Closing the Loop: Control and Robot Navigation in Wireless Sensor Networks

Shawn Michael Schaffert

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-112
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-112.html

September 5, 2006

Acknowledgement

This research was made possible in part by the generous financial support of the Defense Advanced Research Projects Agency under grant F33615-01-C-1895.

Closing the Loop: Control and Robot Navigation in Wireless Sensor Networks

by

Shawn Michael Schaffert

B.S. (University of Nebraska, Lincoln) 1998
M.S. (University of California, Berkeley) 2001

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

in

Electrical Engineering and Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Shankar S. Sastry, Chair
Professor Ruzena Bajcsy
Professor Steven D. Glaser

Fall 2006

The dissertation of Shawn Michael Schaffert is approved:

Chair Date

Date

Date

University of California, Berkeley

Fall 2006

Closing the Loop: Control and Robot Navigation in Wireless Sensor Networks

Abstract

Closing the Loop: Control and Robot Navigation in Wireless Sensor Networks

by

Shawn Michael Schaffert

Doctor of Philosophy in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Shankar S. Sastry, Chair

Wireless sensor networks have received considerable attention for their potential as a cheap, easily deployed, distributed monitoring tool. Recently, researchers have begun to investigate the use of wireless sensor networks to drive closed-loop control systems. However, such composite systems are nontrivial to design due to the system interface dichotomy: control systems typically assume periodic, high frequency sensor updates whereas sensor networks provide aperiodic, low frequency, and laggy sensor updates. Utilizing robot navigation and pursuit-evasion games as benchmarks, our research focuses on improving control system performance by exploiting the properties of wireless sensor networks.

We developed and deployed a real-world, medium-scale wireless sensor network for playing pursuit-evasion games. Using our experience from this deployment, we highlight the difficulties in using sensor network data to accurately localize robots. Several techniques designed to compensate for such difficulties are developed and incorporated into an unified system architecture. To test our architecture, an application-level simulator, accounting for many of the sensor network characteristics that frustrate control design, is developed. This simulator allows us to identify components of our system architecture that can improve the performance of control systems operating in networks of sensors. Amongst the components, intelligent path planning is identified as uniquely important in improving robot localization accuracy during navigation.

Path planning techniques that use information maps, exploiting the knowledge of node topology and sensor models, are developed. Information is a metric for measuring the ability of a region in the environment to aid in robot localization. In particular, for each region in the environment, an information map computes the change in entropy expected by a robot in this area

using Markov localization. We adapt sensor network models for use with information maps and verify the ability of such maps to improve robot localization. Additionally, automatic path planning techniques based on information maps are developed that minimize localization error. We compare the performance of these planners with other path planners, and demonstrate that this technique is effective for generating paths that increase the accuracy of localization while typically requiring fewer detections.

_____

Professor Shankar S. Sastry
Dissertation Committee Chair

To my parents,
Harry and Wilma Schaffert,
for their
endless love,
support,
and inspiration.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I express my sincere gratitude to my research advisor Professor Shankar Sastry for whom I have the greatest amount of respect and admiration. Professor Sastry has offered invaluable guidance and afforded me the opportunity to work with an excellent team of researchers. His continual commitment to excellence in research and his expansive vision are inspiration to us all.

I also thank Professor Ruzena Bajcsy and Professor Steven Glaser for serving on my dissertation committee and prompting insightful questions during the course. Additionally, I thank Professor David Culler for serving on my qualifying exam committee and providing helpful advice during my tenure on the NEST project.

I am thankful for the support and inspiration of my research colleague and good friend Cory Sharp. Cory is an encouraging and insightful person to interact with both as a researcher and on a personal level. I am grateful for the many occasions that Cory listened to my concerns and helped me clarify my thoughts.

A great team of researchers made the pursuit-evasion game experiment possible. For this, I thank Cory Sharp, Alec Woo, Naveen Sastry, Chris Karlof, Phoebus Chen, Fred Jiang, Jaein Jong, Sukun Kim, Phil Levis, Neil Patel, Joe Polastre, Robert Szewczyk, Terrence Tong, Rob von Behren, Kamin Whitehouse, Professor David Culler, Professor Eric Brewer, Professor David Wagner, Professor Kris Pister, and Professor Shankar Sastry.

During my time at Berkeley, I have been fortunate to encounter many other great researchers who have supported my research endeavors in one way or another. For this, I am thankful to Songhwai Oh, Michael Manzo, Bruno Sinopoli, Gilman Tolle, Prabal Dutta, Jonathan Hu, Sukun Kim, Jay Taneja, Bonnie Zhu, Tanya Roosta, Luca Schenato, Hoam Chung, and David Shim. Additionally, I thank Mike Howard, Travis Pynn, Damon Hinson, Peter Ray, and Maria Jauregui for creating a productive and supportive atmosphere at Berkeley.

Over the years, I have been encouraged by many good friends who have endorsed my goals, provided me an alternative perspective, and supported me in numerous ways. Amongst these, I am especially thankful to Johan Vanderhaegen who has been a good friend and a trusted advisor. During many conversation about work and life, Johan afforded me a much deeper understanding. Additionally, for their support in many aspects of my life, I would like to thank my friends Maria Herr, Mark Donahue, Scott Sutton, and Steve Smith.

I cannot fully express the gratitude I owe to my closest companion, Susan Standen. Susan

has, on a day to day basis, supported my efforts and helped me maintain my lucidity. She has especially encouraged my final drive in graduate school. For everything she has done for me, I am grateful.

Finally, I thank my family for their influence and support. My brother, Tyson Schaffert, has been a constant source of encouragement to me. His creativity, drive, and kindness have inspired me to succeed. I owe my deepest gratitude to my parents, Harry and Wilma Schaffert. They have constantly encouraged and supported me in any and all my endeavors. This work is dedicated to them.

This research was made possible in part by the generous financial support of the Defense Advanced Research Projects Agency under grant F33615-01-C-1895.

Chapter 1

Introduction

Each year the size and cost of electronics reduce producing a world ubiquitous with sensing and computing nodes. Observing this trend, researchers have turned to developing hardware and software platforms [46, 88] to support distributed sensing and computation via hundreds or thousands of tiny nodes. Each node, or mote, is designed to be a small, power-efficient sensing platform that supports on-board computation and wireless communication. Using a distributed array of motes, or a wireless sensor network (WSN), provides a new sensing platform useful for a variety of applications.

In the last several years, many of the foreseen applications and challenges [35, 75] of WSNs have been actively researched. WSNs have already shown their usefulness in many distributed sensing arenas such as habitat monitoring [20, 61], meteorology [59], information for first-responders [38], and vehicle detection [30]. During the development of such applications, challenges on several fronts have been investigated by researchers: ad-hoc networking [89, 2], power efficiency [67], in-network data aggregation [60, 62], time synchronization [34], and automatic node localization [93] to list a few.

The need to go beyond pure sensing applications and understand WSNs from a control and actuation perspective has been identified by researchers [18]. Several important control applications such as heating ventilation and air conditioning (HVAC) systems, pursuit-evasion games, and robot navigation stand to benefit by the introduction of a sensor network platform. A WSN system can reduce installation and maintenance costs for HVAC systems by eliminating wires between sensing and control points. Pursuit-evasion games can acquire a global view of the playing field eliminating the need for a pursuer robot to continuously patrol. Finally, robot navigation benefits in several ways. Sensor networks can localize robots for environments (such as inside buildings or outside

under tree cover) that other technologies (such as GPS) cannot. Additionally, these networks allow for the localization of robots without sensors reducing the cost of the robot platform and facilitating applications (such as automated warehouses) that could benefit by deploying fleets of such robots. We refer to these integrated systems as sensor network and control (SNAC) systems, to distinguish them from classical control systems, networked control systems, and systems for control of the network layer.

For a SNAC system to benefit from a sensor network platform, the platform needs to be easily deployable, self-organizing, robust, and long lasting. Many services are required for building a robust SNAC application such as self localization, ad-hoc routing, detection and aggregation, and control. As previously mentioned, most of these services have been studied in recent years. However, limited work has addressed the design of control systems capable of accurately utilizing the incoming information from sensor networks. Such composite systems are nontrivial to design due to the system interface dichotomy: control systems [31, 79] typically assume periodic, high frequency sensor updates whereas sensor networks provide aperiodic, low frequency, and laggy sensor updates.

The goal of our work is to develop a general methodology that facilitates high quality control services for SNAC applications. Additionally, our work focuses on the design of a control scheme for robot navigation in WSNs. Our research approaches these topic step by step. We implement a complex SNAC system and provide practical insight into the challenges it presents. We develop a general system architecture for approaching SNAC system design. We develop several WSN models and a simulator suitable for testing our architecture. Finally, focusing on robot navigation, we adapt localization and path planning techniques for WSNs.

1.1 Related Work

Previous work illuminating our research path is both varied and broad. In addition to addressing control systems and sensor networks, our work uses pursuit-evasion games and robot navigation as benchmarks and illustrative examples. Hence, a comprehensive survey involves delving into several research communities: classical control, sensor networks, distributed systems, estimation, mobile robots, and pursuit-evasion games. Additionally, fundamental to any mobile robot application is the ability for a robot to localize itself, requiring our survey to include results from the localization and tracking communities. In fact, developing a robust sensor network system for robot localization is useful outside our area of research in a variety of areas such as the hybrid field

of mobile sensor networks. Applications such as event sensitive monitoring [17, 70] require mobile sensors to gather around interesting events increasing the sensing accuracy.

This survey explores, in order, estimation techniques and control (and navigation) systems. Relevant pursuit-evasion game literature is left for the appropriate chapter. The estimation survey covers tracking in sensor networks, localization within sensor networks, and classical mobile robot localization. The control survey covers mobile robot control in sensor networks, classical mobile robot navigation, and extensions to classical control systems.

Localizing a mobile robot can be viewed as a sensor network tracking problem. Such problems have been studied in various forms. Early work [76] uses a combination of classifiers and Kalman filters. Tracking multiple objects is accomplished by first using a series of filters to associate new detections with entities being tracked. Then, the track of each entity is updated using a Kalman filter and the relevant detections. Other researchers [71, 73] working on this same application have employed Monte Carlo techniques. Oh et al [71] show the utility of Markov chain Monte Carlo methods for tracking multiple entities given noisy sensor network detections. The state of a Markov chain models the current estimate of tracks in the network. The transitions of the Markov chain are adapted to model simple transformations of the estimate such as track splitting or merging. The technique is shown to successfully track multiple objects given noisy sensor network detections.

Other sensor network tracking work [62] focuses on developing a generalized architecture for distributed estimation. In this work, the authors consider an architecture allowing nodes to estimate the position of entities using local sensor measurements and shared estimation probabilities from neighboring nodes. Techniques are developed for sharing estimation information amongst nodes to achieve favorable results. An architecture and a set of filters are generated that achieve a distributed Bayesian estimator. Additional work [13] by the authors provides a brief overview of the practical challenges of integrating these techniques for a real deployment.

Probabilistically tracking multiple objects in a efficient manner has also been studied [57, 58]. In this work, multiple objects are tracked using a particle filter. In general, a particle filter requires the use of a joint distribution encompassing probabilities for all the objects being tracked. As the number of objects being tracked increases, such a distribution quickly becomes intractable. An alternative method for managing the distribution is provided. The authors develop a method of joining and splitting distributions to allow objects close together to be tracked with a joint distribution and objects separated by large distances to be individually tracked by marginal distributions. This technique allows for the particle filter to operate on several simple distributions rather than one complex distribution.

Still other researchers have focused on tracking techniques that minimally impact the resources of the WSN. For instance, Zhao et al [97] propose that the execution of general information processing tasks (such as entity tracking) is realized as the solution to an optimization problem. The proposed objective function incorporates quantities such as information gain and communication costs. This technique, specialized for entity tracking, asserts that the estimated tracks are only updated as necessary, minimizing power and bandwidth utilization.

Entity tracking within sensor networks is indeed an active research area encompassing many scenarios and solutions. As already discussed, the focus of this work is on minimizing network resources, tracking multiple objects, or distributed information sharing. The focus of our work, on the contrary, is on the localization of a cooperative mobile robot. Hence, the current literature lacks an appropriate solution for our scenario: multiple object tracking algorithms are too general, distributed information sharing architectures are not necessary, and low resource utilization is not our focus. Furthermore, the cooperative mobile robot localization problem is decidedly different than the researched tracking scenarios: detections of our cooperative agent only occur locally and can be immediately sent (via broadcast) to the agent which assumes the role of a centralized estimator.

Before leaving the sensor network estimation literature, we expand our scope from tracking to localization and general estimation techniques. In this realm, researchers have studied many specialized localization applications such as sniper localization [83] and sensor node localization [93]. However, these applications are tailored for different sensor modalities and estimation of static quantities which are not applicable to the mobile robot localization problem. The mobile robot estimation problem within sensor networks has had limited coverage. In particular, Sinopoli et al [84] consider a linear control system with Kalman filter estimation supplied with sensor network detections. The sensor network is modeled as a monolithic, distributed sensor that either delivers a measurement on time or drops it. The authors show a relationship between the probability of dropping a sensor reading, the system dynamics, and the system stability. This work provides insight into the amount of network losses tolerated by a stable feedback control system. However, the sensor network model is rather limited. It does not model network latency characteristics, and, hence, has no method to incorporate late arriving measurements into the Kalman filter. In contrast, our work does not provide theoretical stability bounds, but instead provides efficient solutions for more realistic and practical sensor network scenarios.

The current sensor network estimation literature does not directly address our application, but does provide an implicit suggestion for basic localization techniques. At the root of these track-

ing and localization algorithms, are Bayesian estimation techniques (or approximations thereof). To provide more specific insight into applying Bayesian estimation, we turn to the mobile robot localization literature.

A large body of research on mobile robot localization has been carried out. Classical techniques such as Kalman filtering (KF) [50] and Markov localization (ML) [36] have been heavily investigated. Combinations and variants of these techniques have also been explored: extended Kalman filtering (EKF) [54], Markov localization-extended Kalman filtering (ML-EKF) [41], and multi-hypothesis tracking (MHT) [48, 5]. Additionally, a wide variety of sequential Monte Carlo methods, or particle filters [32], have been explored: sensor-resetting localization [53], mixture Monte Carlo localization [87], and adaptive Monte Carlo localization [27]. Comparisons of these localization techniques [42, 43] have been performed. In fact, any of these techniques are candidates for interpreting our sensor network data. However, the focus of our work is to develop techniques that provide an estimator with the best information. In the case of mobile robot navigation, we show that Markov localization is a particularly good choice. Since this body of literature tends to assume a traditional robot model with on-board, high speed sensors (such as GPS and laser range finders), these techniques are not adapted to the data characteristics of sensor networks. Fortunately, the mobile robot literature proves more fruitful as we move from estimation to control (navigation).

Mobile robot navigation and control is studied from a variety of perspectives. As we explore the navigation literature, we begin with topics directly related to sensor networks. Then, we expand our search to include generalized path planning topics and adapted classical control techniques.

Robot navigation has been investigated in the context of an autonomous robot deploying a sensor network and using it for coverage, exploration, and navigation [8, 9, 7]. In this work, each node periodically emits a message to nearby receivers. As a robot moves about, it selects the node it has recently received the most messages from as its *current node*. If no messages have been received recently, the robot deploys a new node and selects it as the current node. The current node is used as the robot's location. During the sensor network deployment phase, the robot and network build a graph that probabilistically maps from the set of current robot locations and control actions to the set of possible next robot locations. During coverage and exploration, nodes track which neighboring areas have least recently been explored and direct nearby robots to these areas. During navigation, the sensor network assigns a utility value to each node representing the likelihood of a robot nearby this node reaching the desired destination. The utility values across the network are updated using value iteration. The robot navigates, node by node, to the destination by iteratively navigating to

nearby nodes with the highest utility value. Although this system can explore and navigate areas, the graph it builds is not based on geometric measures, but rather received radio signal strength. This hinders the technique's ability to be combined with other sensing platforms such as laser range finders or GPS. This also makes it difficult for a robot to adapt to its environment by learning its dynamic parameters such as wheel slippage. A final drawback to this approach is that value iteration is slow to converge making it impractical for navigation with frequently changing way-points (such as pursuit-evasion games).

Robot navigation in sensor networks has been considered by other researchers [74, 56]. In this work, motes are placed at known locations and are capable of detecting danger. A system for navigating the network and avoiding danger is developed. Each mote and its neighboring region is assigned a danger level based on how many radio hop counts it is away from danger, implicitly building up a connectivity based graph for robot guidance. This information is used by an artificial potential field algorithm to guide the robot. This work assumes that the robot can localize itself to the extent of determining which nodes are nearby. Furthermore, it is assumed that the robot is capable of navigation to any desired (nearby) node.

Other researchers [22, 24, 23] investigated the ability of a unmanned aerial vehicle (UAV) to deploy, repair, localize, and navigate a sensor network. The UAV is capable of autonomous way-point navigation using an on-board computation and sensing system (including a GPS sensor). The goal for the sensor network deployment and repair phase is to create and maintain a desired network topology. The network is configured to pass messages around in order to detect connectivity holes. Once holes are identified, the UAV is dispatched to the region to deploy new motes. During network localization, the UAV flies a pattern above the sensor network broadcasting its GPS coordinates. Motes receiving these messages infer their locations in GPS coordinates using received radio signal strength. For navigation, an a priori generated path in GPS coordinates is flooded across the network and stored by motes near or on the path. The UAV follows this path by receiving partial path information messages from nearby motes. This navigation technique amounts to an elaborate path dissemination algorithm.

Das et al [28] consider a search and rescue system using mobile robots and sensor nodes. Nodes are deployed to loosely known locations in a map. The authors propose that both the nodes and the robot can be localized using traditional simultaneous localization and mapping (SLAM) [55] techniques. In particular, it is proposed that a Kalman filter is provided received signal strength measurements of messages passed amongst robots and motes. Navigation is achieved by first planning a path along a string of nodes from the current position to the goal position. The network chooses the

string of nodes by passing messages around and achieving a dynamic programming type solution. The robot then follows the planned path using a gradient decent routine.

Our survey could additionally broaden its scope to more generalized sensor networks such as camera networks. However, the research in these areas is not applicable to our SNAC system due to differences in the platform's characteristics (such as fast, reliable networks with slow robot dynamics [47]) or differences in the estimation architecture (such as switching amongst sensors [52] rather than multiple sensor measurement fusion). The literature on mobile robot control and navigation with sensor networks is quite sparse. Techniques that are relevant for our platform use non-geometric based graphs for navigation. However, our work seeks to develop a geometric localization and navigation technique allowing integration with traditional sensors (such as GPS, laser range finders), admitting robot learning (such as learning the robot's wheel slippage), and enabling fine grained control. Furthermore, previous navigation techniques based on received radio signal strength are suited to only specialized environments (such as simple indoor environments with controlled radio connectivity).

The mobile robot community has extensively investigated navigation and control. However, the focus is typically on topics non-applicable to our work such as obstacle avoidance, map building, or coordinated movement. Fortunately, previous work on intelligent path planning is helpful. Makarenko et al [63] consider the problem of integrated exploration entailing path planning, navigation, and localization. The authors assume a traditional robot platform utilizing SLAM [55] techniques with an extended Kalman filter for mapping and robot localization combined with an occupancy gird for obstacle avoidance. In particular, the robot movement algorithm proceeds by suggesting several candidate destinations, choosing a destination, and navigating to the chosen destination. The candidate destinations are suggested for improved exploration. A single destination is found as the solution to an optimization problem over localization accuracy, information gain, and travel time. Finally, the robot navigates to the destination by planning a path that avoids obstacles.

Our work focuses on navigating not for obstacle avoidance, but for improved localization. Hence, both our work and the Makarenko et al work exploits a localizability metric to gauge a position's ability to accurately localize the robot. However, our metric is adapted for a distributed sensing environment and is optimized at every point along the path, rather than at just a few destinations. Additionally, since our focus is on sensor networks, it will become clear that the Gaussian noise model implicit with Kalman filtering is inappropriate. Instead, our work develops a specialized sensor model for use with Markov localization. Finally, the development of our localizability metric is different from the one used by Makarenko et al. In particular, the metric our work uses is

provided in the work by Roy et al [78], but is adapted to our distributed sensor network model. There are a few distinctions between the Makarenko et al and the Roy et al localizability metric. In particular, the former assumes infinite initial covariance and infinite readings at each position, whereas the later assumes a well-defined prior with finite covariance and assumes an expected number of readings at each position. Our work will clarify why an infinite initial covariance is inappropriate for the multiple, local sensor platform of a WSN.

Earlier work [86] utilizing a localizability metric for path planning has also been carried out. This work assumes a robot platform with a ranging sensor and Kalman filtering (similar to the work by Makarenko et al), and a localizability metric consistent with the work by Roy et al. Unfortunately, this work concluded with hand generated path producing less drift during navigation than a path based on the localizability metric.

Moving away from the mobile robot navigation and control literature and entering the classical control arena, our survey finds more practical tips. Some researchers [64, 65] developed real-time compensation techniques for networked control systems. Their work describes a technique that compensates for sensing and actuation jitter for linear, discrete-time systems. The authors assume the jitter is known at run-time and develop an algorithm that simulates the system forward in time and recomputes the control action. Although this technique is only described for linear, discrete time systems, our research applies this idea to an extended Kalman filter to correct for network latency.

Predictive control techniques are also useful for incorporating several goals and dealing with missing or late sensor measurements typical of sensor networks. For instance, Shim et al [82] develop a model predictive control (MPC) framework for navigation of aerial vehicles. The authors use a MPC to incorporate several goals such as obstacle avoidance and way-point navigation. Using a finite time horizon, the MPC is able to plan locally optimal navigation routes. Additionally, such a framework is able to continue operating using only predicted results during periods of missing measurements. Our general control architecture will avail itself of this technique.

Before leaving behind the literature on control and sensor networks, we mention one final result for the interested reader. In a paper by Ye et al [96], the effects of communication protocols on control systems are investigated using realistic network simulations. The authors consider a mobile robot cooperation application where communication between robots is simulated using the Network Simulator [12]. It is demonstrated that a communication scheme with less frequent, but longer packets is superior to one with more frequent, but smaller packets for their application. This counter-intuitive result reminds us that more research is needed on networked control systems.

After a brief overview of the literature relevant to sensor network and control systems – and, in particular, to robot navigation – we find that many topics have not be addressed. In particular, since sensor networks is a new research area, few researchers have considered their use for control or robot navigation. Additionally, since only a few medium to large scale sensor networks have been deployed, our knowledge of practical challenges is limited. Finally, to the best of our knowledge, only one medium to large scale SNAC system has been deployed. This system was our pursuit-evasion game system and is discussed in the next chapter.

1.2 Dissertation Contributions

Our work differs from previous work in several ways. The goal of our work is to develop a practical design methodology for realistic SNAC systems. To this end, our work allows for complex models of the WSN and the control system, and our work focuses on demonstrating good performance rather than finding theoretical bounds on it. We are motivated by the fact that the theoretical work tends to be too specialized to be useful for real deployments, and the practical implementations have yet to adequately address fundamental control challenges.

Our work achieves these goals by implementing, modeling, and evaluating 2 representative SNAC applications: pursuit-evasion games and way-point navigation. Our work includes one of the first, and the largest SNAC implementations. This implementation provides insight into the difficult challenges facing control systems in sensor networks and aids in developing a generalized SNAC system architecture. This work also leads us in developing several realistic application-level WSN models and a corresponding simulator. Using this simulator, we are able to demonstrate the benefit of our SNAC system architecture. The analysis of this system leads us to consider intelligent path planning for robot navigation. Focusing on navigation, we adapt a localizability metric for WSNs, we create a novel path planning routine, and we demonstrate the benefit of this approach. Our final framework allows for simple, yet powerful extensions to path planning and robot localization such as path planning for explicit bandwidth reduction or development of an optimal re-deployment strategy. In summary, these contributions are

- first, largest SNAC implementation
- identify fundamental SNAC challenges
- develop a general SNAC system architecture

- develop practical, application-level models for WSNs
- develop an application-level simulator for SNAC systems
- demonstrate the benefit of our SNAC system architecture
- tailor a localizability metric for WSNs
- develop and demonstrate the benefit of a novel path planning technique for robots in WSNs
- establish a localization and path planning framework enabling simple, powerful extensions

1.3 Thesis Outline

Our work is presented in the following manner. Chapter 2 presents our implementation of a pursuit-evasion game played within a wireless sensor network. First, the game is introduced with relevant background and related work. Then, the hardware and software platform is described. Next, the algorithms for sensing, communication, and control are described. Results collected from the deployment are discussed. Finally, the practical lessons learned and the challenging aspects of this application are presented.

Using the results from our PEG deployment, Chapter 3 develops a practical system architecture for general SNAC systems. Three main challenge categories along with potential solutions are discussed. This chapter concludes with a framework that unifies these solutions. Chapter 4 develops and uses a SNAC system simulator. First, a set of system models are developed for the WSN and robot platform. Then, several components of the aforementioned architecture are instantiated. Finally, a set of simulations are performed to test our controller design and evaluate the effect of path planning on localization accuracy.

The path planning simulations leads us to adapt a localizability metric for WSNs in Chapter 5. This chapter begins with an overview of robot localization and a localizability metric. Next, the details of computing this metric are discussed and a set of simulations is performed. The next chapter, Chapter 6, moves to adapt this metric for more realistic WSN models and to begin testing its usefulness for localization and path planning. Several simulations are provided to demonstrate the ability of this metric to gauge localization performance.

Building on this result, Chapter 7 develops techniques to generate automatic paths based on the localizability metric. An interpretation of entropy for path planning is given and several

closed-loop simulations are provided as evidence this technique reduces localization error. Finally, Chapter 8 reviews the results of this work and provides some concluding remarks.

Chapter 2

Pursuit-Evasion Game

To date, we have implemented several sensor network and control (SNAC) systems, ranging from light monitoring and tracking, to RC car tracking with a pan-tilt camera, to pursuit-evasion games on outdoor sensor networks [81, 21]. This chapter provides an overview of the July 2003 pursuit-evasion game implementation, providing enough detail to serve as a point of reference for later discussions on challenges and solutions for SNAC systems. Our pursuit-evasion game, or PEG, is a distributed network of wireless sensors that aids a cooperative agent (pursuer) in tracking an uncooperative agent (evader). These games, in various forms, have been extensively studied. Researchers [45, 44] have investigated the game theoretic optimal strategies developing Nash solutions. For continuous (polygonal) regions, other researchers [95] have determined bounds on the number of pursuers needed for capture. For games occurring on a graph, researchers [72] have studied how to search for evaders, and others [1] have studied the number of pursuer needed to catch an evader. For a distributed sensing platform, Demirbas et al [29] have studied pursuit-evasion games on graphs whose vertices are covered by sensor nodes. The authors assume that communication is symmetric and induces a fully connected graph. Additionally, they assume that sensor nodes can exactly detect the pursuer and evader when the robots are at the node. Finally, they assume that a pursuer robot can a priori navigate between nodes. The authors develop a tunable pursuit algorithm that allows for a trade-off between capture time and energy efficiency. The algorithm functions by having the network maintain a tree rooted at the evader. The tree is updated with recent detections of the evader. Our work, considers the practical challenges for implementations of pursuit-evasion games on sensor networks. Other work [91, 51] implementing pursuit-evasion games has focused on different platforms, considering vision based robots or multiple pursuer policy choices. PEG provides a real world, medium scale (larger pure sensing systems [21, 3] have been deployed since)

SNAC system deployment from which we learn simple, effective, pragmatic design and modeling methodologies. PEG provides an extensive SNAC benchmark requiring solutions to many challenges:

- detection and disambiguation of mobile agents
- distributed coordination and sharing of mote resources
- in-network processing and aggregation
- routing to mobile agents
- closed-loop control

The services developed to confront these challenges are developed with the whole system in mind to reduce overall latency and provide the pursuer the ability to react in real-time.

Being one of the first medium scale, distributed tracking implementations using resource constrained wireless sensors, PEG provides us with a road map of the challenges ahead, new ideas from the lessons learned, and a grocery list of tools needed. The hardware and software platform along with the basic information flow is introduced in Section 2.1. Section 2.2 provides a detailed overview of the algorithms used. The results are summarized in Section 2.3. Finally, Section 2.4 discusses the lessons learned from PEG. In later chapters, the experience with PEG informs an exploration of the challenges control systems face along with providing a starting point for developing SNAC models and simulators. More details of the entire PEG deployment is presented in our published work [81].

2.1 Platform

This section describes the hardware and basic software architecture used for PEG. PEG is played on a 20 meter by 20 meter outdoor field with 121 motes and 2 ground robots. The motes are uniformly deployed in an 11 by 11 grid with 2 meter spacing. The robots are sequentially released onto the field: first the evader enters, then a few seconds later, the pursuer enters. A base station outside the field monitors and displays system statistics online. In the following, we describe the mote, robot, and base station hardware along with the basic flow of information amongst them.

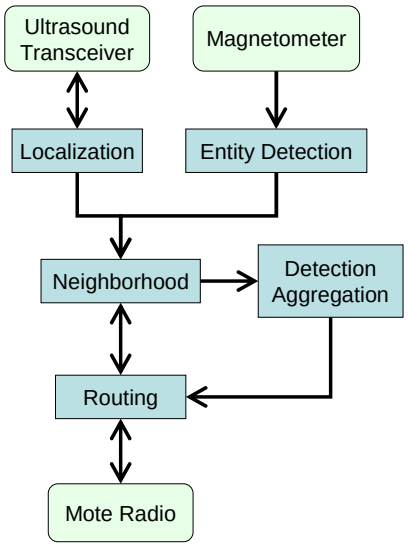
The motes used for PEG are the Berkeley Mica2Dot's [46, 88] shown in Figure 2.1a. They are equipped with a wireless radio, ultrasonic transceiver, magnetometer, and outdoor enclosure.

(a) Mica2Dot mote. (b) Pioneer robot.

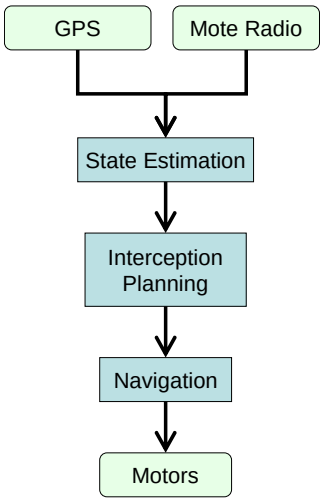
Figure 2.1: Hardware used during the July 2003 pursuit-evasion game deployment.

The computational resources are provided by the 8-bit 4 MHz Atmel ATMEGA128L CPU with 128 kB of instruction memory and 4 kB of RAM. The on-board radio, the Chipcon CC1000 operating at 1 GHz, provides about 2 kB/s of shared network bandwidth for applications after accounting for network overhead. The communication range varies heavily depending on the antenna and the environment; during PEG, with the use of a quarter wave length antenna and the mote a few inches off a level playing field (comprised of dirt and short grass), our maximum communication range was about 30 meters. The ultrasonic transceiver operates at 25 kHz and uses a top mounted deflector cone to provide about 2 meters of ranging in the plane of the playing field. The magnetometer is tuned to detect changes in the magnitude of magnetic field near the mote; it detects our robots up to about 3-4 meters away. The hardware is mounted in an outdoor enclosure with a flexible base allowing for extended communication range and resilience to robot impact. Finally, the motes run the small, embedded TinyOS [46, 88] operating system

The evader and pursuer robots are identical outdoor, four-wheeled robots (Figure 2.1b) equipped with mobile computers, wireless radios, and GPS units. The on-board computer is a 266 MHz Pentium2 CPU with 128 MB of RAM and a 20 GB hard drive running Linux. Each robot uses an 802.11 wireless radio for communication to the base station. The GPS unit provides the robot's



(a) Mote data flow.



(b) Robot data flow.

Figure 2.2: Flow of information (magnetic detections and position) during the pursuit-evasion game deployment.

position to within 2 cm every 0.1 seconds. Each robot has a top speed of about 1 m/s and a motor control subsystem capable of independently controlling the velocity of each of the four wheels. The evader robot is driven remotely by a person, and the pursuer robot is controlled autonomously by on-board control software. The base station uses a 802.11 wireless link to receive robot controller statistics and a high-gain mote antenna to snoop the mote network.

The final software architecture contains many application services related to entity detection and pursuit (localization, sensing, routing, state estimation, interception planning, etc) and even more services supporting these (ranging, tree building, leader election, etc). We provide a brief overview of only those services directly along the entity detection and pursuit path. A flow chart of these services, split between the sensor network and robot hardware, is shown in Figure 2.2.

The mote data flow, shown in Figure 2.2a, is designed to provide detection reports to the pursuer. Since these reports relay the location and strength of a magnetic detection, a localization service and entity detection service are fundamental. The localization service utilizing the ultrasound transceiver (and sub-services such as ranging) is responsible for localizing the mote relative to an anchor mote in the network. Due to sporadic errors of this service, the motes used a position value provided by the user during the PEG trials. The implementation and applicability of the automatic localization service is further explored in other work [93]. The entity detection service filters

measurements from the magnetometer and reports robot detections. The information collected by the localization and entity detection service is stored and shared by the neighborhood service. This service provides a distributed, loosely synchronized storage system allowing neighboring motes to share information. Using the information stored by this service, the detection aggregation service compiles location and detection information to be sent to the pursuer. Finally, the routing service is responsible for dispatching messages amongst motes and the mobile pursuer.

The robot data flow, shown in Figure 2.2b, is designed to interpret the detection reports from the mote network and guide the pursuer to the evader. The pursuer receives entity detection reports over its mote radio and pursuer position information over the GPS channel. Using this information, the pursuer estimates the state (position, orientation, etc) of itself and the evader. The interception planning service determines the path the pursuer should follow in order to efficiently capture the evader. Finally, the navigation service interfaces with the motors and ensures the desired path is followed.

2.2 Algorithms

This section describes the algorithms making up the aforementioned services. Before describing the flow of detection events within the sensor network or the pursuer, we investigate the mechanism that links these two systems: the routing layer. Then, starting with the mote platform, we explain the mechanisms that sense entities, share detections, and aggregate readings. Next, turning to the robot platform, we discuss how detections are processed, how the system state is estimated, how an interception path is planned, and how this path is followed.

The mote routing layer is responsible for delivering messages from many (motes) to few (mobile agents). Using landmark routing [89], we partition the routing service into 2 pieces: route from many to a landmark, and route from a landmark to few. Landmark routing requires that the user chooses a landmark and, using the landmark as a root node, builds a spanning tree. A spanning tree is built by flooding the network with a beacon that tracks hop count. Each mote will choose its parent as the mote whom it heard broadcast the beacon with the lowest hop count. However, this approach is vulnerable to generating asymmetric links and back edges.

Asymmetric links are generated because a mote does not verify the bidirectional reliability of the link between itself and its parent. We address this issue by filtering out messages that originate from an unreliable link. In particular, if the message is received with a received signal strength indicator (RSSI) below a preset threshold, or from a mote located too far away according

to localization information, the packet is dropped. The RSSI threshold is set to a value that provides a highly reliable link based on small scale experiments using the same platform.

Back edges are caused by frequent message collision amongst motes re-broadcasting the beacon. These collisions cause a nearby mote to choose a parent whose beacon messages did not collide, often resulting in a higher hop count parent than necessary. This issue is addressed by requiring that each receiving mote waits a random amount of time t_b less than an upper threshold T_b before broadcasting the beacon. Additionally, if another beacon is heard before a mote has broadcasted, it must choose another time $t_b < T_b$ and begin waiting again.

Once a spanning tree is built, any mote can route to the landmark. Routing to the mobile agent is achieve by building a *crumb trail*. The mobile agent periodically broadcast a beacon to a nearby mote. Using the spanning tree already in place, this beacon is routed to the landmark. Along the way, each connecting mote is left with a *crumb*, the identity of the previous mote during this communication. Any message at the landmark can now be routed back to the mobile agent by reversing the crumb trail. Additionally, crumbs are associated with a timeout and are erased from motes when they become stale. With the routing service in place, we turn to the mote's internal operations.

