Case Study and Experiments of Control over Sensor Networks

Phoebus Wei-Chih Chen

Acknowledgement

There is a long list of people to thank. See the report for details.

Shankar Sastry, Songhwai Oh, Sarah Bergebreiter, Phil Loarie, Rob Szewczyk, Winthrop Williams, Terence Tong, Rodrigo Fonseca, Prabal Dutta, Jonathan Hui, Jaein Jeong, Sukun Kim, Michael Manzo, Tanya Roosta, Shawn Shaffert, Cory Sharp, Bruno Sinopoli, Jay Taneja, Gilman Tolle, Kamin Whitehouse, Bonnie Zhu, Mike Howard, David Culler, David Shim, Hoam Chung, Jonathan Sprinkle, Rick Groff, Neils Hoven, Matthew A. dAlessio.

Case Study and Experiments of Control over Sensor Networks

by

Phoebus Wei-Chih Chen

B.S. (University of California, Berkeley) 2002

A thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science
in

Engineering - Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Shankar Sastry, Chair
Professor Kristofer Pister

Fall 2005

Case Study and Experiments of Control over Sensor Networks

Abstract

Case Study and Experiments of Control over Sensor Networks

by

Phoebus Wei-Chih Chen

Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Shankar Sastry, Chair

This report studies some examples of how to integrate control systems with sensor networks. In particular, we study how a sensor network can be used as the observer in a control feedback loop for Pursuit-Evasion Games (PEG). Some of the main challenges to using sensor networks for control are properly modelling the lossy communication channel and nonuniform sensor coverage of a sensor network, deriving performance parameters/metrics from the model such as latency and transmission error rate, and devising a control law which effectively utilizes these performance parameters.

We approach this problem by studying some models and simulations of sensor networks and by building testbeds and running experiments. First, we simulate the use of a network connectivity parameter in an optimal path planning controller for pursuit of a target traversing through a sensor network. Then, we look at an example of bounding the performance of a sensor network control application — finding a *probabilistic barrier* solution to a single pursuer/single evader pursuit-evasion game. From the implementation side, we discuss the design of an indoor and and outdoor sensor network testbed for multiple-target tracking, a piece of the estimator in the control loop for multiple-evader pursuit-evasion games. Finally, we discuss some preliminary experimental results of multiple-target tracking on these sensor network testbeds.

Contents

List of Figures

List of Tables

Acknowledgements

First off, thanks to my advisor Professor Shankar Sastry for supporting my research and giving me the freedom to explore my interests. I would also like to thank my colleague Songhwai Oh who has given me much advice on research and advice throughout the writing of this thesis.

The hardware setup of the indoor robot testbed in 330 Cory had much guidance from Cory Sharp and Rob Szewczyk, and some circuit debugging expertise from Winthrop Williams. On the software side, Kamin Whitehouse provided advice on integrating matlab, Terence Tong helped with setting up the ethernet backbone, Rodrigo Fonseca provided a collection of scripts useful for configuring/managing the network, and Cory Sharp helped answer many of my questions on TinyOS. The actual testbed construction involved Bonnie Zhu, Songhwai Oh, Tanya Roosta, Michael Manzo, and Bruno Sinopoli. And of course, Sarah Bergebreiter provided much help with the COTSBOTs used in the testbed. Special thanks also goes to Phil Loarie for helping acquire hardware and computer resources for various parts of the project.

The setup of the outdoor testbed at the Richmond Field Station for the Multiple Target Tracking experiment could not have happened without the work and support of the students under David Culler and Shankar Sastry. The ways in which they contributed are too numerous to list here. The team of students who helped in the effort for the NEST Final Experiment are: Prabal Dutta, Jonathan Hui, Jaein Jeong, Sukun Kim, Michael Manzo, Songhwai Oh, Tanya Roosta, Shawn Shaffert, Cory Sharp, Bruno Sinopoli, Jay Taneja, Gilman Tolle, Kamin Whitehouse, and Bonnie Zhu. Special thanks also goes to Mike Howard for infrastructure support when setting up the wifi networks and David Shim, Hoam Chung, and the rest of the UAV group for assisting and accommodating us at RFS.

Not to be forgotten are the postdocs in my cubicle Jonathan Sprinkle and Rick Groff, both of whom providing useful insights about graduate school and research in general (and a big thanks to Jon for lending me his LaTeX manual in times of need!). Thanks to Neils Hoven and Matthew A. d'Alessio for the LaTeX template.

Curriculum Vitæ

Phoebus Wei-Chih Chen

Education

2002	University of California Berkeley B.S., Electrical Engineering and Computer Sciences
1998	Manalapan High School Science and Engineering Magnet Program

Personal

Born	1980, USA

Chapter 1

Introduction

1.1 Background on Sensor Networks

Recently, there has been great commercial and research interests in a new networked embedded system platform – wireless sensor networks (WSNs). Wireless sensor networks consist of small, low power sensor nodes, sometimes called motes or "smart dust", which have a radio, power supply, microprocessor, and sensors. These sensor nodes can be distributed throughout the environment to form ad-hoc networks that route sensed data back to a gateway of some computing infrastructure. This gateway is often called a *basestation* and the computing infrastructure can be a personal computer or a server on the internet. Sensor networks are a step in the direction of ubiquitous, pervasive computing, which predicts that one day our environment can be seamlessly integrated with computers and cyberspace [1], [2].

The numerous applications for sensor networks include structural health monitoring [3]–[5], environment and habitat monitoring [6]–[8], monitoring of manufacturing equipment [9], security of container shipping, temperature monitoring for office building heating and ventilation systems, and locating snipers [10]. For many of these applications, wireless sensor networks are attractive because they remove the high wiring costs in installing sensor networks. Furthermore, they often are meant to run for years after deployment because the hardware and software for these embedded systems are designed with low power battery operation in mind. In fact, there have been many hardware developments in the last few years to enable low power sensor network platforms [11]–[13].

Much of the current sensor network research on the software and systems side are in routing, software architecture, programming abstractions, basic system services like time synchronization and localization, power management, and crytography/security. On the theory and modelling side, sensor network research covers distributed signal processing, sensor placement and coverage, estimation and detection, and models for simulation. While much progress has been made in both theory and implementation, there are very few theoretical tools available for accurately predicting the performance of a sensor network prior to deployment. Instead, sensor network systems are often built first, then analyzed. In order to build reliable sensor networks larger than the typical deployment of tens to a few hundred sensor nodes, the gap between theory and implementation needs to be closed.

1.2 Sensor Networks and Control

The development of wireless sensor networks for gathering data from our physical environment leads naturally to the development of wireless sensor and *actuator* networks (WSANs) to sense and manipulate our environment. In fact, there are already products on the market for wireless home automation [14], essentially wireless actuator networks for turning on/off lights and appliances. However, for wireless sensor and actuator networks to reach their full potential, there needs to be tight coupling between sensing and actuation. Essentially, a control theory for WSANs needs to be developed.

Fortunately, many of the immediate commercial and home applications involving distributed actuators do not have strict timing requirements. For instance, WSANs for heating and ventilation systems most likely would not need millisecond response times. Also, the actuation modalities are often limited to sending batch commands to equipment connected to the sensing and actuation nodes. These may be to turn off faulty equipment or switch equipment to different modes of operation, and do not require intricate coordination between different pieces of equipment. WSANs are limited to applications with loose timing requirements not because there is no market for fast-response, wireless, distributed control systems, but because the technology is not reliable and mature enough yet.

Wireless sensor and actuator networks can be divided into two classes: those with mobile actuators and those with fixed, immobile actuators. Fast-response WSANs with fixed actuators have potential to increase efficiency and throughput of manufacturing by coordinating production on a factory floor. They may also serve a role in distributed control of large structures, such as vibration damping of buildings and bridges during earthquakes or strong winds [15]–[17]. But WSANs with fast response rates will have the largest impact on applications where the actuators have mobility, particularly in the area of distributed robotics. In fact, the WSAN itself can *be* the distributed robots, as is done in [18]–[20]. Teams of distributed robots can be used for search and rescue missions [21] and mapping unknown terrain [22]. The key research challenges in this area are localization, navigation through unknown terrain, and coordination of the robots, for example to ensure stable robot formations [23].

In control theory, we are concerned with the stability, control effort, overshoot, and settling time of controllers. But in order to design good controllers, we need a model of the system and an understanding of the quality of our model [24]. This includes a model of the plant as well as models of the actuators, the observers, and the communication links between the different components in the control system. Given these models, we wish to develop controllers and assess their performance. When the resulting system is too complex or involves modelling uncertainty, we try to derive *bounds* and *guarantees* on the controller's performance. A good example of this is the work of Sinopoli, et al. [25] which studies how the the loss rates of communication links between the controller, the observer, and actuator can affect the stability of a linear control system. The work finds a threshold γ where if the communication links have a loss rate above γ, the control system will become unstable over time.

When designing controllers for a wireless sensor and actuator network, we face the challenge of modelling a system where the actuators, observers, and controllers are distributed

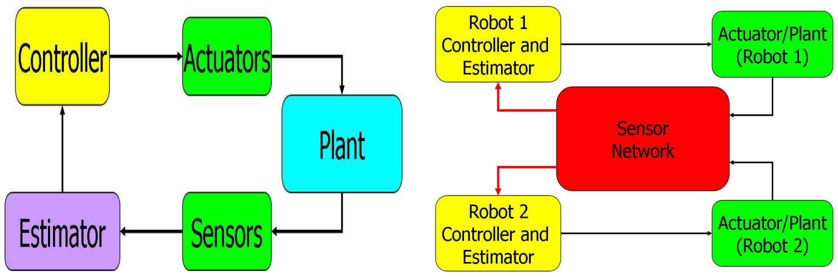


Figure 1.1. (left) The classic feedback control loop. (right) Simple feedback control loops of 2 robots navigating with the aide of a sensor network. In this simple diagram note that the lossy communication links in red are only within the sensor network and between the sensor network and the controller.

all over the nodes in the network. A design in this general setting must consider how to decompose a control law so it can be distributed over the processing units of the nodes, and how to account for loss of synchronization of control commands and observations at different parts of the system. We consider a simpler setting, the navigation and coordination of robots using a sensor network[1] as the observer in the system. This "centralizes" most of the computation and actuation on the robots, and distributes the observation over the nodes in the network. If we assume an ideal communication channel between the robots and good computational capabilities on each robot, we confine the modelling of the distributed system to the sensor network (see Fig. 1.1). The principles behind the control of these distributed, networked systems can then be studied and generalized to distributed robots and WSANs.

Our candidate application for studying controllers for sensor networks is the Pursuit-Evasion Game (PEG), where a group of pursuers try to chase and capture a group of targets. The topic of pursuit-evasion games has been studied as a generalization of control problems for many years [26], [27]. Here, instead of modelling the environment as adding noise to a system, we model the environment as an adversary trying to make the control system unstable so as to expose the controller's worst case performance.

Typically, in a pursuit-evasion game the problem is to find regions in the state space of the game that guarantee capture/evasion under optimal play and the corresponding optimal control laws. There have been efforts in recent years to bound the performance of such games

[1]We will use the term 'sensor network' interchangeably with 'wireless sensor networks' throughout the text.

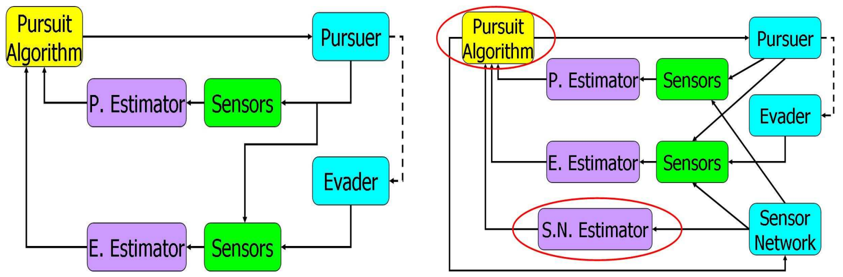


Figure 1.2. The flow of control and observation signals in a (left) traditional pursuit-evasion game and a (right) pursuit-evasion game over sensor networks. We have separated the actual sensors from the rest of the sensor network for illustrative purposes. The "sensor network" controls how the sensor readings reach the pursuer and evader estimators. Note that the controller/pursuit algorithm can control the sensor network as well – for instance to control congestion, enable more power hungry sensor filtering algorithms, etc. The dashed arrows indicates that the pursuer can *indirectly* control an evader if the evader is actively evading. The challenge is to build a good sensor network estimator/model and a compatible control law, both circled in red above.

in a probabilistic sense, as in [28] and [29]. Furthermore, in [30] and [31], pursuit-evasion games have been combined with sensor networks, using the sensor network as the observer for the pursuers. Figure 1.2 illustrates some of the differences between traditional PEGs and PEGs over sensor networks.

This report studies some simple models of sensor networks for controllers and describes the implementation of a sensor network for pursuit-evasion games. While both thrusts are in the early stages, the goal is to try different methods of modelling a sensor network and designing control laws and then compare this with reality. The first chapter describes the issues in modelling a sensor network for control, and studies an example of a path planning controller for a robot using a sensor network to help it pursue a randomly moving target. The second chapter describes a model of a sensor network for computing bounds on the performance of a pursuer in the classic *Lifeline* pursuit-evasion game described by Isaacs [26]. The third chapter describes the construction and implementation of two sensor network testbeds for testing control laws and *multiple target tracking*, an important estimation step for pursuit-evasion games with multiple evaders/targets. The fourth chapter describes the results of a demonstration of a multiple target tracking algorithm on a sensor network testbed. Finally, we conclude with a roadmap of future research directions.

Chapter 2

Modelling of Sensor Networks

One of the critical steps when designing control applications that use sensor networks is the development of a good model of the sensor network as the observer in the system. Like observers in traditional control systems, the observations from sensor networks are noisy. Similar to observers in distributed/networked control systems, the communication link between the observer and the controller may be lossy, leading to missed observations. However, sensor networks are a special class of observers which possess characteristics that allow us to say more about how the observations are lossy and noisy.

There are two categories of characteristics which fundamentally define the performance of a sensor network: *static* characteristics and *dynamic* characteristics. Static characteristics are parameters of a sensor network that are "designed" prior to deployment. They include the hardware limitations on the sensor nodes such as minimum/maximum radio range and sensing coverage, the software design decisions such as the type of routing algorithm used to disseminate data and when/how often to transmit data, and the physical placement of the nodes. The dynamic characteristics are parameters of a sensor network that are monitored after deployment for a "health status" of the network such as the power level of the nodes (which is dependent on the network load), the actual radio range and probabilities of successful transmission as a function of distance for each node, and the placement/distribution of faulty nodes in the network.

These characteristics translate to the actual performance of a sensor network in a complex manner. For instance, there is a strong relationship between the physical placement of the sensor nodes and the quality of the communication links with the controller. The communication link quality in turn affect the actual routing topology used to gather data, which also depends on the routing algorithm. The routing topology together with the communication link qualities determines the communication bottlenecks between the sensors and the controller, which affects congestion. Finally, congestion affects the latency and transmission success rate of observations to the controller. Similarly, the physical placement of the nodes could affect the noise of the sensing measurements. The data collected at nearby sensor nodes often are correlated, so a higher node density may imply lower noise after smoothing the aggregated observations.

From a control engineer's perspective, we need tools to relate these sensor network characteristics to parameters that can be used to help design a control law. Some key control parameters would be latency, transmission failure rate, false negative/false positive rate of discrete observations or sensing noise for continuous observations. It may be, however, that a simple number for transmission failure rate is not enough. For instance, because of congestion or radio interference patterns from external sources, transmission failures are correlated and may come in bursts. The control law could benefit from information on the typical duration of transmission failure, and whether the distribution of these lossy communication time intervals are really random. In a sense, this is the problem of taking a detailed, node-level model and environment model and translating it to an abstract, network-level model for use by a controller.

While computing the exact relationship between sensor network characteristics and control parameters may be hard/impossible, what is really necessary are tools to compute control parameter performance *guarantees* given a set of sensor network characteristics. It is then possible to use these performance guarantees to design a robust control law. We wish to make statements like the following:

> We are given a grid deployment of X nodes running platform Y spaced Z feet apart, running a simple threshold crossing detector and using routing algorithm A to route data back to a centralized controller. Assume no node failures and

an ideal disk radio connectivity model. We can guarantee that the transmission failure rate is below B percent, and C percent of the packets reach the destination with latency no more than 100 ms.

The long term goal is to study typical sensor network deployment scenarios and develop canonical methods for approximating/bounding the control performance parameters. In the process, it may be useful to define new performance parameters and provide examples of control algorithms that use them. In the sections below, we describe a simple node-level model of a sensor network and try to extract a connectivity performance parameter to use in a path planning control law. In the next chapter, we will look at another example of computing performance guarantees for a different control application, pursuit with active evasion, using the same general node-level sensor network model.

2.1 Sensor Network Model

We start by modelling the sensing and communication coverage of individual sensor nodes with a simple disk model. From these models for individual nodes and their placement, we construct a routing topology using a given routing algorithm and then combine it with a packet transmission scheme to get a basic model of the communications network. This sensor network model can be used for target-tracking simulations. In fact, it shares many characteristics with the sensor network model in [32].

2.1.1 Sensor Network Node Model

We assume that N_s sensor nodes are deployed over the surveillance region $\mathcal{R}$ to measure the position of the target. Let $s_i \in \mathcal{R}$ be the location of the i-th sensor node and let $S = \{s_i : 1 \le i \le N_\mathrm{s}\}$.

2.1.1.1 Sensing Model

Denote the sensing range for node i as $R_{s_i} \in \mathbb{R}$. If the target is at $x \in \mathcal{R}$ and there exists i such that $r_i = \|s_i - x\| \le R_{s_i}$, then sensor i can detect the presence of the target with probability

$$p_{s_i} = \begin{cases} \frac{\beta}{\beta + r_i^\alpha}, & \text{if } r_i < R_{s_i} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

for some appropriate constants α and β. This is similar to the SINR-like communication model described below in Equation 2.4.

Let $\mathbf{x}^{\mathbf{e}}(t) = [x_1^e(t), x_2^e(t), \dot{x}_1^e(t), \dot{x}_2^e(t)]^T$ denote the position and velocity of the target. For simplicity, we assume the following linear measurement model on the measurements y.

$$y(t) = C\mathbf{x}^{\mathbf{e}}(t) + v(t), \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.2}$$

where v is a noise process. For instance, v can be a white Gaussian noise process with zero mean and covariance Q^s. This is used later in Section 2.2 so we can use a Kalman Filter to estimate the position of a target for tracking and pursuit of the target. Otherwise, v can be a Uniform noise process, where $v(t) \in \{[r\cos\theta, r\sin\theta]^T : 0 \leq r \leq r_q, 0 \leq \theta \leq 2\pi\}$ is a zero mean uniform random variable on a circle with radius r_q. This model is used in Section 3.3 for deriving an "uncertainty ball" for the position of a target when bounding the performance of a lifeline game.

In real deployments of sensor networks, a sensor may provide ranging and angle to a detected target rather than position like in Equation 2.2. However, we can use data fusion from multiple sensors to get position estimates. This is explained in more detail in Appendix D using a model developed by Songhwai Oh.

2.1.1.2 Communication Model

We assume a disk communication model. Each sensor node i has a maximum radio transmission and reception range of R_{c_i}. Node i can communicate with any other node j within this communication radius with probability p_c. We propose two different communication models. First, let $r_{ij} = \|s_i - s_j\|_2$, and let $R_{c_{ij}} = \min(R_{c_i}, R_{c_j})$.

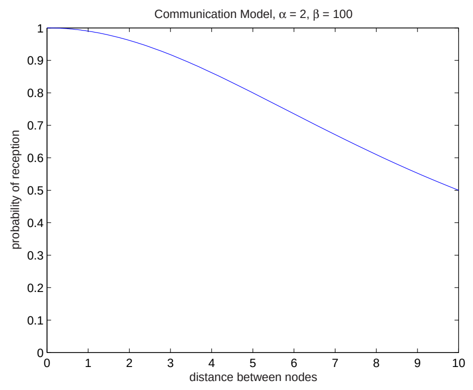


Figure 2.1. Node to node connection probability, based on the model in Equation 2.4.

Linear Communication Model

$$p_{c_{ij}} = \begin{cases} 1 - \frac{r_{ij}}{R_{c_{ij}}}, & \text{if } r_{ij} < R_{c_{ij}} \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

The communication range for each node can be reduced by lowering R_{c_i}.

Although this communication model does not match the laws governing radio signal strength decay, it is the simplest to describe and may be useful for deriving whole-network models by making calculations more tractable.

SINR-like Communication Model

$$p_{c_{ij}} = \begin{cases} \frac{\beta}{\beta + r_{ij}^{\alpha}}, & \text{if } r_{ij} < R_{c_{ij}} \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

This particular model was chosen because it can somewhat approximate the decay model in [33] by proper choices of α and β (see Fig. 2.1). Also, it resembles the form of the standard SINR model except normalized to 1 because it is a probability.

Note that in the SINR-like model, the communication radius R_{c_i} can be chosen for each node i such that any node j within the radius will be connected to node i with a probability above a threshold probability η. This corresponds with designing sensor nodes that keep neighbors in their routing tables only if at least η of the packets sent get through.

2.1.2 Sensor Network Routing Model

Using our disk communication models above, we can generate a communication graph $G = (S, E)$ where $(s_i, s_j) \in E$ if and only if node i can communicate with node j with nonzero probability. We make as many reasonable simplifying assumptions as possible to make the analysis tractable, keeping in mind that we can add more complexity to our model in the future. The model ignores congestion issues by assuming each communication link $e \in E$ transmits/fails independently. The model also assumes that the link probabilities are static over time.

The choice of a routing algorithm depends on the application being run on the sensor network. For instance, in data collection, we wish to optimize data flow from many sensor nodes to one base station. On the other hand, in command dissemination, we wish to optimize data flow from the base station out to the nodes. We are concerned with tracking a target with a mobile controller (for instance, a robot pursuing the target) which would need a routing algorithm optimizing the flow from a group of nodes near the target to a mobile base station.

Rather than use the routing topology (routing tree/mesh) generated by a particular implementable routing algorithm for our model of the sensor network, we choose a routing topology that maximizes the end-to-end transmission success rate (later also referred to as the connection probability) between any two nodes assuming one transmission per link. This ideal routing topology may not be achievable by a deployable routing algorithm because it requires each node to know the global connectivity graph. Nevertheless, there may be implementable routing algorithms whose generated routing topologies yield transmission success rates close to that for the ideal routing topology. We may be able to route to local "distribution-center" nodes near our destination, and each node may not need complete knowledge of the global connectivity graph to find near-optimal routing paths.

To calculate the end-to-end transmission probability given our routing topology,[1] we assume that there are link level acknowledgements, retransmissions at each link, and a

[1]In actuality, a lower bound on the end-to-end transmission probability, as we will see in Section 2.1.2.3.

TTL (time-to-live, like in TCP/IP to count the total number of link transmissions before a packet is discarded) counter for each packet. The necessity of link level acknowledgements and retransmissions is because the transmission success rate with no retransmission drops off exponentially with the number of hops.[2] The inclusion of a TTL counter is fitting for tracking applications because very old observations are less useful to the controller than new observations and can be discarded.

The routing scheme for delivering a packet is simple and standard. The source node is given the destination node, finds the next node in the routing path from the routing topology, and transmits the packet to that node. If it does not hear an acknowledgement, it transmits again to the same node. Each attempted transmission decrements the TTL, and when the TTL is 0, the packet is discarded. This is repeated at each node on the path until the packet reaches the destination.

Thus, our model of the sensor network is a function that takes the positions of the source and sink for network traffic (the position of the target and the position of the controller respectively), selects the two nodes in the network with the best connectivity to the source and the sink, and returns a connection probability for the routing path between these nodes. This connection probability can be used to design a control law to pursue the target as described in Section 2.2. The routing topology and routing scheme can be used to simulate the sensor network and get the latency and success/failure of individual transmissions.

2.1.2.1 Maximum Transmission Probability Criteria

The goal of the routing topology generation algorithm is to find a path that maximizes

$$p_{path} = \prod_{i=1}^{k} p_i \tag{2.5}$$

where k is the number of hops in a given path and each link has a probability of successful transmission p_i.

Unfortunately, the optimum routing path with respect to the criteria in Equation 2.5

[2]Even five hops over links with $p_i = 0.9$ would yield an end-to-end transmission probability of $p_{path} \approx 0.59$. For an extended discussion, see [34].

may be different from the optimum routing path with retransmissions (see examples in Section 2.1.2.4). This is usually not a big problem because the optimum non-retransmission path often has a connection probability and latency path length close to that of the optimum retransmission path.[3]

2.1.2.2 Routing Topology Generation

To find the optimum connectivity paths between every node with respect to the criteria in Equation 2.5, we use a variant of the Floyd-Warshall Algorithm [35], an algorithm to find the all-pairs shortest paths. Instead of adding path weights in each iteration of the algorithm, we multiply them, and instead of taking the minimum, we take the maximum. The input to the algorithm is an $N_s \times N_s$ symmetric matrix of transmission probabilities P where $P_{ij} = p_{c_{ij}}$ from the node-level communication model. The output of the algorithm is an $N_s \times N_s$ predecessor matrix M where entry M_{ij} is the next-hop neighbor when routing from node i to node j. The Floyd-Warshall algorithm takes $O(n^3)$ to compute, where $n = N_s$, the number of nodes in the network.

