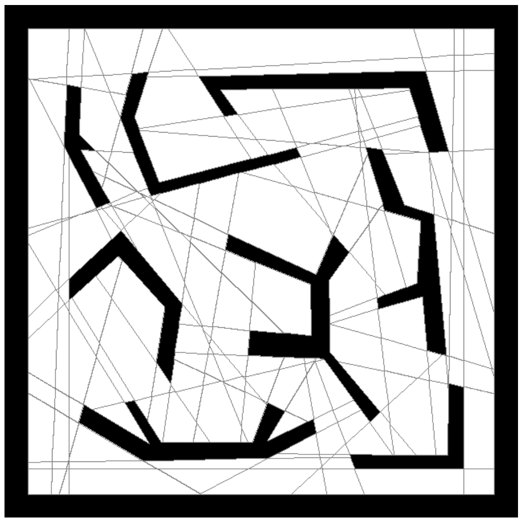
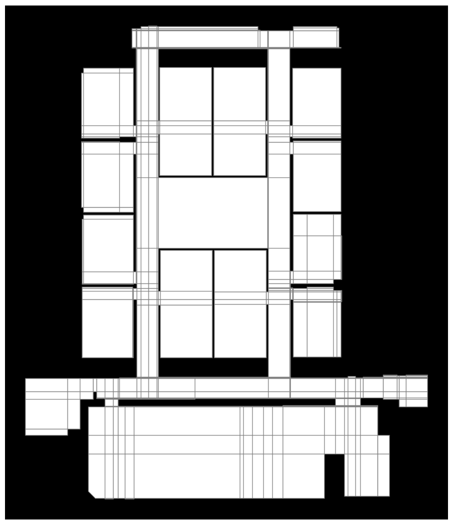

(g) *Cityblocks w/o bitangents*

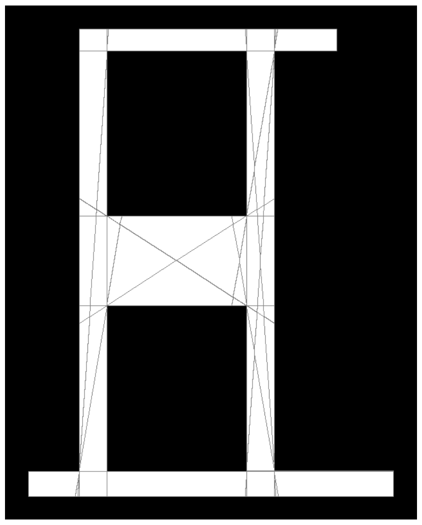
Figure 40: *(continued)* The decompositions of sample environments used for testing. Versions with and without bitangents included are shown, but some of the environments were too complex to include bitangents. The stats of the environments are summarized in Table 1.



(h) *Maze w/o bitangents*



(i) *L457 w/o bitangents*



(j) *L457mod w/ bitangents*

Figure 40: *(continued)* The decompositions of sample environments used for testing. Versions with and without bitangents included are shown, but some of the environments were too complex to include bitangents. The stats of the environments are summarized in Table 1.

Map	V	r	Lb	Lr	La	I	**N**	Td
room1 (*w/o bit.*)	28	12	28	20	0	49	**26**	0.000622
room1 (*w/ bit.*)	28	12	28	20	45	205	**197**	0.003
room2 (*w/o bit.*)	34	17	34	28	0	76	**48**	0.000946
room2 (*w/ bit.*)	34	17	34	28	54	243	**233**	0.00379
room3 (*w/o bit.*)	64	36	64	56	0	148	**98**	0.00201
room3 (*w/ bit.*)	64	36	64	56	126	645	**638**	0.0121
cityblocks (*w/o bit.*)	68	64	68	128	0	600	**517**	0.00655
maze (*w/o bit.*)	82	55	82	110	0	348	**259**	0.00461
l457 (*w/o bit.*)	225	116	225	202	0	791	**594**	0.0144
l457mod (*w/ bit.*)	18	11	18	18	24	72	**65**	0.00141

Table 1: The characteristics of various test environments and their resulting decomposition data. Many of the maps are decomposed both with and without bitangent (aspect) lines. The data presented are: *V*, the number of vertices in the environment; *r*, the number of reflex (convex) vertices; *Lb*, the number of line segments contributed by the environment boundary; *Lr*, the number of line segments contributed by reflex lines; *La*, the number of line segments contributed by aspect (bitangent) lines; *I*, the number of line segment intersections found; *N*, the final number of polygons created; *Td*, the time to compute the decomposition, in seconds.

to order their incoming segments. Using a data structure such as a binary tree, the ordered insertion happens with $O(log(I))$ as each is created, and when intersections overlap (as is the case when more than two segments join at a point) then the segment lists are merged together. The search for existing intersections at the same point and the tree insertion can be performed at the same time.

3. Progress through the ordered list of intersections, and for each adjacent pair of segments coming into or leaving the intersection, classify them as being the start, end, or continuation of a polygon based on their relative orientation. Add polygon starts to the list of open polygons, otherwise search through the list of open polygons to find those with dangling edges corresponding to the segments in question and apply either edge transitions or close the polygon and move it to a closed list. Iterating through segments at each intersection is an $O(1)$ procedure, there are I intersections to loop through, and when searching the open polygon list, it will only have as many entries as there are polygons intersecting a constant-x sweep line through the environment. This count could be as high as N, the number of polygons, but will be much less in practice. The total complexity should therefore be around $O(I)$.

Steps 1 and 2 are simultaneous, and 3 happens sequentially, so the total decomposition complexity is around $O(L^2 log(I) + I)$. I will generally be much less than $L(L-1)/2$ (since the actual valid intersections are a subset of all segment pair intersection checks), so the total complexity will be dominated by the $L(L-1)/2$ intersection checks, for an approximate $O(L^2)$ complexity. This is seen in Figure 41, where the computation time is plotted log-log against the line segment count and a line of slope 2 is fit to the data. There are algorithms for computing this arrangement more efficiently: for example, [13] is an $O(n \log n + k)$ algorithm for computing all k intersections between n line segments in the plane and finding the corresponding faces. The complexity improvement is due to more efficient line segment intersection.

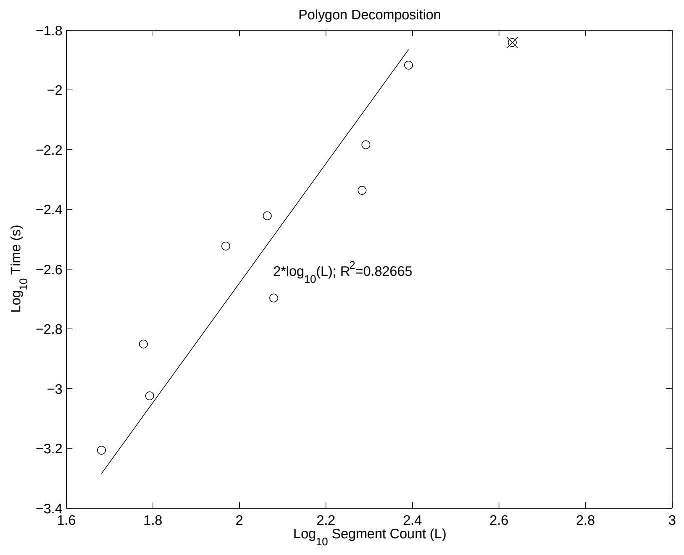


Figure 41: Polygonal decomposition time versus line segment count, plotted log-log. A line of slope 2 is fit to the data, demonstrating a quadratic time complexity. The far right point was removed as an outlier.

Further calculation time information is shown in Table 2. For each decomposition, we calculate the mutual visibility graph, the adjacency graph, and all-points shortest paths for each of these graphs.

Our computation of the MVG proceeds as follows:

1. Loop through each pair of polygons: $N(N-1)/2$ tests

2. For each pair, check if their centroids are visible to each other and eliminate the pairing if not: up to $O(V)$ checks using the naive visibility approach of checking for intersection with each boundary segment

3. Create a convex hull around the two polygons, and find the two line segments added that did not belong to the original polygons: $O(1)$ (related to number of polygon vertices which will be around 8 total)

4. Check if either of these line segments intersect the boundary: $O(V)$ checks using the naive approach

Map	N	Tmv	Tad	Tpv	Tpp
room1 (*w/o bit.*)	26	0.0292	0.00311	0.00323	0.00233
room1 (*w/ bit.*)	197	0.277	0.076	0.84	0.342
room2 (*w/o bit.*)	48	0.0538	0.00545	0.00973	0.0115
room2 (*w/ bit.*)	233	0.405	0.111	1.34	0.532
room3 (*w/o bit.*)	98	0.16	0.0223	0.0614	0.0457
room3 (*w/ bit.*)	638	2.8	0.823	22.8	12.6
cityblocks (*w/o bit.*)	517	4.48	0.634	13	6.4
maze (*w/o bit.*)	259	0.749	0.161	1.06	0.751
l457 (*w/o bit.*)	594	5.81	0.807	15	9.75
l457mod (*w/ bit.*)	65	0.038	0.00778	0.0262	0.0155

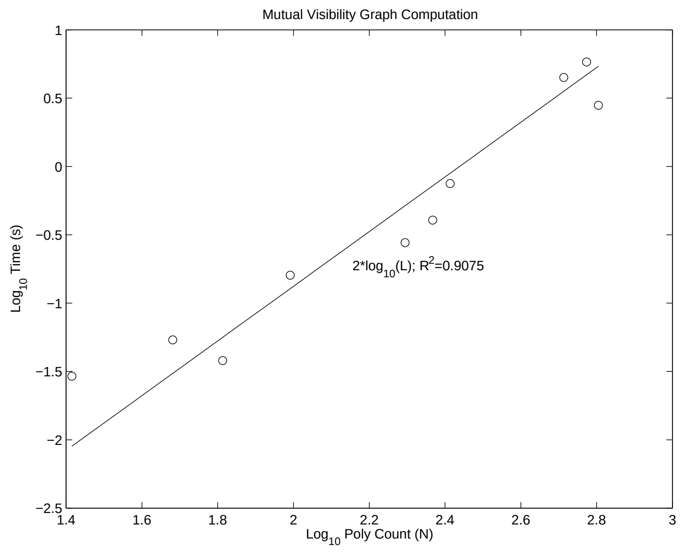
Table 2: Calculation time information for each of the test environments. For each environment, we show the time to compute the mutual visibility graph (*Tmv*), the time to compute the adjacency graph (*Tad*), the time to compute the all-pairs shortest paths for the mutual visibility graph (*Tpv*), and the time to compute the all-pairs shortest paths for the adjacency graph (*Tpp*).

5. Check each environment hole to see if it is within the convex hull of the polygon pair: $O(1)$

The total complexity is $O(VN^2)$, but will be dominated by N^2 since the number of polygons is generally much higher than the vertex count when using bitangent lines.

Building the adjacency graph is done by checking each polygon pair and seeing how many vertices of the first are found within the second (lying on the boundary counts as being contained within). If the number is two or more, then they are adjacent. The total number of checks is $N(N-1)/2$, or $O(N^2)$ complexity.

The time complexity of both graph computations is shown in Figure 42. The computation time is plotted log-log against the polygon count. A line of slope 2 is fit in both cases.



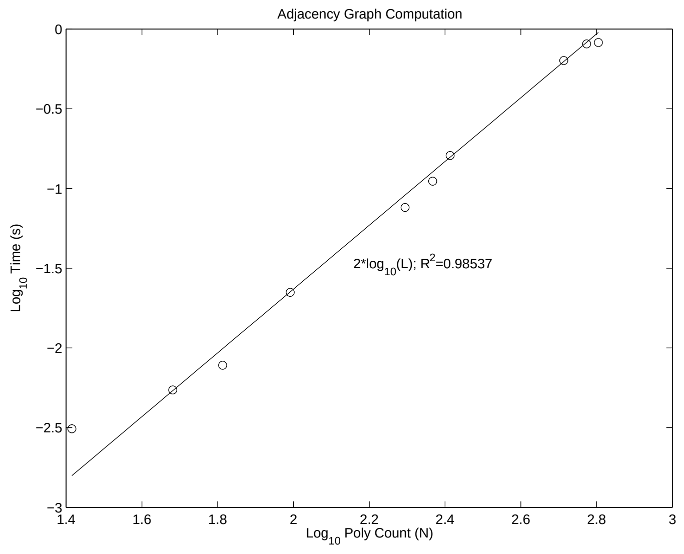


Figure 42: Time complexity analysis of the computation of the mutual visibility and adjacency graphs. The computation time is plotted log-log against the polygon count and lines of slope 2 are fit to each, demonstrating the quadratic complexity of the task.

We compute the all-pairs shortest-paths by repeatedly applying Dijkstra's algorithm to successive nodes. For a graph with $|V|$ nodes and $|E|$ edges, the complexity of Dijkstra's algorithm is known to be $O(|V|^2+|E|)$ when a simple linked list is used to store the list of nodes to be processed. For the worst-case of a complete graph, $|E| = |V|(|V|-1)/2$, so the complexity will be dominated by $|V|^2$. Our polygons are the nodes, so our Dijkstra complexity will be $O(N^2)$. This is repeated for each node, for a total shortest-path complexity of $O(N^3)$. The adjacency graph is less dense than the MVG, so the running time ends up being less, but both scale the same. Both computations are investigated in Figure 43 where the computation times are plotted log-log against the polygon count and lines of slope 3 are fit to both, demonstrating the cubic dependence of the all-pairs shortest path computation on polygon count.

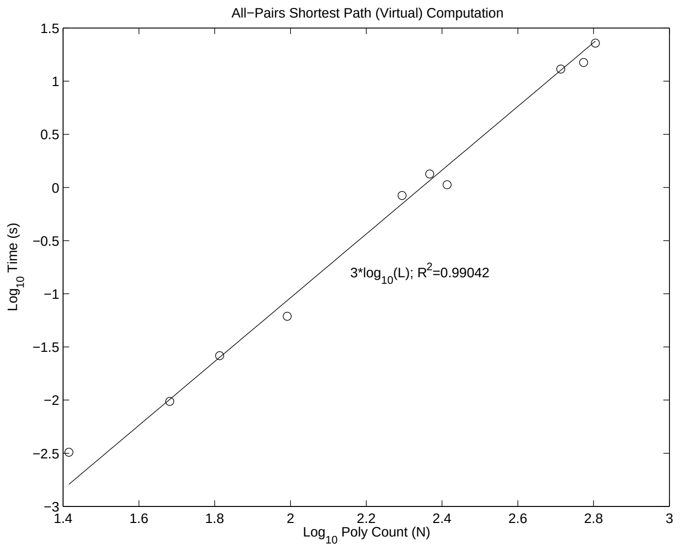
4.1.2 Deployment Complexity

The previous section was discussing the complexity of computations that are preformed only once. However, during operation, calculating Steiner trees and redeployments will be an ongoing task, and so we turn now to consideration of their complexities.

Optimal Steiner Tree

The cost of computing the Steiner tree for the various map decompositions and target counts is shown in Figures 44 and 45. The computation was done repeatedly and the average is indicated along with bars corresponding to the min and max values. From [80] (Theorem 5.7), we have that the time complexity of the Dreyfus-Wagner algorithm, with the shortest paths precomputed, is $O(3^k n+2^k n^2)$. We are interested in verifying the dependence on both variables and so analyze the time dependence by fixing one and varying the other. We will form regressions for the data in two ways: we will fit curves to the natural values, and we will fit lines of fixed slope to the data plotted log-log or linear-log.

In Figure 44, the computation time is plotted with respect to polygon count for



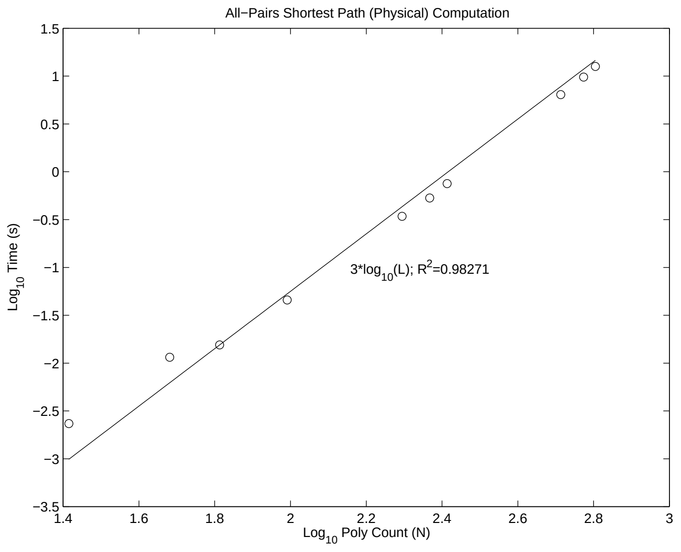
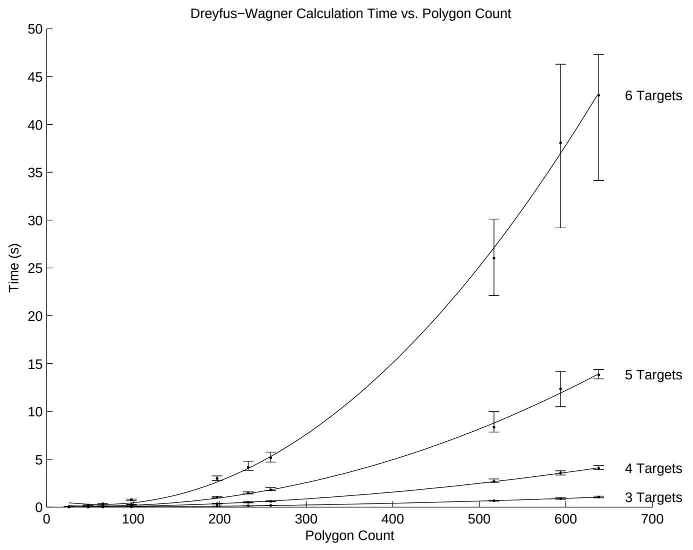


Figure 43: Time complexity analysis of the computation of the all-pairs shortest path for both the MVG and the adjacency graph. The computation time is plotted log-log against the polygon count and lines of slope 3 are fit to both, demonstrating their cubic dependence on polygon count.



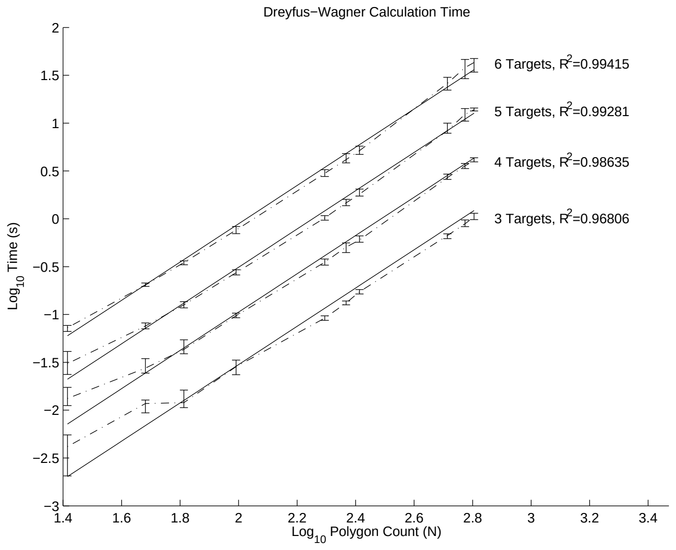
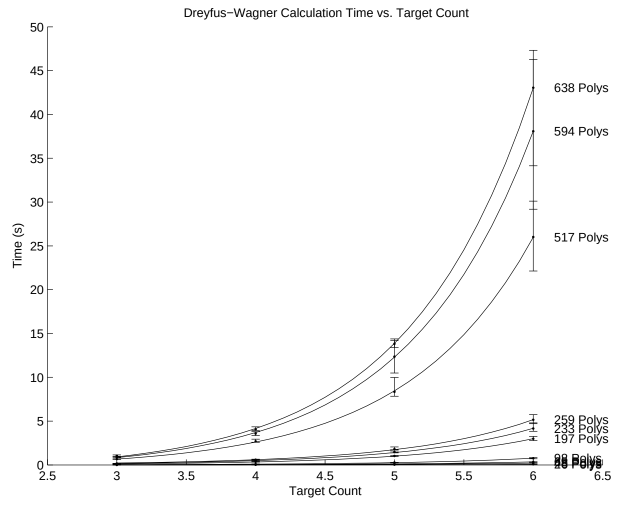


Figure 44: Timing for computation of the optimal Steiner tree using the Dreyfus-Wagner algorithm, looking at the computation time as a function of polygon count for a variety of fixed target counts. The average of several trials is shown along with error bars indicating the minimum and maximum values. In the first plot, the data is plotted normally and a quadratic is fit to each data set. In the second plot, the data is plotted log-log and a line of slope 2 is fit to each data set fairly well, demonstrating a quadratic dependence of computation time on polygon count.