The detection sequence at the m^{th} mote begins with the entity detection component. This component is fed measurements of the absolute magnetic field via the magnetometer. Due to drift in the magnetometer, the raw magnetic field readings B^m_{raw} are high pass filtered to generate B^m. If the processed value B^m is above a preset threshold B_{report}, B^m along with the mote's position z^m is broadcast to neighboring motes via the neighborhood service. After which, the mote must wait for T_{report} seconds before reporting another reading. We refer to these single magnetometer reading packets as *detection messages*.

The local neighborhood information sharing implementation used was Hood [94]. Hood operates by broadcasting local updates to the state disregarding whom its neighbors are or if they can hear it. Motes who hear a Hood broadcast and consider the broadcasting mote to be a neighbor, update their local state. For our purposes, Hood was configured to form a magnetic detection neighborhood with radius R_{hood}. Additionally, this neighborhood was set to prune detections older than T_{hood} seconds.

The mote constantly monitors the neighborhood state, and, if it has the largest magnetic detection of any of its neighbors, the mote elects itself as a leader. As a leader mote, it aggregates all the detection readings it received in the last T_{report} seconds into one packet and sends this via the routing service to the pursuer. We refer to these aggregated magnetometer reading packets as *event*

messages. Once a mote is a leader, it must wait T_{leader} seconds before becoming a leader again.

When the pursuer receives an event message, it updates its estimate of the system state, and using this estimate, it tracks the evader. The block diagram for this process is shown in Figure 2.3. Two coordinate systems are used by the pursuer: GPS coordinates relative to a nearby landmark and mote network coordinates aligned with the mote deployment grid. Details of these coordinate systems are not important since a fixed homogeneous coordinate transformation between the two systems is assumed to be known. Unless explicitly stated by using a GPS subscript, all quantities are in terms of the mote coordinate system. The pursuer receives $(\tilde{l}^{p,1}_{gps}, \tilde{l}^{p,2}_{gps})$, an estimate of its field position in GPS coordinates, which is transformed into the pursuer position estimate $(\tilde{l}^{p,1}, \tilde{l}^{p,2})$ in mote coordinates.

Many techniques [77, 90] exist for tracking and estimating moving entities. However, the choice of such algorithms is limited by the robot dynamics and sensors. In particular, given the accuracy of the GPS, the pursuer's state is estimated using simple averaging techniques. More specifically, $(\tilde{l}^{p,1}, \tilde{l}^{p,2})$ are smoothed with a N^p_l step time window average to produce the position estimate $(\hat{l}^{1,p}, \hat{l}^{2,p})$. Then, the N^p_θ most recent position estimates are used (pairwise) to form several orientation estimates, which are averaged to form the orientation estimation $\hat{\theta}^p$.

The pursuer estimates the evader's state using received event messages. First, each detected position $(\hat{l}^{?,1}, \hat{l}^{?,2})$ is classified as being caused by noise, the pursuer, or the evader. After which, only evader detections are kept and sent to the evader state estimation service. Several techniques exists [71, 15, 33, 40] for classifying and tracking objects in a sensing region. For estimation of the evader, we note that since our robots can change their wheel speed within about 0.2 seconds, and the network only reports detections about every 1-3 seconds, we can use a simple kinematic model accounting only for a robot's maximum speed s_{max}:

$$l_{k+1} = l_k + (s_{max}T)u_k \tag{2.1}$$

where $s_{max} > 0$, the control $u_k \in [-1, 1]^2$, and the sampling period $T > 0$. With our model, we associate measurements close to the evader with the evader, and using an average of these measurements, we estimate the evader's position. In particular, assuming that the error in any detected position may be as much as d_n (according to the 2-norm), and that the capture radius d_c is larger than this error, detections within d_n of the pursuer's estimated location are ignored. These messages, if not detections of the pursuer or noise, are detections of a captured evader. Furthermore, any detections farther than $2d_n + s_{max}t$ from the evader's last estimated position, are considered noise and ignored where t is the time since the last measurement. All remaining messages are assumed to be

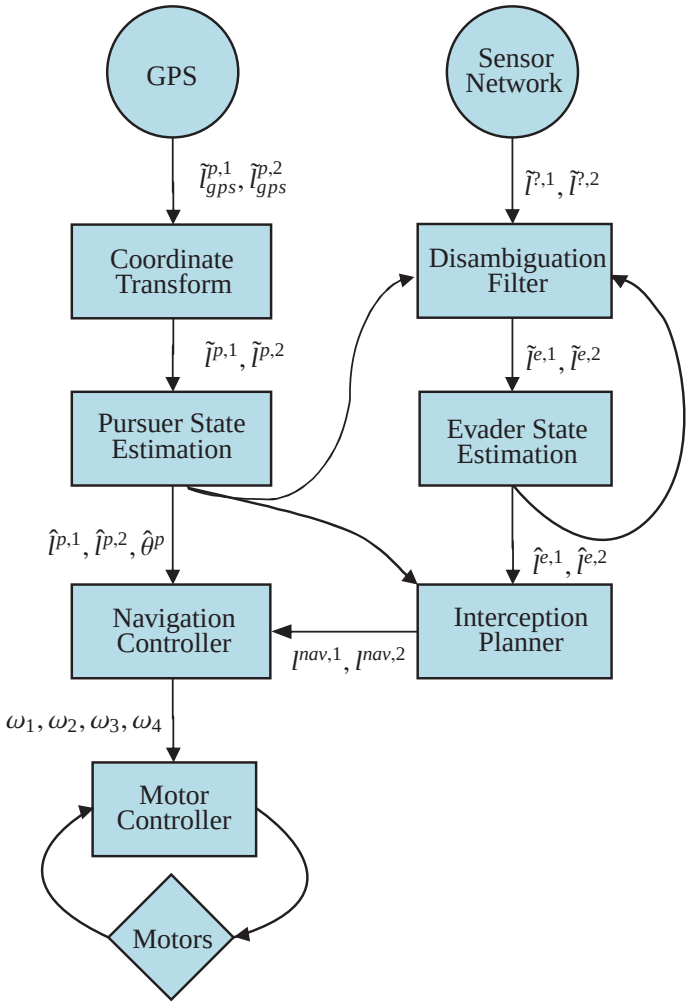


Figure 2.3: The pursuer's estimation and pursuit control system.

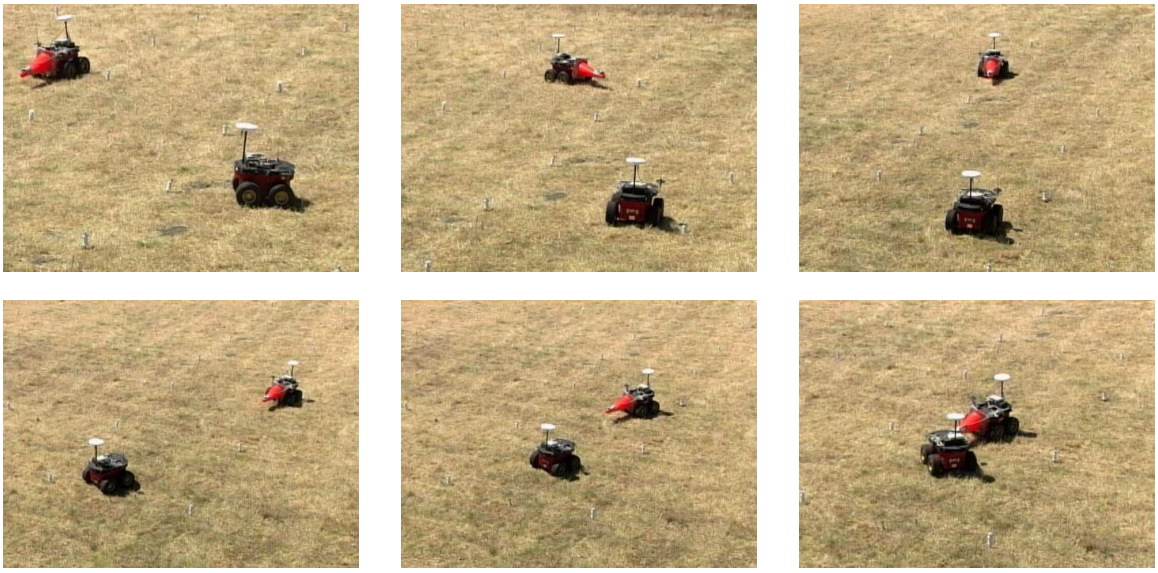


Figure 2.4: Stills from a 26 second pursuit-evasion sequence. The pursuer and evader start in the upper left and lower right corners, respectively. The evader is captured in the last frame.

the position detection of the evader and are labeled $(\tilde{l}^{e,1}, \tilde{l}^{e,2})$. These messages are used to estimate the position of the evader $(\hat{l}^{e,1}, \hat{l}^{e,2})$.

Using the estimated state of the pursuer and evader, the interception planner generates a way-point $(l^{nav,1}, l^{nav,2})$ for the pursuer to go to in order to intercept the evader. Our interception planner simply generated the way point at the evader's estimated position. The interception controller also implements 2 safety specifications: keep the pursuer inside the playing field and do not collide with the evader. Keeping the pursuer inside the playing field is achieved by planning a path to the center of the playing field if the robot ever leaves. Avoiding collisions with the evader is achieved by halting the pursuer when it is within the capture radius of the evader and restarting the pursuer when the evader moves farther away. Finally, the robot is directed to the new way-point via the point navigation controller managing the robot wheel velocities $\omega^1, \omega^2, \omega^3, \omega^4$.

2.3 Results

PEG was deployed outdoors and many of the parameter values were set after online testing. In particular, B_{report} was set so most motes would detect a robot within 3-4 meters. Mote neighborhoods were set to include a 9 by 9 grid of motes by letting $R_{hood} = 3$ meters. The timeouts

were chosen to be 0.5 seconds: $T_{report} = T_{hood} = T_{leader} = 0.5$ seconds. Finally, the pursuer's position and orientation were estimated by windows of length $N_l^p = 2$ and $N_\theta^p = 4$, respectively.

PEG was ran a half-dozen times, and, in each run, the pursuer successfully captured the evader. One particular run is shown in Figure 2.4. In the first frame (upper left), the pursuer (robot with the orange cone attached) is shown in the upper left and is oriented down and to the left. The evader is in the lower right and is oriented facing right. As the sequence proceeds, the evader (driving in reverse) turns towards the bottom of the frame and drives away from the pursuer. Meanwhile, the pursuer turns to face the evader and drives towards it. The last frame shows the result: the evader is captured. From start to finish, this sequence spans about 26 seconds.

Unfortunately, PEG was insufficiently instrumented to capture the necessary data. Hence, we instrumented and re-deployed PEG on a 7 by 7 grid with 2 meter spacing. The primary focus of our re-deployment was to study the detection-routing-reaction chain (latency, loss, noise, etc). To study this, a single robot is driven around in the field and the network activity is monitored. In particular, a base station snoops on the network packets and the robot's GPS position. Figure 2.5 shows 3 views of such a run: an overhead view of mote detections, an overhead view of estimated robot position, and a time line of detection and estimation error. Figure 2.5a provides a view of the robot path in the mote network. The solid line is the GPS measured path of the robot which starts at about $(0.4, 7.6)$ at $t = 0$ seconds and concludes around $(-0.1, 9.6)$ at $t = 145$ seconds. The robot path is additionally demarcated by a square for every 20 seconds of run time. Motes who elect themselves as leaders and report event messages are shown as blue stars. A dashed blue line is drawn from each leader to the position of the robot at the time the event message was received. Initial tests of this deployment revealed that the motes at $(4, 10)$ and $(4, 12)$ were frequently reporting false detections; these mote detections were disabled to prevent them from saturating the network, and, hence, are not shown in this figure.

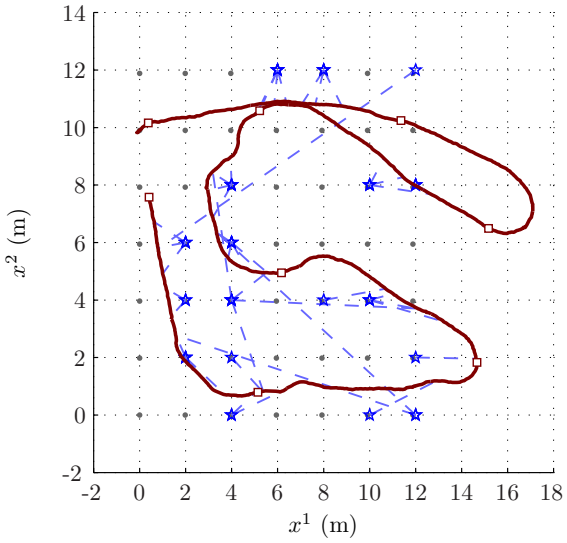
Sifting through this figure, we notice several phenomenon. Short dashed lines tend to indicate a good detection with minimal latency. For example, the motes at $(10, 8)$ and $(12, 8)$ tend to report the robot's position reliably and quickly. Slightly longer dashed lines indicate a higher latency communication link between the leader mote and the robot. Long dashed lines, such as the one connecting to the mote at $(12, 12)$, represent false detections. The motes at $(12, 12)$, $(12, 0)$, and $(4, 4)$ were misbehaving and frequently reporting false detections. Poor detections made robot position estimation difficult as shown in Figure 2.5b. This figure removes the leader indicators of the previous figure and adds the estimated robot position (solid orange line). The estimated robot position was generated offline using a Kalman filter and the sensor network reports.

The Kalman filter assumes simple (linear) point mass kinematics with no movement noise. The sensing error covariance is tuned to the detection error (see the blue dashed line in Figure 2.5c). Additionally, the Kalman filter is initialized with a perfect state estimate. In effect, the Kalman filter is simply acting like a smoothing function for the detections. From Figure 2.5b, it is clear that the system is extremely noisy especially with the false reports. Figure 2.5c shows a time line of the sensor network detection error (blue dashed line) and the Kalman filter estimation error (solid orange line) with detection receive times (light blue vertical lines). The detection error is calculated as the difference between the robot's GPS measured position and the sensor network's most recently reported detection. Clearly, this error is quite high. Often, the robot is not even detected within the correct grid cell or even the correct 9 by 9 grid cell surrounding the robot. In fact, the mean error of raw detections is 2.42 meters, indicating that the robot is often detected in the wrong grid cell. Over the course of the 145 second journey, the robot receives 48 detections, providing an average of 1 detection every 3.02 seconds.

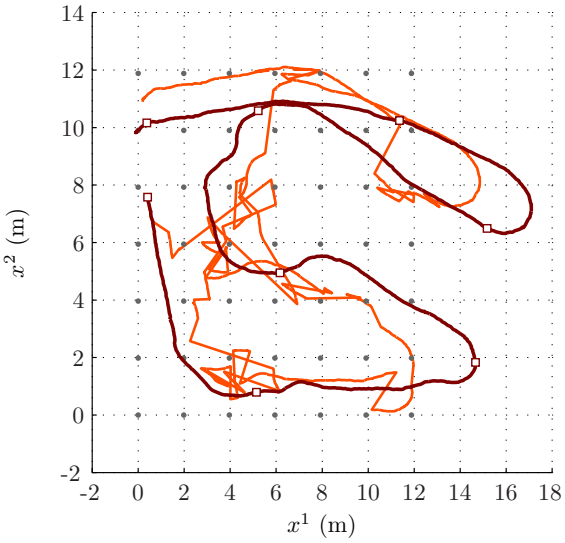
Additionally, as we will discuss shortly, we estimate the mean latency of the network to be about 1.75 seconds. Given this high latency, low bandwidth, and high error system, our Kalman filter (with perfect initial state and zero movement noise) still generates 1.96 meters mean position error; the unprocessed detections have 3.25 meters mean position error. Clearly, a real system with movement noise, running a feedback loop on such data would be quite limited in speed and accuracy; compare this with a 10 Hertz, 2 cm accuracy GPS system, for instance.

To understand the intrinsic quality of the sensor network data, we more aggressively filter the raw detections to generate the results shown in Figure 2.6. In particular, we remove reports from (additional) misbehaving motes by silencing motes $(12, 12)$, $(12, 0)$, and $(4, 4)$. To account for latency, we shift the reports backwards in time and observe that a time shift in the range of 1.6 to 2.5 seconds is helpful in reducing the detection error. We observe that a time shift of 1.75 seconds provides a minimum amount of detection error. After which, we apply the same Kalman filter to estimate the robot position. Our robot now only receives 41 measurements (about 1 measurement every 3.54 seconds), but is able to achieve a mean detection error of 1.69 meters (13.78% less error than the previous Kalman filter estimate).

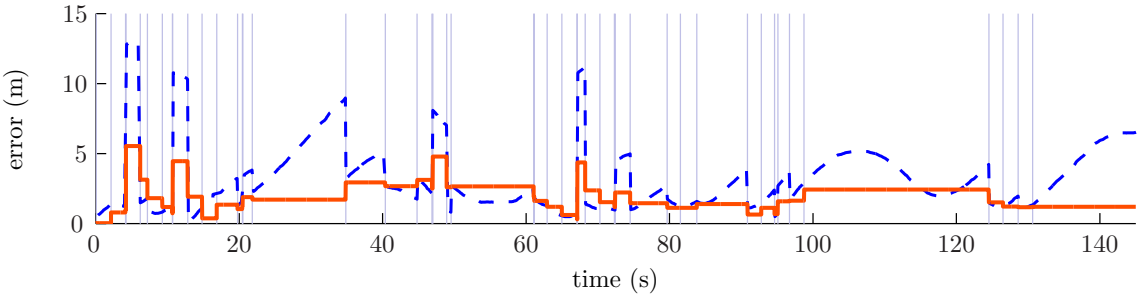
Over the span of this path, the detection error averages 2.60 meters, and 1.53 meters after application of the Kalman filter. We observe that even after applying our (non-causal) filter to hand picked measurements, the sensor platform still proves challenging for support of high speed, high accuracy control systems. The next section summarizes the lessons learned during this deployment.



(a) GPS measured robot path (solid black) demarcated every 20 seconds (black squares) with leader motes (blue stars) and received message times (blue dotted lines).

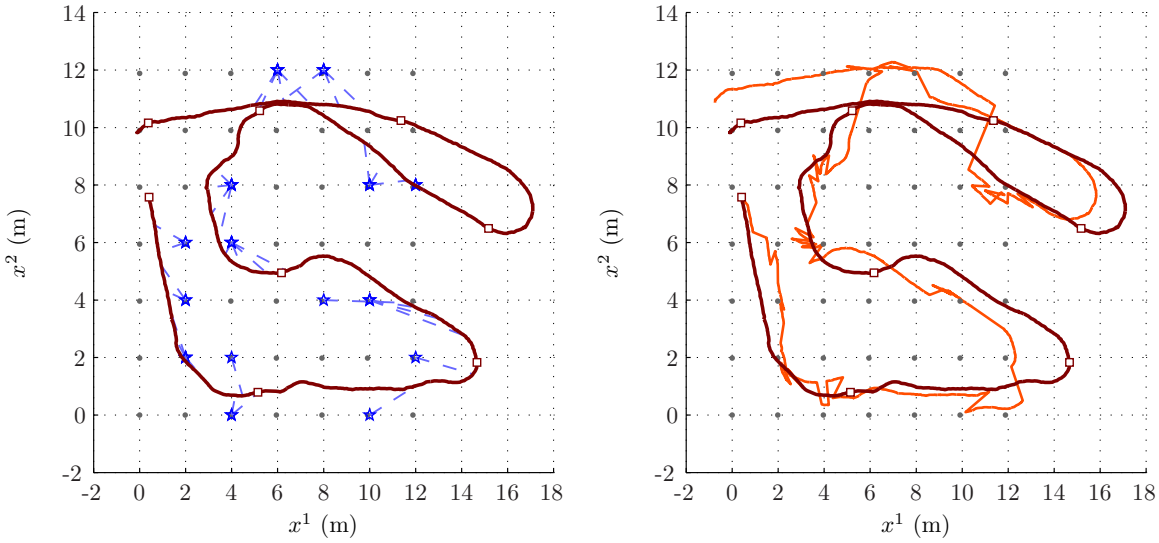


(b) GPS measured robot path (solid black) demarcated every 20 seconds (black squares) with Kalman filter estimated position (solid orange).



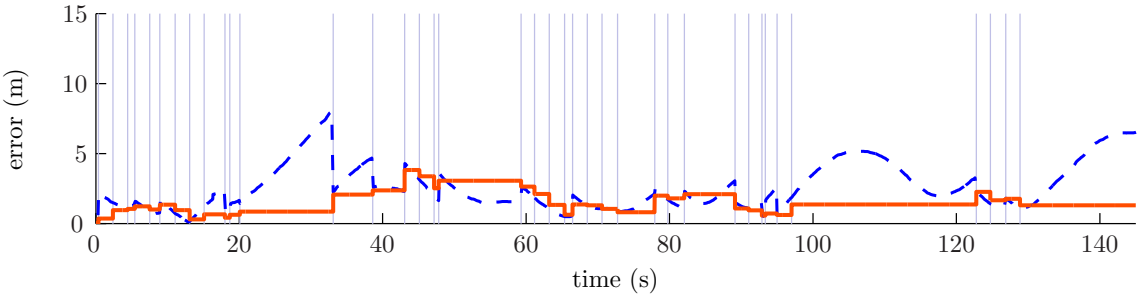
(c) Position error: sensor network detected (dashed blue) and Kalman filter estimated (solid orange).

Figure 2.5: Results collected from a single robot traversing a 7 by 7 sensor network. Faulty motes $(4, 10)$ and $(4, 12)$ have been manually silenced.



(a) GPS measured robot path (solid black) demarcated every 20 seconds (black squares) with leader motes (blue stars) and received message times (blue dotted lines).

(b) GPS measured robot path (solid black) demarcated every 20 seconds (black squares) with Kalman filter estimated position (solid orange).



(c) Position error: sensor network detected (dashed blue) and Kalman filter estimated (solid orange).

Figure 2.6: Results collected from a single robot traversing a 7 by 7 sensor network. Faulty motes ((4, 10) and (4, 12)) and noisy motes ((12, 12), (12, 0), and (4, 4)) have been manually silenced.

2.4 Lessons

PEG taught us many practical lessons for design, development, and deployment of a medium-scale sensor network and control system. We learned that mote enclosures should be tailored specifically for development or deployment, that every physical interaction with a mote is likely to cause damage, that a high gain antenna snooping on the network is an invaluable debugging tool, and that many additional software services for in situ debugging and interaction are needed. However, here, our focus is on using the sensor network as a sensor for a control system.

From a control system's perspective, perfect WSN data would be such that the node locations are known, and, for all time, the physical quantity to be sensed is known at every node with no latency. However, PEG demonstrated our controller would, aperiodically and with a low data rate, receive noisy measurements at loosely known regions of space-time. We categorized properties of the PEG WSN that make traditional control techniques challenging to apply:

- Sensor error
  - sensor noise
  - modeling error
  - inter-mote calibration error
  - timing error
  - localization error
- False events
  - spurious detections
  - missing detections
- Network induced error
  - aperiodic
  - low data rate
  - latency

Sensor error refers to the error in the sensing system during detection of true events (i.e., excluding any false positives or negatives) due to sensor noise, modeling error, inter-mote calibration error, timing error, and localization error. Additional error is induced in the system by spurious

and missing events, caused by, for example, faulty hardware or environmental noise. Finally, network characteristics denote the tendency of sensor measurement packets to arrive aperiodically at a low rate and with significant latency.

Just as other researchers [85] have discovered experimentally the difficulties of sensing applications using sensor networks, PEG has done this for control systems. This chapter has described the difficulty in using sensor network platforms to inform control systems. In the next chapter, we identify several techniques to overcome these difficulties. Later chapters evaluate the performance of these techniques.

Chapter 3

Practical System Architecture

Our PEG deployment demonstrated that several properties of sensor networks (discussed in Section 2.4) frustrate control system design. This chapter investigates techniques for overcoming these challenging properties and develops an architecture that unifies these techniques. The aforementioned challenges can be address in many ways ranging from hardware design to algorithm parameter choice. In our work, the localization and routing algorithms along with the mote hardware is assumed to be fixed and given. Additionally, the node density and deployment strategy is also assumed to be provided a priori. Hence, our approach focuses on using combinations of simple algorithms to improve the overall system performance. In later chapters, the developed architecture is tested with a series of detailed simulations.

Furthermore, as our architecture is evolved, we will keep in mind that many useful sensor network services have already been developed. In particular, services such as localization [93], time synchronization [34], data query and aggregation [60], and run-time configuration utilities [69] have been developed. Additionally, other architectures for networked control systems have been studied. Brooks [14] developed an architecture that allows for graceful degradation of a control system as connectivity decreases, provided that actuators always receive input from the lowest level controller, a controller designed to implement basic, safe functionality. However, the design of such a controller is left to the reader. Our work seeks to develop an architecture for SNAC systems based on existing services and seeks to design such a control system.

3.1 Sensor Error

Sensing error due to noise, modeling disparities, inter-mote calibration and timing differences, and localization error can be addressed in a variety of ways. A solution to noise is to obtain more frequent samples. However, the number of samples is fundamentally limited by both the hardware and the bandwidth of the network, in addition to any self-imposed limits designed to meet other goals such as reduced power consumption. Many of theses requirements can be addressed by efficient use of the bandwidth. In particular, for a network that supports multiple clients and objectives, the networking layer can operate in a *multimodal* fashion, each mode tailored to each objective. For instance, during PEG, the network could operate both as a localize pursuer service and a track evader service. Messages used to localize the pursuer could be designed to incur less overhead since they require only single hop communication.

Additional error due to modeling discrepancies, inter-mote calibration differences, and localization error can be addressed with a space-time model of the system variables and an accompanying estimation technique (such as probabilistic models with maximum likelihood estimation techniques). We refer to such a model and estimation technique as a *neighborhood sensor model*. Not only can such models accurately interpret sensor readings, but some researchers [92] have used them to calibrate sensors. Finally, timing differences amongst motes induce additional error, and can be addressed by accounting for time discrepancies in the neighborhood model or by utilizing a *coordination service*. Such a service, possibly built on top of a *time synchronization* service, would ensure neighboring motes sample their sensors at the same time.

3.2 False Events

False events, composed by spurious and missing readings, contribute to additional error in the sensing platform. Such events can be caused by broken or incorrectly calibrated hardware. In this case, problematic motes must be identified and action must be taken to ensure these motes do not increase the overall system error. One method of identifying a faulty mote is to perform a *status query*, a request for the current status, on a mote. A network-wide status query provides a list of motes that either diagnose themselves as faulty or are not responding to radio messages. Another method of determining the status of motes is to use a neighborhood sensor model for a group of motes and a known stimuli to determine via collaboration if some motes are faulty or incorrectly calibrated. Spurious readings, often caused by environmental noise or faulty hardware, can be

identified using validation or verification techniques. Verification uses a *hand shaking* protocol between the sensor network and a cooperative entity to ensure that only the requested measurements are announced by the mote network. For instance, a cooperative robot might periodically emit a low power radio message requesting nearby motes to reply with ranging measurements. Far away motes that reply to the robot can be labeled as potentially faulty. Sensor measurements from nearby motes can also be validated using a neighborhood sensor model as previously discussed.

Once groups of motes have been identified as potentially problematic, a *sensor network maintenance routine* can be applied that automatically calibrates, restarts, replaces, or turns off such motes. Automatically calibrating a mote involves using a neighborhood sensor model to predict the mote's sensor field based on measurements from nearby motes, comparing this with the mote's actual measurements, and adjusting the mote's calibration parameters accordingly. Restarting or turning off a mote would require issuing such a command to the mote over the radio. Replacing the mote would involve navigating to the faulty mote's location and deploying new hardware. Finally, for cooperative robots, hardware identified as faulty can be ignore and avoided by employing an *intelligent path planning* routine.

3.3 Network Induced Error

Properties of the networking system such as multi-hop packet delivery, broadcast collision, and shared bandwidth induce detection error when messages arrive aperiodically, with high latency, and at a low data rate. Aperiodic arrival of detection messages is caused both by the event triggered nature of distributed entity detection and by the stochastically varying latency intrinsic in the routing layer. Detections can potentially be made periodic with low jitter values by sensing coordination as previously mentioned. Latency and low data rates can be addressed by optimizing communication for multiple control modes (as previously mentioned), artificially slowing down the dynamics, or by using a *predictive controller*. A predictive controller assumes a parametric model of the dynamics, and is composed of a goal controller with a state and parameter estimator allowing it to operate in the face of missing and late measurements. Finally, tight bounds on when sensor measurements occur are often not provided (such as during PEG). This can be addressed with network-wide time synchronization, hand shaking when measurements should be performed, or with a *neighborhood routing model* that estimates the time of detections based on characteristics of the network.

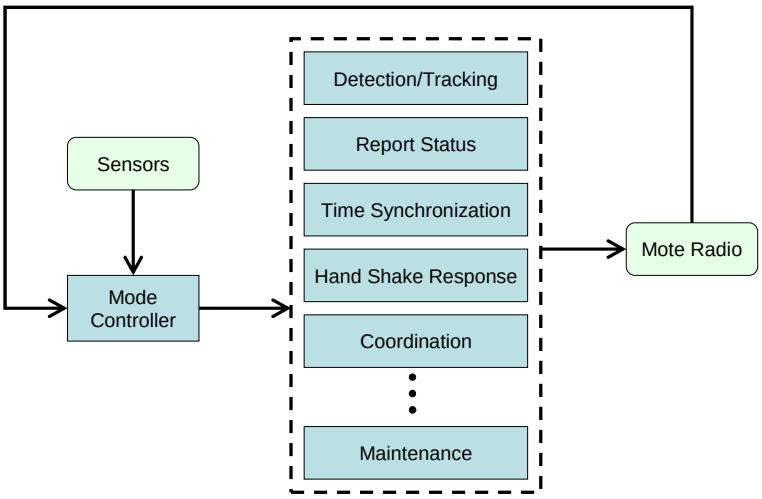


Figure 3.1: Mote information flow. Responding to messages from the radio, the mode controller selectively invokes services. Services interact with the environment by querying sensors and sending messages.

3.4 Unified Framework

This section combines the previous design solutions into a practical system architecture for SNAC systems overcoming challenging sensor network properties such as sensor error, false events, and network induced error. The techniques identified above indicate that this unified framework should implement a subset of the following services:

- Predictive controller
  - Parametric model
  - Goal controller
  - State and parameter estimation
- Neighborhood model
  - Sensing
  - Routing
- Intelligent path planning
- Multimodal controller

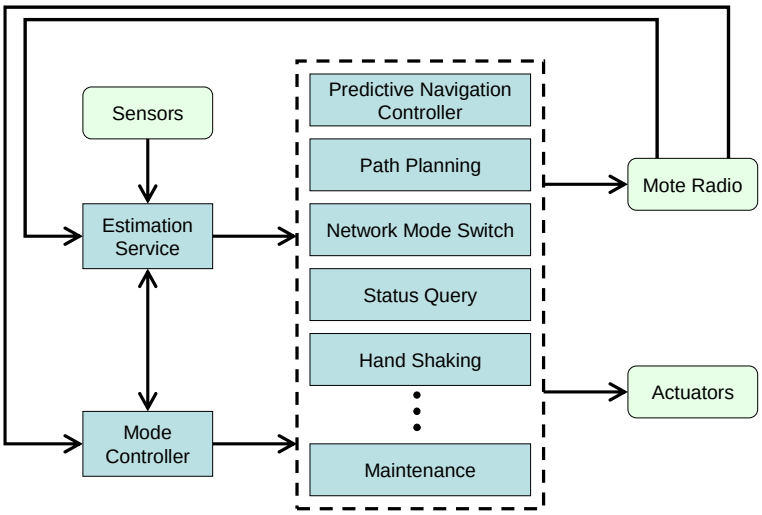


Figure 3.2: Agent information flow. Using radio messages, mode information, and (optionally) on-board sensors, the estimation service infers the system-wide state. Using additional messages and state estimation, the mode controller invokes services to achieve the agent's overall goal. Services interact with the environment by sending messages and engaging actuators.

  - Networking
  - Control strategy
- Coordination
- Status query
- Time synchronization
- Hand shaking
- Sensor network maintenance routine
  - Calibrates
  - Restart
  - Replace
  - Shut off

Using this list as a guide, we propose a 2 part architecture: the mote architecture shown in Figure 3.1 and the agent architecture shown in Figure 3.2. Both designs are similar with the

exception that the mote does not perform estimation or actuation. The 2 frameworks work together in a loose client-server relationship. The mote generally either runs a fixed set of services or selectively invokes services according to agent demand. Occasionally, the mote may invoke services autonomously; for instance, a watchdog may invoke a mote's maintenance service and cause a reboot.

The high level flow of information in the mote architecture is event-triggered. Responding to radio messages, the mode controller selectively invokes various services. Various sensing services, such as detection and tracking, send detection messages triggered by sensing events. Additionally, for a typical tracking application, the mode controller may continuously run a time synchronization and tracking service while invoking other services such as status reports or maintenance when requested by the agent.

The agent architecture can be implemented as a time-triggered architecture. Periodically, using any recently received messages and mode switching information, the agent actively estimates system-wide parameters related to sensor calibration, routing, and robot dynamics. The mode controller using transition cues from the network and a current system-wide state estimate selectively invokes various services. Invoked services influence the system's behavior by sending messages and controlling the actuators. For instance, during a PEG application, the mode controller may continuously invoke navigation and path planning services to track an evader, while selectively invoking different networking modes and maintenance routines to balance the robot's localization and tracking error.

Together these architectures enable the aforementioned design rules. The mode controller allows us to switch between various networking and control modes. The navigation controller and path planning services utilizes the predictive controller and intelligent path planning concepts. The estimation services utilizes neighborhood models. Finally, various services on both architectures take advantage of techniques such as coordination, time synchronization, status query, hand shaking, and maintenance. Some of the design rules and correlating services are explored in more detail in later chapters.

Chapter 4

The Sensor Network and Control System Simulator

Previous chapters have described real world SNAC implementations, the challenges these systems present, and methods of addressing these challenges. This chapter, develops formal models of the SNAC system and an accompanying simulator. Furthermore, this chapter performs several simulations. The first set of simulations compare components of the previously developed system architecture with a traditional design. The next set of simulations establishes the importance of path planning for robot navigation in sensor networks.

4.1 System Models

Using our experience from PEG and results from the literature [37, 11], a set of models and a simulator for a mobile robot embedded within a sensor network is developed. In order to accurately capture the most challenging aspects of sensor networks, models are developed to account for sensing noise, sensor saturation, calibration differences, packet collision, radio reception range, multi-hop latency, finite battery lifetime, faulty hardware, and loose time synchronization. Additionally, the robot model captures the actuator noise and nonholonomic dynamics. In the following sections, models for the sensor, the communication layer, the mote hardware, the detection and aggregation algorithm, and the robot dynamics are developed.

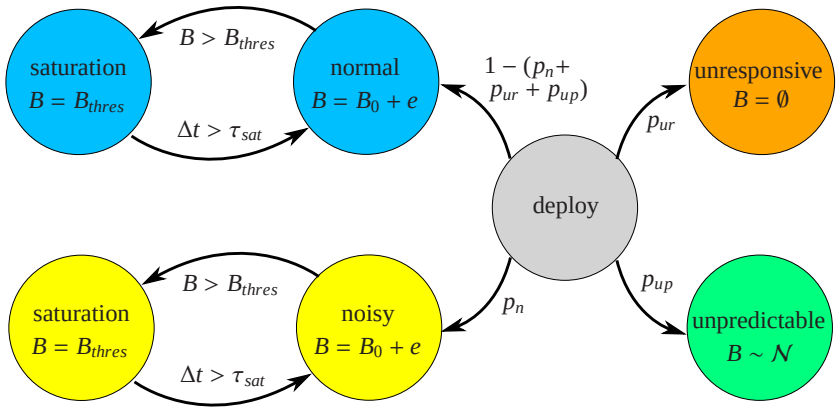


Figure 4.1: Sensor state transition diagram.

4.1.1 Sensing

For PEG, the motes used on-board magnetometers to detect entities in the network. Our experience with the mote hardware indicates that the measured value reported by a mote is a function of the state of the sensor and the magnetic field near the mote. In particular, each sensor could be in any one of 5 states: normal, noisy, unresponsive, unpredictable, or saturated. A sensor in a normal state reports a value $B = B_0 + e$ where B_0 is the magnitude of the magnetic field and e is normally distributed noise. A noisy sensor reports the same measured value, but typically has much higher noise variance due to a priori exposure to intensive magnetic fields or faulty hardware. An unresponsive sensor never reports measurements due to, for instance, a weak battery or a faulty radio. Sensors in a unpredictable state report a normally distributed value. This state models faulty sensors that tend to report only noise typically due to exposure to intensive magnetic fields. Finally, a sensor in the saturated state reports a constant measured value $B = B_{sat}$. Sensors enter saturation when exposed to large magnetic fields and typically resume normal operation after a short period of time.

