DISTRIBUTED MOTION PLANNING IN ROBOTIC SENSOR NETWORKS

by

Zhenwang Yao

M.A.Sc., Simon Fraser University, 2005

B.Sc., University of Science and Technology of China, 1998

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2011

APPROVAL

Name: Zhenwang Yao

Degree: Doctor of Philosophy

Title of Thesis: Distributed Motion Planning in Robotic Sensor Networks

Examining Committee: Dr. Paul Ho, Chair
Professor, Engineering Science, SFU

Dr. Kamal Gupta, Senior Supervisor
Professor, Engineering Science, SFU

Dr. Daniel Lee, Supervisor
Professor, Engineering Science, SFU

Dr. Mohammed Hafeeda, Supervisor
Professor, Computer Science, SFU

Dr. Bozena Kaminska, SFU Examiner
Professor, Engineering Science, SFU

Dr.Gaurav Sukhatme, External Examiner
Professor, Computer Science,
University of Southern California

Date Approved: 25 May 2011

Declaration of
Partial Copyright Licence

Abstract

Recently there has been tremendous interest in sensor networks for its ubiquitous applications, and in many of these applications, robots have became an integral part of the system, and therein robot mobility and network communication are two deeply coupled components. In this thesis, we investigate some interesting interplays between communication and mobility.

The first half of the thesis studies *communication-assisted motion planning* of robots, where a static sensor network deployed in the environment is used to navigate robots. We revisit some existing researches in wireless communication from the perspective of robot motion planning, and propose an effective and efficient distributed algorithm for robot navigation based on communication backbone. Toward another direction, we see the emerging trend of more sophisticated in more capable sensor networks, where sensors (such as cameras) give a spatial map rather than a single reading. We integrate the classic sampling-based planning techniques, and propose a distributed probabilistic roadmap algorithm for such applications. The proposed method is also able to deal with physical obstacles, and navigates robots through potential narrow passages, as traditional sensor networks with simple sensors are not able to.

The second half the thesis discusses *communication-constrained motion planning* of robotic sensor networks, where a team of mobile robots form a mobile sensor network, and actively maintain connectivity of the system so that robots can always communicate with each other, either directly or via other robots. We propose a novel hierarchical distributed cooperative control scheme based on communication backbone of the network: Backbone-Based Connectivity Control (BBCC). Key advantages of BBCC are that it is completely general in that it can deal with arbitrary system topologies; it is a distributed method using

only two-hop neighbor information; and finally, it has low communication cost. Furthermore, we look into potential local minimum issues that can arise because multiple objectives are considered. Our empirical observations motivate a classification of local minima based on the underlying cause, and we outline strategies to escape these minima.

To my dear wife, Vivien!

Acknowledgments

First, I would like to thank my advisor, Dr. Kamal Gupta, for the freedom and encouragement he gave me in developing my ideas, for his patience, and for all those hours he had spent with me in exploring the subject.

I would like to thank my supervisory committee Dr. Deniel Lee and Dr. Mohammed Hafeeda. Discussions with them were extremely valuable and improved my project to a great extend.

I also would like to thank all my colleagues at Robotics Algorithms and Motion Planning Lab at Simon Fraser University, for many inspiring discussions, coffee breaks, and parties.

Last, but certainly not the least, I like to thank my family. To my parents and my wife for their encouragement, support and love for all these years.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

There has been tremendous interest in sensor networks in the past decade due to their ubiquitous applications [88]. Such applications include environmental monitoring [68], human health monitoring [79], civil structure monitoring [58], intelligent transportation systems [47], battlefield surveillance [66], enemy detection [82], and many others. While sensor networks provide new capabilities for perceiving the physical world, they lack the ability to adapt and interact with it, which normally requires mobility and actuation. Bringing robots into the sensor network system can greatly alleviate such limitation and introduce new capabilities [25, 51, 97]. Mobile robots can be used to actively deploy sensors over a terrain [14, 99], to do active sampling and reduce the number of sensors [108], and to improve coverage of a sensor network [56, 75]. Furthermore, a group of mobile robots equipped with sensors and potentially actuators constitute a mobile actuator and sensor network [92], where sensor nodes can move on their own and interact with the physical environment. Such a mobile actuator and sensor network is capable of performing more sophisticated and dynamic tasks, such as emergency search and rescue [49], autonomous ocean sampling[31], chemical plume tracing and neutralizing [23], convoy protection [87], improving communication performance [94], and so on. At the same time, bringing robot mobility into sensor networks also introduce new problems and challenges in many aspects of such systems [104], including navigation and control, self-organization, etc. In my research at the Robotic Algorithms and Motion Planning (RAMP) Lab, at Simon Fraser University, I studied some of the problems for those cases where mobility is deeply coupled with communication, and

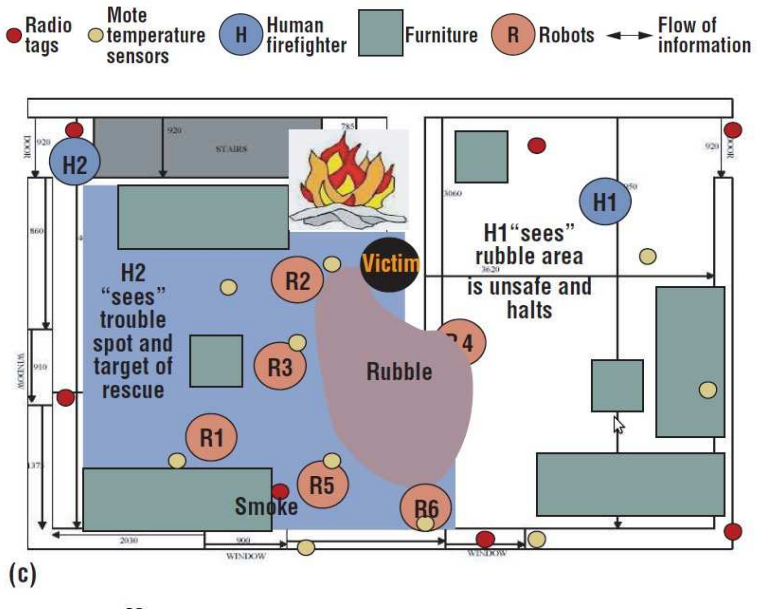


Figure 1.1: Robots and sensor network as first responders. Figure courtesy of [49].

I have been looking into the interrelationship between communication and mobility from motion planning point of view. In particular, we study two problems: *communication-guided motion planning*, where communication provides guidance to the robot motion, and *communication-constrained motion planning*, where communication imposes constraints on robot motion.

Our motivating application is to use robots and sensor network as first responders in emergency rescue, first proposed in [49]. Figure 1.1 shows such an application scenario, where the aim is to make rescuing safer and more intelligent by leveraging latest communication and robotic technologies. Such a system consists of different elements including human fire-fighters, sensor networks with various sensors for perception and navigation, and multiple robots for assistive functions and rescue tasks. These elements interact with each other and accomplish rescue tasks in a cooperative way: (i) The temperature sensor and surveillance camera sensor network can be used to safely guide human fire-fighters or rescue robots through a fire scene toward victims; (ii) Often multiple fire-fighters (or robots) need to carry a victim out in a stretcher, or move furniture in order to make room for rescue, and ideally the sensor network should also be able to provide guidance in these cooperative

tasks. (iii) There can also be cooperative tasks of larger scale among fire-fighters and rescuing robots. For example, when there are not enough sensors, a team of robots can be sent into the burning building to deploy more sensors, or thoroughly search an area for victims. During the tasks, robots/fire-fighters need to communicate with each other in a timely fashion, so they can collaborate for better decision and faster response. These cooperative tasks can be abstracted into two problems:

1. **Communication-guided motion planning.** This is for the problem of using sensor networks to guide robots (or humans) toward a goal across a hazardous environment. In such a case, robots might not have enough knowledge of an environment due to their limited sensing capability, not to mention dynamic changes in the environment. However, by communicating with a sensor network deployed in an environment to continuously perceive changes, robots can respond to events outside their perception ranges, and move with guidance obtained from the network.

2. **Communication-constrained motion planning.** This is for the problem of controlling a group of robots while maintaining connectivity among them, i.e., all robots are required to remain connected to each other (either directly, or via other robots). Such a connectedness constraint is essential in coordinated and cooperative control, since team members need to communicate and share information with each other, and more importantly, in many cases connectedness is a necessary condition for stability of the system [50, 60].

These problems, however, are challenging, as such a distributed system imposes a unique set of characteristics and constraints. The first challenge is how to tackle these problems in an efficient, responsive and scalable way. Sensor nodes normally have limited computational power and battery life, and can not afford high volume of communication and extensive movement, at the same time, we expect the system to have fast response to the dynamic environments, while the size of a system (in terms of number of sensor nodes) can be as large as thousands nodes or even more. In these cases, reliance on global information for decisions should be avoided whenever possible, as gathering global information is energy and time consuming. We are looking for distributed algorithms that primarily use local information to solve the problem, since such algorithms are in general more efficient, and have better scalability.

The second challenge is how to incorporate global objectives and constraints in a distributed framework. Whereas distributed methods that use only local information to achieve overall system objectives are intrinsically more desirable, in many of these problems, a certain degree of performance optimality is desired, for example, minimize traveling distance (i.e., shortest path); or a certain constraints need to be satisfied, besides achieving goals, for example, robots must stay connected to each other (i.e., connectivity constraint). Note, that such optimization and constraints may require global information, and may be difficult to be incorporated in a distributed framework, without inducing high communication cost.

1.2 Thesis contributions

This thesis makes key contributions for each of these two problems.

- For communication-guided motion planning problem:
  - we propose a novel communication backbone based distributed method for robot navigation amidst a wireless sensor network. The proposed method finds a safe path with less communication cost compared to existing methods.
  - We further extend our approach to use more sophisticated sensors capable of giving a detailed map for its sensing region, and propose a distributed sampling based planning framework that takes into account physical obstacles. To the best of our knowledge, ours is first such distributed algorithm that takes into account the maps obtained via sensors.
- For the problem of communication-constrained motion planning:
  - we propose a novel distributed paradigm, Backbone-Based Connectivity Control (BBCC), to deal with connectivity constraint among a team of mobile robots. Key advantages of the proposed BBCC paradigm are that it can deal with arbitrary system topologies, and it is a distributed method and uses only two-hop neighbor information.
  - The proposed BBCC above uses potential field based techniques and hence suffer from local minimum problems. To deal with the problem, we propose a preliminary somewhat empirical categorization of different types of local minima that

can arise and distributed strategies to deal with these local minima. To our best knowledge, no other distributed algorithm has attempted such categorization.

1.3 Related works

In this section we briefly review some of the research literature related to robot motion planning and sensor networks, and some existing work on combining the two.

1.3.1 Robot motion planning

Robot motion planning algorithms have been intensively studied, and comprehensive review for single robot motion planning algorithms can be found in [52]. In the last decade, probabilistic sampling-based methods have been shown to be effective in solving many difficult motion planning problems. The idea behind these methods is to construct a connectivity roadmap in the configuration space (C-space) by randomly placing landmarks (configurations) into the C-space, and trying to set up the connections between neighboring landmarks. Probabilistic Roadmap Method (*PRM*) [46], Rapidly-exploring Random Tree (*RRT*) [53, 48] are most used sampling based algorithms.

These sampling based motion planning algorithms have inspired a few algorithms for the robot navigation problem in sensor network [3, 15, 20, 4], including ours [100, 103]. For navigation and path planning in sensor networks, the key challenge is how to do the path planning in a distributed way [20], since there is no central global representation of the environment, and the perception of the environment comes from distributed sensors. Note that there is another thread of research studying distributed motion planning from the perspective of parallelism [40, 5, 67, 89], which focuses on distributing computation into different processors. All these works, however, assume processors have access to either a shared or a private copy of a global C-space representation, and therefore address a different problem.

1.3.2 Wireless sensor networks

Good reviews of wireless sensor networks research in general can be found in [104, 1]. Akyidiz et al. [1] gave a thorough review on sensor network applications, design factors, and communication architecture. A more recent review [104] by Yick et al. focused on

recent development, and touch on broader issues in wireless sensor networks. Both surveys identified many open research problems in various aspects of the field. Here, we only briefly review some closely related topics.

Data-centric routing algorithms

Many parts of our research are closely related to topology control and routing in wireless sensor networks, which is very a challenging problem due to the inherent characteristics of sensor networks [2], such as its relatively large network size, constrained energy, and application dependence design. Traditional IP-based host-centric routing techniques are not suitable for wireless sensor networks. Instead, data centric routing is more effective and efficient, where nodes are not addressed by their addresses but by the data they sense. Directed diffusion [44] is an important data-centric and application-aware routing and data-aggregation scheme. The basic idea of Directed Diffusion is that, instead of flooding raw data, it propagates an interest (e.g. maximum temperature in the network) through the network in a hop by hop fashion, and each sensor node receives and aggregates the interest and sets up a gradient toward the sensor nodes from which it receives the interest. By building the gradient, a shortest-path tree is created for further data aggregation and avoid flooding. Directed Diffusion has been studied extensively with many variants and extensions [16, 78, 80], and shown to be a very effective technique to reduce energy consumption, and achieve scalability.

Clustering algorithms and communication backbone

Another technique for a similar purpose is clustering for hierarchical routing for topology management. A flat network can waste energy and cause latency in communication and tracking events, and hence it can result in poor scalability. Clustering is a technique to scale down networks with a large number of nodes by creates an hierarchy for the network. The basic idea is to group a set of nodes based on their physical proximity, and represent each group with a single node as *clusterhead*. Good review of clustering algorithms can be found in [22, 42, 96, 12].

One of the first clustering algorithms for sensor network is LEACH algorithm proposed by Heinzelman et al. [39], which randomly rotates clusterheads. However, LEACH assumes single-hop communication from clusterheads to the sink node (base station), and requires

strict time synchronization between clusterheads and nodes within a local cluster. A more general technique for clustering algorithms is to use graph domination and its variants. A *dominating set*, D, is a set of vertices that makes all vertices of the graph either in D or adjacent to at least one vertex in D. Formally,

$$D \subseteq V, \quad \forall u \in V - D, \quad \exists v \in D \quad \text{s.t. } (u, v) \in E.$$

The members of a *dominating set* (DS) can be used to represent clusterheads, each of which forms a cluster with their neighbors. A *connected dominating set* (CDS), $C \subseteq V$, is a dominating set of G, such that the subgraph induced by C is connected. A CDS in general includes a set of clusterheads as in the dominating set, and gateways that connect them. Thus connected dominating set can be used as *communication backbone* [26], which is widely used in communication to reduce broadcast redundancy and thereby energy consumption [83]. Unfortunately, the problems of finding a *minimum dominating set* (MDS), and *minimum connected dominating set* (MCDS) have been proved to be NP-hard [34]. Due to the hardness of the domination problems, different algorithms use different heuristics to choose dominating sets as clusterheads (and gateways), and such heuristics can be based on node ID [55], degree [36], mobility [11]. In this thesis, we adopt the TMPO (Topology Management by Priority Ordering) algorithm by Bao and Garcia-Luna-Aceves [10], which choose MCS/MCDS locally based on a comprehensive heuristic combining multiple criteria.

Localization

In our research, we assume the sensor network is well localized, and a mobile robot can be localized with assistant of the network. There are many research in localization algorithms for sensor networks [59, 9, 17], with or without GPS. These algorithms covers indoor [41] and outdoor [19] environments, single-hop [37] or multi-hop [61] networks, and use different distance measurements, such as radio signal strength (RSS) [41], angle-of-arrival (AOA) [62], time-difference-of arrival (TDOA) [76], or simply hop counts [61]. Most of them assume the existence of anchor nodes that already know their locations, with a few being anchor-free [37] and mobile robots have been used as mobile beacons for localization [29, 33, 81, 70]. There are also localization algorithms for mobile sensor networks, based on the Monte Carlo localization method [90].

1.3.3 Robot navigation with static sensor networks

Several distributed algorithms have been proposed for the problem of navigating an individual robot using sensor networks, and most of these algorithms are based on Directed Diffusion. One class of the algorithms propagates a navigation field over the entire sensor network: messages flood from the goal, so that each sensor node will have knowledge about best movement to reach goal. The algorithms in [13, 54, 93, 63] fall into this category, and they differ in the definition of navigation field based on different objectives. All the above algorithms, however, use flooding to propagate a navigation field, which is not efficient in terms of network energy consumption due to high communication volume. Recently, inspired by PRM, roadmap based methods have been proposed to reduce flooding and hence communication cost [3, 15, 20, 4]. Rather than propagating the navigation field over the entire network, these methods navigate the robot through a roadmap, a smaller subset of the network. To be specific, Alankus et al. [3] and Bhattacharya et al. [15] proposed to find a feasible path incrementally as the robot travels along the roadmap, and flooding is reduced by limiting query to only nodes in vicinity of the robot; Bhattacharya et al. [15] further reduced communication by building a virtual grid road-map in the area, and limiting query to nodes close to roadmap edges. Buragohain et al. of [20] proposed to scale down the original network by building a skeleton graph based on geographic information. More recent by Alankus et al [4] proposed to build a roadmap by randomly choosing a certain number of nodes as milestones, and making connections among them. However, all these methods still need a certain degree of flooding for all nodes in roadmap construction, and for non-roadmap nodes to compute the navigation field to guide the robot toward the roadmap; therefore they do not take full advantage of the roadmap.

Besides, all existing works for the problem assume only simple sensors, such as temperature sensors, are available, and hence the entire sensing region of a sensor is either free or in danger (so we call such sensing model as "binary" model), and they do not take into account physical obstacles in the planning phase, and leave physical obstacle avoidance to the execution phase and require replanning when the robot detects an obstacle in the way [4]. This may result in quite lengthy and costly paths. In addition, the binary sensing model is too simple for more sophisticated applications, especially in the context of visual sensor network [84], or, for example, in emergency rescue applications, where a networked of mobile robots are equipped with laser scanners.

1.3.4 Connectivity control for multirobot formation

There is extensive literature for multi-robot systems, and good reviews can be found in [21, 7], and a recent book by Bullo et al. [18]. Connectivity constraints have been considered in the context of different problems, such as cooperative exploration [91, 73], and relay communication [77]. More recent works have addressed connectivity constraints in path and motion planning of networked systems. Among these works, algorithms in [65, 24, 30, 32] assume each robot has its own goal configuration to achieve, and those in [85, 45, 106, 107] deal with more general objectives (e.g., consensus, or formation control) on top of connectivity constraints. Some of the algorithms are centralized [24, 32, 106] and suffer from high dimensionality of the problem (i.e., high dimensionality of the composite state space) for large systems and may not be suitable for real time applications. Distributed methods that use only local information are intrinsically more desirable. However, as pointed out in [86], it is difficult to embed connectivity constraints into geometric and analytical models typically used in distributed motion control or planning algorithms, due to the combinatorial and global nature of connectivity constraints.

Due to difficulty of the problem, many works make simplifications. Some [65, 30] simplified the problem by assuming a predefined and fixed topology of system (e.g., "constraint graph"), and others [45, 27] assumed either the goal formation to be a subgraph of the initial formation, or vice versa. Clearly, fixed topology does not capture full dynamics of multi-robot systems, since a system topology changes as robots move in and out of communication range of each other. Such presumptions on formation topology are quite limiting, are not realistic in many applications, and may prevent certain tasks from being achieved. The authors in [85, 107] considered more general solutions. They are similar in that they both use certain sub-graphs to represent system topologies, and guarantee connectivity by maintaining existing links in the representative sub-graphs. However, these approaches have drawbacks. The algorithm in [85] uses information flow to represent the system topology, and updates the information flow based on local connectivity robustness. The resulting information flow is a non-expanding topology: once a robot is within in two hops of another robot, the two robots are not allowed to be further apart in subsequent stages. This is still limiting and may not be suitable for a general task. The algorithm in [107] uses spanning subgraphs to capture the system topology, and determining creation/deletion of a connection is done by auction, which may involve all robots of the system and hence induce high

communication cost.

1.4 Thesis overview

The first half of the thesis studies *communication-assisted motion planning* of robots, where a static sensor network deployed in the environment is used to navigate robots. We start with circular robots, and propose a backbone based roadmap algorithm; then we investigate navigation with more general robots and more sophisticated sensing model in sensor network, and look into methods doing finer-grain path planning for robots with non-trivial size and shapes. The second half the thesis discusses *communication-constrained motion planning* of mobile sensor networks, where a team of mobile robots form a mobile sensor network, and actively maintain connectivity of the system throughout a task. We propose a general and efficient Backbone-based cooperative control scheme for the problem, and also look into local minimum issues that arise with the problem.

1.4.1 Backbone based roadmap for robot navigation

We propose a backbone based roadmap framework for robot navigation. This is inspired by distributed clustering algorithms in sensor networks for constructing communication backbone to eliminates unnecessary flooding. For robot navigation problem, we first extract the backbone of the sensor network via a clustering algorithm adapted from TMPO algorithm [10], and use the backbone as roadmap for path planning. It is advantageous to use backbone network as a roadmap: There is no need of flooding in order to construct the backbone; A node decides whether to become a backbone node, based on its 2-hop neighbor information. More importantly, backbone systematically captures the system connectivity, and some of its properties make it desirable in our motion planning problems: (i) Backbone preserves connectedness. The backbone has same number of connected components as the original network. (ii) Backbone provides a hierarchical representation of the system topology and scales down the original network. The size of backbone depends on network connectivity, and when connectivity is high, the number of clusterheads decreases, resulting in a smaller backbone. Hence, good scalability is promised. (iii) The backbone spreads out in the network, such that every network node is at most 1-hop away from a clusterhead (and hence backbone). As shown later in Section 2.5, this property provides performance guarantee (in terms of robot traveling distance).

1.4.2 Distributed roadmaps for navigation using spatial sensors

As mentioned earlier, most existing works for robot navigation in sensor networks assume point robot and binary sensing model. We propose a distributed sampling based framework for sensor networks whose sensors are equipped with more sophisticated senors, for example, cameras in the context of visual sensor networks. These sensors provide much richer information, e.g., a spatial map rather than a single reading. At the same time, the navigation tasks in this case are usually more complicated. For instance, in the emergency rescue application, collapsed walls may block some areas, and a feasible path may run through a narrow passage of some sort lying across sensed regions of multiple sensors. In order to effectively navigate a rescue robot or a fire-fighter through the rubble, these multiple sensors need to cooperate with each other, take into account obstacles when planning paths, and thereby plan a feasible and more efficient path.

The proposed distributed sampling based planning algorithms, *Distributed PRM* (D-PRM) and *Distributed RRT* (D-RRT), systematically incorporate a general spatial sensing model for each sensor, and take into account obstacles in determining feasible paths. Each sensor creates a local roadmap (a patch) similar to the classic PRM or RRT, but only for its locally-sensed environment. Two different patches of roadmap are "stitched" together with a set of *relay points* lying in the common region shared by the two patches. Sensor nodes mutually negotiate the connectivity of their patches by sending messages regarding the status of their respective relay points. When two adjacent sensor nodes see a relay point free, it becomes a connecting point for the two patches. To find the shortest path on distributed roadmaps, a distributed navigation field is created across the sensor network, which maps each sample in local roadmaps into distance to the desired goal, and the best path is computed by gradient descent. The proposed algorithms are general and applies to robots with non-trivial shapes, and even for formations with multiple robots. To the best of our knowledge, this is the first work to study the distributed path planning in sensor networks with complex spatial sensing capability.

1.4.3 Backbone based connectivity control for mobile robots

We propose a Backbone Based Connectivity Control (BBCC) scheme for a team of mobile robots, where system topology changes when mobile robots move. In order to maintain connectivity among robots, such that all robots can communicate with one another either

directly or via other other robots, the proposed BBCC scheme uses communication backbone to represent the dynamic topology of the system. Backbone of the mobile robot network is updated in real time to capture the dynamic topology of the system and to impose motion constraints on robots so that network connectivity is maintained. To be more specific, BBCC maintains the system connectivity with a two-levels hierarchy: it first maintains a connected backbone, by maintaining existing connections (communication links) in the backbone; and then for a non-backbone robot, one of the backbone robots is chosen as leader, and connection to the leader is maintained. The overall philosophy is similar to [85, 107] in that they all use certain sub-graphs to represent system topologies, and guarantee the connectivity by maintaining existing links in the representative sub-graphs.

BBCC has several advantages compared to existing methods: (i) BBCC makes no assumption on system topology, and can deal with arbitrary initial and goal formations. Because communication backbone is essentially a connected dominating set, it captures the system connectivity nicely, and provides effective representation of the system connectivity. (ii) BBCC is a distributed scheme and does not require global message exchange. Backbone can be constructed and updated in a distributed fashion, using only two-hop neighbor information.

1.4.4 Local minimum escape scheme for connectivity control

Similar to most existing works for connectivity control, BBCC uses a potential field based technique to maintain critical links, and suffers from local minima problem when multiple criteria are considered, such as achieving goal, maintaining connectivity, and avoiding collisions. Furthermore, as mentioned in [86], in some scenarios, using only local information is doomed to failure and global decision needs to be made in order to achieve a certain task. We extend BBCC framework to a general motion planning framework that is capable of escaping from local minima, and making global decisions when necessary. To the best of our knowledge, our scheme is the first distributed approach to attack the local minimum problem in mobile networks.

In the proposed scheme, local minima are detected when one or more robots do not progress toward their goals. We classify local minima into three different categories: Type-I (Regional obstacle-induced local minimum), Type-II (Individual connectivity-induced local minimum), and Type-III (Structural compound local minimum). Different types imply different natures and causes of the minima, and need different escaping strategies. In the first

category, the robot may be able to escape the minimum merely by simple local behavior (e.g., we use Random Walk strategy), whereas in latter two categories, a robot needs help from others in order to make global decisions for local minimum escaping, and we use Backbone-based Navigation, and Backbone-based Leader-following respectively. These two strategies incorporate distributed global decision making and exploit existing backbone constructed under BBCC toward this purpose. Backbone based Navigation strategy uses the backbone to take advantage of the knowledge (sensing) embedded in the entire network system, gathers path planning information (roadmap) that is beyond sensing and communication range of one single robot, and provides guidance to robots to escape Type-II local minimum. Backbone based *Leader-following* strategy tries to achieve maximum mobility by reducing the number of connectivity constraints, and looks for maximum reconfigurability in order to escape Type-III local minimum.