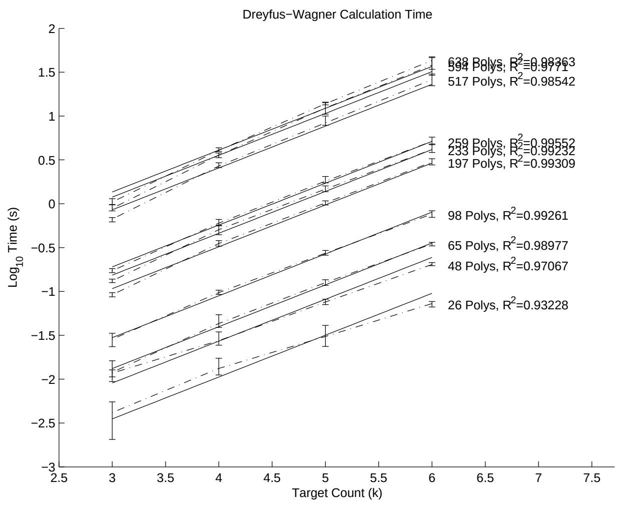


Figure 45: Timing for computation of the optimal Steiner tree using the Dreyfus-Wagner algorithm, looking at the computation time as a function of target count for a variety of fixed polygon counts. The average of several trials is shown along with error bars indicating the minimum and maximum values. In the first plot, the data is plotted normally and a base-3 exponential is fit to each data set. In the second plot, the data is plotted with the time transformed by $\log_{10}$ and a line of slope $\log_{10}(3)$ is fit to each data, demonstrating a base-3 exponential dependence of computation time on target count.

a variety of target counts. In the first plot, a quadratic is fit to the data; in the second, the data is plotted log-log and a straight line of slope 2 is fit over each data set, corresponding to a quadratic relationship between polygon count and runtime. According to the R^2 values, the fits in the log-log case are fairly good.

In Figure 45, the computation time is plotted with respect to target count for a variety of polygon counts. The first plot has a base-3 exponential fit to the data; the second plot is linear-log, with the time transformed by $\log_{10}$, and a straight line of slope $\log_{10}(3)$ is fit over each data set to demonstrate the base-3 exponential dependence. The fit is also very close.

Redeployment

The redeployment happens in essentially two steps:

1. For each tree topology, the dynamic programming step is performed: for each node, the conditionally optimal cost is found by looping through each of N possible placements of the node and minimizing over the N possible positions of each neighboring node connected by an edge in the topology. For k nodes in the topology, there are $k-1$ edges in total, so the complexity for a single dynamic programming step is $O(kN^2)$.

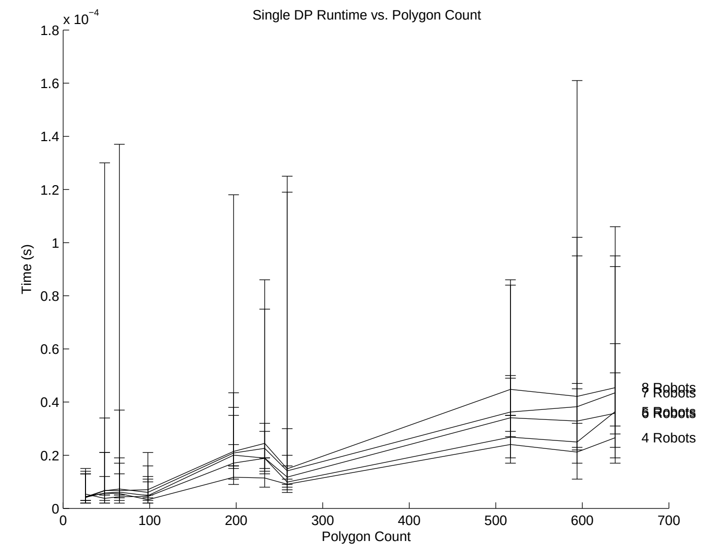
2. Single-step topology modifications are performed by removing each edge of the topology in turn ($k-1$ edges), creating two disjoint trees, and then joining these trees by connecting each possible pair of vertices. For a tree split into two trees of size $|T_a|$ and $|T_b|$, there are $|T_a||T_b|-1$ possible other trees to be formed.

Assigning a complexity to the topology iteration portion of this process is challenging. The number of trees to check is smallest when removing the link for a leaf node, and the line topology has the smallest number of leaves and so has the most topologies to check. If the topology were a star, the number of topologies to check for each link removal grows linearly, and hence the total complexity is quadratic. For a line topology, some of the links lead to linear complexity, and some lead to almost quadratic, so the complexity is more than quadratic but less than cubic. The complexity of the topology checking falls somewhere between the two and is dependent on the current topology. It should be noted that if all possible topologies are checked, rather than just single-edge modifications, then there are a total of n^{n-2} trees possible over n vertices [12].

For the single run of the dynamic programming, we expect $O(kN^2)$ complexity, and some timing results are shown in Figures 46 and 47. In Figure 46, the timing with respect to polygon count is plotted for fixed robot count in both normal and log-log plots. In Figure 47, the timing with respect to robot count is plotted for fixed polygon count in both normal and log-log plots. It is not possible to find a regression in this data set because there is too much variance in the timing values. Increasing the accuracy of these time values remains future work.

The total redeployment task of computing the dynamic program for iterated topologies should have complexity greater than $O(k^3N^2)$ but less than $O(k^4N^2)$. Figures 48, 49, and 50 show the gathered timing information, again plotted both normal and log-log and with timing compared against polygon count and robot count. What is interesting is that a cubic relationship was expected for robot count, but in reality the dependence fits better to a quartic for lower polygon counts and to the 5th power for higher ones, possibly increasing beyond that.

One possible reason for this discrepancy is implementation-dependent. In the overview of the redeployment calculation, we skipped the step where a computation of the conditionally optimal cost is backtracked in order to solve for the actual positions. This is analogous to reinforcement learning value function techniques where a dynamic program must be solved to extract the optimal policy after computation. We could do something akin to computing with the state-action function instead: namely, keep track of conditionally optimal costs as well as the position choices that



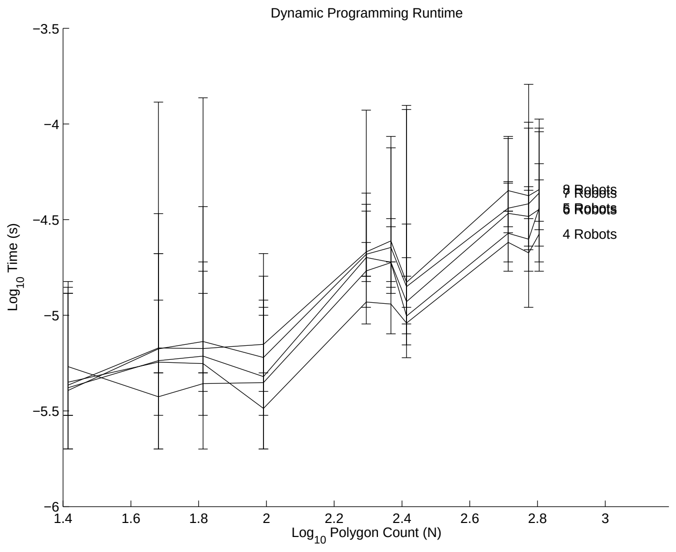
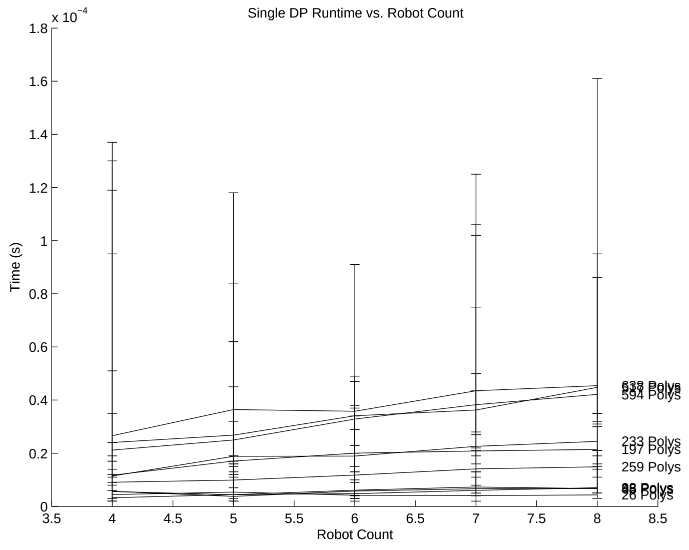


Figure 46: Timing of a single run of the dynamic programming task, plotted against the polygon count for fixed robot count. The data is plotted both normally and log-log, but the variance is too high to make use of because of timing errors due to resolution and scheduling. The relationship is expected to be quadratic.



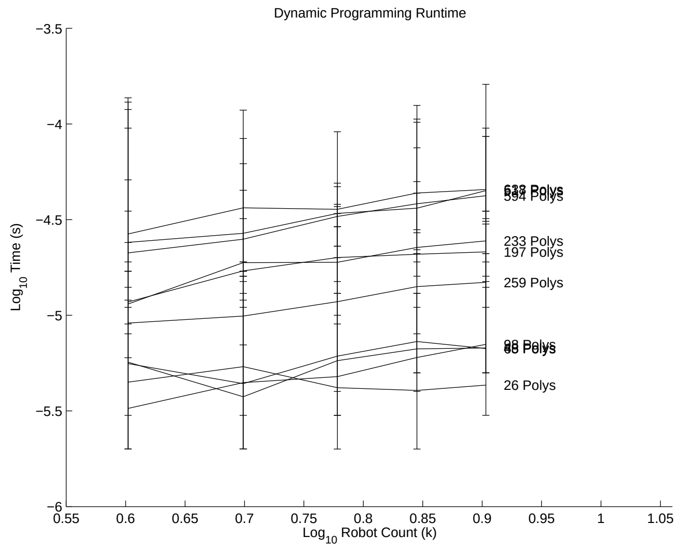


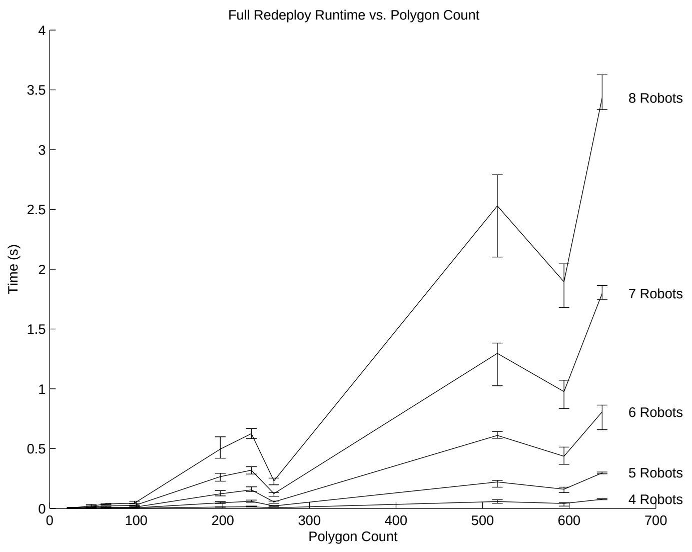
Figure 47: Timing of a single run of the dynamic programming task, plotted against the robot count for fixed polygon count. The data is plotted both normally and log-log, but the variance is too high to make use of because of timing errors due to resolution and scheduling. The relationship is expected to be linear.

Algorithm Step	Complexity
Polygon Decomposition	$O(L^2)$
Mutual Visibility Graph	$O(N^2)$
Adjacency Graph	$O(N^2)$
All-Pairs Shortest Paths	$O(N^3)$
Steiner Tree	$O(3^k N + 2^k N^2)$
Redeployment (Single DP Step)	$O(kN^2)$ *(theoretical)*
Redeployment (Full)	$O(k^4 N^2) \sim O(k^5 N^2)$

Table 3: Summary of the complexity of each step in the algorithms presented. Here L is the number of line segments used for performing the polygon decomposition, N is the number of polygons found in the decomposition, and k is either the number of targets in the Steiner Tree computation or the number of robots in the redeployment calculation. The complexities were verified experimentally except for the Single Dynamic Programming Step of the Redeployment since the timing variance was too great to make a conclusion.

led to them, eliminating the need for backtracking at the expense of memory, but this was not done here. In addition, our implementation performs this backtracking to solve for optimal positions every time a topology with lower cost is found. In the worst case, every new topology checked could be better and so the backtracking is performed for every topology. The scaling would suggest that since this backtracking is approximately $O(kN^2)$ and is performed together with a forward step, the scaling should be the same but with a factor of 2 in the worst case; however, since $N^2 >> k^3$ for the number of robots we are dealing with, the results will be inflated by any extra uses of the quickly-scaling dynamic programming portion of the computation. Deferring backtracking until the computation is complete should eliminate this bias.

All of the scaling is summarized in Table 3.



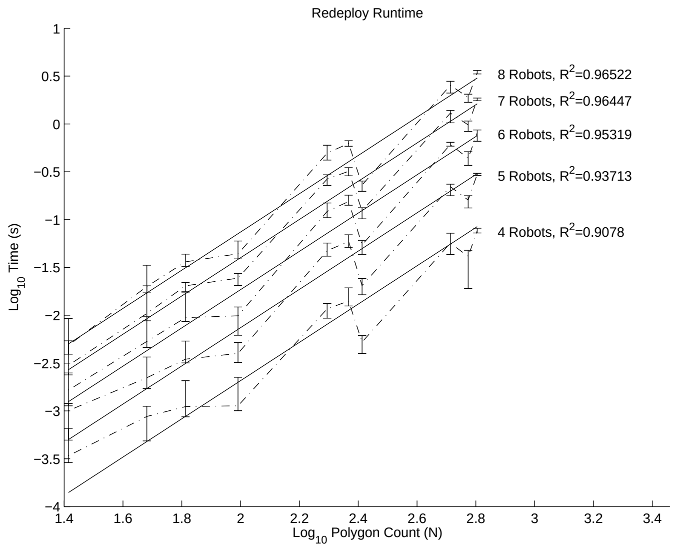


Figure 48: Timing of the total redeployment computation, plotted against the polygon count for fixed robot count. The data is plotted both normally and log-log. The relationship is expected to be quadratic, and in the second plot, a line of slope 2 is fit to the data fairly well, suggesting this is the case.

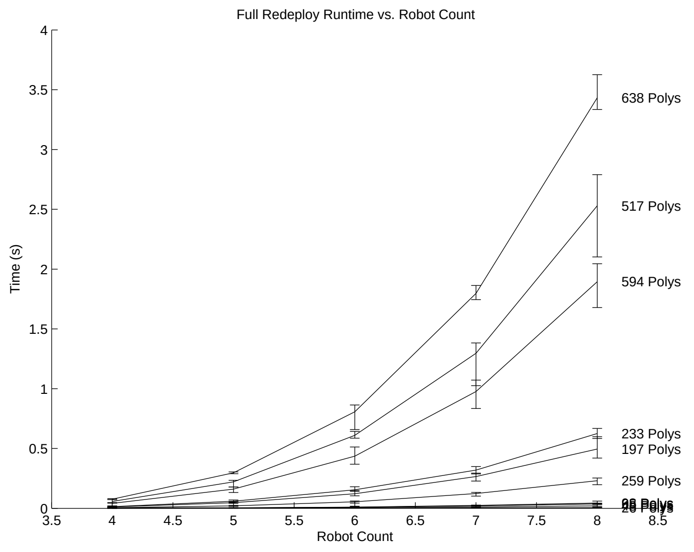


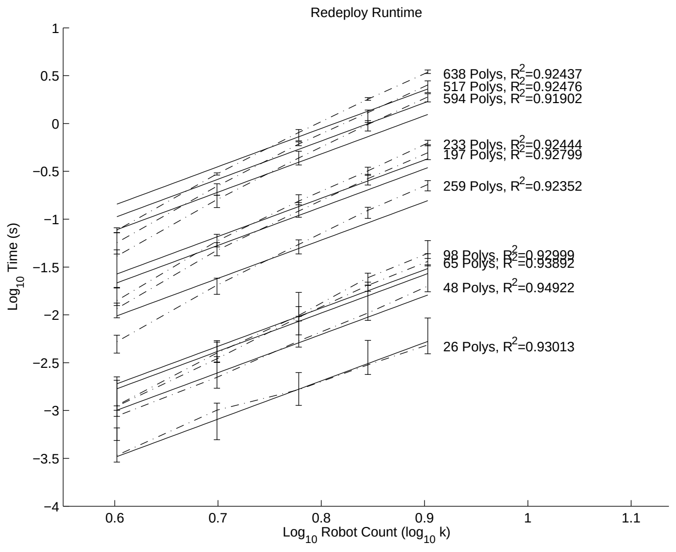
Figure 49: Timing of the total redeployment computation, plotted against the robot count for fixed polygon count. The data is plotted normally; the expected complexity is greater than cubic but less than quartic. There does seem to be a relationship, but it turns out to be neither of these, as seen in Figure 50.

4.2 Single-Agent Path Planning

Before we can deal with combining the paths of multiple robots to avoid collisions and maintain visibility, we must first understand how to plan a path for a single robot. We will do this in two ways: first using geodesics within the polygonal environment, and then using cell-sequences in the adjacency graph of the environment and its associated visibility cell decomposition.

4.2.1 Computation of Geodesics

Given the start and goal positions for an individual agent, we compute the desired trajectory by finding the geodesic connecting the two positions within the polygonal environment [55]. In general, the geodesic is defined as the shortest path between two points. In a Euclidean space, geodesics are straight lines, but in curved spaces



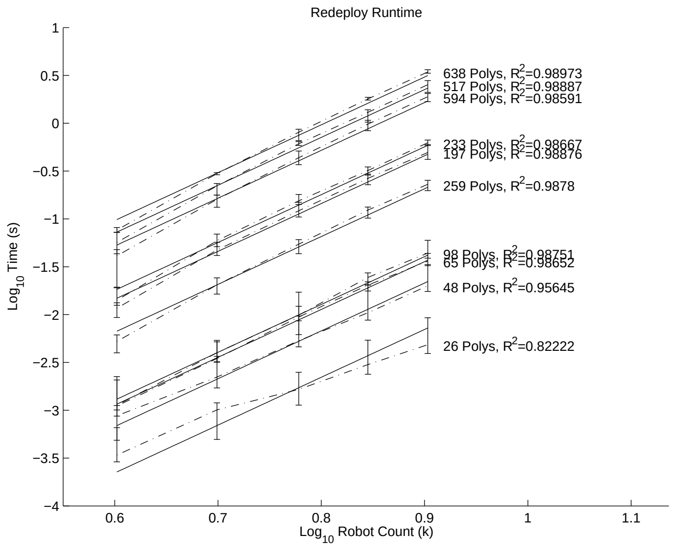


Figure 50: Timing of the total redeployment computation, plotted against the robot count for fixed polygon count. The data is plotted log-log and the plots shown different slope lines fit to the data, slope 4 in the first and slope 5 in the second. The lower polygon count cases fit the slope 4 line better, but as the polygon count rises, the slope 5 line fits better, suggesting a deeper relationship between the polygon count and robot count in determining the algorithm complexity.

or spaces with holes, they become more exotic. On the surface of a sphere, for example, geodesics consist of the great circle arcs formed from taking a section of a diametric circle passing through the two points in question. Geodesics are not necessarily unique; if the points on the sphere are antipodal, such as the north and south poles on a spherical model of the Earth, then there are an infinite spectrum of geodesics between the two corresponding to all of the lines of longitude.