2.1.2.3 Connection Probability Calculation

To calculate the connection probability of a routing path with retransmissions, let us assume that each packet is assigned a TTL (time-to-live) n and consider a path with k hops. This connection probability should be greater than p_{path} from Equation 2.5. We argue that the excess transmissions $m = n - k$ should be used to retransmit on the *worst* links and boost their transmission probabilities to get a *lower bound* on the connection probability of a chosen path, $\check{p}_{path}$. The rationale for this is simple. Suppose we had one retransmission, $m = 1$. During the routing of a packet, the only time that we we would retransmit on a good link in a path and not retransmit on the worst link in the path is if the good link failed and the worst link succeeded. Thus, conceptually, when we use the retransmission to boost the probability of a good link in the path, there is an implicit understanding that the

[3]A proof of this is currently unavailable. The author is currently looking for an algorithm that efficiently finds the maximum end-to-end connectivity path with retransmissions.

Algorithm 1 Generation of Sensor Network Routing Topology (A variation on the Floyd-Warshall Algorithm)

```
Input: $P$
Output: $M$
$D^{(0)} \leftarrow P$
for $j = 1$ to $N_s$ do
    for $i = 1$ to $N_s$ do
        $M_{ij}^{(0)} \leftarrow \begin{cases} 0 & \text{if } i = j \text{ or } D_{ij} = 0 \\ i & \text{if } i \neq j \text{ or } D_{ij} > 0 \end{cases}$
    end for
end for
for $k = 1$ to $N_s$ do                    ▷ $N_s$ is used here as the maximum path length
    for $j = 1$ to $N_s$ do
        for $i = 1$ to $N_s$ do
            $D_{ij}^{(k)} \leftarrow \max(D_{ij}^{(k-1)}, D_{ik}^{(k-1)} \cdot D_{kj}^{(k-1)})$
            $M_{ij}^{(k)} = \begin{cases} M_{ij}^{(k-1)} & \text{if } D_{ij}^{(k-1)} \geq D_{ik}^{(k-1)} \cdot D_{kj}^{(k-1)} \\ M_{kj}^{(k-1)} & \text{if } D_{ij}^{(k-1)} < D_{ik}^{(k-1)} \cdot D_{kj}^{(k-1)} \end{cases}$
        end for
    end for
end for
$M = M^{(n)}$
```

transmission over the worst link is on average more successful than the good link. This is a contradiction.

When designing a controller that uses the sensor network, we need *guarantees* that a path connection probability is above a threshold. Therefore, our sensor network model returns the lower bound on the path connection probability $\check{p}_{path}$ as the path connection probability. Algorithm 2 calculates $\check{p}_{path}$ using as input $\mathbf{p}^{(ij)}$, a vector of link probabilities for the routing path between two nodes i and j, and n, the number of total transmissions (successful transmissions plus retransmissions) allowed by the routing algorithm.

Algorithm 2 Path Connection Probability Lower Bound Calculation, $\check{\mathbf{p}}_{\mathbf{path}}$

```
Input: $\mathbf{p}^{(ij)}, n$
Output: $\check{p}_{path}$
$\mathbf{p}^{new} = \mathbf{p}^{(ij)}$
$k = length(\mathbf{p}^{(ij)})$
$m = n - k$
for $h = 1$ to $m$ do
    $i = \arg\min(\mathbf{p}^{new})$                    ▷ The weakest link in $\mathbf{p}^{new}$
    $\mathbf{p}_i^{new} = 1 - (1 - \mathbf{p}_i^{(ij)})(1 - \mathbf{p}_i^{new})$
end for
$\check{p}_{path} = \prod_i \mathbf{p}_i^{new}$
```

2.1.2.4 Routing Model Limitations

The routing topology generated from Algorithm 1 optimizes end-to-end transmission success rates given no retransmissions. Unfortunately, this routing topology is no longer optimal when we allow for retransmissions. We illustrate this with two examples. p denoted below is the success probability of a communication link.

1. Different Path Lengths Scenario (Fig. 2.2, left)

- Three Nodes A, B, C with source A and destination C. Links are $A \to B(p = 0.9), B \to C(p = 0.9), A \to C(p = 0.8)$.
- The total number of transmissions is 2. Using the FW-variant, we get the best path to be $A \to B \to C$ with $p = 0.81$. However, we can retransmit twice using the shorter link $A \to C$ to get $p = 1 - (1 - 0.8)^2 = 0.96$.

2. Same Path Length Scenario (Fig. 2.2, right)

- Four Nodes A, B, C, D with source A and destination D. Links are $A \to B(p = 0.4), B \to D(p = 0.5), A \to C(p = 0.3)$, $C \to D(p = 0.64)$.
- If the number of total transmissions is 2, then the optimal path would be $A \to B \to D$ with $p = 0.2$ (whereas the other path would have $p = 0.192$). However, if the number of total transmissions is 3 and we retransmit on the weakest link, the optimal path would instead be $A \to C \to D$ with $p = 0.64(1-(1-0.3)^2) = 0.3264$ (whereas the other path would have $p = 0.5(1 - (1 - 0.4)^2) = 0.32$).
- Method for generating other examples

  Let p_1 and p_2 be the probability of the weakest link in path 1 and path 2 repectively. Similarly, let P_1 and P_2 repectively be the product of the probabilities of all the other links in path 1 (or path 2). Let $p_1 > p_2$, and of course $0 < p_1, p_2, P_1, P_2 \leq 1$.

  $$p_1 P_1 > p_2 P_2 \qquad \text{and} \qquad (1 - (1 - p_1)^2) P_1 < (1 - (1 - p_2)^2) P_2$$

  This simplifies to satisfying the inequalities

  $$\frac{2 - p_1}{2 - p_2} \frac{p_1}{p_2} < \frac{P_2}{P_1} < \frac{p_1}{p_2} \tag{2.6}$$

  First, pick p_1 and p_2 such that $p_1 > p_2$. Then, select P_2/P_1 such that Equation 2.6 is satisfied, and scale P_1 and P_2 to be between 0 and 1.

This implies that the optimal routing topology may be different for each n, the number of total transmissions. However, if we assume that the routing algorithm only considers links with high transmission success rates, the range of link probabilities for different links will

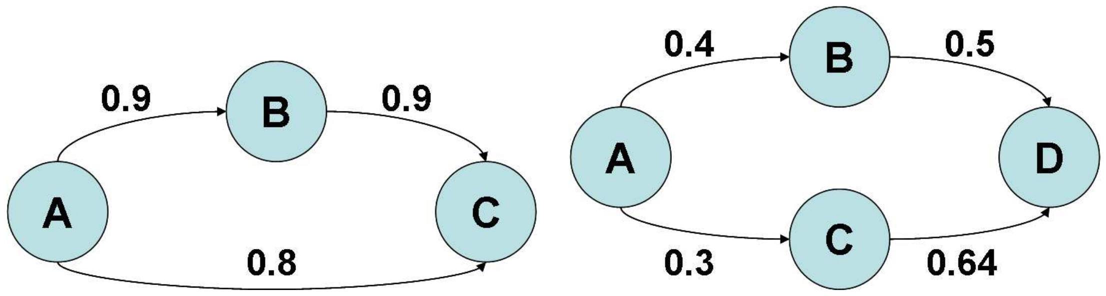


Figure 2.2. Example scenarios with different optimal paths given a different number of total transmissions with (left) varying length paths and (right) same length paths. See the text for details.

be smaller. This means the non-optimality scenario with different path lengths described above is less likely to arise. The non-optimality scenario of paths with the same length described above is less likely to arise in general, which is why a method was described for generating examples.

Regardless, the routing topology generated by Algorithm 1 is useful for comparison with routing topologies generated by other deployable routing algorithms and for developing specialized routing algorithms in the future. An implementation of a deployable routing algorithm would have to overcome these limitations of Algorithm 1 and the routing scheme described in Section 2.1.2:

Centralized computation The algorithm requires knowledge of the link quality between all nodes. It does not take advantage of the sparsity of the graph or that most connections are to neighboring nodes.

Immediate knowledge of destination The routing scheme assumes the source has immediate knowledge of the sink address. If the controller is moving, this knowledge may not be immediate.

Large storage requirements per node Each node must store N_s routing paths entries, one for each potential destination node. This corresponds to storing one row of the matrix M generated by the algorithm.

No diversity Congestion and interference from external sources have not been considered. Therefore, the routing topology has only one route between any two pair of nodes. In a real deployment, link probabilities can fluctuate considerably over time.

No account for latency The routing topology is not optimized for latency, which is important for control applications.

2.2 Path Planning through Sensor Networks

The sensor network model derived in Section 2.1 was tailored toward controllers that needed to track targets. As stated in Chapter 1, we will study pursuit-evasion games where the sensor network is used as the observer for the pursuit controller. In this section, to simplify the problem further, we assume that the evader is not actively evading the pursuer but instead is just a target that travels a random trajectory. This will provide an example of how a network performance parameter can be incorporated into an optimal control framework — in this case a path planning problem for the pursuer.

2.2.1 Problem Formulation

For simplicity, consider one pursuer chasing one target in the surveillance region $\mathcal{R}$ covered by sensor nodes. Note that this problem setup can be easily extended to the case with many pursuers and many targets using the assignment method [31]. Let n_x be the dimension of the target and pursuer's state space, $\mathbf{x^p}(t) \in \mathbb{R}^{n_x}$ be the state of the pursuer and $\mathbf{x^e}(t) \in \mathbb{R}^{n_x}$ be the state of the target/evader at time t for $t \in \mathbb{N}$.

The target and pursuer move around $\mathcal{R}$ with the vehicle dynamics described below. The objective of the pursuer is to capture the target while minimizing energy expenditure and maintaining good connectivity with the network for updates on the target's position. Capture is defined as $\|\mathbf{x^p}(t)-\mathbf{x^e}(t)\| \leq l$ where l is the radius of capture. A precise definition of the pursuer objective follows in Section 2.2.3.

2.2.2 Vehicle Dynamics

The state of the pursuer at time t is $\mathbf{x^p}(t) = [x_1^p(t), x_2^p(t), \dot{x}_1^p(t), \dot{x}_2^p(t)]^T$, where $[x_1, x_2]$ is a position in $\mathcal{R}$ along the usual x and y axes and $[\dot{x}_1^p(t), \dot{x}_2^p(t)]$ is a velocity vector. The control signal is an acceleration input and is denoted by $\mathbf{u}(t) \in \mathbb{R}^2$. A discrete-time linear dynamic model is used

$$\mathbf{x^p}(t+1) = A^p \mathbf{x^p}(t) + B^p(\mathbf{u}(t) + \mathbf{w^p}(t)), \tag{2.7}$$

where $\mathbf{w^p}$ is a white Gaussian noise process with zero mean and covariance Q^p modelling actuator noise,

$$A^p = \begin{bmatrix} 1 & 0 & \delta & 0 \\ 0 & 1 & 0 & \delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B^p = \begin{bmatrix} \frac{\delta^2}{2} & 0 \\ 0 & \frac{\delta^2}{2} \\ \delta & 0 \\ 0 & \delta \end{bmatrix} \tag{2.8}$$

and δ is the sampling period.

The state of the target at time t is $\mathbf{x^e}(t) = [x_1^e(t), x_2^e(t), \dot{x}_1^e(t), \dot{x}_2^e(t)]^T$. We model the dynamics of the target as

$$\mathbf{x^e}(t+1) = A^e \mathbf{x^e}(t) + B^e \mathbf{w^e}(t) \tag{2.9}$$

where the control signal $\mathbf{w^e}(t)$ is a white Gaussian noise process with zero mean and covariance Q^e, and

$$A^e = \begin{bmatrix} 1 & 0 & \delta & 0 \\ 0 & 1 & 0 & \delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B^e = \begin{bmatrix} \frac{\delta^2}{2} & 0 \\ 0 & \frac{\delta^2}{2} \\ \delta & 0 \\ 0 & \delta \end{bmatrix}. \tag{2.10}$$

2.2.3 Pursuer Control Law

The pursuer knows the dynamic model of the target, but it does not have access to the control inputs applied by the target. Therefore, it needs updates on the target's position from the sensor network to capture the target. The goal of our control law is to choose a pursuit path that maintains good connectivity with the sensor network as the pursuer chases after the target. More precisely, we are concerned with whether packets originating near the target (detections) reach the pursuer.

Connectivity with the network is a function both of the position of the pursuer and the position of the target in the network. Let us measure connectivity with the network using the function $\gamma(t) = \gamma(\mathbf{x^p}(t), \mathbf{x^e}(t))$, the probability of the pursuer receiving a packet originating from a detecting sensor near the target when the target and pursuer are positioned at $\mathbf{x^e}(t)$ and $\mathbf{x^p}(t)$, respectively, at time t. $\gamma(t)$ can be calculated using our sensor network model in Section 2.1.2.3. Assuming each node has the same radio communication model, select the closest node to the target as the source node and the closest node to the pursuer as the sink node. Then, apply Algorithm 2 to the path between the source and sink to get the connection probability of the route through the sensor network. For the last hop from the sensor network to the pursuer, we assume a fixed number of transmission attempts (e.g. 3), with the probability of the last link given by the disk communication model for the last node (Eqs. 2.3 or 2.4). The product of these probabilities yields $\gamma(t)$.

We define the relation between connectivity and the error in the estimate on the target's position through $P(t)$, the covariance of the estimate of the target's position at time t.

$$P(t) = \mathbb{E}\Big[\big(\mathbf{x^e}(t) - \mathbb{E}[\mathbf{x^e}(t)|y_{1:t}]\big)\big(\mathbf{x^e}(t) - \mathbb{E}[\mathbf{x^e}(t)|y_{1:t}]\big)^T \Big| y_{1:t}\Big] \tag{2.11}$$

where $y_{1:t}$ are the observations provided by the sensor network from time 1 to t. Because the target has linear dynamics, the covariance of the target's position $P(t)$ can be computed from the Kalman filter. The covariance of the target's position can be computed as described by Sinopoli in [25]:

$$P(t) \quad = \quad P'(t) - \gamma(\mathbf{x^p}(t), \mathbf{x^e}(t))P'(t)C^T(CP'(t)C^T + R)^{-1}CP'(t) \tag{2.12}$$

where

$$P'(t) = AP(t-1)A^T + Q^e. \tag{2.13}$$

Note that P is a function of $\gamma(t)$, which is in turn a function of $\mathbf{x^p}$, which is a function of $\mathbf{u}$. Therefore, the pursuer can indirectly affect P though its control inputs $\mathbf{u}$. Hence, the problem can be formulated as finding the control signals minimizing a tradeoff between the

- distance between the pursuer and the target

- control effort

- error in the position estimate of the target.

We do this in a model predictive control (MPC) framework [36]. Despite lacking a strong theoretical foundation for bounds on stability and robustness in nonlinear settings, MPC has been successfully used for many control problems, particularly in chemical engineering. We choose MPC because is easy to incorporate additional constraints and the controller is relatively easy to implement. Particularly, if the cost function is convex, the controller can be easily implemented via a convex optimization solver.

To describe model predictive control, let t_0 be the current time and T be a time horizon beyond t_0. In MPC, we predict the outputs of our system over the time horizon using the previous observations and a process model, in our case the model for the target's vehicle dynamics. Then, the controller computes the inputs $\mathbf{u}^*(t)$ for $t = t_0, ..., t_0 + T$ that optimize a cost function over this time horizon. Next, the controller applies the input $\mathbf{u}^*(t_0)$. This entire procedure is repeated at the next time step $t_0 + 1$ — the controller computes the predicted outputs of the system for the next time horizon by incorporating new observations, computes the optimal control inputs for the next time horizon, and applies the input $\mathbf{u}^*(t_0 + 1)$.

In our problem setup, the finite horizon cost function is:

$$\begin{aligned} J(\mathbf{u}) &= \lambda_1 \sum_{t=t_0}^{t_0+T-1} \mathbb{E}[\|\mathbf{x^p}(t) - \mathbf{x^e}(t)\|^2] + \lambda_2 \sum_{t=t_0}^{t_0+T-1} \|\mathbf{u}(t)\|^2 \\ & \quad + \lambda_3 \sum_{t=t_0}^{t_0+T-1} trace(\mathbb{E}[P(t)]) \qquad (2.14) \\ \text{where} \quad & 0 \leq \lambda_i, \qquad \lambda \text{ are weighting constants} \\ & \mathbf{u} = \{\mathbf{u}(t) : t_0 \leq t \leq t_0 + T - 1\} \end{aligned}$$

The reason for the expectations around $P(t)$ and $\mathbf{x^e}(t)$ is because $\mathbf{x^e}(t)$ is a random variable that depends on $\mathbf{w^e}(t)$, a white Gaussian noise process (Eq. 2.9), and $P(t)$ is a function of $\mathbf{x^e}(t)$.

Let us denote the optimal control signals by $\mathbf{u}^*$. Notice that $\mathbf{x^p}$ and $P(t)$ are functions

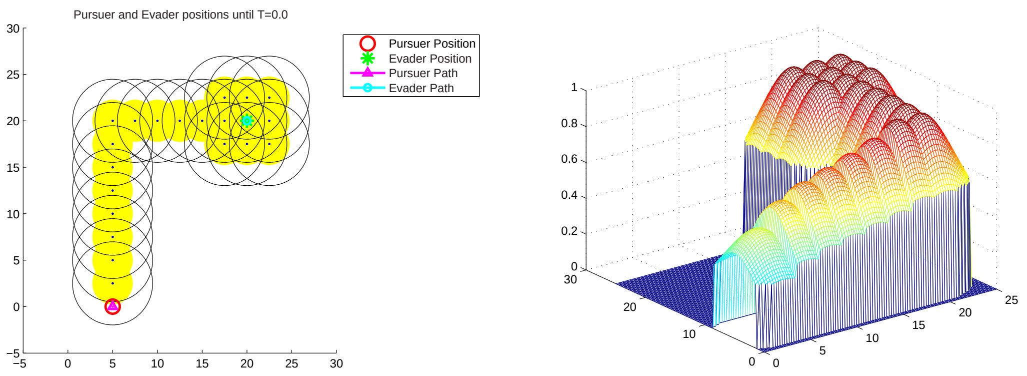


Figure 2.3. (left) Sample sensor node deployment. (right) The corresponding connection probability map using the sensor model in Equation 2.4. (This figure is best viewed in color.)

of $\mathbf{u}$. Then the optimization problem is as follows:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} J(\mathbf{u}), \tag{2.15}$$

such that the vehicle dynamics in Equations 2.7 and 2.9 hold.

2.2.4 Convexity Issues

Unfortunately, our optimization problem is inherently nonconvex. This is because $\gamma(t)$ is a nonconvex function of $\mathbf{x^p}(t)$ and $\mathbf{x^e}(t)$. Recall that we are assuming a disk model for the communication range of each sensor node. Although a disk is convex, a union of disks is in general not convex. Furthermore, locally $\gamma(t)$ decays with increasing distance from the center of the last hop transmission node. These two factors cause γ to have many local minima (see Fig. 2.3).

Nonconvexity means that our controller will likely find control inputs that drive the state to local minima in the cost function and not find the optimal solution. Many different approaches were attempted to reformulate the problem and get approximate solutions but unfortunately none have been very satisfactory. These attempts are described in Appendix E.

2.2.5 Simulation and Evaluation

To evaluate the performance of the connectivity-considerate control law described by Equations 2.14 and 2.15, I wrote a simulator in MATLAB called *PEGSim*. The main loop of the simulator takes discrete time steps and simulates the detection and node-level routing of packets using the sensor network model described in Section 2.1, calculates the pursuer's control inputs using the control law described in Section 2.2.3, and simulates the motions of the pursuer and the target using Equations 2.7 and 2.9. The simulation uses the standard MATLAB optimization toolbox function `fminunc` to find the minimum when calculating the control inputs for the pursuer. The code for PEGSim can be found in Appendix A.1.

The purpose of the simulation was to evaluate whether running our new control law yields better performance than a naive LQG (Linear Quadratic Gaussian - a Kalman Filter composed with a Linear Quadratic Regulator) controller. Performance was measured by the average time to capture the target and the number of runs where the target escaped. Escape meant that the target was not captured within an alloted time T_g. I also simulated some controllers that optimized variants of the cost function in Equation 2.14.

2.2.5.1 Isolating the Connectivity Optimization Term

To test whether the optimization problem actually helps the pursuer stay connected to the network, I remove the pursuit term in our cost function.

Considering only the connectivity term and control effort, we have the optimization problem:

$$\begin{aligned}
\mathbf{u}^* &= \left(\arg\min_{\mathbf{u}} \lambda_1 \sum_{t=t_0}^{t_0+T-1} trace(\mathbb{E}[P(t)]) + \lambda_2 \sum_{t=t_0}^{t_0+T-1} \|\mathbf{u}(t)\|^2 \right) \\
P(t) &= P'(t) - \gamma(t) P'(t) C^T (C P'(t) C^T + R)^{-1} C P'(t) \\
&\text{s.t.} \\
P'(t) &= A P(t-1) A^T + Q \\
\gamma(t) &= \gamma(\mathbf{x^p}(t), \mathbf{x^e}(t)) \quad \text{(as defined in Sec. 2.2.3)} \\
\mathbf{x^p}(t+1) &= A^p \mathbf{x^p}(t) + B^p(\mathbf{u}(t) + \mathbf{w^e}(t))
\end{aligned}$$

$$\mathbf{x^e}(t+1) = A^e\mathbf{x^e}(t) + B^e\mathbf{w^e}(t) \tag{2.16}$$

and all the variables are defined in the same manner as before.

Alternatively, we may wish to bypass the complexity of calculating the expected covariances, and use just a sum of $\gamma(t)$. The optimization problem then becomes:

$$\begin{aligned}
\mathbf{u}^* &= \left(\arg\min_{\mathbf{u}} \lambda_1 \sum_{t=t_0}^{t_0+T-1} \mathbb{E}[\gamma(t)] + \lambda_2 \sum_{t=t_0}^{t_0+T-1} \|\mathbf{u}(t)\|^2\right) \\
\gamma(t) &= \gamma(\mathbf{x^p}(t), \mathbf{x^e}(t)) \\
\mathbf{x^p}(t+1) &= A^p\mathbf{x^p}(t) + B^p(\mathbf{u}(t) + \mathbf{w^e}(t)) \\
\mathbf{x^e}(t+1) &= A^e\mathbf{x^e}(t) + G^e\mathbf{w^e}(t)
\end{aligned} \tag{2.17}$$

2.2.5.2 Simulation Setup and Results

I simulated the pursuer chasing the target using the LQG controller and the connectivity-covariance pursuit controller (Eq. 2.14). The simulations were done on 20 autogenerated 50×50 grid regions of 100 uniformly randomly distributed nodes (see Fig. 2.4 for a sample layout). We chose 5 different starting positions for the pursuer and the evader. This meant that for each type of controller, we ran 100 runs. Capture occurred when the pursuer was within 1 unit distance from the evader, and escape was when $T = 20$ time units had passed without capture. Each simulation step (the time to deliver a packet one hop) was 0.1 time units.[4] The routing algorithm allowed 10 total transmission attempts, with parameters for the sensing and communication SINR probability models $\alpha_s = 2, \beta_s = 100$ and $\alpha_r = 2, \beta_r = 50$ respectively. For the connectivity-covariance pursuit controller, we actually use a look ahead horizon of $T = 3$, or 30 steps, but only allow control inputs for $t = t_0 + (0.1, \ldots, 1)$, as is often done in model predictive control so the optimization algorithm focuses on adjusting control inputs in the closer time horizon. In other words, we force $u(t) = 0$ for $t > 1$. This is slightly different than the formulation in Equation 2.14. The weighting factors for the terms in Equation 2.14 $\lambda_1, \lambda_2, \lambda_3$ are all 1.

[4]This means the time must be rescaled on the equations for the costfunctions and vehicle dynamics above.

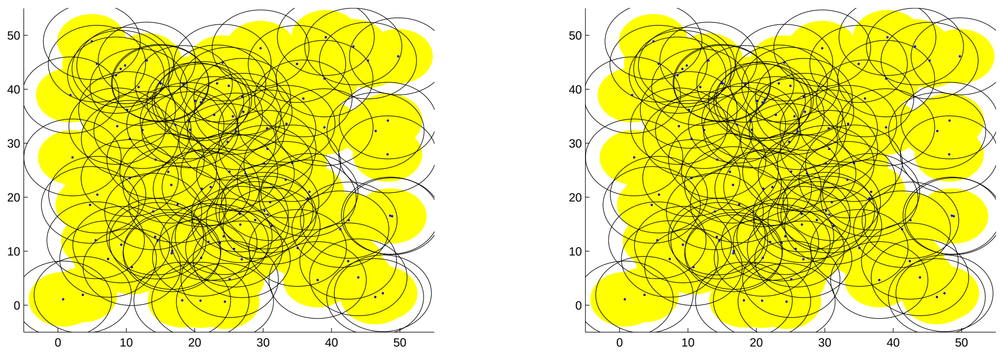


Figure 2.4. Two samples of uniform random distribution of 100 sensor nodes used in the simulations, with their corresponding sensing and communication radii. The sensing regions are in yellow, and the communication radius is denoted by a black circle for each node.