1.5 Thesis organization

The remainder of the thesis is organized as follows. In Chapter 2, we present the backbone based roadmap for a single circular robot navigation in static sensor networks. In Chapter 3, we investigate the robot navigation problem with more sophisticated sensing model and more general robots. In Chapter 4 and 5, we study the problem of controlling a team of mobile robots with connectivity constraints. The Backbone-based Connectivity Control scheme is presented in Chapter 4, and in Chapter 5, we discuss the local minimum issues that arise in the problem. Finally, conclusion and future work is presented in Chapter 6.

Chapter 2

Backbone-based Roadmaps for Robot Navigation

2.1 Overview

In this chapter, we study the robot navigation problem in a static sensor network. Consider a sensor network consists of a set of sensor nodes, $\mathcal{S} = \{s_1, \cdots, s_n\}$, in the environment. A sensor node can measure state of the environment (e.g., temperature) within its sensing range, d_s. Danger areas (e.g., with excessive heat) can be detected by sensors, if the sensor reading is beyond a certain threshold. We assume that sensor nodes know their location, $\{x_1, \cdots, x_n\}$, and a simple unit disk model [72]. Namely, two nodes can communicate with each other, if they are within distance, d_c, the communication range. The network formed by sensor nodes is modeled as a proximity graph, $G(V, E)$, whose vertices, $V = \{1, \cdots, i, \cdots, n\}$, represent sensor nodes, and an edge (i, j) represents the communication link between nodes i and j. The set of neighbors of node i is denoted by $N(i) = \big\{j \big| \, \|x_j - x_i\| \leq d_c\big\}$.

The robot, $\mathcal{A}$, is assumed to be a circular robot. It is mounted with sensing and wireless communication devices that can communicate with sensor nodes within communication range, d_c. The robot responds to a certain event (e.g., a victim in a rescue scene [49]). When a sensor node s_g in the network detects a target event, the sensor network navigates the robot $\mathcal{A}$ toward s_g while avoiding danger areas (e.g. fire).

The challenges of the problem come from the fact that (i) the robot has very limited

sensing range, and does not have global knowledge of the environment, and (ii) the environment may change over time. With the sensor network, the problem can be solved in two steps. (1) **Path planning**: this is done within sensor network alone to find a feasible path in terms of sensor nodes; (2) **Path navigation**: with continuous interaction between the sensor network and the robot, the sensor network navigates the robot along the feasible path by moving the robot from one sensor node to another. Most existing methods for the problem reviewed in Chapter 1 adopt similar scheme. But these methods require flooding among sensor nodes, and result in high communication cost. In this chapter, we propose a different roadmap method, which eliminates flooding in the path planning phase.

The proposed method is inspired by distributed clustering algorithms for constructing communication backbone to eliminates unnecessary flooding. The proposed backbone-based roadmap works as follows. It first extracts the communication backbone of the static sensor network, and uses the backbone as roadmap for path planning. The overall scheme is shown in Figure 2.1. The main activities (shown in ovals) of the scheme include backbone-based roadmap construction and update, path planning, and path execution.

1. **Backbone construction and update.** The sensor network constructs the communication backbone when the system starts, and keeps updating the backbone when nodes detect dangers or their energy levels drop to a certain level.

2. **Path planning.** In response to a certain event, the robot needs to move to a certain *goal*. Then path planning is initiated (by the goal), and uses the constructed roadmap to find a safe path (a communication route).

3. **Path execution.** Then robot follows the found path from one node to another, by continuously communicating and interacting with the network.

Since the backbone of the sensor network is computed via a clustering algorithm adapted from TMPO algorithm (Topology Management with Priority Ordering) [10], we first briefly review the original TMPO algorithm in Section 2.2, before we present the proposed algorithm in Section 2.3. Various extensions are discussed in Section 2.4, with some theoretical results in Section 2.5. In Section 2.6 we show simulation results, followed by summary in Section 2.7.

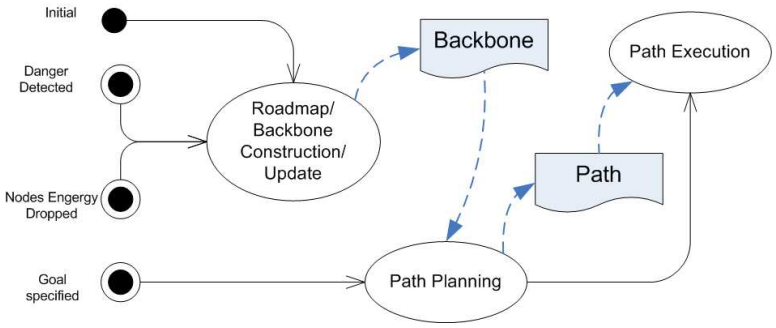


Figure 2.1: Robot navigation scheme in static sensor networks. The oval shapes represent main activities, and the rectangles with curved bottom represent data of interest.

2.2 Background: TMPO

2.2.1 Backbone construction

TMPO computes a *dominating set* (DS) as *clusterheads*, and then choose *gateways* and *doorways* (a special type of gateways) to connect clusterheads. Clusterheads, gateways, and doorways form a *connected dominating set* (CDS), the communication backbone. With TMPO, a node decides whether or not to be a CDS member based on a knowledge of its 2-hop neighbors, and their priority, which is a function of node energy and mobility. The criteria are as follows, and note that these criteria are stated with respect to local 2-hop neighbor information.

A node becomes a clusterhead (**CH**), if it satisfies *either* of the following conditions:

(a) (C.1) (b) (C.2)

Figure 2.2: Clusterhead election.

(C.1) It has the highest priority among its 1-hop neighbor, as shown in Figure 2.2(a);

(C.2) It has the highest priority among some node's 1-hop neighbors, as shown in Figure 2.2(b);

It has been proved that clusterheads elected based on (C.1) or (C.2) make a dominating set. Furthermore, for any clusterhead, the closest clusterhead (if there exists one) is at most 3-hop away. To form the backbone, connections between clusterheads need to be established to make them connected. If two clusterheads are only 1-hop away, the link between them is kept. If two clusterheads are 2-hop away, and there is no other clusterhead in between, a *gateway* is needed to connect them. If two clusterheads are 3-hop away, and there is no other clusterhead in between, a *doorway* is needed to bring them one-hop closer, and a *gateway* is needed to connect the doorway and the other clusterhead. Simply put, the shortest path (of length 3) with the highest priority node is used to connect these two clusterheads, and the node (with the highest priority) becomes *doorway* and connects to one of the clusterheads, and a common neighbor (with the highest priority) of the doorway and the other clusterhead is elected as a gateway to connect the doorway and the other clusterhead. More specifically, a node becomes a doorway (**DW**), if it satisfies *all* of the following conditions:

(D.1) It has one clusterhead, c_1, as 1-hop neighbor.

(D.2) It has another clusterhead, c_2, as 2-hop neighbor, but no other clusterhead neighboring c_2.

(D.3) c_1 and c_2 are not neighbors, and there is no other nodes connecting c_1 and c_2;

(D.4) There is no other path between c_1 and c_2 that has a higher priority node.

A node becomes a gateway (**GW**), if it satisfies *all* of the following conditions:

(G.1) It has two disjoint clusterheads, or one clusterhead and one doorway, n_1 and n_2 as 1-hop neighbors;

(G.1) There is no 1-hop neighbor that is a common neighbor of n_1 and n_2, and has higher priority.

2.2.2 Performance

The main objectives to evaluate a backbone construction algorithm include: size of constructed backbone, message overhead, and time taken to construct and update the backbone, and maintenance of the backbone. In practice it is difficult to achieve these objectives at the same time. The best approximation algorithm, proposed in [95], gives a constant

approximation ratio 8 with respect to minimum backbone size, which is by far the best approximation ratio for the problem, and it also achieves message complexity of $O(n \log n)$, which has been shown as the message complexity lower-bound for non-trivial[1] backbone construction. The backbone construction therein requires multiple negotiation phases: it first computes a spanning tree, and then extracts the backbone based on the computed spanning tree. As a consequence, constructing and updating the spanning tree requires global consensus, and hence demands $O(n)$ time and $(O \log n)$ messages for construction and each update.

TMPO, on the other hand, is not an approximation algorithm in the sense that it does not guarantee an approximation ratio with respect to the optimal backbone size, and the constructed backbone may it may contain cycles. However, the empirical study in [10] shows that TMPO performs well in practice. In fact, the size of backbone is not the most important objective in our application. Instead, the more important objective is to keep the message overhead associated with creating and maintaining the backbone low, to effectively take into account multiple factors such as safety and overall longevity of the network. TMPO does not requires multiple global negotiation phases, and it constructs and updates solely based on 2-hop neighbor information. As shown in Section 2.5, TMPO achieve message complexity of $O(|CDS| \cdot \Delta)$, where $|CDS|$ is the size of backbone and Δ is the maximum degree of the network. Furthermore, backbone construction in TMPO is based on a comprehensive definition of priority, which can be easily adapted to take into account safety and energy consumption factors in our application.

2.3 Backbone for navigation

2.3.1 Node priority for navigation

In [10], priority is defined as a function of energy and mobility, and a node that has higher energy and lower mobility is more likely to be a clusterhead, thereby achieving longer lifetime, and more stable backbone. We adapt and generalize this definition for robot navigation. As we consider static sensor networks in this problem, we remove the mobility factor, but we take into account the distance to dangers for path safety. We define the priority P_i, of

[1] A connected dominating set is said to be trivial if it consists all nodes. For example the CDS for a ring graph is trivial because it contains all nodes.

node i, as a function of energy and safety:

$$\begin{aligned} P_i(d_o, E) &= B_1 \bigoplus B_2 \bigoplus B_3 \\ B_1 &= \lfloor d_o \cdot \log^2(1 - 0.9E) \rfloor \\ B_2 &= d_o \\ B_3 &= d_o \cdot node\_id \end{aligned} \tag{2.1}$$

where B_i is a bit-string and $\bigoplus$ is bit-concatenation operation, d_o is the distance to danger, and $E \in [0, 1]$ is remaining energy. The $\bigoplus$ defines different priority for the three terms, B_1 is in the most significant bits therefore has highest priority. As $d_o \geq 0$, the priority will be non-negative, and when $d_o = 0$, $P_i(d_o, E) \equiv 0$. When the battery is depleted ($E = 0$), the logarithmic term goes to zero, and B_1 becomes zero.

2.3.2 Roadmap construction

The backbone construction procedure essentially elects members of CDS, as described in Section 2.2. We make two key modifications to the original TMPO algorithm, since the navigation problem imposes safety constraints. First, with the priority defined in Eq.(2.1), nodes that are further away from danger, and have more energy are more likely to be elected in the CDS. In order to eradicate the possibility of electing a node in danger, which has zero priority, we include an extra criterion for CDS election:

(E.1) A zero-priority node is not eligible to be a CDS node.

Furthermore, during the election, a node simply ignores a neighbor, if this neighbor has zero-priority, as if the neighbor were not in the list of neighbors. Equivalently, the election is done with respect to the network with all nodes in danger removed. Note that inclusion of (E.1) may result in a disconnected backbone, even if the original network is a connected one. In such a case, the backbone will be the union of CDS for each connected component. For the sake of easy description, we may interchange the terms of *CDS* and *backbone* throughout the dissertation.

The second modification is regarding information propagation after election. In the original TMPO algorithm, clusterheads and doorways need to propagate their type (CH, DW, or GW) information, because election of doorways depends on which nodes are clusterheads in neighborhood, and election of gateways depends on information of clusterheads and doorways. In our problem, in order to (further) reduce the communication volume for

goal dissemination in later stages, we propose to propagate extra information for doorways and gateways. That is, besides type information, a doorway or gateway should also propagate the information about which clusterheads (or doorways) it connects to. In this way, backbone nodes are connection-aware, and only handle messages from directly connected nodes.

The backbone roadmap is constructed by the distributed algorithm detailed in Algorithm 2.1, and the same algorithm runs in every sensor node. When CDS nodes are elected, connections among them are formed implicitly by constructing **employers** and **employees**. For a doorway, **employers** is a list of clusterheads it connects, for a gateway **employers** is a list of clusterheads and doorways it connects, and a clusterhead has an empty set of **employers**. Once a node changes its type (e.g., newly elected as a clusterhead), it propagates its type by broadcasting a *TYPE_CHG* message for two hops. *TYPE_CHG* message includes **employers**, and upon reception of the message, clusterheads and doorways update their **employees** by including all doorways and gateways that connect them to another clusterhead. Clearly, a gateway has an empty set of **employees**. **employers** and **employees** will be used in the next stage for goal dissemination for navigation.

2.3.3 Goal dissemination

The purpose of goal dissemination is to notify every node of the specified goal, so that every node can provide guidance (to the robot) when the robot is in the neighborhood. Now that we have constructed the backbone as a roadmap, the goal dissemination propagates a potential field over the network via the roadmap, and the best path is found by following the field. The definition of path quality depends on applications, and different potential functions can be used, e.g., [4] uses a weighted combination of path length and maximum danger level. While nothing prevents one from adopting other potential functions, here we simply choose the shortest path in the roadmap, as we have already taken safety into consideration when constructing the roadmap, and all nodes in the roadmap are in safe areas. The goal node initiates the goal dissemination procedure by broadcasting a *GOAL* message, and backbone nodes forward the message to every node of the network. Algorithm 2.2 shows how a sensor node (including non-backbone nodes) handles the *GOAL* message, and update related information: current distance to goal (**hopstogoal**), and best movement toward goal (**nexttogoal**). If the received message gives a better path to goal, a node updates the information, and the message is re-broadcast only if the receiving node is a backbone node

Algorithm 2.1: Roadmap construction

```
1  employers ← ∅; type ← Regular;
2  if ((C.1) or (C.2)) and (E.1) then
3  |  type ← CH;
4  else
5  |  if (D.1-4) and (E.1) then
6  |  |  type ← DW;
7  |  |  employers ← CHs that it connects;
8  |  endif
9  |  if (G.1-2) and (E.1) then
10 |  |  type ← GW;
11 |  |  employers ← CHs and DWs that it connects;
12 |  endif
13 endif
14 if type changes then
15 |  omsg.msgid ← TYPE_CHG;
16 |  omsg.content.type ← type;
17 |  omsg.content.param ← employers;
18 |  Broadcast omsg;
19 endif
20 if imsg received and imsg.msgid=TYPE_CHG then
21 |  if from 1-hop neighbor then Broadcast imsg;
22 |  if (type = CH or DW) then Update employees;
23 endif
```

Algorithm 2.2: Goal Dissemination

```
1  forme ←false;    //Is the message for me?
2  if imsg received and imsg.msgid=GOAL then
3  |  if (type is CH or GW or DW) then
4  |  |  if sender ∈ employees then forme ←true;
5  |  |  else if sender ∈ employers then forme ←true;
6  |  else
7  |  |  forme ←true;
8  |  endif
9  |  if forme = true and imsg.hops+1<hopstogoal then
10 |  |  imsg.hops ← hopstogoal ← imsg.hops+1;
11 |  |  nexttogoal ← imsg.sender;
12 |  |  if type ≠ Regular then Broadcast imsg;
13 |  endif
14 endif
```

(Line 9-13). The number of messages is reduced by limiting rebroadcasting (forwarding): A backbone node only processes messages from its **employers**, **employees**, and the goal node; in these cases a flag **forme** is set to indicate a valid message as in Line 2-8. If a message comes from any other nodes, the receiving node simply discards the message; A regular node receives and processes *GOAL* messages but never forwards the message.

2.3.4 Robot navigation

After the procedure above, a potential field is computed over the network via the roadmap, with **nexttogoal** pointing to goal in each sensor node. During execution, with the cooperation of sensor nodes, the robot finds the best path by following the field. The procedures on both sensor network and robot sides are shown in Algorithms 2.3 and 2.4. When a robot moves amidst the sensor network, it constantly broadcasts a *QUERY* message, and waits for response from sensor nodes in the neighborhood. When sensor nodes receive *QUERY* message, they respond with *NAVIG* messages which contain **hopstogoal** and **nexttogoal**. This information for every node has been updated in the goal dissemination stage. When receiving *NAVIG* message, the robot chooses the best movement (i.e., smallest number of hops to goals). After execution, the query-respond-move procedure repeats again, until the goal is reached.

Remarks In Algorithm 2.2, the shortest path is computed based on hop-count distance model. Li and Rus [54] showed that the hop-count distance between two sensor nodes is well related to their Euclidian distance, especially when the network density is high. The Euclidian distance of a k-hop path has the expectation of kE, and the deviation of $\sqrt{k}d$, where E and d is the expectation and the deviation of the length of a hop. So it is appropriate to navigate the robot based on the shortest path computed in Algorithm 2.2.

2.4 Other considerations

2.4.1 Dynamic environments

Algorithm 2.5 shows how the roadmap adapts to dynamic dangers. In Line 1- 6, when a sensor node detects danger, d_o becomes 0, and it broadcasts a *DANGER* message. In Line 7-12, nodes receive the message, update their value of d_o, and forward the message up to a certain distance, d_{max}. Upon change of d_o, node priority changes, and re-election of CDS

Algorithm 2.3: Navigation (in robot)

```
1  repeat
2      hopstogoal ← ∞;
3      Broadcast QUERY message;
4      while not timeout do
5          if imsg received and imsg.msgid=NAVIG then
6              if (imsg.hopstogoal < hopstogoal) then
7                  hopstogoal ← imsg.hopstogoal;
8                  nexttogoal ← imsg.nextogoal;
9              endif
10         endif
11     endw
12     if hopstogoal ≠ ∞ then
13         Robot moves toward nexttogoal;
14     else
15         Robot makes random movement;
16     endif
17 until hopstogoal =0 ;
```

Algorithm 2.4: Navigation (in nodes)

```
1 if imsg received and imsg.msgid=QUERY then
2     omsg.hopstogoal = hopstogoal;
3     omsg.nexttogoal = nexttogoal;
4     Broadcast omsg;
5 endif
```

is done locally again to adjust the roadmap for the changed environment (Line 13). When some nodes in backbone have changed, the navigation field over the previous backbone needs update. As local changes in backbone may result in global changes in the navigation field, rather than updating the field locally, a *STALE* message is sent to the goal node, and another round of goal dissemination will be initiated (Line 14).

Remarks (i) Danger detection may involve non-backbone nodes, as a non-backbone node can also detect dangers. (ii) The propagation of danger information may need a small degree of flooding depending on the value of d_{max}. However, note that there is no flooding needed in backbone election/re-election (i.e., roadmap construction), where all messages are sent at most 2 hops.

Algorithm 2.5: Roadmap Maintenance

```
   /* Upon sensor reading changes.                                          */
 1 if reading > T then
 2 |  d_o ← 0;
 3 |  omsg.msgid = DANGER;
 4 |  omsg.danger.dist = 0;
 5 |  Broadcast omsg;
 6 endif
   /* Upon receiption of DANGER msg.                                        */
 7 if imsg received and imsg.msgid=DANGER then
 8 |  if imsg.danger.dist+1< d_o < d_max then
 9 |  |  imsg.danger.dist← d_o ←imsg.danger.dist+1;
10 |  |  Broadcast imsg;
11 |  endif
12 endif
   /* Wait for a certain period, then:                                      */
13 Call procedure in Algorithm 2.1;
14 if type changes then Send a STALE message to goal node;
```

2.4.2 Load balance: network longevity

Generally backbone nodes should be kept alive for navigation, and hence they consume more energy. To achieve overall longevity of the sensor network, nodes in the network should share their roles as backbone nodes. With the definition of priority in Eq.(2.1), this can be easily achieved. Remaining energy is represented by a set of discrete levels. When the energy drops to the next level, a node broadcasts an *ENERGY* message to notify its neighbors

within two hops. Upon reception of the message, its neighbors recompute the priority and reelect backbone if needed.

2.4.3 Implementation issues

Synchronization For simplicity of explanation in description and theoretic analysis, we assume synchronous communication. In our implementation, asynchronous mechanism is used to avoid congestion and synchronous sudden loss of the old network states (e.g., if all neighboring sensor nodes decide to compute backbone at the same time). As in [10], we use a simple random time slot offset for a node to uniformly distribute the local backbone re-election and communication over the time horizon.

Robustness In our problem, sensor nodes are static, and we do not globally update the backbone topology periodically as in [10], since such updates can be costly. Instead, we only update backbone locally when it is indeed necessary: dangers are detected, or a node's battery drops to a certain level. In realistic scenarios, the backbone may not always reflect the actual environment. For instance, a node can be burnt before it has a chance to send out a *DANGER* message, or imperfect communication may result in inconsistent neighbor information. This needs to be taken into consideration, in order to make the proposed algorithm robust enough to implement on a real system. What we do is somewhat similar to the strategy in [4]. We verify the path on the execution phase: before the robot moves, it confirms (with current associated sensor node) that the next sensor node is indeed alive and safe. If otherwise is indicated, a local CDS re-election is initiated, and the robot waits until the backbone is updated and a new movement is given.

2.5 Theoretical results

We define a subset $V_{bad} \subseteq V$ as nodes in dangerous regions, $V_{good} = V - V_{bad}$ as nodes in safe regions, and V_{rdmp} as roadmap nodes. G_{good} is a subgraph of G, induced by V_{good}. $G_{rdmp}(V_{rdmp}, E_{rdmp})$ is the constructed backbone roadmap, and $(u, v) \in E_{rdmp}$ if and only if $u, v \in V_{rdmp}$ and u is in **employers** or **employees** list of v. We have the following lemma:

Lemma 1. *(i) All nodes in the roadmap constructed as in Algorithm 2.1 are safe nodes. That is $V_{rdmp} \subseteq V_{good}$. (ii) V_{rdmp} is a CDS of G_{good}. If G_{good} is a connected graph, G_{rdmp} is a connected graph.*

Proof. Recall that the algorithm elects backbone nodes based on priority defined in Eq.(2.1). When a node, s_i, is in a danger area, $d_o = 0$, and consequently $P_i = 0$. A node with zero priority will not be selected, due to condition (E.1). Therefore, $V_{rdmp} \subseteq V_{good}$. The second half of the lemma follows from the fact that, on election, a node ignores all neighbors with zero-priority. When s_i is in danger areas, it notifies all its neighbors, and all its neighbors take s_i out of consideration on backbone selection. The neighbor list (in each node) that the CDS election is based on is same as the neighbor list in G_{good}. Therefore the resulting V_{rdmp} is a CDS of G_{good}. With V_{rdmp} being a connected dominating set, G_{rdmp} has exact same number of connected components as G_{good}. So, when G_{good} is a connected graph, G_{rdmp} is a connected graph. □

Corollary 1. (**Correctness**) *Assume $d_s > d_c/2$, if there exists a safe sensor path in the original network, G, then there exists a safe one in the constructed backbone, G_{rdmp}.*

Proof. A more general statement of Lemma 1 is: V_{rdmp} consists of CDSs of all connected components of G_{good}. Assume that the sensing range d_s is larger than half of the communication range $d_c/2$, which implies that moving between two safe nodes results in a safe path. With Lemma 1, the correctness of the algorithm follows. □

Recall that we have already taken safety into account when constructing the backbone/roadmap: nodes in danger (with zero priority) will never be backbone nodes; the closer is a node to danger, the smaller the priority it has, and the less chance it has to be chosen as a backbone node. We take the shortest path in the roadmap as the best path, which gives a feasible path. The next theorem gives bounds on the path length.

Theorem 1. (**Path length**) *For a connected graph G_{good}, the shortest path found in the constructed roadmap, G_{rdmp}, is bounded by the shortest path found in G_{good}:*

$$D_{rdmp}(u, v) \leq 3D_{good}(u, v) + 2$$

where $D_{rdmp}(u, v)$ and $D_{good}(u, v)$ are lengths of the shortest path between u and v, in G_{rdmp}, G_{good}, respectively.

Proof. We show that we can always find a path in backbone related to the shortest path. From Lemma 1, vertices of G_{rdmp} is a CDS of G_{good}. Assume, in the worst case, all nodes in a shortest path between u and v are not in the backbone. Consider Figure 2.3(a). The

squared nodes are clusterheads, the dark/grey round nodes are doorway/gateway nodes, and all other white nodes are regular nodes. The thick lines represent connections in backbone, and the thin ones represent those not in backbone. (The path u, w_1, w_2, w_3, v is an example of the worse case shortest path.)

As shown in Figure 2.3(b), since each vertex must be dominated by a clusterhead, clusterheads (e.g. c_1 and c_2) dominating two adjacent vertices (e.g. w_1 and w_2) are at most 3-hops away. As w_1 and w_2 are not in backbone, there must exist another path of length ≤ 3 between c_1 and c_2 in backbone (e.g. $c_1 - a - b - c_2$). There must be at least one connection between one of $\{w_1, w_2\}$ and one of $\{b, c\}$, otherwise, w_1, and w_2 would have been in backbone.

For the start and goal nodes(i.e., u, and v), each of them should have a dominating clusterhead node within 1-hop. So in total, $D_{rdmp}(u, v) \leq 3D_{good}(u, v) + 2$. Figure 2.3(a) also gives a worst case scenario, showing that the bound is tight; once any vertex in the shortest path becomes a backbone node, or two of them share a clusterhead, the path length reduces. □

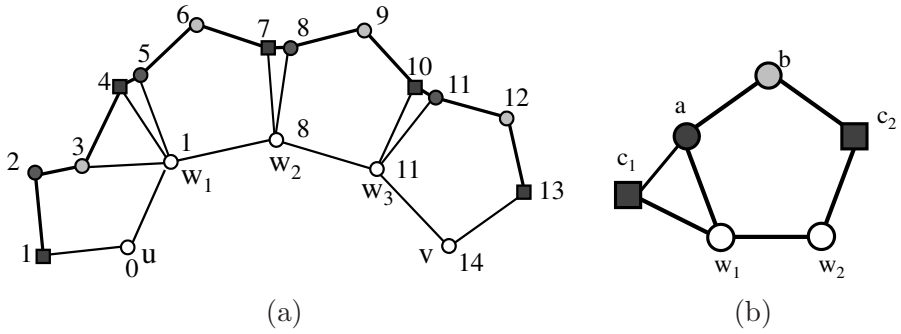


Figure 2.3: Bound on path length.

As mentioned, Algorithm 2.1 based on TMPO does not guarantee an approximation ratio with respect to the size of MDS. Nevertheless, the following known results on the size of minimum connected dominating set give a rough idea of the size of backbone based roadmaps. Let γ_c denote the cardinality of minimum connected dominating set, and Δ the maximum vertex degree of the graph.