Given a two-dimensional robot, the question of ensuring that the robot does not run into the walls is handled by the common trick of inflating the walls by an amount equal to the bounding radius of the robot shape. That way, the robot can be treated as a point traveling in the expanded environment.

The geodesic query happens frequently, so an efficient routine for calculating it online is used. Most of the computation is handled offline and only some simple visibility computation and value lookup needs to be done at runtime. The offline steps consist of computing the Visibility Graph and then constructing all shortest paths between reflex vertices. The online steps consist of finding the visible reflex vertices at the start and goal locations, then looping over pairs and looking up their shortest path lengths and constructing the shortest path using the pre-calculated path information. All of these steps are now explained in detail.

Preliminaries The polygonal environment is represented as an indexed list of vertex positions and a corresponding list of the two edges connected to each vertex. Each edge is directed; this allows us to represented polygons with holes since we orient each edge with free space on its left side. Computationally speaking, given a polygon vertex p and the edge vector $\boldsymbol{v}$ emanating from it, a point q is in the interior of the polygonal environment, with respect to this edge, if $\boldsymbol{v} \times (q - p) > 0$. Because the free space of our environment is interior, the boundary edge of the environment is directed counter-clockwise, while the boundary of any interior holes are directed clockwise.

This information is maintained in three indexed lists: the Vertices list maintains an indexed set of coordinates of each vertex in $\mathcal{R}^2$, the Next list associates with each vertex index the index of the vertex succeeding it along the polygon edge, and the Prev list associates with each vertex index of the vertex preceding it. The forward edge vector from vertex i would be calculated as $\textsc{Vertices}(\textsc{Next}(i)) - \textsc{Vertices}(i)$. Keeping the list of both the next and previous vertices allows for easy construction of the edges local to a given vertex at various places in the algorithms.

Visibility Graph Computation The Visibility Graph for the given polygonal environment is a graph where each vertex of the polygon forms a node and an edge is joined between two nodes if their corresponding vertices are visible. In this case, visibility exists if the line segment joining the two vertices does not intersect any other line segment in the environment. Though there are more efficient ways to calculating the graph, this simple description already gives rise to the naive algorithm that we use.

The naive algorithm is shown in Algorithm 8. There are various tricks that can speed up the computation, but it remains an $O(n^3)$ algorithm. Other implementations have been found that run faster: Lee's algorithm runs in $O(n^2 \log n)$ time [58], and the Overmars and Weizl algorithm runs in $O(n^2)$ time [75]. However, since the visibility graph computation can be performed *a priori*, we have no need for a faster implementation.

With the resulting graph, we can take any vertex and quickly determine which other vertices are visible to it, speeding up the calculation of shortest paths to follow.

Vertex Shortest Path Computation With the visibility graph in place, we can easily calculate the shortest path between any two vertices in the polygonal environment. By calculating all such shortest paths beforehand, the bulk of the geodesic calculation is reduced to a table lookup. Only a subset of the vertices need to be considered for calculation; *reflex* vertices are the only ones who will be found in

Algorithm 8 NaiveVisibilityGraph(P)

Require: A polygonal environment P with vertices V and joining edges E.

Ensure: A visibility graph G joining the corners of P that are visible.

```
1: for $v \in V$ do
2:     for $u \in V,\ u \neq v$ do
3:         $r \leftarrow u - v$
4:         for $e \in E$ do
5:             if $r$ intersects $e$ then
6:                 Break and check next $u$
7:             end if
8:         end for
9:         Add $(u, v)$ to $G$
10:    end for
11: end for
12: return $P$
```

any shortest path. Starting with each reflex vertex as a source, the shortest paths to all other vertices are found using the Dijkstra algorithm and the distance information and node chains are stored for later use.

Finding the reflex vertices is as simple as checking the cross product of the adjacent edges. The ith vertex is a reflex vertex if:

$$(\textsc{Vertices}(\textsc{Next}(i)) - \textsc{Vertices}(i)) \times (\textsc{Vertices}(\textsc{Prev}(i)) - \textsc{Vertices}(i)) < 0 \tag{48}$$

Geodesic Query With the shortest paths between reflex vertices pre-computed, performing a geodesic query requires simply finding all reflex vertices visible from the start and end nodes and then selecting the pair that lead to the smallest total length. Once this pair is selected, the information about the node chains from the shortest path computation is used to reconstruct the path.

4.2.2 Computation of Adjacency Paths

Given the robot's current polygonal cell and its target cell, we would like to find a sequence of adjacent cells that the robot can traverse to get between the two. This is accomplished by finding shortest paths in the adjacency graph for the polygonal decomposition.

The adjacency graph has the individual polygonal cells as nodes and an edge between two nodes if they share a common edge. Thus travel is possible between connected cells by passing through this edge. For the decompositions shown in Figs. 20 and 21, the corresponding adjacency graphs are shown in Fig. 51. Fig. 52 shows the adjacency graph for a more complicated environment.

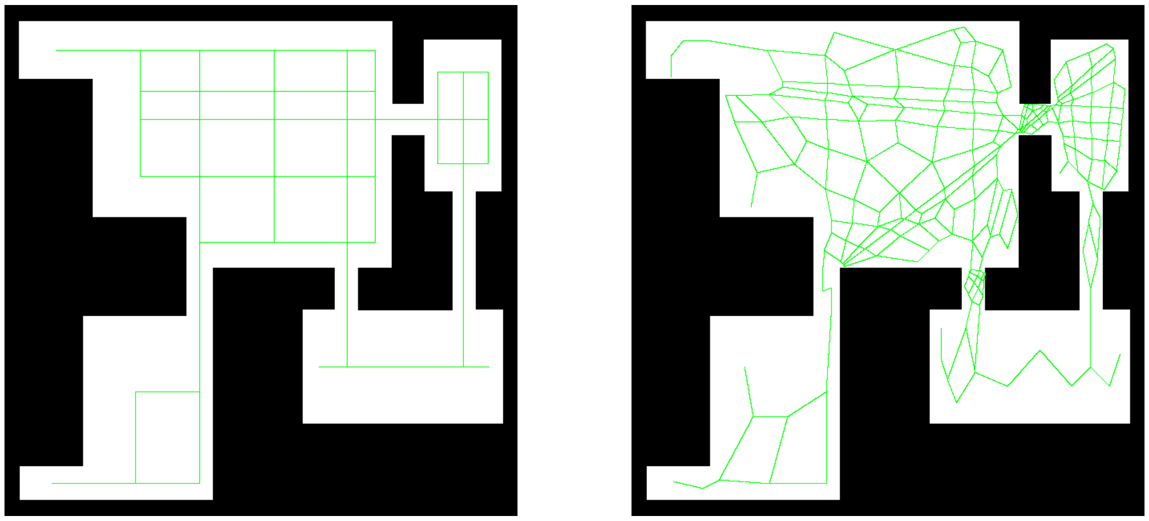


Figure 51: The adjacency graphs for the decompositions of Figs. 20 and 21. Two cells are connected if they share a common boundary edge.

The shortest path is computed with edge lengths being taken as the distance between the centroids of the adjacent cells. If a smoother path is desired, a small penalty can be added to each edge length to encourage paths to use less jumps. Any

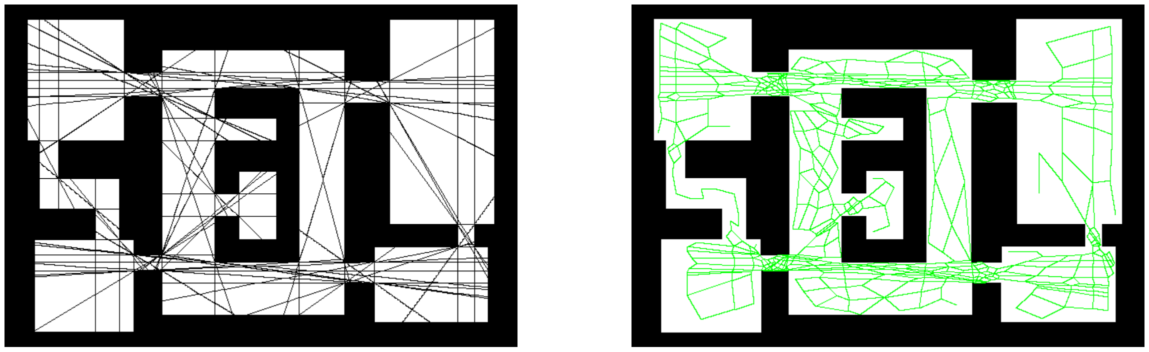


Figure 52: The aspect decomposition of a complicated environment and its corresponding adjacency graph.

standard shortest path algorithm can be used, in our work we have pre-computed all shortest paths offline using the Dijkstra algorithm.

Examples of adjacency paths computed in this way are shown in Fig. 53 for the aspect decompositions whose adjacency graphs are seen in the previous Figs. 51 and 52.

In contrast to geodesics, whose path is represented as a series of line segments joining the start and end points, adjacency paths are a sequence of cells that can be traversed, in order, to travel from the start cell to the end cell.

4.3 Multi-Agent Path Planning

Given a group of agents with assigned target positions, the next challenge is to plan paths for the agents so that they reach their destinations while avoiding collisions with each other.

There are three established methodologies for coordinating multiple agents in such a way. One is to develop controllers based on artificial potential fields for each individual agent and then join them together with an inter-agent repulsive potential in order to simultaneously move towards the objective while avoiding collisions. This has the advantage of being simple to implement, but offers no guarantees that the

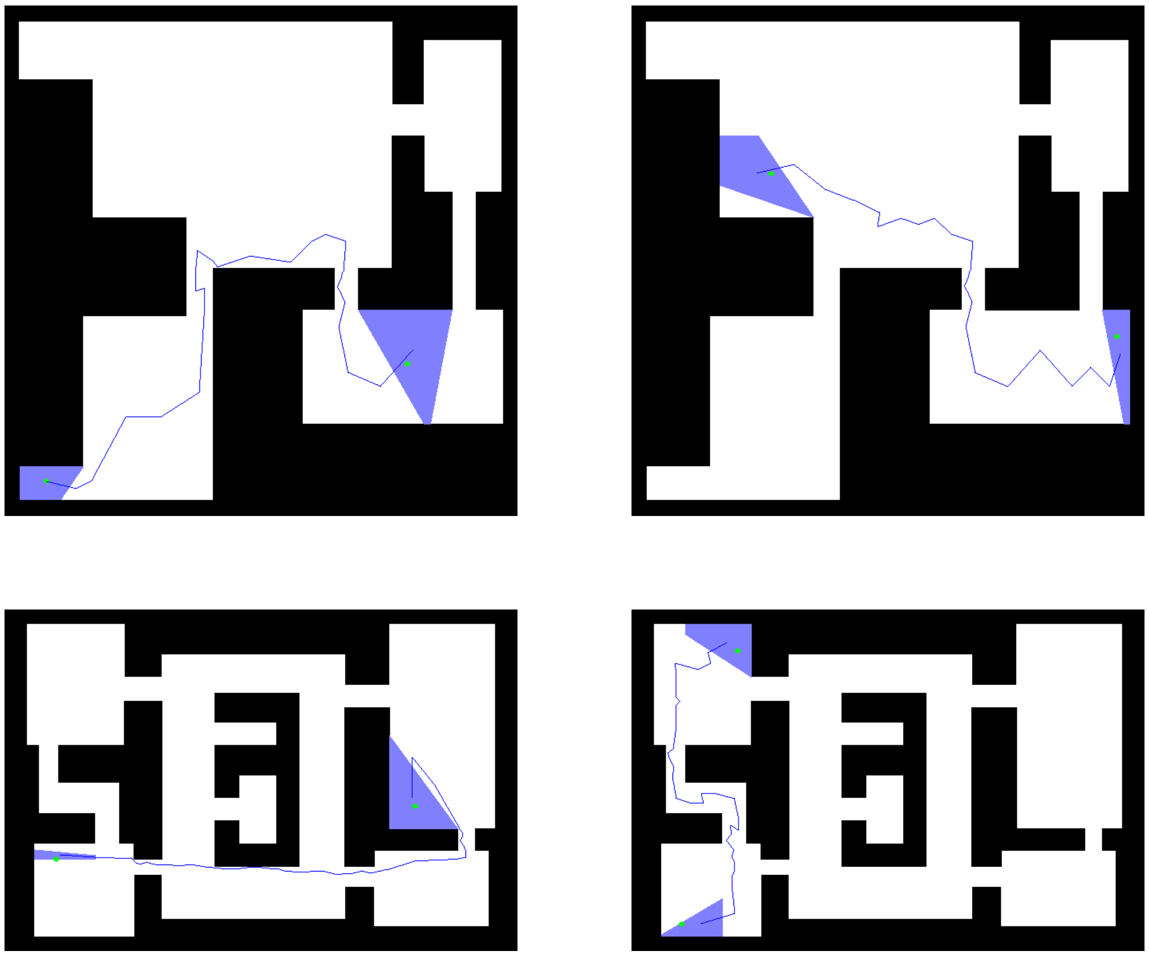


Figure 53: Example adjacency paths for the aspect decompositions of the environment as seen in Figs. 51 and 52.

agents will actually reach their goals without getting stuck in local minima of the aggregate potential field. A second methodology is to solve a joint planning problem taking into account the states of all agents and enforcing non-collision constraints. However, the complexity of this planning problem grows quickly as the number of agents grows.

Our solution for accomplishing this is taken from [89, 57]. It could be seen as a hybridization of the above two methods: each agent plans its respective solution path, but then we *schedule* the collective execution of these plans in such a way as to avoid collisions. There is a reduction in complexity as compared to global planning because the full state of each agent is reduced to a single variable: the amount of its path that has been executed.

4.3.1 Combining Two Geodesic Paths

Given the kth agent's start point, $\boldsymbol{x}_k^s$, and end point, $\boldsymbol{x}_k^g$, the single-agent solution path is simply the geodesic from $\boldsymbol{x}_k^s$ to $\boldsymbol{x}_k^g$ within the expanded polygonal environment. For the kth agent, this path is a function, $P_k : [0,1] \to \mathcal{R}^2$, where the path variable γ indicates how far along the path we are. $P_k(0) = \boldsymbol{x}_k^s$ and $P_k(1) = \boldsymbol{x}_k^g$ and $P_k(\gamma)$ gives points in-between.

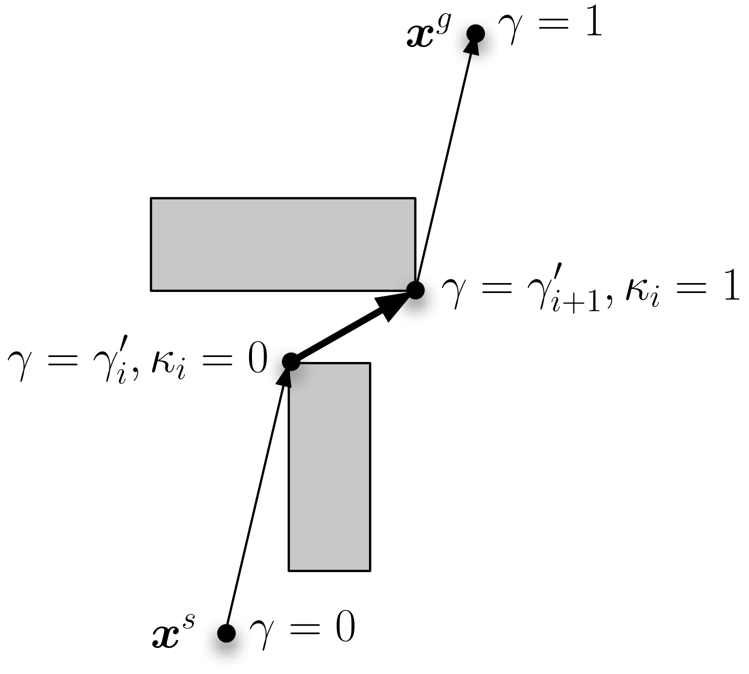

$\boldsymbol{x}^g$
$\gamma = 1$
$\gamma = \gamma'_{i+1}, \kappa_i = 1$
$\gamma = \gamma'_i, \kappa_i = 0$
$\boldsymbol{x}^s$
$\gamma = 0$

Figure 54: A highlighted straight-line segment of the geodesic between $\boldsymbol{x}^s$ and $\boldsymbol{x}^g$. The ith segment starts when the path-variable $\gamma = \gamma'_i$ and ends when the path-variable $\gamma = \gamma'_{i+1}$. The local path-variable κ_i ranges from 0 to 1 along this segment.

From our construction of the geodesic in polygonal environments, this path is composed of a sequence of straight-line segments as shown in Fig. 54. If the geodesic for the kth agent is composed of N segments, there is an increasing sequence of path-variable values $\gamma = \{\gamma'_0 = 0, \gamma'_1, \ldots, \gamma'_N, \gamma'_{N+1} = 1\}$ whose evaluations return the positions of the start and end points of the various segments. For example, the ith segment ranges from $P_k(\gamma'_i)$ to $P_k(\gamma'_{i+1})$; we use a local path-variable κ_i to describe this segment. The local path function for this ith segment of the geodesic for the kth agent is just a linear combination of the endpoints: $P^i_k(\kappa_i) = (1 - \kappa_i)P_k(\gamma'_i) + \kappa_i P_k(\gamma'_{i+1})$.

The total path-variable value in the ith segment can be similarly calculated from the local path-variable value as:

$$\gamma = (1 - \kappa_i)\gamma'_i + \kappa_i \gamma'_{i+1} \tag{49}$$

Given two single-agent paths, it is possible that as the robots move along their paths, there will be path-variable combinations that lead to a collision of the two robots. These interactions can be clearly seen in a *coordination diagram* representation of the path pair, as seen in Fig. 55.

Movement in the coordination diagram corresponds to movement along the robot paths. This is illustrated in Fig. 56, where 3 different locations in the coordination diagram are selected and their respective robot positions are demonstrated. Note that movements in the coordination diagram produce complicated motions in physical space. Configuration A located on the edge of the set of collision configurations in the center of the diagram, and so the robots are touching each other. As the configuration moves along one axis in the coordination diagram, to configuration B, robot 1 moves along its path while robot 2 stays still. In configuration C, robot 2 has moved through the collision area because robot 1 is no longer in a coordinate that would block its motion.

The collision areas in the coordination diagram can be more complicated if the motions are complicated, such as in Fig. 57. We only are interested in straight-line segments however, so the coordination diagrams between geodesic paths take on very specific structure.