Cost Function	Capture Rate	Average Capture Time
LQG	92 / 100	3.74
Conn-Cov Pursuit	92 / 100	3.83

Table 2.1. Simulation results. Most scenarios yielded similar performance by both control algorithms, with a few where the connectivity-covariance controller taking longer to capture the target.

Unfortunately, the performance of the connectivity-covariance pursuit controller did not show any improvement over the LQG controller (see Table 2.2.5.2 and Fig. 2.5). I simulated the connectivity-covariance controller and the summed connectivity controller to better understand the contribution of the connectivity parameter to our control law. In those simulations, the connectivity-covariance controller often did not move the pursuer to maintain good connectivity with the nodes near the target. Upon closer investigation, the problem was found to be the optimization routine getting stuck in local minima. As mentioned in Section 2.2.4, the many attempts to reformulate the problem have not yet yielded a good solution. More work, and possibly a drastic change in problem perspective may be needed.

2.2.5.3 Remarks on Sensor Network Deployment Classification

During the simulations, it became apparent that it would be difficult to sample the space of sensor network physical deployment layouts effectively for comparison of pursuit

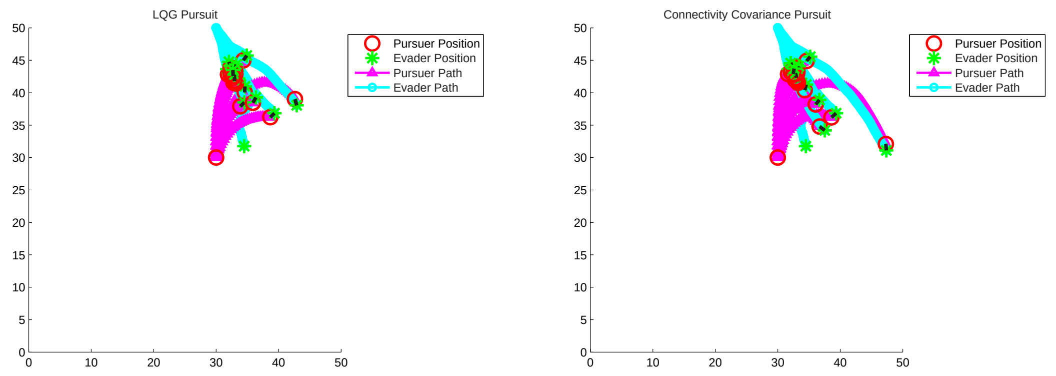


Figure 2.5. Simulation results for (left) 20 runs using the LQG controller (mean capture time = 4.03, 19/20 captured) and (right) 20 runs using the connectivity-covariance controller (mean capture time = 4.24, 19/20 captured) given the same starting position of the pursuer and the same initial trajectory of the evader until capture by either algorithm. For each of the 20 runs, the underlying sensor network topology/physical layout is different, hence not shown. A black line connects a pursuer/target pair when the target is captured. (This figure is best viewed in color. For more figures, see Appendix C.1.)

control laws. In fact, while one may think that there are many scenarios where an optimal controller that maintains good network connectivity can capture a target when one that does naive LQG does not, it is not so clear how to generate such a scenario for testing.

In Figure 2.6, we see a layout of sensor nodes that may have a higher probability of the target escaping when pursued by a LQG controller. Note that upon receipt of messages, our connectivity-considerate controller will take a path through sensor nodes. There is a moment when the target will be out of communication range because of a partition in the network. This is key: for our connectivity-considerate controller to work better than an LQG controller that drives straight to the last position of the target, the pursuer must have difficulty correcting its estimate of the target position when arriving at the target's old position. Our connectivity-considerate pursuer will have a better estimate of which direction the target escapes in, and hopefully move close enough to "catch the scent of the trail" of the target again. The LQG controller may predict the target position incorrectly and head off in the wrong direction.

A method to classify the physical layout of different sensor network deployments is necessary to help in limiting the sensor network sample space when running simulations to compare the performance of different control laws. For instance, one criterion that affects

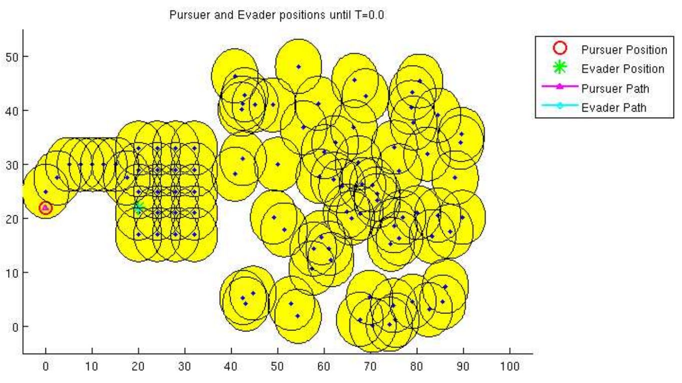


Figure 2.6. Scenario where, with appropriate parameter tuning, we should see capture by the connectivity-considerate controller and loss of tracking by the naiive LQG controller. (This figure is best viewed in color.)

control law performance is whether the network is partitioned. Another often studied characteristic is the sensor coverage of the physical space of the deployment.

Chapter 3

Performance Bounds on the Lifeline Pursuit-Evasion Game

This chapter attempts to bound the performance of pursuit-evasion games played over sensor networks, finding initial conditions that can guarantee capture of the evader by the pursuer. In particular, we study the classic *Lifeline* game described by Isaacs in [26] when a deployed sensor network aides the pursuer in locating an evader. This game is of particular interest because it is an abstraction of the problem of protecting oil pipelines with a sensor network. In fact in December of 2004, a large effort was made to design sensor network hardware and software for this problem [37].

Unlike the classic Lifeline game studied by Isaacs, in the game played over the sensor network the pursuer no longer has perfect information. Furthermore, there is a nonzero probability that observations do not reach the evader's position. As a result, the solution will rely on the notion of a *probabilistic* barrier. That is, for a fixed probability p_{conn}, we can find a barrier/surface separating the game state space such that all initial states on one side of the barrier will result in capture with probability p_{conn}. p_{conn} will depend entirely on the probabilities associated with the wireless communication channel in the sensor network, and not with predictions on how the evader will play the game. In this sense, we can still speak of the pursuer playing against the worst-case behavior of the evader.

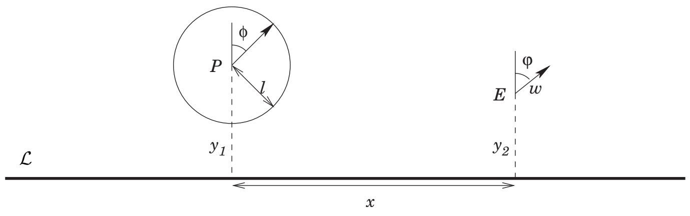


Figure 3.1. Classic lifeline game setup. (Reproduced from [26])

3.1 Background: The Classic Lifeline Game

The classic Lifeline game is described in example 9.5.1 of [26].[1] We have a pursuer $\mathbf{x^p}$ and an evader $\mathbf{x^e}$, both exhibiting simple motion in a half plane bounded by the infinite line $\mathcal{L}$. The objective of the pursuer is to capture the evader, while the objective of the evader is to reach $\mathcal{L}$. Capture occurs when $\|\mathbf{x^p} - \mathbf{x^e}\| < l$.

3.1.1 Vehicle Dynamics

Only the relative speed of the pursuer and the evader matter, hence the pursuer is assumed to have speed 1 and the evader to have speed w. Furthermore, to keep the game interesting, we assume $w < 1$. Otherwise, the evader can always win the game.

Assume the game is played on an xy-plane $\mathcal{R}$, with $\mathcal{L}$ running along the x-axis. The states of the system are x, the distance between the x-coordinates of the pursuer and the evader, y_1, the y-coordinate of the pursuer, and y_2, the y-coordinate of the evader (See Fig. 3.1). We refer to them collectively as $\mathbf{x} = (y_1, y_2, x)$. The control input for the pursuer is the steering angle ϕ and the control input for the evader is the steering angle ψ.

The Kinematic Equations (KE) for the system are:

$$\begin{aligned} \dot{y}_1 &= \cos\phi \\ \dot{y}_2 &= w\cos\psi \\ \dot{x} &= w\sin\psi - \sin\phi \end{aligned} \qquad (3.1)$$

[1]See this reference or Appendix F for the general techniques used below to solve differential games.

3.1.2 Solution

The Main Equation (ME) for the system is:

$$\max_{\psi} \min_{\phi} [(v_1 \cos\phi - v_3 \sin\phi) + w(v_2 \cos\psi + v_3 \sin\psi)] = 0 \tag{3.2}$$

where v_1, v_2, v_3 are the coordinates of the normal vector to the barrier surface (see Appendix F for details).

The Retrograde Path Equations (RPE) for the system are:

$$\begin{aligned} \mathring{v}_i &= 0, \quad i \in \{1,2,3\} \\ \mathring{y}_1 &= \frac{v_1}{\rho_1} \\ \mathring{y}_2 &= -w\frac{v_2}{\rho_2} \\ \mathring{x} &= v_3 \left(\frac{1}{\rho_1} - \frac{w}{\rho_2} \right) \end{aligned} \tag{3.3}$$

$$\text{where } \rho_1 = \sqrt{v_1^2 + v_3^2}, \quad \rho_2 = \sqrt{v_2^2 + v_3^2}$$

The *terminal surface* $\mathcal{C}$ consists of two regions, $\mathcal{C}_1$ and $\mathcal{C}_2$. $\mathcal{C}_1$, the capture surface, is the exterior of the cylinder with axis along the line $\{\mathbf{x} : x = 0, y_1 = y_2\}$. Horizontal slices of this cylinder correspond to fixing y_1, the position of the pursuer relative to $\mathcal{L}$. $\mathcal{C}_2$, the escape surface, is the $y_2 = 0$ plane. Termination on $\mathcal{C}_1$ means that the pursuer wins, whereas termination on $\mathcal{C}_2$ means that the evader wins. The semipermeable surface serving as the barrier must pass through $\mathcal{K}$, the boundary between $\mathcal{C}_1$ and $\mathcal{C}_2$.

The valid *playing region*, denoted $\mathcal{E}$, is in the quarter space $\{\mathbf{x} : y_1 \geq 0, y_2 \geq 0\}$ excluding the region within the cylinder.

$\mathcal{K}$, the intersection of the two terminal surfaces, can be parameterized by

$$\begin{aligned} y_1 &= l \cos s \\ y_2 &= 0 \\ x &= l \sin s, \quad -\frac{\pi}{2} \leq s \leq \frac{\pi}{2} \end{aligned} \tag{3.4}$$

Using the general procedure for solving games of kind outlined in [26], we get equations

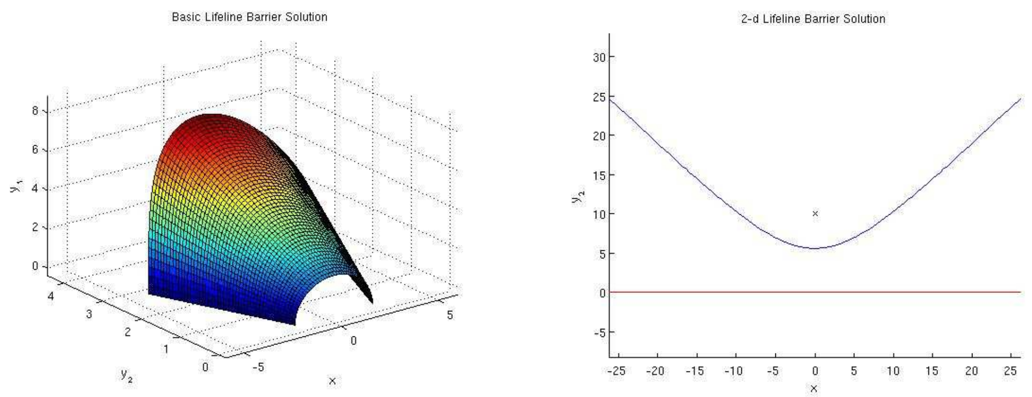


Figure 3.2. (left) Full barrier solution for the Lifeline game. (right) Barrier solution for the Lifeline game when the position of the pursuer is fixed with respect to the lifeline. 'X' marks the position of the pursuer, the blue line is the barrier, and the red line is the lifeline $\mathcal{L}$. (This figure is best viewed in color.)

describing the barrier:

$$\begin{aligned} y_1 &= (l+\tau)\cos s \\ y_2 &= w\tau\sqrt{1-w^2\sin^2 s} \\ x &= (l+(1-w^2)\tau)\sin s \end{aligned} \tag{3.5}$$

The barrier surface is depicted on the left in Figure 3.2. To interpret this graph, fix the position of the pursuer with respect to the lifeline $\mathcal{L}$ by setting y_1 constant. This is equivalent to taking a horizontal slice of the barrier, and is depicted on the right in Figure 3.2. The *capture zone* lies above the barrier and the *escape zone* lies below the barrier.

The optimal control laws on the barrier surface obtained from the ME are given by:

$$\begin{aligned} \cos\bar{\phi} = -\frac{v_1}{\rho_1}, &\qquad \sin\bar{\phi} = \frac{v_3}{\rho_1} \\ \cos\bar{\psi} = \frac{v_2}{\rho_2}, &\qquad \sin\bar{\psi} = \frac{v_3}{\rho_2} \end{aligned} \tag{3.6}$$

$$\text{where } \rho_1 = \sqrt{v_1^2+v_3^2}, \quad \rho_2 = \sqrt{v_2^2+v_3^2}$$

$$v_1 = \cos s, \quad v_2 = \sqrt{(1/w^2)-\sin^2 s}, \quad v_3 = \sin s$$

3.2 Sensor Network Model

We assume the same node-level sensor network model used in Section 2.1.1, and use the same notation. For sensing, we use the linear measurment model in Equation 2.2 with a uniform noise process v. The communication model can be either the linear communication model given in Equation 2.3 or the SINR-like communication model in Equation 2.4, although it could also be from empirical data. All we need is the connection probability p_c for each pair nodes.

We assume that we are given a routing topology, whether it be one generated by any of numerous routing algorithms for sensor networks like [33], [38] or the ideal topology generated in Section 2.1.2. Although most available sensor network routing algorithms are not designed for control applications, we assume they can be adapted by keeping the same routing topology but changing the transmission control scheme. We assume the same transmission scheme using time-to-live (TTL) counters in Section 2.1.2. Using this assumption and the routing topology, we can derive an abstracted model for the sensor network to computer a bound on the performance of the pursuer (the controller).

3.2.1 Abstracted Model for Sensor Network

A sensor network deployment may not be uniform, resulting in "bad" regions (worse sensing resolution, worse connectivity and latency to surrounding neighbors). In the worst case scenario for the pursuer, we can assume that the evader will exploit these bad regions in the network. As such, we will assume the network performs everywhere only as well as the performance at the worst point in the network.

To simplify the representation of a sensor network, we will abstract it using a *n-hop disk model*. This means that given a routing topology, a sensing radius R_s for all nodes, and a chosen probability of connectivity p_{conn}, we can calculate for each position $\mathbf{z}$ in the playing field a disk of radius r_n such that all targets within this disk and within the playing region $\mathcal{R}$ can be detected and have their detections routed to $\mathbf{z}$ with probability greater

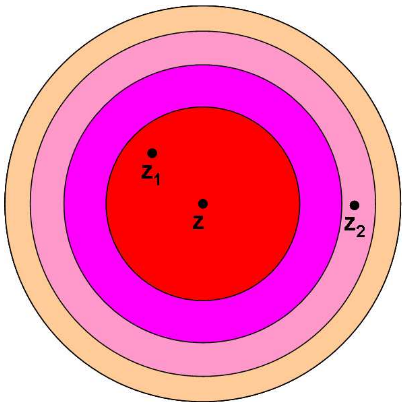


Figure 3.3. Illustration of an n-hop disk model for a point $\mathbf{z}$ in $\mathcal{R}$. Assuming $p_{conn} = 0.9$, we would say that detections from $\mathbf{z_1}$ and $\mathbf{z_2}$ would reach $\mathbf{z}$ with probability greater than or equal to 0.9 in 1 and 3 time intervals respectively.

than p_{conn} after n total transmissions. In this manner, we can draw disks of radius r_n, for each n, around each point $\mathbf{z}$ (See Fig. 3.3).

In the end, we wish to have only *one* of these n-hop disk models abstracting the entire sensor network. Naturally, we take the worst case over all points $\mathbf{z}$ in the playing field. In other words, for each ring corresponding to n hops, we take the smallest radius r_n given by all points $\mathbf{z}$ in the playing field.

3.2.2 Computation of n-hop Disk Abstraction

To calculate the n-hop disk model for every point in the playing field, we need only

1. check that the entire playing region is covered (no sensing gaps in the sensor network)
2. take the minimum r_n in our model from the n-hop disk model of each sensor node (N_s nodes total).

Here, we are assuming that the pursuer is equipped with a superior transmitter and receiver such that if the pursuer is within the communication radius of the last node, it receives the packet with probability 1.

To check for complete coverage of the playing region, one can simply discretize the entire playing space with spacing d and check coverage at each point by at least one of the

N_s nodes with their sensing radii reduced from R_s to $R_s - d$. Of course, we need to also check that all nodes can communicate with each other, or at least the nodes that are out of communication range are not needed to cover the playing field.

The calculation of an n-hop disk model for the network given a threshold probability p_{conn} is described in Algorithm 3. The input to the algorithm is P, an $N_s \times N_s$ matrix of connection probabilities p_c, and the number of total transmissions allowed, n. The output is a vector $\mathbf{r} = (r_1, \ldots, r_n)$ that represents the n-hop disk model for the entire network. This algorithm calls functions *NHopNeighbors*, *getPathProb*, *getGamma*, and *getMaxInscribedDisk*. *NHopNeighbors*(i, j) just returns the neighbors of node i within j hops, which is easily obtained from the routing topology. *getPathProb*(i, k, P) returns a vector of link probabilities for each hop on the path between node i and node k. *getGamma*$(\mathbf{p}^{path}, n)$ is computed via Algorithm 2 of Section 2.1.2.3 and returns a lower bound on the end-to-end connection probability along a path. Finally, *getMaxInscribedDisk*$(\mathcal{N}^j_{conn}, s_i, R_s)$ is simply a function that returns the maximum radius of a disk centered at s_i that is covered by all disks of radius R_s with centers in $\mathcal{N}^j_{conn}$. To compute this, one could make approximations by discretizing the space.

If we were to assume that *all* the link probabilities were a constant p_c, then we could calculate the m-hop connection probability given n total transmissions, γ_m, as

$$\begin{aligned} \gamma_m &= (1 - (1 - p_c)^{\lfloor \frac{m}{n} + 1 \rfloor})^j (1 - (1 - p_c)^{\lfloor \frac{m}{n} \rfloor})^{n-j} \\ j &= m \bmod n \end{aligned} \tag{3.7}$$

This equation comes from using the $n - m$ retransmissions to boost the transmission probability on the weakest link. Using this equation would simplify the calculations in Algorithm 3.

In any case, $\mathbf{r}$ describes delay, n, as a function of distance, r_n. This can be converted to a delay function,

$$d(r) = \arg\max_n (r_n : r_n < r) \quad , \tag{3.8}$$

a composition of step functions with non-differentiable points. Should we wish to make $d(r)$ smooth, we can upper bound the delay function with a polynomial interpolated through the

Algorithm 3 n-hop disk given threshold probability p_{conn}

```
Input: $S$, $P$, $n$                              ▷ see text and Section 2.1 for details
Output: $\mathbf{r}$
for $i = 1$ to $N_s$ do
    for $j = 1$ to $n$ do
        $\mathcal{N} = \text{NHopNeighbor}(i, j)$
        for $k \in \mathcal{N}$ do
            $\mathbf{p}^{path} = \text{getPathProb}(i, k, P)$
            $\gamma_k = \text{getGamma}(\mathbf{p}^{path}, n)$
            if $\gamma_k > p_{conn}$ then
                add $s_k$ to $\mathcal{N}^j_{conn}$
            end if
        end for
        $r_{ij} = \text{getMaxInscribedDisk}(\mathcal{N}^j_{conn}, s_i, R_s)$
    end for
end for
$\forall j, \mathbf{r}_j = \max_i(r_{ij})$
```

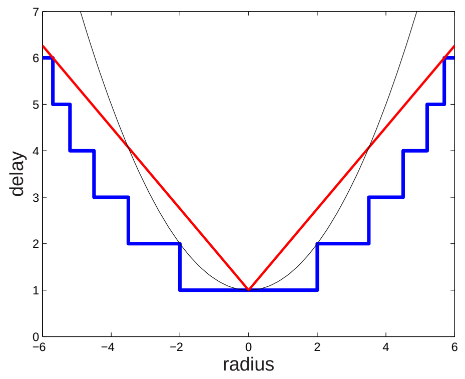


Figure 3.4. Sample plot of delay as a function of distance, in blue. The red line is a smooth linear approximation of the delay function, and the black line is a smooth quadratic approximation.

points of discontinuity (See Fig. 3.4). For instance, we can use a "linear" function (the $|\cdot|$ function), or a quadratic function for the delay to approach ∞ faster as the radius increases.

3.3 Formulation of the New Lifeline Game

To account for latency and missing observations because of a sensor network, we introduce a variable for the evader's *uncertainty radius*, l_e, into the classic Lifeline game. Let us set $r = \|\mathbf{x}^{\mathbf{p}} - \mathbf{x}^{\mathbf{e}}\| = \sqrt{(y_1 - y_2)^2 + x^2}$ and let the delay function, a monotonic function of r, be written as $d(r)$. Then

$$l_e = wd(r) + r_q \tag{3.9}$$

where r_q is from the sensing model with uniform noise described in Section 2.1.1.1.

The modified game is set up as follows:

Definition *Delayed Game Setup*: The new game uses the same KE as the classic lifeline game (3.1). The evader has perfect information on the position of the pursuer and itself at the current time t. The pursuer has perfect information on his own position at t, but only knows the position of the evader at time $t - n$ within a ball of radius r_q, where $n = d(r)$ is the delay before a packet reaches the pursuer with probability greater than p_{conn} as given by (3.8). The *uncertainty ball* $\mathcal{B}^e$ containing the position of the evader at time t has radius l_e and is centered at the evader's measured position $\mathbf{x}^{\mathbf{e}}(t - n)$. ■

Let us call the modified game *Game A* and define the objectives for the pursuer and the evader.

Definition *Game A — the non-zero sum delayed game*: Assume the general setup of the Delayed Game. The objective of the pursuer is to capture the evader before any of $\mathcal{B}^e$ crosses the lifeline. The objective of the evader is to reach the lifeline $\mathcal{L}$ without being captured.

Capture from the evader's perspective occurs when $r < l$. Capture from the pursuer's perspective occurs when $r < l - l_e$, where we assume that l_e eventually is less than l as $\mathbf{x^p}$ approaches $\mathbf{x^e}$. Escape from the evader's perspective occurs when $y_2 \leq 0$. Escape from the pursuer's perspective occurs when $y_2 - l_e \leq 0$.

The game terminates when either the evader escapes from the evader's perspective or the evader is captured from the pursuer's perspective. Let $\mathcal{C}_2^A$ denote the "escape" surface in the game and $\mathcal{C}_1^A$ denote the "capture" surface in the game.[2] ■

Note that because the objectives of the pursuer and the evader are slightly different, Game A is no longer a zero-sum game.

We wish to show that there exists a zero-sum game, *Game B*, where if the evader can win in Game A starting at position $\mathbf{x_0^e}$, he can also win in Game B starting at position $\mathbf{x_0^e}$. This means that Game B is an *easier* game than Game A for the evader. We choose Game B to simply be Game A played entirely using the pursuer's value function.

Definition *Game B — the zero-sum delayed game*: Assume the general setup of the Delayed Game. The objective of the pursuer is to capture the evader before any of $\mathcal{B}^e$ crosses the lifeline, and the objective of the evader is to make some part of $\mathcal{B}^e$ cross the lifeline before capture. Capture occurs when

$$r < l - l_e \quad , \tag{3.10}$$

[2]We cannot speak of semipermeable surfaces for non-zero sum games, but the notion of a terminal surface is still valid.

and escape occurs when

$$y_2 - l_e \leq 0 \quad , \tag{3.11}$$

where again we assume l_e eventually is less than l as $\mathbf{x^p}$ approaches $\mathbf{x^e}$. The capture and escape terminal surfaces are $\mathcal{C}_1^B$ and $\mathcal{C}_2^B$ respectively. ■

Game B is a zero-sum game because the objective of the pursuer is the complement of the objective of the evader.

We can actually interpret Game B as a game where the evader plays in the same frame of reference/time as the pursuer. Although the control law for the evader controls the *current* position of the evader, the evader is playing with respect to the *future* position of the pursuer. We assume that the evader can calculate the latency before the pursuer receives updates on the evader's current position. Furthermore, the evader can predict the future position of the pursuer if the pursuer is playing with the optimal policy $\bar{\phi}$ and the evader knows all past states of the game.

Pursuit-evasion games are naturally *games of kind*, meaning the outcome of the game is binary — escape or capture. As such, the players in the game need only play optimally on a subset of the states of the game. For instance, if a pursuer is close to capturing an evader when far away from the lifeline, he can choose to be idle for a short period before pursuing and capturing the evader. This means that there are multiple "optimal" strategies available to each player that give the same outcome of the game.