Theorem 2. (**Bounds on optimal roadmap size**) *For any connected graph G,*

1. $\gamma_c \leq n - \Delta$; *[38]*

2. $\frac{n}{(\Delta-1)} \leq \gamma_c \leq 2m-n$, *where* n, m *is the number of vertices and edges of* G *respectively. The equality for lower bound is attained if and only if* $\Delta = n-1$, *and equality for upper bound is attained if and only if* G *is a path graph. [74]*

Theorem 3. (**Communication complexity**) *Let* $|CDS|$ *be the size of the backbone. The roadmap construction procedure in Algorithm 2.1 takes* $O(|CDS| \cdot \Delta)$ *messages. The goal dissemination procedure in Algorithm 2.2 takes* $O(|CDS|)$ *messages, if the messages are time-sorted.*

Proof. Once a node is elected as a CDS node, a *TYPE_CHG* message will be sent for 2 hops. That is at most $(1+\Delta)$ messages for every node in CDS. Thus, in total, the number of messages is $O(|CDS| \cdot \Delta)$.

Algorithm 2.2 finds a shortest path in the backbone. To show its communication complexity, we first assume the messages are time-sorted, which is a concept introduced in [8]. Message are said to be time-sorted if messages from a closer node arrive earlier than those from a further node. With such an assumption, if a node receives two *GOAL* messages, the one received later is further away from goal. So every node forwards only the first message it receives, and hence the total number of messages generated is $|CDS|$. Even though messages are not time-sorted in general, also shown in [8] is that the messages become time-sorted if we introduce a small extra wait time before broadcasting in each node. □

2.6 Computer simulations

We have performed simulations to show performance of proposed method using backbone as roadmap. We used an in-house developed software simulator for senor network, which models the unit disk communication, and simple point robots. In the following simulations, sensor nodes are uniformly distributed within a $1000m \times 1000m$ field.

2.6.1 Comparison with *AER*

The work closest to our approach is Adaptive Embedded Roadmaps (*AER*) recently proposed in [4]. It builds a roadmap by randomly choosing a certain number of nodes as milestones, and making connections among them. It works as follows, and for more details, please refer to the original paper. (1) Each sensor node decides weather or not to be a

milestone in the roadmap with a predefined probability p (we used $p = 0.1$ in our simulator, as in the original paper). (2) Milestone nodes broadcast *Neighbor-Discover* messages to discover the neighboring milestones. (3) Neighboring milestones respond with unicast *Neighbor-Found* messages, along the discovered path connecting two milestones. (4) Once neighboring milestones are discovered with the best (with respect to a defined *goodness* function) routes connecting them, unicast *Edge-Create* messages are sent to finalize edges connecting neighboring milestones; Nodes receiving the message become edge nodes. (5) When a goal is specified, the goal node becomes a milestone, and a similar neighbor discovery and edge creation procedure start as above to connect the goal node into the roadmap. After that the rest of the goal dissemination procedure are similar to ours in Algorithm 2.2.

To compare with *AER*, we have done simulations for different sizes of network, and different communication ranges. In the first simulation, the communication range is fixed at 100m, and the network size ranges from 200 to 800 sensor nodes. In the second simulation, the network size is fixed at 200 nodes, and the communication range varies from 100m to 300m.

Table 2.1: Performance in networks with different size, and communication range fixed to 100m.

Size	$\overline{d}$	Method	M_{con}	M_{goal}	Roadmap		Path	
					N_v	N_e	L_o	L
200	5	TMPO	1049	157	121	196		20
		AER	1135	272	96	150	18	21
300	8	TMPO	1624	288	165	322		18
		AER	2952	713	149	282	17	19
400	11	TMPO	2231	377	210	427		19
		AER	5665	1256	191	434	18	20
500	14	TMPO	2850	479	243	568		19
		AER	7852	2083	239	523	17	19
600	17	TMPO	3578	299	266	679		17
		AER	13099	3233	309	776	17	18
700	20	TMPO	4108	339	310	789		17
		AER	18913	4289	359	946	17	17
800	22	TMPO	4814	352	338	884		17
		AER	25305	6321	423	1174	17	17

The results are shown in Table 2.1 and 2.2. The performance metrics we compared include: (1) M_{con}, number of messages needed to construct a roadmap, which includes

Table 2.2: Performance in networks with different communication range, and network size fixed to 200 nodes.

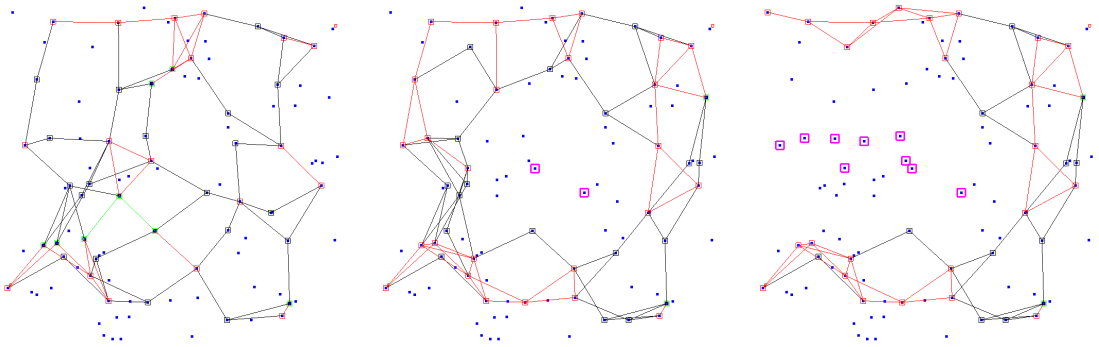
Range	$\overline{d}$	Method	M_{con}	M_{goal}	Roadmap		Path	
					N_v	N_e	L_o	L
100m	5	TMPO	1049	157	121	196	18	20
		AER	1135	272	96	150		21
150m	15	TMPO	1119	132	97	218	11	12
		AER	3161	639	86	193		13
200m	21	TMPO	1153	102	75	177	8	9
		AER	5754	860	95	233		9
250m	32	TMPO	1193	54	50	119	6	7
		AER	9465	896	92	296		7
300m	44	TMPO	1236	37	29	61	5	6
		AER	11346	859	74	251		6

messages used to collect neighborhood information; (2) M_{goal}, number of messages needed for goal dissemination. A message can either be a broadcast message or a unicast message, i.e., they both count as 1. (3) roadmap size, represented by number of nodes (N_v) and the number of links (N_e) used in the constructed roadmap. For AER, we set the probability of being a landmark to be 0.10 as used in [4], and roadmap size includes edge nodes. (4) path length, L, in terms of number of hops. For each case, we also computed the average degree ($\overline{d}$) of the entire network, and the optimal path length L_o.

The simulation results show that TMPO (i) generates fewer messages for roadmap construction and goal propagation, and (ii) produces a smaller roadmap when the network connectivity is high (network size $\geq$ 600 in Table 2.1, and range $\geq 200m$ in Table 2.2). The smaller number of messages in TMPO is mainly due to use of broadcasting as opposed to unicast used in AER. The smaller roadmap for high connectivity is achieved because of the nature of dominating set; intuitively when the connectivity is higher, it allows a node to dominate more nodes, and hence a smaller dominating set in general. Please note that, empirically in our simulations at least, the length of paths computed by both algorithms is generally fairly close to optimal, even though we have given a looser theoretic bound on path length for the TMPO algorithm whereas there is not such guarantee in AER.

2.6.2 Roadmap changes upon danger

In this simulation we show how the roadmap changes in response to dynamic dangers, where a sensor has reading beyond a predefined threshold. Figure 2.4(a) shows the initial roadmap. Sensor nodes (represented by ⊡) detect some dangers when they get excessive readings, as described in Algorithm 2.5, they reset their priority to 0, and hence lose their privilege to become backbone nodes. These sensor nodes send *DANGER* messages, and the roadmap thus adapts to these dangers, as shown in Figure 2.4(b) and (c), and in extreme cases, it may result in a disconnected roadmap.



(a) (b) (c)

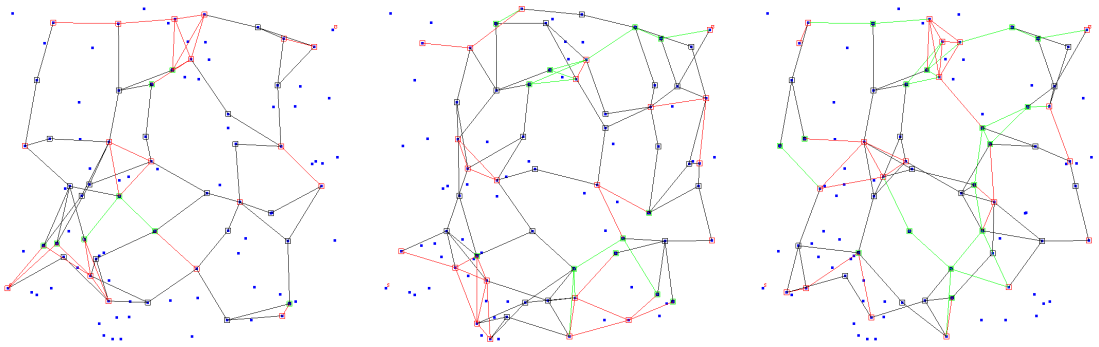
Figure 2.4: Roadmap changes with dynamic danger. ⊡ represents danger

2.6.3 Roadmap changes over time

In this simulation we show that nodes can share their roles as backbone nodes to average out the energy consumption to achieve a longer life of the network. Figure 2.5(a) shows the initial roadmap, Figure 2.5(b) shows the roadmap after 1 hour (scalable), and Figure 2.5(c) shows the roadmap after 2 hours. We can see the roadmap changes over time, and contains different set of nodes.

2.7 Summary

In this chapter, we proposed a new method for robot navigation in sensor networks. The method extracts backbone of the sensor network via a clustering algorithm, and uses the backbone as planning roadmap. The constructed backbone consists of a connected dominating set of the (safe portion of the) network, and is elected based on the defined priority



(a) (b) (c)

Figure 2.5: Roadmap adapts over time, as nodes need to share their roles as backbone nodes to enhance network longevity.

which takes energy and distance to danger into account to achieve network longevity and path safety. As the backbone can be constructed in a distributed way with only 2-hop neighbor information, the method avoids flooding in roadmap construction and path planning. Since the backbone is a connected dominating set of the sensor network, it closely related to network connectivity: The backbone has exact same number of connected components as the original sensor network, and any safe sensor node is at most one hop away from the backbone. As a result, it guarantees to find a feasible path if there exists one, and moreover, it provides a performance guarantees on the length of the computed path with respect to the optimal solution.

Chapter 3

Distributed Roadmaps for Navigation

3.1 Overview

In the previous chapter, we assume binary sensing model for sensors: in measuring state of the environment (e.g., temperature) within its sensing range, the sensor returns a single reading, if the reading is beyond a certain threshold, then the entire sensing region of the sensor node is deemed to be *Danger* (e.g., too hot), otherwise is deemed to be free. In this chapter, we study the navigation problem under a more general sensing model, where a sensor node can sense obstacles and other potential dangers within its sensing range. We propose a distributed sampling based planning algorithm, *Distributed PRM* (D-PRM) and *Distributed PRM* (D-RRT) , to systematically incorporate the general spatial sensing model for each sensor. It takes into account obstacles in determining feasible paths. Each sensor creates a local probabilistic roadmap (a patch) similar to the classic PRM [46], or RRT [53] , but only in its locally-sensed environment. Two different patches of roadmap are "stitched" together with a set of *relay points*, lying in the common region shared by the two patches. Sensor nodes mutually negotiate the connectivity of their patches by sending messages regarding the status of their respective relay points. When two adjacent sensor nodes see a relay point free, it becomes a connecting point for the two patches. To find the shortest path on distributed roadmaps, a distributed (discrete) navigation field is created across the sensor network, which maps each landmark (i.e., a random sample) in

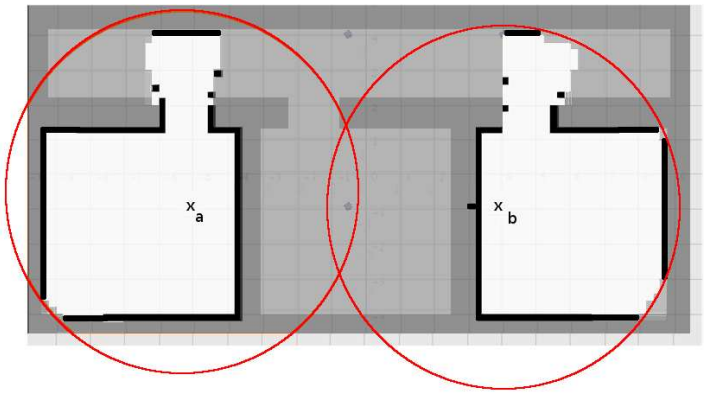


Figure 3.1: Sensor model of nodes (a) and (b). Black represents *Blocked*, white represents *Free*, and gray represents *Unknown*. Circle is the sensing range.

local roadmaps into distance to the desired goal, and the best path is computed by gradient descent.

Similar to previous chapter, consider a system with a stationary sensor network and a robot. A sensor node can sense obstacles and other potential dangers within its sensing range, d_s, and creates a map, $\mathcal{H}_i$, for its local physical environment, as shown in Figure 3.1. A point on a map, $\mathcal{H}_i$, is in one of three different states: *Free*, *Blocked*, or *Unknown*, and we denote the free portion as $\mathcal{H}_i^{free}$. The distributed representation of the environment is $\mathcal{H} = \{\mathcal{H}_1, \mathcal{H}_2, \cdots, \mathcal{H}_n\}$. Certainly two maps may overlap, and we call the overlapped region of two maps as *relay zone*, $RZ(i,j) = \mathcal{H}_i \bigcap \mathcal{H}_j$. Note that it is possible that the same point in the overlap region has different states in two maps, for example due to occlusion, a point may be *Free* in $\mathcal{H}_i$, but *Unknown* in $\mathcal{H}_j$.

For simplicity, we will start with a point (circular) robot as in the previous chapter, $\mathcal{A}$, for which the workspace and the configuration space (C-space) are identical. Then we consider a holonomic robot with nontrivial shape, where orientation of the robot needs to be taken into account, and the robot can translate and rotate freely, i.e., its configuration space is 3-dimensional (x, y, θ). Finally, we look into navigation of a robot formation for cooperative tasks. In all cases, robots are mounted with sensor and wireless communication devices that can communicate with sensor nodes within d_c, as well as with each other in the formation case. We assume a robot knows its own location, x_A, with respect to a global frame. This can be realized either by mounting the robot with devices such as GPS, or

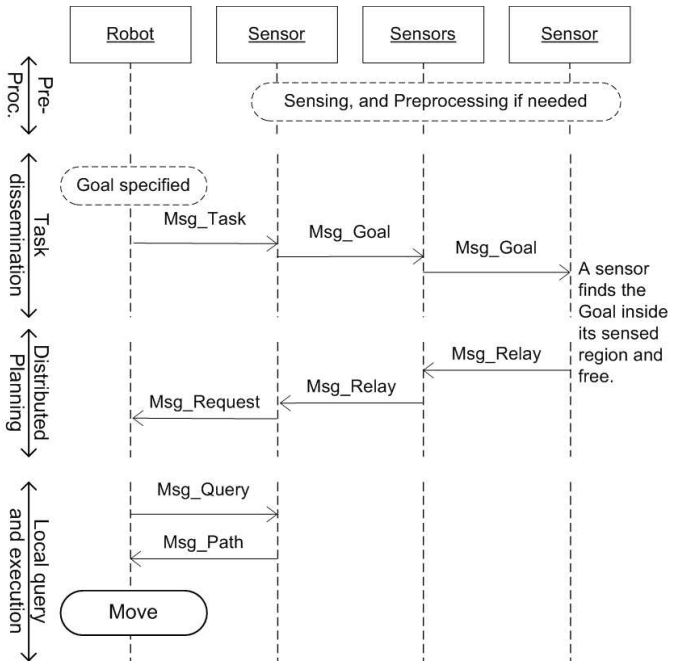


Figure 3.2: Messaging in distributed path planning.

using localization techniques reviewed in Section 1.3.2.

Thus, the distributed path planning problem is to find a distributed path,

$$\Pi = \left\{\Pi_1, \Pi_2, \cdots, \Pi_m\right\}$$

for the robot to go from the current position, x_s, to a desired goal, x_g, such that (i) Π is a continuous path, i.e., $\Pi_k(\tau), \tau \in [0,1]$, and $\Pi_k(1) = \Pi_{k+1}(0)$, and (ii) each path segment, Π_k in a sensor node, is guaranteed to be collision free inside the sensed local environment.

3.2 Solution outline

The proposed distributed sampling based framework consists of four different phases as shown in Figure 3.2. Detailed algorithms are given in Section 3.3-3.5.1, and the general solution is outlined as follows. For conceptual simplicity we present these phases as distinct and in a sequential order. In practice, these phases may overlap or interleave, especially when there are dynamic changes in the environment.

3.2.1 The four phases

Local perception, pre-processing and roadmaps

In this phase, the sensor network gains knowledge of the system connectivity and the environment. Sensor nodes broadcast their basic information (e.g., positions), establish connections with their neighbors, and perceive their local environments by using sensors (e.g., cameras, laser, etc.). Pre-processing of planning is also done in this phase. For example, D-PRM creates a C-space roadmap, $\mathcal{R}_i = (\mathcal{V}_i, \mathcal{E}_i)$, for the locally-sensed environment, where $\mathcal{V}_i$ is the set of landmarks (random samples), and $\mathcal{E}_i$ is the set of connections among landmarks. The set of neighbors of a landmark, v, is denoted by $\mathcal{N}_i(v) = \{u | (u, v) \in \mathcal{E}_i\}$. Note that each roadmap is local, there is no sharing of roadmaps, and hence there is no communication involved, as far as local roadmap building is concerned. In our simulation, each sensor node is assumed equipped with a laser range-finder to sense the environment, and uses grid map as local world representation, however other choices of sensors (e.g. cameras) and world representation (e.g., continuous function) are also possible, and our algorithms easily extend to these cases as well.

Task dissemination

When the robot wants to go somewhere, it sends out a request (in *MSG_TASK* message) to the sensor network asking for direction. A sensor node that has the goal inside its sensed region then initiates distributed planning, in the next phase.

Distributed planning

This is the main phase. We model the distance to goal as a generic cost function, $C(v)$, and local roadmaps in different sensor nodes are "stitched" together by propagating a distributed navigation field based on $C(v)$. The propagation starts from the sensor node that senses the goal, and is relayed to it neighboring sensor nodes by sending *MSG_RELAY* messages, which contain the information regarding cost function values at shared relay points. With the navigation field, each sensor node knows the local segment of the path (to the goal) from any point inside its local environment by following the navigation field, which is essentially the gradient descent on the cost function.

Query and execution

The robot queries for the path from the sensor network (with *MSG_QUERY* message), and moves from one sensor node to another. A sensor node receiving the query message returns the local segment of the path to goal (in *MSG_PATH* message) based on its local navigation field. The robot moves along the returned segment of the path, reaching next sensor node and repeats the query/execution procedure again until the desired goal is reached.

3.2.2 Relay sets - the stitches

A roadmap, $\mathcal{R}_i$, defines local C-space connectivity corresponding to $\mathcal{H}_i$, and connectivity between $\mathcal{R}_i$ and each of its neighbors, say $\mathcal{R}_j$, is defined by a *relay set*, $RS(i,j)$, within $RZ(i,j)$, the *relay zone*. As shown in Figure 3.3, a relay set consists of relay points, and each relay point comprises two relay landmarks, one each from the two neighboring roadmaps, $\mathcal{R}_i$ and $\mathcal{R}_j$. The two landmarks are at the same position but their status may differ in two roadmaps. When the two landmarks are both *Free*, they are implicitly "stitched" together (as in inset 1), and the relay point becomes a connecting point between $\mathcal{R}_i$ and $\mathcal{R}_j$; Otherwise, they are disconnected, as shown in inset 2, where the dark dot is occluded by obstacles, and hence its status is unknown. The connectivity of the relay set is determined by *MSG_RELAY* messages in the planning phase (detailed in the next section).

There are two main ways of choosing relay sets: in a deterministic manner, or in a probabilistic manner. For instance, one choice would be to lay a grid of uniform resolution d_r in $RZ(i,j)$ and the relay points correspond to the center of grid cells. Another choice would be random samples in $RZ(i,j)$. The latter is more in line with the probabilistic philosophy of sampling based algorithm, however, they carry higher communication overhead, since messages will need to contain position information of each sample.

For point robots, we chose to use the deterministic manner for simplicity, and to reduce the communication overhead, we further restricted the set of relay points to be on a line segment connecting the two intersecting points of communication boundary circles, as shown in Figure 3.3. However, when the environment gets more complicated, particularly with narrow passages, a full-blown fine-resolution grid might be needed in order to find the solution. Unfortunately, such a deterministic manner quickly suffers from the perils of increased dimensionality, if we change the relay sets from points on a line segment to a 2-d grid, and to a 3-d grid for general robots. Therefore, in these cases, we chose the probabilistic

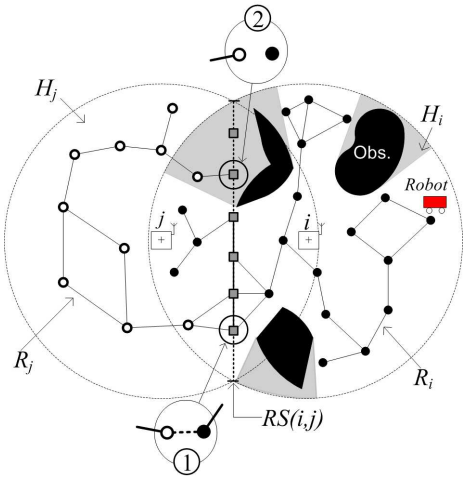


Figure 3.3: Distributed Roadmaps. Dark dots represent landmarks for node i, white dots represent landmarks for node j, and gray squares represent relay points. As shown in the insets, each relay point comprises two relay landmarks, one each from the neighboring roadmaps.

relay sets, with sophisticated sampling techniques, such as Bridge Test sampling in [43], in order to reduce the number of samples in relay zones while maintaining good connectivity,

3.2.3 Notation Summary

- $G = (V, E)$: proximity graph of the system;
- $N(i)$: neighboring sensor nodes of a node i;
- $\mathcal{H}_i$: local physical environment perceived by node i;
- $\mathcal{R}_i = (\mathcal{V}_i, \mathcal{E}_i)$: C-space roadmap for environment $\mathcal{H}_i$;
- $\mathcal{N}_i(v)$: neighboring landmarks of v on $\mathcal{R}_i$;
- $C_i(u)$: cost to goal from a local landmark $u \in \mathcal{V}_i$;
- $\pi_i(u)$: next landmark in the path from u to goal.
- $RZ(i, j) = \mathcal{H}_i \cap \mathcal{H}_j$: relay zone (in physical space) between node i and j;
- $RS(i, j) = \mathcal{R}_i \cap \mathcal{R}_j$: relay set (in C-space) between node i and j;

We use i, j, k to refer to sensor nodes in the sensor network, and u, v, w for landmarks in a roadmap.

3.3 Distributed roadmaps for point robots

3.3.1 Planning

In planning phase, a distributed navigation field is computed over the local roadmaps. For each landmark u in $\mathcal{R}_i$, we define two functions: $C_i(u)$ and $\pi_i(u)$, where $C_i(u)$ is the distance from u to the desired goal, and $\pi_i(u)$ is the next landmark in the shortest path from u to the desired goal. Note that (u, v) points in the negated gradient direction of $C_i(u)$,

$$\pi_i(u) = \arg\min_{v \in \mathcal{N}_i(u)} \big(C_i(v) = C_i(u) + \|v - u\|\big).$$

The procedure of updating the navigation field is given in sub-routine, *Update_Field*() (Line 16-25, Algorithm 3.1), which is essentially a distributed Dijkstra algorithm. It maintains U as a list of landmarks with the smallest cost so far (often called open list in the literature), and examines all adjacent landmarks of every $u \in U$. If the condition

$$v \in \mathcal{N}_i(u) \text{ and } \big(C_i(v) < C_i(u) + \|v - u\|\big)$$

in Line 19 holds, it means a shorter path has been found from v to the goal, via u. Therefore, v is added into U. There are two ways to trigger the navigation field update:

- A sensor node that has the desired goal within its sensed region initiates the planning by adding the desired goal as a landmark in the roadmap, and sets its distance to goal as 0, then updates its navigation field (Line 2-7, Algorithm 3.1).
- When a sensor node i receives an *MSG_RELAY* message from one of its neighbor, say j, it compares its own relay set to those in the message, if some in the message give any smaller cost (better path) for any relay points in the set, these relay points are updated to the smaller cost, and these newly-improved relay points are added into U for update of the navigation field (Line 8-15).

As shown in Line 26-29, after updating the navigation field, if the cost ($C(v)$) for any relay point in a relay set, $R(i, k), k \in N(i)$, has been improved, it means a shorter path is found for the relay point, and a *MSG_RELAY* message is sent to notify node k. The message contains a list of relay points and their newly-improved costs.

Algorithm 3.1: Local PRM Planning of Sensor Node i

begin
  if $q_g \in \mathcal{H}_i$ **then** /* sensed the goal? */
    Add q_g into roadmap $\mathcal{R}_i$;
    $v_g \leftarrow (q_g)$; $C_i(v_g) \leftarrow 0$;
    $U \leftarrow \{v_g\}$;
    Update_Field(U); /* see sub-routine */
  endif
  if *MSG_RELAY:*$\big(j, \{C_j(v), v \in RS(i,j)\}\big)$ *received* **then**
    foreach $v \in RS(i,j)$, *s.t.* $C_j(v) < C_i(v)$ **do**
      $C_i(v) \leftarrow C_j(v)$;
      $U \leftarrow U \bigcup \{v\}$;
    endfch
    Update_Field(U);
  endif
end

Sub-Routine: Update_Field(U)
begin
  foreach $u \in U$ **do** /* Loop until $U = \emptyset$ */
    foreach $v \in \mathcal{N}_i(u)$ **and** $C_i(v) < C_i(u) + \|v - u\|$ **do**
      $C_i(v) \leftarrow C_i(u) + \|v - u\|$;
      $\pi_i(v) \leftarrow u$;
      $U \leftarrow U \bigcup \{v\}$;
    endfch
    $U \leftarrow U \backslash \{u\}$;
  endfch
  for $k \in N_i$ **do**
    $V_u \leftarrow \big\{v \big| v \in RS(i,k),$ and $C_i(v)$ is improved$\big\}$;
    Send MSG_RELAY:$\big(k, \{C_k(v), v \in V_u\}\big)$ to k;
  endfor
end

3.3.2 Path Query

When the navigation field has been computed as in previous section, an *MSG_REQUEST* message is sent to the robot, indicating that the sensor network is ready to help navigate it. To move toward the desired goal, the robot constantly interacts with sensor nodes of the network, as shown in Algorithms 3.2 and 3.3:

- The robot sends an *MSG_QUERY* message to sensor nodes around, and waits (Line 3-5 in Algorithm 3.2).
- A sensor node receives the message, and looks up its navigation field, computes the segment of the best (shortest) path, and sends the path segment back to the robot in an *MSG_PATH* message (Algorithm 3.3).
- The robot may receive multiple *MSG_PATH* messages from different sensor nodes. It decodes the segment in each message, picks the best path, and moves along the chosen path segment (Line 5-10 in Algorithm 3.2). When it reaches the end of the segment, it approaches another sensor node, and it sends out another query message. The robot repeats such query-and-move procedure until it reaches the desired goal.