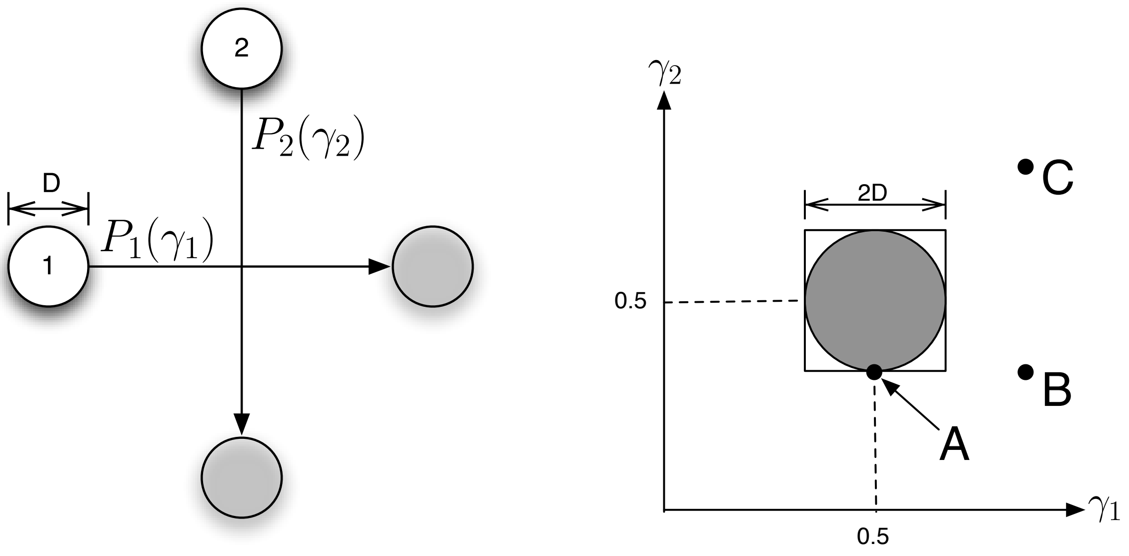


Figure 55: By looking at a pair of paths, values of the path-variables where the robots collide can be seen in a coordination diagram. In this case, the robots collide at the halfway point along each of their paths and at nearby values of γ, due to their size.

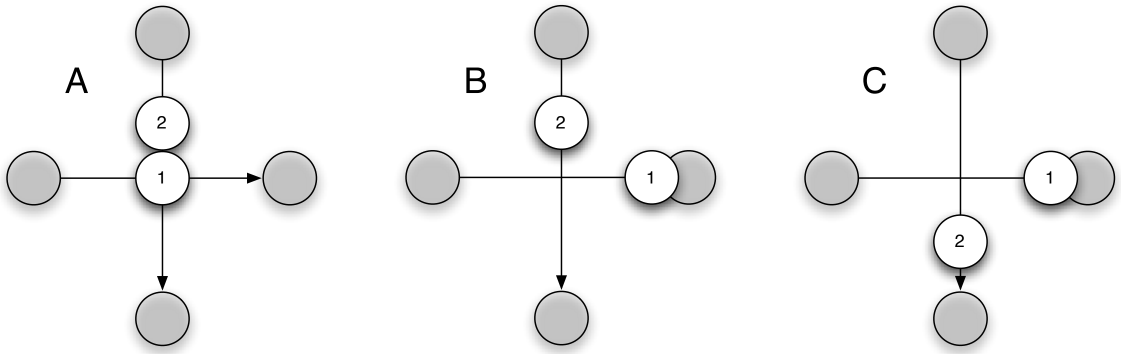


Figure 56: The positions of the robots in physical space, corresponding to three coordinates of interest in the coordination diagram of Fig. 55.

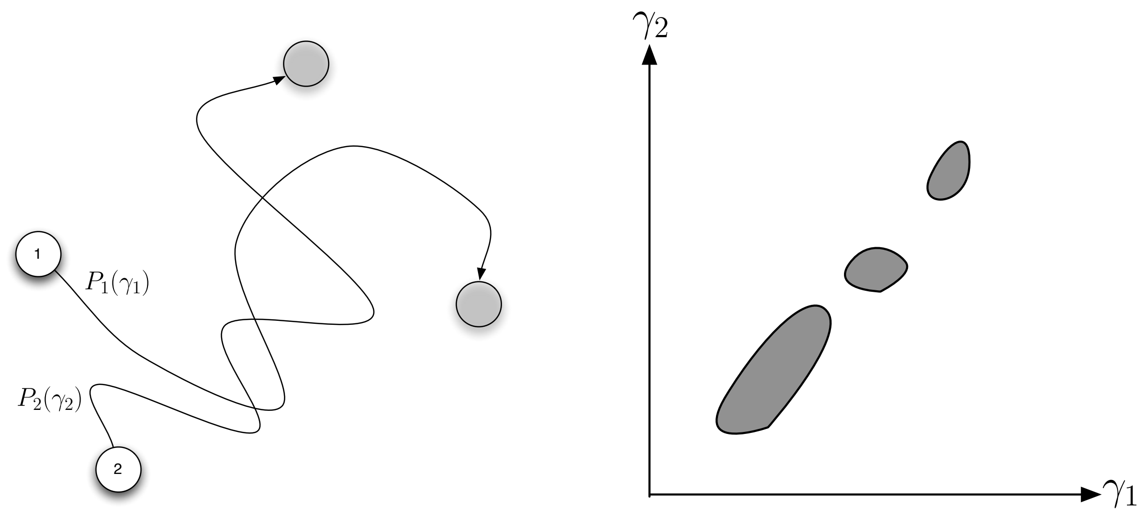


Figure 57: A more complicated pair of paths and the associated coordination diagram. Our geodesic paths will never be this complicated since we are only interested in paths composed of straight-line segments.

For two circular-shaped objects, a collision occurs if the distance between their centers drops below $R_1 + R_2$, where R_1 and R_2 are the radii of the objects. Looking at one pair of segments, one for each path, the collision situation looks like Fig. 58.

If we express the straight-line segments as $P_k^i(\kappa_i) = \boldsymbol{p}_i + \kappa_i \boldsymbol{v}_i$, then the equation for $D^2(\kappa_i, \kappa_j)$ is:

$$\begin{aligned} D^2(\kappa_i,\kappa_j) &= \|(\boldsymbol{p}_i + \kappa_i\boldsymbol{v}_i) - (\boldsymbol{p}_j + \kappa_j\boldsymbol{v}_j)\|^2 \\ &= [(\boldsymbol{p}_i - \boldsymbol{p}_j)^T(\boldsymbol{p}_i - \boldsymbol{p}_j)] + [2(\boldsymbol{p}_i - \boldsymbol{p}_j)^T\boldsymbol{v}_i]\kappa_i + [2(\boldsymbol{p}_i - \boldsymbol{p}_j)^T\boldsymbol{v}_j]\kappa_j \\ &\quad +[-2\boldsymbol{v}_i\boldsymbol{v}_j]\kappa_i\kappa_j + [\boldsymbol{v}_i^T\boldsymbol{v}_i]\kappa_i^2 + [\boldsymbol{v}_j^T\boldsymbol{v}_j]\kappa_j^2 \end{aligned}$$

and we see it as a quadric in κ_i, κ_j. If the radii of the two robots are R_{k_1} and R_{k_2}, then any combination of κ_i and κ_j that leads to $D^2(\kappa_i, \kappa_j) < (R_{k_1} + R_{k_2})^2$ will result in collision. Moving the squared radius term to the left-hand side of the inequality, we want to characterize the region defined by:

$$D^2(\kappa_i, \kappa_j) - (R_{k_1} + R_{k_2})^2 < 0 \tag{50}$$

One of the simplifications proposed by [89] is to calculate the bounding box for

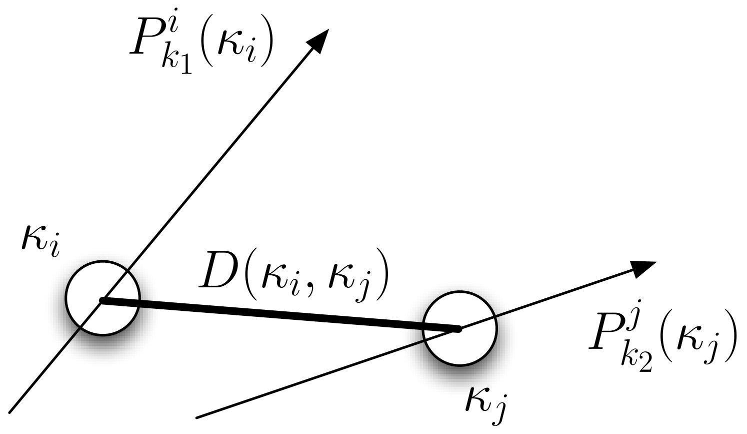


Figure 58: While calculating the interaction between paths k_1 and k_2, we consider just the interaction of the ith straight-line segment of k_1 and the jth straight-line segment of k_2. When the robots are at positions κ_i and κ_j in the local path-variable values of their respective segments, the distance $D(\kappa_i, \kappa_j)$ is shown. If this distance drops below the sum of the bounding radii of the two robots, then a collision occurs.

this region rather than dealing with it as a quadric inequality. We do this below, but with one additional note: only the portion of the inequality region that lies within the $[0, 1] \times [0, 1]$ coordinate box needs to be considered. This necessitates finding absolute bounds for the inequality region and then shrinking them as necessary if it turns out that the portion within our coordinate box does not reach these full extents.

Characterizing the Collision Region The standard form for the quadric inequality in two variables is:

$$A\kappa_i^2 + B\kappa_i\kappa_j + C\kappa_j^2 + D\kappa_i + E\kappa_j + F < 0 \tag{51}$$

where for our problem we have:

$$\begin{aligned}
A &= \boldsymbol{v}_i^T \boldsymbol{v}_i \\
B &= -2\boldsymbol{v}_i^T \boldsymbol{v}_j \\
C &= \boldsymbol{v}_j^T \boldsymbol{v}_j \\
D &= 2(\boldsymbol{p}_i - \boldsymbol{p}_j)^T \boldsymbol{v}_i \\
E &= -2(\boldsymbol{p}_i - \boldsymbol{p}_j)^T \boldsymbol{v}_j \\
F &= (\boldsymbol{p}_i - \boldsymbol{p}_j)^T (\boldsymbol{p}_i - \boldsymbol{p}_j) - (R_{k_1} + R_{k_2})^2
\end{aligned}$$

This can be written in homogenous coordinates as:

$$\begin{bmatrix} \kappa_i \\ \kappa_j \\ 1 \end{bmatrix}^T \begin{bmatrix} A & \frac{B}{2} & \frac{D}{2} \\ \frac{B}{2} & C & \frac{E}{2} \\ \frac{D}{2} & \frac{E}{2} & F \end{bmatrix} \begin{bmatrix} \kappa_i \\ \kappa_j \\ 1 \end{bmatrix} = \tilde{\boldsymbol{\kappa}}^T Q \tilde{\boldsymbol{\kappa}} < 0 \tag{52}$$

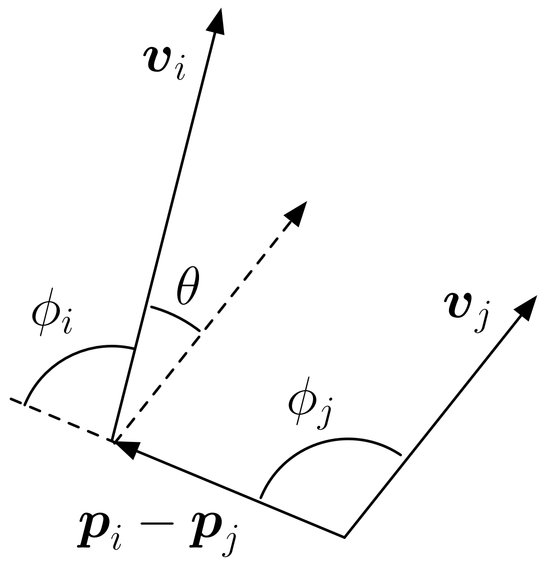


Figure 59: Angles between segments and their connecting vector.

The family of curves represented by quadric equations are known as conic sections, and their characterization is well known. If the determinant $|Q|$ is 0, then the conic is degenerate. We can calculate this quantity explicitly:

$$\begin{aligned}
|Q| &= F\begin{vmatrix} A & \frac{B}{2} \\ \frac{B}{2} & C \end{vmatrix} - \frac{E}{2}\begin{vmatrix} A & \frac{B}{2} \\ \frac{D}{2} & \frac{E}{2} \end{vmatrix} + \frac{D}{2}\begin{vmatrix} \frac{B}{2} & C \\ \frac{D}{2} & \frac{E}{2} \end{vmatrix} \\
&= ACF + \frac{BDE}{4} - \frac{B^2F}{4} - \frac{AE^2}{4} - \frac{CD^2}{4}
\end{aligned}$$

By writing the inner products in terms of the angles between the corresponding vectors, as seen in Fig. 59, we have:

$$\begin{aligned}
A &= \|\boldsymbol{v}_i\|^2 \\
B &= -2\|\boldsymbol{v}_i\|\|\boldsymbol{v}_j\|\cos(\theta) \\
C &= \|\boldsymbol{v}_j\|^2 \\
D &= 2\|\boldsymbol{p}_i - \boldsymbol{p}_j\|\|\boldsymbol{v}_i\|\cos(\phi_i) \\
E &= -2\|\boldsymbol{p}_i - \boldsymbol{p}_j\|\|\boldsymbol{v}_j\|\cos(\phi_j) \\
F &= \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2 - (R_{k_1} + R_{k_2})^2
\end{aligned}$$

and we can simplify the determinant even further:

$$\begin{aligned}
|Q| &= ACF + \frac{BDE}{4} - \frac{B^2F}{4} - \frac{AE^2}{4} - \frac{CD^2}{4} \\
&= \|\boldsymbol{v}_i\|^2\|\boldsymbol{v}_j\|^2\left((\cos^2(\theta) - 1)(R_{k_1} + R_{k_2})^2 \ldots\right. \\
&\quad + \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^2(1 - \cos^2(\theta) - \cos^2(\phi_i) - \cos^2(\phi_j) \ldots \\
&\quad \left. + 2\cos(\theta)\cos(\phi_i)\cos(\phi_j))\right)
\end{aligned}$$

And from Fig. 59 we can see the substitution $\phi_j = \theta + \phi_i$, which allows us to simplify even further to get:

$$|Q| = -\|\boldsymbol{v}_i\|^2\|\boldsymbol{v}_j\|^2(R_{k_1} + R_{k_2})^2\sin^2(\theta)$$

From this we know that the conic will only be degenerate if $\theta = 0$, requiring that the line segments are parallel. In all other cases, the conic is non-degenerate.

Further information is obtained from the sign of the determinant of the primary sub-matrix:

$$|Q_{11}| = \begin{vmatrix} A & \frac{B}{2} \\ \frac{B}{2} & C \end{vmatrix}$$

We can calculate $|Q_{11}|$ to be:

$$\begin{aligned} |Q_{11}| &= AC - \frac{B^2}{4} = (\boldsymbol{v}_i^T \boldsymbol{v}_i)(\boldsymbol{v}_j^T \boldsymbol{v}_j) - \frac{(-2\boldsymbol{v}_i^T \boldsymbol{v}_j)^2}{4} \\ &= \|\boldsymbol{v}_i\|^2 \|\boldsymbol{v}_j\|^2 - \|\boldsymbol{v}_i\|^2 \|\boldsymbol{v}_j\|^2 \cos^2(\theta) \\ &= \|\boldsymbol{v}_i\|^2 \|\boldsymbol{v}_j\|^2 (1 - \cos^2(\theta)) \end{aligned}$$

From this we can establish that $|Q_{11}| \geq 0$, with $|Q_{11}| = 0$ only when the line segments are parallel.

With this knowledge, we can characterize the solution inequality as follows:

1. When the line segments are not parallel: we know the conic is not degenerate because $|Q| \neq 0$, and since $|Q_{11} > 0$, we know the conic is an ellipse (or has no solution, which would result in an imaginary ellipse).

2. When the line segments are parallel: we know the conic is degenerate because $|Q| = 0$, and since $|Q_{11}| = 0$ as well, it consists of two parallel straight lines.

Bounding the Collision Region We bound the portion of the collision region (which we have identified as either an ellipse or the region in-between two parallel lines) in two steps: first we find the global bounds for the region, and then if they lie outside of the $[0, 1] \times [0, 1]$ coordinate box, we solve for the intersection points on the corresponding edge and shrink the bounding region appropriately. This is shown in Fig. 60, where the ellipse extends outside the coordinate box and the bounding box for the valid region is smaller than what would be obtained by simply clipping the full global bounding box. This procedure is important: since we are considering each path segment pair individually, only positions within the path segments can be considered for collisions, and if we take a bounding box for the global collision states, we will be overestimating the collision region to an extent that may be detrimental to the future planning.

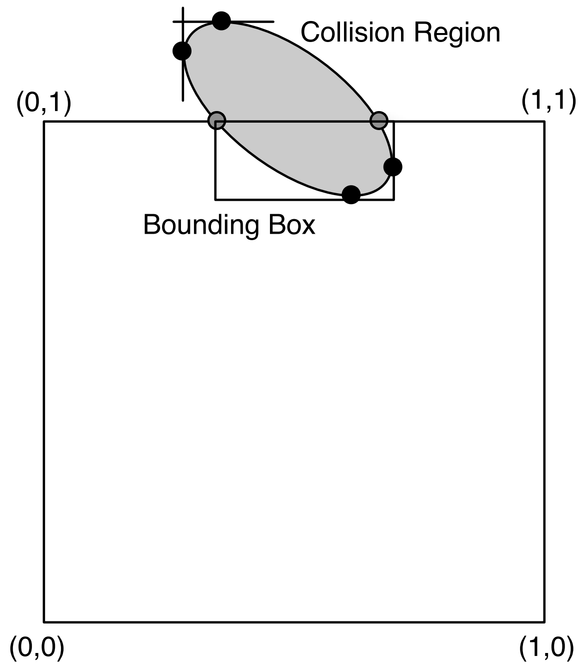


Figure 60: Finding the bounding box of the collision region within our $[0, 1] \times [0, 1]$ coordinate box.

Assuming we have an ellipse, we find the global minimums and maximums by simple algebra. If we were to fix κ_j and look at the quadric as a function of κ_i only, then we have a quadratic:

$$(A)\kappa_i^2 + (B\kappa_j + D)\kappa_i + (C\kappa_j^2 + E\kappa_j + F) = 0$$

Graphically speaking, we are drawing horizontal lines at varying heights, and solving for the two roots of the resulting quadratic will give us the two κ_i values where that horizontal line intersects the ellipse (if such an intersection exists). However, at the top and bottom extrema, there will be only one solution to the quadratic: a double root. This occurs when the discriminant is zero, so we simply solve for when:

$$(B\kappa_j + D)^2 - 4A(C\kappa_j^2 + E\kappa_j + F) = 0$$

This gives us another quadratic, this time in κ_j, whose roots give us the two κ_j coordinates where horizontal lines intersect the ellipse only once, i.e. the top and bottom:

$$(B^2 - 4AC)\kappa_j^2 + (2BD - 4AE)\kappa_j + (D^2 - 4AF) = 0 \tag{53}$$

We can find the quadratic revealing the κ_i values at the left and right extrema by substituting the other way:

$$(C)\kappa_j^2 + (B\kappa_i + E)\kappa_j + (A\kappa_i^2 + D\kappa_i + F) = 0$$

then the discriminant:

$$(B\kappa_i + E)^2 - 4C(A\kappa_i^2 + D\kappa_i + F) = 0$$

gives us the quadratic:

$$(B^2 - 4AC)\kappa_i^2 + (2BE - 4CD)\kappa_i + (E^2 - 4CF) = 0 \tag{54}$$

Taken together, the roots of Eqs. 54 and 53 give us the global bounding box. However, if one of the bounds lies outside one of the edges of our $[0,1] \times [0,1]$ box, then we need to find the points along the edge where the ellipse intersects. For example, if, as shown in Fig. 60, the upper κ_j bound exceeds $\kappa_j = 1$ but the lower bound is still below, then we know that there will be two κ_i values signifying the left and right intersection with the $\kappa_j = 1$ boundary, found by plugging $\kappa_j = 1$ into Eq. 54 and solving for the roots of the quadratic. This procedure is executed four times, once for each edge, resulting in a possible 8 intersection points, as seen in Fig. 61.