During an experiment, a sensor can occasionally transition between states as shown in Figure 4.1. Before an experiment begins, all sensors are in the initial state of deploy. As each mote is placed in the field, its sensor transitions to the unpredictable, unresponsive, noisy, or normal state with probability p_{up}, p_{ur}, p_n, or $1 - (p_{up} + p_{ur} + p_n)$, respectively. Once a sensor becomes unresponsive or unpredictable, it remains this way (until maintained or re-deployed). Sensors that are deployed in the normal or noisy states occasionally transition to the saturation state during normal operation if the detected field B exceeds the saturation threshold B_{thres}. Once the sensor has

been in saturation for a duration of τ_{sat} seconds, it returns to its previous state. Furthermore, if a mote is deployed in the normal (noisy) state, its error bias μ_k is normally distributed as $\mathcal{N}(0, \sigma_\mu^2)$ ($\mathcal{N}(0, \tilde{\sigma}_\mu^2)$) and its error variance σ_k^2 is normally distributed as $\mathcal{N}(0, \sigma_\sigma^2)$ ($\mathcal{N}(0, \sigma_\sigma^2)$).

The actual field the sensor is exposed to (denoted by B_0) is modeled as a magnetic dipole far field. In particular, each detectable entity is assumed to be a magnetic dipole with dipole moment $\vec{m} = \begin{bmatrix} 0 & 0 & M \end{bmatrix}^T$ at a height of d_m (in meters) above the field, where $M > 0$ (in Am^2) and d_m is much larger than the length of the dipole. Letting $\vec{r}$ be the vector from the dipole to a particular mote, the magnitude of the magnetic field (in Teslas) at the mote is

$$B_0 = \frac{\mu_0}{4\pi ||\vec{r}||^5} \left\| 3(\vec{m} \cdot \vec{r})\vec{r} - ||\vec{r}||^2 \vec{m} \right\| \tag{4.1}$$

where μ_0 is the permeability of free space (in Wb/(Am)). Therefore, with an entity at $\begin{bmatrix} x_v & y_v \end{bmatrix}^T$ in the plane, and a mote on the ground at $\begin{bmatrix} x_m & y_m \end{bmatrix}^T$, it can be shown that

$$B_0 = \frac{3\mu_0 d_m M \left[\rho^2 + \left(\frac{2d_m^2 - \rho^2}{3d_m} \right)^2 \right]^{1/2}}{4\pi(\rho^2 + d_m^2)^{5/2}} \tag{4.2}$$

where $\rho = \left\| \begin{bmatrix} x_m - x_v & y_m - y_v \end{bmatrix}^T \right\|$ is the distance (in meters) in the plane between the mote and the entity. Note, the largest field, attained when the entity is directly on top of the mote, is

$$B_0(\rho = 0) = \frac{\mu_0 M}{2\pi d_m^3} \tag{4.3}$$

4.1.2 Communication

The communication model abstracts the combined interaction of the lossy radio channel with the multi-hop routing protocols. In particular, 2 communication scenarios are considered: message broadcasting (single-hop communication) and multi-hop routing. The broadcast communication model accounts for medium access control (MAC) delays, packet collision, and non-deterministic reception ranges. In particular, when a mote sends a broadcast message, the packet is first tested for collision. With probability p_{drop}, the message collides and is lost. If not lost, the message is assigned a communication delay of $\tau_{lag} \sim \mathcal{U}[\tau'_{lag}, \tau''_{lag}]$. After τ_{lag} seconds have elapsed, the message is delivered to each neighbor at distance d with probability $p_{dist}(d)$. The reception probability is modeled after the work by Ganesan et al [37] and Cerpa et al [19]. In particular, this work determined that the reception probability is not 1 even at a distance 0 and does not drop to 0

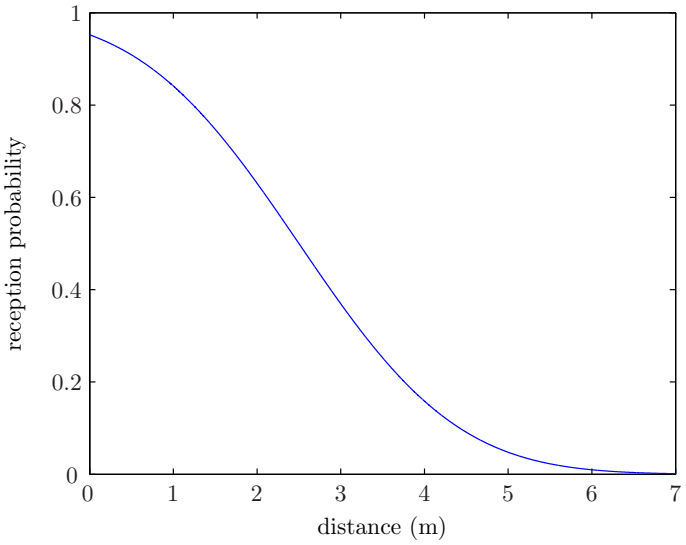


Figure 4.2: Single-hop reception probability $p_{dist}(d) = \mathcal{R}(d, \mu, \sigma)$ for $\mu = 2.5$ and $\sigma = 1.5$.

even for rather large distances. This reception probability $p_{dist}(d)$ is approximated here with

$$p_{dist}(d) = \mathcal{R}(d, \mu, \sigma) = 0.5\left(1 - erf\left[\frac{d-\mu}{\sqrt{2}\sigma}\right]\right) \tag{4.4}$$

for some fixed values μ, σ. An example of this function for $\mu = 2.5$ and $\sigma = 1.5$ is shown in Figure 4.2.

The multi-hop routing model accounts for packet loss and lag induced by each hop. This model assumes the underlying routing algorithm ensures a robust network topology that provides a route between any sender and receiver pair. Hence, each single-hop link along a multi-hop route is assumed to be reliable and the reception probability model $p_{dist}(d)$ is ignored. However, since each transmission may still be loss due to collision, each hop may fail with probability p_{drop}. Furthermore, it is assumed that the number of hops between any sender and receiver pair can be approximated by $\hat{n}_{hop} = \lceil \frac{d_r}{\hat{d}_{hop}} \rceil$ where d_r is the straight line distance between the sender and receiver and $\hat{d}_{hop}$ is the estimated average distance per hop. Using this model, each multi-hop message is first duplicated for each potential receiver. Then, for each message, the straight line distance to the receiver d_r and the estimated hop count $\hat{n}_{hop}$ is computed. Then, each hop $i \in \{1, 2, \ldots, \hat{n}_{hop}\}$ either fails with probability p_{drop} and the message is lost, or the hop succeeds and adds an additional lag of $\tau^i_{lag} \sim \mathcal{U}[\tau'_{lag}, \tau''_{lag}]$ to the overall packet latency $\tau_{lag} = \sum_i \tau^i_{lag}$. If all hops succeed, the message is delivered τ_{lag} seconds later to its receiver.

4.1.3 Mote Platform

The mote platform model accounts for the variation of times motes come online and each mote's finite battery lifetime. In particular, motes may be manually switched on one at a time or the network may be sent a start-up radio message. In either case, individual motes will become active at different times. This is modeled by allowing the initial sensing time τ_s of a mote be uniformly distributed: $\tau_s \sim \mathcal{U}[\tau_s', \tau_s'']$. Furthermore, once a mote is turned on, it's battery (or potentially other vital hardware) will eventually fail. This is modeled by setting a uniformly distributed expiration time τ_e for each mote: $\tau_e \sim \mathcal{U}[\tau_e', \tau_e'']$.

4.1.4 Detection Routine

The detection routine models the entity detection and aggregate algorithm that is programmed on the motes. This models the same algorithm implemented by PEG. In particular, each mote with a measured magnetic field B larger than a given threshold B_{thres} announces the measured value over the radio to its neighboring motes using a detection message. A detection message contains the announcing mote's id, location, and the measured magnetic field. Once a mote announces a detection, it must wait for at least T_{report} seconds before reporting a detection again. Additionally, any mote receiving a detection message will store the message for T_{report} seconds before discarding it. If a detecting mote measures a magnetic field larger than the magnetic field reported by any of its neighbors within the last T_{report} seconds, this mote will elect itself as a leader. As a leader, it aggregates all the detection reports from its neighbors (within the last T_{report} seconds) into an event packet and sends this packet (via the multi-hop routing layer) to the pursuer robot.

4.1.5 Robot

The robot model accounts for the kinematics of the pursuer and evader robots. For PEG, the pioneer robot from ActivMedia was used. However, for many other test beds, the COTS-BOTS [11] platform, a mote controlled small RC car, is used. The robot model approximates the nonholonomic, car-like kinematics of the COTS-BOTS that can virtually instantaneously set its steering angle and wheel velocity. In particular, the position and orientation $\begin{bmatrix} x_k^1 & x_k^2 & \theta_k \end{bmatrix}^T$ evolve

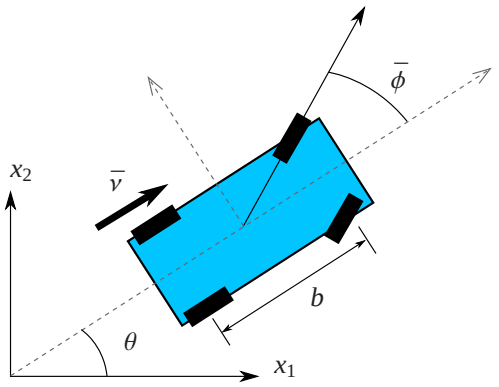


Figure 4.3: The robot model's state and parameters.

according to

$$\begin{bmatrix} x^1_{k+1} \\ x^2_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x^1_k \\ x^2_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} b\cot(\bar{\phi}_k)[\sin(\theta_k + \eta_k) - \sin(\theta_k)] \\ b\cot(\bar{\phi}_k)[\cos(\theta_k) - \cos(\theta_k + \eta_k)] \\ \eta_k \end{bmatrix} \tag{4.5}$$

$$\eta_k = \frac{\bar{v}_k T \sin(\bar{\phi}_k)}{b} \tag{4.6}$$

$$\tag{4.7}$$

where b is the wheelbase, η_k is the change in orientation, T is the sampling period, $\bar{v}_k$ is the wheel velocity, and $\bar{\phi}_k$ is the steering angle as shown in Figure 4.3. Additionally, the achievable values for wheel velocity and steering angle are limited by the hardware:

$$\bar{v}_k \in [v_{min}, v_{max}] \tag{4.8}$$

$$\bar{\phi}_k \in [\phi_{min}, \phi_{max}] \tag{4.9}$$

$$\tag{4.10}$$

where $v_{min} < 0 < v_{max}$ and $\phi_{min} < 0 < \phi_{max}$. The applied wheel speed and steering angle are noisy functions of the user supplied control values v_k and ϕ_k. In particular,

$$\bar{v}_k = v_k + \epsilon^v_k \tag{4.11}$$

$$\bar{\phi}_k = \phi_k + \epsilon^\phi_k \tag{4.12}$$

$$\epsilon^v_k \sim \mathcal{N}(0, \sigma^2_v) \tag{4.13}$$

$$\epsilon^\phi_k \sim \mathcal{N}(0, \sigma^2_\phi) \tag{4.14}$$

Finally, the control bounds $\{v_{min}, v_{max}, \phi_{min}, \phi_{max}\}$ are considered fixed, unknown parameters of the system.

4.2 Agent Design

This section develops several of the system architecture components from Section 3.4 including neighborhood estimation services, a predictive controller, and a path planner. First, an extended Kalman filter is developed for state estimation. Then, two modifications to the estimation system are proposed: network latency compensation and faulty node filtering. Next, a predictive navigation controller is designed. Finally, a feedback controller for navigation is developed to compare with the predictive controller.

4.2.1 State Estimation

In this section, an extended Kalman filter (EKF) [6] is designed to provide state estimates to the controller. In following sections, the estimator is augmented to compensate for false positives and communication delay. An EKF is designed analogously to a Kalman filter by linearizing nonlinearities in the dynamics at each point of estimation. For each new time step $k+1$ and new sensor measurement y_{k+1}, the EKF iteratively computes the new state estimate $\hat{x}_{k+1|k+1}$ and the error covariance $P_{k+1|k+1}$. The update occurs iteratively in 2 steps: a proprioceptive update and a preceptive update. The proprioceptive update evolves the state forward to realize $\hat{x}_{k+1|k}$ according to the system dynamics with no noise, and evolves the error covariance forward according to

$$P_{k+1|k} = F_k P_{k|k} F_k^T + Q_k \tag{4.15}$$

where Q_k is the noise covariance of the dynamics and F_k is the Jacobian of the dynamics. For the car dynamics, it can be shown that

$$F_k = \begin{bmatrix} 1 & 0 & -v_k T \sin(\theta_k) \\ 0 & 1 & v_k T \cos(\theta_k) \\ 0 & 0 & 1 \end{bmatrix} \tag{4.16}$$

for a car moving in a straight line, or

$$F_k = \begin{bmatrix} 1 & 0 & b\cot(\phi_k)[\cos(\hat{\theta}_{k|k} + \eta_k) - \cos(\hat{\theta}_{k|k})] \\ 0 & 1 & b\cot(\phi_k)[\sin(\hat{\theta}_{k|k} + \eta_k) - \sin(\hat{\theta}_{k|k})] \\ 0 & 0 & 1 \end{bmatrix} \tag{4.17}$$

for a car that is turning where $\eta_k = \frac{v_k T \sin(\phi_k)}{b}$.

The preceptive update is computed, using the proprioceptive update, as

$$K_{k+1} = P_{k+1|k} H_{k+1}^T (H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1})^{-1} \tag{4.18}$$

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1}(y_{k+1} - H_{k+1}\hat{x}_{k+1|k}) \tag{4.19}$$

$$P_{k+1|k+1} = (I - K_{k+1} H_{k+1}) P_{k+1|k} \tag{4.20}$$

where R_{k+1} is the covariance of the sensor noise and H_{k+1} is the Jacobian of the measurement function. Since event messages can be transformed into a single position estimate (using, for instance, a center of mass computation), the measurement function is taken to be

$$y_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \\ \theta_k \end{bmatrix} + \epsilon \tag{4.21}$$

where ϵ is the sensor noise which is normally distributed with covariance R_k. Hence, for this model, H_{k+1} is given by the constant output matrix:

$$H_{k+1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.22}$$

4.2.2 Network Latency Compensation

Using the above computations, the state can be estimated for each time step using knowledge of the control values and the new sensor measurements. However, the sensor network does not provide a sensor measurement at each time step. Furthermore, received measurements can be out of order or late. Hence, the input to the EKF is adapted to account for these characteristics. If a new sensor measurement is not available, only the proprioceptive update is applied. If a new sensor measurement is available it is used to correct the state estimate τ_d seconds in the past where τ_d is the estimated mean latency of received messages. After a past state estimate is corrected, the filter is re-applied to generate a current estimate.

4.2.3 Faulty Node Filtering

To account for faulty motes that frequently report false positives, a message validation filter is applied to all incoming messages. This filter estimates a packet's validity probabilistically as $p(v|n)$ where n is the number of sensor readings in the packet and $v \in \{\text{valid}, \text{invalid}\}$ is the

packet's validity. Applying maximum likelihood estimation, the validity of a packet with $\tilde{n}$ sensor readings is the argmax over v of $p(\tilde{n}|v)$ assuming a uniform prior distribution for v. Clearly, this model either accepts or rejects packets based on the number of sensor readings.

4.2.4 Predictive Controller

A model predictive controller [82] (MPC) is developed for way-point navigation of the car-like robot. This controller achieves the destination point x_f using two maneuvers: turning and driving straight. First, during the turning phase, the vehicle is driven in a tight circle taking the vehicle from a starting state $\begin{bmatrix} x_a & \theta_a \end{bmatrix}^T$ to a intermediate state $\begin{bmatrix} x_b & \theta_b \end{bmatrix}^T$. Second, during the straight phase, the vehicle is driven straight ($\phi = 0$) to $\begin{bmatrix} x_f & * \end{bmatrix}^T$ where the orientation is not specified.

The computation is carried out by first determining the quadrant q^1 (relative to the robot coordinate system) the destination point lies in. Next, it is determined whether the destination point lies inside the turning radius of the robot at its initial position (r^1 = true) or not (r^1 = false). Using these two computations, the controller chooses an initial control value:

- $(v, \phi) = (v_{max}, \phi_{max})$ if $(q^1, r^1) \in \{(2, \text{false}), (4, \text{true})\}$
- $(v, \phi) = (v_{min}, \phi_{max})$ if $(q^1, r^1) \in \{(3, \text{false}), (1, \text{true})\}$
- $(v, \phi) = (v_{max}, \phi_{min})$ if $(q^1, r^1) \in \{(1, \text{false}), (3, \text{true})\}$
- $(v, \phi) = (v_{min}, \phi_{min})$ if $(q^1, r^1) \in \{(4, \text{false}), (2, \text{true})\}$

This control action is applied exactly long enough for the robot to reach an intermediate state that admits a straight path to the destination. Next, a control action that drives the robot straight to the destination is computed by determining which quadrant q^2 the destination currently lies in:

- $(v, \phi) = (v_{max}, 0)$ if $q^2 \in \{1, 2\}$
- $(v, \phi) = (v_{min}, 0)$ if $q^2 \in \{3, 4\}$

The second control action is applied exactly long enough for the robot to reach the destination.

4.2.5 Feedback Controller

A simple state feedback controller is developed to challenge the performance of the predictive controller. To simplify future analysis, the feedback controller is kept quite simple: way-points are assumed to be far away (in comparison to the wheelbase) and way-points are considered

to be successfully achieved when the car is within $d_{goal} > b$. Given these constraints, the feedback controller is designed to proportionally track the desired orientation (the robot orientated towards the goal) and reduce the wheel speed as the destination becomes close. In particular, the control values are

$$\begin{bmatrix} \nu_k \\ \phi_k \end{bmatrix} = \begin{bmatrix} \hat{v}_{max}[1 - e^{-\|\hat{x}_k - x_f\|}] \\ p_\phi(\hat{\theta}_k - \theta_k^d) \end{bmatrix} \tag{4.23}$$

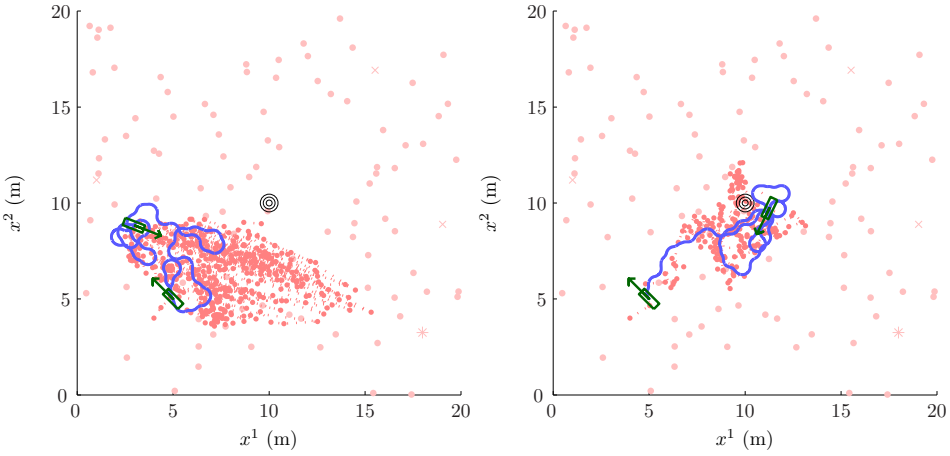
where $\theta_k^d = \arctan(\frac{x_f^2 - \hat{x}_k^2}{x_f^1 - \hat{x}_k^1})$ is the desired orientation, $\begin{bmatrix} \hat{x}_k & \hat{\theta}_k \end{bmatrix}^T$ is the state estimate at time k, $\hat{v}_{max}$ is a fixed a priori estimate of the maximum speed of the vehicle, and $p_\phi \in \mathbf{R}^+$ is a control parameter.

4.3 Controller Comparison Simulations

Using the aforementioned models and control systems, several simulations are performed to compare the performance of the control architecture from Section 3.4 to that of a simple feedback controller. In particular, the estimation accuracy and navigation ability of five different system architectures is compared. The simulator is configured to emulate the PEG sensor network. In particular, the parameter values used are given in Appendix A unless otherwise noted.

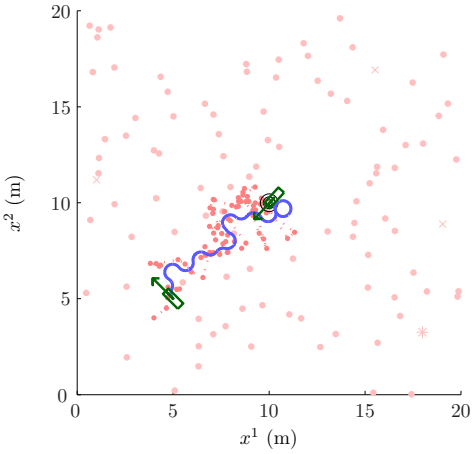
Five system architectures are compared. System 1 uses the feedback controller and the plain EKF previous designed. The plain EKF estimator does not use network latency compensation or faulty node filtering. System 2 augments the estimator of System 1 with faulty node filtering. System 3 augments the estimator of System 2 with network latency compensation. System 4 uses the model predicative controller, the EKF, and faulty node filtering. Finally, system 5 augments the estimation of system 4 with network latency compensation.

For each simulation, 10 trials were ran, the destination goal was set to $\begin{bmatrix} 10 & 10 \end{bmatrix}^T$, and the goal tolerance was set to 10 cm. Each trial was ran until the goal was achieved or 150 seconds elapsed. An example trial for each system is shown in Figure 4.4. For each figure, the gray symbols represent the locations of sensor nodes with each symbol denoting the node's status: normal (gray dot), non-responsive (gray x), noisy (gray asterisk), and unpredictable (gray pentagram). Furthermore, the robot's starting and ending states are denoted by a green rectangle and arrow. The desired ending location is drawn as 3 concentric black circles. Finally, the estimated robot position during the trial is denoted by light red dots connected by a light red dotted line. Each smaller figure is captioned with the system architecture components used: Feedback (FB), extended Kalman filter (EKF), model predictive controller (MPC), faulty node filtering (FNF), and network latency compensation (NLC).

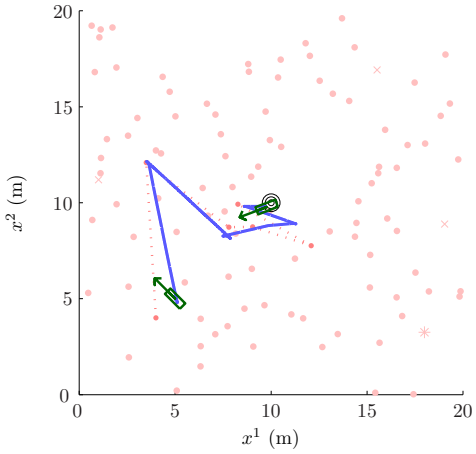


(a) System 1: FB, EKF

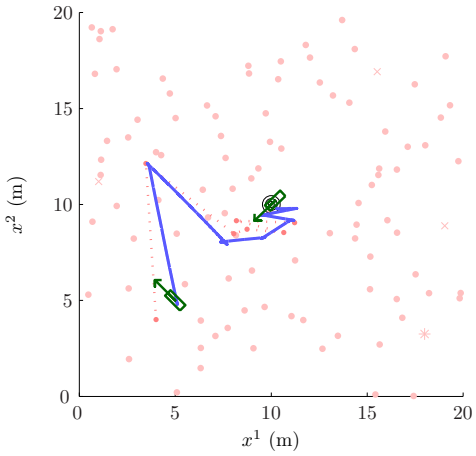
(b) System 2: FB, EKF, FNF



(c) System 3: FB, EKF, FNF, NLC



(d) System 4: MPC, EKF, FNF



(e) System 5: MPC, EKF, FNF, NLC

Figure 4.4: Example trial for 5 different control architectures depicting path and estimation results.

System	n_{goal}	$\bar{t}_{goal}$	$\bar{n}_{est}$	$\bar{e}_x$	$\bar{e}_\theta$
1	0	NA	5001.0	2.98	1.95
2	3	81.1	4311.5	1.21	1.51
3	1	66.1	4721.2	1.82	2.01
4	2	108.5	8.6	1.15	0.56
5	4	111.7	8.4	1.14	0.54

Table 4.1: Simulation results comparing 5 different system architectures for 10 trials. The first 3 system utilize a feedback architecture and the last 2 utilize a MPC based architecture. Shown above is the number of goal achieving trials n_{goal}, the mean destination travel time $\bar{t}_{goal}$, the mean number of estimations required $\bar{n}_{est}$, the mean estimated position error $\bar{e}_x$, and mean estimated orientation error $\bar{e}_\theta$.

Referring to this figure, it is shown that for system 1, the estimated path is smeared towards the unpredictable mote in the lower right of the field. This is expected since the estimator does not account for such faulty nodes. Consequently, the estimation is poor and the feedback controller drives the robot aimlessly in circles. The next system, utilizes the faulty hardware model and does much better. The estimator is able to ignore the faulty node and is able to reasonably estimate the position of the robot guiding it close to the desired destination. System 3 is an attempt to compensate for the lag of the network. For this trial, the system promptly achieves the desired goal position. However, as will be addressed later, this turns out to not be true in general. The estimation tends to lags sufficiently behind the the robot's actual position and causes instability in the feedback system. The next system switches to the model predictive controller with the EKF and faulty node filtering. This system achieves the goal with low estimation error. In particular, since this system only estimates the robot's state after an entire control sequence has been carried out, the controller can pause and allow laggy sensor messages to reach it before each estimation. Finally, system 5, augments the previous system with network latency compensation further reducing the estimation error. This system does not suffer with the application of network latency compensation as system 3 does since the MPC does not immediately react to each incoming message and can pause at the completion of each control sequence.

The simulation results are summarized in Table 4.1. Notice, the basic feedback system is incapable of ever achieving the goal. System 2 achieves a significant improvement in performance by using faulty node filtering. However, as previously mentioned, network latency compensation tends to induce instability for system 3 and reduces the overall performance of the controller. The

Path	t_d	$\bar{n}_{est}$	$\bar{e}_x$	$\bar{e}_\theta$	$\bar{n}_{det}$	$\bar{n}_{evt}$	$\bar{e}_{evt,pos}$
1	60.5	1	3.90	0.47	1328.0	229.1	3.56
2	62.9	2	2.77	1.73	1186.2	230.2	3.82
3	65.1	2	2.10	0.53	1604.8	266.3	3.37
4	109.6	8	1.71	0.78	2199.0	375.8	4.40

Table 4.2: Simulation results comparing 4 different paths across 20 trials. Shown above is the travel time t_d, the mean number of estimations required $\bar{n}_{est}$, the mean estimated position error $\bar{e}_x$, the mean estimated orientation error $\bar{e}_\theta$, the mean number of detections $\bar{n}_{det}$, the mean number of event messages sent $\bar{n}_{evt}$, and the mean estimated position error without filtering $\bar{e}_{evt,pos}$.

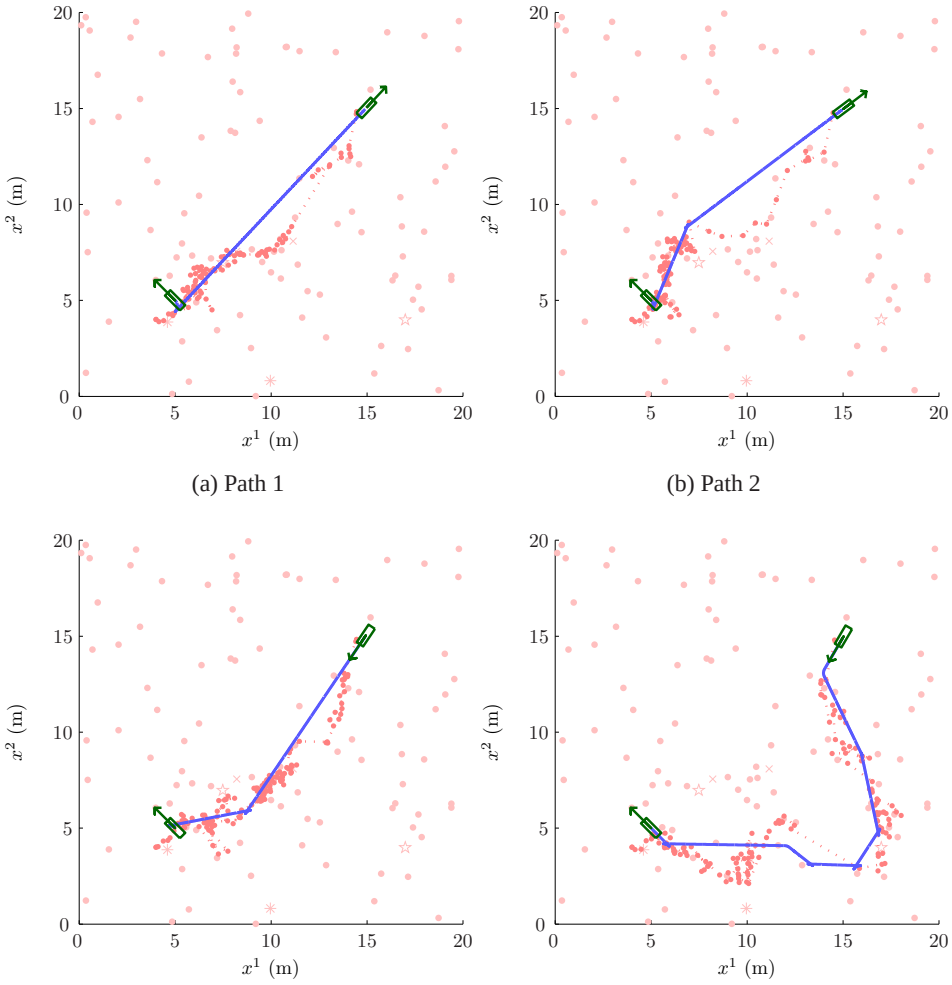
last 2 systems improve performance by replacing the feedback controller with the MPC. In particular, system 5 achieves the best results. This architecture achieves the goal 40% of the time, requires very little estimation overhead, and reduces the position and estimation error by 61.7% and 72.3%, respectively, compared to the basic feedback system.

4.4 Path Comparison Simulations

Using the aforementioned models and system architecture, several simulations are performed to compare the effect path planning has on robot localization accuracy. In particular, using the final system architecture (system 5) from the previous section, four different routes to the destination are followed. The results are compared to determine if path planning (exploiting the sensor network topology) significantly effects localization accuracy. The simulator is configured to emulate the PEG sensor network. In particular, the parameter values used are given in Appendix A unless otherwise noted.

For these simulations, 20 trials are performed and the destination goal was set to $\begin{bmatrix} 15 & 15 \end{bmatrix}^T$. Example trials using the four chosen routes are shown in Figure 4.5. Path 1 simply goes directly to the destination ignoring the sensor network topology. Path 2 is allowed to slightly deviate from straight to encounter a region more densely covered by nodes. Path 3 is allowed the same flexibility, but takes a different path. Finally, path 4 attempts to enter as many densely covered areas as possible on the way to the destination.

The results are summarized in Table 4.2. As expected, path 1 is the quickest path to the destination and the control architecture only estimates the robot state once the destination is reached. Since nodes are scattered to either sides of this route, the estimated path has high error, but



(a) Path 1

(b) Path 2

(c) Path 3

(d) Path 4

Figure 4.5: Example trial for 4 different fixed routes depicting path and estimation results.

the estimated orientation has low error. The second path does well initially (see Figure 4.5b), but the latter sections of this path are detected by nodes to the lower right of the actual position. This results in a reduction in estimated position error, but an increase in estimated orientation error. The third path mends the trouble with the latter part of the second path by routing down and to the right (see Figure 4.5c). This has the effect of slightly increasing the number of detections and achieves good estimation results. Path 4 attempts to only navigate in dense regions and seeks reduced error. This route does reduce the estimated position error at the sake of increased estimated orientation error and increased number of detections (utilization of bandwidth). In fact, this path, although chosen to reduce the innate sensor error $\bar{e}_{evt,pos}$, has actually increased it by 23.6% over the basic straight path. This can also been seen by Figure 4.5d. The first half of the route has high detection error with motes detecting the robot either too high or too low. However, even with poor detections, this path achieves the lowest position estimation error. From this, we see that it is unclear how the robot should choose its path. This will be addressed in more detail in later chapters.

The simulations have confirmed that both control architecture and path planning have significant effect on the system performance. Components of the unified system architecture have proved to be beneficial for SNAC system operating in realistic simulations. However, it is unclear how path planning should be addressed to improve system performance. In the following chapters, path planning to achieve reduced localization error and bandwidth utilization is investigated.

Chapter 5

Information Maps

The remaining material in this thesis focuses on robot localization in sensor networks exploiting an information metric. This chapter presents information maps, a tool for determining the localizability of a robot in a region. Previous work [78, 63, 86] exploiting information for robot localization has been in the context of a robot equipped with on-board sensors. Our work applies a global information view, or information map, to robots localizing with sensor network data. In the following chapters, we develop sensor network models and approximations of these models suitable for information map computation. For instance, this chapter considers many binary sensors distributed in a field; later chapters extend this sensor model and develop approximations suitable for efficient computation. Additionally, our work provides a comprehensive look at information maps (formal presentation of the algorithm with time complexity analysis), and realistic sensor network localization simulations utilizing information maps. Finally, a later chapter will develop a novel path planning routine exploiting information that outperforms several other (non-information based) techniques for accurately localizing a robot in a sensor network.

This chapter provides an overview of information maps. First, an analytical overview of Markov localization is presented. Next, the information metric is introduced. Then, using this metric, an algorithm is presented that computes information for all regions of the robot pose space generating an information map. Finally, several simulations are performed revealing the computation time required and providing insight into the information topology of a sensor network.

5.1 Markov Localization

This section describes how Markov localization [36] is used to estimate the pose, or state, of a robot. First, the system model, composed of the robot's dynamics and the sensor network model is described. Next, this model is adapted for use with Markov localization. Finally, the steps of localizing a robot using Markov localization are outlined.

Markov localization is a Bayesian estimation technique [49] for estimating a robot's pose given a model of the system and periodic sensor readings. Markov localization requires that the pose belongs to a finite space and that the system is specified probabilistically using a Hidden Markov Model (HMM). However, a robot's pose often lives in an infinite, continuous space and evolves according to a set of difference or differential equations. In order to rectify this difference, the infinite pose space is partitioned and the dynamics are converted to a HMM.