Algorithm 3.2: Robot Query for Paths

```
1  begin
2      x ← Robot current position;
3      path ← ∅, c ← ∞;
4      Send MSG_QUERY message with x;
5      repeat
6          Receive MSG_PATH:(p, l), where p is the found path segment and l is the cost defined;
7          if l < c then
8              path ← p, c ← l;
9          endif
10     until timeout ;
11     Robot execute path p;
12     Repeat from Line 3, until p(1) = x_g;
13 end
```

3.4 Distributed roadmaps for geometric shapes

The key change in extending above algorithm to robots with shapes is that the orientation of the robot now matters, which makes a 3-dimensional C-space for the robot, as opposed

Algorithm 3.3: Sensor Respond to Query with Distributed PRM

begin
  Receive MSG_QUERY:(x) from the robot;
  $u_s \leftarrow \{x\}$;
  $U \leftarrow \{v \in \mathcal{V}_i | \|u_s - v\| < r, (u_s, v)$ collision free$\}$;
  $C_i(u_s) \leftarrow \min_{v \in U} \left(C_i(v) + \|u_s - v\|\right)$;
  $\pi_i(u_s) \leftarrow \arg\min_{v \in U} \left(C_i(v) + \|u_s - v\|\right)$;
  $l \leftarrow C_i(u_s)$, $\Pi_i \leftarrow \{u_s\}$;
  while $\pi_i(u_s) \neq$ **nil do**
    $\Pi_i \leftarrow \Pi_i \bigcup \{\pi_i(u_s)\}$;
    $u_s \leftarrow \pi_i(u_s)$;
  endw
  Send MSG_PATH:(Π_i, l);
end

to the previous 2-dimensional space, and therefore the roadmap in each sensor node is constructed in the 3-dimensional space, so is relay sets.

3.4.1 Notify robot shape

As shown in Figure 3.2, with *Distributed PRM*, each sensor node constructs roadmap in the pre-processing phase. To plan for robots with shapes, sensor nodes certainly need to have a knowledge of robot geometry in order to have effective local roadmaps. We assume either sensor nodes have such knowledge in advance, or such knowledge can be injected into the network via an *MSG_SHAPE* messages. In the later case, the *MSG_SHAPE* messages include a list of points representing the robot's polygonal shape.

3.4.2 Probabilistic relay sets for 3D C-space

In the previous case of deterministic relay set, a relay set is uniquely defined by two neighboring nodes. For robots with shapes, we use probabilistic relay sets, and sensor nodes exchange random samples in relay zone by sending messages. Relay points in a relay set are randomly generated by different sensor nodes and then sent to each other, and this results in the "directed" representation of relay sets. In each sensor node, two relay sets are kept for each neighbor, one inbound, the other outbound. For example, in sensor node, i, for a neighbor $j \in N(i)$, $RS^i(i,j)$ is the outbound relay set whose points are generated by i and to be sent to its neighbor j, whereas $RS^i(j,i)$ is the inbound relay set whose points are generated by j, and sent to i via *MSG_POINTS* messages. In general, $RS^i(i,j) \neq RS^i(j,i)$, on the other hand, $RS^i(i,j) = RS^j(i,j)$, as $RS^j(i,j)$ is regenerated from a *MSG_POINTS*

Algorithm 3.4: Roadmap and relay set construction for sensor i

```
1  begin
       //Sampling and creating roadmap
2      for i=1 to MAX do
3          q_r ← RandomConfig();
4          if q_r is collision free then
5              Add q_r into roadmap R_i;
6              J ← {j | ||x_j − x_r|| < d_c, j ∈ N(i)};
7              foreach j ∈ J do Add q_r to RS(i, j);
8          endif
9      endfor
       //Sending new relay points to neighbors
10     foreach k ∈ N(i) do
11         S_k ← newly-added samples in RS(i, k);
12         Send MSG_POINTS:(k, S_k) to node k;
13     endfch
       //Getting new relay points from neighbors
14     if MSG_POINTS:(j, S_j) received then
15         foreach q_n ∈ S_j do
16             Add q_n into RS(j, i);
17             if q_n is collision free then Add q_n into roadmap R_i;
18         endfch
19     endif
20 end
```

message sent by i: node j adds relay set $R^i(i,j)$ received from node i to its own $R^j(i,j)$. The roadmap and relay set construction procedure is detailed in Algorithm 3.4. Line 2-9 generate random landmarks, and add them to the roadmap. If a landmark is within the relay zone with a neighbor, it will be added into the outbound relay set $R^i(i,j)$, as in Line 6-7. In Line 10-13, new relay points in the outbound relay points are sent to corresponding neighbors. In Line 14-19, node i receives *MSG_POINTS* message, extracts new relay points, and adds these points into inbound relay set $RS^i(j,i)$; collision free points thereof are further added into roadmap $\mathcal{R}_i$. Note that in Algorithm 3.4, since all relay sets are with respect to i, we omit the superscript i in the pseudo-code.

RandomConfig() in Line 3, generates random landmarks, and it can be implemented with different sampling strategies. A straightforward choice is uniform sampling. However, as known in sampling based path planning literature, uniform sampling does not yield good performance in difficult environments, particularly with narrow passages. More advanced sampling strategies, such as the Bridge Test Sampling [43], have been proposed to deal with such environments. The common idea behind these strategies is to bias toward and put more samples in difficult regions. Here, we adopt the Bridge Test (BT for short hereafter) sampling, which starts with an in-collision configuration, and find another in-collision configuration nearby, and then takes the mid-point of the two, if it is collision free. As shown in Algorithm 3.5, we employ a hybrid sampler combining uniform sampling and BT sampling, where a ratio w,

$$\omega = \frac{\text{current \# of bridge test samples}}{\text{current \# of uniform samples}} \tag{3.1}$$

and a threshold η are defined. Only when $\omega < \eta$, BT sampling strategy is enabled (Line 5-3.5), otherwise it simply uses the uniform sampling. In this way, roadmaps will have enough samples to cover difficult areas as well as open areas.

3.5 Other extensions

3.5.1 Distributed roadmaps for cooperative tasks

The extension of *D-PRM* to multi-robot formation is relatively straightforward if we assume that we have a rigid formation, where all robots relative position are fixed. We use a polygonal shape to bound the entire formation, as shown in Figure 3.4. Then we apply the motion planning scheme above for the bounding polygon, and find a path for the center of

Algorithm 3.5: RandomConfig: Random C-space Sampler

```
1  begin
2      q_a ← A uniform sample in C-space;
       /* ω:  in Eq.(3.1); η:  predefined threshold.                    */
3      if q_a is collision free and ω ≥ η then
           /* Keep uniform samples                                      */
4          return q_a;
5      else if q_a is NOT collision free and ω < η then
           /* To attain BT samples                                      */
6          q_b ← A sample around q_a with N(q_a, σ);
7          if q_b is NOT collision free then
8              q_m ← midpoint of (q_a, q_b);
9              q_r ← A sample around q_m with N(q_m, σ);
10             if q_r IS collision free then return q_r;
11         endif
12     return NIL;                          /* Otherwise, ignore the sample */
13 end
```

formation. The paths for individual robots are computed based on their relative position to the center of formation.

With multi-robot formation, one robot is elected as the leader, and others follow the leader. Only the leader interacts with the navigating sensor network, sending *MSG_TASK* and *MSG_QUERY* messages, and receiving *MSG_REQUEST* and *MSG_PATH* messages, as shown in Figure 3.2. The interaction between the leader and the sensor network is regarding the entire formation rather than the leader robot itself. When the leader sends the *MSG_TASK* message, the bounding polygon is embedded in the message; and the planned path, with respect to the center of the formation, is returned in the *MSG_PATH* message. When the leader receives the message, it shares the path information with followers, and the leader and followers translate the received formation path into their individual paths.

To simplify the problem, we assume robots in the formation are circular robots, namely even though the orientation of the formation matters, the orientation of each individual robot does not have effect on formation orientation, and only the positions of the robots define the orientation of the formation. Assign a frame $\mathcal{F}$ to the center of the robot formation, and denote $\mathbf{x}_F = (x_F, y_F, \theta_F)$ the position and orientation of the frame, and assume a robot k in the formation has relative position to the center of the formation, with respect to $\mathcal{F}$, $\mathbf{p}_k^F = (x_k^F, y_k^F)$. Therefore, the planned formation path is translated into each robot's local

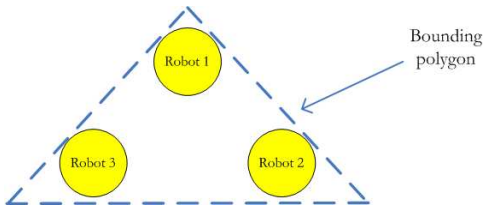


Figure 3.4: Shape representing a formation.

path as follows, with respect to a universal frame,

$$\mathbf{x}_k = \begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_F \\ y_F \end{pmatrix} + R(\theta_F) \cdot \mathbf{p}_k^F$$

where $R(\theta_F)$ is the 2×2 rotation matrix.

3.5.2 Distributed Random Tree (D-RRT)

We also implemented *Distributed RRT*(*D-RRT*), based on *RRT*. The detailed algorithms of D-RRT planning and local query are shown in Algorithm 3.6 and 3.7 respectively. *D-RRT* can be seen as a special case of *D-PRM*, however, unlike *D-PRM*, in *D-RRT* the local roadmap $\mathcal{R}_i$ is actually a forest, a set of trees, instead of a general graph as in *D-PRM*. Every tree in $\mathcal{R}_i$ either roots at a relay point in a relay set, or at the goal. (Line 3 and 8 in Algorithm 3.6.) It represents a free space reachable from the goal, and hence every time the goal changes, the tree needs to be regenerated. Other difference between *D-RRT* and *D-PRM* are as follows.

- In order to guarantee the overall roadmap to form a tree structure instead of a graph, growing trees (forest) inside each sensor node is delayed into the planning phase, otherwise the overall (proximity) graph structure of the sensor network may induce a cycle when forests are stitched together. As shown in Algorithm 3.6, Line 6-11, a tree in $\mathcal{R}_i$ is created and starts to grow only after a relay point is reached by a forrest in its neighbors (therefore the *MSG_RELAY* message). The update sub-routine, Line 14-32, grows the forest and updates the navigation field at the same time, whereas *D-PRM* creates a roadmap in the preprocessing phase, and only updates the navigation field in the planning phase (as in Algorithm 3.1).

- This limits parallelism of the planning phase in that a sensor node will not start growing its forrest until some of its replay points have been reached from the expanding tree. To compensate, *D-RRT* sends an MSG_RELAY message as soon as a relay point is connected to the forest (Line 29), so that the neighbor can start growing its forest upon reception of the message. *D-PRM*, on the other hand, waits until local navigation field update finishes, and sends out only one message for all improved relay points in a relay set, since *D-PRM* cares for the shortest path in the roadmap generated in the preprocessing phase.
- Since $\mathcal{R}_i$ is a forest, it defines a unique navigation field: $\pi_i(u)$ always points to the parent node of u. This difference also results in slight differences in the query phase (on the sensor side), as shown in Algorithm 3.7.

3.6 Discussions

Sensing and communication ranges

In our implementation, while obstacles occlude sensing, they do not prevent communication (no line-of-sight constraint for communication). These assumptions result from our choice of modality for communication (wireless). Our methodology can be adapted to different types of communication constraints. Generally speaking, the communication constraint can indirectly impact map generation and computation of relay points. In order to ensure that the robot is able to communicate with at least one sensor node at any time while moving along the planned path, the local roadmap of a sensor node must be generated in the overlap of its communication and sensing regions. Relay points must be generated in the overlap of the communication regions of two neighboring sensor nodes. So, for example, one could easily incorporate line-of-sight constraint for communication, by (i) generating local roadmap for each sensor node in region visible to the node, and (ii) generating relay points in regions that are visible to both neighboring sensor nodes, which requires some change on Line 6 in Algorithm 3.4 to take field of view into account.

Complexity and Robustness

As shown in Figure 3.2, different messages occur in different phases. *MSG_POINTS* messages are sent in roadmap and relay point construction phase, and they occur only once.

Algorithm 3.6: Local RRT Planning of sensor node i

begin
  if $q_g \in \mathcal{H}_i$ **then** `/* sensed the goal? */`
    Create a tree rooted at q_g, and add the tree into $\mathcal{R}_i$;
    $v_g \leftarrow (q_g)$; $C_i(v_g) \leftarrow 0$;
  endif
  if *MSG_RELAY:*$\big(j, \{C_j(v), v \in RS(i,j)\}\big)$ *received* **then**
    foreach $v \in RS(i,j)$, **and** $C_i(v) = \infty$ **do**
      Create a tree rooted at v, and add it into $\mathcal{R}_i$;
      $C_i(v) \leftarrow C_j(v)$;
    endfch
  endif
  Update $\mathcal{G}_i$ and F_i; `/* see sub-routine */`
end

Sub-Routine: Update $\mathcal{G}_i$ and F_i
begin
  `/* Similar to RRT, but` $\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i)$ `is a forest rather than a single tree. */`
  repeat
    $v_{rand} \leftarrow$RandomConfiguration();
    $v_{near} \leftarrow$NearestNeighbor(v_{rand});
    $v_{new} \leftarrow$ Extend(v_{near}, v_{rand});
    if *IsCollisionFree*(v_{near}, v_{new}) **then**
      $\mathcal{V}_i \leftarrow \mathcal{V}_i \bigcup \{v_{new}\}$;
      $\mathcal{E}_i \leftarrow \mathcal{V}_i \bigcup \{(v_{near}, v_{new})\}$;
      $C_i(v_{new}) \leftarrow C_i(v_{near}) + \|v_{near} - v_{new}\|$;
      $\pi_i(v_{new}) \leftarrow v_{near}$;
    endif
    for $k \in N(i)$ **do**
      $V_u \leftarrow \big\{v \big| v \in RS(i,k),$
          $C_i(v) = \infty$, and $\|v - v_{new}\| < \epsilon, \big\}$;
      Send MSG_RELAY:$\big(k, \{C_k(v), v \in V_u\}\big)$ to k;
    endfor
  until *timeout* ;
end

Algorithm 3.7: Sensor Response to Query with Distributed RRT

begin
  Receive MSG_QUERY:(x) from the robot;
  $u_s \leftarrow$NearestCollisionFreeNeighbor(x);
  $l \leftarrow C_i(u_s) + \|x - u_s\|$;
  $\Pi_i \leftarrow \{x, u_s\}$;
  while $\pi_i(u_s) \neq$ **nil do**
    $\Pi_i \leftarrow \Pi_i \bigcup \{\pi_i(u_s)\}$;
    $u_s \leftarrow \pi_i(u_s)$;
  endw
  Send MSG_PATH:(Π_i, l);
end

MSG_GOAL, *MSG_RELAY* messages are sent every time a new goal for the robot is specified. *MSG_QUERY* and *MSG_PATH* are sent when the robot executes and moves along the computed path.

In constructing roadmaps and relay points (Algorithm3.4), the number of *MSG_POINTS* messages, is $O(n\Delta)$, where n is the number of sensor nodes and Δ is the average node degree of the sensor network. As pointed out in [98], with n randomly placed nodes, the necessary and sufficient condition to form a connected network is that each node connects to $O(\log n)$ neighbors. Therefore, Δ in a typical sensor network is $O(\log n)$, and hence the communication complexity for construction phase is $O(n \log n)$. In search phase (Algorithm 3.1), the number of *MSG_GOAL* and *MSG_RELAY* messages is $O(n^2)$. In execution phase (Algorithms 3.2 and 3.3), the number of *MSG_QUERY* and *MSG_PATH* messages is (n).

A centralized algorithm, where sensor nodes send their local environment map (or the local roadmap) to the robot, and then use a traditional PRM, would certainly be worse in communication complexity. The total number of messages needed to build the global map (or global roadmap) in the worst case would be $O(n^2)$, since each sensor node need to set up a route to the robot. More significantly, when sending a local map (or the local roadmap), the size of each message is much larger. Even with a modest 256×256 sensing resolution, the size of each local map (and hence the message) would be $8K$ bytes, about 30 times larger than the message size in the D-PRM case, which consists of information about relay points, about 240 bytes (typically 20 relay points $\times$ 12 bytes for each relay point location, (x, y, θ). If a local roadmap were to be sent instead, still the message size is significantly larger. It would consist of the entire local roadmap, which is about 2-3 times larger than relay points sent in *D-PRM* case.

Given sensor data is correct and the assumptions and constraints of our communication model are satisfied, *D-PRM* is a correct algorithm in that the path it computes is guaranteed to be collision free. In addition, *D-PRM* is intrinsically robust to communication failure in that the overall roadmap will adapt to communication failure. If a node fails during the planning phase, it is automatically excluded from consideration, since there is no message in and out of the failed node; if a node fails during execution phase, it will not respond to robot enquiry, and the robot will choose amongst the paths sent by other nodes, and continue with the best alternative path available. However, once the failed nodes result in a disconnected communication graph, and consequently a disconnected roadmap, the algorithm may fail to find a feasible path due to disconnection. Also note that the sampling based algorithm

provides only probabilistic assurance in that with more landmarks, the probability that it finds an existing solution is higher. However, it is possible that even if there is a feasible path, the algorithm may not find it due to a limited number of landmarks in the run time allowed.

Extension to higher dimensional robots

In previous section, we have shown the extensions to 3-DOF robots with shapes, and robot formations. The extension to high-DOF robots, such as a mobile manipulator or even a snake robot is conceptually straightforward, as long as the geometries and kinematics of the robot are given in *MSG_SHAPE*, by which sensor nodes are notified of robot C-space.

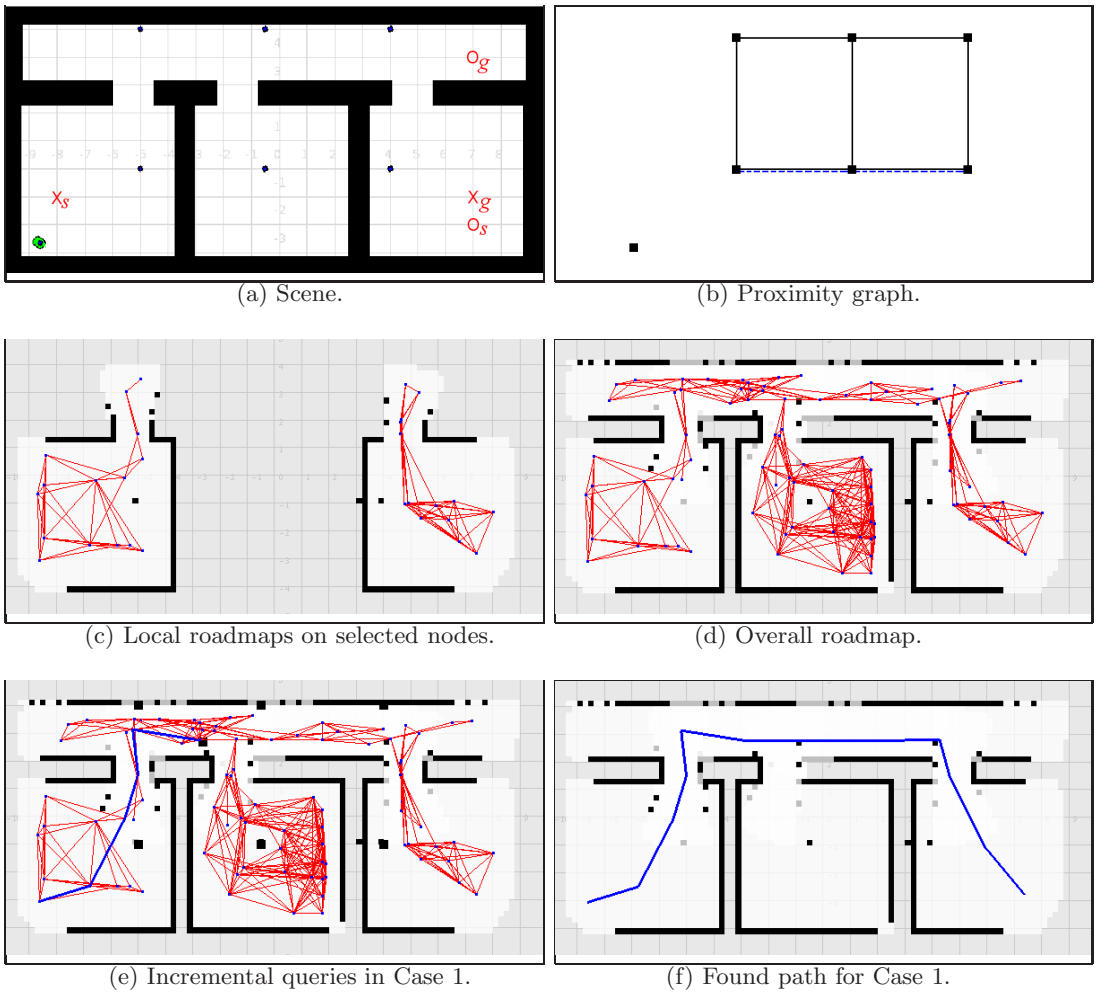
Other assumptions

Many specific choices in our implementation, such as use of laser scanner as a sensor, grid map representation of environment, etc. can be modified. The key here is to have a map in each individual sensor node that can be used to determine if a robot configuration is free or not. Such occupancy information can be obtained from camera images using standard image processing, such as stereo-vision. Such image/vision processing algorithms with overhead cameras can be easily incorporated in visual sensor networks such as the ones mentioned in [84].

We do assume stationary networks, however, it does not have to be a permanently static sensor network. For example, in an emergency rescue application such as the one mentioned in the introduction, it can be a mobile sensor network, (where each sensor node itself is a mobile robot) that becomes stationary temporarily to help navigate a rescue robot or a fire fighter.

3.7 Computer simulations

We now present simulation results to show the effectiveness of the proposed distributed sampling based planning method. Our simulation is based on Player/Stage [35] integrated with the sensor network simulator presented in Chapter 2. The network layer of the previous simulator is now replaced with the *wsn* (wireless sensor network) model that comes with Player/Stage to transmit user data of different sizes. Robots and sensor nodes are equipped

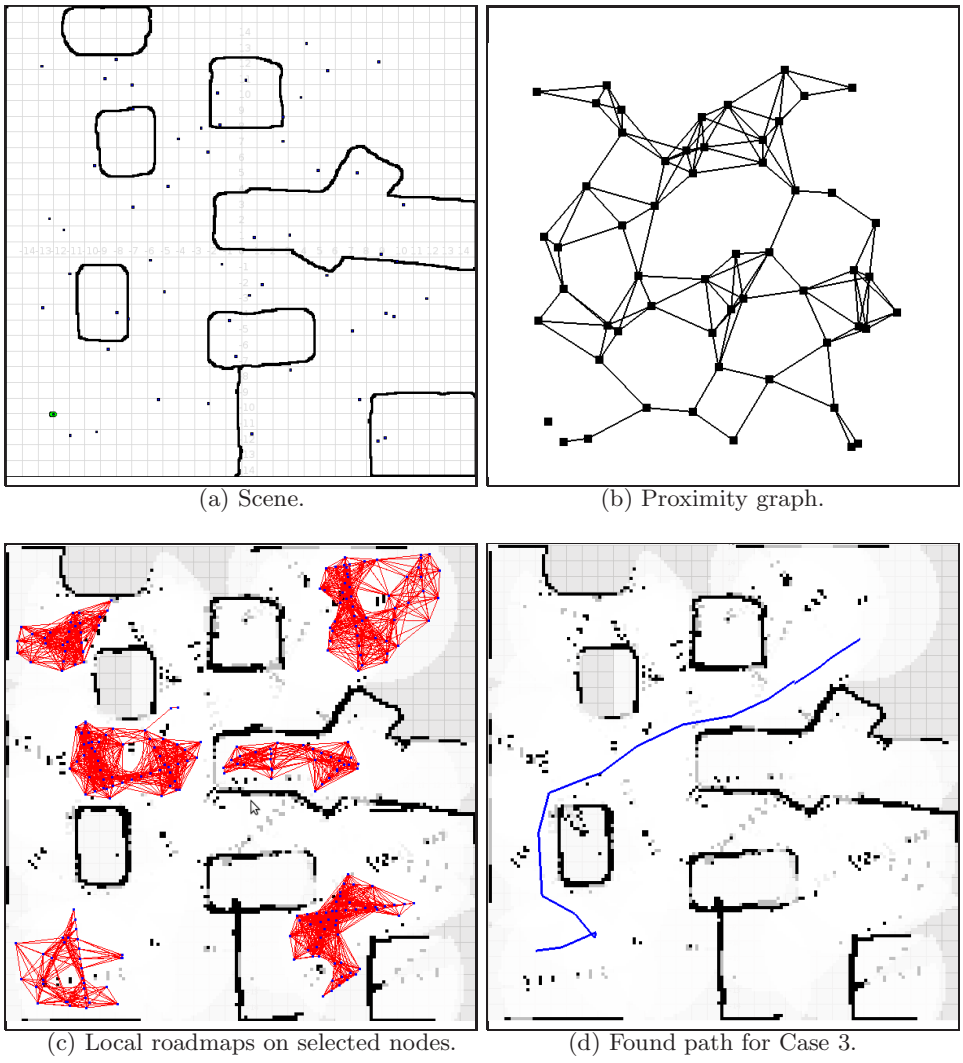


(a) Scene. (b) Proximity graph.