To really speak in terms of optimal strategies, we need to embed these two games in equivalent *games of degree*, where the value of the game for the players lie in $\mathbb{R}$. The values would correspond to the time to capture/time to escape for the pursuer and evader respectively. In the classic Lifeline game, we have for V the value of the game:

$$V = \begin{cases} -\operatorname{dist}(\mathbf{x^e}, \mathcal{L}), & \text{evader captured} \\ \|\mathbf{x^p} - \mathbf{x^e}\| - l, & \text{evader escapes} \end{cases} \tag{3.12}$$

Note that V agrees with the original game of kind, with $V > 0$ corresponding to escape and $V < 0$ corresponding to capture. Furthermore, this piecewise definition of V agrees at $V = 0$ when the evader is captured just as it reaches $\mathcal{L}$. To maximize/minimize V, the pursuer and the evader must play optimally at all times during the game. The embedding of Game B can be done in a similar manner:

$$V = \begin{cases} -\operatorname{dist}(\mathbf{x^e}, \mathcal{L}) + l_e, & \text{evader captured} \\ \|\mathbf{x^p} - \mathbf{x^e}\| - l + l_e, & \text{evader escapes} \end{cases} \tag{3.13}$$

Since Game A is a non-zero sum game, there is a different value function for the evader and the pursuer. Let's call the *value of the game for the evader* V_A^e, where $V_A^e \geq 0$ for escape from the evader's perspective and $V_A^e < 0$ for capture from the evader's perspective. We can define the embedding analogously to Equation 3.12.

Recall from [26] that the strategies ϕ and ψ are functions of the state, $\mathbf{x}$, and the derivative of the value function, V_x. Then for game A (B) we can define the *pursuer's optimal control law* as $\bar{\phi}_A$ ($\bar{\phi}_B$) and the *evader's optimal control law* as $\bar{\psi}_A$ ($\bar{\psi}_B$). We wish to guarantee that the pursuer can win a game should he play with his optimal strategy.

Theorem 3.3.1. *Let the initial state of both Game A and Game B be* $\mathbf{x_0}$. *Assume the pursuer can win Game B if he plays with strategy* $\bar{\phi}_B$. *Then, if the pursuer plays* Game A *with strategy* $\bar{\phi}_B$, *he can force* $V_A^e < 0$.

Proposition 3.3.2. *Assume the pursuer plays with strategy* $\bar{\phi}_B$ *in both Game A and Game B. If the evader can force* $V_A^e \geq 0$ *in Game A, then the evader can win in Game B.*

Proposition 3.3.3. *Assume the pursuer plays with strategy* $\bar{\phi}_B$ *in both Game A and Game B. If the evader can force* $V_A^e \geq 0$ *in Game A using strategy* $\bar{\psi}_A$, *then the evader can win in Game B using the same strategy* $\bar{\psi}_A$.

Proof. Note that Proposition 3.3.3 implies Proposition 3.3.2. Furthermore, Proposition 3.3.2 is the contrapositive of Theorem 3.3.1. Therefore, if we prove Proposition 3.3.3, then we have proved the theorem.

We need to show that if the pursuer plays the strategy $\bar{\phi}_B$ and the evader plays the strategy $\bar{\psi}_A$ for both games, $V_A^e \geq 0 \Rightarrow V_B \geq 0$ (where V_B is the value of Game B).

Recall that the KE for Game B is the same as that of Game A, and similar to that of the Classic Lifeline Game (Eq. 3.1). The main difference lie in the terminal surfaces of the games. Note that to reach the "escape" terminal surface of Game A, $\mathcal{C}_2^A$, from the playing region $\mathcal{E}$, one must pass through the "escape" terminal surface of Game B, $\mathcal{C}_2^B$. This is because $V_A^e \geq 0$ implies that $y_2 = 0$. But note that in Game B, $y_2 = 0$ implies that $\mathcal{B}^e$ has crossed the lifeline, or that $V_B \geq 0$. Therefore, if the evader causes $V_A^e \geq 0$ in Game A using $\bar{\psi}_A$, $\bar{\psi}_A$ will force $V_B \geq 0$ in Game B. □

3.4 Solution to the New Lifeline Game

We wish to find a probabilistic barrier for Game A, but because Game A is not a zero-sum game, the techniques developed by Isaacs for differential games cannot be applied directly to solve it. However, using Theorem 3.3.1, it is sufficient to solve Game B to obtain a probabilistic barrier and an optimal strategy for the pursuer and apply them to Game A. The capture zone delimited by the probabilistic barrier for Game B would lie within the capture zone for Game A.

We begin by defining the terminal surface of Game B. The capture and escape surfaces are derived from (3.10) and (3.11) using equalities instead of inequalities. The intersection of the capture and escape surfaces, $\mathcal{K}$, is the solution to

$$\begin{aligned} r &= l - (wd(r) + r_q) \\ y_2 &= wd(r) + r_q \end{aligned} \tag{3.14}$$

If we assume that the delay function is quadratic with constant coefficient c and value d_0 at r = 0, $d(r) = cr^2 + d_0$, then after substitution into (3.14) and solving for $\mathcal{K}$ in $\mathcal{E}$ we get:

$$y_2 = \frac{(2wcl + 1) - \sqrt{(wcl + 1)^2 - 4wc(wcl^2 + wd_0 + r_q)}}{2wc} \tag{3.15}$$

Where we chose the smaller of the two solutions to the quadratic equation so $l - y_2 > 0$. If we assume the delay function is linear with constant coefficient c and value d_0 at r = 0, $d(r) = cr + d_0$, we get:

$$y_2 = \frac{wcl + wd_0 + r_q}{1 + wc} \tag{3.16}$$

Recall that $r = \sqrt{(y_1 - y_2)^2 + x^2}$. $\mathcal{K}$, the intersection of the barrier with the terminal surface, is therefore a slice of the tapering cylinder described in Equation 3.14 at the specified value of y_2, which is a circle of radius $l - y_2$. Therefore, we can parameterize $\mathcal{K}$ with parameter s as

$$\begin{aligned} y_1 &= (l - y_2)\cos s + y_2 \\ y_2 &= K, \\ x &= (l - y_2)\sin s, \quad -\frac{\pi}{2} \le s \le \frac{\pi}{2} \end{aligned} \tag{3.17}$$

where K is a constant given by Equation 3.15 or Equation 3.16, depending on the chosen delay model. Taking the derivative with respect to the parameter s yields

$$\begin{aligned} \frac{dy_1}{ds} &= -(l - y_2)\sin s \\ \frac{dy_2}{ds} &= 0 \\ \frac{dx}{ds} &= (l - y_2)\cos s \end{aligned} \tag{3.18}$$

We know that the normal vector v should be perpendicular to the barrier. Therefore, $v \cdot \frac{d\mathbf{x}}{ds} = 0$ (again, $\mathbf{x} = (y_1, y_2, x)$). This yields

$$-v_1(l - y_2)\sin s + v_3(l - y_2)\cos s = 0$$

$$\text{or} \quad v_1 = \cos s, \quad v_3 = \sin s \tag{3.19}$$

which is exactly the same v_1 and v_3 as in the classic lifeline game. Using the same procedure in [26] we also get $v_2 = \sqrt{(1/w^2) - \sin^2 s}$.

In fact, since the KE are the same as the classic lifeline game, we can reuse the RPE in Equation 3.3. Therefore, the steps for solving the barrier are the same as those in [26].

The final equations describing the barrier in Game B are:

$$\begin{aligned} y_1 &= (l - K + \tau)\cos s + K \\ y_2 &= K - w\tau\sqrt{1 - w^2\sin^2 s} \\ x &= (l - K + (1 - w^2)\tau)\sin s \end{aligned} \tag{3.20}$$

where

$$K = \frac{wcl + wd_0 + r_q}{1 + wc} \tag{3.21}$$

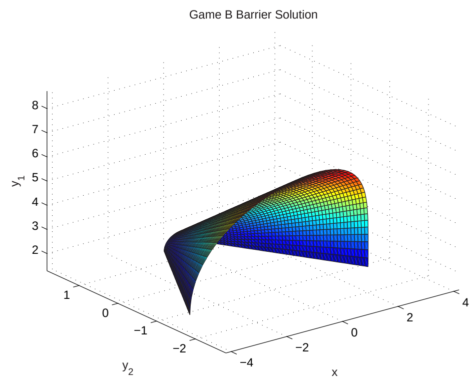


Figure 3.5. Barrier solution for Game B, assuming linear delay. Note that it looks similar to the barrier for the classic lifeline game. (This figure is best viewed in color.)

for linear delay and

$$K = \frac{(2wcl + 1) - \sqrt{(wcl + 1)^2 - 4wc(wcl^2 + wd_0 + r_q)}}{2wc} \tag{3.22}$$

for quadratic delay.

Note that Equation 3.20 closely resembles Equation 3.5. See Figure 3.5 for the new barrier with linear delay (which is similar to the barrier with quadratic delay).

Surprisingly, because the normal vector v and the ME for Game B is the same as those for the classic lifeline game, the optimal strategies for the pursuer and the evader also are the same. Mathematically, this comes about because the Kinematic Equations are not dependent on the state, and hence $f_{ij} = 0$ and $\dot{v}_i = 0$, leading to the simple optimal control laws in Equations 3.6.

3.5 Extending the Framework

This chapter provides a sample framework for thinking about how to bound the performance of pursuit-evasion games on sensor networks. It introduces the notion of calculating a probabilistic barrier to find capture and escape zones for a lifeline game, and a method for abstracting a sensor network to an n-hop disk model used in such calculations.

Much work remains to be done to extend this framework. First of all, the approximations used in this paper provide very loose bounds on the performance of the pursuer, due to loose bounds on worse case sensor coverage and communication latency. Also, the problem setup and the models proposed for the sensor network and players can be enriched considerably. For example, we should:

- account for consecutive missing packets
- assume false alarms in sensing
- extend the problem to multiple pursuers and evaders
- account for gaps in sensor network sensing coverage
- enrich the vehicle dynamic models
- allow for obstacles in the playing field.

If we extend the problem to multiple pursuers and evaders, we may also want a simple model of network congestion.

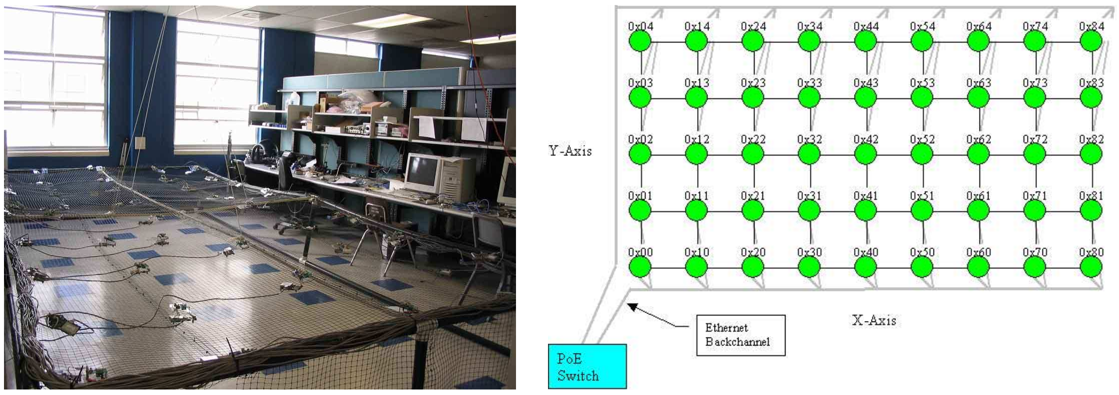
We hope that providing performance guarantees on pursuit-evasion games over sensor networks will lead to insight on how to provide performance guarantees on general sensor network control applications. These performance guarantees can then aide in the design and deployment of sensor networks.

Chapter 4

Sensor Network Testbeds

To make a connection between theoretical models/simulations of sensor networks and reality, it is necessary to set up sensor network testbeds to verify that the modelling assumptions make sense. This chapter describes the setup of an indoor sensor network testbed with robots and an outdoor, large-scale, long-term deployment testbed. The construction of both testbeds was a combined effort of many individuals (see the Acknowledgments section).

One of the main thrusts behind these testbeds was to implement a multiple-target tracking (MTT) algorithm on the sensor network and study/verify its performance. The multiple-target tracking problem is the problem of associating detection events with the targets that generate them. Target disambiguation is part of the estimation step of a feedback control loop involving the pursuit of targets. This problem has been studied extensively by Songhwai Oh [39]–[41], especially in the context of receiving detection events over sensor networks [32], [42]. [31] shows how the output of a multiple-target tracking algorithm can serve as input to a controller for a pursuit-evasion game. For a brief background on MTT and particularly Songhwai's implementation that was tested on our testbeds, the Markov chain Monte Carlo data association (MCMCDA) multiple-target tracking algorithm, please see Appendix G.


0x04 0x14 0x24 0x34 0x44 0x54 0x64 0x74 0x84
0x03 0x13 0x23 0x33 0x43 0x53 0x63 0x73 0x83
0x02 0x12 0x22 0x32 0x42 0x52 0x62 0x72 0x82
0x01 0x11 0x21 0x31 0x41 0x51 0x61 0x71 0x81
0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80
Y-Axis
X-Axis
Ethernet Backchannel
PoE Switch
Figure 4.1. (left) The indoor robot sensor network testbed. (right) The layout for the deployment, with the lower left hand corner being the lower right hand corner in the picture.

4.1 Indoor Robot Sensor Network Testbed

A 45 node indoor testbed was designed and deployed for the purpose of studying control and tracking of robots using sensor networks. This sensor network was deployed in a 9×5 grid with 2.5 ft spacing (10 ft by 20 ft total) on a plastic mesh netting with room below for robots to navigate without colliding into the nodes of the network (see Fig. 4.1). The frame holding up the sensors is constructed of PVC piping and does not contain metal since we are using magnetometer sensors. The nodes have an ethernet backchannel for programming and for monitoring the state of the system, but communicates over the wireless radio for the sensor network applications just as it would in a real world deployment. Furthermore, the nodes can be perpetually powered by the ethernet backchannel since the channel follows the Power-over-Ethernet standard, obviating the need to replace batteries.

4.1.1 Hardware Platform

The sensor node platform used for our testbed is the mica2dot [43], an embedded, low-power, wireless platform running TinyOS [44]. The mica2dot has a 4 MHz ATMega128 processor with 4 kB of RAM and 128 kB of program memory, and a CC1000 radio that provides up to 38.4 kbps transmission rate. Each node is equipped with a Honeywell HMC1002 2-axis magnetometer to sense the moving targets. This is the platform that was

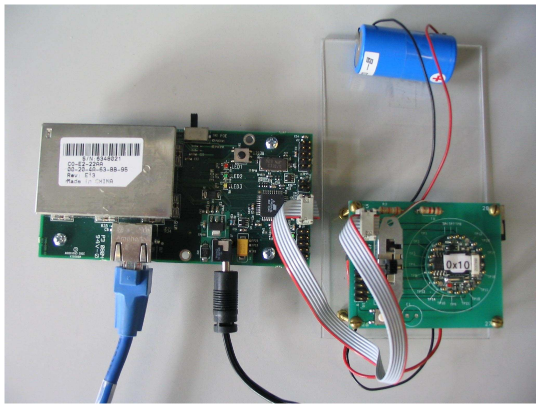


Figure 4.2. eMote (on the left) and assembled mica2dot mote with hardware adapter board (on the right). The hardware adapter board is the large rectangular printed circuit board, and blocks from view the magnetometer and power conversion boards that are underneath. The eMote runs a web server and is assigned a static ip address for access from computers connected to our Power-over-Ethernet switch.

developed and used in the NEST midterm demonstration [45]. A hardware adapter board was developed for it to operate while wired to the ethernet channel.

The assembled sensor nodes are depicted in Figure 4.2. The hardware adapter board has four main functions. First, it exposes the essential pins to interface with the eMote (Ethernet Programming Board) which allows monitoring data to be sent back over the ethernet and for programs to be downloaded onto the motes. The original design could not connect to both the eMote and the power conversion board. The power conversion board is necessary to provide 5 volts to the magnetometer from the 3.3 volt eMote power source. Second, it exposes an interface that allowed the next generation of motes, the Telos mote [46], to interface with the mica2dot sensor boards should we choose to change platforms. Third, it allows for the sensor node assembly to be easily disconnected from the wired interface and run off a battery without reassembly. Lastly, the board exposes a reset and off switch for the mica2dot motes. For a schematic of the hardware adapter board, see Appendix B.

The robots used for tracking and control were COTBOTs, remote control cars controlled by mica2 sensor nodes that are designed by Sarah Bergbreiter [18] (see Fig. 4.3). The mica2 nodes could communicate wirelessly with both the mica2dot sensor nodes and with a basestation computer using the same protocol. For sensing the robots, we strapped a a 1”

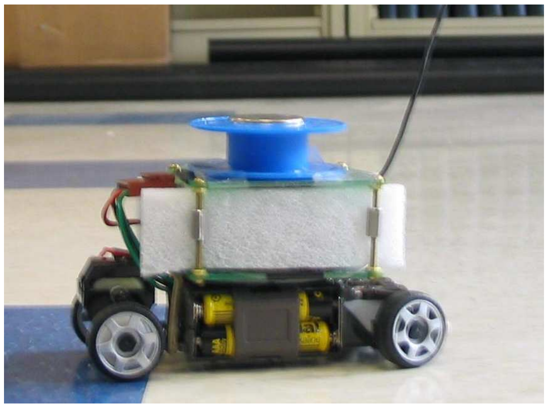


Figure 4.3. COTSBOTS with a magnet on top in a plastic holder. The mica2 mote is hidden behind the white shock absorbers. We also placed magnets atop styrofoam columns on the COTSBOTs to bring them closer to the sensors and to separate it from the electronics. The robots are about 6" long and 3" wide.

diameter, 0.125 inch thick neodymium magnet to the top of the car. The magnetometers on the mica2dot sensor nodes detect distortions in the magnetic field whenever the robots drive by [47].

4.1.2 Software

The software running on the sensor nodes are written in NesC [48] and run on TinyOS [49], an event-driven operating system developed for wireless embedded sensor platforms. Besides providing a nice framework for writing drivers for hardware components on the underlying system, TinyOS also allows users to compose an application from a large library of modular components, including routing layers, time synchronization, power management policies, and others.

For the indoor robot testbed, a software environment/interface was written to allow MATLAB robot controller code to be used to control the COTSBOTs (see Appendix A for references to the code repository). The idea is to do rapid controller development in MATLAB and test the controller on the COTSBOTs by sending control commands over the network. The environment relies on software interface code written by Kamin Whitehouse to bridge MATLAB and the Java communication libraries for TinyOS. It provides basic functions for command line control and calibration of the COTSBOTs, logging and display of data from the sensor network, and a framework for running multiple MATLAB sensor network applications simultaneously and sharing data structures. For instance, the environment provides commands for simultaneously connecting, disconnecting, logging data, and plotting data of multiple sensor network MATLAB applications written following a standard format. This has been a rather useful tool for interacting with the sensor network in real time during experiments.

4.1.3 Sample Applications

A few sample applications have been developed on this testbed. The first was an application to test the sensors which lit up the LEDs on the sensor motes when the magnetometer signal crossed threshold, leaving a trail of lights behind a moving COTSBOT carrying a magnet. The second was an application to verify the MCMCDA MTT algorithm on a small scale testbed before applying it to a larger outdoor testbed.

The tracking scenario was that of two crossing targets moving at constant speed. Due to the small physical size of the network, there is a single supernode and hence no track-level data association.[1] Also, to retain enough distinct data points to form meaningful tracks on our small testbed, we did not aggregate sensor readings.

The sensor nodes trigger a detection event when

$$\sum_{i=t-W+1}^{t} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} > \eta, \tag{4.1}$$

where t is the current detection time, x_i and y_i are the x and y axis magnetometer readings, and W is the window size. We chose this detection function because of its low computation requirement and robust performance, despite a tendency for the bias to drift on the magnetometers (see [47] for more details). We used $W = 5$ and $\eta = 80$ in our experiments.

We used the Minimum Transmission routing protocol [33] to route the sensor readings back to a base station for calculations. This algorithm uses link estimation to choose routing topologies that give good end-to-end reliability.

We performed multiple tests. The results of one is shown in Figures 4.4 and 4.5. The

[1]See [32] for a discussion on supernodes and track-level data association.

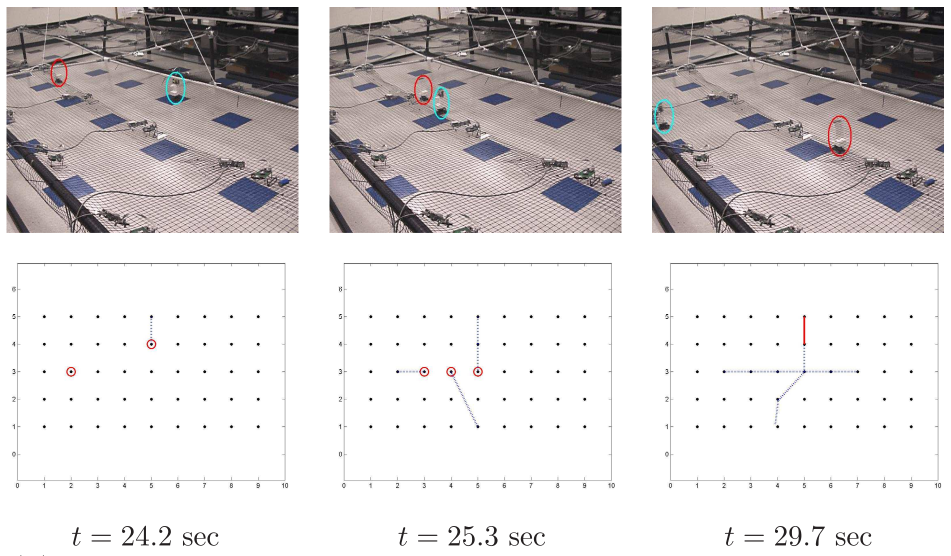


Figure 4.4. (top row) Snapshots from the experiment of crossing tracks, proceeding left to right. We are looking at the grid of nodes from the bottom right corner of the testbed. (bottom row) Corresponding tracks found by the MCMCDA algorithm at each time step. Reporting sensor nodes are circled in red, postulated tracks are represented by dashed lines, while established tracks are represented by red solid lines. Note that the algorithm makes a bad association at $t = 25.3$ sec, but corrects for it by $t = 29.7$ sec.

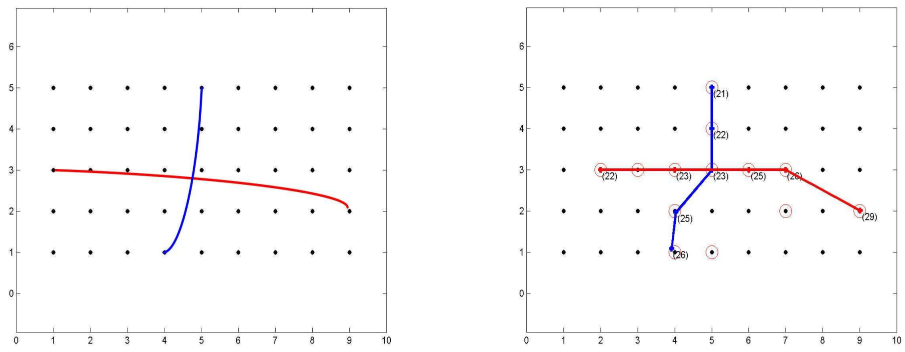


Figure 4.5. (left) Trajectories of targets from the experiment. (right) Final established tracks from the MCMCDA algorithm with all observations superimposed. There were some false alarms and missing observations during our experiment, as well as nearby sensings, which were not in the sequences shown above.

experiments showed that the algorithm performed well in the presence of false alarms and missing detections.

The next application to develop for this testbed is a controller that uses the sensor network readings to help steer the COTSBOT. As with most robots, it is very difficult to accurately calibrate the steering angles on the COTSBOT and its speed, making it difficult to steer a robot using dead reckoning. While the magnetometers provide rather low resolution on the position of the robot, we hope to see whether a robot can repeatedly trace a circle using just the sensor network to sense its position. This is a critical step before running simple pursuit-evasion games on this testbed.

4.2 Outdoor Sensor Network Testbed

The next step after verification of the MCMCDA multiple-target tracking algorithm on the indoor testbed was to verify the MCMCDA MTT algorithm on a larger, long term, outdoor deployment which would more closely resemble the simulations in [32]. The remainder of this chapter describes the general infrastructure and setup of the large scale testbed, as well as how it was configured for the multiple-target tracking experiment. Chapter 5 describes the challenges and results of a large scale demonstration of the multiple-target tracking algorithm to a live audience.

4.2.1 Background: Sensor Network Infrastructure

The outdoor testbed was deployed at the Richmond Field Station (RFS) of UC Berkeley. The sensor network infrastructure for the RFS testbed was a composition of many research platforms and services primarily designed by professor David Culler's students. For details beyond the descriptions below, see [50].

4.2.1.1 Hardware Platform

A group of computer science students designed the *Trio* sensor nodes used for the outdoor deployment [51]. The *Trio* node is a combination of two components from previous designs: a *Telos B* mote [46] and a *Trio* sensor/power conditioning board based off of the sensor board for the eXtreme Scaling Mote (XSM) [52] and the Prometheus solar power system [53]. Figure 4.6 shows the components and the assembled Trio node in a weatherproof casing.