First, the infinite pose space is partitioned. In the following, a bar is placed over variables related to the infinite pose space, such as $\bar{l}$ or $\bar{\mathcal{L}}$, to distinguish them from the finite pose space, such as l or $\mathcal{L}$. The infinite pose space $\bar{\mathcal{L}}$ is partitioned into a finite number of pose spaces $\mathcal{L} = \{\bar{\mathcal{L}}_i\}$. Typically, this partitioning is based on a regular grid, although other techniques such as topological [68] and tree-based [16] are used. For our work, the continuous pose space is a A by A square anchored at the origin in $\mathbf{R}^2$. This space is partitioned into a regular M by M grid where the $(i, j)^{th}$ partition, or cell, is denoted as $\mathcal{G}_{i,j}$. More precisely, the pose spaces are described by

$$\bar{\mathcal{L}} = [0, A] \times [0, A] \subset \mathbf{R}^2 \tag{5.1}$$

$$\mathcal{L} = \{\mathcal{G}_{i,j}\}_{i,j} \tag{5.2}$$

$$\mathcal{G}_{i,j} = [(i-1)\Delta, i\Delta] \times [(j-1)\Delta, j\Delta] \subset \bar{\mathcal{L}} \subset \mathbf{R}^2 \tag{5.3}$$

$$\Delta = \frac{A}{M} \tag{5.4}$$

where $i, j \in \{1, 2, \ldots, M\}$. For notational convenience, $\langle i, j \rangle$ is used to denote a pose l in the partition space that represents the $(i, j)^{th}$ grid cell $\mathcal{G}_{i,j}$ with center $\bar{g_{i,j}}$, inducing the following relationship:

$$l = \langle i, j \rangle = \mathcal{G}_{i,j} \iff \bar{l} \in \mathcal{G}_{i,j} \iff \bar{g_{i,j}} = \text{Center}(\mathcal{G}_{i,j}) \tag{5.5}$$

Next, the system dynamics are adapted for Markov localization. The robot's pose $\bar{l} \in \bar{\mathcal{L}}$ is assumed to evolve according to the difference equation

$$\bar{l}_{k+1} = f(\bar{l}_k, u_k, k) \tag{5.6}$$

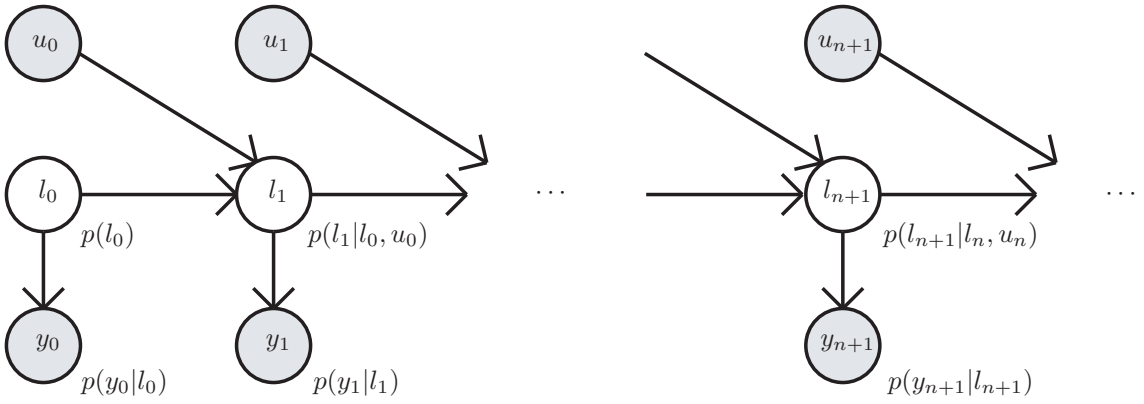


Figure 5.1: A hidden Markov model.

for each time step $k \in \{0, 1, 2, \ldots\}$, and each control value $u_k \in \mathbf{U}$. At each time step k, our sensor network measures the current pose $\bar{l}_k$ as

$$y_k = h(\bar{l}_k, k) \tag{5.7}$$

where $y_k \in \mathcal{Y}$.

Adapting these dynamics to a HMM can be challenging. In a later section, the necessary steps are performed; for the time being, it is assumed that the system is already adapted to a HMM. The HMM (see Figure 5.1) is modeled with a hidden state $l_k \in \mathcal{L}$ representing the pose, an output y_k representing the sensor network measurements, and a transition input u_k representing the control. The relationships between the variables are specified probabilistically: the evolution from one state l_k to the next l_{k+1} is described by $p(l_{k+1}|l_k, u_k)$, the sensing relationship is described by $p(y_k|l_k)$, and the initial pose at time $k = 0$ is described by $p(l_0)$. For notational simplicity, $p(z)$ and $p(z_0)$ are written to indicate $p(Z = z)$ and $p(Z = z_0)$, respectively.

Now, the Markov localization algorithm is investigated. This algorithm provides an iterative method for estimating the pose of the robot given periodic sensor readings. The algorithm is developed by computing the posterior pose estimation (distribution) given the prior pose estimation (distribution), a sensor reading, and a control input. The iterative computation is revealed by a thoughtfully chosen distribution α:

$$\alpha(l_n) \;\coloneqq\; p(l_n, y_0, \ldots, y_n | u_0, \ldots, u_{n-1}) \tag{5.8}$$

$$\alpha(l_0) \;\coloneqq\; p(l_0) \tag{5.9}$$

This definition of α allows an iterative computation of the posterior pose estimation at time $n + 1$

given the relevant quantities at time n:

$$\begin{aligned}
\alpha(l_{n+1}) &= p(l_{n+1}, y_0, \ldots, y_{n+1} | u_0, \ldots, u_n) && (5.10)\\
&= \sum_{l_n} p(l_{n+1}, y_0, \ldots, y_{n+1} | l_n, u_0, \ldots, u_n) p(l_n | u_0, \ldots, u_n) && (5.11)\\
&= \sum_{l_n} p(y_0, \ldots, y_{n+1} | l_{n+1}, l_n, u_0, \ldots, u_n) p(l_{n+1} | l_n, u_0, \ldots, u_n) p(l_n | u_0, \ldots, u_n) && (5.12)\\
&= \sum_{l_n} p(y_0, \ldots, y_n | l_{n+1}, l_n, u_0, \ldots, u_n) p(y_{n+1} | l_{n+1}, l_n, u_0, \ldots, u_n) \cdot \\
&\qquad p(l_{n+1} | l_n, u_0, \ldots, u_n) p(l_n | u_0, \ldots, u_n) && (5.13)\\
&= \sum_{l_n} p(y_0, \ldots, y_n | l_n, u_0, \ldots, u_{n-1}) p(y_{n+1} | l_{n+1}) p(l_{n+1} | l_n, u_n) p(l_n | u_0, \ldots, u_{n-1}) && (5.14)\\
&= \sum_{l_n} p(l_n, y_0, \ldots, y_n | u_0, \ldots, u_{n-1}) p(y_{n+1} | l_{n+1}) p(l_{n+1} | l_n, u_n) && (5.15)\\
&= p(y_{n+1} | l_{n+1}) \sum_{l_n} p(l_{n+1} | l_n, u_n) \alpha(l_n) && (5.16)
\end{aligned}$$

Referring to equation (5.16), we see that $\alpha(l_{n+1})$ is computed by first evolving $\alpha(l_n)$ forward using the system dynamics $p(l_{n+1}|l_n, u_n)$ (the proprioceptive step), and then by weighting the result with new sensor readings $p(y_{n+1}|l_{n+1})$ (the preceptive step). Once $\alpha(l_{n+1})$ is computed, the pose estimation distribution is computed as

$$p(l_{n+1} | y_0, \ldots, y_{n+1}, u_0, \ldots, u_n) = \frac{\alpha(l_{n+1})}{\sum_{l_{n+1}} \alpha(l_{n+1})} \tag{5.17}$$

To reduce this distribution to a single point estimate, several methods may be used such as finding the argmax or taking the expectation over l_{n+1}. We prefer the later and estimate the pose as

$$\hat{l}_{n+1} = \mathbf{E}(p(l_{n+1} | y_0, \ldots, y_{n+1}, u_0, \ldots, u_n)) = \mathbf{E}(\frac{\alpha(l_{n+1})}{\sum_{l_{n+1}} \alpha(l_{n+1})}) \tag{5.18}$$

Although conceptually straightforward, the above algorithm neglects to normalize α at each step, leading to large numerical errors over time. This is solved by replacing α with the normalized version β:

$$\begin{aligned}
\beta_m^r(l_n) &\coloneqq p(l_n | y_0, \ldots, y_r, u_0, \ldots, u_m) && (5.19)\\
\beta_0^0(l_0) &\coloneqq p(l_0) && (5.20)
\end{aligned}$$

Following steps similar to those used above, we find a 2 step iteration over β:

$$\begin{aligned}
\beta_n^n(l_{n+1}) &= \sum_{l_n} p(l_{n+1} | l_n, u_n) \beta_{n-1}^n(l_n) && (5.21)\\
\beta_n^{n+1}(l_{n+1}) &= \frac{p(y_{n+1} | l_{n+1}) \beta_n^n(l_{n+1})}{\sum_{l_{n+1}} p(y_{n+1} | l_{n+1}) \beta_n^n(l_{n+1})} && (5.22)
\end{aligned}$$

Here the proprioceptive and preceptive steps naturally reveal themselves in equation (5.21) and equation (5.22), respectively. To initialize this algorithm, at time $n = 0$, we start with $p(l_0)$ and a sensor reading y_0. Since the robot is assumed to be stationary prior to $n = 0$, $u_{-1} = \mathbf{0}$ could be used in the computation, but instead, we prefer to only compute the preceptive update for this time step. Hence, we set $\beta_0^0(l_0)$ to the prior $p(l_0)$ to indicate that the proprioceptive step is skipped. We again estimate the pose with an expectation:

$$\hat{l}_{n+1} = \mathbf{E}(\beta_{n+1}^{n+1}(l_{n+1})) \tag{5.23}$$

This section introduced our general system, and applied Markov localization to it. The next section will investigate which regions of our environment (pose space) are well suited to Markov localization.

5.2 Information

This section introduces a metric for determining the performance of Markov localization applied to different regions of the sensor network. A measure of a region's ability to provide sensor measurements that reduce the estimation error is presented. Such a measure is referred to as information [78] or a localizability metric [63]. Information in our context can be understood by referring back to localization. Recall that Markov localization provides more than just the pose estimate $\hat{l}$; it provides the posterior distribution β encompassing all of the knowledge of the current pose. In particular, the β distribution can be use to determine the estimation error. Hence, by observing the change in the β distribution during the preceptive phase of localization, it can be determined which sensor readings are most instrumental in reducing estimation error; i.e., which regions have the most information. The development of information is done in 3 steps: entropy is defined, information is defined, and system assumptions are imposed to simply the representation of information.

Entropy [26] is defined for a discrete distribution $p(z)$ as

$$\mathcal{H}(p(z)) = -\sum_z p(z)\log(p(z)) \tag{5.24}$$

If $p(z)$ is localized to one point, the entropy is 0. As $p(z)$ spreads out across several points, its entropy increases toward infinity. Entropy is said to measure the *disorder* in a distribution; in our context, entropy is used to measure estimation error. Hence, the estimation error of a pose estimation distribution β, is given by its entropy $\mathcal{H}(\beta)$.

Next, the change in estimation error (entropy) during the preceptive phase of localization is computed. It is assumed that a robot arrives at a pose $l_0 \in \mathcal{L}$, computes the proprioceptive pose estimate $p_{l_0}(l)$, receives a sensor reading y, and, computes the preceptive pose estimate $p_{l_0}(l|y)$. The notation $p_{l_0}(l)$ and $p_{l_0}(l|y)$ is used in place of $\beta_n^n(l_{n+1})$ and $\beta_n^{n+1}(l_{n+1})$, respectively, to focus on a single preceptive update occurring at the pose l_0. In other words, this development is not interested in how the robot arrived at l_0. Hence, it is assumed that the proprioceptive pose estimate $p_{l_0}(l)$ is known, thereby neglecting all the details leading up to its computation. From this, information $I : \mathcal{L} \rightarrow [0, \text{inf})$ is defined as the *decrease* in estimation error (entropy) during the preceptive phase:

$$I(l_0) = \mathcal{H}(p_{l_0}(l)) - \mathcal{H}(p_{l_0}(l|y)) \tag{5.25}$$

In practice, the prior $p_{l_0}(l)$ and the sensor reading y are unknown a priori. This dilemma can be partially solved by eliminating y with an expectation:

$$I(l_0) = \mathcal{H}(p_{l_0}(l)) - E_y[\mathcal{H}(p_{l_0}(l|y))] \tag{5.26}$$

However, $p_{l_0}(l)$ cannot be eliminated in such a fashion: it is a complex quantity, resulting from potentially many steps of Markov localization, making it difficult to compute. Instead, when computing information, an approximation for $p_{l_0}(l)$ based on observations of the dynamics is substituted. This substitution is addressed in later sections along with each particular implementation. To simplify our forthcoming computations, 2 additional requirements are imposed: $p_{l_0}(l)$ is solely a function of $l - l_0$ and the set of possible sensor values $\mathcal{Y}$ is finite. The first requirement enforces that $p_{l_0}(l)$ has constant shape; hence, it has constant entropy: let $H_0 = \mathcal{H}(p_{l_0}(l))$. The second requirement allows information to include an (efficient) summation over sensor values. With these assumptions, information can be reduced further, arriving at the formulation determined by previous work [78]:

$$\begin{aligned}
I(l_0) &= H_0 - E_y[\mathcal{H}(p_{l_0}(l|y))] && (5.27)\\
&= H_0 - \sum_y p_{l_0}(y) H(p_{l_0}(l|y)) && (5.28)\\
&= H_0 + \sum_y p_{l_0}(y) \sum_l p_{l_0}(l|y) log(p_{l_0}(l|y)) && (5.29)\\
&= H_0 + \sum_y p_{l_0}(y) \sum_l \frac{p_{l_0}(l,y)}{p_{l_0}(y)} log(\frac{p_{l_0}(l,y)}{p_{l_0}(y)}) && (5.30)\\
&= H_0 + \sum_y \sum_l p_{l_0}(l,y) log(\frac{p_{l_0}(l,y)}{p_{l_0}(y)}) && (5.31)
\end{aligned}$$

Armed with a formula for information, we turn to detailing the steps needed to compute it. Understanding information and entropy as measures is discussed more in Section 7.1.

5.3 Information Map Computation

Using the analytical representation for information, this section formalizes the steps required to compute information across all possible poses. The final representation (matrix, grid, or 2 dimensional plot) is referred to as an *information map*. To compute the information map, an algorithm is presented that walks through the computation of each term in equation (5.31). Additionally, we investigate the computational complexity of this algorithm and review the alternative algorithm (suggested by other researchers [78] using a traditional robot platform) that reduces the overall complexity.

The steps required to exactly compute the information map are given by Algorithm 1. Computing the information given in equation (5.31), in turn, requires the computation of 3 terms: H_0, $p_{l_0}(l,y)$, and $p_{l_0}(y)$. As mentioned in Section 5.2, H_0 is independent of l_0; hence, H_0 is calculated once at the start of the algorithm (line 1). The remaining 2 terms must be calculated for each l_0 (line 2).

Algorithm 1 Information Map Computation

1: compute H_0
2: **for all** $l_0 \in \mathcal{L}$ **do**
3: compute $p_{l_0}(l)$
4: **for all** $(l,y) \in \mathcal{L} \times \mathcal{Y}$ **do**
5: compute $p(y|l)$
6: compute $p_{l_0}(l,y) = p_{l_0}(l)p(y|l)$
7: **for all** $y \in \mathcal{Y}$ **do**
8: compute $p_{l_0}(y) = \sum_{l \in \mathcal{L}} p_{l_0}(l,y)$
9: compute $I(l_0) = H_0 + \sum_{y \in \mathcal{Y}} \sum_{l \in \mathcal{L}} p_{l_0}(l,y) log(\frac{p_{l_0}(l,y)}{p_{l_0}(y)})$

Before computing $p_{l_0}(l,y)$ (line 6), 2 more terms are required: the prior $p_{l_0}(l)$ (line 3) and the sensor model $p(y|l)$ (line 5). Both these terms are implementation (and model) specific, so the computational details are postponed until needed. To lend more traction to these topics, the reader is directed to Section 5.4 where $p_{l_0}(l)$ is modeled as a uniform distribution over a square and $p(y|l)$

is modeled as a simple binary sensor. In general, our simulations use a simple prior distribution (uniform or normal) that is computationally easy to shift to l_0 during each iteration of the algorithm. The last term required to compute information is $p_{l_0}(y)$ (line 8) which directly follows from $p_{l_0}(l, y)$. Finally, information $I(l_0)$ (line 9) is computed as directed by equation (5.31).

A second look at Algorithm 1 reveals the time complexity. Working through this, the time required by basic operations (add, subtract, multiply, log, etc) is replaced and reduced to constants c_k without further explanation. Since the computation of $p_{l_0}(l)$ and $p(y|l)$ is model specific, the time required for these computations is denoted as t_p and t_s, respectively. Finally, since $\mathcal{L}$ and $\mathcal{Y}$ are finite sets, their cardinalities are denoted as n_l and n_y, respectively.

Starting at the beginning of the algorithm, note that H_0 is an entropy computation, which entails n_l logs, n_l multiplications, and $n_l - 1$ subtractions. Hence, the total time required for H_0 is $c_1 n_l - c_2$. The loop on line 2 requires n_l iterations, which serves as a multiplicative factor for the remaining computation. Rolling lines 3 through 6 together requires $t_p + n_l n_y(t_s + c_3)$ where c_3 represents the multiplication required for $p_{l_0}(l, y)$. Similarly, lines 7 and 8 entails $n_y(c_4 n_l - c_4)$ since for each y, $p_{l_0}(y)$ entails $n_l - 1$ additions at a cost of c_4. Finally, line 9 requires $n_l n_y$ divisions, logs, and multiplications and $(n_l - 1)(n_y - 1) + 1$ additions, totaling $c_5 n_y n_l - c_6(n_l + n_y) + c_7$. Putting these terms together (with the loop on line 2), the time complexity of Algorithm 2 is

$$(c_1 n_l - c_2) + n_l[t_p + n_l n_y(t_s + c_3) + n_y(c_4 n_l - c_4) + (c_5 n_y n_l - c_6(n_l + n_y) + c_7)] \tag{5.32}$$

Further reducing and combining constants, the big-O time complexity is

$$n_l^2[n_y(t_s + c_8) - c_6] - c_9 n_l n_y + n_l(t_p + c_{10}) - c_2 \in O(n_l^2 n_y t_s + n_l t_p) \tag{5.33}$$

where we have intentionally left t_p and t_s in the big-O notation since these terms are potentially functions of n_l and n_y. Note, the first and second terms of the complexity are associated with computation of the sensor model and the prior, respectively. If t_p and t_s are constants (as they are for many of our simulations), the time complexity reduces to $O(n_l^2 n_y)$. Hence, a practical implementation of Algorithm 1 must carefully implement the most deeply nested computations (lines 5 and 6) in order for the algorithm to remain feasible.

To reduce the time spent in lines 5 and 6, Algorithm 2, an approximate information map algorithm (developed by other researchers [78] using a traditional mobile robot platform), is reviewed. This algorithm exploits tacit assumptions about locality within the system. In particular, since the prior $p_{l_0}(l)$ gains most of its support within a small neighborhood of l_0, information computation is restricted to this neighborhood. More specifically, when $p_{l_0}(l)$ is generated, an approximate

Algorithm 2 Approximate Information Map Computation

compute H_0
for all $l_0 \in \mathcal{L}$ **do**
    compute $p_{l_0}(l)$ and the approximate support set L_{l_0}
    compute the approximate set of sensor readings Y_{l_0}
    for all $(l, y) \in L \times Y$ **do**
        compute $p(y|l)$
        compute $p_{l_0}(l, y) = p_{l_0}(l)p(y|l)$
    for all $y \in Y$ **do**
        compute $p_{l_0}(y) = \sum_{l \in L} p_{l_0}(l, y)$
    compute $I(l_0) = H_0 + \sum_{y \in Y} \sum_{l \in L} p_{l_0}(l, y) log(\frac{p_{l_0}(l,y)}{p_{l_0}(y)})$

support set $L_{l_0} \subset \mathcal{L}$ for $p_{l_0}(l)$ containing the *most relevant* points is also generated. Then, a reduced set of possible sensor values Y_{l_0} based on L_{l_0} is generated. The reduced sensor space is the primary contributor to reduced computation time. These optimizations are particularly well suited for the sensor network model where a robot only excites nearby sensor nodes. In effect, this algorithm switches from computing information at l_0 using the entire space $\mathcal{L} \times \mathcal{Y}$ to using only a localized space $L_{l_0} \times Y_{l_0}$.

Now, it is shown that reducing the computation space dramatically reduces the time complexity. The time required to compute the prior and its support set is denoted by $\hat{t}_p$. Additionally, the time required to compute the reduced sensor measurement space L_{l_0} is denoted by $\hat{t}_y$. Finally, the cardinality of the reduced spaces L_{l_0} and Y_{l_0} is denoted by $\hat{n}_l$ and $\hat{n}_y$, respectively. Walking through Algorithm 2 in a fashion similar to above computations, the time complexity is found to be

$$(c_1 n_l - c_2) + n_l[\hat{t}_p + \hat{t}_y + \hat{n}_l\hat{n}_y(t_s + c_3) + \hat{n}_y(c_4\hat{n}_l - c_4) + (c_5\hat{n}_y\hat{n}_l - c_6(\hat{n}_l + \hat{n}_y) + c_7)] \quad (5.34)$$

$$= n_l\hat{n}_l[\hat{n}_y(t_s + c_8) - c_6] - c_9 n_l\hat{n}_y + n_l(\hat{t}_p + \hat{t}_y + c_{10}) - c_2 \quad (5.35)$$

$$\in O(n_l\hat{n}_l\hat{n}_y t_s + n_l(\hat{t}_p + \hat{t}_y)) \quad (5.36)$$

Comparing equation (5.36) and equation (5.33), it is clear that the approximate algorithm has reduced the cost associated with the sensor model and potentially increased the cost associated with the prior and sensor measurement space. Again, if t_s, $\hat{t}_p$, and $\hat{t}_y$ are constants, the time complexity reduces to $n_l\hat{n}_l\hat{n}_y$ resting on the shoulders of the sensor model. Since the cardinality of the reduced spaces L_{l_0} and Y_{l_0} is much smaller than the whole space, the time complexity has been significantly

reduced. Based on this evidence, our work uses exclusively Algorithm 2 for simulations. Furnished with a feasible method of computing information maps, the next section presses forward with several simulations.

5.4 Simulations

Having previously traced through all the steps necessary to compute an information map, this section computes several such maps for a sensor network model. First, a basic robot prior and sensor network model are presented. Then, an information map is computed for this system and is discussed along with the required CPU time. Finally, to illuminate the effect a prior model has on the information map, several maps with varying priors are computed and discussed.

Field	size	100m x 100m
	grid	200 x 200
	sensor distribution	uniform
	N_s	30
Sensor	model	discrete,binary
	r_s	10 cells
	p_s	0.75
Estimator	prior model	discrete, box
	$\hat{r}_p$	*varies*

Table 5.1: System models and parameters used for computing several information maps.

A summary of the system details is provided in Table 5.1. The simulated sensor network has $N_s = 30$ sensors uniformly distributed on the playing field that spans 100m by 100m and is divided by a 200 by 200 regular grid. The global sensor model $p(y|l)$ is an independent combination of all the node sensor models:

$$p(y|l) = \prod_{m=1}^{N_s} p(y_m|l) \tag{5.37}$$

Each sensor is represented by the *discrete binary sensor model*. For the m^{th} node, during each sensing period, this model assumes that the sensor either reports a detection ($y_m = \circledast$) with probability p_s or remains silent ($y_m = \emptyset$) with probability $1 - p_s$. More precisely, the detection distribution for the m^{th} sensor, with sensing radius r_s, located at $z_m \in \mathcal{L}$ is given by

$$p(y_m = \circledast|l) = \begin{cases} p_s \text{ if } \|l - z_m\| \leq r_s \\ 0 \text{ otherwise} \end{cases} \tag{5.38}$$

where $\|\cdot\|$ is the infinity norm on the partition space $\mathcal{L}$ and is the larger of either cells horizontally or cells vertically. In this norm, a radius of r covers a $2r+1$ by $2r+1$ square of cells; hence, $d = 2r+1$ is referred to as the diameter of such a set of cells.

A binary sensor model may seem trivial, but it accurately captures the details of certain mote sensors used in real deployments. Indeed, during PEG, the magnetometer sensor (with an r^3 response drop-off and threshold detection) amounted to a binary detector. Other researchers [4] have also identified the usefulness of a binary sensor model in sensor network environments. For this simulation, the sensor has a detection radius of $r_s = 10$ cells and a detection probability of $p_s = 0.75$.

The robot prior $p_{l_0}(l)$ is taken to be uniform on a square of grid cells. In particular, $p_{l_0}(l)$ is represented by the *discrete box prior* model with a radius of $\hat{r}_p$ cells:

$$p_{l_0}(l) \sim U(\{l_0^1 - \hat{r}_p, l_0^1 - \hat{r}_p + 1, \ldots, l_0^1 + \hat{r}_p\} \times \{l_0^2 - \hat{r}_p, l_0^2 - \hat{r}_p + 1, \ldots, l_0^2 + \hat{r}_p\}) \tag{5.39}$$

Hence, $p_{l_0}(l)$ is uniform across a $\hat{d}_p$ by $\hat{d}_p$ square of cells centered at l_0 where $\hat{d}_p = 2\hat{r}_p + 1$.

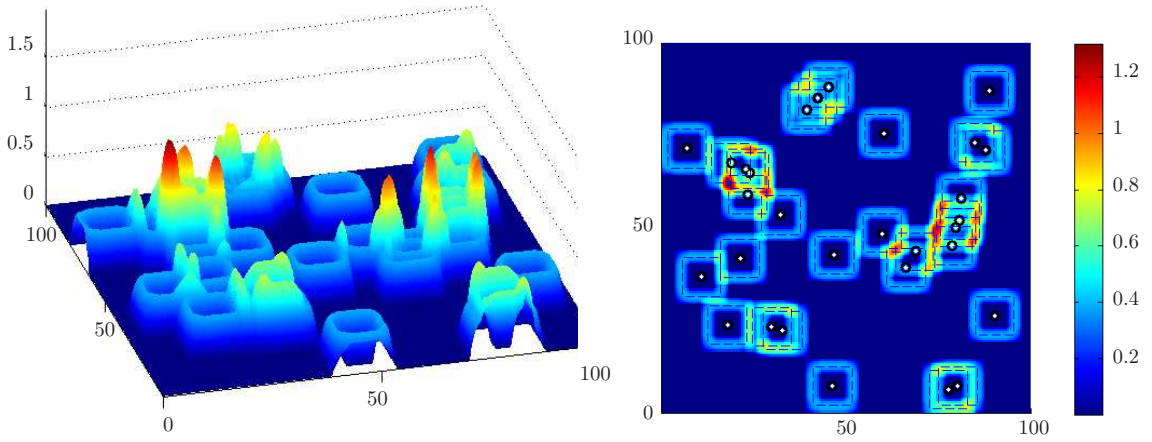
Before the simulation can commence, the computation of L_{l_0} and Y_{l_0} must be outlined. Computing L_{l_0} is straightforward: the exact support for $p_{l_0}(l)$ (i.e., the $\hat{d}_p$ by $\hat{d}_p$ square centered at l_0) is used. To compute $Y_{l_0} \subset \mathcal{Y} = \{y = (y_1, y_2, \ldots, y_{N_s}) | y_m \in \{\emptyset, \circledast\}\}$, observe that sensors *close* to l_0 can either report ($y_m = \circledast$) or not report ($y_m = \emptyset$), whereas, sensors *far* from l_0 will not report ($y_m = \emptyset$). Hence we let Y_{l_0} be such a set:

$$Y_{l_0} = \{y = (y_1, y_2, \ldots, y_{N_s}) | y_m \in Y_{l_0}^m\} \tag{5.40}$$

where

$$Y_{l_0}^m = \begin{cases} \{\emptyset, \circledast\} \text{ if } \|l_0 - z_m\| \leq r_s + \hat{r}_p \\ \{\emptyset\} \text{ otherwise} \end{cases} \tag{5.41}$$

Using the above details, the simulation proceeds for varying prior radii. The information map computed with $\hat{r}_p = 4$ cells is displayed in Figure 5.2 as both a 2-dimensional and 3-dimensional plot. In the 2-dimensional plot, nodes and their sensing regions are identified by white dots and dark blue dashed lines, respectively. The color background on the plots represent the amount of information. These 2 plots with a vibrant color scheme are included to highlight the variation in information across our playing field. In particular, information poor regions occur in 2 places: far from sensor coverage and close to node locations. For regions far from sensing coverage it is intuitive that the robot gleans no estimation help. In these regions, nodes do not report



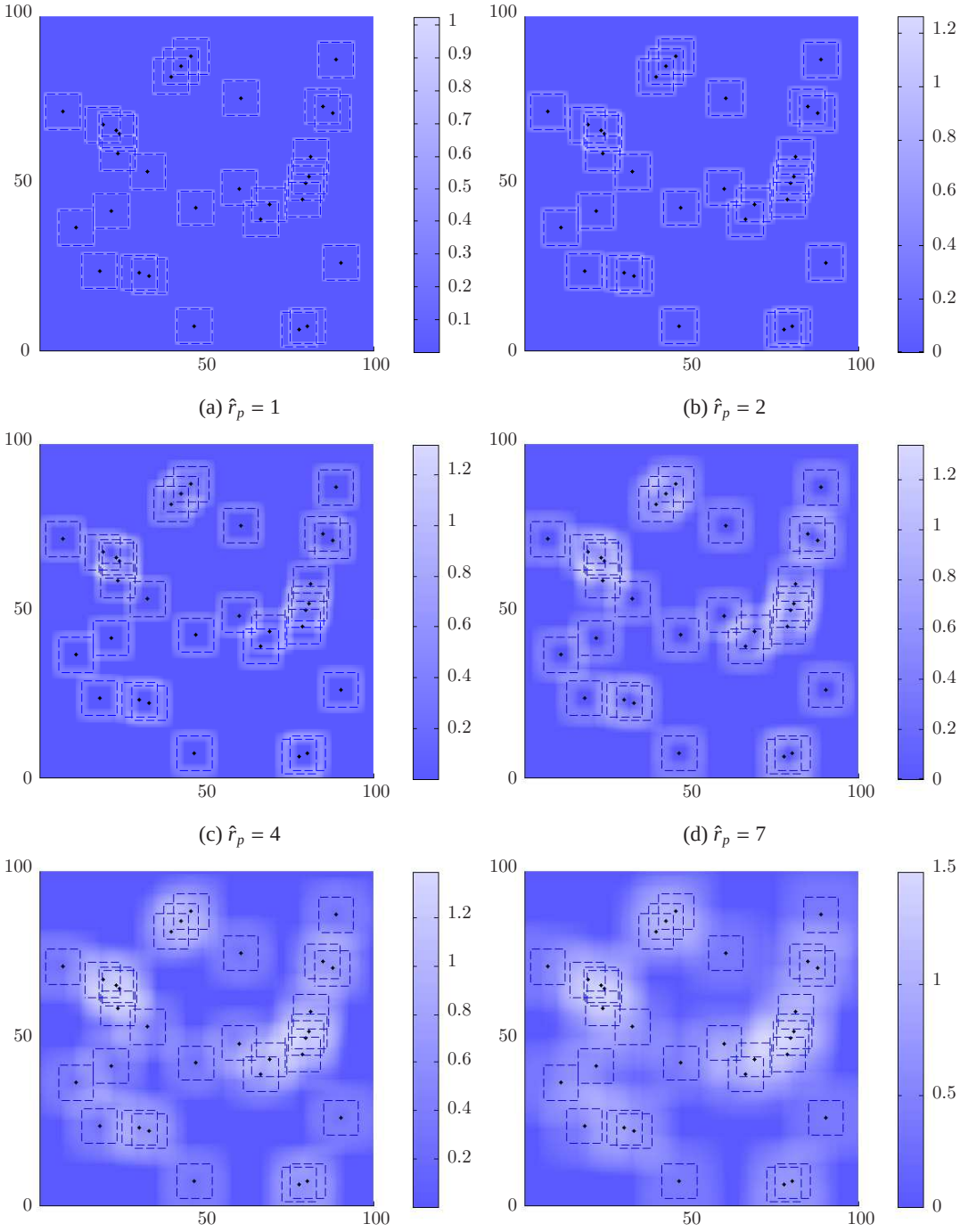
(a) 3-dimensional view

(b) 2-dimensional view

Figure 5.2: An information map computed using discrete, binary sensors and an uniform prior with radius $\hat{r}_p = 4$ cells as detailed in Table 5.1.

detections which is expected by Markov localization; hence, no information is provided to the estimator. However, it may seem surprising that no information is gained by a robot very close to a node. This occurs when a robot next to a node has its prior entirely enclosed in the node's sensing radius. Whether the node detects the robot or not is of no consequence to the robot since it already knows its pose with a greater accuracy than the node can provide. Turning to the information rich regions, we might have guessed that these regions were at the center of nodes, but that fallacy has been dispelled. Instead, the highest information regions are those close to a node's edge of detection, or, better yet, in regions close to several sensing edges. In these regions, the robot's prior tends to be partially in and partially out of a sensing region. If the mote reports, the estimation routine only retains the outside part of the prior; if the mote does not report, only the inside part is retained. Hence, these regions provide a good way for the robot to quickly shave off part of its prior and reduce its estimation error. In a nutshell, the robot is better off near detection edges where the chances of being detected and going undetected are roughly equal. Hence, a robot that navigates between several sensing regions without being detected often attains a lot of information. Later sections will show that robots navigating intelligently *around* nodes often do better than ones navigating *close to* nodes.

Information maps computed for 6 different prior radii ($\hat{r}_p \in \{1, 2, 4, 7, 12, 17\}$) are shown in Figure 5.3. These plots, and all the remaining information plots in this paper, use a 2-dimensional



(a) $\hat{r}_p = 1$ (b) $\hat{r}_p = 2$

(c) $\hat{r}_p = 4$ (d) $\hat{r}_p = 7$

(e) $\hat{r}_p = 12$ (f) $\hat{r}_p = 17$

Figure 5.3: Information maps computed using discrete, binary sensors and an uniform prior with varying radius $\hat{r}_p$ as detailed in Table 5.1.

representation. Additionally, to increase the legibility of future plots that include more symbols, the plots switch to a muted color scheme of blue tones. These plots show that as the prior's radius increases, fewer regions in the map are void of information. This occurs because a large prior has a good chance of intersecting one or more sensing regions, and, regardless of what the sensor network reports, the large prior can be trimmed down. In particular, notice that with $\hat{r}_p \in \{1, 2, 4, 7\}$, the center of a node's sensing region has a information null since the robot's prior is small enough to fit entirely in the sensing region. However, once the prior's radius $\hat{r}_p$ exceeds the sensing radius r_s in the last 2 plots, the entire node's sensing region becomes useful. However, the sensing center still remains a relatively low information region, contradicting the intuitive notion that a robot benefits the most by driving close to nodes. Additionally, these plots underscore the importance of choosing an accurate prior for ensuring that information maps are representative of localizability.

The approximate information map algorithm is carried out in MATLAB R14 on a 2.6GHz P4 with 1GB RAM. The computation time required for each simulation is summarized in Table 5.2. The mean time required to compute each map steadily increases from 17.01 minutes for a small prior to 116.77 minutes for a large prior. Furthermore, the mean time required to update each cell varies from 25.5 ms to 175.2 ms. This indicates that a robot can update the information map in real time to compensate for a changing environment. For instance, if a node goes online or offline, the information map should be updated in the $2(\hat{r}_p + r_s) + 1$ diameter square centered at this node. For our simulations with $r_s = 10$ cells and $\hat{r}_p = 4$ cells, the map could compensate for such a node (updating a 29 by 29 region of the map) in about 26.6 seconds. Hence, occasional changes in the environment could be accounted for in real time.

$\hat{r}_p$	cell computation	grid computation	
	mean (ms)	total (s)	total (min)
1	25.5	1020.72	17.01
2	28.9	1154.50	19.24
4	31.6	1264.28	21.08
7	51.1	2044.17	34.07
12	98.6	3942.63	65.71
17	175.2	7006.23	116.77

Table 5.2: Time required by a 2.6GHz Pentium4 with 1GB RAM running MATLAB R14 to compute an information map using the parameters from Table 5.1.

This chapter presented an overview of information maps for sensor networks. Markov localization and information were reviewed analytically and algorithmically. Special attention was

paid to the time complexity of computing information for an entire region. A system model for a robot localizing in a sensor network was developed and several simulations were performed with this model. The simulation results provide evidence that robots localizing in sensor networks will benefit by navigating close to sensor's edge of detection. The next chapter investigates this idea more formally, and compares the localization accuracy of robots that obey and disregard this idea.

Chapter 6

Localization using Information Maps

Now that the generation of information maps has been throughly studied, the remaining work will focus on applying information to aid in localization. Our work methodically builds up to simulations of realistic, closed-loop navigation systems starting with simple, open-loop, zero noise systems. This chapter focuses strictly on open-loop systems to validate the applicability of information maps. First, the steps required to compute localization updates needed by the simulator are presented. Next, an initial localization simulation is perused and used to demonstrate the ability of an information based path to improve localization performance over other path techniques. After this, the basic system models are relaxed to more realistic models and are adapted for use with information maps and Markov localization. Finally, the new models are used to evaluate the ability of information based paths to improve localization accuracy and confidence.