(c) Local roadmaps on selected nodes. (d) Overall roadmap.

(e) Incremental queries in Case 1. (f) Found path for Case 1.

Figure 3.5: Simulation scene for Case 1 and Case 2.



(a) Scene. (b) Proximity graph.

(c) Local roadmaps on selected nodes. (d) Found path for Case 3.

Figure 3.6: Simulation scene for Case 3 and Case 4.

with such *wsn*, and communicate with each other, and they also carry a laser scanner which can detect obstacles (in surveillance application, sensors can be overhead cameras).

In the first scenario, we have a relatively small office-like environment with three rooms, as shown in Figure 3.5(a). Six sensor nodes with the associated proximity graph forming a grid (Figure 3.5(b)) help to navigate robot between rooms. Since every room has only one door facing the corridor, the only way to go from one room to another is through the corridor. The coordinates of the lower-left and upper-right corners are $(-9, -4)$ and $(9, 4)$, respectively. We have 2 cases in this scenario, each corresponding to a different start/goal. In Case 1, the robot moves from $X_s : (-7, -2)$ to $X_g : (7, -2)$; and in Case 2 (used for comparison in Table 3.1), the robot moves from $O_s : (7, -3)$ to $O_g : (7, 3)$, as shown in Figure 3.5(a).

In Case 1, the robot wants to go from the leftmost room to the rightmost one. Most existing distributed planning works, in this case, will give the shortest path in the sensor network (i.e., the dotted line in Figure 3.5(b)), and replan the path when the robot moves closer and detects the blocking walls, because they do not take physical obstacles into account during planning. With our proposed method, each node senses its local environment within the sensing region, then creates a local roadmap. Fig 3.5(c) shows the sensing regions for sensors 1 and 5, and roadmaps therein. We emphasize that each node maintains only its own local roadmap. These local roadmaps are "stitched" together and form an implicit roadmap, as in Fig 3.5(d). In order to compute a feasible path for the robot, a navigation field is propagated over the implicit roadmap, with the goal as the unique global minimum. Then, the robot queries the sensor network for the path and moves along the planned path (Fig 3.5(e) for Case 1), and the final path for Case 1 is shown in Fig 3.5(f).

In the second scenario, we simulated a larger outdoor environment, where polygonal shapes simulate high grounds, or dangers (e.g., minefield), and the sensor network consists of 55 sensor nodes randomly dropped (Figure 3.6(a)). The resulting proximity graph is shown in Figure 3.6(b). The coordinates of the lower-left and upper-right corners are $(-15, -15)$ and $(15, 15)$, respectively. As indicated in Figure 3.6(a), the robot moves from $X_s : (-12, -12)$ to $X_g : (12, 12)$ in Case 3, and from $O_s : (-4, -6)$ to $O_g : (3, -10)$ in Case 4. Figure 3.6(c) shows only selected local roadmaps for clarity. The final path executed by the robot for Case 3 is shown in Figure 3.6(d).

Table 3.1 gives a summary of some important metrics and parameters in the above simulations, including size of the communication network ($|V|$ and $|E|$), size of roadmap

Table 3.1: Simulation Results for Point Robots

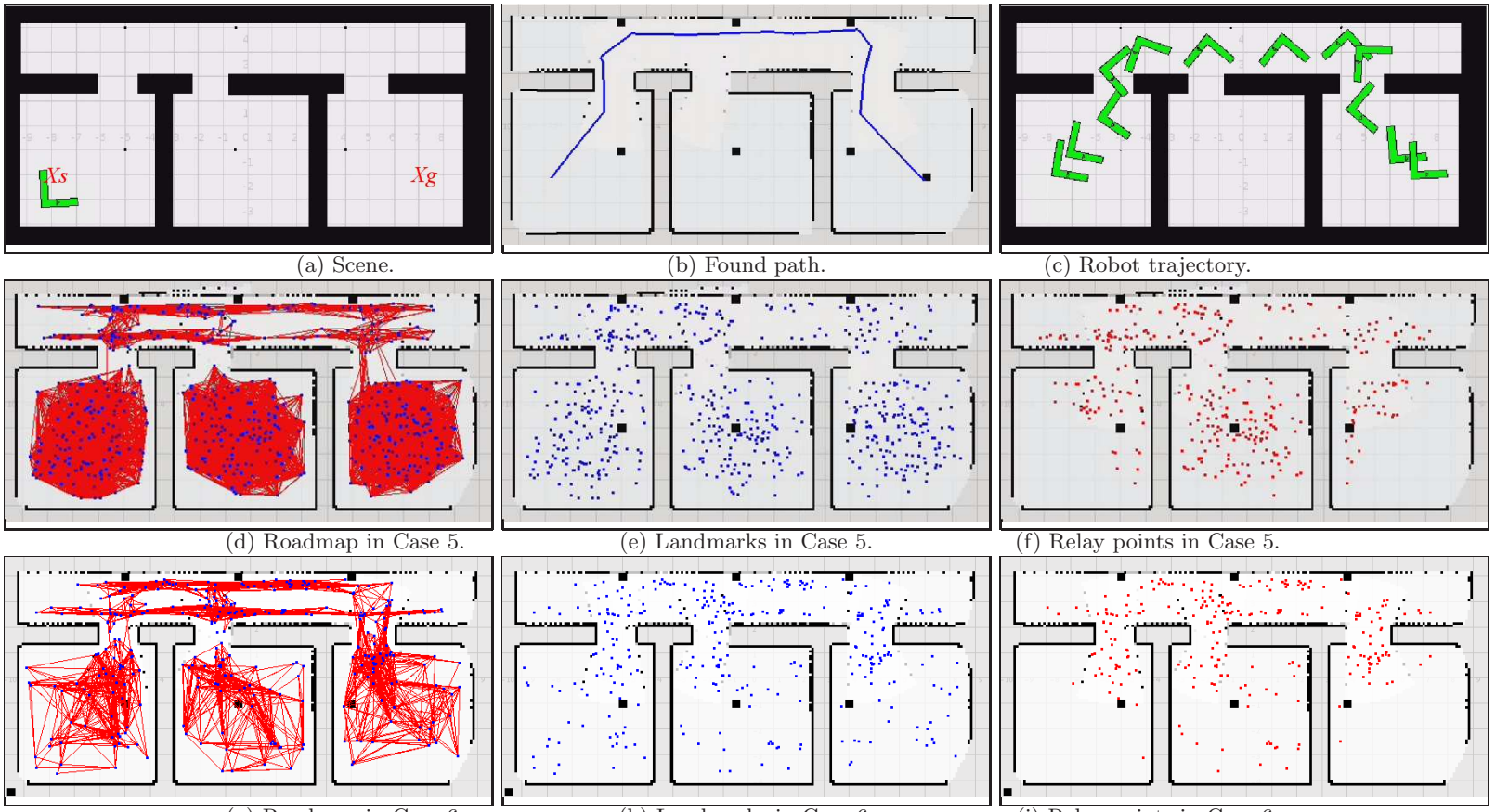
	Case 1	Case 2	Case 3	Case 4
$\|V\|$	6	6	55	55
$\|E\|$	7	7	129	129
$\mathbf{N}_{lm}$	149	139	996	1010
$\mathbf{N}_{msg}$	22	22	333	362
$\mathbf{N}_{cc}$	13706	12283	207579	226095
T_{plan}	5.0	1.5	26.6	4.2
L_{path}	24.5	9.6	43.0	16.7

Table 3.2: Uniform sampling vs. BT sampling in D-PRM for general robots and formations

	Case 5 (Uniform)	Case 6 (BT)	Case 7 (Uniform)	Case 8 (BT)
$\|\|V\|\|$	6	6	55	55
$\|\|E\|\|$	7	7	129	129
$\mathbf{N}_{lm}$	944	232	894	501
$\mathbf{N}_{relay}$	636	242	2840	1601
$\mathbf{N}_{msg}$	133	56	1015	1051
$\mathbf{N}_{cc}$	3291985	222232	4961373	1438760
T_{plan}	8.3	5.9	36.0	39.3
L_{path}	24.9	24.9	40.6	44.1

($\mathbf{N}_{lm}$, number of landmarks), total number of messages ($\mathbf{N}_{msg}$), total number of collision checks ($\mathbf{N}_{cc}$), planning time (T_{plan}, seconds between MSG_TASK sent, and the last $MSG\_REQUEST$ received), and the traveling distance of the robot along the path (L_{path}). The results are averaged over 12 runs, with the roadmap being regenerated in every run.

Note that in D-PRM, planning involves all sensor nodes in the network, so the size of the roadmap and the total number of messages are similar in the same environment (e.g., Cases 1 and 2, or Cases 3 and 4). However, the planning time depends on the distance between start and goal. Since message passing is a major part of the planning, existence of a shorter path results in less intermediate sensor nodes involved in finding the path, and hence less message passing before the MSG_RELAY reaches the robot, hence a shorter planning time.



(a) Scene. (b) Found path. (c) Robot trajectory.

(d) Roadmap in Case 5. (e) Landmarks in Case 5. (f) Relay points in Case 5.

(g) Roadmap in Case 6. (h) Landmarks in Case 6. (i) Relay points in Case 6.

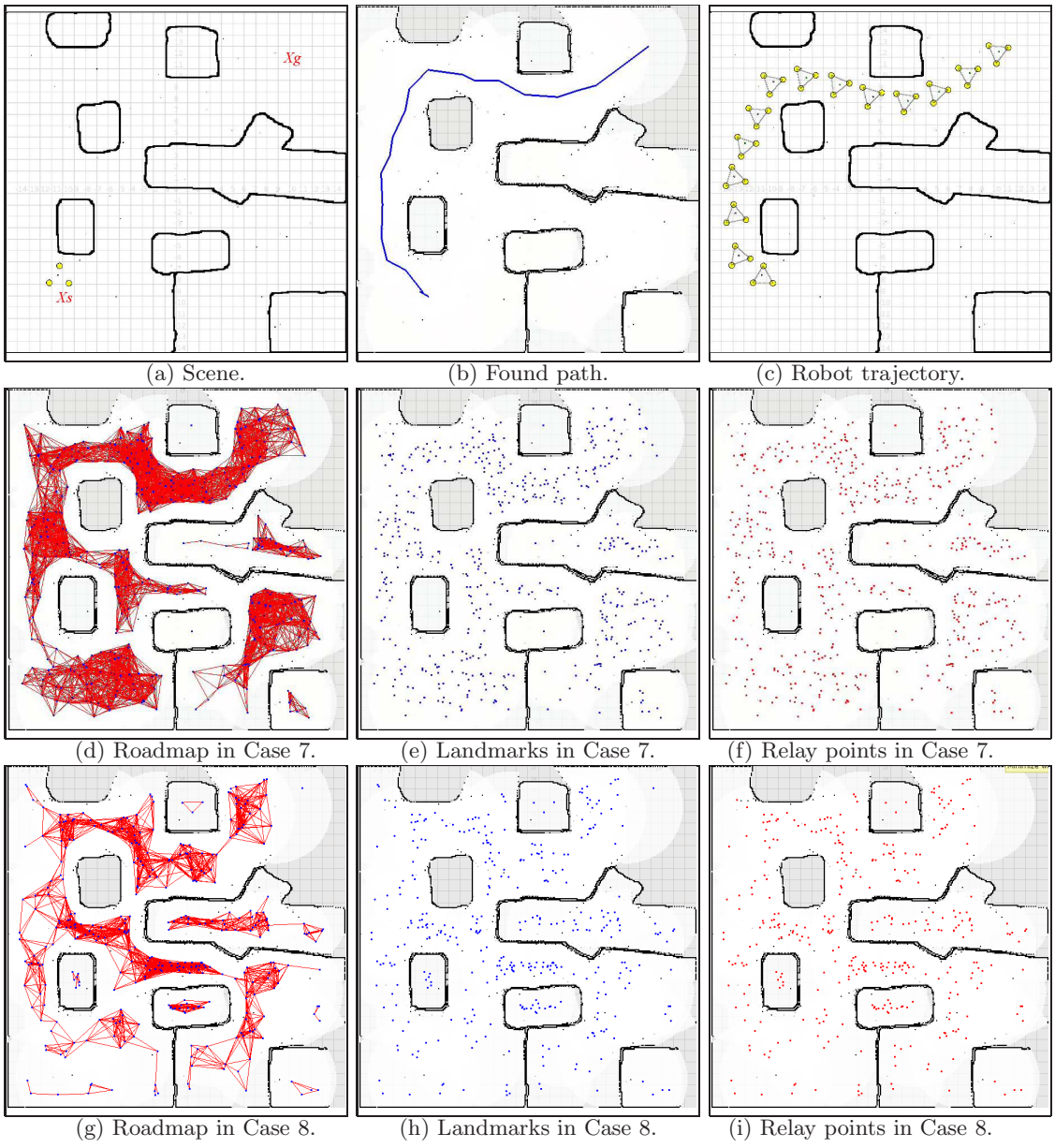
Figure 3.7: Simulation scene for Case 5 and Case 6.

We now show the extension of *D-PRM* for a robot of a nontrivial shape (Cases 5 and 6), and for multiple robots moving in a formation (Cases 7 and 8). Two cases for each extension, using different relay set schemes, one for uniform sampling, and the other for (hybrid) BT sampling. To show the extension for robots with shaped, we reused the scenario and the task in Case 1: the robot moves from $X_s : (-7, -3)$ to $X_g : (7, -3)$ in an office environment, as shown in Figure 3.7(a). The difference is in this case, instead of a circle robot, the robot is L-shaped, as shown in Figure 3.7(a). Two different cases, Cases 5 and 6 use different choice of relay points. Case 5 uses uniform samples as relay points, and Case 6 adopts the BT samples. In two cases, the planned path and execution are similar, as shown in Figure 3.7(b) and (c). With uniform sampling, the distributed roadmap is shown in Figure 3.7(d), Therein landmarks are shown in Figure 3.7(e), and the portion of which used as random relay points is shown in Figure 3.7(f). With bridge-test strategy, resulting roadmap, landmarks and relay points are shown in Figure 3.7(g),(h) and (i), respectively.

In the last set of simulation, Case 7 and 8, we demonstrate our extension of D-PRM to robot formations. As shown in Figure 3.8(a), in an environment similar to the second scenario we simulated a team of 3 robots forming a triangle formation, whose center moves from $X_s : (-12, -12)$ to $X_g : (12, 12)$. The distributed roadmap is shown in Figure 3.8(b), Therein landmarks are shown in Figure 3.8(c), and the portion of which used as relay points is shown in Figure 3.8(d).

Table 3.2 shows the results of using uniform sampling and BT sampling in D-PRM for general robots and formations. With random relay set, D-PRM generates more messages than Cases 1 and 3 where relay set are chosen from a line segment, and this is because exchange of random relay points between sensor nodes requires extra messages. At the same time, with general robots and formations, the problems become much more difficult, which can be seen from increase of the number of collision checks. When comparing the uniform sampling and BT sampling, clearly BT sampling induced fewer landmarks and relay points: by more than 60% between Case 6 and 5, and by 40% between Case 8 and 7.

We also tested *D-RRT* in simulations. One notable difference is that the paths found by *D-RRT* are generally more tortuous than those found by *D-PRM*. For example, Figure 3.9 shows a path found by *D-RRT* for Case 3, and compared to Figure 3.6(d) the path is more zigzag. Such a difference results from the nature of the algorithm: *D-PRM* tries to find an optimal path (in the roadmap) for the robot whereas *D-RRT* tries to find a feasible one. Another observed difference is *D-RRT* consumes $20-50\%$ more messages than *D-PRM*, and



(a) Scene. (b) Found path. (c) Robot trajectory.

(d) Roadmap in Case 7. (e) Landmarks in Case 7. (f) Relay points in Case 7.

(g) Roadmap in Case 8. (h) Landmarks in Case 8. (i) Relay points in Case 8.

Figure 3.8: Simulation scene for Case 7 and Case 8.

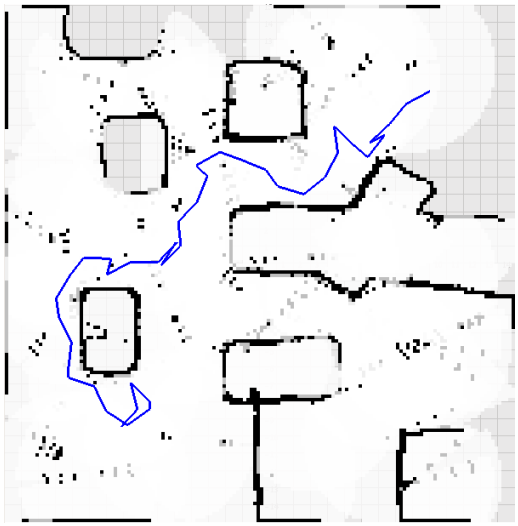


Figure 3.9: Path found by D-RRT.

this is mainly because of the compensation for parallelism as mentioned earlier in Section 3.5.2: In *D-RRT*, in order to maintain overall tree structure, growing forrest in side a sensor node is delayed until some relay points of the sensor node have been reached from the goal; To compensate, more messages are sent to trigger forest growing in neighboring sensor nodes as soon as possible.

Another conceptual difference between *D-PRM* and *D-RRT* is that *D-PRM* is a multi-query method in that the generated roadmap can be reused for different tasks, whereas *D-RRT* is single-query in that the random tree is generated only in the reachable space from a specific goal; therefore when a different goal is specified, *D-RRT* needs to regenerate a brand new random tree, and hence another round of messaging is required. When *D-PRM* propagates the navigation field in the distributed phase starting from a specific goal, it essentially grows a tree structure in the roadmap, and the navigation field only propagates to the reachable space from the goal. The difference from growing such a tree structure to the random tree in *D-RRT* is that *D-PRM* makes use of the pre-generated landmarks where as *D-RRT* needs to regenerate landmarks on the fly, and hence requires more computation. In our application of navigation in sensor network, where energy is scarce resource for sensor nodes and computation and messaging is energy consuming, *D-PRM* is therefore certainly a better choice.

3.8 Summary

In this chapter, we proposed a distributed sampling based planning algorithm for robot navigation in sensor networks. It is particularly applicable to networks where each sensor node is equipped with sophisticated sensors capable of giving a map for its sensing region. To keep communication cost low, there is no global representation of C-space in our algorithm. Instead, each sensor node creates a local roadmap within its locally-sensed environment, and these local roadmaps are "stitched" by a set of relay points, which lie in the overlapping sensing regions of sensor nodes. When the desired goal of the robot is specified, a navigation field is propagated over the implicit roadmap, and gives directions to the desired goal. The robot moves toward its goal by continually querying the sensor network for the directions.

Even though we focus on static sensor networks as navigation assistant, our algorithm can be extended to mobile sensor networks, where mobile nodes can sense the environment while moving, and carry larger maps accumulated along their trajectories. Another direction of future work is to support cooperative collision checking. Our current algorithm for general robots (and formations) assumes that the local environments and relay zones always have large enough free space to contain the entire shape, and collision checks by individual node is enough to find a feasible path. However, there are more difficult and extreme cases. For example, in some cases, because of occlusion, two neighboring sensor nodes can only see a small part of relay zone from different side; in other case, the dimension of robot is simply too large, and it has to span multiple maps of adjacent sensor nodes. In these cases, search for collision-free path requires cooperative collision checking from different sensor nodes.

Chapter 4

Backbone-Based Connectivity Control

4.1 Overview

In this chapter, we study the problem of controlling networked mobile robots while maintaining connectivity among them, i.e., all robots are required to remain connected to each other (either directly, or via other robots). As mentioned before, such connectedness constraint is essential in coordinated and cooperative control. For example, in the emergency rescue application in Chapter 1, rescue robots need to communicate with each other so that they can have better collaboration and coordination throughout the task. More importantly, in many cases connectedness is a necessary condition for stability of the system [60]

We consider a group of n mobile robots, $\mathcal{A} = \{1, 2, \cdots, n\}$, with their positions denoted as $\mathcal{X}(t) = \big\{x_1(t), x_2(t), \cdots, x_n(t)\big\}$. Robots know their own positions, and can communicate with each other within a communication range d_c. We model interaction of the group as a time-varying proximity graph, $G(t) = \big(V(t), E(t)\big)$, whose vertices represent robots, $V(t) \equiv \mathcal{A}$, and edges represent communication links between robots. Let (i, j) denote an edge between robots i and j, thus,

$$(i, j) \in E(t) \Longleftrightarrow d(i, j) = \|x_i(t) - x_j(t)\| \leq d_c$$

Each robot, i, has its own goal configuration, x_i^g, so $\mathcal{X}_g = \{x_1^g, x_2^g, \cdots, x_n^g\}$. The problem is how to maneuver the group of robots to reach their respective goals, with the constraint that $G(t)$ remains connected throughout the task.

As we have reviewed in Chapter 1, this is a difficult problem, and existing approaches to the problem either fail to handle general cases, or result in high communication costs. On the other hand, in Chapter 2, we showed that communication backbone provides a good representation of network topology, and a good tool to scale down the network. Inspired by that, in this chapter, we propose a distributed Backbone Based Connectivity Control (BBCC) framework for the problem. It first maintains a connected backbone, by maintaining existing connections in the backbone; and then for a non-backbone robot, one of the backbone robots is chosen as leader, and connection to the leader is maintained. The proposed BBCC framework works as follows:

1. With the communication graph, $G(t)$, BBCC first constructs the backbone, $G_B(t) = \big(V_B(t), E_B(t)\big)$, where $E_B(t) \subseteq E(t)$ and $V_B(t) \subseteq V(t)$, in a distributed fashion. The backbone consists of backbone robots and connections among them, and these robots and connections are critical for the system connectivity. Figure 4.1 shows an example of backbone-based hierarchy. There are 3 backbone robots: A, B, and C, where A, B are clusterheads, and C is a connecting gateway. A_i's are non-backbone robots associated with A, and A along with A_i's forms one cluster. B_i's are non-backbone robots associated with B, and they form the second cluster.

2. Based on the constructed backbone and respective goals, motion of each robot, $\dot{x}_i(t)$, is determined. For the backbone robots, we formulate the backbone as a constraint graph, and motion control is derived (via a judicious use of potential fields) such that every connection in backbone is maintained; and for non-backbone robots, we use, loosely speaking, a sort of leader-follower control (explained later in Section 4.3) with the associated backbone robot as the leader. In Figure 4.1, to guarantee connectivity, connections (A, C), (B, C) are preserved, and within its own cluster, A is the leader and $A_i's$ maintain connections to A. Same for B and B_i's.

3. After robots move with constraints of backbone for a period of time, T, communication graph $G(t)$ may have changed, and thus the backbone $G_B(t)$ is updated, and robots move with the new backbone as constraint graph.

We stress that BBCC is a distributed scheme, there is no central global representation of $G(t)$, $G_B(t)$, and update of $G(t)$, $G_B(t)$ and $\dot{x}_i(t)$ are done locally. However, some synchronization is necessary to make the proposed scheme work correctly as explained in

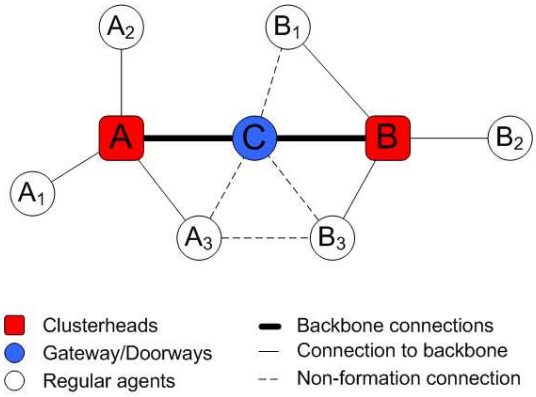


Figure 4.1: Backbone-based hierarchy.

Section 4.4. With communication backbone being a general and effective representation of system connectivity, BBCC provides a more flexible (in that it makes no assumption on system topology) and more efficient (in terms of communication cost, since it uses only local 2-hop neighbor information) solution than existing ones.

Generally speaking, backbone robots have less freedom than non-backbone robots, since a non-backbone robot is only constrained by its single leader whereas a backbone robot is constrained by all its backbone neighbors. We can look at it this way: backbone robots are mainly responsible for keeping core connectivity while non-backbone robots are mainly responsible for reconfiguration, by switching to different backbone robots from time to time. To compute the backbone, we again adapt TMPO algorithm for backbone construction and real-time update, and make the constructed backbone suitable for motion planning/control purposes. The heuristics to compute backbone robots, and for a non-backbone robot to switch to another backbone robot is based on the motion planning objective: getting closer to their goals.

4.2 Backbone based hierarchy

4.2.1 Robot priority for connectivity control

Definition of priority depends on tasks of the system, and performance objectives of concern. [10] defined priority as a function of energy and mobility in order to achieve longer lifetime, and more stable backbone. In Chapter 2, we brought into account distance to dangers in its

priority while taking out mobility, for safe navigation among static sensor networks. Here, we define priority for motion planning purpose. In this problem, robots move at similar speeds toward their respective goals, and energy consumption induced by communication is much smaller compared to that by mobility. Therefore, we do not include mobility (velocity) and energy factors in our priority definition, and instead we define priority as a function of goal (which is the key objective of motion planning), such that a robot closer to its goal will be assigned a higher priority, and hence more likely to be elected as a backbone robot. More specifically, let $P(i)$ be the priority of robot i.

$$P(i) = P_T(i) \bigoplus P_I(i)$$

where $\bigoplus$ is bit-concatenation operation, $P_I(i)$ is a unique index (e.g. robot ID) of the robot in order to make the priority unique across the network, and $P_T(i)$ is the priority based on the given task. Note that P_T occupies the higher bits of P, thus a robot with higher P_T always has a higher priority. Since each robot has its own goal, we can define $P_T(i)$ as a function of current distance to its goal:

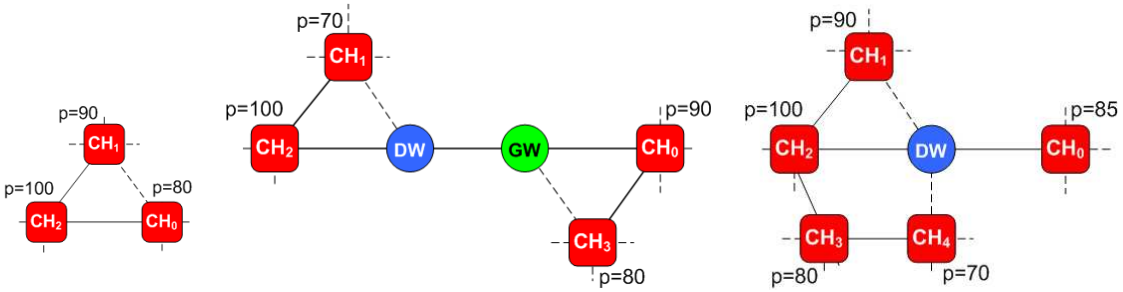
$$P_T(i) = \begin{cases} MAX = 1/d_e, & \text{if } \|x_i - x_i^g\| \leq d_e \\ 1/\|x_i - x_i^g\|, & \text{Otherwise} \end{cases} \tag{4.1}$$

where d_e is a small "end-game" threshold.