The Telos B mote is the latest in a line of wireless sensor network platforms developed by

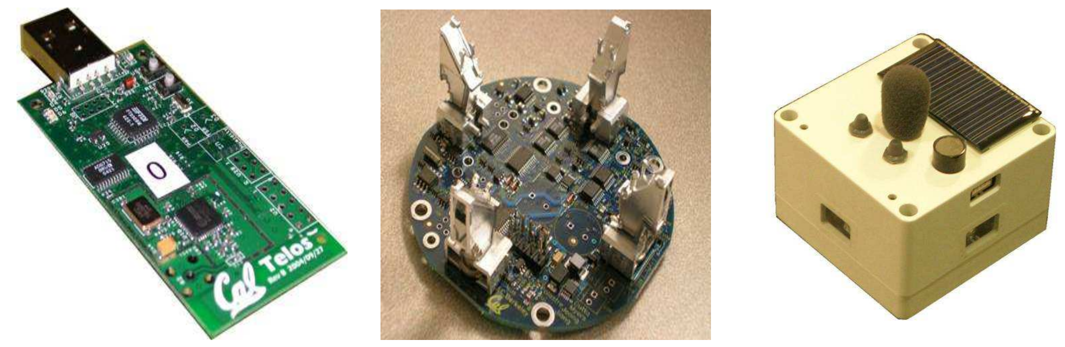


Figure 4.6. (left) Telos B. (middle) Trio sensor board. (right) Assembled Trio node. On top is the microphone, buzzer, solar panel, and user and reset buttons. On the sides are the windows for the Passive Infrared Sensors and a USB port.

UC Berkeley and Moteiv Corporation for the Networked Embedded Systems Technology (NEST) project. It features an 8MHz Texas Instruments MSP430 microcontroller with 10kB of RAM and 48kB of program flash and a 250kbps, 2.4GHz, IEEE 802.15.4 standard compliant, Chipcon CC2420 radio. The Telos B mote provides lower power operation than previous COTS (commerical off the shelf) motes (5.1 μA sleep, 19 mA on) and a radio range of up to 125 meters, making it the ideal candidate for large-scale, long-term deployments.

The Trio sensor board includes a microphone, a piezoelectric buzzer, x-y axis magnetometers, and four passive infrared (PIR) motion sensors. In addition, it contains power conditioning and solar-power charging circuitry to enable long-term deployments without changing batteries.

4.2.1.2 Software Services

Similar to the indoor robot testbed, the software running on the sensor nodes run on TinyOS and are written in NesC. The event detection application running on the nodes rely on software services developed by David Culler's students. These services include network reprogramming [54], a set of services allowing an interactive command line environment to quickly reconfigure parameters on the nodes [55], routing for dissemination of commands and collection of data [56], and a multi-moded event generator for testing and detection.[2]

[2]I developed this module from a code base provided by Cory Sharp and Gilman Tolle.

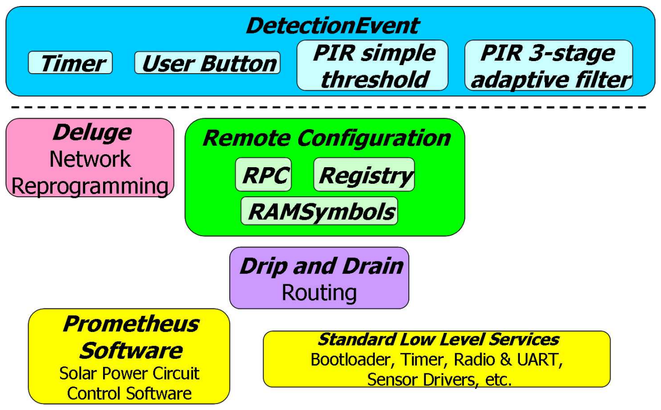

DetectionEvent
Timer
User Button
PIR simple threshold
PIR 3-stage adaptive filter
Deluge Network Reprogramming
Remote Configuration
RPC
Registry
RAMSymbols
Drip and Drain Routing
Prometheus Software Solar Power Circuit Control Software
Standard Low Level Services Bootloader, Timer, Radio & UART, Sensor Drivers, etc.

Figure 4.7. Core sensor network software services used by the event detection application for multiple-target tracking. Other services available on the platform that were not used are not displayed. *Detection-Event* can also be included in other applications as a service/module, despite its depiction here as the main application (see Sec. 4.2.2.2).

(see Fig. 4.7). Unfortunately, due to code size limitations, the final version of the software did not incorporate, *Nucleus* [56], a lightweight network management layer that allows a user to make queries on the network status, although it was helpful during the development cycle. A summary of the highlights of the services is given below.

The network reprogramming tool, *Deluge*, allows the user to disseminate program images wirelessly over the network. It also allows multiple program images to be stored in flash memory so a reboot command can be sent over the network to switch between programs. Furthermore, Deluge provides a *Golden Image* slot that cannot be overwritten by a wirelessly propagated image, allowing nodes to revert to a guaranteed correct image without reprogramming the node should there be an operator error programming wirelessly. This tool was used during the development and testing cycle to share the deployed network with other users running other programs.

The command line interface for interacting with the sensor network, *PyTOS*, allows for remote procedure calls (RPC), setting and querying parameters in an application (Registry and RamSymbols), and scripting data processing and display functions for incoming data using Python. The tool was heavily used during the experiment to quickly change event generation modes, tune detection parameters, and check on the network status. Useful scripts were written to display node battery and capacitor voltages and set node locations during development and testing.

The routing layers consist of *Drip* for disseminating of commands to the network and *Drain* for collecting data from the network. Both protocols were written for the Nucleus Management System and have been adopted to both send commands from PyTOS to the network and to collect detection reports from the network. They are relatively low-bandwidth and robust to communcation link failure, dynamically rebuilding the routing tree as necessary.

4.2.2 Instrumenting the Multiple-Target Tracking Application

Following the setup in [32] for a hierarchical multiple-target tracking application, the implementation consists of two main components: the event detection application on the sensor nodes and the multiple-target tracking algorithm centralized at the basestation. The use of “supernodes”, sensor nodes with more computational power and longer radio ranges, to compute tracks on subregions so only track-level associations are necessary at the basestation was not implemented. The targets to be tracked were people walking through a field.

4.2.2.1 Sensor Characterization

For the multiple-target tracking application, Michael Manzo and I found the passive infrared (PIR) sensors to be most effective for sensing human subjects moving through a sensor field. We ran tests on the XSM mote before the Trio mote was ready, expecting the same results because the XSM mote had the same sensing circuit and sensors. The acoustic sensor is inadequate because it requires intensive signal processing and a consistent acoustic signature from human subjects to be effective. Furthermore, the wind generates significant noise in the acoustic signal, and our deployment location often had strong gusts of wind. The magnetometers have difficulty detecting human subjects carrying rare earth magnets (Five 1” diameter 0.125” thick disk magnets) and large quantities of iron (5 large crowbars) at distances beyond 2 meters, with weak signals for large quantities of iron even at a distance of 1 meter.

The PIR sensors provide an effective range of approximately 8 meters, with sensitivity varying depending on weather conditions and time of day. Unfortunately, the variability in the signal strength of the PIR sensor reading prohibits easy extraction of ranging information from the sensor, and we were relegated to use PIR sensors as binary detectors. Figure 4.9 shows one of the plots mapping the PIR detection rate/signal strength as a function of position relative to the sensor. Although it is difficult to see from Figure 4.9, the configuration of the PIR sensors on the XSM and on the Trio nodes makes detection better on the sides of the mote than on the corners.

4.2.2.2 The Event Detection Application

The event detection application consists primarily of the *DetectionEvent* module, a multi-moded event generator, wired to the Drip and Drain routing services to send reports back to the basestation. The module provides four modes of event generation – events generated periodically by a timer, events generated by pressing a button on the mote, events generated by the raw PIR sensor value crossing a threshold, and events generated by a three-stage filtering, adaptive threshold, and windowing detection algorithm for PIR sensor readings developed by the University of Virginia (UVa) [57]. The timer and user-button generated events serve as diagnostic and testing tools to tell which nodes are alive and running the proper program image after network reprogramming. For simplicity, no data aggregation or in-network processing (except when using the UVa filtering algorithm) is performed on the sensing readings. The raw sensor reading of the simple threshold detector or a "confidence" value for the filtering and adaptive threshold detector (see [57] for details) is included with each detection event report. See Appendix A.2 for the reference to the code repository for this application.

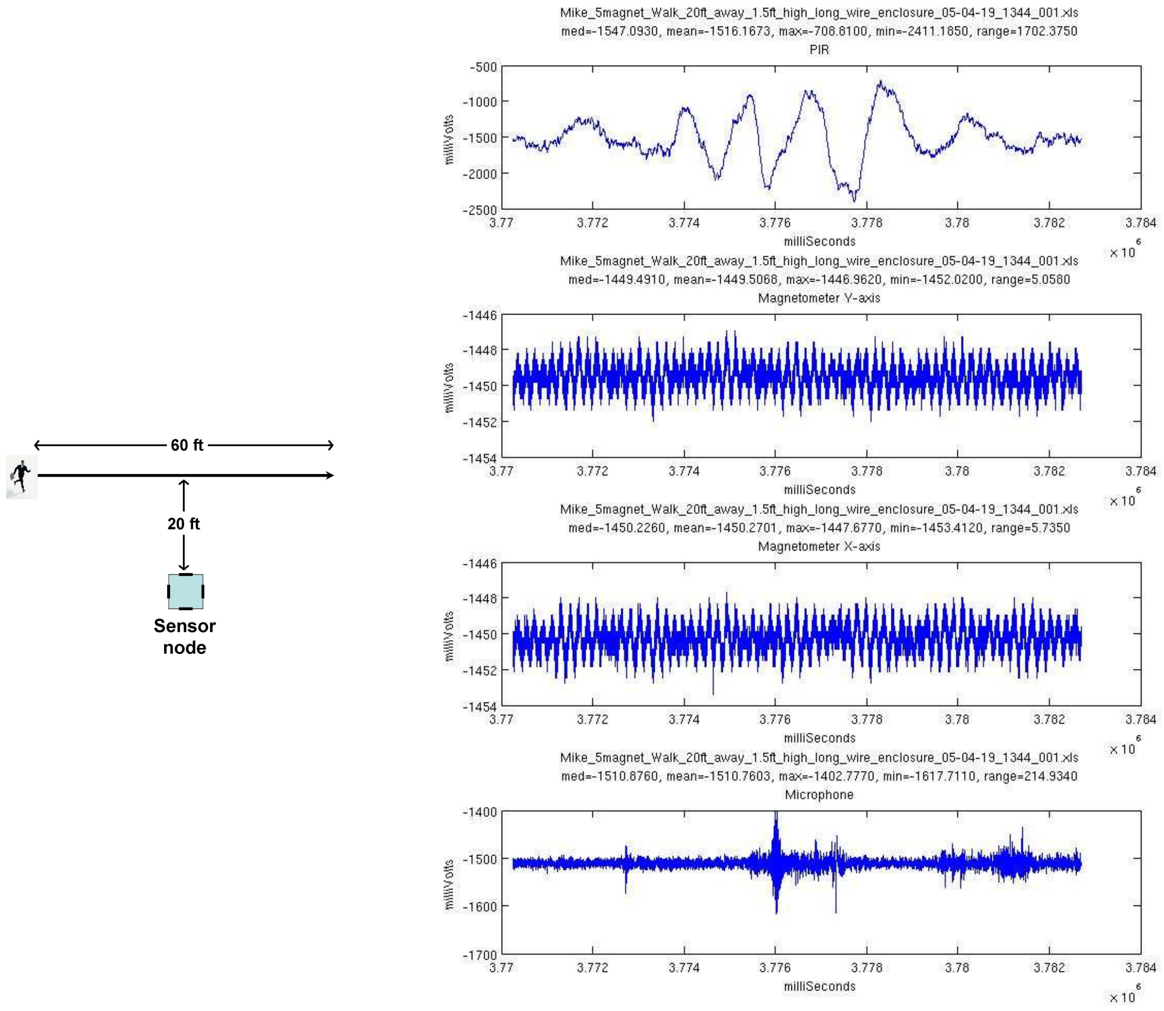


Figure 4.8. (left) Raw sensor readings experiment setup. The human target was walking and carrying 5 rare earth magnets in his pocket. (right) Raw sensor readings. (top) PIR sensor signal; (top-middle) Y-axis magnetometer signal; (bottom-middle) X-axis magnetometer signal; (bottom) Acoustic signal. The acoustic signal was used for ground truth. The target would shout at the PIR sensor as he was walking by.

4.2.2.3 Integration with the MCMCDA Multi-Target Tracking Algorithm

The MCMCDA multiple-target tracking algorithm used in the simulations for [32] was implemented in MATLAB and C++. The same codebase was run in real time for the implementation on the RFS testbed, with some modifications. The data from the sensor network is relayed through a standard TinyOS Java application called SerialForwarder to MATLAB. The data is then timestamped in MATLAB and stored in a matrix, which is polled periodically by the MTT algorithm for calculations. The data in the matrix can also

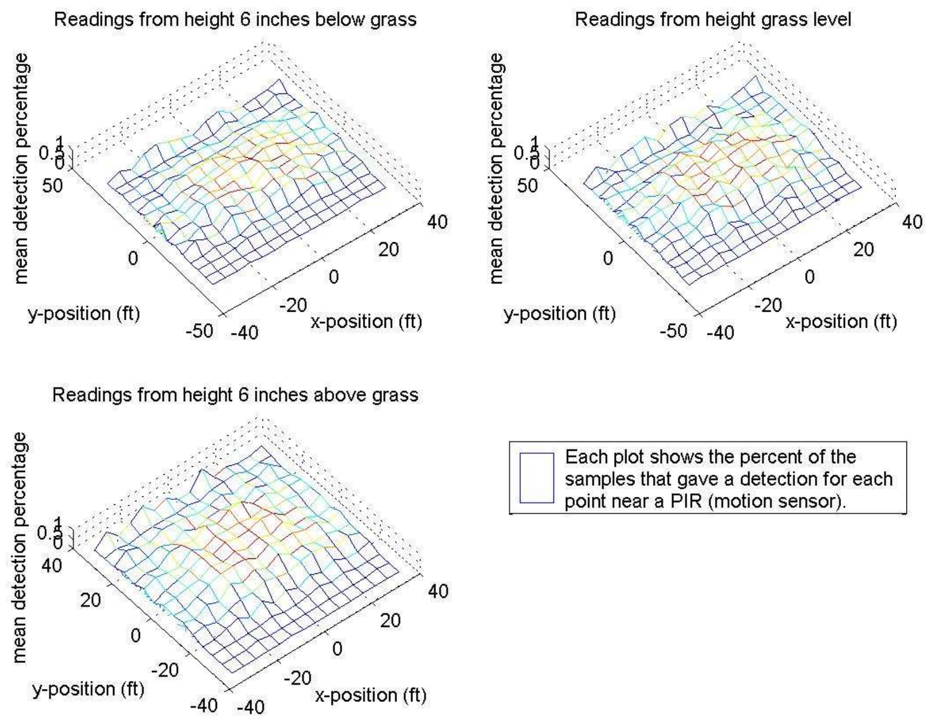


Figure 4.9. Detection rate for positions surrounding a sensor node. The detection rate is related to the sensor signal strength. All readings were taken in a meadow with a median grass height of approximately 16 inches. The results are separated into 3 plots, one for each height of the sensor with respect to the grass height. (This figure is best viewed in color.)

be saved for replaying the tracking scenario later and tuning the MTT algorithm. This setup is similar to that used for the indoor robot testbed.

Two new simple visualization tools in MATLAB were developed to help test and debug the event detection application and the integration with MATLAB. One is an oscilloscope type of application displaying the intensity of sensor readings. The other reads in a configuration file specifying node locations and displays a map of disks representing detection reports, with color representing event type and size/line thickness representing the intensity of the sensor reading/how recently we heard the report (Fig. 5.2 was generated from this tool). These tools are part of the code repository referenced in Appendix A.2.

To overcome the difficulty of using the PIR sensor as a binary detector, Songhwai developed a multi-sensor fusion algorithm to find the maximum likelihood position of a target given sensor readings from a set of nodes using spatial correlation (see Appendix G for details). The fused detections are used by the MCMCDA multiple-target tracking algorithm. Since we do not know the number of targets in advance, the fusion algorithm can provide incorrect and inconsistent position reports to the tracking algorithm. However, the inconsistency in position reports are later fixed by the tracking algorithm using temporal correlation.

The next chapter describes a demonstration of the multiple-target tracking algorithm on this testbed.

Chapter 5

Demonstration of Multiple-Target Tracking on Sensor Networks

On August 30, 2005, UC Berkeley had a large demonstration for the conclusion of the Networked Embedded Systems Technology (NEST) project funded by DARPA. The project was a collaborative effort of over 15 researchers and staff members that showcased the latest state of the software and hardware technologies as well as the theoretical contributions for modelling and designing applications to run over sensor networks (see the Acknowledgements Section for the list of participants). The demonstration was held out at the Richmond Field Station on a sunny day around 11 AM - 12 PM. The main application demonstrating the integration of all the hardware, software services, and theory was the multiple-target tracking demo.

5.1 Experimental Setup

The deployment on the day of the demo consisted of 557 nodes. 144 of these nodes were sectioned off for the multiple-target tracking experiment (see Fig. 5.1). Of the 144 nodes, 6 nodes were faulty on the day of the demo (see Fig. 5.2). The nodes were approximately placed on a 12×12 grid with 5 meter spacing. Their actual locations were obtained via

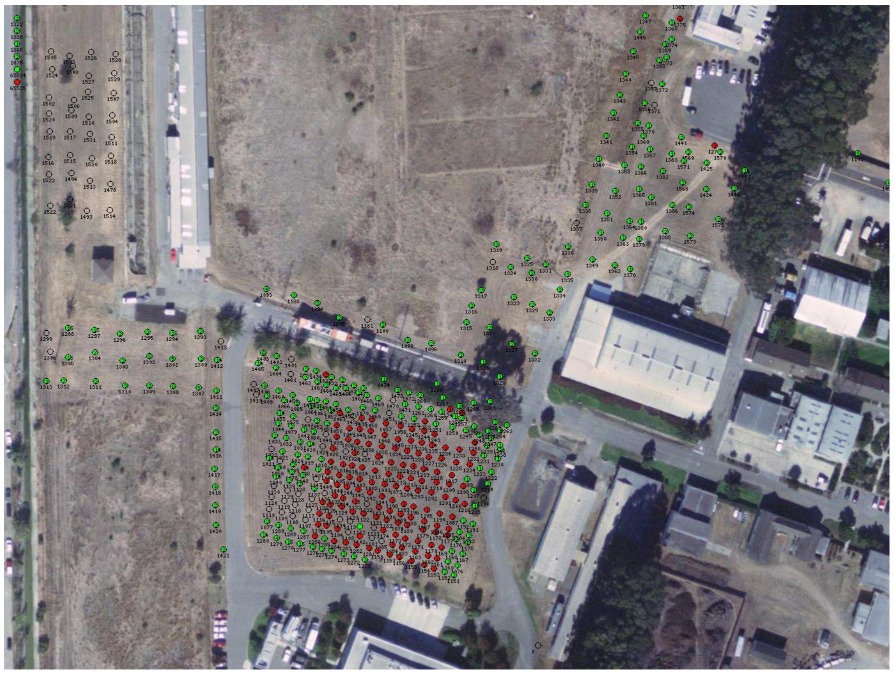


Figure 5.1. Map of the main section in the Richmond Field Station deployment. The nodes in red are mostly those that have been partitioned from the network. The nodes in green are those that have communicated back to the base station recently. (Picture courtesy of Gilman Tolle. This figure is best viewed in color.)

GPS[1], as displayed in Figure 5.1. During the demonstration, we did not use the nodes' actual GPS locations but instead assumed they were placed on an exact 5 meter spacing grid. Despite this, the tracking algorithm was robust enough to accurately track the targets.

The tracking experiment was held on a short grass field, mowed down to about 3 inches in height. Each node was elevated about 2 feet off the ground via tripods to prevent obstruction of the PIR sensors by grass and uneven terrain. The nodes were oriented in the same direction for maximum solar panel exposure throughout the day and for consistency in laying out the sensing regions. See Figure 5.3 for a photo of the experiment setup.

The gateway to the sensor network was a mote connected to a personal computer, marked by *TOSBase* in Figure 5.4. For the purposes of displaying the application to an audience sitting outside the sensor field, the personal computer routed the data packets back to a laptop near the audience via ethernet. The laptop then timestamped the returning

[1]0.4 m accuracy at one sampled location

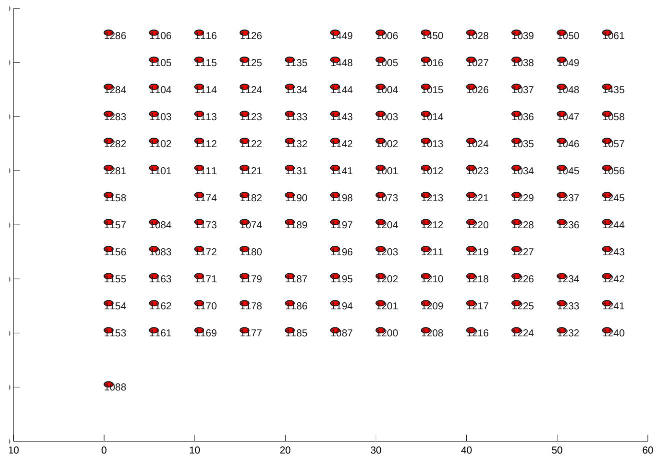


Figure 5.2. Map of live sensors in the 144 node deployment for testing the multiple-target tracking algorithm. 6 sensors were dead. (The node in the lower left hand corner is plotted incorrectly and should be inside the field, 5 grid spaces right and 4 grid spaces up from the lower left hand corner of the deployment.)

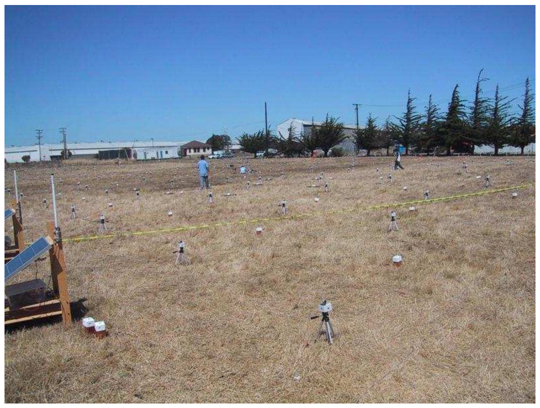


Figure 5.3. Sensor deployment for the MTT demo. (The XSM nodes on wooden blocks and foam pads are part of another concurrent demonstration.)

packets, ran the multi-target tracking algorithm in MATLAB, and displayed the results on a large screen.

Due to time contraints and the complexity of the experiment, some simplifications were made to the demonstration. Rather than use the three-stage PIR detection code provided by the University of Virginia [57], we chose instead to use a simple threshold crossing detector. While this meant that the threshold on the motes needed to be manually tuned every half hour for good performance, it was simple, performed well, and was easy to diagnose if any system parameters were set wrong. Also, the solar-powered charging system was not used

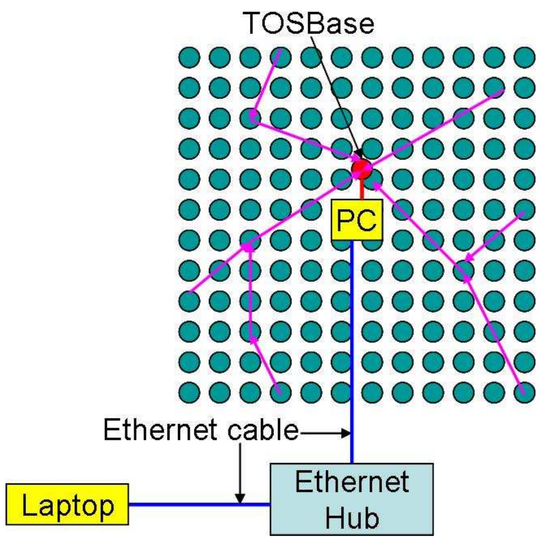


Figure 5.4. 144-node sensor network setup for the multi-target tracking demo on August 30, 2005.

to power the nodes during the multi-target tracking demo. Instead, all the nodes were powered via USB cables. Nonetheless, all the communication was wireless.

5.2 Experiments

Four types of tracking experiments were demonstrated: The tracking of one target, the tracking of two crossing targets, the tracking of three crossing targets, and the tracking and simulated pursuit of two targets. All targets were walking humans. In all four scenarios, the tracking algorithm was successful. For an overview of the MCMCDA MTT algorithm, see Appendix G.