As Fig. 61 explains, we will always have eight points under consideration, with the possibility that some may be coincident if a particular extremum lies within the coordinate box. The final step of the procedure is to simply find the bounding box of the eight points, taking into consideration only those which are within the coordinate box.

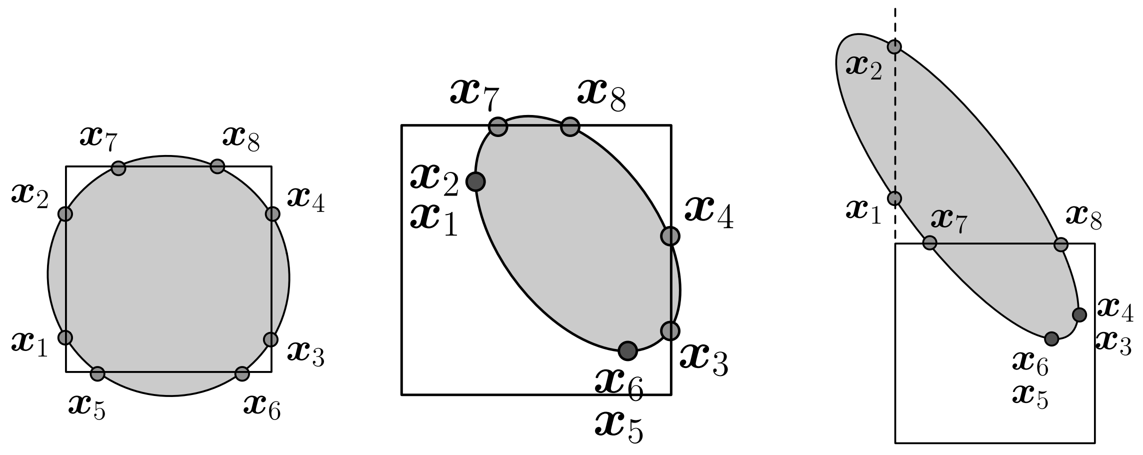


Figure 61: The points $\boldsymbol{x}_1$ through $\boldsymbol{x}_8$ are placed at the intersections of the ellipse with the coordinate box. If the ellipse bounds do not exceed the coordinate box, then both points are placed coincident on the extremum. The points are placed even if they do not lie within the coordinate box, so long as the ellipsoid bounds exceed the given coordinate; these points will be culled from consideration later.

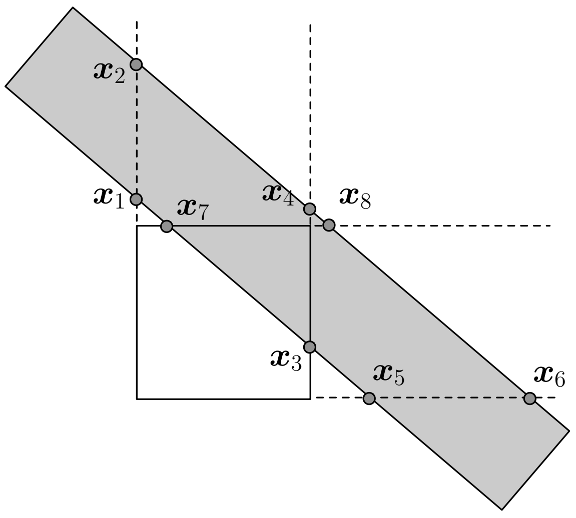


Figure 62: When the collision set is degenerate (the line segments are parallel), intersections are taken with each of the four coordinate edges and we proceed as normal.

If the intersection region is degenerate (when the line segments are parallel), then we automatically must take the intersection with all four coordinate edges, as seen in Fig. 62. The bounding procedure continues as normal from then on.

There are two special cases to be handled when all of the intersection points lie outside of the coordinate box:

1. Each point pair $((\boldsymbol{x}_1, \boldsymbol{x}_2)$, $(\boldsymbol{x}_3, \boldsymbol{x}_4)$, etc) straddles the coordinate box. In this case the collision set covers the coordinate box and so the bounding box is the entire region.

2. Each point pair lies either above or below the coordinate region. In this case the collision set lies entirely outside the coordinate box and so there is no collision possible between these two segments.

The entire CollisionBound algorithm is given in Algorithm 9. The function SolveQuadratic takes the (A, B, C) coefficients of the standard quadratic equation, $Ax^2 + Bx + C = 0$, and returns the ordered pair of the two roots, (x_{min}, x_{max}).

Algorithm 9 CollisionBound$(\boldsymbol{p}_1, \boldsymbol{v}_1, \boldsymbol{p}_2, \boldsymbol{v}_2, S)$

Require: Two line segments $(\boldsymbol{p}_1, \boldsymbol{v}_1)$ and $(\boldsymbol{p}_2, \boldsymbol{v}_2)$ and a separation distance S.

Ensure: A bounding box over the local path-variables for the pairs that where the robot separation drops below S.

```
1: $A \leftarrow \boldsymbol{v}_i^T \boldsymbol{v}_i$
2: $B \leftarrow -2\boldsymbol{v}_i^T \boldsymbol{v}_j$
3: $C \leftarrow \boldsymbol{v}_j^T \boldsymbol{v}_j$
4: $D \leftarrow 2(\boldsymbol{p}_i - \boldsymbol{p}_j)^T \boldsymbol{v}_i$
5: $E \leftarrow -2(\boldsymbol{p}_i - \boldsymbol{p}_j)^T \boldsymbol{v}_j$
6: $F \leftarrow (\boldsymbol{p}_i - \boldsymbol{p}_j)^T (\boldsymbol{p}_i - \boldsymbol{p}_j) - S^2$
7: if $B^2 - 4AC = 0$ then // Degenerate Section
8:         $x_- \leftarrow -1, x_+ \leftarrow 2, y_- \leftarrow -1, y_+ \leftarrow 2$
```

```
else // Ellipse
    $(x_-, x_+) \leftarrow \textsc{SolveQuadratic}(B^2 - 4AC, 2BE - 4CD, E^2 - 4CF)$
    $(y_-, y_+) \leftarrow \textsc{SolveQuadratic}(B^2 - 4AC, 2BD - 4AE, D^2 - 4AF)$
    if $x_- \geq 1$ or $x_+ \leq 0$ or $y_- \geq 1$ or $y_+ \leq 0$ then
        return $\emptyset$
    end if
end if

if $x_- < 0$ then // Intersect with Left Coordinate
    $(q_1, q_2) \leftarrow \textsc{SolveQuadratic}(C, E, F)$
    $\boldsymbol{x}_1 \leftarrow (0, q_1)$
    $\boldsymbol{x}_2 \leftarrow (0, q_2)$
else // Find Left Extremum
    $(q, -) \leftarrow \textsc{SolveQuadratic}(C, Bx_- + E, Ax_-^2 + Dx_- + F)$
    $\boldsymbol{x}_1, \boldsymbol{x}_2 \leftarrow (x_-, q)$
end if
if $x_+ > 1$ then // Intersect with Right Coordinate
    $(q_1, q_2) \leftarrow \textsc{SolveQuadratic}(C, B + E, A + D + F)$
    $\boldsymbol{x}_3 \leftarrow (1, q_1)$
    $\boldsymbol{x}_4 \leftarrow (1, q_2)$
else // Find Right Extremum
    $(q, -) \leftarrow \textsc{SolveQuadratic}(C, Bx_+ + E, Ax_+^2 + Dx_+ + F)$
    $\boldsymbol{x}_3, \boldsymbol{x}_4 \leftarrow (x_+, q)$
end if

if $y_- < 0$ then // Intersect with Lower Coordinate
    $(q_1, q_2) \leftarrow \textsc{SolveQuadratic}(A, D, F)$
    $\boldsymbol{x}_5 \leftarrow (q_1, 0)$
    $\boldsymbol{x}_6 \leftarrow (q_2, 0)$
else // Find Lower Extremum
```

```
37:         $(q, -) \leftarrow \textsc{SolveQuadratic}(A, By_- + D, Cy_-^2 + Ey_- + F)$
38:         $\boldsymbol{x}_5, \boldsymbol{x}_6 \leftarrow (q, y_-)$
39: end if
40: if $y_+ > 1$ then // Intersect with Upper Coordinate
41:         $(q_1, q_2) \leftarrow \textsc{SolveQuadratic}(A, B + D, C + E + F)$
42:         $\boldsymbol{x}_7 \leftarrow (q_1, 1)$
43:         $\boldsymbol{x}_8 \leftarrow (q_2, 1)$
44: else // Find Upper Extremum
45:         $(q, -) \leftarrow \textsc{SolveQuadratic}(A, By_+ + D, Cy_+^2 + Ey_+ + F)$
46:         $\boldsymbol{x}_7, \boldsymbol{x}_8 \leftarrow (q, y_+)$
47: end if
48: $bbox \leftarrow \emptyset$
49: for Each intersection point, $\boldsymbol{x}_i$ do
50:         if $\boldsymbol{x}_i$ inside coordinate box then
51:                 Expand $bbox$ to include $\boldsymbol{x}_i$
52:         end if
53: end for
54: if $bbox = \emptyset$ then
55:         if $\boldsymbol{x}_1(2) < 0$ and $\boldsymbol{x}_2(2) > 1$ and $\boldsymbol{x}_3(2) < 0$ and $\boldsymbol{x}_4(2) > 0$ and $\boldsymbol{x}_5(1) < 0$ and
            $\boldsymbol{x}_6(1) > 1$ and $\boldsymbol{x}_7(1) < 0$ and $\boldsymbol{x}_8(1) > 1$ then // Collision Set Surrounds
56:                 $bbox \leftarrow [0, 1] \times [0, 1]$
57:         end if
58: end if
59: return $bbox$
```

For multi-segment paths, the CollisionBound algorithm is applied to each pair of segments (one from each path), then the resulting bound is transformed into the global path coordinates using Eq. 49. Putting all of these bounds together results in the box-approximation to the true coordination diagram. This process is demonstrated in Fig. 63.

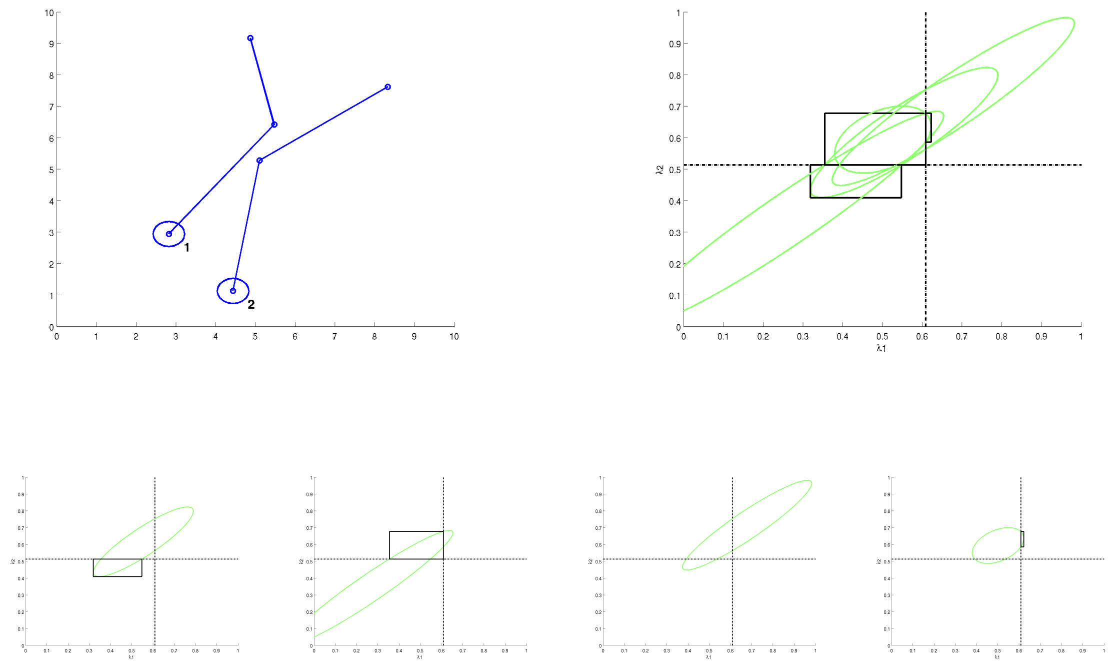


Figure 63: An example of a multisegment path interaction. The upper-left diagram shows the paths of the two robots, and the upper-right diagram shows the resulting bounding box union taken from the interaction of each of the segment pairs. Below are the results of CollisionBound for each segment pair. From these it is evident that only the portion of the quadratic falling within the local coordinate box is bounded, and for some segment pairs it is possible for the interaction to be completely outside of this coordinate box. The union of the bounding boxes contains the true interaction set.

4.3.2 Combining Three or More Geodesic Paths

When there are more than two robots, then the dimension of the coordination diagram will grow accordingly. However, the diagram has special structure: if two of the robots are colliding, any position of the other robots is invalid. Because of this, the collision regions for one pair of robots become cylinders in the general coordination diagram.

For example, in Fig. 64, the earlier example of two robots crossing is shown with a third robot that does not interact. When the first and second robots collide, any value of γ_3 is invalid. This example continues in Fig. 65, where this time the third robot does cross paths. The interaction of each pair of robots is projected into the full three-dimensional coordination diagram.

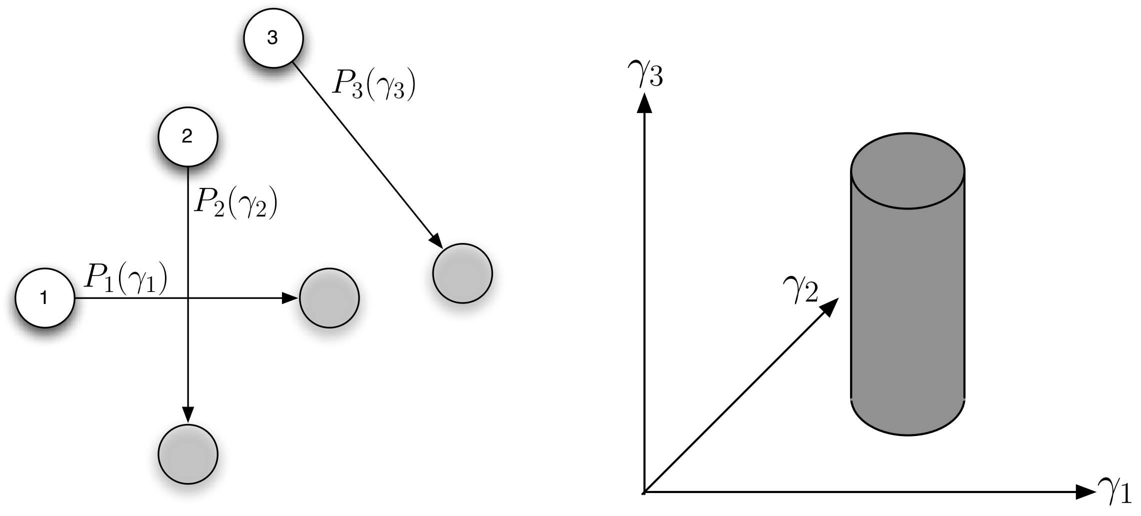


Figure 64: When more robots are added, the interactions of any pair of robots will become a cylinder in the full coordination diagram since any position of the other robots cannot change the fact that two of them have collided.

For multi-segment paths, the projection becomes more complicated but is still composed of a union of cylinders. In Fig. 66 is an example of a three-robot system with multi-segment paths interacting.

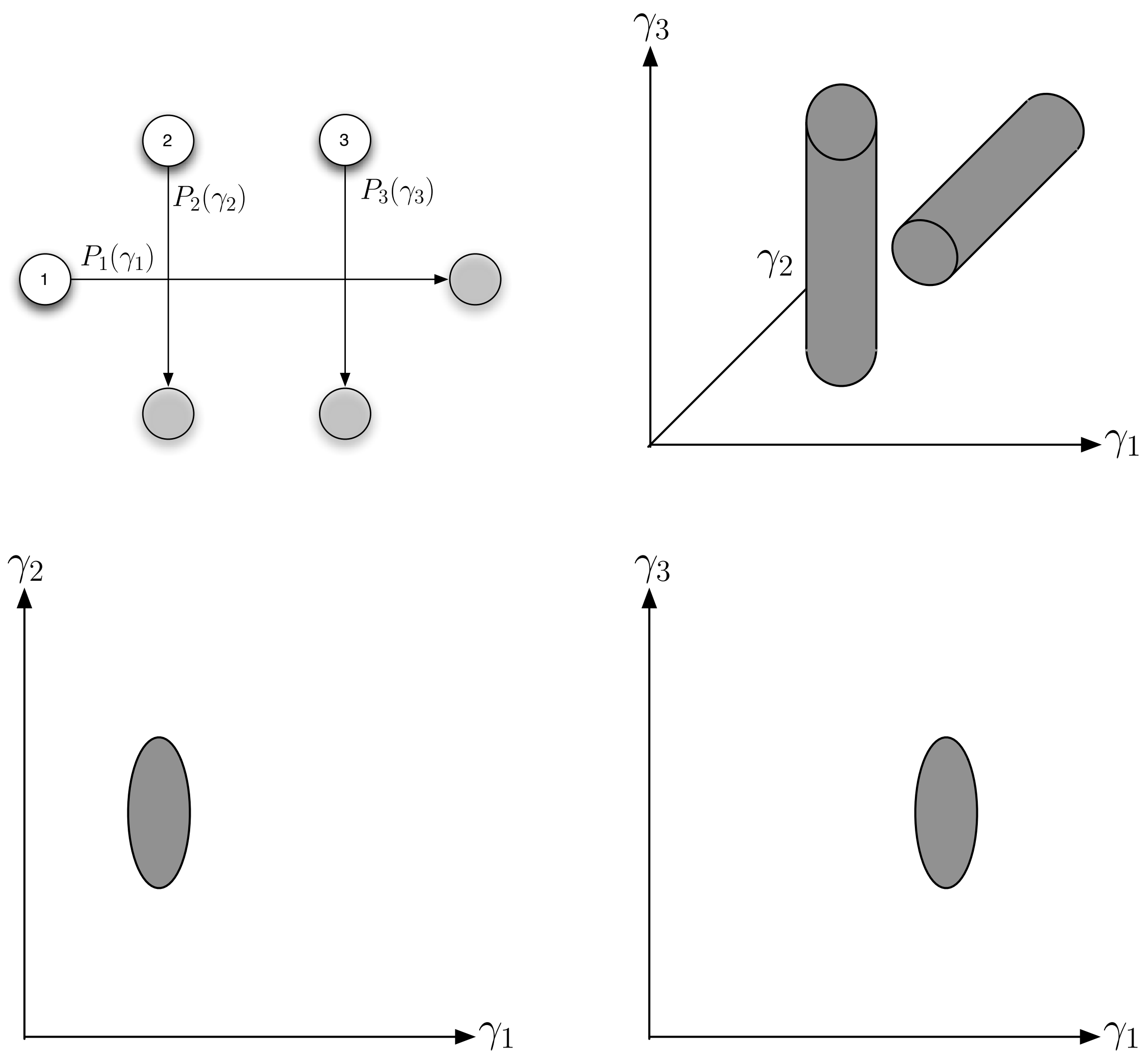


Figure 65: Another example of interactions between pairs of robots resulting in cylinders in the full coordination diagram. Here the coordination diagrams of the two pairs of interest are shown below, and their projection and union is shown in the upper-right.