Similar to the backbone construction procedure in Section 2.3, a robot decides whether or not to be a backbone robot (CDS member) based on a knowledge of its 2-hop neighbors, and their priorities, and there are three types of robots in the backbone, **clusterheads (CH)**, **doorways (DW)**, and **gateways (GW)**. We collectively call clusterheads, doorways, gateways as backbone robots, and all others as non-backbone robots. The backbone construction procedure in this section is very close to that in Section 2.3, except that we remove the condition regarding zero-priority ((E.1) in Section 2.3.2).

4.2.2 Backbone with less connections

The aim of the backbone construction procedure above is to elect as few robots (i.e., vertices) as possible to be members of backbone. However, it is not concerned about number of connections (i.e., edges) within backbone, and in fact may generate redundant connections. Redundant connections in backbone impose unnecessary constraints for the planned motion, since we use backbone as the constraint graph for motion control, therefore we remove



(a) (b) (c)

Figure 4.2: Refine connections. Dotted edge are removed from backbone for less constraints.

redundant connections in the backbone as much as possible. However, such refinement should rely on local information only so that it does not introduce significant computational and communication overhead. For instance, we avoid constructing the minimum spanning tree, which reduces connections to minimum, but requires global message exchanges and hence results in high communication cost. Rather, we use only local rules, and apply the following to reduce the number of connections (while ensuring connectivity).

1. Refine connections between clusterheads: In general a clusterhead (directly) connects to clusterheads in 1-hop neighborhood, however to avoid redundant connections where three clusterheads make a triangle, as shown in Figure 4.2(a), the connection between two clusterheads with lower priority will be removed from backbone connections.

2. Refine connections between doorways and clusterheads: A connection between a doorway and a clusterhead is removed if there is another neighboring clusterhead with higher priority connected to this doorway, as shown in Figure 4.2 (b).

3. Refine connections between gateways and clusterheads: When more than two neighboring clusterheads share a gateway, we remove connections to the lower-priority clusterheads. A connection between a gateway and a clusterhead is removed if there is another neighboring clusterhead with higher priority connected to this gateway. As shown in Figure 4.2 (c), GW is a gateway for $\{CH_0, CH_1\}$, $\{CH_0, CH_2\}$, $\{CH_0, CH_3\}$ and $\{CH_0, CH_4\}$; however, since CH_2 has the highest priority among other clusterheads, connections from GW to CH_1, CH_3, CH_4 are removed. Moreover, the whole path between $\{CH_1, CH_4\}$ is removed. This is because CH_1 and CH_4 are connected to two adjacent clusterheads with higher priority.

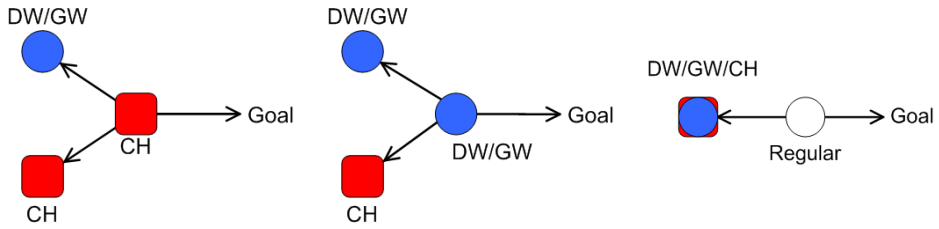


Figure 4.3: Motion constraints to robots.

4.3 Motion control with backbone

Our control scheme uses the backbone derived above to impose motion constraints on individual robots. Simply put, the motion of a backbone robot is constrained by its neighbors in backbone, so that it remains within the communication range of the neighbors; and the motion of a non-backbone robot is constrained by a chosen backbone robot (e.g., the corresponding clusterhead) that it is associated with, as shown in Figure 4.3. We use potential functions for connectivity maintenance and goal achievement, and different potential functions are defined for different purposes. Furthermore, backbone robots and non-backbone robots have different potential functions.

4.3.1 Connectivity potential

First we define a general attractive potential between i, and its neighbor j:

$$U_{i,j}^{C} = \begin{cases} \frac{1}{2}\left(\frac{1}{d_c - d_{ij}} - \frac{1}{d_c - d_L}\right)^2 & d_L \leq d_{ij} \leq d_c \\ 0 & d_{ij} \leq d_L \end{cases} \tag{4.2}$$

where $d_{ij} = \|x_i - x_j\|$, d_c is the communication range, and d_L is a predefined distance of influence of connectivity. This general potential is then used to maintain connectivity constraints between different types of nodes as follows.

For a non-backbone robot, connectivity is maintained by following a leader in backbone. Thus, its connectivity potential is defined with respect to the chosen leader. Denote L_i as the leader for a non-backbone robot, i. We choose L_i as the neighboring *backbone robot* that is closest to current robot's goal, x_i^g, i.e.,

$$L_i = \arg \min_{l \in G_B(t)} \left\{ d(x_i^g, l) \middle| (l, i) \in G \right\} \tag{4.3}$$

Note that there are alternative criteria for leader election. For example, instead of using a task-based criterion as above, we can choose a leader based on priority (e.g. the neighboring *clusterhead* with the highest priority). Once a leader is selected, the connectivity potential is defined as,

$$U_i^C = U_{i,L_i}^C \tag{4.4}$$

For a backbone robot, the total connectivity potential of i is defined as:

$$U_i^C = \sum_{j \in N_b(i)} U_{i,j}^C + \sum_{k \in N_f(i)} U_{i,k}^C \tag{4.5}$$

where $N_b(i) = \{j | (i,j) \in E_B\}$ is the set of neighbors of i in the backbone, and $N_f(i) = \{k | i = L_k, k \notin V_B\}$ is the set of followers of backbone robot i, and can be computed from two-hop neighbor information. It is worth noting that our leader-follower control scheme is not strictly a leader-follower scheme, since the leader's potential is affected by motion of followers. This mitigates the known problem of poor disturbance (e.g. due to obstacles) rejection in leader-follower formation control scheme, which may cause connections between the leader and its followers to be broken.

4.3.2 Task potential

Each robot (backbone or non-backbone), i, has its own goal, x_i^g, which imposes an attractive potential on robot i, denoted by U_i^T, also called the task potential, and is given by:

$$U_i^T = \frac{1}{2}\left\|x_i - x_i^g\right\|^2 \tag{4.6}$$

4.3.3 Collision potential

We define a general repulsive potential for collision avoidance between robots, as well as for obstacle avoidance.

$$\Phi(d) = \begin{cases} \frac{1}{2}\left(\frac{1}{d} - \frac{1}{d_s}\right)^2 & d < d_s \\ 0 & d \geq d_s \end{cases} \tag{4.7}$$

where d is a distance to collision (with other robots or obstacles), and d_s is a predefined safe distance threshold. For obstacle avoidance, $U_i^O = \Phi(d_i^o)$, where d_i^o is distance from robot i to the closest obstacle. For collision avoidance between robots i, and j, $U_{i,j}^A = \Phi(d_{ij})$, and we differentiate between backbone and non-backbone robots in collision avoidance. A non-backbone robot needs to avoid collision with all other robots, backbone and non-backbone

ones. A backbone robot has privilege over non-backbone robots, and hence it only considers avoiding collision with other backbone robots. So,

$$U_i^A = \begin{cases} \sum_{(i,j)\in G} U_{i,j}^A & \text{if } i \text{ is a non-backbone robot} \\ \sum_{(i,j)\in G_B} U_{i,j}^A & \text{if } i \text{ is a backbone robot} \end{cases} \tag{4.8}$$

4.3.4 Motion of robots

Thus, the composite potential of robot i is given by:

$$U_i = \gamma \cdot \left(U_i^O + U_i^A\right) + \alpha \cdot U_i^T + \beta \cdot U_i^C \tag{4.9}$$

where α, β and γ are predefined scalar weights. Motion (velocity) of robot i is thus given by following the negated gradient of U_i:

$$\dot{x}_i = -\nabla U_i \tag{4.10}$$

Note that this first order model does not explicitly limit acceleration, however, in practice, it is not an issue unless the mobile robots are moving at high speeds.

4.4 Discussions

4.4.1 Synchronization

In our algorithm, robot motion is derived based on backbone connections, so the motion derivation should be done only after the backbone computation is "settled down", and hence synchronization is needed. We use timers for synchronization purpose, robots are scheduled to update backbone every (predefined) T seconds. In our simulations, robots simply stop moving every T, wait for T_{BB} until the new backbone is stably computed, and then resume moving. In practical applications, the stopping can be relaxed to $\dot{x}_i T_{BB} \ll d_c$, i.e. robots move fairly slowly so that the distance travelled in time T_{BB} is much less than the communication range.

4.4.2 Correctness

Lemma 2. *If $G(t)$ is connected, $G_B(t)$ computed by BBCC is connected.*

This follows directly from the original TMPO algorithm. As mentioned in Section 4.2, we have three main modifications to the original TMPO algorithm. Our definition of priority does not change uniqueness of the priority assignment, based on which the elected clusterheads are guaranteed to form a dominating set. Our second modification, that of bidirectional backbone connections, does not change the result of backbone election either, but rather just adds more information associated with the elected connections. The last modification for reducing backbone connections does change the number of connections in the backbone, however it is straightforward to prove that the connection refining procedure in Section 4.2.2 does not change existence of a path between two clusterheads, even though the refinement may increase the path length between two robots. This is because a connection is removed only if there exists another path connecting the two robots (but with higher priority robots in the path).

Correctness of the motion control method based on potential field can also be proved. We have defined continuous potential functions for connectivity maintenance and obstacle avoidance. For example, for the connectivity potential function defined in Eq.(4.2), it blows up when the a backbone connection is about to break, i.e., $\lim_{d_{ij}\to d_c^-} U_{i,j}^C = \infty$. Similarly, when collision occurs, $U_i^O = \infty$, or $U_i^A = \infty$. Thus we have,

Theorem 4. *If all robots are collision free and connected at t_0, then BBCC is guaranteed to maintain the system to be collision free and connected for $t \geq t_0$.*

Proof. Based on Lemma 2, we only need to prove that $G(t)$ remains connected between two backbone updates. Assume the backbone is updated at time t_0, and then at $t_0 + T$. We prove that $G(t)$ remains connected and collision free for $t \in (t_0, t_0 + T)$. Denote X as the stack vector of all robot position vectors, i.e., $X = (x_i^T, \cdots, x_n^T)^T$, and define an overall potential for the entire system as,

$$U(X(t)) = \sum_{i=1}^{n} \left(\alpha U_i^T + \frac{1}{2}\beta U_i^C + \gamma\Big(U_i^O + \frac{1}{2}U_i^A\Big)\right)$$

Note U is not simply summation of U_i (defined in Eq.(4.9)), because in $\sum_i U_i$ connectivity and collision potentials between two robots are counted twice, and that is why there is the $\frac{1}{2}$ factor with them. With this definition, $\dot{X} = (\dot{x}_1^T, \cdots, \dot{x}_n^T)^T = -\nabla_X U = (-\nabla^T U_i, \cdots, -\nabla^T U_n)^T$, where $\dot{x}_i = -\nabla U_i$ as in Eq.(4.10), This means our control strategy does not increase the potential U during the period. i.e.,

$$U(X(t)) \leq U(X(t_0)) \qquad \forall t \in (t_0, t_0 + T)$$

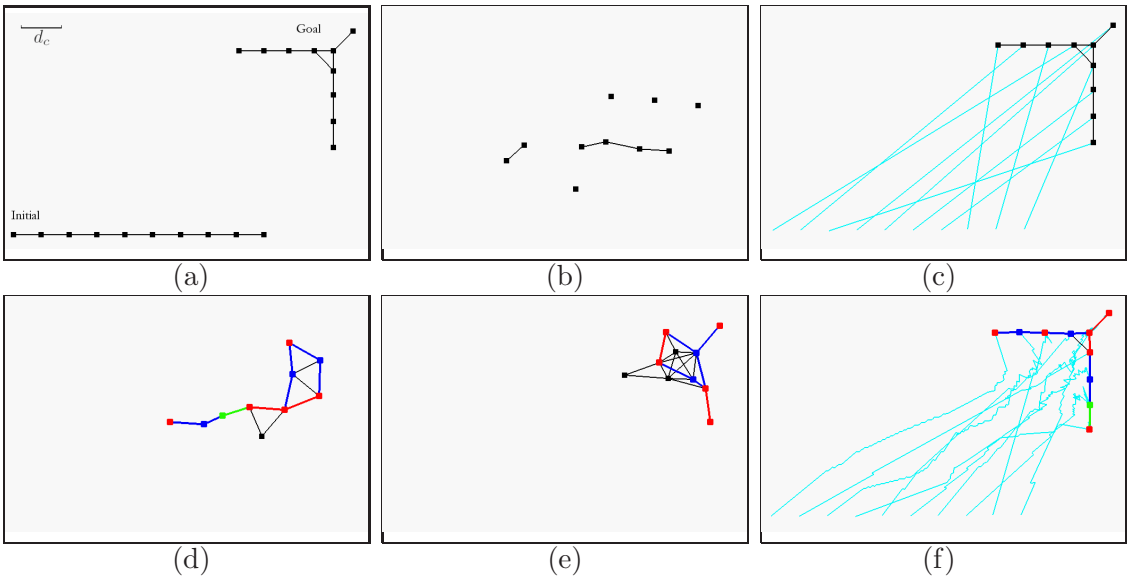


Figure 4.4: Comparison of motion without and with connectivity constraints. (a) Initial and goal formations. (b) Without connectivity constraints, robots become disconnected. (c) Trajectories for all robots without connectivity constraints. (d)-(e) With connectivity constraints, robots remain connected. (f) Robot trajectories with connectivity constraints.

Clearly, if collision occurs or connectivity constraint is violated, $U(t)$ reach infinity. As the system is collision free and connected at t_0, $U(X(t_0)) < \infty$, it remains collision-free and connected during $(t_0, t_0 + T)$. □

4.4.3 Local minima

We model the goal achievement and connectivity maintenance as attractive potentials, and obstacle avoidance as a repulsive potential, then robots follow the negative gradient of the composite potential. As expected, such composite potential may have local minima, and some robots or the entire team may get stuck. One can conceivably construct navigation functions, which have no local minima, but this generally requires global knowledge (which is not available in our problem) and tuning of parameters can be difficult especially for dynamic environments. Furthermore, connectivity constraint aggravates the local minima problem. In the next chapter (Chapter 5), we study possible strategies to escape local minima, especially those ones due to connectivity constraints.

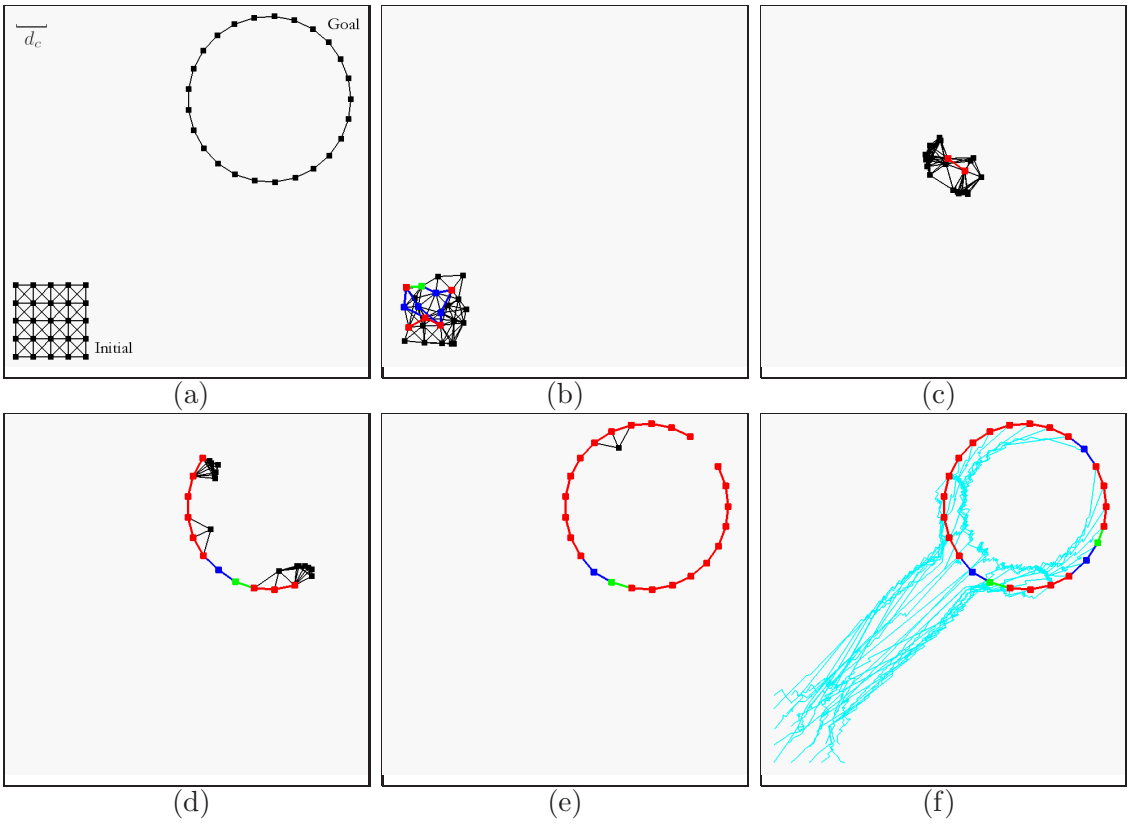


Figure 4.5: Motion with connectivity constraints. (a) Initial (complete graph) and goal formations. (b)-(e) Snapshots along the path. (f) Trajectories for all robots.

4.5 Computer simulations

We now present simulations to show the effectiveness of the proposed BBCC method. We simulate a team of robots moving in an arena of $120m \times 120m$, from an initial formation to a goal formation. To show the generality of the approach, we use different initial and goal formations ranging from a simple path graph to a complete graph, from a grid graph to a circular graph, and labeling (vertex indices) of these formations is random and independent from simulation to simulation. Unless otherwise indicated, the number of robots is 10, and communication range between robots is set to 15m. For the BBCC method, we chose $T = T_{BB} = 2$ second, and the maximum speed of robots is $0.5m/s$.

4.5.1 Without connectivity constraints

We first use potential field (similar to Eq.(4.9)) to move the formation, but without taking into account connectivity constraints (i.e., no connectivity potential term ∇U_i^C). The initial and goal formations are shown in Figure 4.4(a). Figure 4.4(b) shows that robots become disconnected along the trajectory (in Figure 4.4(c)).

4.5.2 With connectivity constraints

Figure 4.4(d)-(f) show that, for the same problem as above, taking connectivity constraints into account ensures robots remain connected along the trajectory. In the snapshots, backbone is shown in color: red, green and blue nodes are clusterhead, doorway and gateway robots respectively; red, green and blue connections are connections between clusterheads, between clusterhead and doorway, and between gateway and clusterhead/doorway respectively; and non-backbone robots and connections are shown in black. Another simulation is shown in Figure 4.5. The system has 25 robots, and communication range is set to 8m. In both cases, the proposed BBCC is able to maneuver the team into the final formations without being disconnected. Since in both cases, the initial and goal formations are dramatically different, robots have to remain fairly close while moving in order to reconfigure without disconnection. As mentioned earlier, in general backbone robots have less freedom than non-backbone robots, since a non-backbone robot is only constrained by it single leader whereas a backbone robot is constrained by all its backbone neighbors. Intuitively, backbone robots are mainly responsible for keeping connectivity while non-backbone robots are mainly responsible for reconfiguration. If there is no "better" (in the sense of Eq.(4.3), i.e., closer to goal topology) connection for a non-backbone robot, it sticks to current backbone robot (i.e., L_i in Eq.(4.3) remains the same) until a better one comes in. So robots tend to move closer to one another looking for "better" ones. As robots move closer to one another, the system connectivity increases and consequently the size of the backbone (i.e., number of robots and connections in backbone) decreases and hence reconfiguration freedom increases. When a non-backbone robot approaches its goal, its priority increases quickly (as in Eq.(4.1)), and it is more likely to became a backbone node. As more and more non-backbone robots become backbone robots and reconfigure to goal topology, remaining non-backbone robots can seek their goals while remaining connected to the current backbone.

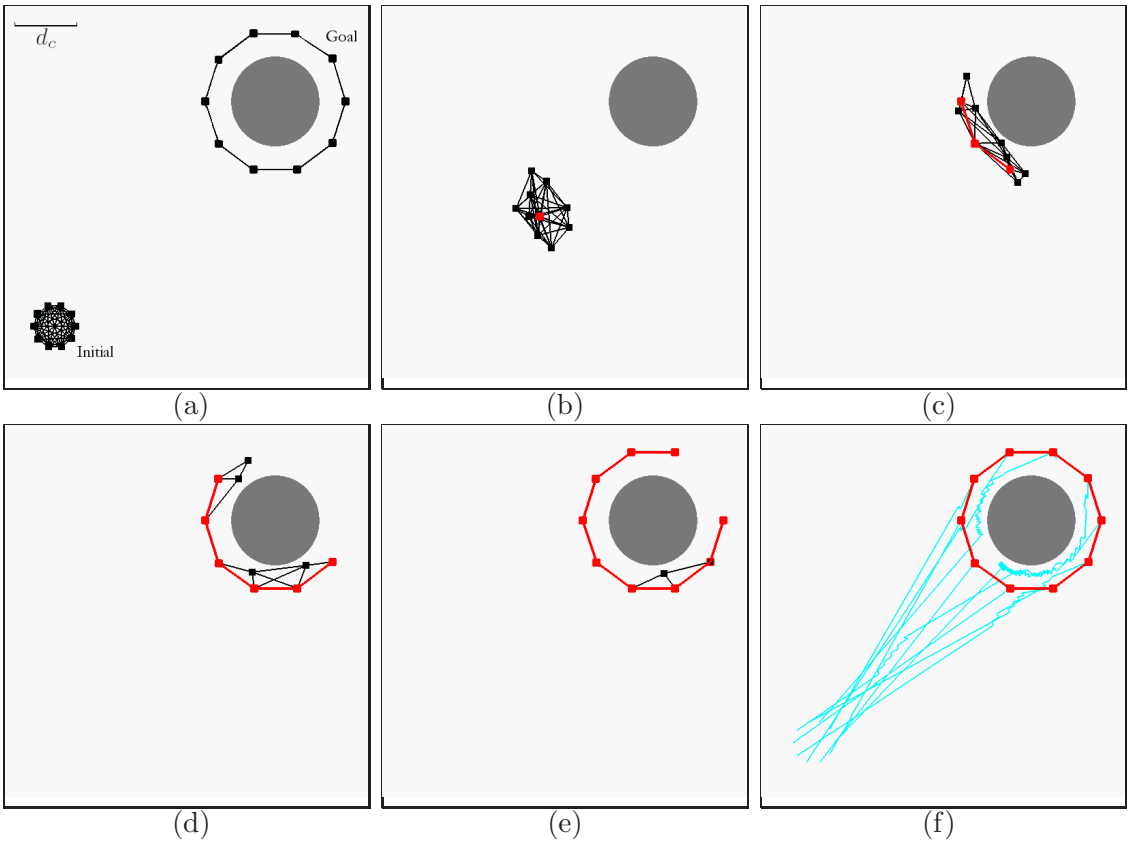


Figure 4.6: Motion with connectivity constraint in the presence of obstacles. (a) Initial and goal formations. (b)-(e) Snapshots along the path. (f) Trajectories for all robots.

4.5.3 With obstacles

In Figure 4.6, robots move in an environment with obstacles. Robots start with a complete graph, and the goal formation is a circular graph wrapping around an obstacle. The proposed BBCC is able to maneuver the team into the goal formation. However it can get stuck in local minimum. These issues are explored in detail in Chapter 5.

4.6 Summary

In this chapter, we proposed a novel distributed BBCC scheme for cooperative control with connectivity constraint. BBCC uses a two level hierarchy based on communication backbone of mobile robots as the key to maintaining connectivity. Robots in the team are categorized

into backbone robots and non-backbone robots, and thus connectivity is maintained at two levels. For backbone robots, backbone connections are maintained by imposing them as constraints over robots' movement. For non-backbone robots, with certain backbone robots chosen as leaders, a leader-follower type scheme is used to maintain their connections to backbone. At the same time, non-backbone robots search for reconfiguration by choosing different leaders. Unlike many existing approaches to the problem, BBCC does not make any assumption on system topology, and deal with arbitrary initial and goal formations. Moreover, it is a distributed method using only two-hop neighbor information; and hence has low communication cost. Computer simulations have been done to verify the proposed framework, where BBCC is able to find paths while maintaining connectivity. The algorithm is able to handle obstacles, although in some situations with obstacles, the team got into local minima. In next chapter, we will look into local minimum issues that arise in the problem.

Chapter 5

Distributed Strategies for Local Minimum Escape

5.1 Overview

In previous *BBCC* framework, when computing robot motion, we model the goal achievement and connectivity maintenance as attractive potentials, and obstacle avoidance as a repulsive potential. The robots then follow the negated gradient of the composite potential. As expected, such composite potential may have local minima, and some robots or the entire team may get stuck. Our empirical observations motivate a classification of these local minima based on different underlying causes. While the interplay between these underlying causes is complex, our empirically based classification helps us suggest strategies to cope with these local minima.

In this chapter, we present the classification and corresponding strategies in details. We first classify local minima into three different categories: Type-I (Regional obstacle-induced local minimum), Type-II (Individual connectivity-induced local minimum), and Type-III (Structural compound local minimum), and different types imply different natures of the minima. In the first category, the robot may be able to escape the minimum merely by simple local strategies (e.g.,random walk), whereas in latter two categories, a robot needs help from others in order to make global decisions for local minimum escaping. Corresponding to these different types, three respective strategies are used to tackle these local minima: *Random Walk*, *Backbone-based Navigation*, and *Backbone-based Leader-following*.