In the three target tracking experiment, the targets entered the field at different times and two targets crossed paths. This demonstrated the algorithm's ability to dynamically estimate the number of targets and its ability to disambiguate targets at crossing points using linear models of the targets' dynamics. Also, one could see the tracking algorithm correct previous track hypotheses as it received more detections. Furthermore, it demonstrated that the routing protocol and infrastructure was able to deliver detection events to the basestation with consistently low latency such that the receive-side timestamps on the data

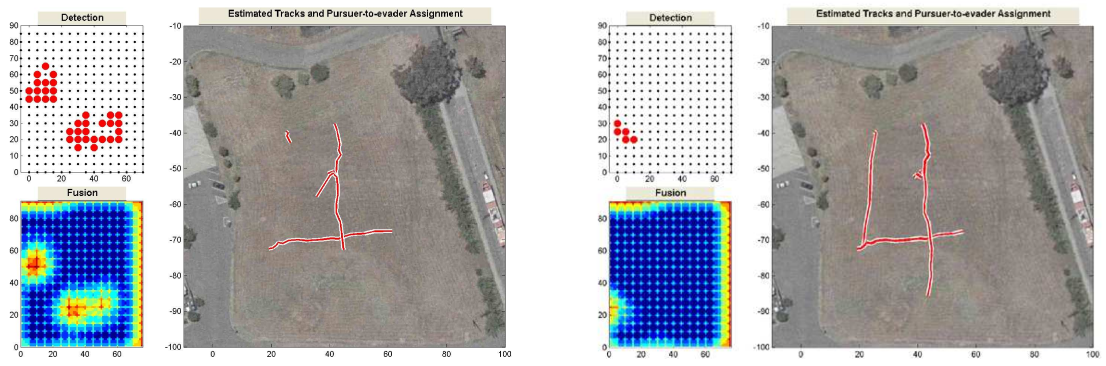


Figure 5.5. Estimated tracks with three people walking in the field during (left) and near the end (right) of the experiment. Note from comparing the two displays that track hypotheses are corrected later as more detection events arrive. *Explanation of the Display*: (upper left) Detection panel. Sensors are marked by small dots and detections are shown as large disks. (lower left) Fusion panel showing the fused likelihood. (right) Estimated Tracks panel showing the tracks estimated by MCMCDA. (This figure is best viewed in color.)

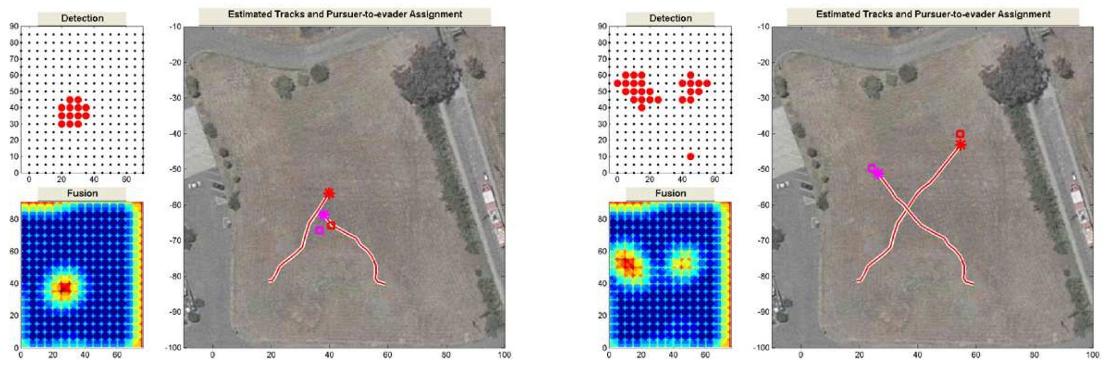


Figure 5.6. Two targets pursued by two simulated pursuers before and after crossing paths. The pursuer-to-evader assignment panel on the right shows the estimated evader positions as stars, their estimated tracks, and the pursuer positions as squares. Pursuer-evader assignment pairs are denoted by matching colors. (This figure is best viewed in color.)

packets were accurate enough for the MTT algorithm. Figure 5.5 shows the multi-target tracking results with three people walking through the field.

In the two target simulated pursuit demo, we had two live crossing targets pursued by two simulated pursuers. This was a demonstration of the pursuit-evasion game described in [31]. Each pursuer was running a robust minimum-time-to-capture control law and the pursuer-evader assignment minimized the total expected capture time, i.e. the time to capture the last evader. The controller and the tracking algorithm were all running in real time. The results are shown in Figure 5.6.

Experiment	Duration (sec)	Number of Reporting Nodes	Number of Reports
1 target	53.5	44	293
2 target	138.5	99	992
3 target	116.4	87	1001
2 target pursuit	136.5	91	819

Table 5.1. Multple Target Tracking Statistics from the Demo on 8/30/2005.

5.3 Preliminary Analysis

Although the demonstration was not instrumented to carry out careful experiments, we were able to look at a log of the packets received at the TOSBase and extract some information about the temporal nature of the observations coming from the sensor network. Table 5.3 shows that in our experiments running under 2 minutes, we got on the order of 300 to 1000 packets with our simple threshold detector.

To get a better sense of the variability of the data rates over time, we made a raster plot of the arrival of packets over time for the single target tracking demo in Figure 5.7. Note that the data comes in bursts, though the average report rate appears to be around 10 packets a second. In the three target tracking demo, it appears that the average report rate reachs around 20 packets a second, as shown in Figure 5.8. However, it is very clear that at one point during the tracking experiment, there is a large spike in the number of packets. This was observed to be the case in the other two target tracking demos as well (See Appendix C.2 for more plots).

A reasonable conjecture is that this spike in traffic only occurs during target track crossing. This was the case for the 2 target and 3 target tracking demos, but in the 2 target pursuit demo the spike in traffic occured slightly after track crossing. Nonetheless, the plots do seem to suggest that the channel goes silent shortly before the flood of reports, which would indicate the congestion in the network is likely to be the problem. More experiments need to be done to confirm this. We would like to timestamp at the basestation mote to make sure that this is not because the processor on the laptop is being overloaded, forcing MATLAB to timestamp the packets too late.

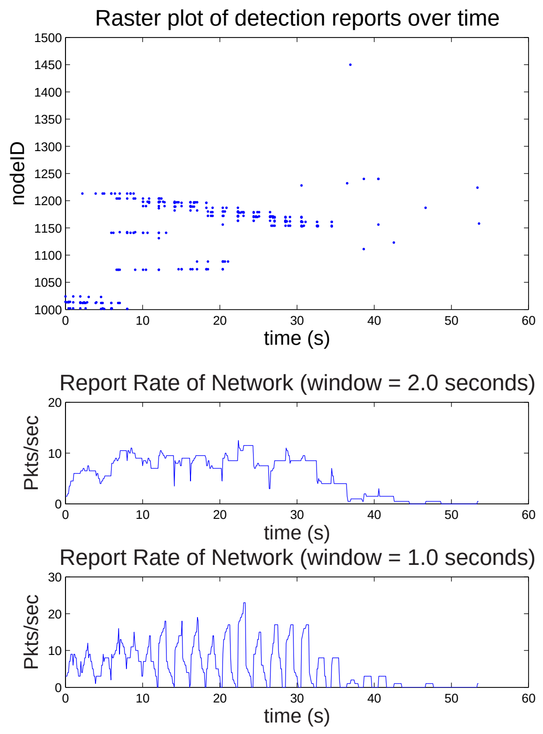


Figure 5.7. Raster plots of reports over time by node ID, and the corresponding report rates for the single target tracking demo. Report rates at time t are calculated from the number of reports in the window *before* time t.

Note that as illustrated in the left plot in Figure 5.8, given our simple threshold detector we often get 6, 7, or sometimes even 10 reports of a detection per node as a target walks by. This is expected, as the PIR sensor signal is a sinusoid and we report each time it crosses the threshold. The implementation of the multiple-target tracking algorithm handles this by grouping together the detections from one node at a specified time interval and treating it as one detection. A more sophisticated detection algorithm such as the 3-stage adaptive threshold filter by the University of Virginia would quell reports from the sensor for a period of time after detection, though figuring how long to quell reports requires some fine tuning of the algorithm. Quelling reports also removes the redundancy of sending multiple reports, making the reliability of the routing more critical to the performance of the tracking algorithm.

In [58], the performance of the multiple-target tracking algorithm was analyzed in simulation with respect to sensing range, sensor localization error, transmission failure, and

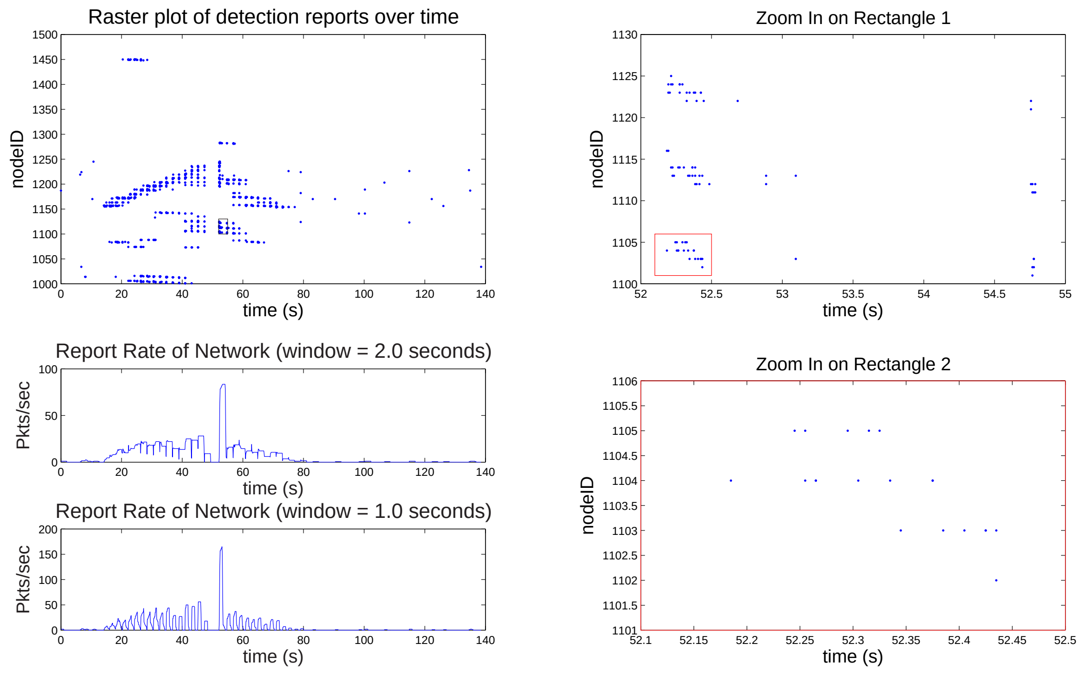


Figure 5.8. (left) Report raster plots and report rates for the three target tracking demo. (right top) Zoom in on black rectangle in left raster plot (right bottom) Zoom in on red rectangle above. (This figure is best viewed in color.)

communication delays. To run a similar analysis on the multiple-target tracking implementation, we would have to know the true position of the targets at each point in time. Also, the sensing range for the sensor nodes can be emulated by raising or lowering the detection threshold. There is no need to measure performance with respect to sensor localization error because sensors in this experiment did not get their positions from a localization algorithm, but from hand placement and an assumption that they lay on a grid. To measure the transmission success rate while tracking targets, we need to add sequence numbers to the detection packets. Finally, to get a sense of the latency of the network, we need to ping the network during the tracking experiment.

Chapter 6

Conclusion

The field of sensor networks has advanced a lot in the last few years. However, for it to become a reliable infrastructure for high performance control systems, we must develop better models and abstractions for treating a deployed sensor network as a whole. This requires an iterative approach of modelling the network and extracting performance parameters for use by controllers, and then designing controllers to exploit the information in these parameters. In the process, we will run into difficulties in modelling the network and designing the controller (like nonconvexity) that may require us to rethink our assumptions and approaches.

The first two chapters were attempts at integrating a sensor network connectivity parameter into an optimal control path planning framework and providing loose bounds on a simple lifeline pursuit-evasion game played over a sensor network. The last two chapters described the physical implementation of sensor network testbeds and a multiple-target tracking algorithm for sensor networks to get a sense of real sensor network performance. There are many opportunities to extend the work in these areas.

More work is needed on the modelling side for a better model of sensor network data patterns and its relation to the physical placement of sensors and choice of routing schemes. Furthermore, some applications may be able to exploit data aggregation and in network processing, so their implications on a sensor network model needs to be studied as well. We also need to develop tighter bounds on the solutions of pursuit-evasion games over sensor networks. For instance, assuming that the network performs everywhere only as well as its worst point results in an overly pessimistic estimation of sensor network performance and should be re-evaluated. Also, simple control experiments involving the indoor testbed need to be run to understand the basic behavior of sensor networks in a control setting. The simplest of these experiments would be to steer a robot using only the sensor network. A multi-target tracking experiment on the outdoor testbed controlling the parameters described at the end of Section 5.3 would help us understand the performance of an estimation application on a large sensor network and point out characteristics that we need to model carefully.

Once we can properly model sensor networks as a communication medium for control applications, we can apply this knowledge to the study of distributed robotics. There, we can consider the issues of adding mobility to the sensor nodes and navigation through a physical environment. We will also be faced with the challenge of decomposing a controller for decentralized computation, and studying its robustness with repect to the quality of the communication channels. While many challenges lie ahead, the potential impact of control over wireless sensor networks and control over wireless sensor and actuator networks/distributed robotics is enormous.

References

[1] D. Culler, D. Estrin, and M. Srivastava, "Overview of sensor networks," in *IEEE Computer, Special Issue in Sensor Networks*, August 2004.

[2] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, "Connecting the physical world with pervasive networks," in *IEEE Pervasive Computing*, vol. 1(1), 2002, pp. 59–69.

[3] S. Kim, "Wireless sensor networks for structural health monitoring," Master's thesis, Univ. of California, Berkeley, 2005.

[4] Y. Zhang and L. Cheng, "Issues in applying wireless sensor networks to health monitoring of large-scale civil infrastructure systems," *Proc. of the Structures Congress and Exposition*, pp. 75 – 86, 2005.

[5] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *SenSys '04: Proc. of the 2nd international conference on Embedded networked sensor systems.* New York, NY, USA: ACM Press, 2004, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031498

[6] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," in *SenSys '05: Proc. of the 3rd international conference on Embedded networked sensor systems.* New York, NY, USA: ACM Press, 2005, pp. 51–63. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098925

[7] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler, "An analysis of a large scale habitat monitoring application," in *SenSys '04: Proc. of the 2nd international conference on Embedded networked sensor systems.* New York, NY, USA: ACM Press, 2004, pp. 214–226. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031521

[8] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in zebranet," in *SenSys '04: Proc. of the 2nd international conference on Embedded networked sensor systems.* New York, NY, USA: ACM Press, 2004, pp. 227–238. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031522

[9] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis, "Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea," in *SenSys '05: Proc. of the 3rd international conference on Embedded networked sensor systems.* New York, NY, USA: ACM Press, 2005, pp. 64–75. [Online]. Available: http://doi.acm.org/10.1145/1098918.1098926

[10] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," in *SenSys '04: Proc. of the 2nd international conference on Embedded networked sensor systems.* New York, NY, USA: ACM Press, 2004, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031497

[11] M. Last, B. Leibowitz, B. Cagdaser, A. Jog, L. Zhou, B. Boser, and K. S. J. Pister, "Toward a wireless optical communication link between two small unmanned aerial vehicles," in *Proc. of the IEEE International Symposium on Circuits and Systems*, vol. 3. Piscataway, NJ, USA: IEEE, May 2003, pp. III–930–3.

[12] B. Warneke and K. S. J. Pister, "An ultra-low energy microcontroller for smart dust wireless sensor networks," in *2004 IEEE International Solid-State Circuits Conference*, Piscataway, NJ, USA, February 2004.

[13] M. D. Scott, B. E. Boser, and K. S. J. Pister, "An ultralow-energy adc for smart dust," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 7, pp. 1123–1129, September 2003.

[14] X10, "X10 home automation systems," http://www.x10.com.

[15] S. Seth, J. P. Lynch, and D. M. Tilbury, "Wirelessly networked distributed controllers for real-time control of civil structures," in *Proc. of the American Control Conference*, vol. 4. Piscataway, NJ 08855-1331, United States: Institute of Electrical and Electronics Engineers, June 8-10 2005, pp. 2946–2952.

[16] P. P. Research, "Structural control networks," http://www.structural-control.at, November 2005.

[17] K. D. Frampton, "Distributed group-based vibration control with a networked embedded system," *Smart Materials and Structures*, vol. 14, no. 2, pp. 307–314, April 2005.

[18] S. Bergbreiter and K. S. J. Pister, "Cotsbots: An off-the-shelf platform for distruted robotics," in *IROS*, October 27-31 2003.

[19] M. B. McMickell, B. Goodwine, and L. A. Montestruque, "Micabot: a robotic platform for large-scale distributed robotics," in *2003 IEEE International Conference on Robotics and Automation*, vol. 2. Piscataway, NJ, USA: IEEE, 14-19 September 2003, pp. 1600–1605.

[20] K. Dantu, M. Rahimi, H. Shah, S. Babel, A. Dhariwal, and G. S. Sukhatme, "Robomote: enabling mobility in sensor networks," in *2005 Fourth International Symposium on Information Processing in Sensor Networks*. Piscataway, NJ, USA: IEEE, 25-27 April 2005, pp. 404–409.

[21] G. A. Kantor, S. Singh, R. Peterson, D. Rus, A. Das, V. Kumar, G. Pereira, and J. Spletzer, "Distributed search and rescue with robot and sensor teams," in *The 4th International Conference on Field and Service Robotics*, July 2003.

[22] L. Navarro-Serment, R. Grabowski, C. Paredis, and P. Khosla, "Millibots: The development of a framework and algorithms for a distributed heterogeneous robot team," *IEEE Robotics and Automation Magazine*, vol. 9, no. 4, December 2002.

[23] J. T. Feddema, C. Lewis, and D. A. Schoenwald, "Decentralized control of cooperative robotic vehicles: theory and application," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 852–864, October 2002.

[24] J. C. Doyle and G. Stein, "Multivariable feedback design: concepts for a classical/modern synthesis," *IEEE Transactions on Automatic Control*, vol. AC-26, no. 1, pp. 4–16, February 1981.

[25] B. Sinopoli, L. Schenato, M. Franceschetti, K. Poolla, M. Jordan, and S. Sastry, “Kalman filtering with intermittent observations,” *IEEE Transactions on Automatic Control*, September 2004.

[26] R. Isaacs, *Differential games; a mathematical theory with applications to warfare and pursuit, control and optimization.* New York: Wiley, 1965.

[27] T. Basar, *Dynamic noncooperative game theory.* London ; San Diego: Academic Press, 1995.

[28] J. P. Hespanha, K. H. Jin, and S. Sastry, “Multiple-agent probabilistic pursuit-evasion games,” in *Proc. of the 38th IEEE Conference on Decision and Control*, vol. 3, 1999, pp. 2432–2437.

[29] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry, “Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation,” *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 662–669, Oct. 2002.

[30] B. Sinopoli, C. Sharp, L. Schenato, S. Schaffert, and S. S. Sastry, “Distributed control applications within sensor networks,” *Proc. of the IEEE*, vol. 91, no. 8, pp. 1235–1246, August 2003.

[31] L. Schenato, S. Oh, P. Bose, and S. Sastry, “Swarm coordination for pursuit evasion games using sensor networks,” in *Proc. of the International Conference on Robotics and Automation*, 2005.

[32] S. Oh, L. Schenato, and S. Sastry, “A hierarchical multiple-target tracking algorithm for sensor networks,” in *Proc. of the International Conference on Robotics and Automation*, April 2005. [Online]. Available: http://www.eecs.berkeley.edu/~sho

[33] A. Woo, T. Tong, and D. Culler, “Taming the underlying challenges of reliable multihop routing in sensor networks,” in *SenSys*, November 5-7 2003.

[34] S. Kim, R. Fonseca, and D. Culler, “Reliable transfer on wireless sensor networks,” in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*. Piscataway, NJ, USA: IEEE, October 4-7 2004, pp. 449–459.

[35] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms.* Cambridge, Massachusetts: MIT Press, 1990.

[36] E. F. Camacho and C. Bordons, *Model predictive control.* London ; New York: Springer, 2004.

[37] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang, M. Nesterenko, R. Shah, S. Kulkarni, M. Aramugam, L. Wang, M. Gouda, Y. ri Choi, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker, “Project exscal [wireless sensor network],” in *Proc. of First IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, ser. Lectures in Computer Science, vol. 3560. Springer-Verlag, 2005, pp. 393–394.

[38] G. Tolle and D. Culler, “Design of an application-cooperative management system for wireless sensor networks,” in *Proc. of the Second European Workshop on Wireless Sensor Networks*, 2005, pp. 121–132.

[39] S. Oh, J. Kim, and S. Sastry, “A sampling-based approach to nonparametric dynamic system identification and estimation,” in *Proc. of the American Control Conference*, June 2004, pp. 873–879.

[40] S. Oh, S. Russell, and S. Sastry, "Markov chain Monte Carlo data association for general multiple-target tracking problems," in *Proc. of the 43rd IEEE Conference on Decision and Control*, Dec. 2004.

[41] S. Oh and S. Sastry, "A polynomial-time approximation algorithm for joint probabilistic data association," in *Proc. of the American Control Conference*, June 2005, pp. 1283–1288.

[42] ——, "Tracking on a graph," in *Proc. of the Fourth International Conference on Information Processing in Sensor Networks*, April 2005.

[43] Berkeley, "Mica2dot hardware design files," http://www.tinyos.net/scoop/special/hardware, 2003.

[44] "Tinyos," http://www.tinyos.net/.

[45] 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 *Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, January 2005, pp. 93–107.

[46] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[47] Honeywell, "1- and 2- axis magnetic sensors," http://www.ssec.honeywell.com/magnetic/datasheets/hmc1001-2&1021-2.pdf.

[48] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: a holistic approach to networked embedded systems," in *ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*, vol. 38, USA, June 9-11 2003, pp. 1–11.

[49] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *ASPLOS-IX*, Cambridge, MA, USA, November 2000.

[50] Berkeley, "NEST final experiment website," http://nest.cs.berkeley.edu/nestfe/index.php/Main_Page, September 2005.

[51] P. Dutta, J. Hui, J. Jeong, S. Kim, C. Sharp, J. Taneja, G. Tolle, K. Whitehouse, and D. Culler, "Trio: Enabling sustainable and scalable outdoor wireless sensor network deployments," in *N/A, submitted*, 2006.

[52] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, "Design of a wireless sensor network platform for detecting rare, random, and ephemeral events," in *Proc. of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[53] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *Proc. of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[54] J. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems*, 2004.

[55] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler, "Marionette: Providing an interactive environment for wireless debugging and development," in *N/A, submitted*, 2006.

[56] G. Tolle, "A network management system for wireless sensor networks," Master's thesis, Univ. of California, Berkeley, 2005.

[57] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, T. He, A. Tirumala, Q. Cao, J. Stankovic, T. Abdelzaher, and B. Krogh, "Lightweight detection and classification for wireless sensor networks in realistic environments," in *SenSys*, November 2005.

[58] S. Oh, L. Schenato, P. Chen, and S. Sastry, "A scalable real-time multiple-target tracking algorithm for sensor networks," University of California, Berkeley, Tech. Rep. UCB//ERL M05/9, February 2005.

[59] J. Liu, J. Liu, M. Chu, J. Reich, and F. Zhao, "Distributed state representation for tracking problems in sensor networks," in *Proc. of 3nd workshop on Information Processing in Sensor Networks (IPSN)*, April 2004.

[60] J. Liu, J. Liu, J. Reich, P. Cheung, and F. Zhao, "Distributed group management for track initiation and maintenance in target localization applications," in *Proc. of 2nd workshop on Information Processing in Sensor Networks (IPSN)*, April 2003.

[61] S. Meguerdichian, F. Koushanfar, G. Qu, and M. Potkonjak, "Exposure in wireless ad hoc sensor networks," in *Proc. of 7th Annual International Conference on Mobile Computing and Networking*, July 2001, pp. 139–150.

[62] D. H. Shim, H. J. Kim, and S. Sastry, "Decentralized nonlinear model predictive control of multiple flying robots," in *42nd IEEE International Conference on Decision and Control*, vol. 4. Piscataway, NJ, USA: IEEE, 9-12 December 2003, pp. 3621–3626.

[63] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2003.

[64] T. Schouwenaars, B. D. Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *European Control Conference*, September 2001.

[65] D. Reid, "An algorithm for tracking multiple targets," in *IEEE Transaction on Automatic Control*, vol. 24(6), December 1979, pp. 843–854.

[66] Y. Bar-Shalom and T. Fortmann, *Tracking and Data Association.* Mathematics in Science and Engineering Series 179 Academic Press, 1988.

[67] T. Kurien, "Issues in the design of practical multitarget tracking algorithms," in *Multitarget-Multisensor Tracking: Advanced Applications*, Y. Bar-Shalom, Ed. Artech House: Norwood, MA, 1990.

[68] S. Oh, S. Russell, and S. Sastry, "Markov chain Monte Carlo data association for multiple-target tracking," University of California, Berkeley, Tech. Rep. UCB//ERL M05/19, June 2005.

[69] I. Beichl and F. Sullivan, "The metropolis algorithm," in *Computing in Science and Engineering*, vol. 2(1), 2000, pp. 65–69.

[70] M. Jerrum and A. Sinclair, “The Markov chain Monte Carlo method: An approach to approximate counting and integration,” in *Approximations for NP-hard Problems*, D. Hochbaum, Ed. PWS Publishing, Boston, MA, 1996.

[71] G. O. Roberts, “Markov chain concepts related to sampling algorithms,” in *Markov Chain Monte Carlo in Practice*, ser. Interdisciplinary Statistics Series. Chapman and Hall, 1996.