6.1 Markov Localization Revisited

Before beginning any localization simulations, this section takes a methodological look at Markov localization. Using the analytical introduction to Markov localization from Section 5.1, this section outlines the necessary computational steps. Additionally, the time complexity is analyzed and an approximate alternative to exact Markov localization is reviewed.

The presentation of the Markov localization computation first walks through the iterative update for time $n + 1$, and then returns to discuss the initialization of the algorithm at time 0. The update algorithm for time $n + 1$ is shown in Algorithm 3. The algorithm is initialized (line 1) with a new sensor reading y_{n+1}, the last control value u_n, and the previous iteration's posterior distribution $\beta^n_{n-1}(l_n)$. The remainder of the algorithm computes, in order, the proprioceptive update given by

equation (5.21), the preceptive update given by equation (5.22), and the expected pose given by equation (5.23). For each possible new pose l_{n+1}, the proprioceptive update $\beta_n^n(l_{n+1})$ is achieved on line 6. Prior to this, the transition probability $p(l_{n+1}|l_n, u_n)$ is computed (line 4) and applied (line 5) for each possible previous pose l_n. In general, the computation of $p(l_{n+1}|l_n, u_n)$ must be handled with care to ensure this part of the update algorithm is feasible for a large pose space $\mathcal{L}$. Since this computation is dependent on the dynamics, the computational details are addressed later for specific models. Next, the algorithm prepares to compute the preceptive update (achieved on line 11). First, the sensor model $p(y_{n+1}|l_{n+1})$ is computed (line 7) using the new sensor reading, and the result is merged with the proprioceptive distribution (line 8). Again, the computation of $p(y_{n+1}|l_{n+1})$ is dependent on the system model; hence, the details are delayed until necessary. Next, the algorithm exits the loop over l_{n+1}, and computes the normalization constant ζ_{n+1} (line 9). Finally, for each new pose l_{n+1}, the preceptive update is computed (line 11). The last step (line 12) is to compute the estimate of the pose using an expectation of the posterior $\beta_n^{n+1}(l_{n+1})$.

To initialize the algorithm at time 0, only a preceptive update, as mentioned in Section 5.1, is computed. In particular, given a prior $p(l_0)$ and a new sensor reading y_0, the proprioceptive update computation (lines 3-6) is skipped and $\beta_0^0(l_0)$ is set to the prior $p(l_0)$. After which, normal operation is resumed to compute $\beta_0^1(l_0)$ and $\hat{l}_0$. The computational details of the prior $p(l_0)$ are delayed until necessary.

Algorithm 3 Markov localization update

1: given $\beta_{n-1}^n(l_n)$, y_{n+1}, and u_n
2: **for all** $l_{n+1} \in \mathcal{L}$ **do**
3: **for all** $l_n \in \mathcal{L}$ **do**
4: compute $p(l_{n+1}|l_n, u_n)$
5: compute $\gamma(l_{n+1}, l_n) = p(l_{n+1}|l_n, u_n)\beta_{n-1}^n(l_n)$
6: compute $\beta_n^n(l_{n+1}) = \sum_{l_n} \gamma(l_{n+1}, l_n)$
7: compute $p(y_{n+1}|l_{n+1})$
8: compute $\varphi(l_{n+1}) = p(y_{n+1}|l_{n+1})\beta_n^n(l_{n+1})$
9: compute $\zeta_{n+1} = \sum_{l_{n+1}} \varphi(l_{n+1})$
10: **for all** $l_{n+1} \in \mathcal{L}$ **do**
11: compute $\beta_n^{n+1}(l_{n+1}) = \varphi(l_{n+1})/\zeta_{n+1}$
12: $\hat{l}_{n+1} = \mathbf{E}(\beta_n^{n+1}(l_{n+1}))$

Walking through Algorithm 3, the time complexity is computed using the same procedure as that used in Section 5.3. We find the big-O time complexity as

$$n_l[n_l(t_l + c_1) + (n_l - 1)c_2 + t_s + c_3] + (n_l - 1)c_4 + c_5 n_l + (n_l - 1)c_6 \tag{6.1}$$

$$= \quad n_l^2(t_l + c_7) + n_l(t_s + c_8) + c_9 \tag{6.2}$$

$$\in \quad O(n_l^2 t_l + n_l t_s) \tag{6.3}$$

where t_l and t_s are the times required to compute the transition distribution $p(l_{n+1}|l_n, u_n)$ and the sensor model $p(y_{n+1}|l_{n+1})$, respectively. Since the computation of both the transition and sensor model can greatly impact the running time of Markov localization, these computations must be efficient. In particular, the time required for the transition model must be limited since it is magnified by n_l^2. The time required to compute the transition and sensor distributions is discussed in later sections along with the accompanying system models.

If t_l and t_s are constants, the big-O complexity reduces to n_l^2. This term represents the size of the space to be updated. In order to reduce this term, our simulations apply the selective update approach from the literature [10]. This approach approximates the preceptive update step by only using the previously detailed update algorithm for cells with $\beta_n^n(l_{n+1})$ larger than a threshold. The remaining cells are updated with the simple rule

$$\beta_n^{n+1}(l_{n+1}) = \lambda \beta_n^n(l_{n+1}) \tag{6.4}$$

where λ is chosen to normalize the resulting $\beta_n^{n+1}(l_{n+1})$ distribution. For our simulations, this approach can dramatically reduce the time required for sensory updates.

6.2 Simulations

This section and the remainder of this chapter are dedicated to using information maps to improving robot localization in sensor networks. After discussing information maps (Sections 5.2, 5.3, and 5.4) and the details of robot localization (Sections 5.1 and 6.1), this section combines the two concepts in simulation. First, a full system model, now including robot dynamics, is presented. Then, an initial set of localization simulations are performed. Finally, the results are discussed.

The localization simulations entail a robot traversing a sensor network, receiving measurements, and estimating its location. These simulations are specified by several details:

- the field and sensor layout

Field	size	100m x 100m
	grid	200 x 200
	sensor distribution	uniform
	N_s	30
Sensor	model	discrete, binary
	r_s	10 cells
	p_s	0.75
Estimated Prior	prior model	discrete, box
	$\hat{r}_p$	4 cells
Estimator	prior model	discrete, box
	r_p	50 cells
Dynamics	model	discrete, adjacent cell
	U	4 cells
Pose	l_0	$\begin{bmatrix}5 & 10\end{bmatrix}^T$
	l_{goal}	$\begin{bmatrix}70 & 90\end{bmatrix}^T$
Simulation	trials	100
	path type	xy, yx, straight, node centers, manual

Table 6.1: System models and parameters used during our initial localization simulations.

- the sensor model
- the information map parameters (estimated prior)
- the initial conditions of the estimator
- the robot path
- the robot dynamics

For our initial simulations, these details are summarized in Table 6.1, but require more explanation. Much of the setup is inherited from the previous information map simulations in Section 5.4. More specifically a 100m by 100m field with 30 sensors uniformly distributed is used. Each sensor uses the discrete binary sensor model with a detection radius $r_s = 10$ cells and a detection probability p_s of 0.75. The information map, computed before the localization simulation, uses the discrete box prior model with radius $\hat{r}_p = 4$ cells. The estimator is initialized with the robot prior $p(l_0)$ defined as a discrete box prior with radius $r_p = 50$ cells centered about the robot's initial position. Notice, the distinction between r_p and $\hat{r}_p$: r_p is the initial radius of the prior during simulation, $\hat{r}_p$ is the prior radius assumed during computation of the information map. The route the robot takes is varied across simulations to compare localization error associated with different paths. However,

the robot is required to start at pose l_0 (in the lower left) and end at pose l_{goal} (in the upper right). The robot model is defined using the *discrete adjacent cell dynamics*. The discrete adjacent cell dynamics specify that a robot's pose $l \in \mathcal{L}$ evolves according to

$$l_{k+1} = l_k + u_k \tag{6.5}$$

where

$$u_k \in \mathcal{U} = \{-U, -U+1, \ldots, 0, \ldots, U-1, U\}^2 \subset \mathbf{Z}^2 \tag{6.6}$$

In other words, the robot may move up to U cells (4 cells for this simulation) both horizontally and vertically.

To test the ability of the information map to predict localizability, several different robot path schemes are tested:

- xy: drive in the x-direction, then in the y-direction
- yx: drive in the y-direction, then in the x-direction
- straight: drive in a straight line
- node centers: drive through several sensing centers on the way to the destination
- manual: using the information map, a human chooses a path that takes the robot through high information regions

The first 3 path schemes represent basic paths that the robot might use to get to the destination. These schemes assume no information is known about the environment. The next scheme, node centers, is generated by a path planner that tries to drive through the center of several sensing regions on its way to the destination. This path planner is described in more detail in Section 7.2. For now, it is assumed that this path is provided. The final path scheme, manual, represents a path chosen by us after looking at the information map. This scheme is an attempt by a human to maximize the overall information the robot encounters on its way to the destination.

Since this is the first set of localization simulations, the mechanics and results of the simulation are perused. Prior to running the localization simulation, the information map is computed as described in the previous chapter. The result is shown in Figure 6.1. In this case, 3 high information regions are obvious in the field: in the upper left, in the lower right, and part way up the $(0, 0)$–$(100, 100)$ diagonal. If the information maps are a good measure of localizability, a robot passing through these regions should have less estimation error than a robot that avoids these regions. To test

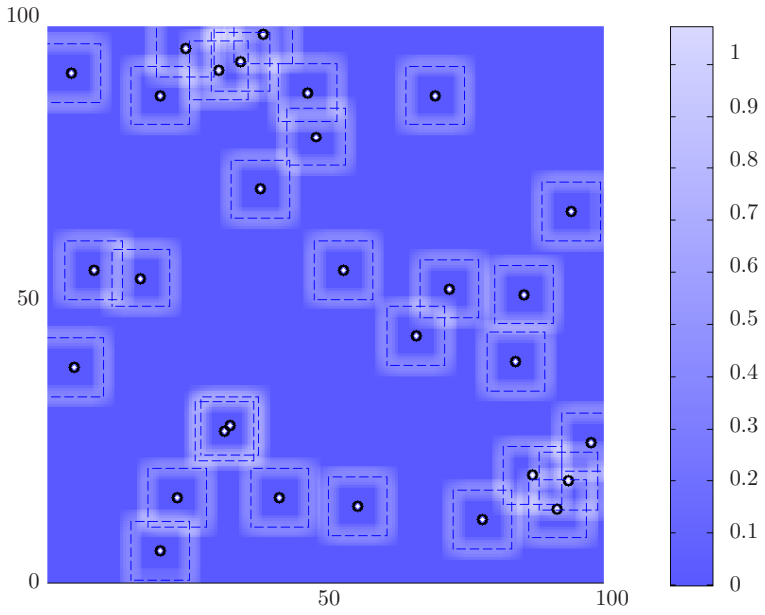


Figure 6.1: Information map for our initial localization simulations. White dots outlined in black represent motes and dotted squares represent their detection region. Background color of the plot represents the amount of information.

Path Type	Length (m)	Travel Time (s)	$\bar{n_d}$	$\bar{n_d}/100$ m
xy	145.00	7.30	19.72	13.60
yx	145.00	7.30	29.44	20.30
straight	103.08	7.10	4.56	4.42
node centers	252.66	20.40	133.92	53.00
manual	289.31	15.40	43.20	14.93

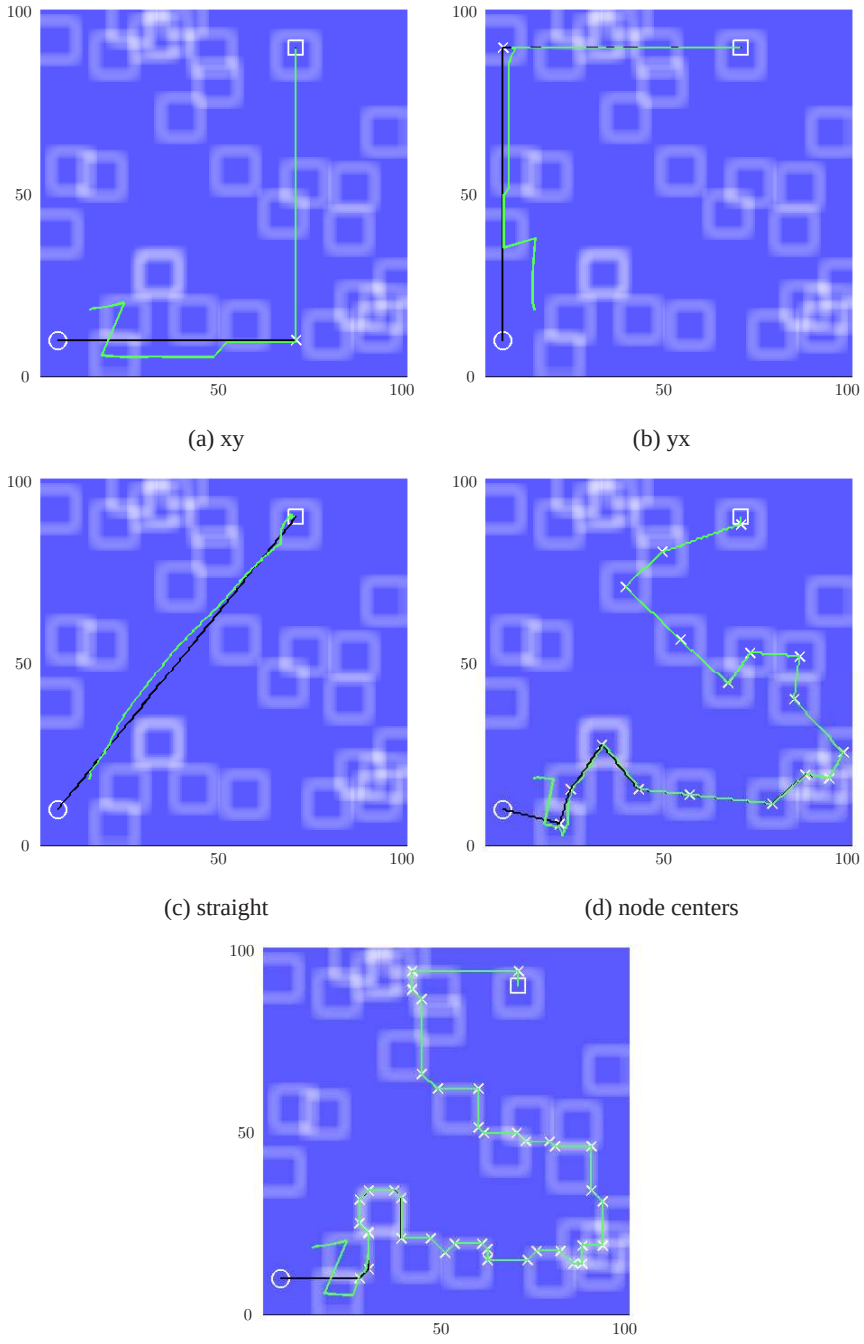
Table 6.2: Path information for each path scheme across all trials during our initial localization simulations.

Path Type	Estimation Error (m)			Estimation Entropy		
	Min	Mean	Max	Min	Mean	Max
xy	2.15	2.39	3.37	2.13	2.56	3.84
yx	2.55	2.69	3.51	3.00	3.33	4.41
straight	9.27	9.29	9.40	6.98	6.99	7.01
node centers	0.80	0.95	1.38	1.11	1.23	1.51
manual	0.65	0.73	1.03	0.57	0.63	0.85

Table 6.3: Estimation error and entropy for each path scheme across all trials during our initial localization simulations.

this, a robot is simulated following each of the aforementioned paths 100 times. Each trial records various statistics about the estimation performance. Each path scheme, estimated path information, and information map are displayed in Figure 6.2. These plots depict an overhead view of each path (solid black line) superimposed on the information map (background color). The white circles and squares represent the starting and ending positions of the robot, respectively. The light green line shows the estimated path for a *typical* trial. A typical trial is a trial such that the difference between its mean estimation error and the mean estimation error across all trials is minimal. For all paths, but the straight path, the typical estimated path eventually converges to the actual path before the destination is achieved. The straight path misses the detection region of all but a few motes making localization difficult. Shortly, we will investigate more precisely how this spoils the localization performance. Before leaving this figure, note that the node centers and manual paths tend to take a similar route that is also longer than other paths. These 2 paths provide an interesting point of comparison allowing us to quantify how much the localization accuracy can benefit with smarter paths.

Across all trials, for each path, several statistics are collected: the path length, the path travel time, the mean number of detections $\bar{n}_d$, the mean number of detections per 100 meters $\bar{n}_d/100$



(a) xy

(b) yx

(c) straight

(d) node centers

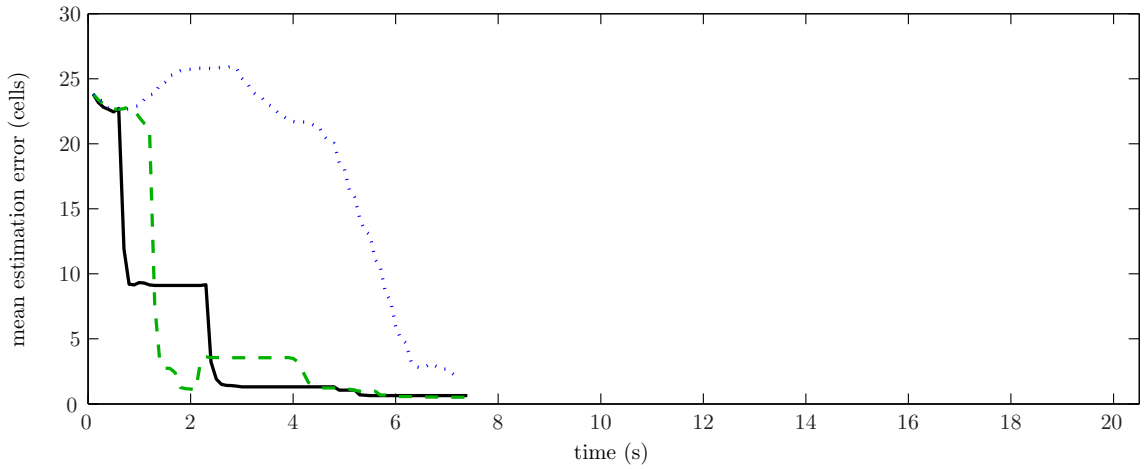
(e) manual

Figure 6.2: Basic simulation overview plots for our initial localization simulations depicting the robot starting (white circle) and ending (white square) positions, the robot path (solid black line), and a typical estimated robot path (solid light green line) superimposed on the information map (background color). Each figure is captioned with the path scheme depicted.

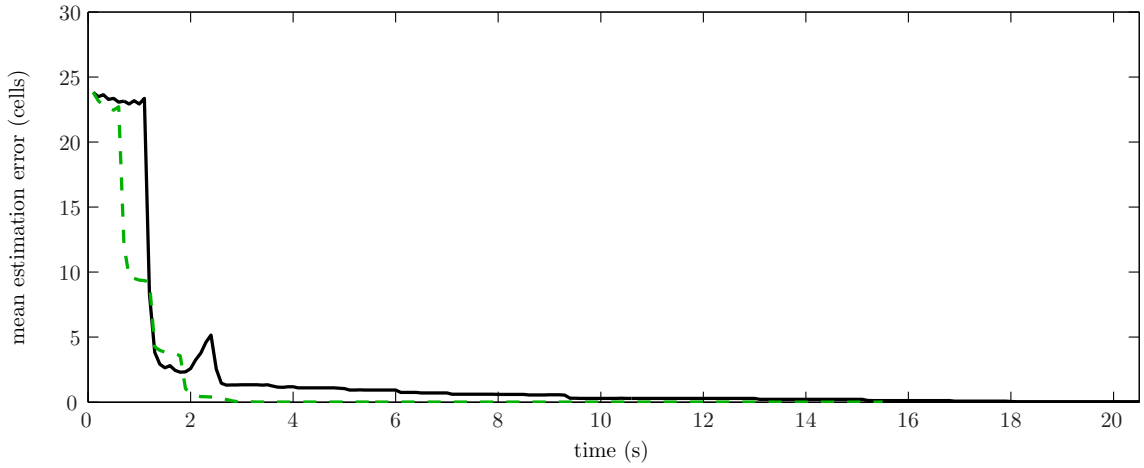
m, and the bounds and the mean of both estimation error and estimation entropy. These statistics are compiled in a path information table (Table 6.2) and an estimation information table (Table 6.3). Referring to Table 6.2, notice the large range in number of detections per 100 meters. The straight path only receives an average of 4.42 detections per 100 meters, which results in high estimation error and estimation entropy (see Table 6.3). On the other end of the spectrum, is the node centers path which generates an average of 53.00 detections per 100 meters by driving through the center of detection regions. In fact, this scheme does reduce the estimation error. One look at the manual path scheme, however, indicates that more detections does not always implies less error. The manual path, in comparison to node centers, is able to achieve lower estimation error with fewer detections. Let's investigate the mechanisms underlying this relationship by turning to the basic paths.

Note that yx underperforms xy, even though yx receives 49.3% more detections. There are 2 mechanisms contributing to this phenomena: the time lapse until first detection and the location of detections with respect to the current β distribution. The xy path takes the robot immediately through high information regions formed at the edge of several detection regions (Figure 6.2a). The yx path scheme (Figure 6.2b), however, requires more time before the robot encounters a detection region, but has the robot drive through the center of a detection region. For all schemes, the prior starts out as a uniform distribution over a large square region. As the robot progresses through the field, this region is reduced by detections (or lack of detections) eliminating possible poses and improving estimation accuracy. Since the prior's support region is initially large, often the best way to quickly reduce it is to get a detection. This first detection reduces the β distribution to (less than) the sensing region of the mote. If the robot is close to the center of this region, the estimated position (the mean of the β distribution) will have low error; if the robot, however, is closer to the edge of this region, the estimated pose will have higher error.

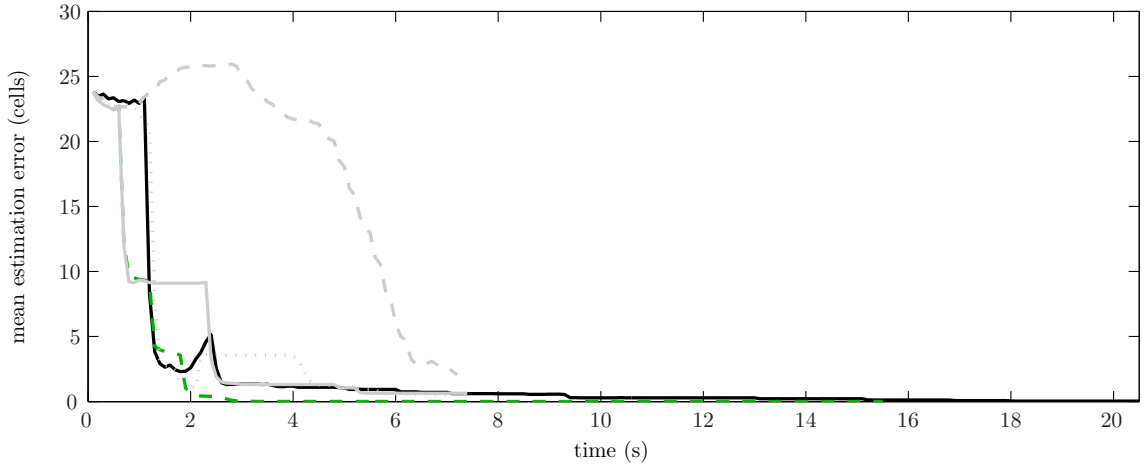
To further observe this phenomena, the average estimation error versus time is displayed in Figure 6.3. Figure 6.3a, shows xy's estimation error sharply falling off just before yx's does. This is caused by xy's path encountering a sensing region before yx's does. However, Figure 6.3a shows that when yx's estimation error falls off a little later (due to it's encounter with a detection region), it drops to a lower level. This is caused by yx leading the robot through the middle of the sensing region and xy leading the robot through the edge of a sensing region. Hence, the mean of the posterior distribution β better approximates the actual robot pose for yx than xy after the initial detections. However, even with yx achieving a better posterior pose estimation by $t = 2$ seconds, and receiving more detections overall, it still under performs the xy path scheme. This is due to xy more consistently reducing the entropy by encountering high information regions (or detection region



(a) xy (solid black), yx (dashed green), straight (dotted blue)



(b) node centers (solid black),manual (dashed green)



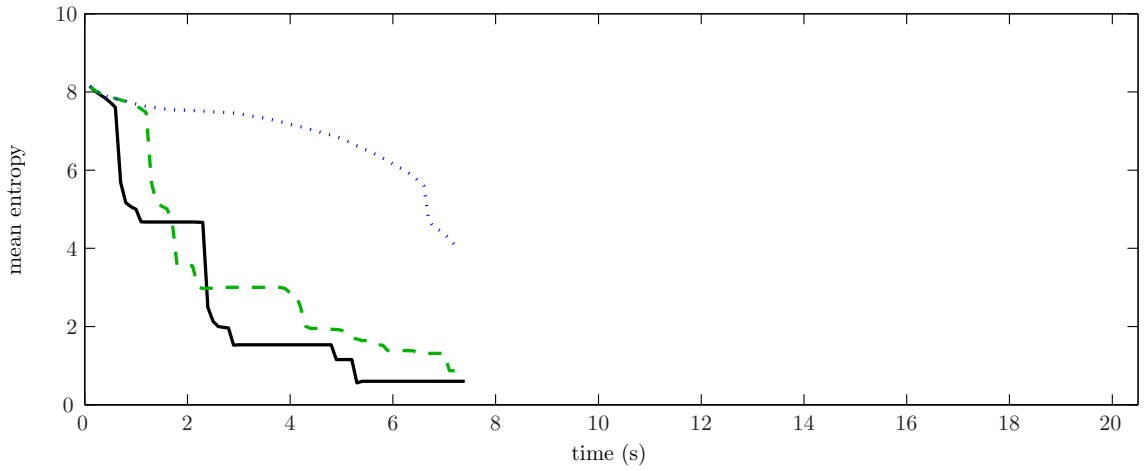
(c) all path schemes

Figure 6.3: Mean estimation error versus time during the initial localization simulations. Captions denote the path schemes displayed.

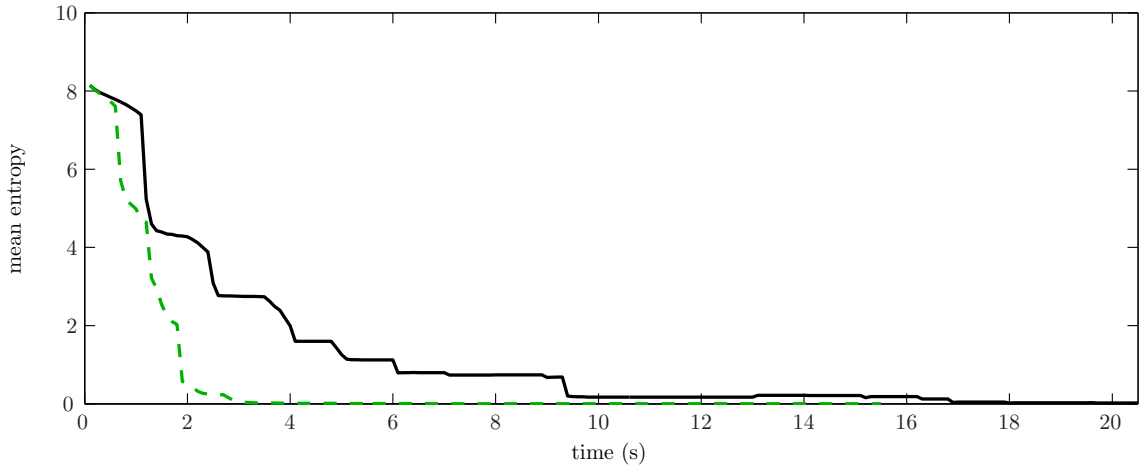
edges). To observe this effect, the mean estimation entropy versus time is shown in Figure 6.4. Notice, xy, as expected, has lower entropy more often than yx. Furthermore, the moment at which yx has lower entropy is during the time yx lies in the middle of a sensing region and receives several detections, amounting to a temporary win for yx at the cost of network bandwidth.

Now, we turn to comparing more precisely the differences in the 2 intelligent path schemes: node centers and manual. Turning back to Table 6.3, notice that the path explicitly using the information map (manual path) has achieved the best estimation performance. In particular, the manual scheme has 23.16% less mean estimation error and 48.78% less mean entropy than the respective next best schemes. Additionally, it is encouraging that the manual path only requires a modest number of detections. The next best performing scheme, node centers, requires 210.00% more detections than the manual scheme. The success of the manual path scheme is reiterated by Figure 6.3 where except for a brief moment, the manual path always maintains the lowest estimation error. Finally, observe that, as expected, the manual path scheme maintains lower estimation entropy at all times than any other scheme as shown in Figure 6.4.

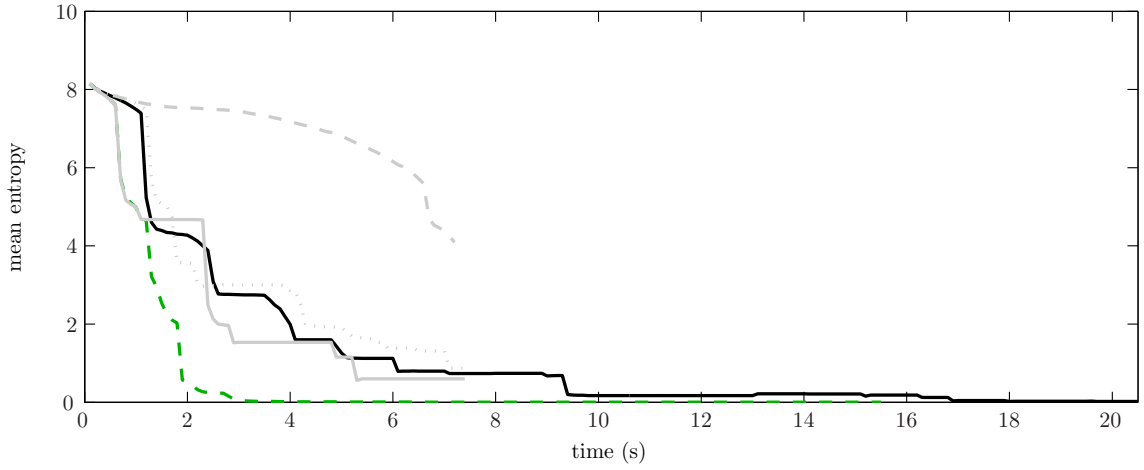
These simulations provide insight into many of the mechanisms at play for good localization. However, since the system dynamics have no noise, the results are focused on how efficiently the system can reduce the large staring prior (a 101 cell by 101 cell square) to an accurate position estimate. This convergence information is summarized in Table 6.4. Notice that only the manual scheme converges to within a 1 cell accuracy of the actual position. Furthermore, the only other path scheme to converge to within 2 cells takes 129.74% longer than the manual path scheme. Overall, use of the information map allows the robot to achieve the lowest mean estimation error (by 23.16%), the lowest mean entropy (by 48.78%), and the best convergence times (56.47% faster to converge to 2 cell accuracy). Additionally, the manual path requires only a modest number of detections per 100 meter (about the same as the 3 simple planners required), providing low bandwidth utilization. Hence, we observe that good localization is not strictly how many detections are received. In fact, by navigating on the edge of detections, the robot maintains low estimation error and reduces the number of detections. The information map does not provide the whole story either. The previous map was computed with a prior radius $\hat{r}_p = 4$ cells implying that it is designed for use with a system whose estimation error lies in this range. We will show that information maps can be useful for maintaining low error, but may not reduce it as quickly as other tools. Presently, information maps have proved themselves as a useful tool for sensor network navigation. Encouraged by this, the next section extends the system models to be more realistic.



(a) xy (solid black), yx (dashed green), straight (dotted blue)



(b) node centers (solid black),manual (dashed green)



(c) all path schemes

Figure 6.4: Mean estimation entropy versus time during the initial localization simulations. Captions denote the path schemes displayed.

Path Type	Mean Time to Converge (s)			
	5.66m (8 cells)	2.83m (4 cells)	1.41m (2 cells)	0.71m (1 cell)
xy	2.33	2.45	—	—
yx	1.46	4.17	—	—
straight	—	—	—	—
node centers	1.33	2.59	4.48	—
manual	1.36	1.43	1.95	1.98

Table 6.4: Path estimation convergence times during the initial localization simulations.

6.3 Relaxing for Continuous Dynamics and Sensors

Now that the use of information maps has been justified, this section investigates how to relax the system models to more closely resemble those of realistic sensor network systems such as PEG. In this section, the discrete grid-based robot dynamics from the previous section are replaced with a continuous space model. Furthermore, the sensor model is relaxed from having a square detection grid to having a continuous circular detection area. Finally, to be more consistent with real-world dynamics, a robot error model is introduced and the prior is derived from a normal distribution rather than from the uniform distribution used previously.

6.3.1 Robot Dynamics

First, the robot dynamics are relaxed to model those of a simple holonomic robot equipped with a compass. The robot is assumed to be able to quickly set its wheel speed and propel itself in any direction. Given these constraints, the robot's state can neglect accelerations and orientation. More specifically, the robot's position in the field is represented with the continuous-valued pose $\bar{l}_k \in \bar{\mathcal{L}} \subset \mathbf{R}^2$ evolving according to the *continuous, normal noise dynamics* given by

$$\bar{l}_{k+1} = \bar{l}_k + u_k + d_k \tag{6.7}$$

where the control $u_k \in \mathbf{U} = [-U, U]^2 \subset \mathbf{R}^2$ and the disturbance values $d_k^1, d_k^2 \sim \mathcal{N}(0, \sigma_d^2)$ with a noise variance σ_d that is a function of the control u_k. In particular, for each component of the control u_k^m, we let

$$\sigma_d^m = \max\left\{\left|\frac{u_k^m}{U}\right|^{\frac{1}{2}} \sigma_{d,0} \ , \ \sigma_{d,0,min}\right\} \tag{6.8}$$

where $\sigma_{d,0}, \sigma_{d,0,min} > 0$. This enforces that, above a given control threshold, the variance varies linearly with the magnitude of the control.

Before Markov localization can be applied to the new dynamics, they must be converted to a transition distribution on the discrete pose space $\mathcal{L}$. This requires that the probability $p(l_{k+1}|l_k, u_k)$ can be computed for each possible discrete transition $(l_{k+1}, l_k) \in \mathcal{L}^2$ and a given control u_k. Since the distribution can be partitioned component-wise as $p(l_{k+1}|l_k, u_k) = p(l^1_{k+1}|l^1_k, u^1_k)p(l^2_{k+1}|l^2_k, u^2_k)$, the transition computation is derived for only one dimension, $m \in \{1, 2\}$. Furthermore, the new (single-dimensional) transition relation is computed in 2 steps: first, the starting pose l^m_k is assumed to lie within a grid $[b^m_1, b^m_2] \subset \bar{\mathcal{L}}^m$ and the ending pose l^m_{k+1} is fixed at a point $z \in \bar{\mathcal{L}}^m$; then, the ending pose l^m_{k+1} is allowed to vary across a grid cell $[a^m_1, a^m_2] \subset \bar{\mathcal{L}}^m$. In this derivation, when a pose is only known to be within a grid cell, the actual pose is assumed to be uniformly distributed across that grid cell.