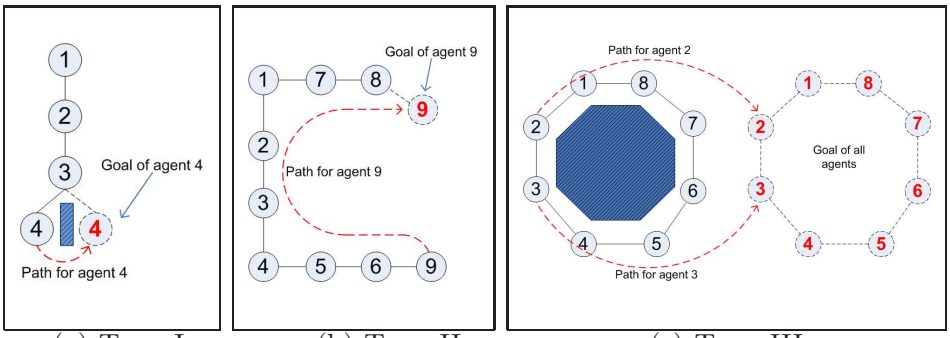
5.2 Local minimum detection and classification

To detect local minima, we keep track of a robot's trajectory. We save the last K positions of the robot, and compute their variance. If the variance is smaller than a threshold, indicating the robot does not progress toward it goal, then this robot is deemed to be in a local minimum. We further categorize local minima into three different levels:

1. **Type I: Regional obstacle-induced local minimum**. This is mainly because of obstacles, and connectivity constraints are not the limiting factor (i.e., they are easily maintained in that region). Intuitively, this is what may occur for a single robot, and local strategies that do not involve other robots, such as random walk, suffice to escape such local minima. Figure 5.1(a) shows a scenario, where a small obstacle blocks robot 4 from its goal, and a random walk (or another local strategy) should be enough for the robot to get around the obstacle and reach its goal. More complicated local minima, where a robot needs help from others in order to make global decisions for escaping the local minimum, are further classified into Type II or III.

2. **Type II: Individual connectivity-induced local minimum**. This type of local minimum is normally caused by connectivity constraints, and it may have weak interaction with local obstacles. In such a scenario, only a single robot (or small number of robots) is in local minimum. Simple local strategies may not be possible for the robot(s) to escape, but it is possible to resolve the minimum by moving individual robots without major connectivity reconfiguration. Figure 5.1(b), shows one example, where connectivity constraint to robot 6 prevents robot 9 from reaching its goal. To escape, global path planning is needed (only) for robot 9 to break out of its current connectivity constraint with robot 6, and successively switching to other robots in order to reach its goal while maintaining connectivity to the network along the shown path.

3. **Type III: Structural compound local minimum**. This type of local minima is also caused by connectivity constraints, but has strong interaction with obstacles and other robots. In such a case, a number of the team members are stuck into local minima, and even worse these local minima are coupled to one another. It is not possible to resolve these minima individually as in the previous case, instead a major connectivity reconfiguration of the entire mobile network is needed in order to move



(a) Type I. (b) Type II. (c) Type III.

Figure 5.1: Categories of local minimum. Dotted nodes are goal positions of corresponding robots

out of these minima. Figure 5.1(c) shows such an example. Clearly in this case, from each robot's local point of view, all edges in the initial cycle formation are critical in maintaining connectivity. In order to achieve the goal formation, some edge has to be broken ($(2, 3)$ in the shown example); and making decision as to which edge to break needs to involve all robots.

5.3 Heuristics for classification and escaping

Although given examples illustrate different types of local minima, the distinctions may not be as clean cut in reality. There is ample scope for investigating these further in a more formal manner. In practice, our overall scheme for classifying and escaping these local minima is shown Figure 5.2. It starts with core BBCC, as introduced in the previous Chapter, and a local minimum is detected when one or more robots have variance of the last K positions smaller than a certain threshold. When a robot detects a local minimum, it first assumes the minimum is a Type-I minimum, and uses simple local strategies (we used random walk for a certain period of time; other local strategies, such as bug algorithm [57], could be used as well.) to escape. If a local minimum persists for M iterations, the minimum is either Type-II or Type III minimum, global assistance is needed in order for the robot to escape. Therefore, at this stage, all robots will be notified of the minimum. When a robot receives such a notification, it stops for further determination (Type II or III) and resolution of the local minima by the entire network. This determination and resolution

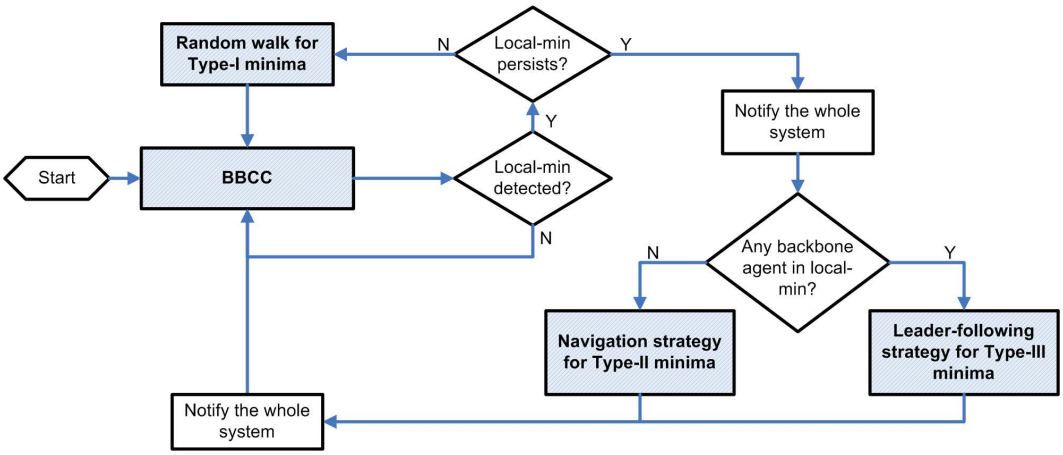


Figure 5.2: Overall scheme.

relies on the hierarchical structure of the network.

The difference between Type II and Type III minima is whether the robot in local minimum is a backbone robot or not. If it is not a backbone robot, it implies a type II minimum; if it is a backbone robot, it implies a type III minimum. The rationale is as follows. Recall that in our hierarchical scheme the backbone is a connected dominating set of the network, and it captures the system topology in that all other robots are just one hop away from the backbone. Backbone robots are key to maintaining connectivity, and non-backbone robots reconfigure around backbone robots. So a backbone robot in local minimum indicates a problematic topology and need a systematic reconfiguration. For Type-II minimum, since no backbone robot is in local minimum, we use the backbone as a navigation roadmap to navigate the (non-backbone) local minimum robot to get as close as possible to its goal. We call this strategy *Backbone-based Navigation*. To escape a Type III minimum, all robots should move closer toward each other to increase the system connectivity, and thereby reduce the number of backbone robots, increase the number of non-backbone robots and increase reconfigurability. We implement a *Backbone-based Leader-following* strategy, which constructs a spanning tree of the stationary backbone in a way such that the root robot of the tree is the robot that is the farthest away from those robots deemed to be at local minima (and hence most likely to be free from local minimum), and acts as a leader. All other robots follow this leader according to the tree hierarchy and move toward their parents instead of their ultimate goals.

It is possible that more than one robot are simultaneously at their respective local minima, and these are handled as follows. A Type-III minimum has the highest priority to be solved, followed by a Type-II, followed by a Type-I minimum. Each strategy handles multiple local minima of the same type. For multiple Type-I minima, robots can random walk at the same time. If there are multiple Type-II minima (but no Type-III minimum), the Backbone-based Navigation strategy creates routes to each robot that is in local minimum, i.e., they are simultaneously handled. The Backbone-based Leader-following strategy naturally handles multiple Type-III minima by selecting the leader to be farthest away from *any* local minima.

5.4 Backbone-based escaping strategies

Since Type-II and Type-III minima need global communication to escape, a robot in either type of local minimum notifies all other robots by sending out a *MSG_DETECT* message, with its information, such as its current and goal position. All robots maintain a list of robots in local minimum, $lmList$. When a robot receives the *MSG_DETECT* message, it stores the corresponding local minimum information into $lmList$, and forwards the message (if it is the first time). At the end, given perfect communication, all robots will have the exact same list. To initiate Backbone-based Navigation or Leader-following escaping procedure, upon receiving the *MSG_DETECT* message, all robots stop moving and thus backbone become stationary; To reflect the connectivity of the stopped network, robots stop updating the backbone during escaping.

5.4.1 Backbone-based Navigation strategy

For Type-II minima, since no backbone robot is in local minimum, we use the stationary backbone as a navigation roadmap to navigate the local minimum robot to get as close as possible to its goal. The backbone-based robot navigation scheme proposed in Chapter 2 for single robot navigation in a static sensor network is ideally suited to be used a local minima escape strategy. Recall that the scheme computes a shortest path from current robot location to a given goal sensor node based on the backbone of the static sensor network. Here, we use a simplified version of the scheme, since the backbone is already constructed by *BBCC*, we only adopt the second (path planning) phase of the scheme (Section 2.3.3). Denote A_m as the non-backbone robot that is in local minimum, and define a cost function

of robot i, to be the distance between robot i and the goal of A_m, x_m^g.

$$C(A_i, A_m) = \|x_i - x_m^g\| \tag{5.1}$$

The basic idea is to propagate a navigation field (with $C(A_i, A_m)$ as the navigation function) over the (stationary) backbone, then A_m follows the field along the gradient descent direction. Details are given below.

1. Subgoal election: For an A_m, the backbone robot with the smallest cost will be chosen as its subgoal. This is a global election, and involves all robots. A backbone robot proposes itself to be a subgoal candidate if it has smaller cost (i.e., closer to x_m^g) than any of its backbone neighbors. Note that the proposal is only based on a robot's (2-hop) neighbor information, and it is possible that there are more than one robot assuming itself to be the sub-goal. To reach a global decision, the planning step follows.

2. Planning: Candidates broadcast a *MSG_PLAN_NBB* message with its own information (including position). A robot receives the message and stores the route to the candidate. A robot may receive more than one message, and if the received message gives a better (smaller-cost, and shorter) path to the goal of A_m, x_m^g, the robot updates the route, and forwards this message if the receiving robot is a backbone robot. At the end, all robots store the shortest path to x_m^g via one of the candidates, the elected subgoal. This procedure is similar to the goal dissemination procedure as in Chapter 2 (Section 2.3), except that we may have multiple sources (i.e., multiple subgoal candidates) here. At the end of the procedure, all robots come to a unified conclusion of who is the winning subgoal for A_m, and every robot has the best route to the subgoal. Please note that a backbone robot only processes *MSG_PLAN_NBB* messages from backbone robots it connects to, and simply discards the messages from any other robots; A non-backbone robot receives and processes *MSG_PLAN_NBB* messages but never forwards the message.

3. Navigation: After previous planning step, all robots store the next robot in the best route to the subgoal of A_m, so they can provide guidance (to A_m) regarding the best movement toward its subgoal. To escape from the local minimum, A_m constantly broadcasts a query to the backbone; backbone robots respond with next via-point based on stored routes; and A_m chooses the best next via-point as "sub-sub-goal",

and move toward that. Such query-respond-move procedure repeats until the subgoal is reached.

4. Back to BBCC: After A_m has reached its subgoal, it broadcasts a *MSG_ESCAPED* message to notify all robots that it is out of local minimum. Robots receive the message and remove the robot from $lmList$. Once all robots have reached their respective subgoals, the system resumes to BBCC. If one or more of the robots are unable to reach their respective sub-goals, the strategy simply reports a failure. In future work, we will explore more sophisticated strategies in such cases.

5.4.2 Backbone-based Leader-following strategy

For a Type-III minimum, the system uses Backbone-based Leader-following strategy. The basic idea is to construct a leader-following tree hierarchy with a leader that is most likely free from local minima, and then move closer toward the leader to increase connectivity and hence reconfigurability. Define a gain function for robot i, as its distance to the local minimum robot A_m.

$$G(A_i, A_m) = \|x_i - x_m\| \tag{5.2}$$

The Leader-following escaping procedure then includes the following steps.

1. Leader candidates: The backbone robot that has maximum gain (i.e., farthest from *any* local minimum robot) is elected as the leader. Similar to subgoal election in previous Navigation strategy, leader election involves all robots. A backbone robot proposes itself to be a leader candidate if it has bigger gain than any of its backbone neighbors.

2. Spanning tree construction and leader election: Spanning tree and global leaders are computed in one go. Leader candidates broadcast a *MSG_PLAN_BB* message with its gain. A robot receives the message, and checks if the received message gives a better (with larger gain, and shorter route) leader. If so it updates its route to the new leader, and if the receiving robot is a backbone robot, it forwards the message. At the end, all robots have routes to the winning leader (A_l) with the highest gain. These routes make a tree hierarchy with A_l as the root. Note that since during the construction, only backbone robots forward the message, the resulting spanning tree has all non-backbone robots as leaves.

3. Leader-following: Once the spanning tree is constructed, robots move in a leader-following fashion. A robot follows its parent in the spanning tree, and moves toward its parent, instead of toward its ultimate goal.

4. Back to BBCC: After the (vertex) connectivity of the local minimum robot has increased by a certain degree (a user-defined parameter in the algorithm), or simply after moving for a certain period of time, the system stops and resumes to the regular BBCC to move to original goal.

5.4.3 Implementation details

We have introduced general ideas of the framework, and skipped some important technical implementation details for clarity. We present these details in this section. Figure 5.3 shows our state machine design for the proposed local minima escaping scheme. The same state machine runs on all robots, but it may take different transitions on different hosts depending on whether the host is a backbone or non-backbone robot, and whether it is a robot in local minimum.

Synchronization

In the framework, the system may switch from BBCC moving mode to Navigation escaping mode, or to Leader-following escaping mode, and then switch back to BBCC. In different modes, robots move with different constraints. Switching between modes needs synchronization, since we need to make sure robots move with proper constraints engaged, otherwise the system may became disconnected. To synchronize switching, we introduce some intermediate states/modes, and some extra messages. While backbone construction involves only local robots and generates $O(\Delta)$ messages for each of backbone robots, synchronization messages (including all messages shown in the Figure 5.3) need to reach all robots, and generate $O(n)$ messages for each round of synchronization, where n is the number of mobile robots.

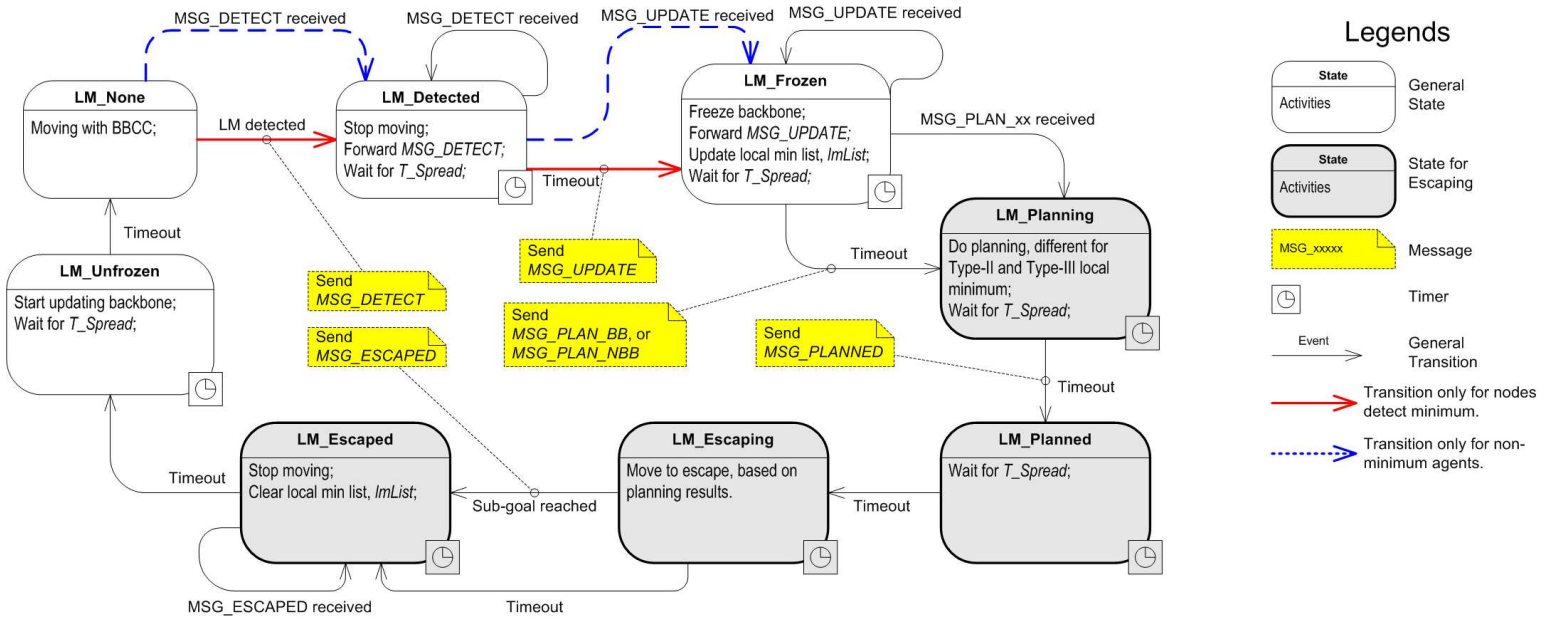


Figure 5.3: State machine implementation for Type-II and Type-III minimum escape with Backbone-based Navigation strategy and Backbone-based Leader-following strategy.

Freezing and unfreezing the backbone Once a robot detects a Type-II or Type-III minimum, it stops moving and broadcasts *MSG_DETECT* message, and robots receiving the message also stop moving. However, robots should not stop updating backbone while any robot is still moving. As a consequence, the backbone may have changed since the time the local minimum is first detected. Therefore, we introduce two extra states, *LM_Detected* and *LM_Frozen*, and an extra message, *MSG_UPDATE* to synchronize the backbone. When a robot first detects a local minimum, or receives the *MSG_DETECT* message, it stops, transits into *LM_Detected* state, and waits for *T_Spread* seconds. Assuming after this wait all robots have stopped and backbone has been stationary, the local minimum robot sends a *MSG_UPDATE* message after timeout to freeze the backbone, and update the *lmList* in each robot. Similarly, when the system resumes from escape mode to BBCC, robots should start updating the backbone before they can start moving. The *LM_Unfrozen* state serves this purpose and makes sure the backbone is updated by waiting for *T_Spread* seconds before transitioning back to *LM_None* state (BBCC).

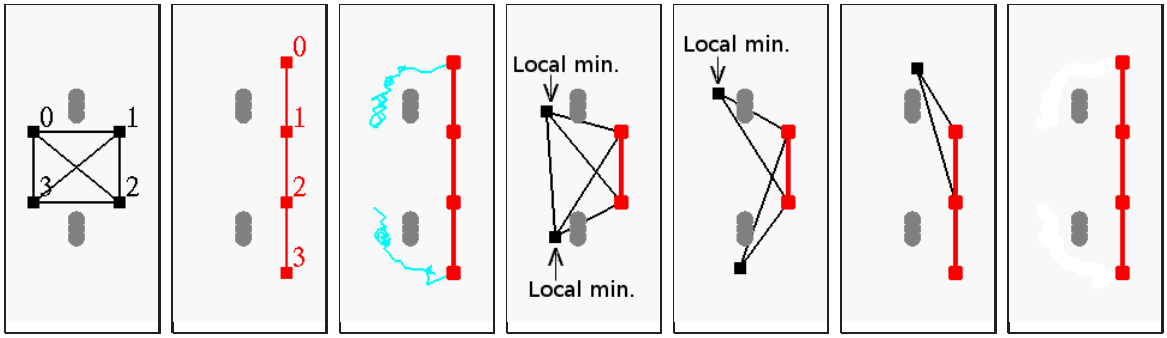
Planning and spanning tree construction In *LM_Planning* state, robots do path planning or spanning tree construction depending on what type of local minimum the system is dealing with. Once the path has been planned, or the spanning tree has been constructed, all robots transition into *LM_Planned* state where escaping robots can prepare for escape. Specifically, in Leader-following escape mode, the spanning tree (instead of the backbone) is used as connectivity constraints, and should be engaged in *LM_Planned* state.

5.5 Computer simulations

We now present simulations to show the effectiveness of the proposed scheme in escaping local minima. We simulate a team of robots moving, in an arena of $120m \times 120m$, from an initial formation to a goal formation. The communication range between robots is set to 15m. Again, we assume unit disk communication model and obstacles do not obstruct communication.

5.5.1 Escaping Type-I minimum

Figure 5.4 shows a case of Type-I minimum, where two small obstacles blocked robot 0 and 3 from their goals. While the robots were trying to reach their goals, the obstacles kept



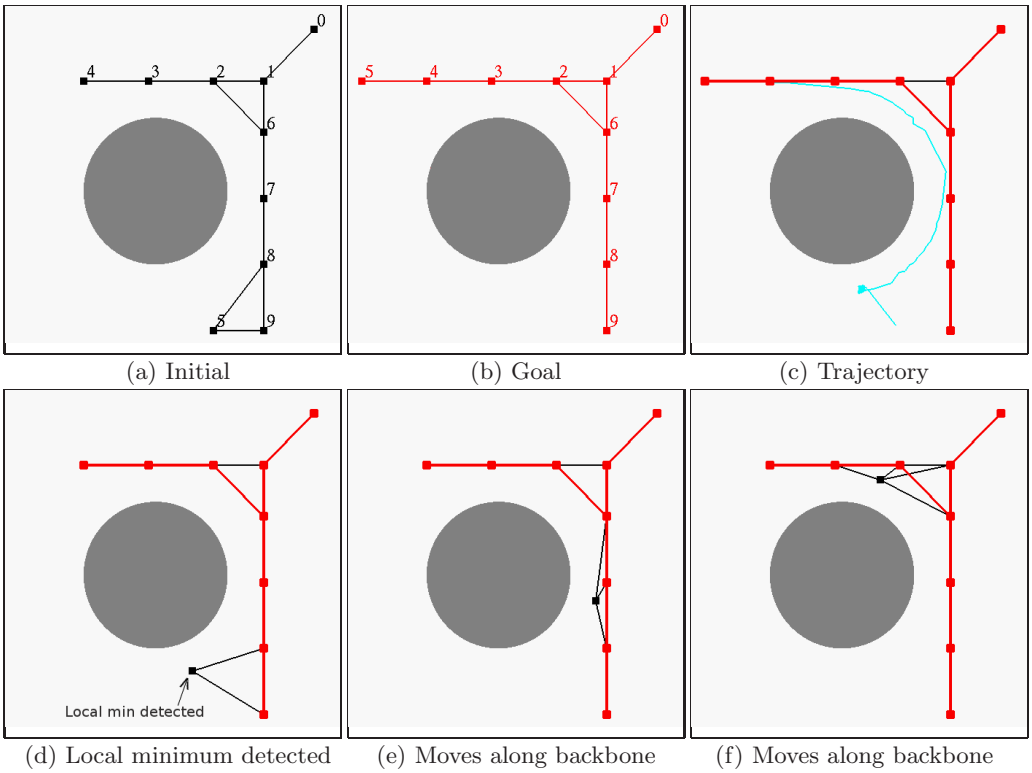
(a) Init. (b) Goal (c) Traj. (d) (e) (f) (g)

Figure 5.4: Escaping from Type-I minimum. (a) Initial formation. (b) Goal formation. The shaded regions are obstacles to be avoided.(c) Successful trajectories of robot 0 and 3 with Random Walk strategy. (d)-(g) Snapshots of the system along the successful trajectories. (d) Local minima detected by robot 0 and 3. (e) Robot 3 escaped the minimum, while robot 0 detected another minimum and eventually escaped with another around of random walk.

pushing them away, and therefore local minima were detected around the obstacles. After detecting that they are in local minima, robots tried random walk to avoid the obstacles, and after random walk the robots moved toward their respective goals again. Due to the random nature, it might take several rounds of random walk to escape the minimum. For example, in the shown simulation, it took robot 0 longer to escape. Other deterministic local strategies may yield better performance.

5.5.2 Escaping Type-II minimum

Figure 5.5 shows a case of Type-II minimum, where all robots were right at their goal position, except one, robot 5, as shown. It tried to move toward its goal position, but the combination of obstacles and connectivity constraints prevented it from doing so, and the robot detected local minimum as shown in (d). After trying random walk for several times, the local minimum persisted, since the obstacle was relatively large. As robot 5 was a non-backbone robot, the local minimum was deemed to be Type-II, and the Backbone-based Navigation strategy was activated for escaping. To navigate robot 5 out of its local minimum, the existing backbone (bold colored lines and robots) was used as planning roadmap, and location of robot 4 was elected as subgoal for 5. In snapshots from (d) to (f), robot 5 moved along the found path, and reached the subgoal. Clearly, from there it could easily reach its final goal.



(a) Initial (b) Goal (c) Trajectory

(d) Local minimum detected (e) Moves along backbone (f) Moves along backbone

Figure 5.5: Escaping from Type-II minimum. (a) Initial formation. (b) Goal formation. (c) A successful trajectory of the local minimum robot, with our Backbone-based Navigation strategy. (d) A local minimum detected. (e)-(f) Snapshots of the system as the local minimum robot along the successful trajectory guided by the stationary backbone.