Appendix A

Code

A.1 PEGSim: Simulation Code for Path Planning over Sensor Networks

The relevant code can be downloaded from `http://www.sourceforge.net` under the **TinyOS** project. It is under: `tinyos-1.x/contrib/ucbRobo/tools/matlab/PEGSim`

The most important code in the simulator is reproduced below.

A.1.1 Main Simulation Loop

```
function PEGSimMain(SNfile,PEfile,Pctrlrfile)
% The main module for the sensor network simulator.  This is a discrete
% time simulator.  The main loop is structured as follows:
% 1) Check for capture... stop if capture
% 2) Increment Time
% 3) Using the position of the pursuer(s) and evader(s), check if the
%     sensor network triggers any readings.  Calculate the readings.
% 4) Calculate/create the packets to pass into the pursuer
% 5) Calculate the pursuer(s) policy
% 6) Calculate the evader policy
% 7) Update the change in position of the pursuer/evader using the
%     vehicle dynamics and noise model
% 8) Repeat
%
% The module can plot the motion of the pursuer/evader in real time, and
% can also save the trajectory and packet reports.
%
% IMPORTANT FLAGS: ReSimFlag
% ReSimFlag determines whether to resimulate Evader Motion using old
% traces.  This flag is not altered in any of the files called by
% PEGSimMain (unless it does not exist, in which case it is set to false),
% and is meant to be altered by the batch simulation script or the user.
% Note that the value of the flag (1, 2, 3, etc.) choose the control cost
% function.
% * Affected data structures are SN and E/E_precomp.
% * It is the responsibility of the user calling PEGSimMain to save
%   E to Eprecomp before the resimulation
% * Remember to
% Most values are listed for easy access when debugging
global Pctrlr; % pursuer controller structure
global P; % pursuer structure
global E; % evader structure
global Eprecomp; % for resimulation
global SN; % sensor network structure
global T;
global dT;
global history;
global ReSimFlag;
if isempty(ReSimFlag)
    ReSimFlag = 0;
end
Tfinal = 40;
T = 0;
dT = 0.5;
if (nargin == 3)
    PEGSimInit(SNfile,PEfile,Pctrlrfile);
elseif (nargin == 2)
    PEGSimInit(SNfile,PEfile);
elseif (nargin == 1)
    PEGSimInit(SNfile);
else
    PEGSimInit('examples/scen21_25x25_fixedSN.mat',...
               'examples/scen21_25x25_fixedPE.mat');
    %PEGSimInit('examples/nodes100_25x25_2.mat'); % args: SNfile, PEfile
end
while (~checkCapture) && (T < Tfinal) % args: P,E
   T = T + dT;
   %% Perfect Information from Sensor Network
   %    arrTime = T; delay = 0;
   %    packets = [0 E.pos(1:2,end)' 0 T]'; %perfect information on Epos
   %    rcvpkts = [arrTime ; packets];
   [delay packets] = SNSim_ralpha(1); % 1 = NoDelay
%  [delay packets] = SNSim_simple(1); % args: P,E,SN; output: negative delay
                                      % means dropped packet
   rcvpkts = pktQueue(delay,packets); % args: T
   %   plotRoute(rcvpkts);
   history.delay{end+1} = delay;
   history.packets{end+1} = packets;
   history.rcvpkts{end+1} = rcvpkts;
   %PpolicyLQG(rcvpkts); % args: P,T
   PpolicyNonLinOpt(rcvpkts); % args: P,T
%   Epolicy; % args: E,T; For now, no evasive action
   PSimMove; % args: P,T
   ESimMove; % args: E,Eprecomp,T
%   plotStepMotion; % args: P,E
   drawnow;
   disp(sprintf('T = %d',T));
end
%plotSN;
plotMotion;
% IMPLEMENTATION NOTES:
% We use global variables for big data structures like P, E, SN in the
% hopes that not copying (pass by value) means it will run faster
```

A.1.2 Sensor Network and Routing Generation

```
function SN = SNSimInit_ralpha(n,dimX,dimY,rt_num,ob_noise,...
                               alphaS,betaS,etaS,alphaR,betaR,etaR,nodes)
% SNSimInit_ralpha(n,dimX,dimY,rt_num,ob_noise,...
%                  alphaS,betaS,etaS,alphaR,betaR,etaR,nodes)
% Initializes/Generates Sensor Network Data Structures.
% Use testParams to find appropriate eta, beta, alpha's.
% Last argument 'nodes' is optional.
% Sensor Model
% 1) sensors sense (x,y) position
% 2) sensor detection probability: betaS/(betaS+dist^alphaS),
%    truncated when p < etaS; betaS unique to each node
%
% Routing/Radio Model
% 1) Mobile-to-Mobile Routing
% 2) Assumes each sensor node knows how to route to every other sensor
%    node ("ideal", not implementable)
% 3) Symmetric Links
% 4) No modeling of congestion in the sensor network.
% 5) Per Hop Probability of Transmission: betaR/(betaR+dist^alphaR),
%    truncated when p < etaR; betaR unique to each node
% 6) Number of retransmissions: SN.rt_num
% 7) End-to-End transmission probability:
%    Find path with maximum probability of transmission to the node
%    closest to the pursuer.  Then "maximize" the probability of the
%    worst hop using the number of retransmissions on the worst hop
%    (think binomial distribution).  Assume we have SN.rt_num2
%    retransmissions on the last hop.
%
% Output:
%      SN.nodes k*n column matrix
%               [x,y,R_s,R_r]' % R_r is used to compute the transmission
%                                probability for the last hop
%      SN.linkP n*n Matrix with probabilities of successful 1-hop
%               transmission between two nodes.
%      SN.connProb n*n Matrix with probabilities of successful end-to-end
%                  transmission between two nodes.
%      SN.routePath n*n cell array of lists (cell array) of nodes in a
%                   path.  Does not include first node, includes last node.
%      SN.pathMat n*n Matrix with next hop neighbor when transmitting from
%                 node i to node j
%      SN.wtMat n*n Matrix used for calculating the routePath
global SN; % sensor network structure
SN = struct; % clean out junk from before
if ~(abs(etaS) <= 1) || ~(abs(etaR) <=1) % sanity check
    error('1','You''ve mixed up the arguments for SNSimInit_ralpha');
end
SN.n = n;
SN.dimX = dimX;
SN.dimY = dimY;
SN.rt_num = rt_num;
SN.ob_noise = ob_noise;
SN.etaS = etaS; % minimum probability
SN.etaR = etaR;
SN.betaS = betaS;
SN.betaR = betaR;
SN.alphaS = alphaS;
SN.alphaR = alphaR;
SN.R_s = (betaS*(1-etaS)/etaS)^(1/alphaS); %minimum sensing radius
SN.R_r = (betaR*(1-etaR)/etaR)^(1/alphaR); %minimum transmission radius
```

```
% good values for alpha,beta,eta: (2,1,0.4)
if (nargin < 12) % need to autogenerate nodes
    SN.nodes = [SN.dimX 0 ; 0 SN.dimY] * rand(2,SN.n);
    node_betaS = (betaS + rand(1,SN.n));
    node_betaR = (betaR + rand(1,SN.n));
    SN.nodes = [SN.nodes ;
                (node_betaS.*(1-etaS)/etaS).^(1/alphaS);
                (node_betaR.*(1-etaR)/etaR).^(1/alphaR);
                node_betaS;
                node_betaR];
else
    SN.nodes = nodes;
end
tic
%% Routing
% Find 1-hop probabilities
SN.linkP = zeros(SN.n);
for i = 1:SN.n
    for j = i:SN.n
        d = norm(SN.nodes(1:2,i) - SN.nodes(1:2,j));
        R = min(SN.nodes(4,i),SN.nodes(4,j));
        bI = SN.nodes(6,i);
        bJ = SN.nodes(6,j);
        if (d < R)
            SN.linkP(i,j) = (bI/(bI+d^SN.alphaR)+bJ/(bJ+d^SN.alphaR)) / 2;
            SN.linkP(j,i) = SN.linkP(i,j);
        end
    end
end
% Use a variant of Floyd-Warshall algorithm to find most reliable paths
D = SN.linkP;
P = (D > 0) - eye(size(D));
P = diag(1:SN.n) * P;
for k = 1:SN.n
    for i = 1:SN.n
        for j = 1:SN.n
            [D(i,j), swap] = max([D(i,j) D(i,k)*D(k,j)]);
            if (swap == 2)
                P(i,j) = P(k,j);
            end % update P
        end
    end
end
SN.pathMat = P;
SN.wtMat = D;
SN.routePath = cell(SN.n);
for i = 1:SN.n
    for j = 1:SN.n
        path = [];
        k = j;
        while (k ~= 0 && P(i,k) ~= i)
            path = [P(i,k) path];
            k = P(i,k);
        end
        if (k == 0)
            path = []; % nodes are not connected
        else
            path = [path j];
        end
        SN.routePath{i,j} = path;
    end
end
% Find End-to-End routing probabilities
% This matrix will be important in Cost Function Calculations, but not
% so much for actual transmission simulation
for i = 1:SN.n
```

```
    for j = 1:SN.n
        route = SN.routePath{i,j};
        if isempty(route) && (i == j)
            SN.connProb(i,j) = 1;
        elseif isempty(route)
            SN.connProb(i,j) = 0;
        else
            routeP = [];
            for k = 1:size(route,2)-1
                routeP(end+1) = SN.linkP(route(k),route(k+1));
            end
            maxPvec = routeP;
            for k = 1:SN.rt_num % boosting minimum link
                [min_p ind] = min(maxPvec);
                maxPvec(ind) = 1-(1-routeP(ind))*(1-min_p);
            end
            SN.connProb(i,j) = prod(maxPvec);
        end
    end
end
time = toc;
disp(sprintf('Routing took %.4f time to calculate',time));
if ~isequal((SN.pathMat == 0),eye(size(SN.pathMat)))
    disp('partition exists in network');
end
% Includes routing length from each node to itself
maxL = 0;
sumL = 0;
sumL2 = 0;
for i = 1:SN.n
    for j = i:SN.n
        l = length(SN.routePath{i,j});
        maxL = max(l,maxL);
        sumL = sumL + l;
        sumL2 = sumL2 + l^2;
    end
end
num = SN.n*(SN.n+1)/2;
SN.meanL = sumL/num;
SN.stdL = sqrt(sumL2/num - (SN.meanL)^2); %E[X^2] = sumL2/num
SN.maxL = maxL;
```

A.1.3 Sensor Network Simulation

```
function [delay, packets] = SNSim_ralpha(NoDelayFlag)
% 1/(R^alpha) Sensor Network Simulator
% Runs simulation of evader detection and routing of information through
% the sensor network to the pursuer.
%
% See SNSimInit_ralpha for more details
%
% Input: global variables P, E, SN, amd T
% Output: packets k*n matrix, k is dim of fields in packet, n is number
%                 of packets
%                 [detect_sensor; x; y; send_sensor; timestamp]
%         delay   number of time steps until packet transmission
global P; % pursuer structure
global E; % evader structure
global SN; % sensor network structure
global T;
if (nargin < 1)
    NoDelayFlag = false;
end
detected = [];
% Detection
for i = 1:E.n % this is unnecessary until we augment the dimension of E
    x = E.pos(1,end);
    y = E.pos(2,end);
    for j = 1:SN.n
      dX = x - SN.nodes(1,j);
      dY = y - SN.nodes(2,j);
      R_s = SN.nodes(3,j);
      b = SN.nodes(5,j);
      r = sqrt(dX*dX + dY*dY);
      if (r < R_s) && (rand < b/(b+r^SN.alphaR))
        % detection!
        detected(:,end+1) = [i; j; x+randn*SN.ob_noise; y+randn*SN.ob_noise];
      end
    end
end
% Transmission
delay = [];
packets = [];
% For now, route everything to pursuer 1 (there's only 1 pursuer)
x = P.pos(1,end);
y = P.pos(2,end);
A = [x - SN.nodes(1,:); y - SN.nodes(2,:)];
A(1,:) = A(1,:).*A(1,:);
A(2,:) = A(2,:).*A(2,:);
A = sqrt([1 1] * A);
[r_min closeNode] = min(A);
% step-by-step simulation of packet transmission
for i = 1:size(detected,2)
  packets = [packets [detected(2:4,i) ; closeNode ; T]];
  route = SN.routePath{detected(2,i),closeNode};
  if (length(route) ~= 0) && (route(end) ~= closeNode)   % Sanity Check
      disp('Problem with SN route generation');
      disp(sprintf('route(end) = %d, closeNode = %d, transmit from %d',...
                   route(end), closeNode, detected(2,i)));
  end
  if (length(route) == 0) && (detected(2,i) ~= closeNode)
      delay = [delay (-1)]; % no transmission since no route
  else
```

```
      ttl = SN.rt_num;
      currNode = detected(2,i);
      k = 1;
      while (ttl > 0) && (k <= length(route))
          if (rand < SN.linkP(currNode,route(k)))
              currNode = route(k);
              k = k+1;
          end
          ttl = ttl - 1;
      end
      if (currNode == closeNode) % handles node to itself trans as well
            % same condition as k > length(route)
          % last hop calculation
          R_r = SN.nodes(4,closeNode);
          b = SN.nodes(6,closeNode);
          p_lastHop = b/(b+r_min^SN.alphaR);
          while (ttl > 0) && ((r_min > R_r) || (rand > p_lastHop))
              ttl = ttl - 1; % doesn't decrement on success **
          end
      end %currNode == closeNode...
      if (ttl > 0)
          % successful transmission!
          if (NoDelayFlag)
              delay = [delay 0];
          else
              delay = [delay (SN.rt_num - ttl + 1)]; % +1 for comment ** above
          end
      else
          % no transmission
          delay = [delay (-1)];
      end %ttl...
  end %length(route)...
end
```

A.1.4 Control Law Simulation

```
function PpolicyNonLinOpt(rcvpkts)
% Updates P.control based on data received through the sensor network
% Uses Nonlinear Optimization to compute the controls
global P;
global Pctrlr; % holds the state of the controller estimators, etc.
global history; % make history hold the values of the Pctrlr
global dT;
if ~isempty(rcvpkts)
    [maxT maxIndices] = max(rcvpkts(6,:));
    lastpkts = rcvpkts(:,maxIndices);
    Epos = mean(lastpkts(3:4,:),2); % average over all received latest
                                    % packets; can use other weighting
                                    % later
else
    Epos = [];
end
% Seed the estimated state in the controller
if Pctrlr.uninit
    if ~isempty(Epos)
        Pctrlr.uninit = false;
        Pctrlr.E = [Epos; 0 ; 0]; %Guess 0 velocity first
        Pctrlr.lastUpdate = maxT;
    else
        P.control = [0 ; 0]; % Don't do anything
        return;
    end
else
    % Kalman Filter to estimate Evader State
    %Innovation Step
    A_e = Pctrlr.Emodel.a;
    C_e = Pctrlr.Emodel.c;
    G_e = Pctrlr.Emodel.b;
    Q = Pctrlr.Q;
    R = Pctrlr.R;
    E_vec = Pctrlr.E(:,end);
    P_mat = Pctrlr.Ecov(:,:,end);
    E_vec = A_e*E_vec; % x_k+1|k
    P_mat = A_e*P_mat*A_e' + G_e*Q*G_e'; %P_k+1|k
    % Correction Step
    if (~isempty(Epos) && Pctrlr.lastUpdate < maxT)
        Pctrlr.lastUpdate = maxT;
        K = P_mat*C_e'*(C_e*P_mat*C_e' + R)^-1; % K_k+1
        P_mat = P_mat - K*C_e*P_mat; % P_k+1|k+1
        E_vec = E_vec + K*(Epos - C_e*E_vec); % x_k+1|k+1
    end
    Pctrlr.E(:,end+1) = E_vec;
    Pctrlr.Ecov(:,:,end+1) = P_mat;
end
% Pursuer State, no need for Kalman Filter
Pctrlr.measPos(:,end+1) = P.pos(:,end)+randn(4,1)*P.meas_std;
% Actual Control Output Calculation Step
% Initial Guess: drive toward the predicted evader position at horizon
%                directly
if isempty(Pctrlr.uHoriz)
    t = Pctrlr.ch*dT;
    u0 = Pctrlr.E(:,end) - Pctrlr.measPos(:,end);
    u0 = 2*(u0(1:2) - u0(3:4)*t)/t^2; %a = 2(x-vt)/t^2
    u0 = u0*dT;
    u0 = u0 * ones(1,Pctrlr.ch); u0 = reshape(u0,[numel(u0) 1]);
else % use previous guess
    u0 = [Pctrlr.uHoriz(:,end) ; zeros(2,1)];
    u0 = u0(3:end);
```

```
end
tic
options = optimset('GradObj','off');
switch(Pctrlr.ctrlChoice)
    case 1
        u = fminunc(@BasicCostFun,u0,options);
    case 2
        u = fminunc(@SumGammaCostFun,u0,options);
    case 3
        u = fminunc(@CovGammaCostFun,u0,options);
    case 4
        u = fminunc(@CovGammaOnlyCostFun,u0,options);
    otherwise
        %default
        u = fminunc(@CovGammaCostFun,u0,options);
        %u = fminunc(@BasicCostFun,u0,options);
end
toc
P.control(:,end+1) = u(1:2); %Use only first control value
Pctrlr.uHoriz(:,end+1) = u;
```

A.1.5 Connectivity-Covariance Pursuit Cost Function

```
function J = CovGammaCostFun(u)
% NEED TO ADD GRADIENT CALC
% J = cost(Ppos-Epos) + cost(u) - trace(Cov) with weights
% Input: u  nT*1 vector, n is the dimension of the control, h is the
%           control time horizon.  Stacked u's, [u(:,1); u(:,2); ...]
% Ouput: Value of the cost, J & g, the gradient
%
% Uses b/(b+x^alpha) radio model.
% Works with PpolicyNonLinOpt.
global P;
global Pctrlr;
global SN; % Placeholder for the Pursuer's estimate of the sensor network
global testT; % 0 means we are not testing, otherwise represents time.
if (~isempty(testT) && (testT ~= 0))
    Epos = Pctrlr.E(:,testT);
    Ppos = Pctrlr.measPos(:,testT);
else
    Epos = Pctrlr.E(:,end);
    Ppos = Pctrlr.measPos(:,end);
end
Gamma = [];
A_p = Pctrlr.Pmodel.a;
B = Pctrlr.Pmodel.b;
A_e = Pctrlr.Emodel.a;
C_e = Pctrlr.Emodel.c;
G_e = Pctrlr.Emodel.b;
Q = Pctrlr.Q;
R = Pctrlr.R;
P_mat = Pctrlr.Ecov(:,:,end);
J = 0;
for t = 1:Pctrlr.ph
    Epos(:,end+1) = A_e*Epos(:,end);
    if (t <= Pctrlr.ch)
        Ppos(:,end+1) = A_p*Ppos(:,end) + B*u(2*t-1:2*t);
    else
        Ppos(:,end+1) = A_p*Ppos(:,end);
    end
    % Gamma(t) = Pctrlr.gamma(Ppos(1:2,t),Epos(1:2,t)); % If precomputed
    x_p = Ppos(1,end);
    y_p = Ppos(2,end);
    x_e = Epos(1,end);
    y_e = Epos(2,end);
    % finding the closest node to the pursuer
    F =  - [x_p - SN.nodes(1,:); y_p - SN.nodes(2,:)];
    F(1,:) = F(1,:).*F(1,:);
    F(2,:) = F(2,:).*F(2,:);
    F = sqrt([1 1] * F);
    [r_minP closeNodeP] = min(F);
    % finding the closest node to the evader
    G = [x_e - SN.nodes(1,:); y_e - SN.nodes(2,:)];
    G(1,:) = G(1,:).*G(1,:);
    G(2,:) = G(2,:).*G(2,:);
    G = sqrt([1 1] * G);
    [r_minE closeNodeE] = min(G);
    % Calculate the probabilities
    p = SN.connProb(closeNodeP,closeNodeE);
    Rp = SN.nodes(4,closeNodeP); %comm radius
    if(r_minP < Rp)
        b = SN.nodes(6,closeNodeP);
        p_lastHop = b/(b+r_minP^SN.alphaR);
        Gamma(t) = p*(1-(1-p_lastHop^3)); %allow 3 retransmissions
    else
        Gamma(t) = 0;
    end
    % Calculate Covariance wrt Gamma
    P_mat = A_e*P_mat*A_e' + G_e*Q*G_e'; %P_k+1|k
    K = P_mat*C_e'*(C_e*P_mat*C_e' + R)^-1; % K_k+1
    P_mat = P_mat - Gamma(t)*K*C_e*P_mat; % P_k+1|k+1
    J = J + trace(P_mat);
end
J = Pctrlr.gWt*J;
if (~isempty(testT) && (testT ~= 0)) %DEBUG
    disp(sprintf('J from Gamma = %.2f',J));
    Gamma
end
for t = 2:Pctrlr.ph+1 %recall that Epos has 2-dim length h+1
    x_diff = Epos(:,t) - Ppos(:,t);
    J = J + x_diff'*Pctrlr.xWt*x_diff;
end
J = J + u'*Pctrlr.uWt*u;
if (~isempty(testT) && (testT ~= 0)) %DEBUG
    disp(sprintf('J final = %.2f',J));
end
% IMPLEMENTATION NOTES:
% Evaluates the expected value for Gamma, the probability for receiving a
% packet, given a vector of control inputs u.
% Note that this function depends on the state of the Pursuer and the
% Estimated State of the Evader.
%
% Technically, we should compute Gamma by using NOT the closest node to the
% pursuer but the node with the best connectivity.  We'll fix this later.
```

A.2 Implementation Code for Testbeds

The relevant code for both the indoor and outdoor testbed can be downloaded from `http://www.sourceforge.net` under the **TinyOS** project.

Detailed instructions on setting up and using the indoor robot testbed is given at: `http://www.eecs.berkeley.edu/~phoebusc/330NEST/welcome.html` . The code for the indoor testbed is located under the Tinyos SourceForge code repository at: `tinyos-1.x/contrib/ucbRobo` . See the `README.ucbRobo` file in that directory for more details. This includes configuration/environment setup instructions, MATLAB tools, NesC libraries and sample NesC applications.

The code for the outdoor testbed is located under the TinyOS SourceForge code repository at: `tinyos-1.x/contrib/nestfe`. Much of this code was written by others working on the NEST project (See the Acknowledgements section). The MATLAB tools described in Section 4.2.2.3 are in `tinyos-1.x/contrib/nestfe/matlab` . The event detection application is under `tinyos-1.x/contrib/nestfe/nesc/apps/TestDetectionEvent` .

The documentation for the NEST final experiment and the outdoor testbed is given at: `http://nest.cs.berkeley.edu/nestfe/index.php/Main_Page` .

The usage instructions for the event detection application are at: `http://nest.cs.berkeley.edu/nestfe/index.php/Detection_Demo#Tutorial` .

The instructions for setting up the multi-target tracking application to run on the outdoor testbed at: `http://nest.cs.berkeley.edu/nestfe/index.php/Detection_Demo#Tutorial` . All of the relevant multi-target tracking code written by Songhwai Oh is in the TinyOS SourceForge code repository under: `tinyos-1.x/contrib/nestfe/matlab/apps/TestDetectionEvent/hmttpeg` .

Appendix B

Indoor Testbed Interface Board Schematic

The schematic below needs modifications to add the LM317 Voltage regulator to step down the voltage feeding into the mica2dot charger board. Without the LM317, the power circuitry on the charger board would be operating outside its specified operating range, producing a noisy power line that affects the radio communication on the mote. This was discovered after manufacturing the board, and was fixed by hand using wirewrap, solder, and an exacto knife.

The modification to the board involves inserting the LM317 Voltage Regulator between the signal `EPRB power` and pin 7 on the four position switch, tying the `ADJ` pin on the LM317 (the gate) to ground. `VIN` on the LM317 connects with `EPRB power` and `VOUT` connects to the four position switch.