Beginning the derivation, we assume that $\bar{l}^m_{k+1} = z$, $l^m_k = [b^m_1, b^m_2]$, and u^m_k is given. Then, the transition distribution is computed as

$$
\begin{aligned}
p(\bar{l}^m_{k+1} = z | l^m_k = [b^m_1, b^m_2], u^m_k) &= p(\bar{l}^m_k + u^m_k + d^m_k = z | l^m_k) && (6.9)\\
&= p(\bar{l}^m_k + d^m_k = z - u^m_k | l^m_k) && (6.10)\\
&= \int p(\bar{l}^m_k = w | l^m_k) p(d^m_k = z - u^m_k - w) dw && (6.11)\\
&= \int_{b^m_1}^{b^m_2} p(\bar{l}^m_k = w) p(d^m_k = z - u^m_k - w) dw && (6.12)\\
&= \frac{1}{b^m_2 - b^m_1} \int_{b^m_1}^{b^m_2} p(d^m_k = z - u^m_k - w) dw && (6.13)\\
&= \frac{1}{b^m_2 - b^m_1} \int_{z-u^m_k-b^m_2}^{z-u^m_k-b^m_1} p(d^m_k = y) dy && (6.14)\\
&= \frac{1}{b^m_2 - b^m_1} \int_{z-u^m_k-b^m_2}^{z-u^m_k-b^m_1} \mathcal{N}(y; 0, \sigma^2_d) dy && (6.15)\\
&= \frac{1}{2(b^m_2 - b^m_1)} \left[\operatorname{erf}\left(\frac{z - u^m_k - b^m_1}{\sqrt{2}\sigma_d} \right) - \operatorname{erf}\left(\frac{z - u^m_k - b^m_2}{\sqrt{2}\sigma_d} \right) \right] && (6.16)
\end{aligned}
$$

where $\mathcal{N}(y; 0, \sigma^2_d)$ is the normal probability distribution function over y with mean 0 and variance σ^2_d. Next, the ending pose is allowed to vary across a cell: letting $l^m_{k+1} = [a^m_1, a^m_2]$, equation (6.16)

can be integrated [39] to compute $p(l_{k+1}^m | l_k^m, u_k^m)$ as

$$\begin{aligned} & p(l_{k+1}^m = [a_1^m, a_2^m] | l_k^m = [b_1^m, b_2^m], u_k^m) \\ &= \frac{1}{2(b_2^m - b_1^m)} \int_{a_1^m}^{a_2^m} \left[\text{erf}\left(\frac{z - u_k^m - b_1^m}{\sqrt{2}\sigma_d} \right) - \text{erf}\left(\frac{z - u_k^m - b_2^m}{\sqrt{2}\sigma_d} \right) \right] \quad (6.17) \\ &= \frac{\gamma}{2(b_2^m - b_1^m)} \left[\zeta(\alpha_1) - \zeta(\alpha_2) - \zeta(\alpha_3) + \zeta(\alpha_4) \right] \quad (6.18) \\ &= \frac{\gamma}{2\Delta} \left[\zeta(\alpha_1) - \zeta(\alpha_2) - \zeta(\alpha_3) + \zeta(\alpha_4) \right] \quad (6.19) \end{aligned}$$

where $\gamma = \sqrt{2}\sigma_d$, ζ is given by

$$\zeta(\alpha) = \frac{\alpha}{\gamma} \text{erf}\left(\frac{\alpha}{\gamma} \right) + \frac{1}{\sqrt{\pi}} e^{-\frac{\alpha^2}{\gamma^2}} \quad (6.20)$$

and the α functions are given by

$$\begin{aligned} \alpha_1 &= a_2^m - b_1^m - u_k^m \quad (6.21) \\ \alpha_2 &= a_1^m - b_1^m - u_k^m \quad (6.22) \\ \alpha_3 &= a_2^m - b_2^m - u_k^m \quad (6.23) \\ \alpha_4 &= a_1^m - b_2^m - u_k^m \quad (6.24) \end{aligned}$$

This provides an analytical method of computing the transition probabilities. However, there are 2 difficulties associated with using this distribution for Markov localization. First, since this computation is derived from a model with normally distributed noise, the transition distribution smears the density across the entire field. It attributes a non-zero probability to even the most remote locations. This unnecessarily slows down the simulations. To avoid this, the noise is truncated in our model to a finite range. A truncated noise model is additionally justified because typically a robot does not experience large movement errors unless it encounters an obstacle or it transported to a different location, neither of which is part of our model. Additionally, for such an environment, a more applicable localization system such as multiple hypothesis tracking [48, 5] could be used. The second difficultly of using this transition distribution is that even with the error limited to a finite range this computation can be slow in practice. This issue is addressed by reformatting the computation of the proprioceptive update in the Markov localization algorithm for this model.

Henceforth, the noise d_k^1, d_k^2 is restricted to the region $[-D, D]$, and the normal distribution is replaced with the truncated normal distribution:

$$\hat{\mathcal{N}}(d^m; 0, \sigma_d^2) = \begin{cases} 0 & \text{if } d^m \notin [-D, D] \\ \frac{\alpha}{\sigma_d \sqrt{2\pi}} \exp(-\frac{(d^m)^2}{2\sigma_d^2}) & \text{otherwise} \end{cases} \quad (6.25)$$

Algorithm 4 Continuous Dynamics Proprioceptive Update

1: given $\beta_n(l_n)$, and u_n
2: **for all** $m \in \{1, 2\}$ **do**
3: compute $\delta^m_{min} = \lfloor \frac{u^m_n - D - \Delta/2}{\Delta} \rfloor$
4: compute $\delta^m_{max} = \lceil \frac{u^m_n + D + \Delta/2}{\Delta} \rceil$
5: let $\chi^m = \{\delta^m_{min}, \delta^m_{min} + 1, \ldots, \delta^m_{max}\}$
6: **for all** $\delta^m \in \chi^m$ **do**
7: compute $\alpha = (\delta^m + 1)\Delta - u^m_n$
8: compute $\Psi^m_{\delta^m} = \frac{\gamma}{2\Delta}[\zeta(\alpha) - 2\zeta(\alpha - \Delta) + \zeta(\alpha - 2\Delta)]$
9: **for all** $\delta^1 \in \chi^1, \delta^2 \in \chi^2$ **do**
10: compute $\beta^{\delta^1,\delta^2}_n(l_n)$: shift $\beta_n(l_n)$ by (δ^1, δ^2) adding zeros where necessary
11: compute $\beta_n(l_{n+1}) = \sum_{\delta^1 \in \chi^1, \delta^2 \in \chi^2} \Psi^1_{\delta^1} \Psi^2_{\delta^2} \beta^{\delta^1,\delta^2}_n(l_n)$

where

$$\alpha = \frac{1}{\text{erf}\,(D/\gamma)} \tag{6.26}$$

Then, to compute the proprioceptive update, lines 2-6 of Algorithm 3 are removed, and an updated computation tailored for the new model is inserted between lines 1 and 2. The new computation, listed as Algorithm 4, exploits 2 properties of the dynamics. First, the transition probability $p(l^m_{k+1}|l^m_k, u^m_k)$ can be reduced to a function of only the distance δ^m between the starting and ending grid cell, rather than a function of the actual grid cells. Second, since the control is known and the noise is bounded, the range χ^m of possible distances δ^m with non-zero transition probability is also known and bounded. More specifically, for any 2 (single-dimensional) grid cells separated by distance δ^m, the α functions in equations (6.21) through (6.24) only depend on δ^m. Hence, the ζ function in equation (6.20) and the transition distribution in equation (6.19) only depend on δ^m, allowing the proprioceptive update to be computed as a shift and a scaling: the prior distribution is shifted by δ^m and scaled by the transition probability for δ^m. Referring to Algorithm 4, the computation proceeds by computing the scaling factor $\Psi^m_{\delta^m}$ for a shift of δ^m in the m^{th} dimension (lines 2-8). For each dimension $m \in \{1, 2\}$, lines 3 through 5 compute the distances χ^m that the robot could move with potentially non-zero transition probability. Then, for each possible distance $\delta^m \in \chi^m$, lines 7 and 8 compute the scaling factor given by equation (6.19). Next, lines 9 and 10 compute, for each 2 dimensional shift (δ^1, δ^2), a shifted prior distribution $\beta^{\delta^1,\delta^2}_n(l_n)$. This distribution is computed

as

$$\beta_n^{\delta^1,\delta^2}(l_n) = \begin{cases} \beta_n(l_n - \langle \delta^1, \delta^2 \rangle) & \text{for } (l_n - \langle \delta^1, \delta^2 \rangle) \in \mathcal{L} \\ 0 & \text{otherwise} \end{cases} \tag{6.27}$$

Finally, line 11 computes the proprioceptive update as a superimposition of shifted and scaled priors.

This extra effort is not in vain; the time required by localization has been reduced significantly. Assuming that $n_\chi = \max\{|\chi^1|, |\chi^2|\}$, then an upper bound on the time complexity of this algorithm is found to be

$$2[c_1 + c_2 + n_\chi(c_3 + c_4)] + n_\chi^2 c_5 + (n_\chi - 1)^2 c_6 \tag{6.28}$$

$$= c_7 n_\chi^2 + c_8 n_\chi + c_9 \tag{6.29}$$

$$\in O(n_\chi^2) \tag{6.30}$$

Hence, the time required for the proprioceptive update is reduced from $O(n_l^2)$ required by the Markov localization algorithm, Algorithm 3, to $O(n_\chi^2)$. Furthermore, assuming that $\mathcal{L}$ is square, then $n_\chi < \sqrt{n_l}$. Typically, since only a small region of the grid cells are reachable for any time step, n_χ is much smaller than $\sqrt{n_l}$. Hence, the time complexity has been reduced from $O(n_l^2)$ to something less than $O(n_l)$.

The last detail required by the new dynamics model is a method of simulating the truncated normal distribution. To do this, the standard method of evaluating the inverse of the cumulative density function with a uniformly distributed random variable is used. In particular, given a random variable $v \sim \mathcal{U}[0, 1]$ and the cumulative density function

$$f(d) = \int_{-\infty}^{d} p(d_k^m) \tag{6.31}$$

the (single dimensional) noise is computed as

$$d_k^m = f^{-1}(v) \tag{6.32}$$

$$= \gamma \text{erf}^{-1}\left(\frac{2v - 1}{\alpha}\right) \tag{6.33}$$

where $\gamma = \sqrt{2}\sigma_d$ and $\alpha = 1/\text{erf}\,(D/\gamma)$.

6.3.2 Robot Prior

Since the error in the robot model is (approximately) normally distributed, the prior is also allowed to assume an (approximate) normal distribution. For a normal prior centered about a

robot at $\bar{l}_0 \in \bar{\mathcal{L}}$ with covariance $\begin{bmatrix} \hat{\sigma}_p^2 & 0 \\ 0 & \hat{\sigma}_p^2 \end{bmatrix}$, the prior on the discrete space $\mathcal{L}$ can be shown to be

$$p(l) = \frac{1}{4}\left[\text{erf}\left(\frac{\bar{g}_{i,j}^1 + \Delta/2 - \bar{l}_0^1}{\hat{\sigma}_p\sqrt{2}}\right) - \text{erf}\left(\frac{\bar{g}_{i,j}^1 - \Delta/2 - \bar{l}_0^1}{\hat{\sigma}_p\sqrt{2}}\right)\right]. \tag{6.34}$$

$$\left[\text{erf}\left(\frac{\bar{g}_{i,j}^2 + \Delta/2 - \bar{l}_0^2}{\hat{\sigma}_p\sqrt{2}}\right) - \text{erf}\left(\frac{\bar{g}_{i,j}^2 - \Delta/2 - \bar{l}_0^2}{\hat{\sigma}_p\sqrt{2}}\right)\right] \tag{6.35}$$

where $\bar{g}_{i,j} \in \bar{\mathcal{L}}$ is the center of cell l. To extend this to a truncated normal, $p(l)$ is set to 0 outside the $[-P, P]^2$ square centered at $\bar{l}_0$ and re-normalized.

6.3.3 Sensor Model

The binary sensor model is relaxed by allowing it to operate on the continuous-valued robot pose space $\bar{\mathcal{L}}$, and the detection area is allowed to become circular. Then, for each sampling period, the probability of detecting a robot at $\bar{l} \in \bar{\mathcal{L}}$ for the k^{th} sensor located at $\bar{a}_k \in \bar{\mathcal{L}}$ is given by the *continuous, circular, binary sensor model*:

$$p(y = \circledast|\bar{l}) = \begin{cases} p_s & \text{if} \quad \|\bar{l} - \bar{a}_k\|_2 \le r_s \\ 0 & \text{otherwise} \end{cases} \tag{6.36}$$

where $r_s > 0$ is the detection radius. Again, for Markov localization, this distribution must be tessellated to find $p(y|l)$. Similar work [66, 36] has been done on discretizing the sensor model for ultrasonic range sensors. Assuming that the robot pose is independent of the sensing model, $p(\bar{l}, y) = p(y|\bar{l})p(\bar{l})$, the distribution can be reduced:

$$p(y|l = \langle i, j\rangle) = p(y|\bar{l} \in \mathcal{G}_{i,j}) \tag{6.37}$$

$$= \frac{\int_{\bar{l}\in\mathcal{G}_{i,j}} p(\bar{l})p(y|\bar{l})}{\int_{\bar{l}\in\mathcal{G}_{i,j}} p(\bar{l})} \tag{6.38}$$

Since the exact robot pose is unknown a priori, it is assumed that $p(\bar{l}) \sim \mathcal{U}(\bar{\mathcal{L}})$, further reducing the distribution:

$$p(y|l = \langle i, j\rangle) = \frac{\int_{\bar{l}\in\mathcal{G}_{i,j}} p(y|\bar{l})}{\int_{\bar{l}\in\mathcal{G}_{i,j}} 1} \tag{6.39}$$

$$= \frac{1}{\Delta^2}\int_{\bar{l}\in\mathcal{G}_{i,j}} p(y|\bar{l}) \tag{6.40}$$

$$= \frac{p_s}{\Delta^2}\mathcal{A} \tag{6.41}$$

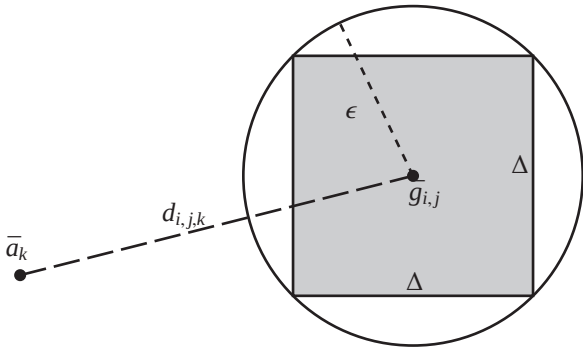


Figure 6.5: Approximating the area $\mathcal{A}$ used by the continuous, circular, binary sensor model.

where Δ is the grid side length and $\mathcal{A} = \text{Area}(\mathcal{G}_{i,j} \cap \{\bar{l} : d(\bar{l}, a) < r_s\})$.

The area $\mathcal{A}$ can be estimated with a few simple approximations. To begin with, the grid cell $\mathcal{G}_{i,j}$ is by circumscribed by a circle of radius $\epsilon = \frac{\Delta}{\sqrt{2}}$ as shown in Figure 6.5. We let $\bar{g}_{i,j} \in \bar{\mathcal{L}}$ be the center of the grid cell $\mathcal{G}_{i,j}$, and $d_{i,j,k}$ be the distance from the center to the k^{th} mote, that is, $d_{i,j,k} = \|\bar{g}_{i,j} - \bar{a}_k\|$. Then, we delineate the area computation for 3 ranges of $d_{i,j,k}$ values:

- $d_{i,j,k} \in [0, r_s - \epsilon]$: $\mathcal{A} = \Delta^2$
- $d_{i,j,k} \in (r_s - \epsilon, r_s + \epsilon]$: $\mathcal{A} = \Delta^2 \left(\frac{(r_s+\epsilon)-d_{i,j,k}}{2\epsilon}\right)$
- $d_{i,j,k} \in (r_s + \epsilon, \infty)$: $\mathcal{A} = 0$

The first and third terms are obvious: if the grid cell is entirely enclosed or entirely outside the sensing area, then the overlapping area $\mathcal{A}$ is Δ^2 or 0, respectively. When $\mathcal{G}_{i,j}$ is partially covered by the sensing region, as a easily computable approximation, we let $\mathcal{A}$ vary linearly with $d_{i,j,k}$. Now, using this in equation (6.41), the approximate discretized sensor model for Markov localization is found to be

$$p(y = \circledast | l) \approx \begin{cases} p_s & \text{if} \quad 0 \le \|\bar{a}_k - \bar{g}\| \le r_s - \epsilon \\ \frac{p_s(r+\epsilon-\|\bar{a}_k-\bar{g}\|)}{2\epsilon} & \text{if} \quad r_s - \epsilon \le \|\bar{a}_k - \bar{g}\| \le r_s + \epsilon \\ 0 & \text{if} \quad r_s + \epsilon \le \|\bar{a}_k - \bar{g}\| \end{cases} \tag{6.42}$$

where $\epsilon = \Delta/\sqrt{2}$.

We have now developed more realistic models for the robot, the sensor network, and the robot prior. These models have been allowed to operate in the continuous space $\bar{\mathcal{L}}$ and have been discretized for use with Markov localization. In the next section, the new models are applied to several simulations.

Field	size	100m x 100m
	grid	200 x 200
	sensor distribution	uniform
	N_s	30
Sensor	model	continuous, circular, binary
	r_s	5 m
	p_s	0.90
Estimated Prior	prior model	discretized, truncated normal
	$\hat{\sigma}_p$	0.75 m
	P	$3\hat{\sigma}_p = 2.25$ m
Estimator	prior model	discrete, box
	r_p	50 cells
Dynamics	model	continuous, normal noise
	U	5 m
	$\sigma_{d,0}$	0.1 m
	$\sigma_{d,0,min}$	0.01 m
	D	$3\sigma_d = 0.3$ m
Pose	l_0	$\begin{bmatrix}5 & 10\end{bmatrix}^T$
	l_{goal}	$\begin{bmatrix}70 & 90\end{bmatrix}^T$
Simulation	trials	100
	path type	xy, yx, straight, node centers, manual, hybrid

Table 6.5: Systems model and parameters used during the localization simulation with improved models.

6.4 Continuous Simulations

Using the new sensor, robot dynamics, and prior models from the previous section, this section runs several localization simulations. First, the simulation setup is described. Then, the results are presented and discussed.

The simulation setup is summarized in Table 6.5. The simulations use the same 100m by 100m field with 30 sensors uniformly deployed as the previous section. The robot's starting and ending points remain the same. The rest of the models, however, have changed. For the sensor nodes, the continuous, circular binary sensor model with detection radius $r_s = 5$ meters and detection probability $p_s = 0.90$ is used. The estimated prior, used for calculating the information map, is a truncated normal with standard deviation $\hat{\sigma}_p = 0.75$ m. The distribution is truncated at the $3\hat{\sigma}_p$ radius which amounts to a 9 cell by 9 cell support square (similar to previous simulations). The estimator is initialized with the same 50 cell radius discrete, box prior. The new continuous, normal noise dynamics model is used with control bound $U = 5$ meters and noise variance $\sigma_d^2 = 0.01$ me-

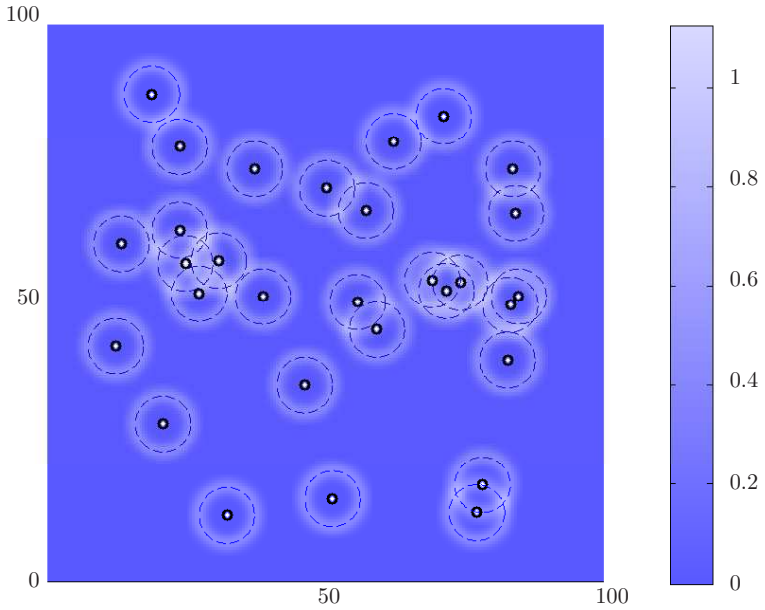


Figure 6.6: Information map using the improved models. White dots outlined in black represent motes, and dotted circles represent their detection region. Background color of the plot represents the amount of information.

ters. The noise remains low for this simulation so the effect the new models have on the simulation can be observed independently. Later, a closed-loop control will be added, and the noise variance will be increased. The same path schemes, save one, are used; the paths, however, they generate are different. One path scheme, hybrid, is added to the line up; it is a combination of 2 schemes: node centers and manual. The hybrid path initially follows the node centers path, but once the estimation entropy has decreased below a given threshold, it jumps over to the manual path and remains on this until the destination is achieved. This path scheme is motivated by the fact that, as noted previously, the information map is tuned to a estimation prior with supporting radius of 4 cells, and that driving through the node centers quickly reduces large initial priors. This path is assumed to be provided for now, but, later in this section, the determination of this path is explored.

As before, the information map is computed prior to simulation. The information map is shown in Figure 6.6. As expected, this information map has circular high information regions as opposed to the previous simulation with square regions. Additionally, the edges of the high information regions tend to roll off smoothly for this new map. This is due to the information map computation using a normal prior with circular support instead of a uniform prior with square

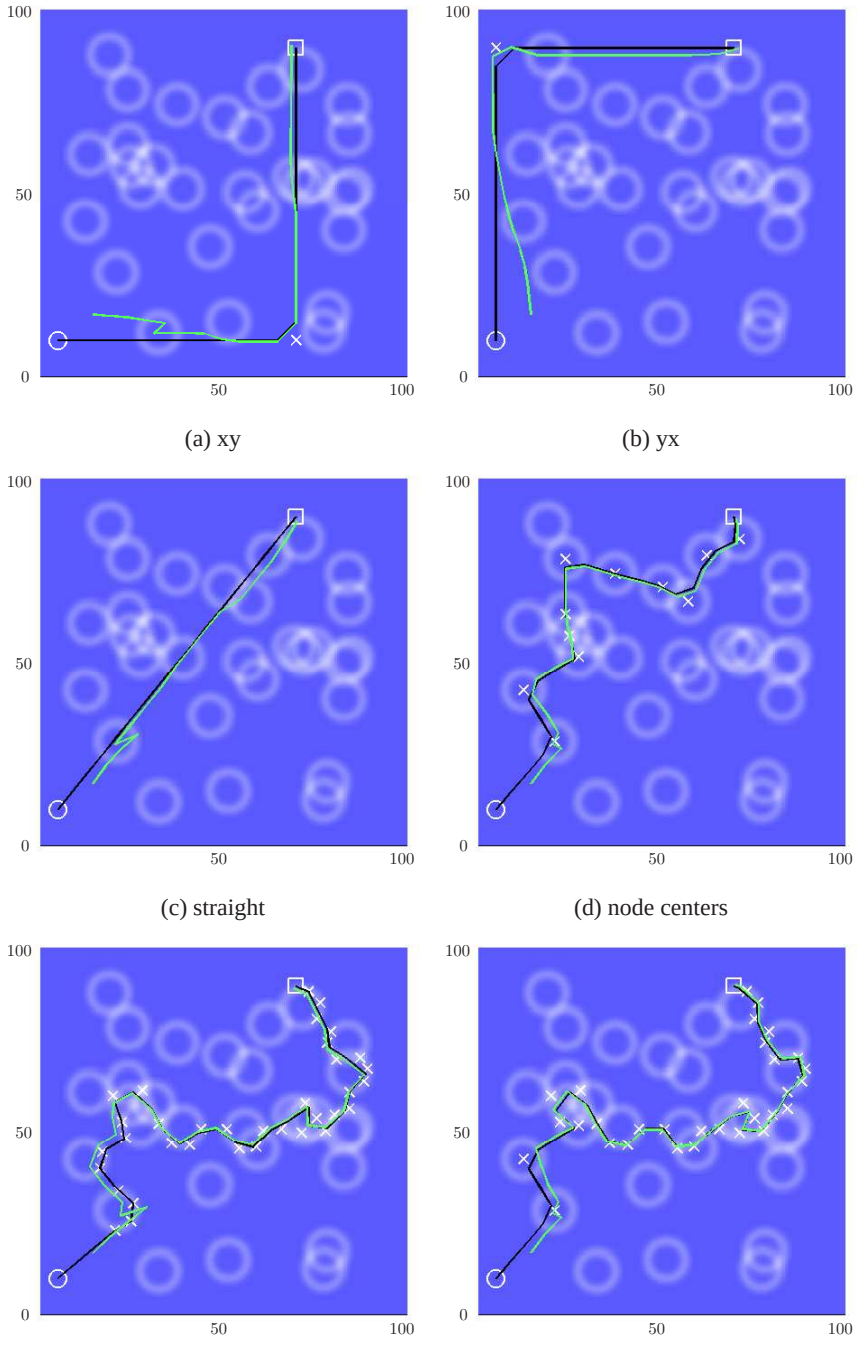
Path Type	Estimation Error (m)			Estimation Entropy		
	Min	Mean	Max	Min	Mean	Max
xy	2.44	2.91	8.22	4.42	4.54	6.05
yx	5.08	5.20	5.32	6.29	6.30	6.51
straight	3.54	3.99	7.29	5.24	5.34	5.92
node centers	1.93	2.15	3.99	3.65	3.76	4.55
manual	2.16	2.32	2.86	3.31	3.38	3.75
hybrid	1.52	1.69	2.66	3.10	3.21	3.55

Table 6.6: Estimation error and entropy for each path scheme across all trials during the localization simulations with improved models.

support. This property also provides the sense that the high information regions have smaller width than before, even though the priors used similar diameter values.

Once the information map is computed, the localization simulation commences. For each path scheme, 100 trials are ran, and various statistics are recorded . Each path scheme, typical estimated path, and information map is shown in Figure 6.7. The 3 basic paths schemes (xy, yx, and straight) use the same paths as before (Section 6.2), but the estimation algorithm clearly has more difficulty in converging to the true path. In particular, the typical estimated paths do not converge to and track the actual path like before. The 3 intelligent schemes typically do much better at converging to the actual path. Notice, unlike before, the node centers and manual path route through different regions of the field. In particular, the node centers path stays to the left of the field and goes through one region with particularly high information. The manual path, on the other hand, goes through this region, then heads to the right and goes through another high information region. Finally, notice that the hybrid path is a combination of the previous 2 routes: initially, it follows the node centers route; then, it follows the manual route.

A summary of the estimation statistics, shown in Table 6.6, looks similar to the previous set of simulations. The intelligent path planners have proven themselves capable of tackling the new system models equally well. The amount the mean estimation error changes from the basic schemes to the intelligent schemes, however, is not as drastic as before. In general, all the path schemes have a more difficult time reducing the localization error. Additionally, the manual path has slightly higher mean estimation error than the node centers path, but lower estimation entropy. This occurs due to the phenomenon mentioned in Section 6.2: the information map is tuned for a small prior, the system starts with a large prior, and the timing and location of initial detections dramatically effects the initial convergence of the estimation error. These properties are the inspiration for the hybrid



(a) xy (b) yx

(c) straight (d) node centers

(e) manual (f) hybrid

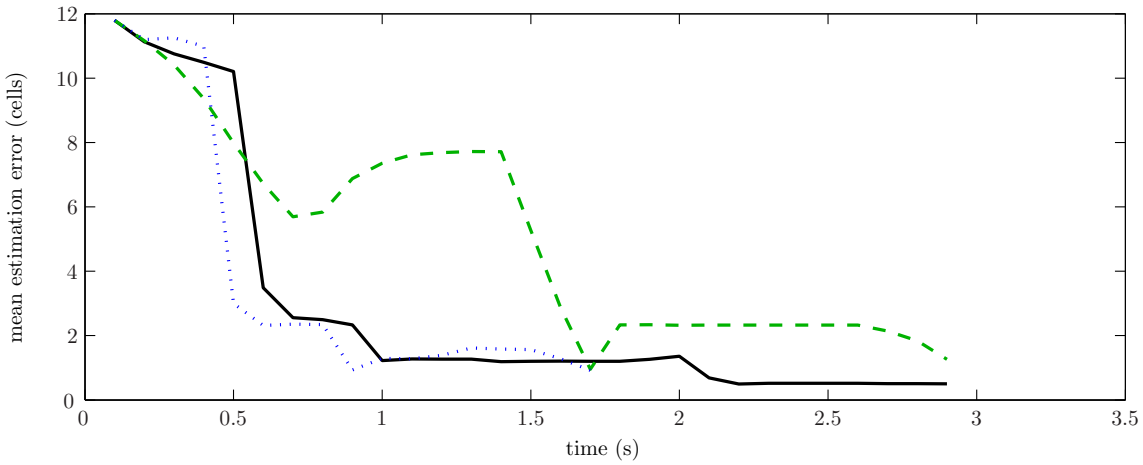
Figure 6.7: Basic simulation overview plots from the localization simulations with improved models. Depicted is the robot starting (white circle) and ending (white square) positions, the robot path (solid black line), and the typical estimated robot path (solid light green line) superimposed on the information map. Each figure is captioned with the path scheme used.

path: use the node centers path to quickly reduce the initial estimation error, then use the manual path to maintain a reduced estimation error. Although, we have yet to discuss the development of the hybrid path, we see from Table 6.6 that the hybrid path achieves 21.40% less estimation error and 5.03% less entropy than the next best paths, respectively. This supports the notion that node centers is good at initializing the estimator and reducing the error to the size of a detection region; whereas, the information based path is good at further reducing and maintaining low error.

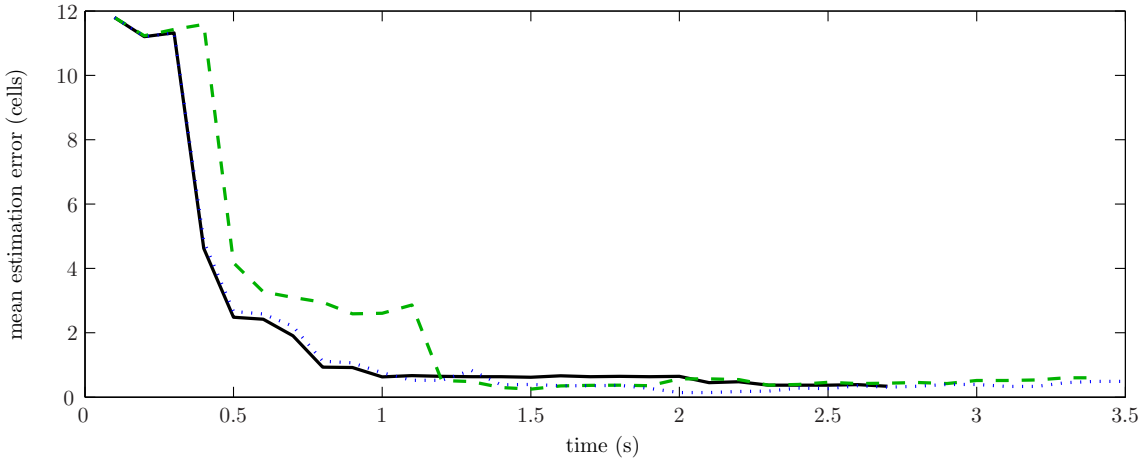
A closer look at the estimation error traces, shown in Figure 6.8, shows more precisely how the estimation error decreases over time. Notice that the manual path lags behind the node centers and hybrid path schemes in reducing the estimation error. The initial sharp drop-off on all the schemes is due to when they (on average) receive their first detection. Since the manual path tends to skirt the detection regions, it takes longer to receive its first detections. The node centers and hybrid scheme are as quick as any scheme at receiving their first detection by driving the robot right to a sensor. Once the manual path, however, reduces the estimation error to levels similar to those of the other 2 intelligent schemes, it performs equally well at keeping the error low. In a later section, more noisy simulations will demonstrate that the manual path actually does better at maintaining low error than the other schemes.

A look at the estimation entropy traces, shown in Figure 6.9, reveals further distinctions amongst the path schemes. While neither node centers nor manual stand out as being the quickest to reduce entropy, it is clear that the manual path converges to a lower entropy value. After t=1.2 seconds, the average entropy of the manual path is about 18.5% less than that of the node centers, providing evidence that the manual path tends to produce a more confident estimate. Again, here we see the hybrid path provides the best of both worlds. Not only does it out perform node center's estimation error, but it keeps pace with and outperforms all the schemes in turns of entropy reduction.

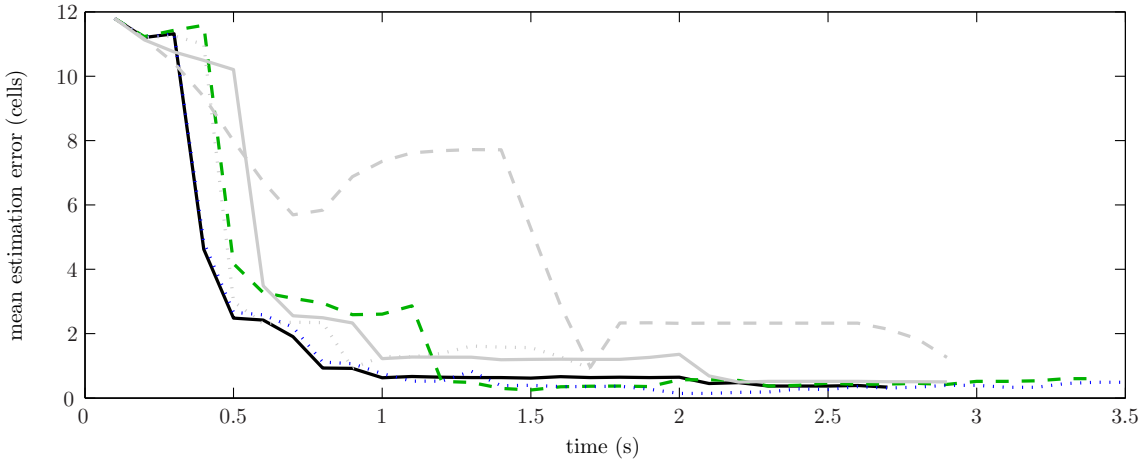
Observing the entropy traces, we can more adequately address how the hybrid path is chosen. A entropy threshold value is chosen for switching between the node centers and manual paths. Since, the manual path exploits the information map, and the information map is tuned to a system with a truncated normal proprioceptive prior with $\sigma_d^2 = 2.25$ cells, the threshold should be set to switch to the manual path when the actual prior is similar to this distribution. The entropy of this distribution is 4.36, hence, the hybrid path switches from node centers once it (on average) achieves an entropy value of this or lower. The node centers path achieves this by the 3rd way point (it achieves entropy of 3.15). So the hybrid path follows the first 3 way points of node centers, then switches to the manual path. In practice, this switching mechanism does not have access to



(a) xy (solid black), yx (dashed green), straight (dotted blue)

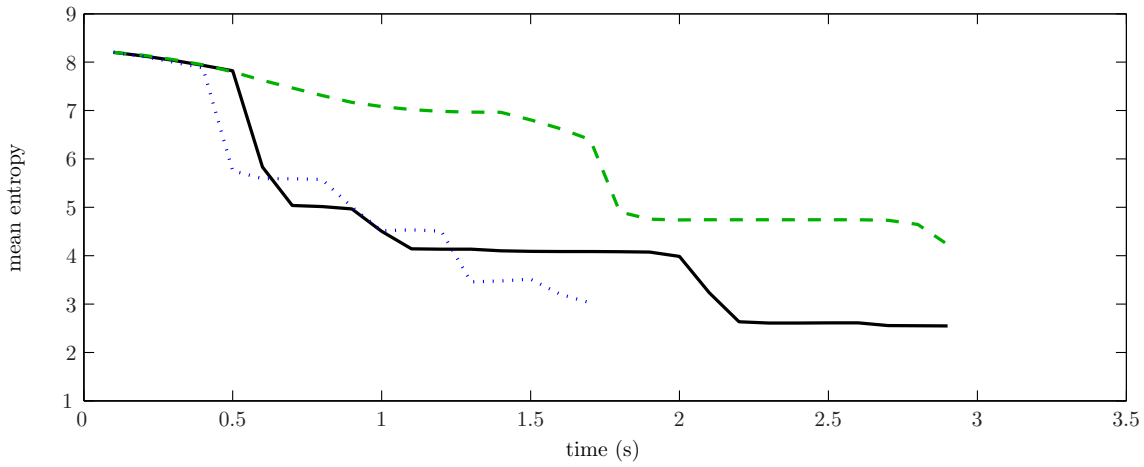


(b) node centers (solid black), manual (dashed green), hybrid (dotted blue)

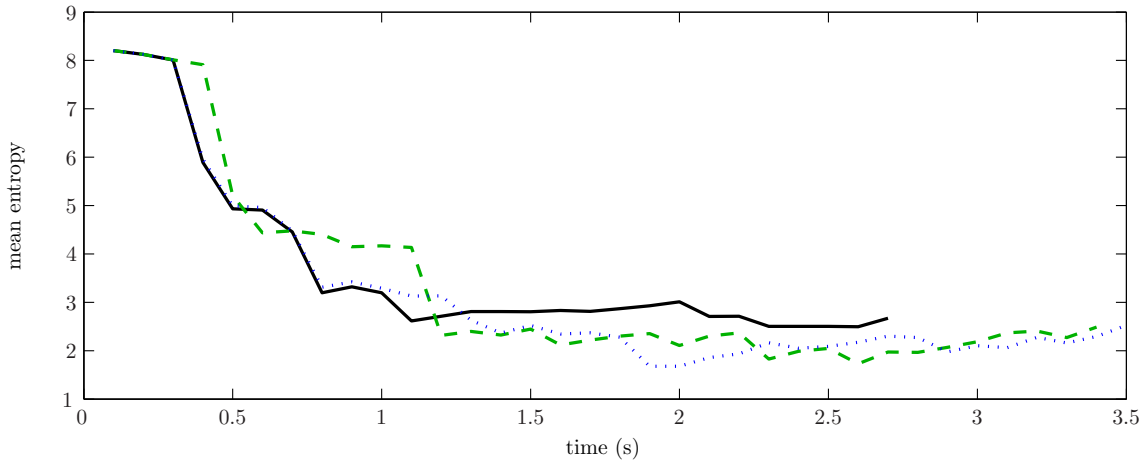


(c) all path schemes

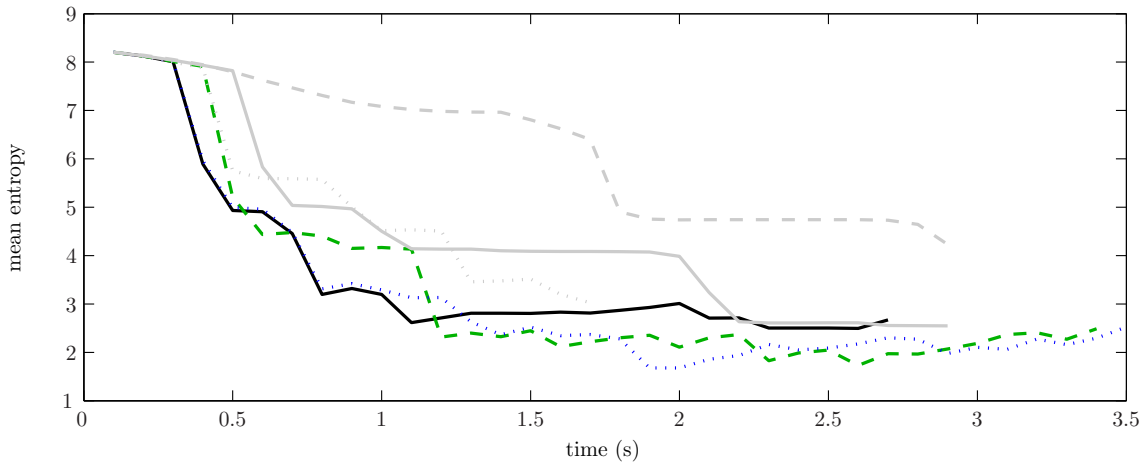
Figure 6.8: Mean estimation error versus time during the localization simulations with improved models. Captions denote the path scheme displayed.