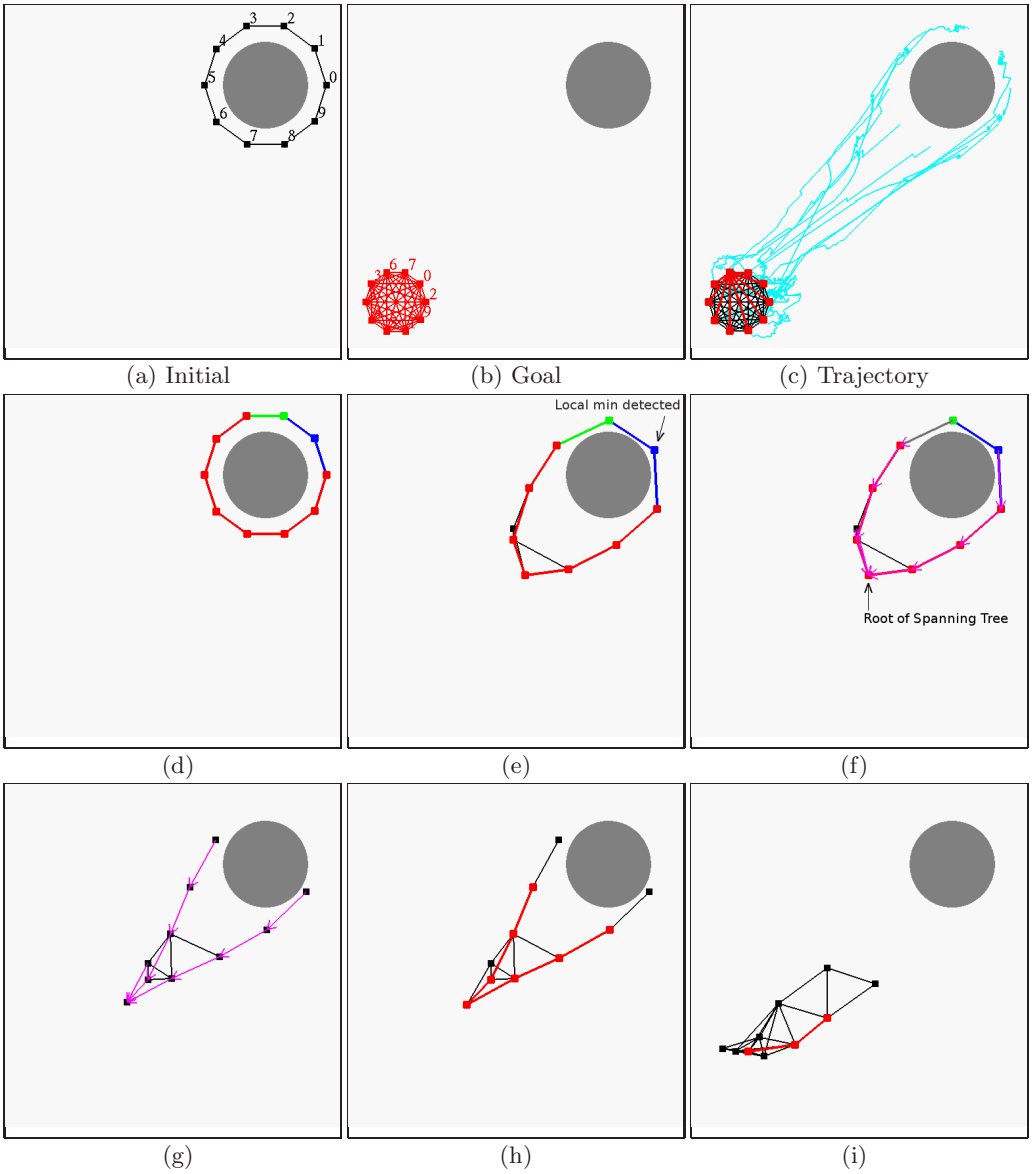
5.5.3 Escaping Type-III minimum

Figure 5.6 shows a case of Type-III minimum, where all robots were wrapped around an obstacle in the initial configuration, and the obstacle was so large that a robot could only communicate with its immediate neighbors (a). In this case, all robots were in the backbone (c), because from every robot's local point of view, all its edges were critical in maintaining connectivity. The goal formation was a complete graph away from the obstacle (as in (a)). In order to achieve the goal, the team had to break some links between robots. When the team tried to move toward the goal formation, connectivity constraints kept the robots from moving any further as in (d). Clearly random walk did not help much in this case, and the system detected a Type-III minimum, as a backbone robot (robot 1 as shown) was in local minimum, and hence the Backbone-based Leader-following strategy was engaged. With the

strategy, a spanning tree was constructed, as shown in (e). The root of the spanning tree (the leader) was the backbone robot that was farthest away from the local minimum robots. Then for a certain period of time, all robots moved and followed this leader, resulting in formation in (g), and from there the system easily reached the goal formation.

5.6 Summary

We have proposed distributed local minimum escape strategies for motion planning with connectivity constraint for mobile networks. The strategies leverage the backbone constructed from our earlier proposal, Backbone Based Connectivity Control. Backbone-based Navigation strategy is adopted for non-backbone robots in local minimum (i.e., a Type-II minimum); then the backbone is used as a roadmap to navigate robots deemed to be in local minimum, to move toward its goal. Backbone-based Leader-following strategy is used when a backbone robot is in local minimum (i.e., a Type-III minimum); then a spanning tree hierarchy, based on the backbone, is established among robots, and robots follow the hierarchy and move closer to each other for reconfiguration. We showed, via computer simulations, that the proposed strategies are effective in escaping local minima.



(a) Initial (b) Goal (c) Trajectory

(d) (e) (f)

(g) (h) (i)

Figure 5.6: Escaping from Type-III minimum. (a) Initial and goal formations. (c) Successful trajectories of all robots, with Backbone-based Leader-following strategies. (d)-(i) Snapshots of the system along the successful trajectories. (d) Backbone of initial formation, and all robots were in the backbone. (e) A local minimum detected. (f) Spanning tree was constructed. (g-h) Moving in leader-following mode. (i) Back to BBCC.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

We have investigated two different types of interaction between robot motion and network communication: *communication-assisted motion planning* for single robot navigation in static sensor network, and *communication-constrained motion planning* for connectivity control in mobile sensor networks.

Communication-assisted motion planning. This problem studies how to use sensor network deployed in a (hazardous) environment to guide a mobile robot through a safe path. Depending on sensors used in the network, we investigated the problem from two different aspects. (i) We started with a simple sensor model, where a sensor only gives a simple reading within the sensing range. In this case, sensing regions with excessive readings are deemed to be dangerous and should be to avoided. We propose a distributed path planning method using communication backbone as a path planning roadmap. In building and maintaining the roadmap, it takes into account safety and network longevity, and therefore the roadmap adapts to dynamic dangers and evolves over time for load balance. (ii) For more sophiscated sensor networks, where sensors give spatial maps within their sensing regions, fine-grain path planning is possible to deal with physical obstacles, and navigate robots through narrow passages. We propose a distributed sampling based planning algorithm, where every sensor node creates a local probobilistic roadmap in its locally-sensed environment; these local roadmaps are "stitched" together by passing messages among nodes

and form a larger implicit roadmap without having a global representation of the environment. Based on the implicit roadmap, a feasible path is computed in a distributed manner. The proposed algorithm applies to robots with non-trivial shapes, as well as multi-robot formations.

Communication-constrained motion planning. This problem studies how to maintain connectivity among a team of mobile robots in a cooperative task. We propose a hierarchical connectivity control scheme based on communication backbone of the network. The key idea of Backbone Based Connectivity Control (BBCC) is to use adaptive backbone to represent the system topology, which is updated in real time to capture the dynamic topology of the system and to impose motion constraints on robots so that network connectivity is maintained. BBCC maintains the system connectivity in two levels: it first maintains a connected backbone, by maintaining existing connections (communication links) in the backbone; and then for a non-backbone robot, one of the backbone robots is chosen as leader, and connection to the leader is maintained. However, BBCC uses potential field based techniques to maintain critical links, so local minimum may arise when the team of robots trying to achieve their respective goals, maintain connectivity and avoid collisions at the same time. To deal with the local minimum problem, we classify local minima into three different categories: Type-I (Regional obstacle-induced local minimum), Type-II (Individual connectivity-induced local minimum), and Type-III (Structural compound local minimum). Different types imply different natures and causes of the minima, and we adopt different escaping strategies: Random Walk, Backbone-based Navigation strategy, and Backbone-based Lead-following strategy for the three different types of local minima respectively.

6.2 Future works

Multi-layer clustering for backbone-based roadmap. To further reduce the size of backbone-based roadmap, the network can be further scaled down by *K-hop clustering* [6] algorithms where every node in the network should have at least one clusterhead within k-hop($k \geq 1$), or by using *multi-layer clustering* [71], where further clustering can be applied on top of the backbone network.

Cooperative collision check for distributed probabilistic roadmap. *Distributed-PRM* assumes that the local environments and relay zones always have large enough free

space to contain the entire shape or formation, and collision checks by individual node is enough to find a feasible path. However, there are more difficult and extreme cases. In some cases, because of occlusion, two neighboring sensor nodes can only see a small part of relay zone from different side; in other cases, the dimension of robot is simply too large, and it has to span multiple maps of adjacent sensor nodes. In these cases, search for collision-free path requires cooperative collision checking from different sensor nodes.

Convergence and stability analysis of BBCC. There have been several recent works on stability and convergence analysis of multi-robot systems [60, 64]. These works are mostly limited to specific tasks, such as consensus, rendezvous, swarming, or flocking, and ignore either connectivity constraints or obstacle avoidance aspects in the tasks. Based on these works, we are looking to analyze backbone based connectivity control scheme for aforementioned specific tasks (e.g. consensus, etc.), and then for the more general problem as we formulated.

More formal local minimum classification and escaping. As a first attempt to attack the local minimum problems, the proposed local minima classification and strategies to deal with them are based on emperical observations and are somewhat heuristic based. We are looking into more systematic ways to treat them.

Implementation on real systems. We tested proposed algorithms based on software simulations, and implementation and experiments on a real system can be quite different and challenging, as many realistic conditions need to be considered, particularly imperfect communication. For example, in BBCC, interference and latency in communication may result in inconsistent neighbor information, and hence may affect the backbone construction and update.

Relationship with modular robot reconfiguration. The problem of motion planning with connectivity constraints are somewhat similar to reconfiguration planning problem for self-reconfigurable modular robots [105], which studies how to plan a motion for each modular robot without disconnecting the system, such that the system will reconfigure from the initial to the goal configuration. Even though modular robots usually assume physical contact for communication, and uses primitive discrete motion model, there are possible connections between them. In BBCC, communication backbone is an adaptive and efficient

way to represent the topology of the system, so it can be a good tool to realize adaptability, robustness and low cost of reconfigurable systems, which are key challenges in modular systems [105]. On the other hand, some existing results in modular robot reconfiguration may be instructive in deriving our theoretic results in the connectivity control problem for mobile networks. For example, Prevas et al.[69] showed that, under certain assumptions, it is always possible to transform from one connected configuration to another connected configuration. Dumitrescu and Pach [28] showed a similar result, and also showed the upper bound of reconfiguration complexity in terms of number of moves to transform between two configurations. Even though these results are only valid for some specific systems (e.g., hexagon or grid modular robots), they may be helpful in finding similar but more general results for mobile networks.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 3/15 2002.

[2] J. N. Al-Karaki and A. E. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE*, 11(6):6–28, 2004. ID: 1.

[3] G. Alankus, N. Atay, Chenyang Lu, and O. Bayazit. Spatiotemporal query strategies for navigation in dynamic sensor network environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3718–3725, 2005.

[4] G. Alankus, N. Atay, Chenyang Lu, and O. Bayazit. Adaptive embedded roadmaps for sensor networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3645–3652, 2007.

[5] N. M. Amato and L. K. Dale. Probabilistic roadmap methods are embarrassingly parallel. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 688–694, 1999.

[6] A. D. Amis, R. Prakash, T. H. P. Vuong, and D. T. Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 32–41, 2000.

[7] T. Arai, E. Pagello, and L. E. Parker. Guest editorial advances in multirobot systems. *Robotics and Automation, IEEE Transactions on*, 18(5):655–661, 2002. ID: 1.

[8] Javed Aslam, Qun Li, and Daniela Rus. Three power-aware routing algorithms for sensor networks. *Wireless Communications and Mobile Computing*, 3(2):187–208, 2003.

[9] M. Bal, Min Liu, Weiming Shen, and H. Ghenniwa. Localization in cooperative wireless sensor networks: A review. In *Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on*, pages 438–443, 2009. ID: 1.

[10] Lichun Bao and J. J. Garcia-Luna-Aceves. Topology management in ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 129–140, New York, NY, USA, 2003.

[11] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms, and Network*, pages 310–315, 1999.

[12] S. Basagni, M. Mastrogiovanni, and C. Petrioli. A performance comparison of protocols for clustering and backbone formation in large scale ad hoc networks. In *Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 70–79, 2004.

[13] M. A. Batalin, G. S. Sukhatme, and M. Hattig. Mobile robot navigation using a sensor network. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 636–641, 2004.

[14] Maxim A. Batalin and Gaurav S. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. *Telecommunication Systems*, 26(2):181–196, 06/01 2004. M3: 10.1023/B:TELS.0000029038.31947.d1.

[15] S. Bhattacharya, N. Alankus Atay G., Chenyang Lu, O. Bayazit, and Gruia-Catalin Roman. *Roadmap Query for Sensor Network Assisted Navigation in Dynamic Environments*, pages 17–36. 2006.

[16] David Braginsky and Deborah Estrin. Rumor routing algorthim for sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 22–31, New York, NY, USA, 2002. ACM.

[17] A. Brooks, S. Williams, and A. Makarenko. Automatic online localization of nodes in an active sensor network. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 5, pages 4821–4826 Vol.5, 2004.

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

[19] Nirupama Bulusu, John Heidemann, and Deborah Estrin. Gps-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, 2000.

[20] C. Buragohain, D. Agrawal, and S. Suri. Distributed navigation algorithms for sensor networks. In *Proceedings of the 25th IEEE International Conference on Computer Communications*, pages 1–10, 2006.

[21] Y. Uny Cao, Alex S. Fukunaga, and Andrew B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–23, March 1997.

[22] Y. Chen, A. Liestman, and J. Liu. *Clustering Algorithms for Ad Hoc Wireless Networks*. Ad Hoc and Sensor Networks. Nova Science Publisher, 2004.

[23] Jren-Chit Chiny, I-Hong Houz, Jennifer C. Houz, Chris May, Nageswara S. Rao, Mohit Saxenay, Mallikarjun Shankar, Yong Yangz, and David K. Y. Yau. A sensor-cyber network testbed for plume detection, identification, and tracking. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 541–542, New York, NY, USA, 2007. ACM.

[24] F. Coutinho, J. Barreiros, and J. Fonseca. Choosing paths that prevent network partitioning in mobile ad-hoc networks. In *Proceedings of the IEEE International Workshop on Factory Communication System*, pages 65–71, 2004.

[25] K. Dantu, M. Rahimi, H. Shah, S. Babel, A. Dhariwal, and G. S. Sukhatme. Robomote: enabling mobility in sensor networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, pages 404–409, 2005.

[26] B. Das and V. Bharghavan. Routing in ad-hoc networks using minimum connected dominating sets. In *Proceedings of the IEEE International Conference on Communications*, volume 1, pages 376–380, 1997.

[27] D. V. Dimarogonas and K. J. Kyriakopoulos. Connectedness preserving distributed swarm aggregation for multiple kinematic robots. *Robotics, IEEE Transactions on*, 24(5):1213–1223, 2008.

[28] Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22(1):37–50, 04/23 2006.

[29] Prabal Dutta and Sarah Bergbreiter. Mobiloc: Mobility enhanced localization, 2003.

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

[31] E. Fiorelli, N. E. Leonard, P. Bhatta, D. Paley, R. Bachmayer, and D. M. Fratantoni. Multi-auv control and adaptive sampling in monterey bay. In *Autonomous Underwater Vehicles, 2004 IEEE/OES*, pages 134–147, 2004. ID: 1.

[32] Alex Fridman, Jay Modi, Steven Weber, and Moshe Kam. Communication-based motion planning. In *Proceedings of the 41st Annual Conference on Information Sciences and System*, pages 382–387, 2007.

[33] A. Galstyan, B. Krishnamachari, K. Lerman, and S. Pattem. Distributed online localization in sensor networks using a moving target. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*, pages 61–70, 2004.

[34] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[35] B. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, June 2003.

[36] M. Gerla and J. Tsai. Multicluster, mobile, multimedia radio network. *Journal of Wireless Networks*, 1(3):255–265, 1995.

[37] Nissanka Priyantha Hari, Hari Balakrishnan, Erik Demaine, and Seth Teller. Anchor-free distributed localization in sensor networks. Technical report, Massachusetts Institute of Technology, 2003.

[38] S. T. Hedetniemi and R. Laskar. *Connected domination in graphs*. Graph Theory and Combinatorics. Academic Press, London, 1984.

[39] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, volume 8, page 8020, 2000.

[40] D. Henrich. Fast motion planning by parallel processing – a review. *J.Intell.Robotics Syst.*, 20(1):45–69, 1997.

[41] Jeffrey Hightower, Gaetano Borriello, and Roy Want. Spoton: An indoor 3d location sensing technology based on rf signal strength. Technical report, University of Washington, Computer Science and Engineering, 2000.

[42] E. Hossain, R. Palit, and P. Thulasiraman. *Clustering in mobile wireless ad hoc networks: issues and approaches*, pages 383–424. Wireless communications systems and networks. Springer US, 2004.

[43] David Hsu. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *In Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4420–4426, 2003.

[44] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans.Netw.*, 11(1):2–16, 2003.

[45] Meng Ji and M. Egerstedt. Distributed coordination control of multiagent systems while preserving connectedness. *IEEE Transactions on Robotics*, 23(4):693–703, 2007.

[46] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566, 1996.

[47] Ara N. Knaian. A wireless sensor network for smart roadbeds and intelligent transportation systems, 2000.

[48] J. Kuffner and S. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, pages 995–1001, 2000.

[49] Vijay Kumar, Daniela Rus, and Sanjiv Singh. Robot and sensor networks for first responders. *IEEE Pervasive Computing*, 3(4):24–33, 2004.

[50] G. Lafferriere, J. Caughman, and A. Williams. Graph theoretic methods in the stability of vehicle formations. In *Proceedings of the American Control Conference*, volume 4, pages 3729–3734, 2004.

[51] Anthony LaMarca, Waylon Brunette, David Koizumi, Matthew Lease, Stefan Sigurdsson, Kevin Sikorski, Dieter Fox, and Gaetano Borriello. *Making sensor network practical with robots*, pages 615–622. Pervasive Computing. Springer Berlin/Heidelberg, 2002.

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

[53] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Dept., Iowa State University, 1998.

[54] Q. Li and D. Rus. Navigation protocols in sensor networks. *ACM Transactions on Sensor Networks*, 1(1):3–35, August 2005.

[55] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[56] Benyuan Liu, Peter Brass, Olivier Dousse, Philippe Nain, and Don Towsley. Mobility improves coverage of sensor networks. In *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 300–308, New York, NY, USA, 2005. ACM.

[57] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[58] Jerome Peter Lynch. An overview of wireless structural health monitoring for civil structures. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1851):345–372, February 15 2007.

[59] Guoqiang Mao, Bar Fidan, and Brian D. O. Anderson. Wireless sensor network localization techniques. *Computer Networks*, 51(10):2529–2553, 7/11 2007.

[60] L. Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on Automatic Control*, 50(2):169–182, 2005.

[61] D. Niculescu and B. Nath. Ad hoc positioning system (aps). In *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, volume 5, pages 2926–2931 vol.5, 2001. ID: 1.

[62] D. Niculescu and Badri Nath. Ad hoc positioning system (aps) using aoa. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1734–1743 vol.3, 2003. ID: 1.

[63] K. J. O'Hara, V. L. Bigio, E. R. Dodson, and A. J. Irani. Physical path planning using the gnats. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 709–714, 2005.

[64] R. Olfati-Saber. Flocking for multi-agent dynamic systems: algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, 2006.

[65] G. A. S. Pereira, A. K. Das, V. Kumar, and M. F. M. Campos. Decentralized motion planning for multiple robots subject to sensing and communication constraints. In *Multi-robot Systems: From Swarms to Inelligent Automata*, volume 2, pages 267–278, Washington, DC, 2003.

[66] K. S. Pister. Tracking vehicles with a uav-delivered sensor network. March 2001.

[67] E. Plaku and L. E. Kavraki. Distributed sampling-based roadmap of trees for large-scale motion planning. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3868–3873, 2005.

[68] R. Pon, A. Kansal, Duo Liu, M. Rahimi, L. Shirachi, W. J. Kaiser, G. J. Pottie, M. Srivastava, G. Sukhatme, and D. Estrin. Networked infomechanical systems (nims): next generation sensor networks for environmental monitoring. In *Microwave Symposium Digest, 2005 IEEE MTT-S International*, page 4 pp., 2005. ID: 1.

[69] K. C. Prevas, C. Unsal, M. O. Efe, and P. K. Khosla. A hierarchical motion planning strategy for a uniform self-reconfigurable modular robotic system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 787–792, 2002.

[70] N. B. Priyantha, H. Balakrishnan, E. D. Demaine, and S. Teller. Mobile-assisted localization in wireless sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 1, pages 172–183 vol. 1, 2005. ID: 1.

[71] R. Ramanathan and M. Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *Mobile Networks and Applications*, 3(1):101–119, 1998.

[72] S. Ramanathan and Errol L. Lloyd. Scheduling algorithms for multi-hop radio networks. In *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*, pages 211–222, New York, NY, USA, 1992. ACM.

[73] Martijn N. Rooker and Andreas Birk. Multi-robot exploration under the constraints of wireless networking. *Control Engineering Practice*, 15(4):435–445, 2007.

[74] E. Sampathkumar and H. B. Walikar. The connected domination number of a graph. *Journal of Mathematical Physical Sciences*, 13:607–613, 1979.

[75] Paolo Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput.Surv.*, 37(2):164–194, 2005.

[76] Andreas Savvides, Chih-Chieh Han, and Mani B. Strivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 166–179, New York, NY, USA, 2001. ACM.

[77] T. Schouwenaars, A. Stubbs, J. Paduano, and E. Feron. Multi-vehicle path planning for non-line of sight communication. *Journal of Field Robotics, special issue on UAVs*, 23(3-4):269–290, 2006.

[78] Curt Schurgers and Mani B. Srivastava. Energy efficient routing in wireless sensor networks. 2001.

[79] Loren Schwiebert, Sandeep K. S. Gupta, and Jennifer Weinmann. Research challenges in wireless networks of biomedical sensors. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 151–165, New York, NY, USA, 2001. ACM.

[80] Rahul C. Shah and Jan M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2002.

[81] M. L. Sichitiu and V. Ramadurai. Localization of wireless sensor networks with a mobile beacon. In *Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on*, pages 174–183, 2004. ID: 1.

[82] Gyula Simon, Miklós Maróti, Ákos Lédeczi, Gyørgy Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor network-based countersniper system. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12, New York, NY, USA, 2004. ACM.

[83] David Simplot-Ryl, Ivan Stojmenovic, and Jie Wu. *Energy-Efficient Backbone Construction, Broadcasting, and Area Coverage in Sensor Networks*, pages 343–380. Handbook of Sensor Networks. 2005.

[84] Stanislava Soro and Wendi Heinzelman. A survey of visual sensor networks. *Advances in Multimedia*, 2009:1–22, 2009.

[85] D. P. Spanos and R. M. Murray. Robust connectivity of networked vehicles. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, volume 3, pages 2893–2898, 2004.

[86] D. P. Spanos and R. M. Murray. Motion planning with wireless network constraints. In *Proceedings of the American Control Conference*, pages 87–92, 2005.

[87] S. C. Spry, A. R. Girard, and J. K. Hedrick. Convoy protection using multiple unmanned aerial vehicles: organization and coordination. In *American Control Conference, 2005. Proceedings of the 2005*, pages 3524–3529 vol. 5, 2005.

[88] National Research Council Staff. *Embedded Everywhere: A Research Agenda for Networked Systems of Embedded Computers*. National Academy Press, 2001.

[89] B. Taati, M. Greenspan, and K. Gupta. A dynamic load-balancing parallel search for enumerative robot path planning. *Journal of Intelligent and Robotic Systems*, 47(1):55–85, 09/23 2006. M3: 10.1007/s10846-006-9067-z.

[90] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.

[91] J. Vazquez and C. Malcolm. Distributed multirobot exploration maintaining a mobile network. In *Proceedings of the 2nd IEEE International Conference on Intelligent Systems*, volume 3, pages 113–118, 2004.

[92] Roberto Verdone, Davide Dardari, Gianluca Mazzini, and Andrea Conti. *Wireless Sensor and Actuator Networks: Technologies, Analysis and Design*. Academic Press, 2008.

[93] A. Verma, H. Sawant, and J. Tan. Selection and navigation of mobile sensor nodes using a sensor network. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, pages 41–50, 2005.

[94] Marcos Augusto Menezes Vieira. *The interplay between networks and robotics: networked robots and robotic routers*. PhD thesis, University of Southern California, 2010.

[95] PengJun Wan, Khaled M. Alzoubi, and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):141–149, 2004.

[96] Yu Wang, WeiZhao Wang, and Xiang-Yang Li. Distributed low-cost backbone formation for wireless ad hoc networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, pages 2–13, May 25 - 27 2005.

[97] M. C. Wicks. Radar the next generation - sensors as robots. In *Radar Conference, 2003. Proceedings of the International*, pages 8–14, 2003. ID: 1.

[98] F. Xue and P. R. Kumar. The number of neighbors needed for connectivity of wireless networks. *Wireless Networks*, 10(2):169–181, March 2004.

[99] Yinying Yang and Mihaela Cardei. Movement-assisted sensor redeployment scheme for network lifetime increase. In *MSWiM '07: Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, pages 13–20, New York, NY, USA, 2007. ACM.

[100] Zhenwang Yao and Kamal Gupta. Backbone-based roadmaps for robot navigation in sensor networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2008.

[101] Zhenwang Yao and Kamal Gupta. Backbone-based connectivity control for mobile networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 12-17 2009.

[102] Zhenwang Yao and Kamal Gupta. Distributed strategies for local minima escape in motion planning for mobile networks. In *Proceedings of the 2nd International Conference on Robot Communication and Coordination*, March 31 - April 2 2009.

[103] Zhenwang Yao and Kamal Gupta. Distributed roadmaps for robot navigation in sensor networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2-7 2010.

[104] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 8/22 2008.

[105] M. Yim, Wei-Min Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian. Modular self-reconfigurable robot systems: Challenges and opportunities for the future. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.

[106] M. M. Zavlanos and G. J. Pappas. Potential fields for maintaining connectivity of mobile networks. *IEEE Transactions on Robotics*, 23(4):812–816, 2007.

[107] Michael M. Zavlanos and George J. Pappas. Distributed connectivity control of mobile networks. In *Proceedings of the 46th IEEE Conference on Decision and Control*, pages 3591–3596, 2007.

[108] B. Zhang, A. Dhariwal, A. Pereira, J. Das, C. Oberg, B. Stauffer, L. Darjany, X. Bai, D. Caron, and G. S. Sukhatme. Adaptive sampling by using mobile robots and a sensor network. Posters., Center for Embedded Network Sensing, 2007.