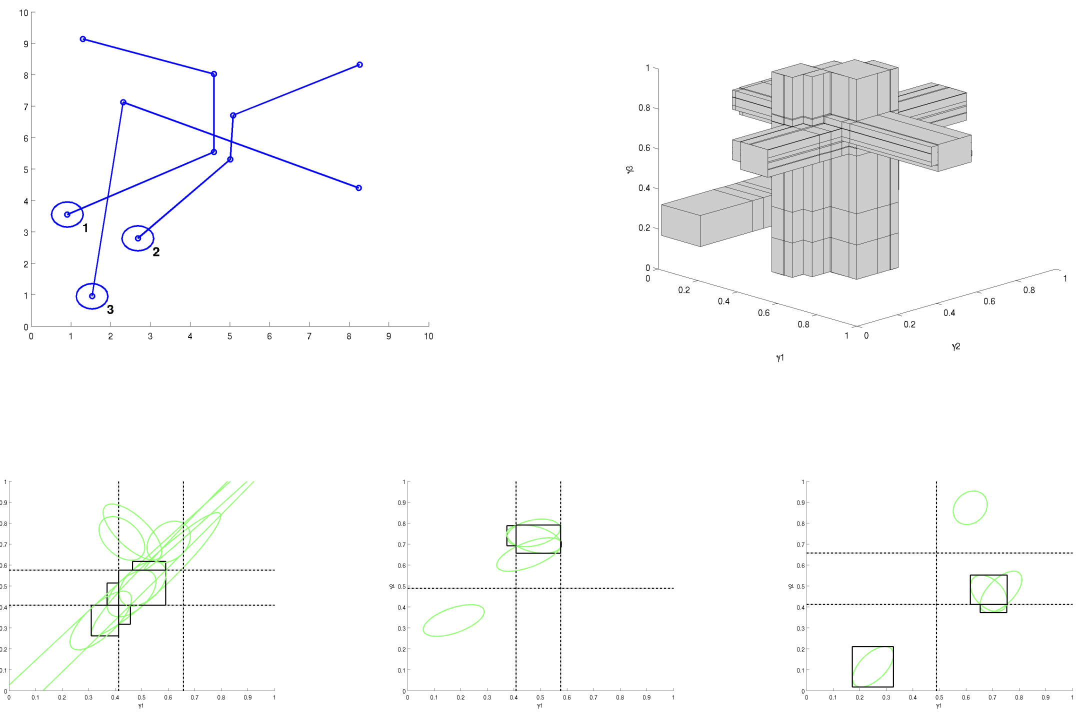


Figure 66: An example of interaction between three robots with multi-segment paths resulting in more complicated projected collision areas. The individual robot-pair collision diagrams below correspond, from left to right, with the interaction of Robot 1 with 2, Robot 2 with Robot 3, and Robot 3 with Robot 1. Ordering the interactions in this way preserves the orientation of the diagrams with respect to the full 3-dimensional picture shown in the upper-right.

4.3.3 Searching the Coordination Diagram of Geodesic Paths

It is important to note that it is possible to quickly calculate free directions to move without having to completely characterize the full N-dimensional projected collision areas. This proceeds in several steps:

1. Create two boolean vectors, *forward* and *backward*, whose ith elements indicate whether it is possible to move forward or backward along the ith axis in the

full coordination diagram. All directions are true initially.

2. Loop through each unique robot pair and consult the corresponding set of bounding boxes surrounding the coordination diagram for this pair.

3. Loop through each box on the diagram and check if the point is touching the box. A few different cases are shown in Fig. 67. Assuming the point is touching the box, then only if the point lies within an edge, such as point B or C in the figure, do directions need to be cut off immediately. However, the directions which lie tangent to the bounding box need to be marked. Each possible direction is labeled as 1+, 1−, 2+, and 2− in the figure. The number of times that each direction is tangent is incremented as tangency is encountered. For example, at point A, the 1+ and 2− directions are incremented. At point B, when the point is first encountered as a corner to the left-hand box, the 1− and 2+ directions are incremented. Then when the point is encountered on the side of the right-hand box, the 2+ and 2− directions are incremented. After all the boxes are accounted for, any direction which was tangent to two or more boxes is an illegal direction, as demonstrated with point B.

The free directions are eliminated as boxes prevent movement. Once the free directions are established, neighbor coordinates can be found by incrementing and decrementing the current coordinate as appropriate, where the incremental distance is determined by the merging of box extents for each axis across all possible pairings.

Once the coordination diagram is established as a set of bounding boxes describing keep-out areas together with the axis pairs they belong to, the final planning step is to find a sequence of motions in path-coordinate space that allow for travel from the initial coordinate, all path coordinates equal to zero, to the final coordinate, all path coordinates equal to one. This is accomplished using an $A^\star$ algorithm where the heuristic and distance functions are taken as the Euclidean distance in the full path-coordinate space.

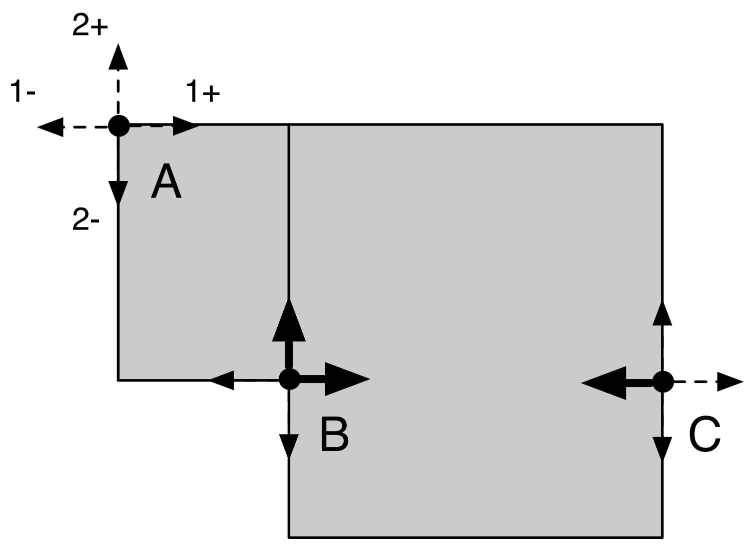


Figure 67: Different cases to consider when eliminating free directions at a given point. For corner points like point A, all directions are free, but two directions are tangent to a box. When paired with another box, such as at point B, a direction being tangent to multiple boxes means it is not free. Points such as C have a non-free direction and two tangent directions.

There is a natural discretization of the space in terms of the bounding boxes. Since motion in a given direction will only be halted by reaching a bounding box in one of the associated coordination diagram, the union of all bounding box limits along this axis serves to define all interesting points.

4.3.4 Searching the Coordination Diagram of Adjacency Paths

Given a set of adjacency paths represented as an ordered list of cells, the search of the coordination diagram is similar to that for geodesics, but instead of boxes bounding collision configurations, we impose graph property constraints on path configurations. In particular, we are interested in maintaining a visibility-chain between all agents and ensuring that collision configurations are avoided.

Given a particular robot configuration, the graph S is the subgraph of the MVG formed by the nodes of the MVG occupied by the robots and any edges that remain. As seen in Fig. 68, although the MVG is always connected, the subgraph need not be. The bold edges are those that remain when the nodeset is limited to only those occupied, but when the node shifts, connectivity can be lost.

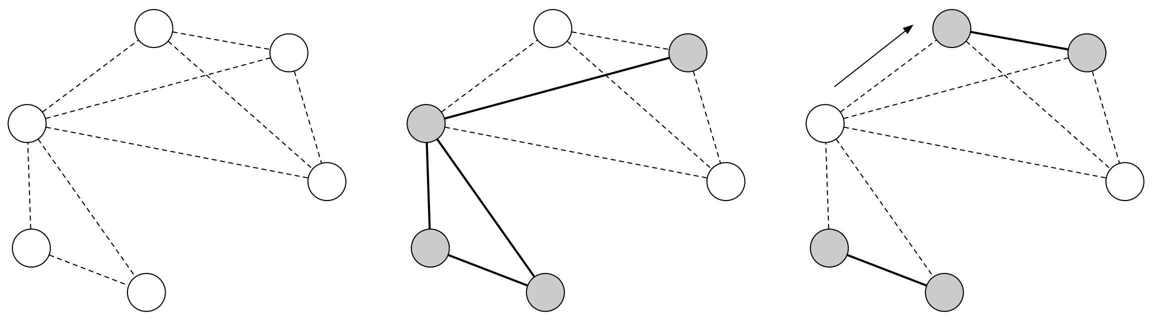


Figure 68: Although the MVG is always connected, a subgraph of the MVG need not be. As the nodes shift from the configuration in the center picture to those in the right picture, the induced subgraph becomes disconnected.

As established earlier, a visibility-chain exists if the subgraph S is connected. This test can be carried out by a breadth-first search from an arbitrary node of S. If any nodes remain untouched after the search terminates, then the subgraph is multiply connected and the visibility-chain is broken.

If a configuration of the robots along their adjacency paths leads to an unconnected subgraph in the MVG, then the configuration is disallowed.

4.3.5 Hybrid Approach

Both the previous techniques, using geodesic paths and using cell-sequences from the adjacency graph, have strengths and weaknesses. The geodesic path technique leads to efficient straight-line movements but fails to account for visibility when deciding movements. The adjacency path technique allows for guarantees of mutual visibility at each step of the process but fails to incorporate collision avoidance.

A hybrid approach is possible: using geodesic paths but refining the splits beyond just straight-line segments and identifying parts of the segments lying within each individual visibility cell. This idea is shown in Fig. 69. The geodesic path is found between the start and goal locations, but the path is split into a series of segments according to membership with visibility segments. The geodesic-scheduling approach can then be augmented with additional constraints requiring maintenance of the deployment topology by ensuring that pairs of segments lie within cell pairs that are mutually visible.

If two robots are required to maintain line-of-sight according to the deployment topology, then for any pair of segments that do not lie within or border upon cells that are mutually visible, the corresponding path variable intervals are blocked out.

There are two fundamental constraints with this approach. First, constraining visibility is impossible to do in this way if topology modification is used since new links may be formed to agents that are not currently visible, but will become visible once the movement is complete. Second, it is possible that the paths between two agents may diverge to the point that visibility can no longer be maintained. This could be the case if two agents choose different homotopy classes of paths when confronted with a hole in the environment. If this happens, the visibility can not be maintained without modifying one or both paths. This simple path coordination can not handle this case; a more sophisticated method should be used.

4.4 Experiments and Simulation

Here we shall see some experimental results motivating the use of visibility as a communication model and simulations demonstrating the deployment of a team of robots to follow a moving target while trying to maintain visibility.

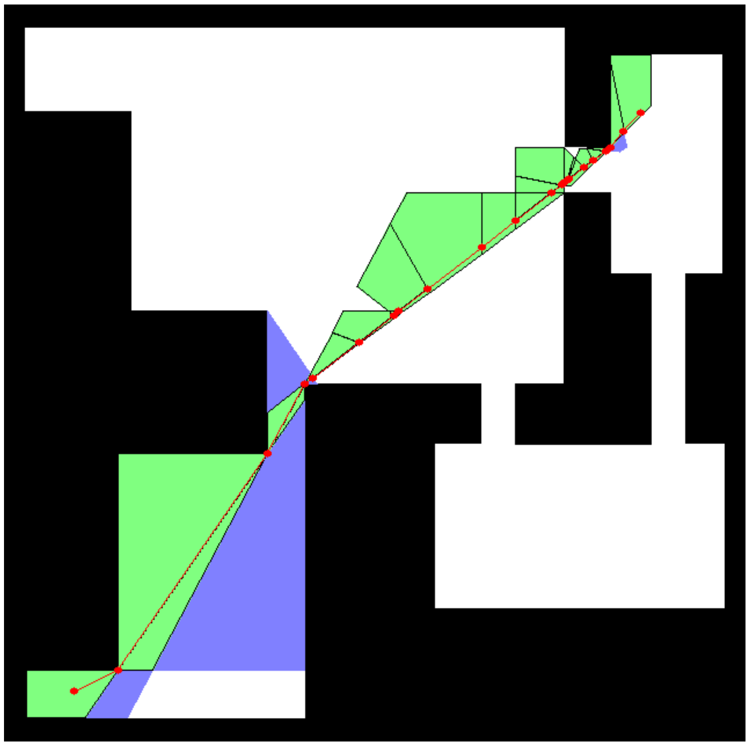


Figure 69: The original geodesic path is refined into segments lying within individual visibility cells (or the boundary between two visibility cells). Doing so enables the extension of the geodesic path scheduling to include visibility constraints by blocking out path variable interval pairs corresponding to segments that do not lie within or border upon cells that are mutually visible if the two robots belonging to these paths are required to maintain visibility according to the deployment topology.

4.4.1 Motivating Visibility as a Communication Model

In order to check whether there is any gain by insisting on visibility between agents in order to facilitate wireless communications, we performed a simple experiment. Lacking any sophisticated wireless communications model, we use signal strength as a proxy for bandwidth; this ignores a great deal of the intricacies happening in the OSI logical link layer and higher, but does give some insight [40]. Our test consists of a simple sender/receiver pair placed at varying distances and with differing line-of-sight conditions. The positions used are shown in Figure 70. At each position, several signal strength measurements were made to demonstrate that visibility has a dramatic effect on the power loss [92]. As can be seen in Figure 71, the positions corresponding to loss of line-of-sight experienced a substantial drop in received signal power. We claim that this appreciably degrades network performance, but plan to use direct bandwidth tests to quantify this in the future.

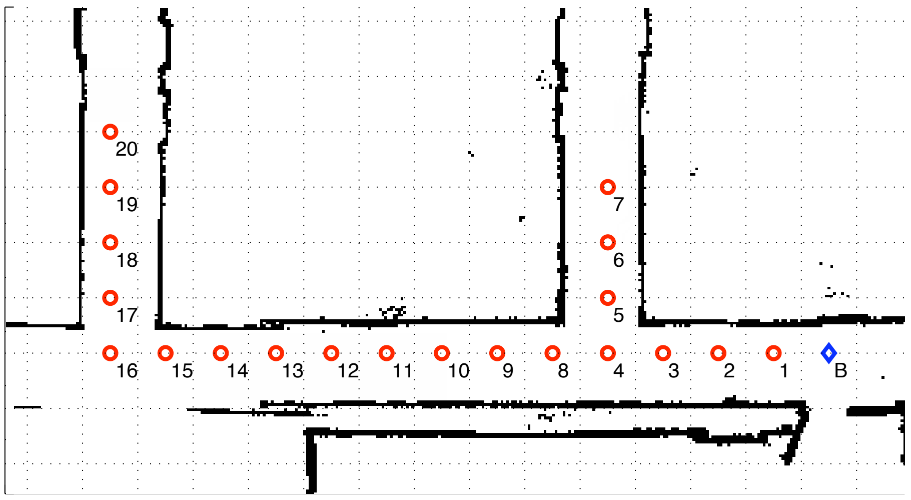


Figure 70: The numbered receiver locations for the signal strength test. The base was placed in the lower-right corner (marked with a B), and the other receiver was moved to the other locations placed at 1m intervals. Some of the positions veer off into the hallway and lose line-of-sight with the base.

4.4.2 Deployment Simulations

For our deployment trials, we use one fixed base, one user-driven agent, and several mobile robots as support. The robots are placed in initial locations and their deployment topology is assigned to be a line, with the fixed base at one end and the agent at the other. Goal points are then repeatedly sent to the agent to simulate wandering or exploring; the goal points are deliberately chosen to be within the current line-of-sight of the agent or just around a visible corner, to further simulate the act of exploration. Each time a goal point is sent, a new deployment is calculated to

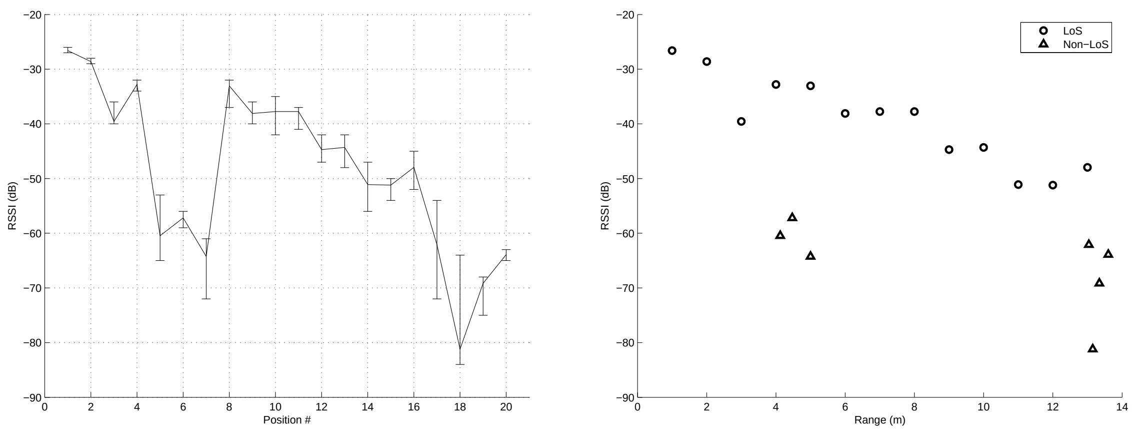


Figure 71: The signal strength measured at various locations, expressed as decibels of loss. In the first plot, the locations are plotted one-by-one, and in the second, the results are plotted according to range, with the circles corresponding to ranges of positions within line-of-sight and triangles corresponding to ranges of positions not within line-of-sight. There is a substantial power drop when line-of-sight is lost.

connect the new goal point with the base station and also produce a mutually-visible tree with minimal movement. It is important to note that this tree will be mutually visible when the movement is complete, but may not be mutually-visible at the instant it is computed. Also, during the motion, the final positions may never be reached because the agent chooses a new point. We set the speeds of the agent and robots to be the same, so in some cases the robots can not keep up with the deployments. What happens is a *best effort* sequence of deployments. Some snapshots are seen in Figures 72, 73, 74, and 75. The focus is on times when the topology shifts as a different path is found to connect the spot that the agent declares it intends to go to. In the figure captions, we note the amount of time that the agent is visible to the robots as a percentage of the entire sequence.

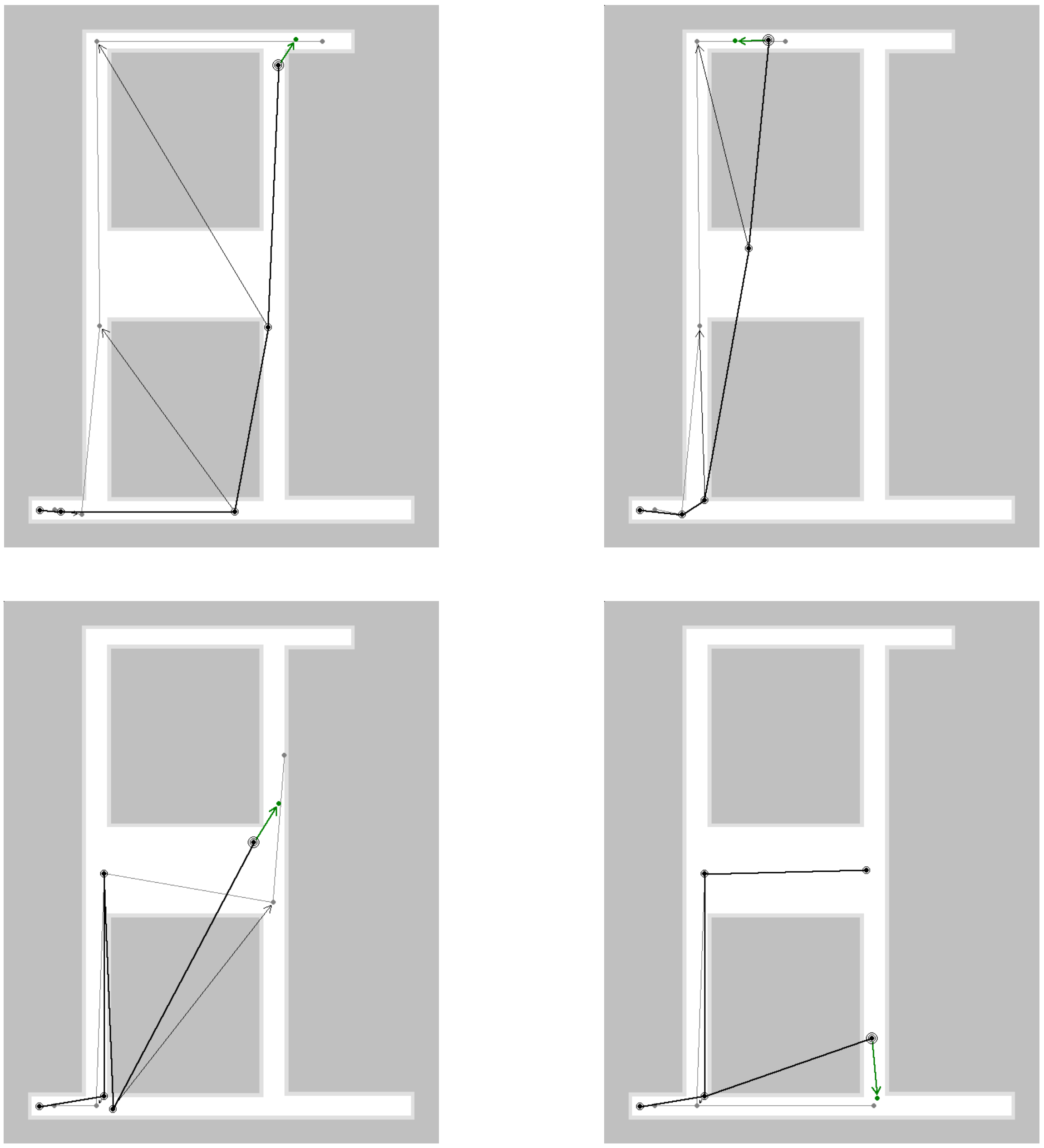


Figure 72: A simulated sequence of deployments in the l457mod environment. The doubly-circled point is the user-controlled agent, the thick black lines indicate the current deployment with robot locations circled, the thin gray lines indicate the resulting deployment calculated to support this new position, and the arrows indicate how each robot moves between the two. The moving agent was visible to at least one robot for 68% of the sequence.