(a) xy (solid black), yx (dashed green), straight (dotted blue)



(b) node centers (solid black), manual (dashed green), hybrid (dotted blue)



(c) all path schemes

Figure 6.9: Mean estimation entropy versus time during the localization simulations with improved models. Captions denote the path scheme displayed.

Path Type	Mean Time to Converge (s)			
	5.66m (8 cells)	2.83m (4 cells)	1.41m (2 cells)	0.71m (1 cell)
xy	0.56	0.69	—	—
yx	1.40	1.61	—	—
straight	0.43	—	—	—
node centers	0.32	0.44	0.87	—
manual	0.41	0.83	1.11	—
hybrid	0.33	0.49	0.79	—

Table 6.7: Estimation convergence times during the localization simulations with improved models.

Path Type	Length (m)	$t_p(s)$	$\bar{n_d}$	$\bar{n_d}$/100 m
xy	145.00	2.80	7.97	5.50
yx	145.00	2.80	1.85	1.28
straight	103.08	1.60	5.34	5.18
node centers	151.03	2.60	19.16	12.69
manual	216.58	3.30	24.36	11.25
hybrid	219.97	3.40	20.49	9.31

Table 6.8: Path information for each path scheme during the localization simulations with improved models.

several prior runs of the node centers scheme, and, hence, the switching would be done based on the run-time value of entropy (with some hysteresis).

For completeness, Table 6.7 shows the estimation error convergence times for the path schemes. This table reiterates that the basic path schemes provide poor and, on average, unreliable convergence. The intelligent schemes are all capable of converging to within 2 grid cells accuracy, with the hybrid path providing the quickest convergence time by 9.20% over the next fastest scheme. Additionally, notice that the hybrid scheme sacrifices a minuscule amount of convergence time initially to gain more later.

The final statistics collected during the simulations, related to path information and detections, is compiled in Table 6.8. This table provides a comparison of the paths based on length, travel time, and number of detections. The manual path lives up to expectations by requiring fewer detections per 100 meters than the node centers path; this is achieved by riding the edges of detection. Even though the hybrid path is the longest path, it only requires a modest 9.31 detections for every 100m of travel. Apparently, the hybrid path has provided the best estimation performance at the lowest bandwidth utilization.

The goal of this work has been to learn how to adapt the well-performing manual scheme for simple systems (as shown in Section 6.2) to a scheme for real systems. In particular, care must be taken to use the information maps at the correct time. We observed that the node centers path works well for quick initialization of a system and the information based path worked well for the fine-tuning and continual maintenance of the estimation system. This leads us to conjecture that a more complete information map path planning system may make use of several information maps, each one tuned to a different prior entropy value. Such a system could switch amongst the information maps as the online entropy values change. This behavior was approximated by the hybrid scheme and was shown to perform well.

To more adequately explore the performance of information based systems, we need to develop and test a closed-loop system with more significant noise levels. Creation of a closed-loop system requires the development of automatic path planners that can rival the manually chosen paths. Looking forward enlightens our previous choices. The reason we have been relying on information maps with relatively small priors is because once a closed-loop system is developed, the entropy will, on average, be low and will stay low due to the closed-loop information map based path planner. These notions are explored in the next chapter.

Chapter 7

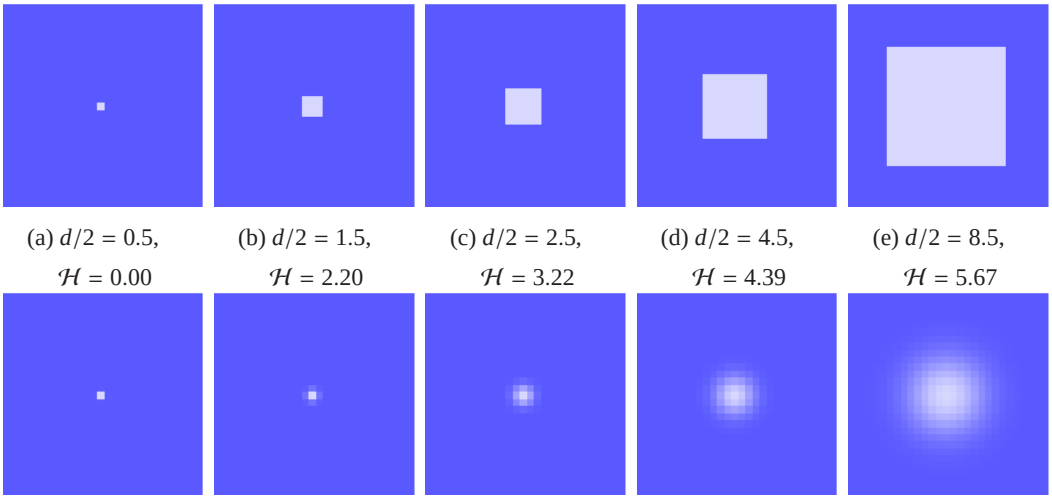
Information Maps and Path Planning

Previous sections have investigated the computation of information maps and their utility in choosing useful paths for navigation. This section builds on this work and develops several techniques for automatic path planning using node information and information maps. First, the semantics of entropy and information and how they can be interpreted as a cost is discussed. Then, using the cost functions, automatic techniques for determining paths are discussed along with their respective time complexity. Finally, closed-loop localization simulations are performed for multiple way-points on multiple maps, and the results are compared. By the close of this section, a complete and fully functional sensor network navigation system will have been developed and evaluated on a set of realistic system models.

7.1 Interpreting Entropy

Entropy provides a measure of a distribution's disorder – the larger the entropy, the more scattered a distribution. The information value for a particular cell provides us with a measure of how much a prior's entropy is expected to change after sensor readings. Taken together, the information map provides a method for estimating how much less scattered the robot's estimation is expected to be after sensor readings. However, the mapping from amount of scatter to entropy change is not clear. This section develops the relationship between entropy and information and provides an interpretation of information as a measure of the expected reduction in a distribution's support.

Consider the relationship between entropy and cells covered. This relationship depends on the distribution used as shown in Figure 7.1. Notice, the entropy values for the normal distribution



(a) $d/2 = 0.5$, $\mathcal{H} = 0.00$ (b) $d/2 = 1.5$, $\mathcal{H} = 2.20$ (c) $d/2 = 2.5$, $\mathcal{H} = 3.22$ (d) $d/2 = 4.5$, $\mathcal{H} = 4.39$ (e) $d/2 = 8.5$, $\mathcal{H} = 5.67$

(f) $3\sigma = 0.5$, $\mathcal{H} = 0.34$ (g) $3\sigma = 1.5$, $\mathcal{H} = 1.62$ (h) $3\sigma = 2.5$, $\mathcal{H} = 2.81$ (i) $3\sigma = 4.5$, $\mathcal{H} = 4.13$ (j) $3\sigma = 8.5$, $\mathcal{H} = 5.50$

Figure 7.1: Entropy values for 2 example distributions with increasing sizes on a 29x29 grid with the mean at cell $(15, 15)$. The top plots are of uniform distributions over a d by d square; the bottom plots are of a normal distribution with given standard deviation. All plots list the entropy value $\mathcal{H}$.

are smaller than those for the uniform distribution. This is always the case: the uniform distribution provides an upper bound. A uniform distribution covering n cells, has a probability density of $p = \frac{1}{n}$ for each cell. The entropy of this distribution is $\mathcal{H} = -\sum_n p \log(p) = \log(n)$. Notice, the logarithmic progression of the entropy versus the area from left to right in Figure 7.1. Given an entropy value $\mathcal{H}$ derived from a uniform distribution, the number of cells covered by the distribution is given by $n = e^{\mathcal{H}}$. Furthermore, assuming a uniform distribution with entropy H_0 (covering $n_0 = e^{H_0}$ cells) is reduced to a uniform distribution with entropy H_1 (covering $n_1 = e^{H_1}$ cells), the decrease in support is $\Delta n = n_0 - n_1 = e^{H_0} - e^{H_1}$. For the information map computation, where a prior with constant entropy H_0 is used, it is convenient to rewrite the change in support as

$$\Delta n = e^{H_0} - e^{H_1} = e^{H_0} - e^{H_0 - I} = e^{H_0}(1 - e^{-I}) \tag{7.1}$$

where we have used that $I = H_0 - H_1$. This term is derived for uniform distributions, but can be used to aid us in interpreting information for other distributions too.

Given this relationship between information and support, overall cost or utility can be interpreted in many ways. We present 2 methods. First, the cost κ could be an affine transformation

of the change in support, such as

$$\begin{aligned} \kappa &= c_0 - c_1(\Delta n) && (7.2)\\ &= c_0 - c_1 e^{H_0}(1 - e^{-I}) && (7.3)\\ &= c_0 + c_2(e^{-I} - 1) && (7.4) \end{aligned}$$

where $c_0, c_1, c_2 > 0$ and c_0 is chosen largest enough such that $\kappa > 0$. This representation assigns low costs to large reductions in number of cells involved in the support. Alternatively, the cost κ could be used to represent the increase in support *diameter*, such as

$$\begin{aligned} \kappa &= c_0 - c_1(\sqrt{n_0} - \sqrt{n_1}) && (7.5)\\ &= c_0 - c_1\sqrt{e^{H_0}}(1 - \sqrt{e^{-I}}) && (7.6)\\ &= c_0 + c_2(\sqrt{e^{-I}} - 1) && (7.7) \end{aligned}$$

where $c_0, c_1, c_2 > 0$ and c_0 is chosen largest enough such that $\kappa > 0$. This representation assigns low costs to large reductions in the diameter of the support. Unless stated otherwise, the cost κ always refers to the former affine cost function. The next section uses the cost interpretation to develop path planners.

7.2 Automating Path Planning

Up to this point, a complete closed-loop navigation system sans a path planner has been developed. This section fills in the gap by developing several path planners, some of which have already been used in previous simulations without explanation. An overview of each path planner along with running time analysis and example paths is presented.

The previously used simple path planners (xy, yx, and straight) and obvious and fixed; hence, the following investigation focuses on only the intelligent path planners. Three such planners are discussed: *information grid*, *node centers*, and *information clusters*. The general modus operandi of each of these planners is the same: encode the space $\mathcal{L}$ and any path information into a weighted, directed graph; then, apply a least cost optimization routine to generate a path in $\mathcal{L}$. The difference in each planner is the manner in which the graph is constructed. For the optimization routine, many tools exists [25, 80], such as Dijkstra, A^*, or Bellman-Ford. To avoid path planners that create loops or back-track significantly, the graph edge weights are restricted to positive values. Hence, generalized dynamic programming routines like Bellman-Ford are not required. Addition-

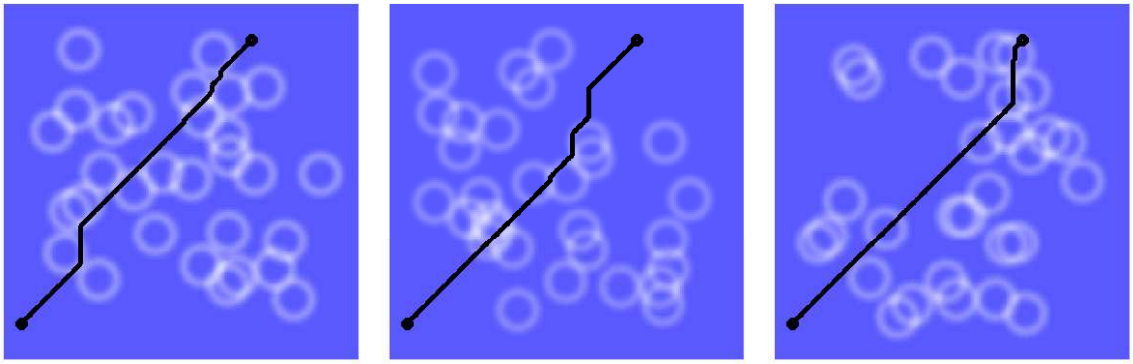


Figure 7.2: Paths generated by the information grid path planner ($c_\kappa = 1$ and $c_d = 10$) for 3 different mote topologies.

ally, since the primary focus is not efficiency, but path effectiveness, heuristic based routines such as A^* are ruled out. For our work, the Dijkstra algorithm is used for finding the lowest cost path.

The Dijkstra algorithm requires a weighted, directed graph G with vertices V and edges E such that, for each edge $e = (u, v) \in E$, the cost of going from u to v is given by $w(u, v)$ defined by the positive cost function $w : E \rightarrow [0, \infty)$. Given starting and ending vertices $u_0, u_f \in V$, the algorithm computes the lowest cost path $P \in V^*$, where V^* is the set of all finite sequences of vertices. For general graphs, the time complexity of this algorithm is $O(n_v^2)$ where $n_v = |V|$.

The most straightforward application of the Dijkstra algorithm is realized by the information grid path planner. This path planner encodes the entire information map in a graph. A similar path planer [78] has been previously developed that additionally incorporates an obstacle map. In our approach, the set of vertices for the graph is defined to be the set of all grid cells $\{\mathcal{G}_{i,j}\}$. Then, each grid cell (vertex) is defined to have an edge connecting to the immediately adjacent grid cells (vertices). Finally, the overall cost to go from vertex $u = g_{i,j}$ to vertex $v = g_{k,l}$ is defined as a linear combination of the ending cell's information cost $\kappa(v) = \kappa(g_{k,l})$ and the 2-norm distance between the cells' centers:

$$w(u, v) = c_\kappa \kappa(v) + c_d \|u - v\|_2 \tag{7.8}$$

where the mixing proportions c_κ, c_d are non-negative. This function assigns far away cells with low information a high cost. Figure 7.2 shows the path generated by the information grid path planner for 3 different node configurations. For each plot, the field spans 100 meters by 100 meters, and the

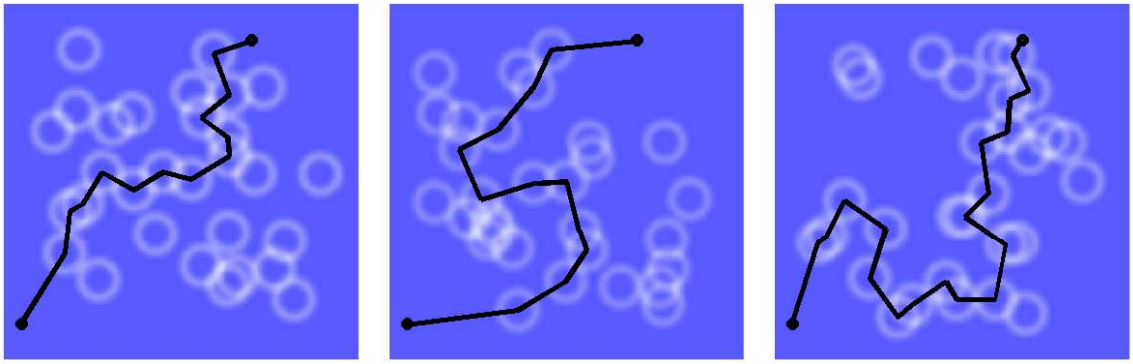


Figure 7.3: Paths generated by the node centers path planner for 3 different mote topologies.

robot starts in the lower left at $\begin{bmatrix} 5 & 10 \end{bmatrix}^T$ and ends in the upper right at $\begin{bmatrix} 70 & 90 \end{bmatrix}^T$. These plots have used $c_\kappa = 1$ and $c_d = 10$. It turns out, for more general values of c_κ and c_d, that this path planner tends to choose fairly straight paths deviating only slightly to encounter higher information regions. Later simulations will demonstrate that the paths generated by this planner tend to reduce travel time to the destination by sacrificing some reduction in estimation error. A disadvantage of this planner is the computation time required. Since it uses all the grid cells as vertices, the number of vertices is $n_v = n_l$, inducing a $O(n_l^2)$ running time. The remaining path planners that are presented require significantly less time.

The node centers path planner embodies the idea that the robot should drive through many detection centers on the way to the destination. This planner uses the set of mote locations and the robot's starting and ending positions as the graph's vertices for the Dijkstra algorithm. It creates edges amongst all mote locations and robot starting and ending positions, and encodes the cost from u to v as the exponential of the 2-norm distance:

$$w(u, v) = e^{\|u-v\|_2} \tag{7.9}$$

The advantage to using an exponential in the cost function is that it encourages the planner to take many small leaps towards the destination (encountering several nodes); whereas, without the exponential, the lowest cost path would be the straight line connecting the starting and destination positions. Three runs of the node centers planner are shown in Figure 7.3. The chosen paths tend to find routes supported by many close detection areas. The planner will prefer a path along many close motes to a path that take large jumps in between groups of motes as shown by the middle plot.

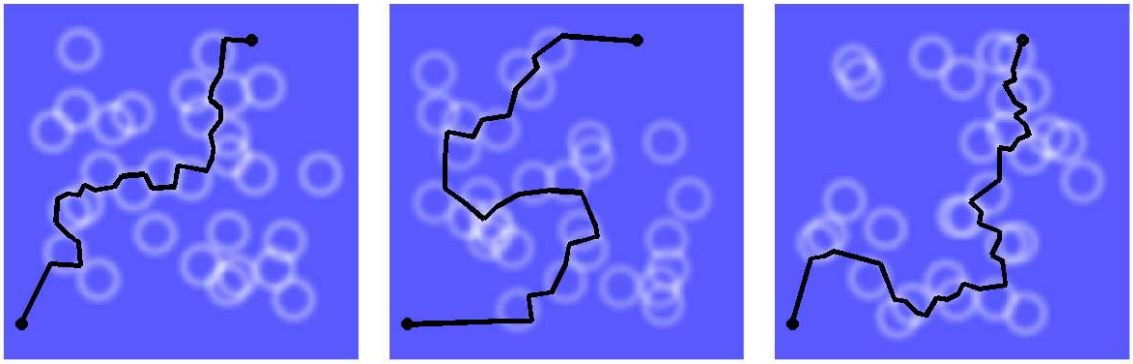


Figure 7.4: Paths generated by the information clusters path planner ($n_v = 100$ and $d_p = 5$) for 3 different mote topologies.

Finally, since this planner only uses the mote locations and 2 robot positions as vertices, the running time is $O(N_s^2)$ where N_s is the number of motes and is typically much less that n_l.

The final path planner, information clusters, combines several of the previous ideas. It attempts to generate a path that prefers many small leaps (like node centers) favoring high information regions (like information grid). This planner uses 2 parameters: the number of candidate vertices n_v and the *padding* d_p around each vertex. Both these parameters effect how candidate vertices for the Dijkstra algorithm are chosen. More specifically, starting with an empty set of vertices, the planner finds the grid cell u with the lowest cost $\kappa(u)$, and adds this grid cell to the set of vertices. Then, all grid cells within d_p (according to the norm on the partition space $\mathcal{L}$) are removed as possible vertex candidates. This procedure now repeats by choosing the next lowest cost vertex, adding it to the set of vertices, and removing neighboring cells. The set of vertices is increased until n_v have been identified. Then, each vertex is connected to every other vertex, and the edge cost of u to v is the same exponential distance used previously:

$$w(u, v) = e^{\|u-v\|_2} \tag{7.10}$$

Examples of this planner for $n_v = 100$ and $d_p = 5$ are shown in Figure 7.4. The paths generated are similar to the node centers paths, but instead of routing through a node center, these paths tend to ride the edge of detection. In fact, the paths produced by this planner are quite similar to the manual paths used in Sections 6.2 and 6.4. Hence, a hybrid planner exploiting node centers and information clusters may provide results superior to each planner alone as demonstrated previously

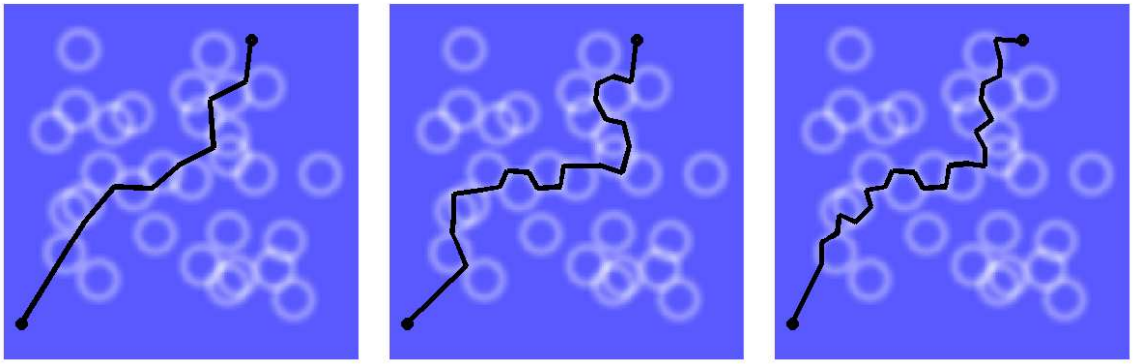


Figure 7.5: Paths generated by the information clusters path planner for a fixed mote topology. The padding is fixed at $d_p = 8$ cells, and n_v varies across the set $\{25, 50, 100\}$ from left to right.

with a hybrid manual and node centers path. Overall, this algorithm has the advantage of choosing intermediate way-points in high information regions and choosing paths with many short jumps between way-points. Finally, the time required for this algorithm is $O(n_v^2)$, and, hence, is set by the n_v parameter which is typically much less than n_l.

One potential disadvantage of the information clusters planner is the need to choose 2 parameters n_v and d_p for a particular deployment. To better understand these parameters, we observe the effect varying them has on the generated paths. In Figure 7.5, the padding has been fixed at $d_p = 8$ cells, and the number of vertices n_v is varied across the set $\{25, 50, 100\}$ from left to right. Increasing the number of vertices provides the planner with more way-points to choose from. Since the padding is fixed, additional way-points are chosen far from previous ones. This tends to make the generated path more jagged: it takes more frequent short jumps. Typically, this generates a path that more precisely follows the high information regions.

In Figure 7.6, the number of candidate vertices is fixed at $n_v = 50$, and the padding d_p is varied across the set $\{2, 4, 8\}$ from left to right. Increasing the padding forces the candidate way-points to be more spread out. As shown in the plots, this tends to generate paths that deviate farther from a straight path than paths with less padding. However, if padding is increased more, the generated paths may adhere closer to straight paths: since candidate way-points are pushed far away from a good route, the path tends to use fewer points creating a straighter line. In the following, we will denote the information clusters planner configured with n_v vertices and d_p padding as information clusters (n_v, d_p).

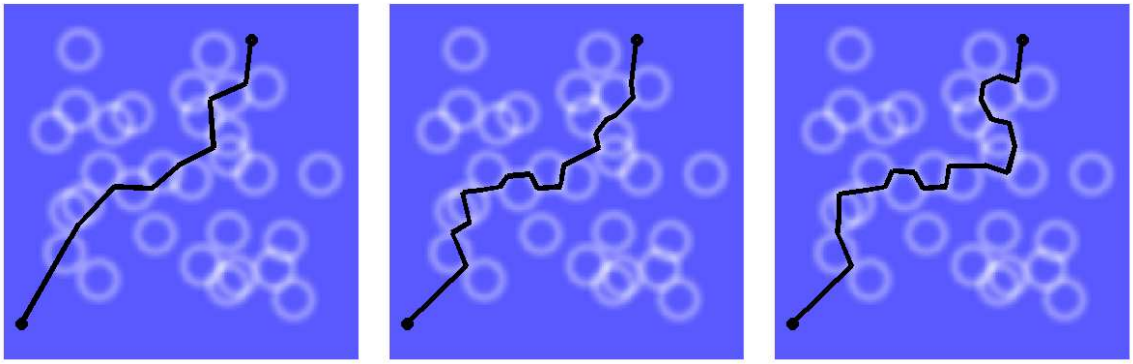


Figure 7.6: Paths generated by the information clusters path planner for a fixed mote topology. The number of candidate vertices is fixed at $n_v = 50$, and the padding d_p varies across the set $\{2, 4, 8\}$ from left to right.

7.3 Closed-Loop Performance

Using the path planners developed in the previous section, this section performs several localization simulations. In particular, the navigation system can now be tested in a closed-loop configuration for several scenarios. This section introduces the localization simulation setup, performs a set of closed-loop simulations, and analyzes the results.

To fully test the navigation system, the localization simulations tests each path planner in a closed-loop configuration for multiple way-points and for multiple sensor network topologies. In particular, the simulator first randomly generates 10 different sensor network topologies. For completeness, these topologies and accompanying information maps are listed in Appendix B. Each topology is generated by uniformly distributing 30 motes on the 100m by 100m field. Then, for each topology, a set of mini-experiments are run. For each mini-experiment, the system configuration and parameters used are listed in Table 7.1. The improved models for the sensor, dynamics, and prior developed in Section 6.3 are utilized. The sensor is configured to have a 90% chance of detecting an object within 5 meters. The information maps are computed using the truncated normal prior with a (single-dimensional) standard deviation of 0.75 m. This prior is truncated at the $3\hat{\sigma}_p$ radius creating a 9 cell by 9 cell square support. The initial estimation of the robot's location is given by a uniform distribution covering a 21 by 21 square of cells. Previous simulations used an initial prior with a much larger diameter (101 cells) since the focus was on estimation convergence. For

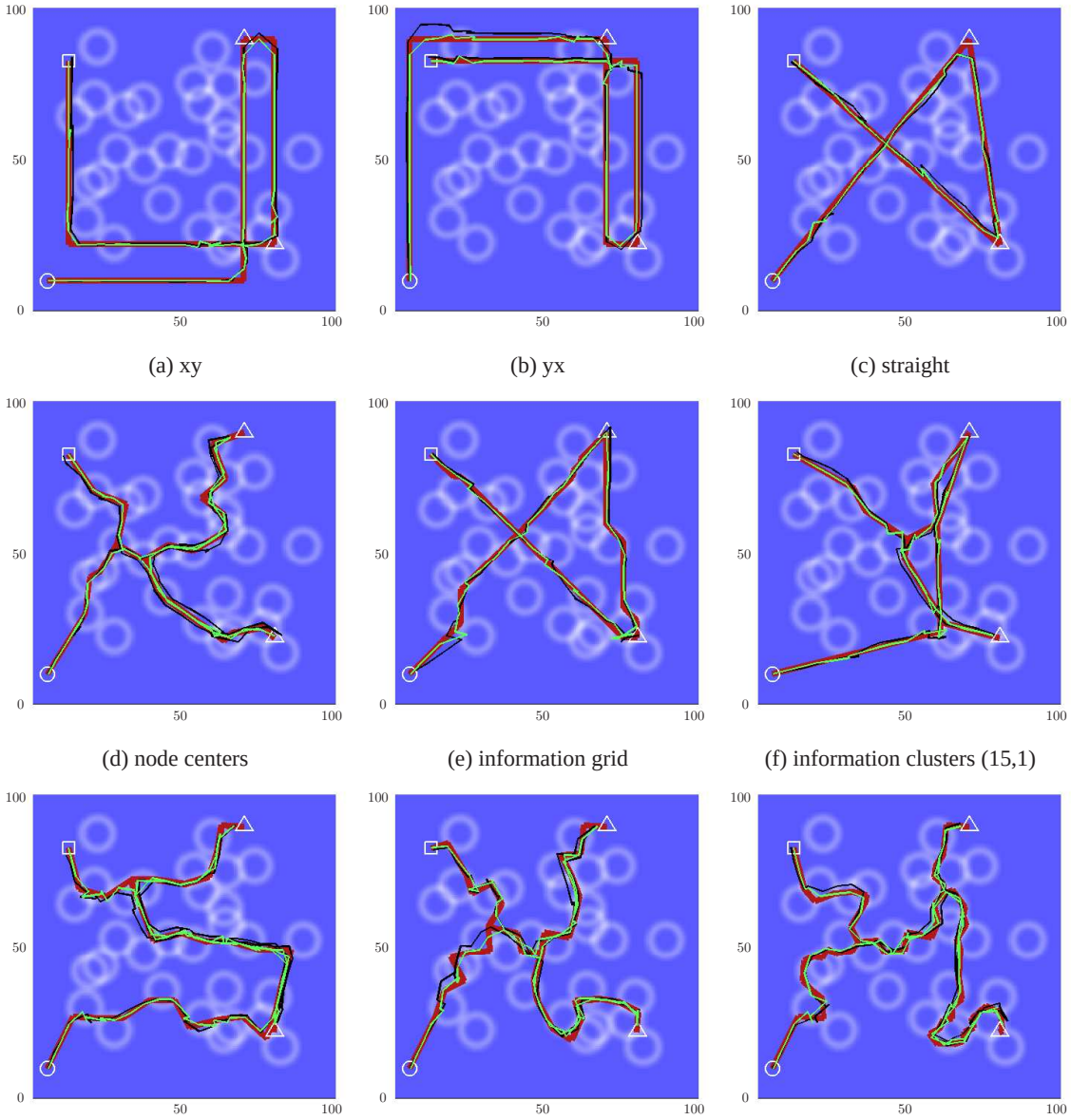
Field	size	100m x 100m
	grid	200 x 200
	sensor distribution	uniform
	N_s	30
Sensor	model	continuous, circular, binary
	r_s	5m
	p_s	0.90
Estimated Prior	prior model	discretized, truncated normal
	$\hat{\sigma}_p$	0.75 m
	P	$3\hat{\sigma}_p^2$ = 2.25 m
Estimator	prior model	discrete, box
	r_p	10 cells
Dynamics	model	continuous, normal noise
	U	5 m
	$\sigma_{d,0}$	0.84 m
	$\sigma_{d,0,min}$	0.05 m
	D	$3\sigma_d$ = 2.51 m
Pose	$l_0, l^1_{goal}, l^2_{goal}, l^3_{goal}$	$\begin{bmatrix}5\\10\end{bmatrix}\begin{bmatrix}70\\90\end{bmatrix}\begin{bmatrix}80\\22\end{bmatrix}\begin{bmatrix}12\\83\end{bmatrix}$
Simulation	trials	100
	path planners	xy, yx, straight, node centers, information grid, information clusters

Table 7.1: System models and parameters used for the closed-loop, multiple map localization simulations.

the current simulations, a smaller prior is used and the focus is shifted to how well the closed-loop system can maintain a tight estimate and closely follow a desired path. The continuous, normal noise model of the dynamics is used with 5 meter control bounds and a noise variance of 0.70 m^2. Again, since these experiments focus on closed-loop performance, the noise variance is increased to 0.70 m^2 from previous experiments that used a minimal noise variance of 0.01 m^2. For each trial, the robot is provided with a starting position at $\begin{bmatrix} 5 & 10 \end{bmatrix}^T$ (in the lower left corner), and given 3 remote way-points to accomplish in order: $\begin{bmatrix} 70 & 90 \end{bmatrix}^T$ (upper, right corner), $\begin{bmatrix} 80 & 22 \end{bmatrix}^T$ (lower, right corner), and $\begin{bmatrix} 12 & 83 \end{bmatrix}^T$ (upper, left corner).

Each mini-experiment compares 9 different path planners. Each planner is depicted in Figure 7.7 with the robot path and estimated path for a typical run of the first mini-experiment. The top 3 plots show the basic path planners tested: xy, yx, and straight. The remaining planners are intelligent planners: a node center, an information grid, and 4 different configurations of information clusters. Clearly, the paths chosen vary heavily depending on the planner and the parameters. An interesting characteristic of the intelligent planners is that sections of previous paths are frequently reused for multiple way-point paths. For example, using the information clusters planner in Figure 7.7i, when the robot navigates from the lower, right corner to the upper, left corner, much of the previous path is reused. This leads us to conjecture that a deployed sensor network implicitly creates several stable, static routes that are consistently useful for localizing a robot. Such routes could prove useful for a variety of reasons: the path planner can reduce the computation required by reusing paths or several unused motes in remote regions could be re-deployed next to a heavy use route.

For each path, resulting statistics are first averaged for each mini-experiment, and then across mini-experiments. Additionally, statistics are collected after the first way-point has been reached providing time for the system to settle. Several key results from the simulations are shown in Table 7.2. Notice that the basic planners are challenged to reduce the mean estimation error below 1.60 meters. Additionally, the most basic intelligent path planner, node centers, improves the estimation performance over the best performing basic planner, straight, by 4.40%. The remaining intelligent planners progressively push the mean estimation error down to the 1.3 meter range and outperform the best simple path planner by anywhere from 12.58% to 19.50%. Next, notice that the node centers planner generates a lengthy path (49.06% longer than the shortest path planner, straight) with a high number of detections per 100 meters (24.34% more detections than the next largest value). Hence, this path planner, which ignores the information map, tends to produce paths



(a) xy (b) yx (c) straight

(d) node centers (e) information grid (f) information clusters (15,1)

(g) information clusters (40,15) (h) information clusters (60,12) (i) information clusters (100,5)

Figure 7.7: Basic simulation overview plots for the multiple way-point, multiple map localization simulations. Depicted is the desired robot path (thick solid red line), the actual robot path (solid black line), and the typical estimated robot path (solid light green line) superimposed on the information map. Each figure is captioned with the path planner and parameters used.

path	mean length (m)	$\bar{n}_d$	$\bar{n}_d$/100 m	mean estimation error (m)	mean entropy
xy	352.00	24.75	7.03	1.67	4.40
yx	352.00	22.31	6.34	1.64	4.40
straight	263.16	23.14	8.79	1.59	4.43
node centers	392.27	66.82	17.06	1.52	4.35
information grid	272.33	26.14	9.60	1.39	4.13
information clusters(15,1)	337.12	42.15	12.52	1.35	4.20
information clusters(60,12)	458.88	63.14	13.72	1.32	4.07
information clusters(40,15)	429.28	56.48	13.16	1.35	4.18
information clusters(100,5)	461.18	60.52	13.13	1.28	3.90

Table 7.2: Simulation statistics collected for each path planner running a closed-loop localization simulation across 10 node topologies with 100 trials each.

which provide little apparent benefit over simple path planners. Other intelligent planners, however, exploiting the information map, tend to generate more suitable paths. For instance, the information grid planner not only generates paths with lengths that rival the straight path planner (only 3.48% longer) and reduces detections per 100 meters to a modest level (only 9.22% more than the best performing basic path planner), but it also reduces the estimation error significantly (anywhere from 12.58% to 16.77% over the basic schemes). Such a planner would likely be preferred to the basic planners in most scenarios.

The information clusters planners, as a group, produce the least estimation error. They also tend to produce some of the longest paths with a high number of detections per meter, though still fewer detections than required by the node centers planner. These planners tend to sacrifice path length and number of detections for reduced estimation error and computation time. Finally, the closed-loop performance provides more support for the information based planners. Due to the accuracy of the estimation, the basic path schemes and the node centers scheme tend to track the desired path to within about 1.03 meters; however, the other planners track the desired path within about 0.85 meters (a 17.48% improvement).

This chapter began with providing a more tangible interpretation of information as cost. Using this interpretation, several path planners were developed that generated varying paths depending on the mote topology. Finally, several sets of closed-loop simulations were performed to demonstrate the benefit of information based planners. In particular, it was demonstrated that 2 varieties of information based planners provided improved performance over all the simple path

schemes and even the node centers planner which exploits node location information. The next chapter provides an overview and summary of our research.

Chapter 8

Conclusions

Our work focused on how to develop control and navigation systems that operate effectively when embedded with real world wireless sensor networks (WSNs). Such networks are composed of many small sensing and computation nodes, or motes. Often times, each mote executes a variety of services such as time synchronization, localization, entity detection, leader election, and message communication. The composite system provides not only a complex distributed sensor, but also an interactive and variable sensing mechanisms. To further our understanding of control system design within such an environment, we began by deploying several real world sensor networks. From this experience, we extracted models of the sensing platform and developed practical techniques for architecting control systems. Next, using robot navigation as our benchmark, we developed a simulator and tested several control system designs. This work identified intelligent path planning as an important method for improving localization and navigation in sensor networks. Using an information topology, we developed several path planning techniques, and demonstrated that these techniques could reduce bandwidth utilization and improve localization accuracy.

One of the initial sensor network and control (SNAC) systems that we deployed is described in Chapter 2. A grid of 121 motes was deployed in an outdoor field for playing mobile robot pursuit-evasion games (PEGs). This chapter began by discussing the general sensing and communication algorithm implemented by the sensor network. These details allowed the reader to understand the complexity of using WSNs as a distributed sensor. Additionally, the control and navigation system utilized by the pursuer robot was presented. This system relied on 2 sensors: the sensor network and an on-board GPS sensor. The GPS sensor was necessary to provide reliable estimates of the pursuer location. Without such estimates, it was unclear how the pursuit algorithm should be designed.

Once deployed, we turned to characterizing the performance and results of this system. In particular, PEG was deemed a successful deployment with the pursuer robot capturing the evader in all runs. However, the performance of the sensor network was less than expected. An analogous deployment indicated that detections from the sensor network arrived every 3.02 seconds, with a latency of 1.75 seconds, and an detection error of 2.42 meters. After the application of several idealistic filtering techniques (zero movement noise Kalman filtering, latency removal, and faulty detection removal), the final estimation error could be reduced to 1.53 meters. We noted that these characteristics are dramatically different from the GPS sensor which provides robot position estimation at 10 Hz with 0.02 meters accuracy. Hence, we identified the primary distinction between a high performance local sensor (GPS) and a large-scale distributed sensor (WSN). In order to facilitate the design of a control system informed solely by a WSN, we identified several characteristics of sensor networks that challenge the performance of traditional control systems. These challenging characteristics were grouped into 3 categories: sensor error, false events, and network induced error.

Using our experience from PEG and the list of challenging characteristics, the next chapter, Chapter 3, developed a general control architecture. In particular, for each challenge, a list of services useful for combating each challenge was developed. In total, several services were identified: predictive control, neighborhood model based estimation, intelligent path planning, multimodal control and networking, sensing coordination, time synchronization, hand shaking, and mote maintenance. Using these services, a unified architecture for both the control system and the motes was developed. We noted that viewing the control architecture as a service based architecture has the advantage of freeing the designer from the traditional control system paradigm. More specifically, a service based architecture allows the designer to create services that interact and vary the sensing system.

After enumerating both a set of challenges and a set of solutions, Chapter 4 began our development of a simulator for further investigating SNAC systems. Using our previous experience and results from the literature, models were developed for the sensor network that account for many properties of real world deployments: sensing noise, sensor saturation, calibration differences, packet collision, radio reception range, multi-hop latency, finite battery lifetime, faulty hardware, and loose time synchronization. More specifically, we developed a sensor model based on the magnetometer used during the PEG deployment. Objects are detected in accordance with a magnetic dipole far field model. Additionally, the operational status of the sensor hardware was modeled with a state machine permitting several states: normal, saturation, noisy, unresponsive, and unpredictable. Next, we developed a communication layer model that operates in 2 states: broadcast

and multi-hop. The broadcast communication model probabilistically accounted for MAC delays, packet collision, and reception ranges. The multi-hop communication model accounted for packet loss and cumulative packet latency. Next, we developed a mote platform model accounting for a mote's finite lifetime and individual time synchronization. Finally, we modeled the detection routine used during the PEG trials. In addition to our WSN models, we also developed a robot model based on car like kinematics.

After developing our system models, we instantiated control and estimation services from our unified control architecture. For the controller, 2 distinct way-point navigation services were developed: feedback and predictive control. For a basic estimation service, an extended Kalman filter was designed. Two additional augmentations to the estimator were developed: network latency compensation and faulty node filtering. Using the aforementioned models and the newly developed control and estimation services, several simulations were ran. First, we ran a set of simulations to compare the performance of the control and estimation services. These simulations reinforced the difficulty of applying basic feedback control to a SNAC system. Additionally, we demonstrated that our estimation and control architecture was able to achieve the best navigation results: the goal was more frequently achieved and the overall estimation error was less. Hence, our control architecture utilizing model predictive control and an estimation system tailored to the sensor network was more effective for way-point navigation than traditional control techniques. Next, to understand the relationship between a robot's path and the localization accuracy, we ran several simulations with varying robot paths. In particular, some paths were chosen to lead the robot through areas densely covered in sensors with the intention to decrease the estimation error. These simulations revealed the complex nature of navigation in sensor networks. Our results indicated that path planning can significantly effect the number of detections, the detection error, and the overall estimation error. However, it was not clear how a path should be planned to consistently improve estimation accuracy. We observed that paths through densely covered areas produced both more and less detection error than our baseline, straight path. Additionally, such paths consistently decreased the position estimation error, but not the orientation estimation error.

Intrigued by these results, in Chapter 5, we began to formalize the relationship between path and localization accuracy. We began by reviewing Markov localization and information. Next, we formalized the computation of an information map, with particular attention being paid to the time complexity. For situations where the computation of the sensor model and prior model require constant time, we showed that the information map computation has a time complexity of $O(n_l^2 n_y)$ where n_l is the cardinality of the pose space and n_y is the cardinality of the measurement space. Not-

ing that this time complexity can be limiting in practice, we reviewed an approximate information map computation algorithm developed by Roy et al [78]. This algorithm was shown to reduce the time complexity by computing across a reduced pose and measurement space. Next, we developed a sensor model and a robot prior model and computed several information maps. In particular, we introduced the discrete binary sensor model and the discrete box prior model. Using these models and a uniformly deployed sensor network, we varied the size of the prior and computed several information maps. We discussed the time required to compute these maps on modern hardware and discussed how the information map could be updated in real time to compensate for sensor nodes coming on-line or going off-line. Additionally, we noted that regions both far from sensing areas and close to node locations were low in information; whereas, regions close to the edge of detection were high in information. We noted that the information topology explains why a path planner developed to simply drive in densely covered regions or close to nodes may fail to provide the best localization results.

Verifying the error reducing ability of information based paths with simulations was carried out in Chapter 6. First, Markov localization was formalized as an algorithm. We noted the time complexity for this computation: $O(n_l^2 t_l + n_l t_s)$ where t_l and t_s are the times required to compute the transition distribution and the sensor model, respectively. Next, we introduced the basic discrete adjacent cell dynamics for robot movement and, using these dynamics, ran several localization simulations for a set of fixed paths. In particular, we compared 5 paths: 3 basic paths (xy, yx, straight), a node by node path (node centers), and a manually chosen information based path (manual). It was demonstrated that the information based path achieves the lowest mean estimation error and mean entropy along its path, while requiring only a modest number of detections. Furthermore, we demonstrated that the node centers path achieves the next lowest error, but requires 210.00% more detections. Hence, it was shown that the simple and intuitive path planning scheme, to go node by node, is far from optimal.

Next, the models used for the localization simulations were replaced with more realistic models. First, the robot dynamics were allowed to become continuous with normally distributed noise that varies with the control magnitude. This model was then systematically tessellated for use with Markov localization. The resulting transition function was a complex function that incurs high computation costs. To address this, the transition function was relaxed to consider only the most likely transitions, and a new update computation was used in place of the standard Markov movement update. It was shown that the new computation reduces the time complexity of the movement update from $O(n_l^2)$ to something less than $O(n_l)$. Next, the robot prior was relaxed to be continuous

and normally distributed. This model was easily tessellated for use in the simulations. Finally, the sensor model was relaxed to have a circular detection region. Again, during the tessellation of this model, a simple approximation was made for the sake of computation time. With the new models and their computationally efficient counterparts, we again ran several localization simulations using the same path schemes as before (xy, yx, straight, node centers, manual) with the addition of one: a hybrid scheme. The hybrid path scheme was developed to use both the node centers path and the manual path in order to quickly reduce and, henceforth, maintain a low estimation error. This scheme was shown to outperform the other schemes.

With verification that information based paths perform well for realistic models, we then moved on to developing automatic path planners in Chapter 7. In this chapter, we presented a brief introduction to entropy, information, and interpretation of information maps. Using this background, 3 automatic path planners were developed: node centers, information grid, and information clusters. The node centers planner embodied the concept of having a robot navigate node by node to the destination. Hence, this planner made use of the node topology. The information grid and information clusters planners additionally incorporated knowledge of the information topology. In particular, the information clusters planner was designed to mimic the manual paths previous used. Next, all the path planners and various configurations of them were tested using a closed-loop, multiple way-point simulation. It was shown that the node centers path planner has performance similar to that of the simple path planners (xy, yx, straight). Whereas, the performance of information based planners outperformed the other planners by providing reduced localization error and path deviation. We observed that, the information grid path planner provided a short path to the destination and low bandwidth utilization and the information clusters planner provided the best localization accuracy and path following abilities.

Overall, our work provided insight into real world SNAC system deployments with an emphasis on way-point navigation. We deployed one of the first and largest SNAC systems. We identified the challenges for such systems and developed a control and estimation solution in the form of a service based architecture. We developed a practical WSN model based on real world experience and previous results from the literature. We combined our architecture and our WSN model into an application level simulator suitable for studying control problems. We demonstrated the benefits of several of the services from our architecture. Then, with a focus on the intelligent path planning service, we generated information based path planners and demonstrated their ability to reduce localization error and path deviation.

Bibliography

[1] M. Aigner and M. Fromme. A game of cops and robbers. In *Discrete Applied Mathematics*, number 8, pages 1–12, 1984.

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

[3] Anish Arora, Rajiv Ramnath, Emre Ertin, Prasun Sinha, Sandip Bapat, Vinayak Naik, Vinod Kulathumani, Hongwei Zhang, Hui Cao, Mukundan Sridharan, Santosh Kumar, Nick Seddon, Chris Anderson, Ted Herman, Nishank Trivedi, Chen Zhang, Mikhail Nesterenko, Romil Shah, Sandeep Kulkarni, Mahesh Aramugam, Limin Wang, Mohamed Gouda, Young ri Choi, David Culler, Prabal Dutta, Cory Sharp, Gilman Tolle, Mike Grimmer, Bill Ferriera, and Ken Parker. Exscal: Elements of an extreme scale wireless sensor network. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2005.

[4] Javed Aslam, Zack Butler, Florin Constantin, Valentino Crespi, George Cybenko, and Daniela Rus. Tracking a moving object with a binary sensor network. In *ACM Conference on Embedded Networked Sensor Systems*, November 2003.

[5] David J. Austin and Patric Jensfelt. Using multiple gaussian hypotheses to represent probability distributions for mobile robot localization. In *International Conference on Robotics & Automation*, April 2000.

[6] Y. Bar-Shalom and T. Fortmann. *Tracking and Data Association*. Academic Press, 1988.

[7] M. Batalin and G. Sukhatme. Efficient exploration without localization. In *IEEE International Conference on Robotics and Automation*, 2003.

[8] M. Batalin and G. Sukhatme. Coverage, exploration, and deployment by a mobile robot and communication network. In *Telecommunication Systems*, 2004.

[9] M. Batalin, G. Sukhatme, and M. Hattig. Mobile robot navigation using a sensor network. In *IEEE International Conference on Robotics and Automation*, 2003.

[10] Michael Beetz, Wolfram Burgard, Dieter Fox, and Armin Cremers. Integrating active localization into high-level control systems. *Robotics and Autonomous Systems*, 23:205–220, 1998.

[11] S. Bergbreiter and K.S.J. Pister. Cotsbots: An off-the-shelf platform for distributed robotics. In *International Conference on Intelligent Robots and Systems 2003*, October 2003.

[12] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCamme, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. In *IEEE Computer*, May 2000.

[13] Alex Brooks, Alexei Makarenko, Tobias Kaupp, Stefan Williams, and Hugh Durrant-Whyte. Implementation of an indoor active sensor network. In *International Symposium on Experimental Robotics*, 2004.

[14] R. A. Brooks. A robust layered control system for a mobile robot. In *IEEE Transactions on Robotics and Automation*, March 1986.

[15] R. R. Brooks, P. Ramanathan, and A. M. Sayeed. Distributed target classification and tracking in sensor networks. In *Proceedings of the IEEE*, August 2003.

[16] W. Burgard, A. Derr, D. Fox, and A. B. Cremers. Integrating global position estimation and position tracking for mobile robots: The dynamic markov localization approach. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 1998.

[17] Z. Butler and D. Rus. Event-based motion control for mobile sensor networks. In *IEEE Pervasive Computing*, volume 2, pages 34–43, October–November 2003.

[18] V. Bychkosvkiy, T. Schoellhammer, and D. Estrin. Control and actuation in data-centric wireless sensor networks. In *AAAI Spring Symposium on Intelligent, Distributed, and Embedded Systems*, 2002.

[19] A. Cerpa, J. L. Wong, L. Kuang, M. Potkonjak, and D. Estrin. Statistical model of lossy links in wireless sensor networks. Technical Report 41, CENS Technical Report, April 2004.

[20] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat monitoring: Application driver for wireless communications technology. In

ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, April 2001.

[21] Phoebus Chen, Songhwai Oh, Michael Manzo, Bruno Sinopoli, Cory Sharp, Kamin Whitehouse, Gilman Tolle, Jaein Jeong, Prabal Dutta, Jonathan Hui, Shawn Shaffert, Sukun Kim, Jay Taneja, Bonnie Zhu, Tanya Roosta, Mike Howard, David Culler, and Shankar Sastry. Closing the loop in sensor networks. In *IEEE International Conference on Robotics and Automation Video*, 2006.

[22] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Autonomous deployment and repair of a sensor net using an unmanned aerial vehicle. In *IEEE International Conference on Robotics and Automation*, 2004.

[23] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Deployment and connectivity repair of a sensor net with a flying robot. In *Proceedings of the Ninth International Symposium on Experimental Robotics*, June 2004.

[24] P. Corke, R. Peterson, and D. Rus. Networked robots: Flying robot navigation using a sensor net. In *Proceedings International Symposium on Robotics Research*, November 2003.

[25] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[26] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[27] Z. Crisman, E. Curre, C. Kwok, N. Ratliff, L. Tsybert, and D. Fox. Team description: Uwhuskies-02. In *RoboCup-2002: Robot Soccer World Cup VI*. Springer Verlag, 2003.

[28] A. Das, G. Kantor, V. Kumar, G. Pereira, R. Peterson, D. Rus, S. Singh, and J. Spletzer. Distributed search and rescue with robot and sensor teams. In *International Conference on Field and Service Robotics*, July 2003.

[29] M. Demirbas, A. Arora, and M.G. Gouda. A pursuer-evader game for sensor networks. In *Self-Stabilizing Systems 2003*, pages 1–16, 2003.

[30] Jiagen Ding, Sing Yiu Cheung, Chin-Woo Tan, and Pravin Varaiya. Signal processing of sensor node data for vehicle detection. In *International IEEE Conference on Intelligent Transportation Systems*, October 2004.

[31] R. Dorf and R. Bishop. *Modern Control Systems*. Prentice Hall, tenth edition, 2004.

[32] A. Doucet, N. De Freitas, and N.J. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, May 2001.

[33] Prabal Dutta, Mike Grimmer, Anish Arora, Steven Bibyk, and David Culler. Design of a wireless sensor network platform for detecting rare, random, and ephemeral events. In *International Conference on Information Processing in Sensor Networks*, 2005.

[34] J. Elson. *Time synchronization in wireless sensor networks*. PhD thesis, May 2003.

[35] Deborah Estrin, Lewis Girod, Greg Pottie, and Mani Srivastava. Instrumenting the world with wireless sensor networks. In *International Conference on Acoustics, Speech, and Signal Processing*, 2001.

[36] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. In *Journal of Artificial Intelligence Research*, 1999.

[37] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical report, UCLA Computer Science Technical Report UCLA/CSD-TR 02-0013, 2002.

[38] Tia Gao, Dan Greenspan, Matt Welsh, Radford R. Juang, and Alex Alm. Vital signs monitoring and patient tracking over a wireless network. In *International Conference of the IEEE Engineering in Medicine and Biology Society*, September 2005.

[39] I.S. Gradshteyn and I.M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, Inc., fifth edition, 1994.

[40] Lin Gu, Dong Jia, Pascal Vicaire, Ting Yan, Liqian Luo, Ajay Tirumala, Qing Cao, Tian He, John A. Stankovic, Tarek Abdelzaher, and Bruce H. Krogh. Lightweight detection and classification for wireless sensor networks in realistic environments. In *ACM Conference on Embedded Networked Sensor Systems*, 2005.

[41] Jens-Steffen Gutmann. Markov-kalman localization for mobile robots. In *Proceedings of the International Conference on Pattern Recognition*, 2002.

[42] Jens-Steffen Gutmann, Wolfram Burgard, Dieter Fox, and Kurt Konolige. An experimental comparison of localization methods. In *International Conference on Intelligent Robots and Systems*, 1998.

[43] Jens-Steffen Gutmann and Dieter Fox. An experimental comparison of localization methods continued. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[44] J. Hespanha, M. Prandini, and S. Sastry. Probabilistic pursuit-evasion games: A one-step nash approach. In *Proceedings of the 39th IEEE Conference on Decision and Control*, December 2000.

[45] J.P. Hespanha and Maria Prandini. Optimal pursuit under partial information. In *In Proceedings of the 10th Mediterranean Conference on Control and Automation*, July 2002.

[46] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, November 2000.

[47] A. Hoover and B.D. Olsen. Sensor network perception for mobile robotics. In *IEEE Conference on Robotics & Automation*, April 2000.

[48] P. Jensfelt and S. Kristensen. Active global localisation for a mobile robot using multiple hypothesis tracking. In *Workshop on Reasoning with Uncertainty in Robot Navigation*, Aug 1999.

[49] M. I. Jordan. *Learning in Graphical Models (Adaptive Computation and Machine Learning)*. The MIT Press, 1998.

[50] R. E. Kalman. A new approach to linear filtering and prediction problems. In *Transactions of the ASME – Journal of Basic Engineering*, volume 82, pages 35–45, 1960.

[51] H. J. Kim, R. Vidal, D. Shim, O.Shankernia, and S. Sastry. A hierarchical approach to probabilistic pursuit-evasion games with unmanned ground and aerial vehicles. In *Proceedings of the 40th IEEE Conference on Decision and Control*, December 2001.

[52] J. Lee and H. Hashimoto. Controlling mobile robots in distributed intelligent sensor network. In *IEEE Transactions on Industrial Electronics*, volume 50, October 2003.

[53] Scott Lenser and Manuela M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1225–1232, 2000.

[54] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. In *IEEE Transaction on Robotics and Automation*, 1991.

[55] J.J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localisation for an autonomous robot. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems*, pages 1442–1447, 1991.

[56] Q. Li, M. DeRosa, and D. Rus. Distributed algorithms for guiding navigation across a sensor network. In *International Workshop on Information Processing in Sensor Networks*, 2003.

[57] Jie Liu, Patrick Cheung, Leonadas Guibas, and Feng Zhao. A dual-space approach to tracking and sensor management in wireless sensor networks. In *ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[58] Juan Liu, Maurice Chu, Jie Liu, Jim Reich, and Feng Zhao. Distributed state representation for tracking problems in sensor networks. In *International Workshop on Information Processing in Sensor Networks*, 2004.

[59] J. Lundquist, D. Cayan, and M. Dettinger. Meteorology and hydrology in yosemite national park: A sensor network application. In *Information Processing in Sensor Networks*, April 2003.

[60] Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. Tinydb: An acquisitional query processing system for sensor networks. In *ACM Transactions on Database Systems*, 2005.

[61] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Wireless Sensor Networks and Applications*, September 2002.

[62] Alexei Makarenko and Hugh Durrant-Whyte. Decentralized data fusion and control in active sensor networks. In *International Conference on Information Fusion*, 2004.

[63] Alexei A. Makarenko, Stefan B. Williams, Frederic Bourgault, and Hugh F. Durrant-Whyte. An experiment in integrated exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[64] P. Martí, G. Fohler, K. Ramamritham, and J.M. Fuertes. Jitter compensation for real-time control systems. In *IEEE Real-Time Systems Symposium*, December 2001.

[65] Pau Martí, Ricard Villa, Josep M. Fuertes, and Gerhard Fohler. Stability of on-line compensated real-time scheduled control tasks. In *IFAC Conference on New Technologies for Computer Control*, November 2001.

[66] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. In *AI Magazine*, 1988.

[67] S. Narayanaswamy, V. Kawadia, R. Sreenivas, and P. Kumar. Power control in ad-hoc networks: Theory, architecture, algorithm and implementation of the compow protocol. In *European Wireless Conference*, 2002.

[68] I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish an office-navigating robot. In *AI Magazine*, volume 16, pages 53–60, 1995.

[69] Nucleus Network Management System Website. Available on the Internet: `http://www.cs.berkeley.edu/~get/nucleus/`.

[70] P. Ogren, E. Fiorelli, and N.E. Leonard. Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment. In *IEEE Transactions on Automatic Control*, August 2004.

[71] S. Oh, S. Russell, and S. Sastry. Markov chain monte carlo data association for general multiple-target tracking problems. In *IEEE Conference on Decision and Control*, 2004.

[72] T.D. Parsons. Pursuit-evasion in a graph. In *Theory and Application of Graphs*, pages 426–441. Springer-Verlag, 1976.

[73] Hanna Pasula, Stuart Russel, Michael Ostland, and Ya'acov Ritov. Tracking many objects with many sensors. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1999.

[74] R. Peterson and D. Rus. Interacting with a sensor network. In *Australasian Conference on Robotics and Automation*, November 2002.

[75] Gregory Pottie and William Kaiser. Wireless integrated network sensors. In *Communications of the ACM*, volume 43, pages 51–58, May 2000.

[76] D.B. Reid. An algorithm for tracking multiple targets. In *IEEE Transactions on Automatic Control*, volume 24:6, 1979.

[77] Branko Ristic, Sanjeev Arulampalam, and Neil Gordon. *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Artech House, February 2004.

[78] Nicholas Roy, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Coastal navigation mobile robot navigation with uncertainty in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999.

[79] Wilson J. Rugh. *Linear System Theory*. Information and System Sciences Series. Prentice Hall, Upper Saddle River, New Jersey 07458, second edition, 1996.

[80] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2002.

[81] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler. Design and implementation of a sensor network system for vehicle tracking and autonomous interception. In *European Workshop on Wireless Sensor Networks*, 2005.

[82] D. Shim, H.J. Kim, and S. Sastry. Decentralized nonlinear model predictive control of multiple flying robots. In *Proceedings of IEEE Conference on Decision and Control*, 2003.

[83] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor network-based countersniper system. In *ACM Conference on Embedded Networked Sensor Systems*, 2004.

[84] B. Sinopoli, L. Schenato, M. Franceschetti, K. Poolla, M. Jordan, and S. Sastry. Kalman filtering with intermittent observations. In *Proceedings of IEEE Conference on Decision and Control 2004*, December 2004.

[85] Robert Szewczyk, Joseph Polastre, Alan Mainwaring, and David Culler. Lessons from a sensor network expedition. In *European Workshop on Wireless Sensor Networks*, January 2004.

[86] Haruo Takeda, Claudio Facchinetti, and Jean-Claude Latombe. Planning the motions of a mobile robot in a sensory uncertainty field. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1994.

[87] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. In *Artificial Intelligence*, 2000.

[88] TinyOS Website. Available on the Internet: `http://www.tinyos.net/`.

[89] P.F. Tsuchiya. The landmark hierarchy, a new hierarchy for routing in very large networks. In *Special Interest Group on Data Communication*, pages 36–42, 1988.

[90] Pravin Varaiya and P.R. Kumar. *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Information and System Sciences Series. Prentice Hall, Upper Saddle River, New Jersey 07458, 1986.

[91] R. Vidal, S. Rashid, C. Sharp, O. Shakernia, J. Kim, and S. Sastry. Pursuit-evasion games with unmanned ground and aerial vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2001.

[92] Kamin Whitehouse and David Culler. Calibration as parameter estimation in sensor networks. In *ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[93] Kamin Whitehouse, Chris Karlof, Alec Woo, Fred Jiang, and David Culler. The effects of ranging noise on multihop localization: an empirical study. In *The International Conference on Information Processing in Sensor Networks*, 2005.

[94] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: a neighborhood abstraction for sensor networks. In *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services*. ACM Press, June 2004.

[95] M. Yamashita, H. Umemoto, I. Suzuki, and T. Kameda. Searching for mobile intruders in a polygonal region by a group of mobile searchers. In *Algorithmica*, volume 31, pages 208–236, 2001.

[96] W. Ye, R. Vaughan, G. Sukhatme, J. Heidemann, D. Estrin, and M. Mataric. Evaluating control strategies for wireless-networked robots using an integrated robot and network simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2001.

[97] Feng Zhao, Jie Liu, Juan Liu, Leonidas Guibas, and James Reich. Collaborative signal and information processing: An information directed approach. In *Proceedings of the IEEE*, 2003.

Appendix A

Sensor Network and Control System Simulator Parameter Values

This section provides a list of the parameter values used by the models in the sensor network and control system simulator discussed in Chapter 4.

- Physical layout
  - Field dimensions: 20 m by 20 m
  - Number of motes: 100
  - Mote distribution: uniform
- Sensor model
  - Dipole height off ground: $d_m = 2$ m
  - Dipole strength: $M = \frac{2\times10^{-4}\pi d_m^3}{\mu_0} = \frac{10^3 d_m^3}{2}$ A/m allowing $B_0(\rho = 0) = 10^{-4}$ T = 1 G
  - Permeability of free space: $\mu_0 = 4\pi \times 10^{-7}$ Wb/(Am)
  - Saturation threshold: $B_{thres} = 980$ mG
  - Saturation measurement: $B_{sat} = 980$ mG
  - Saturation duration: $\tau_{sat} = 2$ s
  - Normal sensors: $1 - p_n - p_{ur} - p_{up} = 94$ %
    - Bias distribution: $\mathcal{N}(0, 20^2)$ mG
    - Noise variance distribution: $\mathcal{N}(0, 20^2)$ mG

  - Noisy sensors: $p_n = 2$ %
    - Bias distribution: $\mathcal{N}(0, 20^2)$ mG
    - Noise variance distribution: $\mathcal{N}(0, 100^2)$ mG
  - Unpredictable sensors: $p_{up} = 2$ %
    - Measurement distribution: $\mathcal{N}(\mu_{up}, \sigma_{up}^2)$
    - Distribution standard deviation: $\sigma_{up} = 50$ mG
    - Distribution mean: $\mu_{up} = -\sigma_{up}\sqrt{2}\text{erf}^{-1}(2p_{detect} - 1) + B_{thres}$ where $p_{detect} = 0.05$ is the probability of the measurement exceeding the detection threshold.
  - Unresponsive sensors: $p_{ur} = 2$ %
- Communication model
  - Packet drop probability: $p_{drop} = 0.30$
  - Single hop lag distribution: $\mathcal{U}[0, 0.3]$ s
  - Reception probability (d in meters) : $p_{dist}(d) \sim \mathcal{R}(2.414, 1)$
  - Average hop distance: $\hat{d}_{hop} = 2.414$ m
- Mote platform
  - Low energy probability: $p_e = 2$ %
  - Mote expiration time distribution: $\mathcal{U}[0, 30]$ s
  - Mote initial sensing time distribution: $\mathcal{U}[0, 1.5]$ s
- Mote algorithm parameters
  - Reporting threshold: $B_{report} = 136$ mG where the sensor will nominally detect at $2\sqrt{2}$ meters
  - Reporting period: $T_{report} = 0.3$ s
- Car model
  - Wheelbase: $b = 0.15$ m
  - Wheel speed bounds: $v \in [-0.25, 0.25]$ m/s
  - Turning angle bounds: $\phi \in [-20, 20]$ degrees providing a 0.52 m turning radius

  - Wheel speed error variance: $\sigma_v^2 = 0.01$
  - Steering angle error variance: $\sigma_\phi^2 = 0.01$

- Feedback controller parameters
  - Wheel speed bound estimate: $\hat{v}_{max} = 0.3$ m/s
  - Proportional constant: $p_\phi = 1$

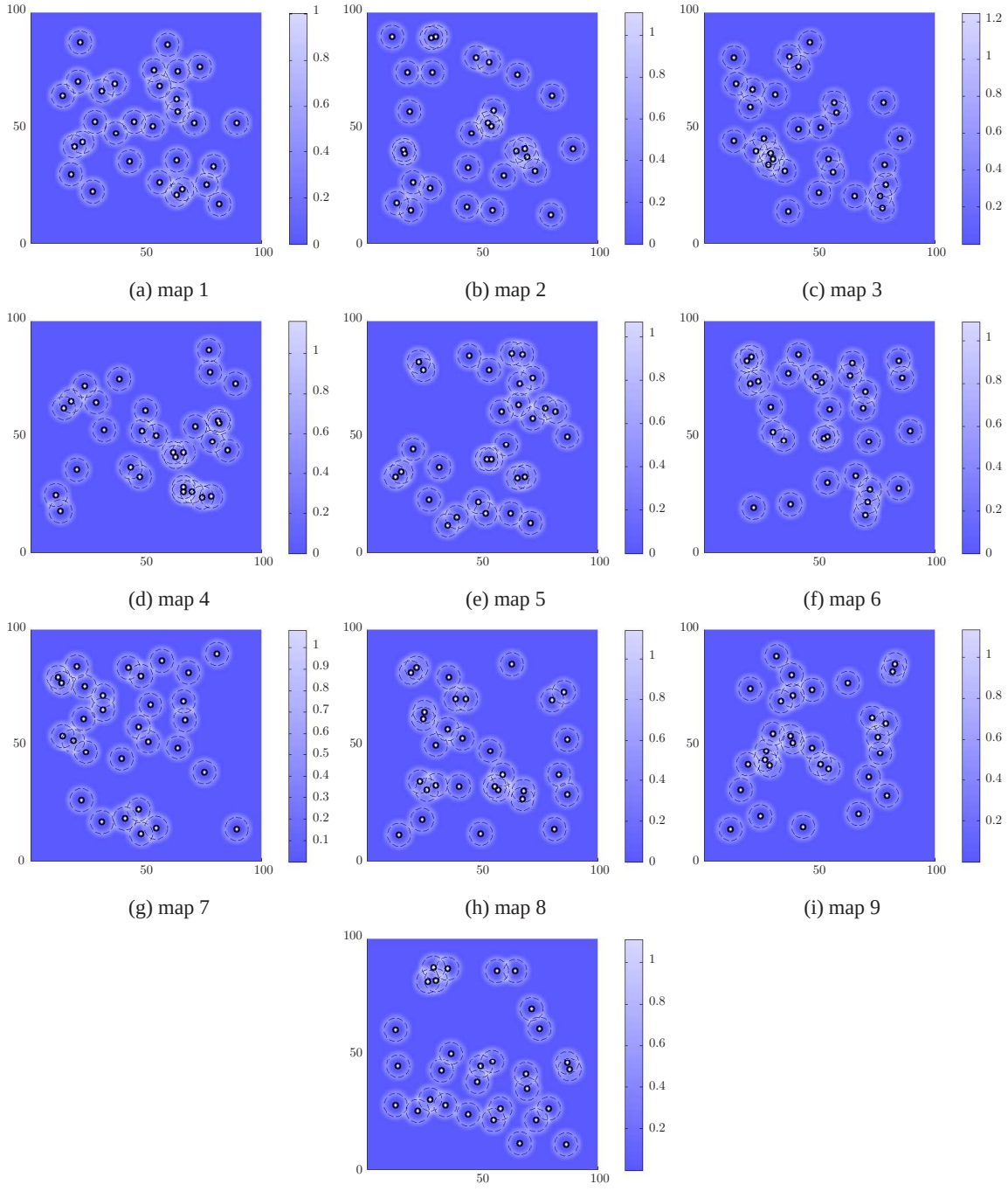
- Estimation parameters
  - $p(n|v) = \begin{cases} 0 & \text{if } n \leq 1\ , \\ 1 & \text{otherwise.} \end{cases}$
  - Expected packet delay: $\tau_d = 1$ s
  - Wheel speed bounds estimate: $[\hat{v}_{min}, \hat{v}_{max}] = [-0.3, 0.3]$ m/s
  - Steering angle bounds estimate: $[\hat{\phi}_{min}, \hat{\phi}_{max}] = [-0.4, 0.4]$ radians

- Simulator parameters
  - Sample time: $T = 0.03$ s
  - Initial state: $\begin{bmatrix} x_0 & \theta_0 \end{bmatrix}^T = \begin{bmatrix} 5 & 5 & 3\pi/2 \end{bmatrix}^T$
  - Initial state estimate: $\begin{bmatrix} \hat{x}_0 & \hat{\theta}_0 \end{bmatrix}^T = \begin{bmatrix} 4 & 4 & \pi/2 \end{bmatrix}^T$
  - Destination position: $x_f = \begin{bmatrix} 10 & 10 \end{bmatrix}^T$

Appendix B

Node Topologies for Closed-Loop Simulations

This section provides a depiction, in Figure B.1, of the node topologies and information maps used during the multiple map, multiple way point, closed-loop simulations of Section 7.3.



(a) map 1 (b) map 2 (c) map 3

(d) map 4 (e) map 5 (f) map 6

(g) map 7 (h) map 8 (i) map 9

(j) map 10

Figure B.1: The 10 sensor network topologies and accompanying information maps used during the closed-loop simulations.