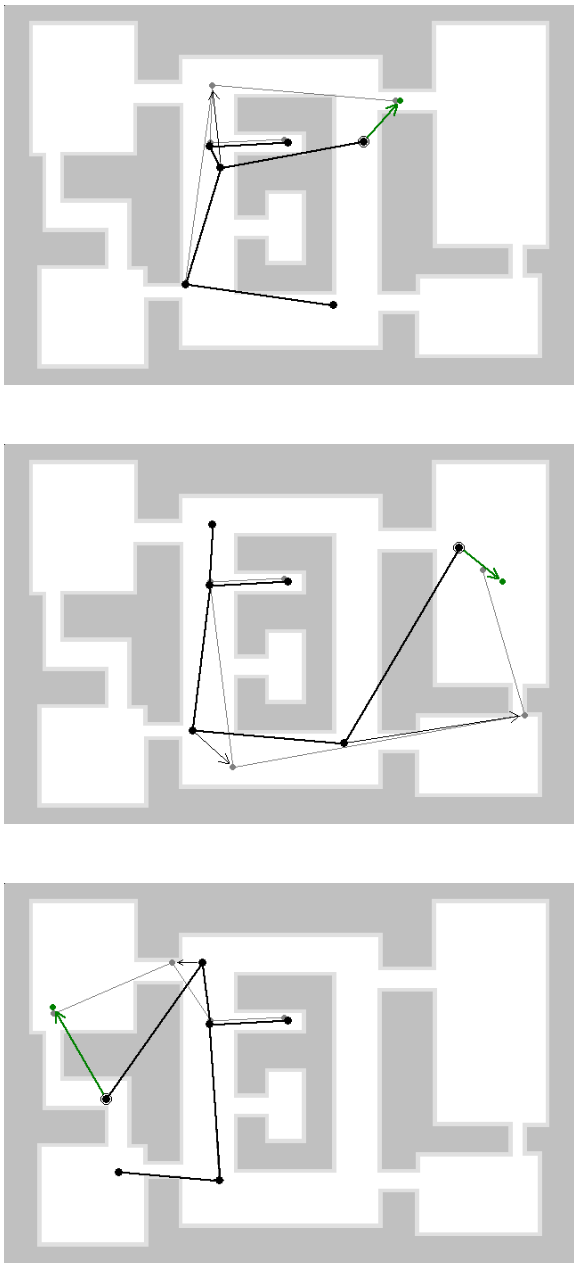


Figure 73: A simulated sequence of deployments in the room3 environment. The doubly-circled point is the user-controlled agent, the thick black lines indicate the current deployment with robot locations circled, the thin gray lines indicate the resulting deployment calculated to support this new position, and the arrows indicate how each robot moves between the two. The moving agent was visible to at least one robot for 81% of the sequence.

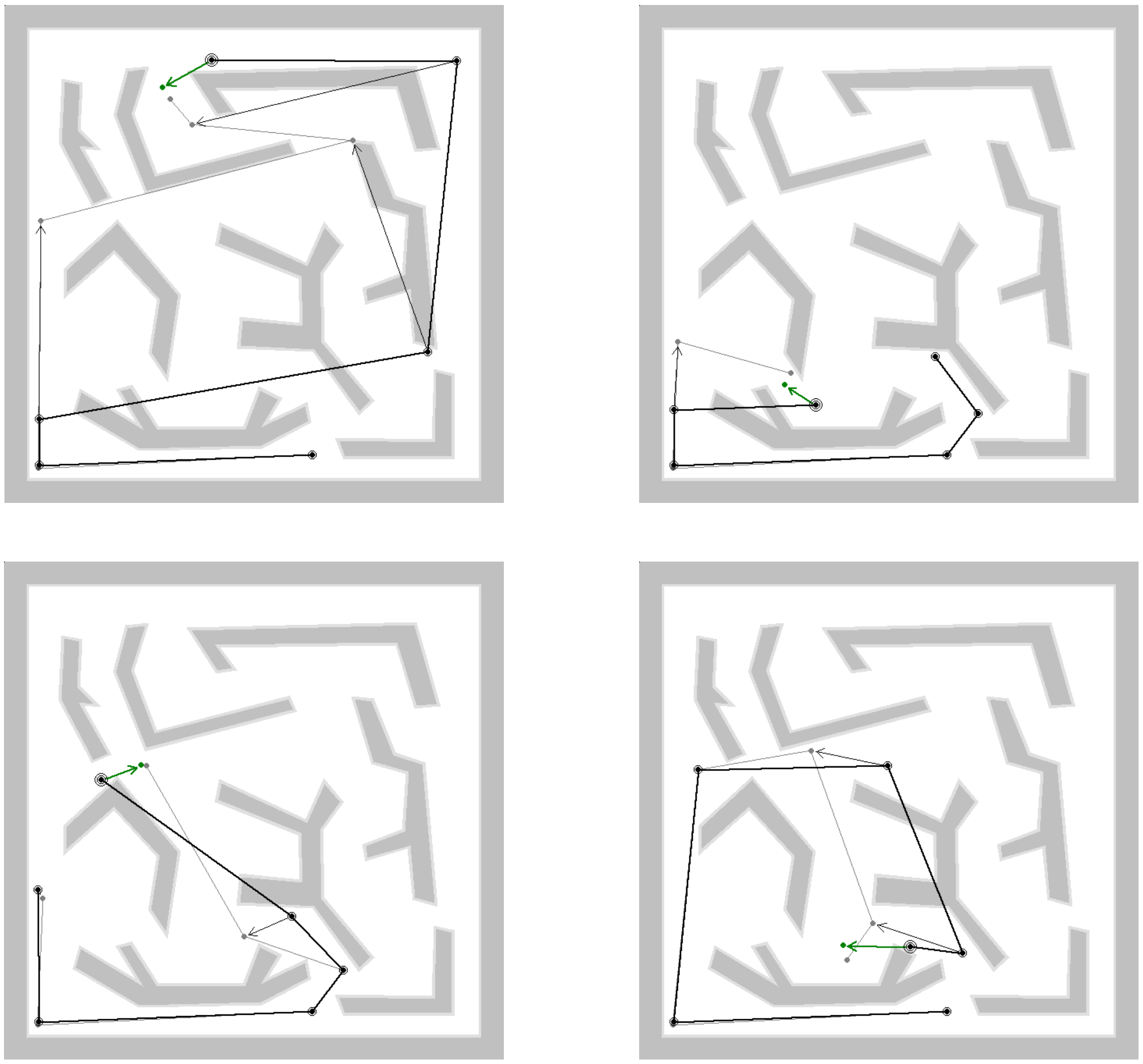


Figure 74: A simulated sequence of deployments in the maze environment. The doubly-circled point is the user-controlled agent, the thick black lines indicate the current deployment with robot locations circled, the thin gray lines indicate the resulting deployment calculated to support this new position, and the arrows indicate how each robot moves between the two. The moving agent was visible to at least one robot for 79% of the sequence.

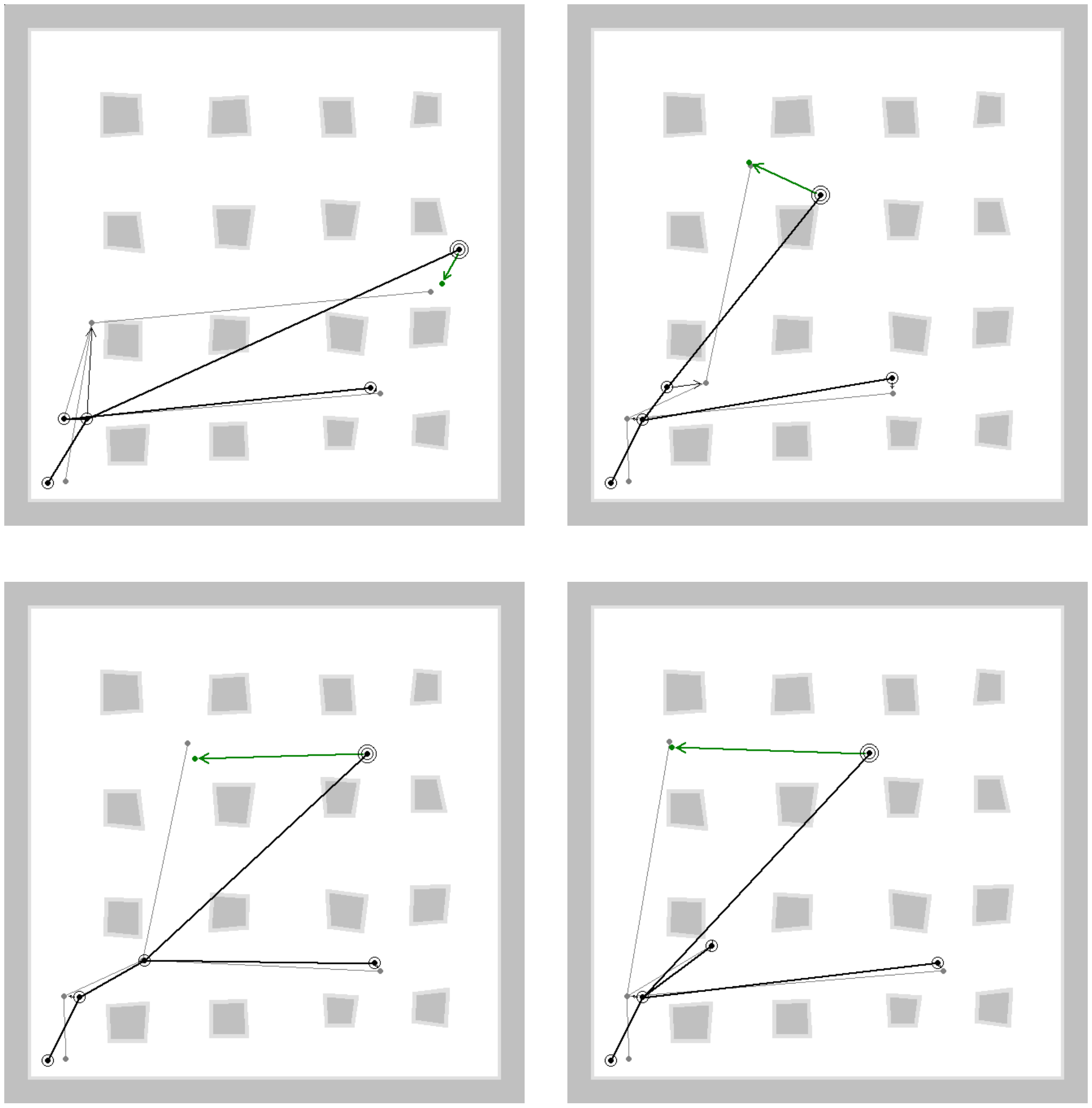


Figure 75: A simulated sequence of deployments in the cityblocks environment. The doubly-circled point is the user-controlled agent, the thick black lines indicate the current deployment with robot locations circled, the thin gray lines indicate the resulting deployment calculated to support this new position, and the arrows indicate how each robot moves between the two. The moving agent was visible to at least one robot for 87% of the sequence.

4.5 Summary

In this chapter we discussed details for executing the deployments found in Chapter 3 and analyzed the complexity of the algorithms found there. We devised movement schedules for the robots by combining geodesic paths found for each individual robot and then sequencing their execution to avoid collisions and maintain visibility according to the deployment topology. The path segments were split according to both their inclusion within the visibility cells and the points where the robots interact through collision. These segment-segment interactions translate into interval constraints on the concatenated path variables of all of the robots and we use $A^\star$ planning to find a schedule in path-variable-space that avoids them.

We performed a simple experiment to motivate the requirement of visibility for communication by demonstrating that loss of visibility causes a large drop in signal strength. Finally, we performed simulated deployment tests by manually driving an agent and computing and sequencing deployments to keep the agent connected to the base station.

Chapter 5

Conclusion

This thesis discussed the problem of using mobile robots to interact with an independent agent as it travels through a complex environment. We considered two aspects of the scenario: localizing and tracking the agent using range-only sensors within line-of-sight, and deploying and moving a team of robots to maintain a chain of visibility for communication between the agent and a fixed base station. We presented the technical material and then provided simulated deployments to demonstrate the concept. Ultimately, this work will prove useful for the stated vision of deploying teams of robots to assist emergency personnel in gathering information while minimizing risk.

In Chapter 2, we solved the problem of localizing an independent agent using range-only measurements from mobile sensors and then choosing sensor movements to speed-up localization time and improve precision. We did this with a novel estimation framework based on using an extended state space to transform a nonlinear problem into a linear one and then using a standard linear filtering technique. In our case, we used a set-based technique with ellipsoid set representations.

From this estimation framework, we drew an analogy to existing techniques in control for information acquisition in probabilistic linear filters and derived equations for the new framework.

Both the estimation and control were implemented on a hardware platform and the technique was validated experimentally, with some interpretation and explanation of weaknesses of the set-based assumption and how the filter robustly recovered.

In Chapter 3, we investigated the problem of deploying a team of robots to maintain a visibility chain with moving targets in a complex environment. We accomplished this by discretizing the environment into polygonal cells and pre-computing visibility information between them. From this, we cast all of our deployment problems as graph computations and used standard algorithms to find the associated graph structures.

We introduced the idea of deployment topologies where the required visibility relationships were separated from the physical robot locations within the environment, and we saw that if the topology is given, finding the optimal robot locations is easy. This led to an algorithm for computing robot motions to accommodate changing target locations while keeping the deployment topology fixed. We extended this by devising a heuristic to check other topologies. These algorithms led to the posing of the *fixed-size Steiner tree* and *minimal movement Steiner tree* problems.

One important building block was devising edge weights for the visibility links within the environment, and we constructed them to express preferences for minimizing robot count, making the deployment as compact as possible, and utilizing larger visibility cells as much as possible.

In Chapter 4, we discussed details for executing the deployments of Chapter 3 and analyzed their algorithmic complexity. We devised movement schedules for the robots by combining geodesic paths found for each individual robot and then sequencing their execution to avoid collisions and maintain visibility according to the deployment topology. The path segments were split according to both their inclusion within the visibility cells and the points where the robots interact through collision. These segment-segment interactions translate into interval constraints on the concatenated path variables of all of the robots and we use $A^\star$ planning to find a schedule in path-variable-space that avoids them.

We performed a simple experiment to motivate the requirement of visibility for communication by demonstrating that loss of visibility causes a large drop in signal strength. Finally, we performed simulated deployment tests by manually driving an agent and computing and sequencing deployments to keep the agent connected to the base station.

5.1 Challenges and Future Work

There remain many barriers to practical implementation of all the techniques presented, despite the hardware experiments performed. Investigating and breaking these barriers would be the most fruitful direction for future work, and we will review a few avenues of research to accomplish this.

In the localization technique of Chapter 2, we saw a nonlinear transformation that handled the problem of range-only measurements, but finding transformations for other nonlinear problems remains an open problem. The work of [19] promises to be the most general treatment for this problem. Once ellipsoid sets are found, translating them into feasibility sets in the original state space remains a challenge and the efficacy of the techniques suggested remains dubious. The bounded error assumption may also be unnecessarily conservative since a few measurement outliers could require the error bound be quite large to avoid inconsistencies such as those encountered during experiment.

Though the analogy between the set-based technique used previously established techniques with Guassian representations produced results which were promising, the link is still technically unmotivated. Moreover, the control strategy itself is a simple greedy approach, and there still remains work to be done in finding planned approaches for carving out the information space efficiently. This would necessitate multi-robot coordination of a different character than that investigated in this work during the discussion of executing robot deployments.

The visibility-based deployment calculations of Chapter 3 all hinge on the special discretization of the polygonal environment. This presents two problems.

First, as the environment complexity grows, the discretization will become increasingly complex as well and the visibility and shortest-path computations that must be computed will begin to take an exorbitant amount of time. The inclusion of the aspect lines in the decomposition is especially costly due to the explosion of cells produced; additionally, many of the cells are small and relatively useless from a practical control standpoint, even though they are important from a theoretical visibility standpoint.

Second, in practical situations, the environment map will not be as clean as those used in this work. Even if the environment is mapped beforehand, this map will likely be constructed using noisy laser scans and integrated into an occupancy grid or segment map. Using the inflection and aspect line approach will either require extensive preprocessing to clean up the map, or we will need an entirely new decomposition concept that preserves the visibility-centered and topological nature of these decompositions but generates the visibility cells in some other way, perhaps more experimentally motivated.

This does not even begin to approach the problem of performing visibility-based planning in an environment that is still being discovered. If a suitable decomposition-generation scheme could be devised that would allow for incorporation of measurements online, there would still remain the problem of integrating new cells with visibility and shortest-path computations in order to make them usable within the framework. If there was a way to do this incrementally, then simultaneous discovery and planning could be possible.

Another useful extension would be to relax the hard visibility constraints and instead assign edge weights based on communications quality using a more general model that takes into account fading from non-line-of-sight conditions. We could use this new set of weights to compute “radio-friendly” deployments. The biggest challenge would be implementing a high-quality communications model.

There still remains the question of robustness. A tree graph is a poor communications topology because a single point-of-failure will disable the communications. Requiring a biconnected graph as in [8] would eliminate this problem, but now the redeployment cannot be solved optimally using dynamic programming. Instead, we would be forced to consider approximate techniques similar to those used for loopy belief propagation in Bayesian networks.

The plan refinement and execution of Chapter 4 also leaves open questions tied to the discretization. In particular, the aspect decomposition creates many cells which are small or awkwardly-shaped, leading to an explosion in the number of segments that must be considered and tested for interaction with each other. Moreover, the small cells are capturing changes in the visibility topology between areas of the environment that may not even be in use by the robot configuration at a given time. This issue could possibly be solved by simplifying the decomposition in a way that effectively “linearizes” around the current deployment and the visibility lines that must be maintained, only creating cell transitions for inflection points that can break line of sight within some horizon of the current deployment. This could create smaller-scale decompositions and thereby greatly reduce the complexity of the mutual-visibility and shortest-path computations, perhaps even scaling down to real-time performance on a reasonably powerful mobile robot computer.

The inability to guarantee that visibility can be maintained during motion is a good reason to question the use of geodesics as a path primitive to use for coordination. Other approaches like dynamic programming over a discretization of the Generalized Voronoi Graph, such as in [50], might be warranted.

There is also a vast gulf of research separating the centralized path sequencing operation used here and a distributed implementation. Even though such an implementation was not a stated goal, current research trends are towards such approaches because they offer the promise of harnessing the collective processing power of several small computers to produce results that would otherwise require one large one. There has been no research in parallelization or distribution of our path sequencing, but the pair-wise nature of the constraint generation would suggest that the amount of communication would be limited. Performing the distributed graph search based on these softly-defined constraints would be a challenge.

However, if the discretization scheme were re-imagined in order to be more closely tied to online measurements, and this could be further extended into efficient online updates of visibility and shortest-path computations, the presented work would provide the foundation for a very powerful planning and control framework to handle visibility maintenance.

Bibliography

[1] S. Alexander, R. Bishop, and R. Ghrist, "Pursuit and evasion in non-convex domains of arbitrary dimensions," in *Proceedings of Robotics: Science and Systems II*, 2006.

[2] K. B. Ariyur and M. Krstic, *Real Time Optimization by Extremum Seeking Control*. New York, NY, USA: John Wiley & Sons, Inc., 2003.

[3] R. Arkin and J. Diaz, "Line-of-sight constrained exploration for reactive multiagent robotic teams," in *Advanced Motion Control, 2002. 7th International Workshop on*, 2002, pp. 455–461.

[4] S. Atiya and G. D. Hager, "Real-time vision-based robot localization," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 6, pp. 785–800, 1993.

[5] T. Bandyopadhyay, M. H. Ang, and D. Hsu, "Motion planning for 3-d target tracking among obstacles," in *Proc. Int. Symp. on Robotics Research*, 2007.

[6] T. Bandyopadhyay, Y. Li, M. H. Ang, and D. Hsu, "A greedy strategy for tracking a locally predictable target among obstacles," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, 2006, pp. 2342–2347.

[7] N. Banerjee, M. D. Corner, D. Towsley, and B. N. Levine, "Relays, Base Stations, and Meshes: Enhancing Mobile Networks with Infrastructure," in *Proc. ACM Mobicom*, San Francisco, CA, USA, September 2008, pp. 81–91.

[8] P. Basu and J. Redi, "Movement control algorithms for realization of fault-tolerant ad hoc robot networks," *IEEE Network*, vol. 18, no. 4, pp. 36–44, July-Aug. 2004.

[9] S. Bhattacharya, S. Candido, and S. Hutchinson, "Motion strategies for surveillance," in *Proceedings of Robotics: Science and Systems*, 2007.

[10] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: design, analysis and applications," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, March 2005, pp. 1653–1664.

[11] K. Briechle and U. Hanebeck, "Localization of a mobile robot using relative bearing measurements," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 36–44, February 2004.

[12] A. Cayley, "A theorem on trees," *Quart. J. Math*, vol. 23, pp. 376–378, 1889.

[13] B. Chazelle and H. Edelsbrunner, "An optimal algorithm for intersecting line segments in the plane," *J. Assoc. Comput. Mach.*, vol. 39, pp. 1–54, 1992.

[14] H. Choset and J. Burdick, "Sensor-Based Exploration: The Hierarchical Generalized Voronoi Graph," *Int. Journal of Robotics Research*, vol. 19, no. 2, pp. 96–125, 2000.

[15] V. Chvátal, "A combinatorial theorem in plane geometry," *Journal of Combinatorial Theory, Series B*, vol. 18, pp. 39–41, 1975.

[16] J. Cortes, S. Martinez, and F. Bullo, "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions," *IEEE Transactions on Automatic Control*, vol. 51, no. 8, pp. 1289–1298, Aug. 2006.

[17] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, “Coverage control for mobile sensing networks,” *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, April 2004.

[18] F. E. Daum, “New exact nonlinear filters: theory and applications,” in *Proceedings of the 1994 Signal and Data Processing of Small Targets*, O. E. Drummond, Ed., vol. 2235, no. 1. SPIE, 1994, pp. 636–649.

[19] ——, “Nonlinear filters: beyond the kalman filter,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 20, no. 8, pp. 57–69, Aug. 2005.

[20] M. C. De Gennaro and A. Jadbabaie, “Decentralized control of connectivity for multi-agent systems,” in *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006, pp. 3628–3633.

[21] O. Devillers, M. Teillaud, and M. Yvinec, “Dynamic location in an arrangement of line segments in the plane,” *Algorithms Review*, vol. 2, pp. 89–103, 1992.

[22] C. Dixon and E. Frew, “Maintaining optimal communication chains in robotic sensor networks using mobility control,” *Mobile Networks and Applications*, vol. 14, no. 3, pp. 281–291, 06 2009.

[23] J. Djugash, S. Singh, and P. Corke, “Further results with localization and mapping using range from radio,” in *Fifth Int’l Conf. on Field and Service Robotics*, 2005.

[24] S. E. Dreyfus and R. A. Wagner, “The steiner problem in graphs,” *Networks*, vol. 1, no. 3, pp. 195–207, 1971.

[25] H. Edelsbrunner, L. J. Guibas, and M. Sharir, “The complexity and construction of many faces in arrangements of lines and of segments,” *Discrete Comput. Geom.*, vol. 5, pp. 161–196, 1990.

[26] Z. Feng, Y. Hu, T. Jing, X. Hong, X. Hu, and G. Yan, “An o(nlogn) algorithm for obstacle-avoiding routing tree construction in the $\{\lambda\}$-geometry plane,” in *ISPD ’06: Proceedings of the 2006 international symposium on Physical design.* New York, NY, USA: ACM, 2006, pp. 48–55.

[27] J. Folkesson and H. I. Christensen, “Graphical slam: A self-correcting map,” in *Proceedings of the International Symposium on Autonomous Vehicles*, 2004.

[28] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, “Monte carlo localization: Efficient position estimation for mobile robots,” in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1999.

[29] M. Fürer and B. Raghavachari, “Approximating the minimum-degree steiner tree to within one of optimal,” *J. Algorithms*, vol. 17, no. 3, pp. 409–423, 1994.

[30] A. Ganguli, J. Cortes, and F. Bullo, “Visibility-based multi-agent deployment in orthogonal environments,” in *Proceedings of the 2007 American Control Conference*, July 2007, pp. 3426–3431.

[31] B. P. Gerkey, R. Mailler, and B. Morisset, “Commbots: distributed control of mobile communication relays,” in *Proc. of the AAAI workshop on auction mechanisms for robot coordination*, 2006, p. 5157.

[32] C. Godsil and G. Royle, *Algebraic Graph Theory.* New York: Springer-Verlag, 2001.

[33] D. K. Goldenberg, J. Lin, A. S. Morse, B. E. Rosen, and Y. R. Yang, “Towards mobility as a network control primitive,” in *MobiHoc ’04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing.* New York, NY, USA: ACM, 2004, pp. 163–174.

[34] H. González-Baños, C.-Y. Lee, and J.-C. Latombe, “Real-time combinatorial tracking of a target moving unpredictably among obstacles,” in *Proceedings*

of the 2002 IEEE Conference on Robotics and Automation, vol. 2, 2002, pp. 1683–1690.

[35] B. Grocholsky, S. Bayraktar, V. Kumar, C. Taylor, and G. Pappas, “Synergies in feature localization by air-ground robot teams,” in *Experimental Robotics IX*. Springer Berlin, 2006, pp. 352–361.

[36] B. Grocholsky, E. Stump, P. Shiroma, and V. Kumar, “Control for localization of targets using range-only sensors,” in *ISER ’06: Proceedings of the 2006 International Symposium on Experimental Robotics*, 2006.

[37] L. Guibas, J.-C. Latombe, S. Lavalle, D. Lin, and R. Motwani, “Visibility-based pursuit-evasion in a polygonal environment,” *Algorithms and Data Structures*, pp. 17–30, 1997.

[38] U. D. Hanebeck, “Recursive nonlinear set-theoretic estimation based on pseudo-ellipsoids,” in *Proceedings of the IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2001, pp. 159–164.

[39] U. D. Hanebeck and G. Schmidt, “Set theoretic localization of fast mobile robots using an angle measurement technique,” in *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, 1996, pp. 1387–1394.

[40] M. A. Hsieh, A. Cowley, V. Kumar, and C. Taylor, “Towards the deployment of a mobile robot network with end-to-end performance guarantees,” in *Proceedings of the 2006 IEEE Conference on Robotics and Automation*, May 2006, pp. 2085–2090.

[41] F. K. Hwang, “A linear time algorithm for full steiner trees,” *Operations Research Letters*, vol. 4, no. 5, pp. 235 – 237, 1986.

[42] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem.* North-Holland, 1992.

[43] V. Isler, S. Kannan, and S. Khanna, “Randomized pursuit-evasion in a polygonal environment,” *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 875–884, Oct. 2005.

[44] A. Jadbabaie, J. Lin, and A. Morse, “Coordination of groups of mobile autonomous agents using nearest neighbor rules,” *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, June 2003.

[45] A. Jadbabaie, N. Motee, and M. Barahona, “On the stability of the kuramoto model of coupled nonlinear oscillators,” in *Proceedings of the 2004 American Control Conference*, vol. 5, June-2 July 2004, pp. 4296–4301.

[46] J. Kahn, M. Klawe, and D. Kleitman, “Traditional galleries require fewer watchmen,” *SIAM Journal on Algebraic and Discrete Methods*, vol. 4, no. 2, pp. 194–206, 1983.

[47] G. Kantor, S. Singh, R. Peterson, D. Rus, A. Das, V. Kumar, G. Pereira, and J. Spletzer, “Distributed search and rescue with robot and sensor teams,” *Field and Service Robotics*, pp. 529–538, 2006.

[48] Y. Kim and M. Mesbahi, “On maximizing the second smallest eigenvalue of a state-dependent graph laplacian,” in *Proceedings of the 2005 American Control Conference*, 2005, pp. 99–103.

[49] K. Konolige, “Large-scale map-making,” in *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2004, pp. 457–463.

[50] K. Konolige, D. Fox, C. Ortiz, A. Agno, M. Eriksen, B. Limketkai, J. Ko, B. Morisset, D. Schulz, B. Stewart, and R. Vincent, “Centibots: Very large scale distributed robotic teams,” *Experimental Robotics IX*, pp. 131–140, 2006.

[51] L. Kou, G. Markowsky, and L. Berman, “A fast algorithm for steiner trees,” *Acta Informatica*, vol. 15, no. 2, pp. 141–145, 06 1981. [Online]. Available: http://dx.doi.org/10.1007/BF00288961

[52] J. B. Kruskal, “On the shortest spanning subtree of a graph and the traveling salesman problem,” *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[53] V. Kumar, D. Rus, and S. Singh, “Robot and sensor networks for first responders,” *IEEE Pervasive Computing*, vol. 3, no. 4, pp. 24–33, Oct.-Dec. 2004.

[54] Y. Kuramoto, *Self-entrainment of a population of coupled non-linear oscillators*, ser. Lecture Notes in Physics. Springer Berlin / Heidelberg, 1975, vol. 39, pp. 420–422.

[55] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[56] S. LaValle, H. Gonzalez-Banos, C. Becker, and J.-C. Latombe, “Motion strategies for maintaining visibility of a moving target,” in *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, 1997, pp. 731–736.

[57] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[58] D.-T. Lee, “Proximity and reachability in the plane.” Ph.D. dissertation, Champaign, IL, USA, 1978.

[59] T. Lefebvre, H. Bruyninckx, and J. De Schutter, *The Non-Minimal State Kalman Filter*, ser. Springer Tracts in Advanced Robotics. Springer Berlin / Heidelberg, 2005, ch. 5, pp. 77–94.

[60] Q. Li and D. Rus, "Sending messages to mobile users in disconnected ad-hoc wireless networks," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking.* New York, NY, USA: ACM, 2000, pp. 44–55.

[61] J. Lin, A. Morse, and B. Anderson, "The multi-agent rendezvous problem," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, vol. 2, Dec. 2003, pp. 1508–1513 Vol.2.

[62] M. Lindhé and K. Johansson, "Communication-aware trajectory tracking," in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 1519–1524.

[63] M. Lindhé, K. Johansson, and A. Bicchi, "An experimental study of exploiting multipath fading for robot communications," in *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[64] R. T. Mailler, "A mediation-based approach to cooperative, distributed problem solving," Ph.D. dissertation, 2004, director-Lesser, Victor R.

[65] M. D. Marco, A.Garulli, A. Giannitrapani, and A. Vicino, "A set theoretic aproach to dynamic robot localization and mapping," *Autonomous Robots*, vol. 16, pp. 23–47, 2004.

[66] K. Mehlhorn, "A faster approximation algorithm for the steiner problem in graphs," *Information Processing Letters*, vol. 27, pp. 125–128, 1988.

[67] Z. Melzak, "On the problem of steiner," *Canadian Math Bulletin*, vol. 4, pp. 143–148, 1961.

[68] N. Michael, M. Zavlanos, V. Kumar, and G. Pappas, "Maintaining connectivity in mobile robot networks," *Experimental Robotics*, pp. 117–126, 2009. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00196-3$_1$4

[69] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "An asynchronous complete method for distributed constraint optimization," in *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems.* New York, NY, USA: ACM, 2003, pp. 161–168.

[70] N. Moshtagh and A. Jadbabaie, "Distributed geodesic control laws for flocking of nonholonomic agents," *IEEE Transactions on Automatic Control*, vol. 52, no. 4, pp. 681–686, April 2007.

[71] K. Murphy, "Bayesian map learning in dynamic environments," in *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[72] R. Murrieta-Cid, R. Monroy, S. Hutchinson, and J.-P. Laumond, "A complexity result for the pursuit-evasion game of maintaining visibility of a moving evader," in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 2657–2664.

[73] R. Murrieta-Cid, B. Tovar, and S. Hutchinson, "A sampling-based motion planning approach to maintain visibility of unpredictable targets," *Autonomous Robots*, vol. 19, no. 3, pp. 285–300, December 2005.

[74] K. Nagatani, H. Choset, and S. Thrun, "Towards exact localization without explicit localization with the generalized voronoi graph," *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 1, pp. 342–348 vol.1, May 1998.

[75] M. H. Overmars and E. Welzl, "New methods for computing visibility graphs," in *SCG '88: Proceedings of the fourth annual symposium on Computational geometry.* New York, NY, USA: ACM, 1988, pp. 164–171.

[76] S. Petitjean, D. Kriegman, and J. Ponce, "Computing exact aspect graphs of curved objects: algebraic surfaces," *Int. Journal on Computer Vision*, vol. 9, pp. 231–255, 1992.

[77] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, vol. 36, pp. 1389–1401, 1957.

[78] J. G. Proakis, *Digital Communications.* McGraw-Hill, 1995.

[79] H. J. Prömel and A. Steger, "A new approximation algorithm for the steiner tree problem with performance ratio 5/3," *Journal of Algorithms*, vol. 36, no. 1, pp. 89–101, 2000.

[80] ——, *The Steiner Tree Problem: A Tour through Graphs, Algorithms, and Complexity.* Vieweg, 2002.

[81] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking.* New York, NY, USA: ACM, 2003, pp. 96–108.

[82] G. Robins and A. Zelikovsky, "Improved steiner tree approximation in graphs," in *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000, pp. 770–779.

[83] L. Ros, A. Sabater, and F. Thomas, "An ellipsoidal calculus based on propagation and fusion," *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 32, no. 4, pp. 430–442, August 2002.

[84] M. Schwager, F. Bullo, D. Skelly, and D. Rus, "A ladybug exploration strategy for distributed adaptive coverage control," in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 2346–2353.

[85] M. Schwager, J. McLurkin, and D. Rus, "Distributed coverage control with sensory feedback for networked robots," in *Proceedings of Robotics: Science and Systems II*, 2006, pp. 49–56.

[86] M. Schwager, D. Rus, and J.-J. Slotine, "Decentralized, Adaptive Coverage Control for Networked Robots," *Int. Journal of Robotics Research*, vol. 28, no. 3, pp. 357–375, 2009.

[87] J. Schwartz and M. Sharir, "On the piano movers problem: I. the case of a two-dimensional rigid polygonal body moving amidst polygon barriers," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 345–398, 1983.

[88] F. C. Schweppe, "Recursive state estimation: Unknown but bounded errors and system inputs," *IEEE Transactions on Automatic Control*, vol. AC-13, no. 1, pp. 22–28, February 1968.

[89] T. Simeon, S. Leroy, and J.-P. Laumond, "Path coordination for multiple mobile robots: a resolution-complete algorithm," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 1, pp. 42–49, 2002.

[90] J. Spletzer and C. Taylor, "A bounded uncertainty approach to multi-robot localization," in *Proceedings of the 2003 International Conference on Intelligent Robots and Systems*, 2003, pp. 1258–1265.

[91] J. Steiner, "Einige gesetze über die theilung der ebene und des raumes," *J. Reine Angew. Math*, vol. 1, pp. 349–364, 1826.

[92] G. L. Stuber, *Principles of Mobile Communication.* Norwell, MA, USA: Kluwer Academic Publishers, 1996.

[93] E. Stump, A. Jadbabaie, and V. Kumar, "Connectivity management in mobile robot teams," in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 1525–1530.

[94] E. Stump, V. Kumar, B. Grocholsky, and P. M. Shiroma, "Control for localization of targets using range-only sensors," *Int. Journal of Robotics Research*, vol. 28, no. 6, pp. 743–757, 2009.

[95] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region," *SIAM J. Comput.*, vol. 21, no. 5, pp. 863–888, October 1992.

[96] H. Takahashi and A. Matsuyama, "An approximate solution for the steiner problem in graphs," *Mathematica Japonica*, vol. 24, pp. 573–577, 1980.

[97] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents, part ii: dynamic topology," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, vol. 2, Dec. 2003, pp. 2016–2021 Vol.2.

[98] ——, "Flocking in fixed and switching networks," *IEEE Transactions on Automatic Control*, vol. 52, no. 5, pp. 863–868, May 2007.

[99] ——, "Stable flocking of mobile agents, part i: fixed topology," in *Proceedings of the 42nd IEEE Conference on Decision and Control*, vol. 2, Dec. 2003, pp. 2010–2015 Vol.2.

[100] O. Tekdas, W. Yang, and V. Isler, "Robotic routers: Algorithms and implementation," *Int. Journal of Robotics Research*, 2009, accepted.

[101] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics.* MIT Press, 2005.

[102] B. Tovar, L. Guilamo, and S. LaValle, "Gap navigation trees: Minimal representation for visibility-based tasks," in *Algorithmic Foundations of Robotics VI*, ser. Springer Tracts in Advanced Robotics, vol. 17. Springer Berlin / Heidelberg, 2005, pp. 425–440.

[103] J. Vleugels and M. Overmars, "Approximating generalized voronoi diagrams in any dimension," 1995.

[104] L. Wang and X. Jia, "Fixed topology steiner trees and spanning forests," *Theoretical Computer Science*, vol. 215, no. 1-2, pp. 359 – 370, 1999.

[105] E. A. Yildirim, “On the minimum volume covering ellipsoid of ellipsoids,” Dept. of Applied Mathematics and Statistics, Stony Brook University, Tech. Rep., 2005.

[106] M. M. Zavlanos and G. J. Pappas, “Controlling connectivity of dynamic graphs,” in *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005, pp. 6388–6393.

[107] ——, “Potential fields for maintaining connectivity of mobile networks,” *IEEE Transactions on Robotics*, vol. 23, no. 4, pp. 812–816, Aug. 2007.

[108] ——, “Distributed connectivity control of mobile networks,” *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1416–1428, Dec. 2008.