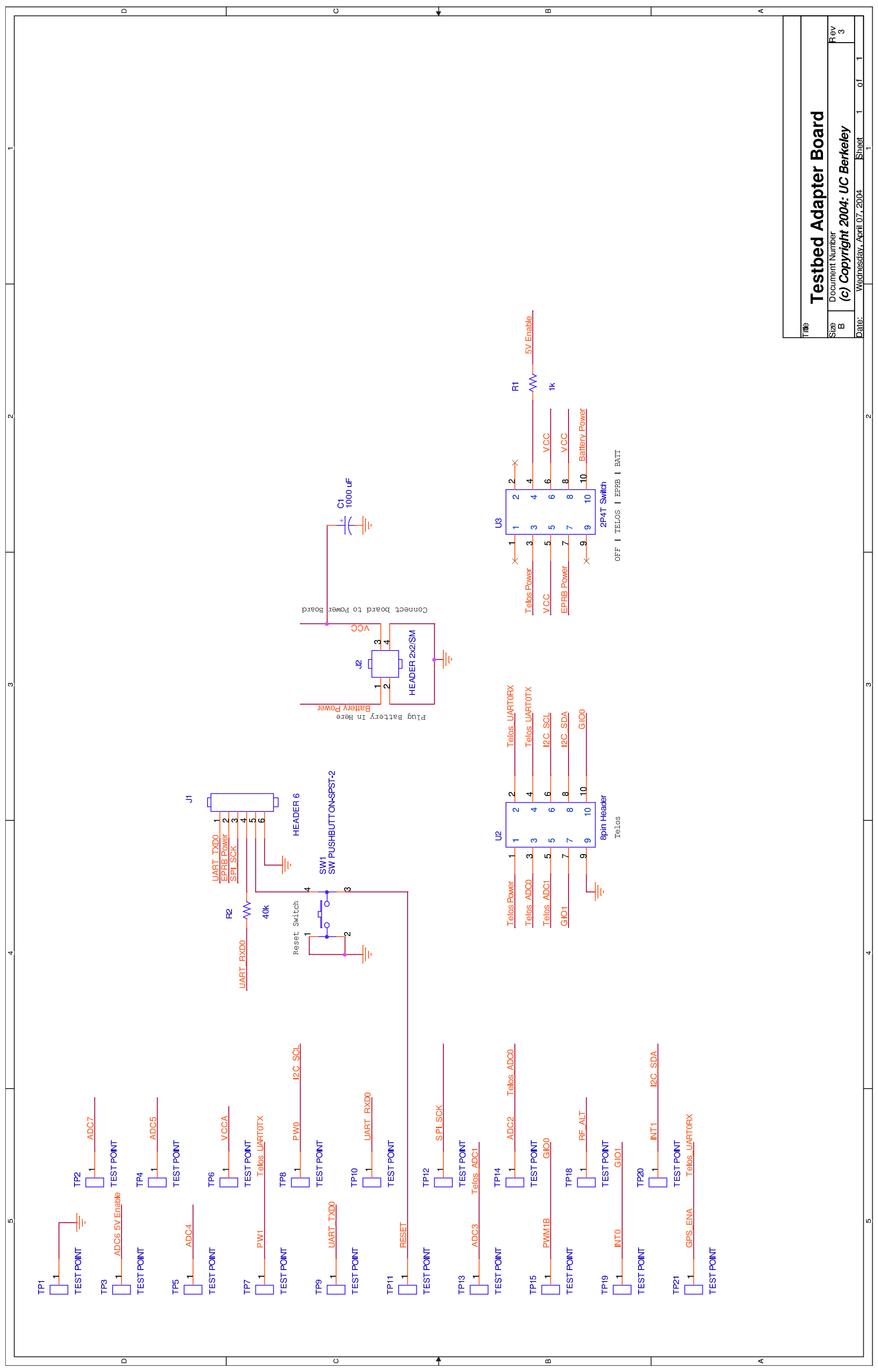


Figure B.1. Indoor testbed interface board

Appendix C

Simulation and Experiment Figures

C.1 Simulation Results

This section contains additional plots of the simulations for Section 2.2.5.2 comparing an LQG controller to a connectivity-convariance controller. Each set of plots is for different starting positions of the pursuer and the evader, with the same experimental setup as described in Section 2.2.5.2. A black line connecting a pursuer and an evader denotes capture during that run. Note that the performance of both controllers are nearly identical. Even changing the weights between the different terms in Eq. 2.14 yields similar performance, though the difference in the trajectories are a little more pronounced.

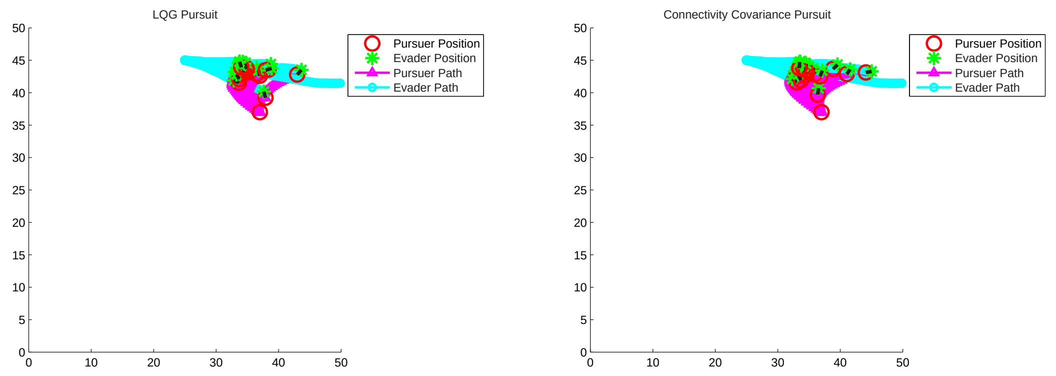


Figure C.1. Simulation results for (left) LQG controller (mean capture time = 3.44, 19/20 captured) and (right) connectivity-covariance controller (mean capture time = 3.43, 19/20 captured). (This figure is best viewed in color.)

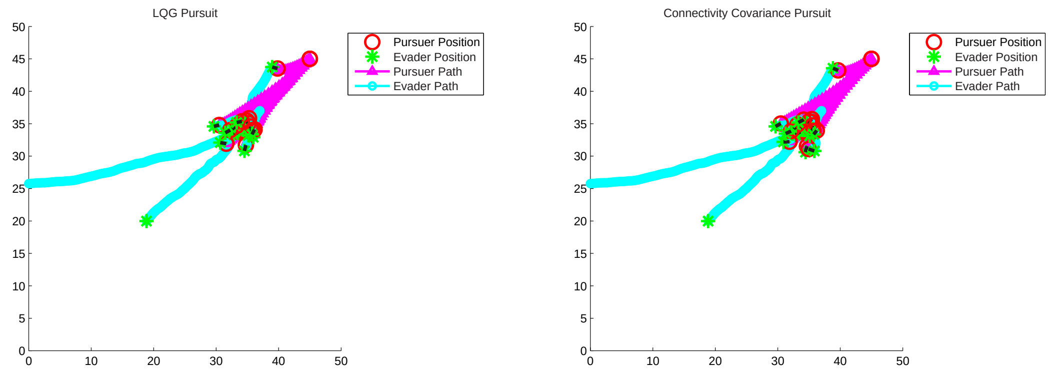


Figure C.2. Simulation results for (left) LQG controller (mean capture time = 3.85, 18/20 captured) and (right) connectivity-covariance controller (mean capture time = 3.92, 18/20 captured). (This figure is best viewed in color.)

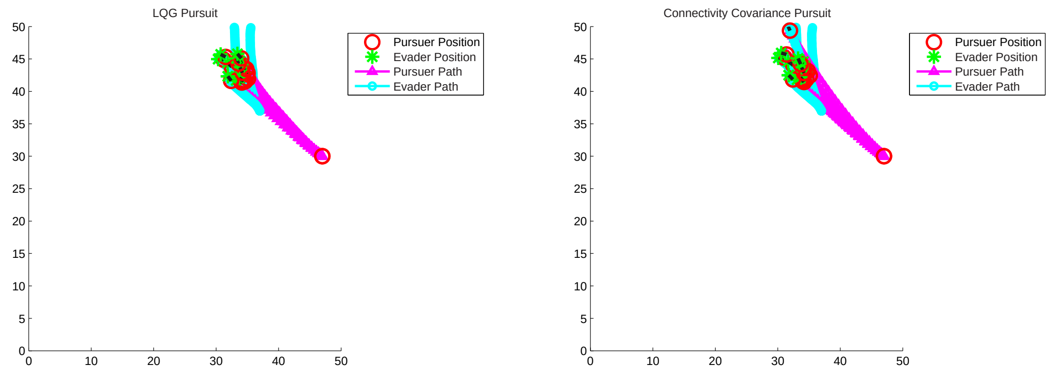


Figure C.3. Simulation results for (left) LQG controller (mean capture time = 3.47, 18/20 captured) and (right) connectivity-covariance controller (mean capture time = 3.64, 18/20 captured). (This figure is best viewed in color.)

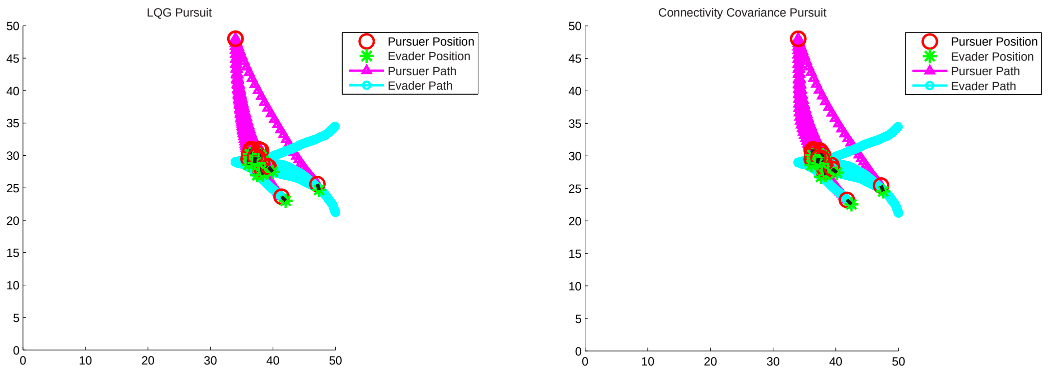


Figure C.4. Simulation results for (left) LQG controller (mean capture time = 3.91, 18/20 captured) and (right) connectivity-covariance controller (mean capture time = 3.95, 18/20 captured). (This figure is best viewed in color.)

C.2 Experiment Results

This section contains additional plots for the multiple target tracking demo on 8/30/2005 described in Section 5.3.

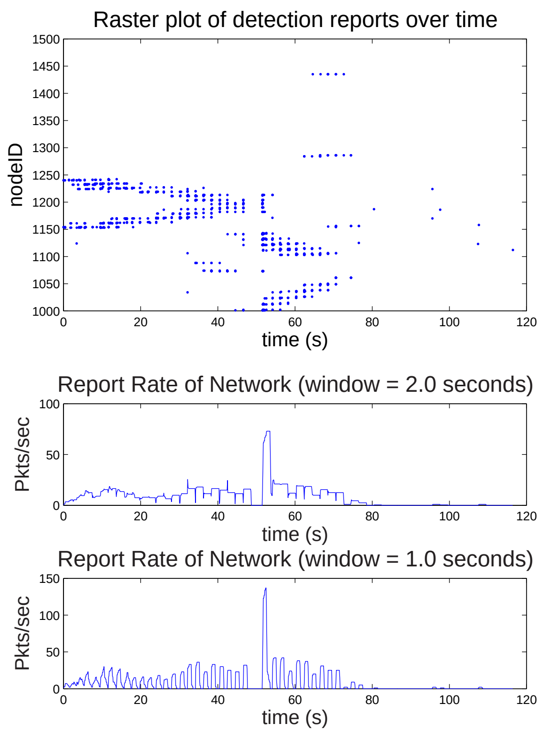


Figure C.5. Raster plots of reports over time by node ID, and the corresponding report rates for the two target tracking demo. Report rates at time t are calculated from the number of reports in the window *before* time t.

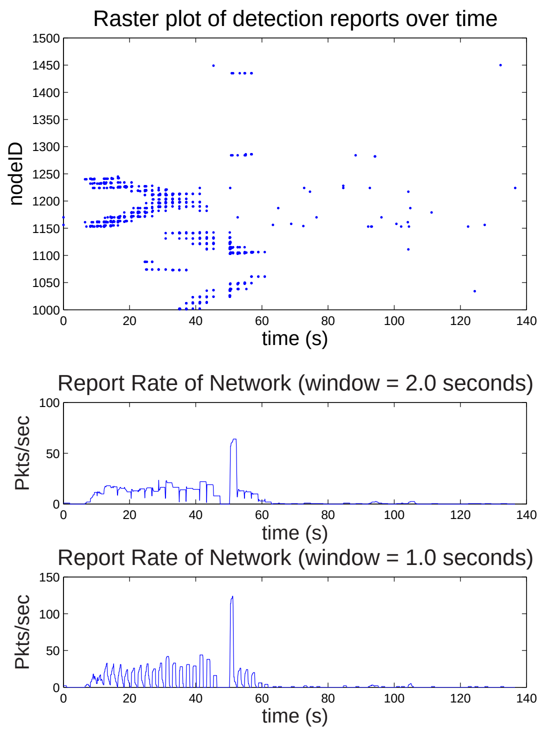


Figure C.6. Raster plots of reports over time by node ID, and the corresponding report rates for the two target pursuit demo. Report rates at time t are calculated from the number of reports in the window *before* time t.

Appendix D

Data Fusion Sensing Model

This section describes the sensing model used for simulations from [58] developed by Songhwai Oh.

As mentioned in Section 2.1.1.1, the sensors typically used in real sensor network deployments can at best give ranging and angle information to a detection event/target. Sensors that give ranging information based on signal strength include acoustic, magnetic, and passive infrared sensors. Thus, signal strength sensing models have been used frequently for sensor networks [59]–[61]. Sometimes signal strength from these sensors cannot provide good ranging estimates, and we are reduced to treating the sensors as binary sensors, as done in Appendix G.2.

But assuming we have better sensors and more sophisticated signal processing algorithms in the future, we could reduce the noise in the sensing signals. A reasonable model for the sensor's signal strength would be

$$z_i = \begin{cases} \frac{\beta}{1+\gamma {r_i}^{\alpha}} + w_i^s, & \text{if } r_i \leq R_s \\ w_i^s, & \text{if } r_i > R_s \end{cases} \tag{D.1}$$

where $r_i = \|s_i - x\|$ and α, β, and γ are constants specific to the sensor type. Assume z_i is normalized such that w_i^s has the standard Gaussian distribution.

Note that this sensing model is similar to the detection probability model in Equation 2.1. Instead of using the detection probability model in Equation 2.1, we can simply

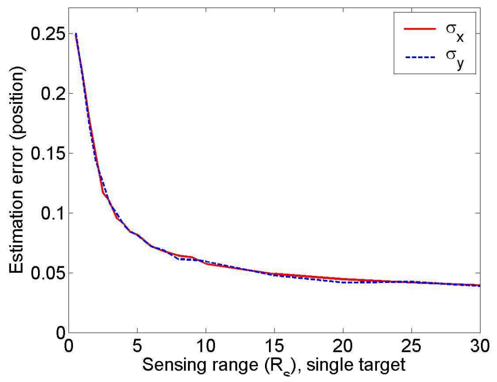


Figure D.1. Single target position estimation error. (Monte Carlo simulation of 1000 samples; Unity corresponds to the separation between sensors; Sensor model: $\alpha = 3$, $\gamma = 1$, $\eta = 2$, and $\beta = 3(1 + \gamma R_s^{\alpha})$.)

look at when the sensor's signal strength crosses as threshold η, where w_i^s introduces randomness so we can speak of a detection probability.

For each i, if $z_i \geq \eta$, where η is a threshold set for appropriate detection probabilities and false-positive probabilities, the node transmits z_i to its neighboring nodes, which are at most $2R_s$ away from s_i, and listens to incoming messages from neighboring nodes less than $2R_s$ away. Note that this approach is similar to the leader election scheme in [60] and we assume that the transmission range of each node is larger than $2R_s$.

For a given node i, if z_i is larger than the signal strength reported by all incoming messages, $z_{i_1}, \ldots, z_{i_{k-1}}$, and $z_{i_k} = z_i$, then the position of the evader is estimated by node i as

$$y = \frac{\sum_{j=1}^{k} z_{i_j} s_{i_j}}{\sum_{j=1}^{k} z_{i_j}} \quad . \tag{D.2}$$

The estimate y corresponds to computing a "center of mass" on the positions of the sensing nodes weighted by their respective signal strengths. Then, node i transmits this observation y up the routing tree in the sensor network. If z_i is not larger than all the sensor signal strengths reported by the incoming messages, node i just continues sensing. Although an individual sensor cannot give an accurate estimate of a target's position, as more sensors collaborate, the accuracy of estimates improves as shown in Figure D.1 taken from [58].

Using thie weighted sensing model, we can approximate the linear sensing model in Equation 2.2. We can reduce the noise covariance Q^s in Equation 2.2 by increasing the number of collaborating sensor nodes for each detection. This can be done by using Figure D.1 to choose the appropriate R_s.

Appendix E

Attempts to Resolve Nonconvexity

This section documents the various attempts to resolve the nonconvexity issues mentioned in Section 2.2.4.

Note that our control problem can be cast in the Nonlinear Model Predictive Control (NMPC) framework, similar to [62]. In [62], the problem was to navigate a helicopter through an “urban canyon” of buildings and obstacles to a chase a moving target. There, the cost terms contributing to the nonconvexity of the cost function repelled the helicopter from the obstacles. On the other hand, in our problem setup of navigating a robot through a sensor field, the cost terms contributing to the nonconvexity of the cost function attract the robot to local minima around certain sensors.

Good performance in the helicopter navigation problem is measured by the helicopter’s ability to avoid crashing into obstacles. The helicopter does not get stuck in local minima because as the target moved away, the cost for remaining stationary increases. Good performance in the sensor network robot navigation problem is measured by whether the robot could capture a target in less time than if it used a LQG controller. So the ideal path would actually want to remain close to local minima in some scenarios, but drift away in other scenarios. If the robot does not balance this tradeoff properly, it will operate in the two extremes of

1. staying near a sensor node (local minima) and not chasing after the target, or

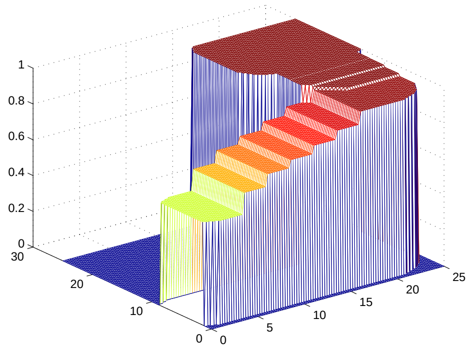


Figure E.1. Sample probability of connectivity map for scenario on the left of Figure 2.3 using the sensor model in Equation 2.4 without accounting for the last hop transmission probability. (This figure is best viewed in color.)

2. chasing after the target with no regard to network connectivity, and behaving like the LQG controller.

Coupled with the difficulty of designing a good scenario to test these controllers, as mentioned in Section 2.2.5.3, we see why even such a simple demonstration of incorporating sensor network quality information into a control law is a nontrivial task.

E.1 Separation of γ to Convex Components

One attempt to sidestep the issue of nonconvexity involves separating the cost terms into convex and nonconvex components (technically, concave and nonconcave, but we will not make a distinction between concave and convex here). Notice that local minima exist in the cost function because the connection probability γ drops off with distance from the last-hop transmitting node. If we calculate γ by only considering the link probabilities in the routing graph and ignore the last hop transmission to the pursuer, in effect only calculating $\check{p}_{path}$ from Algorithm 2, we can remove the local minima (See Fig. E.1). Let's call this new function $\hat{\gamma}$. Although we have removed the local minima (in the strict sense where all neighbors have larger values) $\hat{\gamma}$ has many points with a gradient of zero. Thus, most gradient descent algorithms on $\hat{\gamma}$ will stop whenever they reach a flat region in the cost function, just as if they are stuck in local minima.

Also, despite not having local minima, $\hat{\gamma}$ is not quasiconvex. This is because the sublevel sets of $\hat{\gamma}$ are a union of disks, and in general are not convex. Thus, optimization methods for quasiconvex functions will not work. (For a discussion on quasiconvex functions and finding their optimum by reducing the problem to a series of feasibility tests, see Boyd and Vandenberghe's book on optimization [63].)

E.2 Lagrange Dual

Another attempt at sidestepping the issue of nonconvexity is to find an approximation to the optimization problem by taking the Lagrange dual of Equation 2.17.

The Lagrange dual function $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ is

$$g(\lambda, \nu) = \inf_{x \in D} L(x, \lambda, \nu)_{x \in D} = \inf \Big(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=}^{p} \nu_i h_i(x) \Big) \tag{E.1}$$

Unfortunately, it is not clear how to go about computing the Lagrange dual function of Equations 2.16 or 2.17, because the Lagrange dual of these functions involve taking an infimum over the control input u. u is related to the cost function through γ, which makes writing the Lagrange dual function in closed form difficult/impossible.

E.3 Heuristics for Convex Approximations of γ

Another possible approach is to approximate γ by trying to smooth out its local minima. One heuristic method is to take the weighted average (weight by distance) of the connectivity probability from the three closest neighbors within transmission range. This, unfortunately, has no guarantees that the resulting function is convex (See Fig. E.2).

Yet another approach to approximate γ with a convex function, though tricky to implement, is to "draw a line between the peaks" of γ. What this means mathematically is that at a point between two or three nodes, the connectivity is the weighted sum of the peak connectivity of those nodes. This approach only makes sense when you consider nodes within radio communication range of each other. An optimization routine moving up the

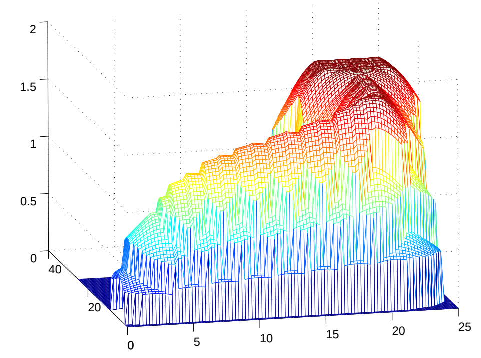


Figure E.2. Connectivity map when we take the weighted average of γ_t from the 3 closest connected neighbors. The "notches" in its curvature show that this is not a convex function.

gradient of such a function would have a tendency to follow the routing tree. This approximation warrants some further investigation, particularly on whether the convergence rate of gradient descent optimization algorithms on this function would be acceptable.

E.4 Alternate Formulation: Ensuring Connectivity Above a Threshold

An alternate formulation of the problem takes advantage of the results by Bruno Sinopoli in [25] which compute the critical connectivity probability, γ_c,[1] for our controlled system to remain stable.

The reformulated problem would be to find a set of controls that minimizes an LQR/LQG objective function while satisfying the constraint of only moving through portions of the network where the probability of receiving detections from the predicted future position of the target is above γ_c. We are assuming here that detections are routed to all the nodes within communication range of $\mathbf{x^p}$, and these nodes broadcast the detection to the controller with a fixed number of retransmissions (for simplicity, we assume one retransmission). Furthermore, we assume the communication links of all the routes fail/succeed independently. For simplicity, we assume that only the nearest node to $\mathbf{x^e}(t)$ detects the target at time t and sends the detection to all nodes within communication range of $\mathbf{x^p}$.

[1]This is a switch of notation from [25] where γ_c was denoted as λ_c.

Let $r_i = \|\mathbf{x^p} - s_i\|$, and denote the set of nodes within communication range of $\mathbf{x^p}$ by $\mathcal{N}_c(\mathbf{x^p}) = \{i : r_i \leq R_{c_i}\}$. Let p_{conn} be the connection probability between the controller and the detecting node. The controller successfully receives a detection when it is delivered by one of the routes with node $i \in \mathcal{N}_c(\mathbf{x^p})$ successfully.

Let $\check{p}^i_{path} p_i$, be the probability of each route with node i. $\check{p}^i_{path}$ is the end-to-end transmission success rate between the sensor nodes on the routing path and is computed via Algorithm 2. p_i is the last-hop connection probability to the pursuer, which is given by one of the node-level communication models (Eqns. 2.3 or 2.4). Given the position of the target and a static network layout, $\check{p}^i_{path}$ is a constant.

The constraint $p_{conn} \geq \gamma_c$ becomes

$$\bigvee_{i \in \mathcal{N}_c(\mathbf{x^p})} \check{p}^i_{path} p_i \geq \gamma_c \tag{E.2}$$

We are dealing with "Or-ed" constraints, meaning that one of many constraints needs to be satisfied. As pointed out by Schouwenaars et al. in [64], in general these types of constraints do not lead to nice convex optimization formulations. Fortunately, these are probabilities so we can rewrite condition E.2 in terms of the probability that transmission along all routes fail:

$$1 - \prod_{i \in \mathcal{N}_c(\mathbf{x^p})} (1 - \check{p}^i_{path} p_i) \geq \gamma_c \tag{E.3}$$

Although this problem formulation is not easily handled by gradient descent algorithms, we can try to convert our nonconvex optimization problem to a geometric programming problem. The motivation is that not all geometric programming problems are convex, but they can be transformed into a convex optimization problem by a change of variables and a transformation of the objective and constraint functions, as shown in [63]. Geometric programming is described briefly below, followed by two (failed) attempts to convert the problem into a geometric program.

In the derivations below, we make a change of variables, replacing the position of the pursuer $\mathbf{x^p}(t)$ with the distance of the pursuer to each neighboring node i, denoted by r_i.

Given the positions of at least three neighboring nodes and their distances to the pursuer $r_i = \|\mathbf{x^p} - s_i\|$, we can reconstruct $\mathbf{x^p}(t)$.

E.4.1 Geometric Programming

This section is an excerpt from [63].

One class of nonconvex optimization problems that can be converted to convex problems are geometric programming problems. Geometric programming problems are of the form:

$$\min \quad f_0(x) \tag{E.4}$$

$$\text{subject to} \quad f_i(x) \leq 1, \quad i = 1, \dots, m \tag{E.5}$$

$$h_i(x) = 1, \quad i = 1, \dots, p \tag{E.6}$$

where $f_0, \dots, f_m$ are posynomials and $h_1, \dots, h_p$ are monomials. The domain of this problem is $\mathcal{D} = \mathbb{R}^n_{++}$; the constraint $x \succ 0$ is implicit.

A *monomial* is a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with $\mathbf{dom} f = \mathbb{R}^n_{++}$, defined as

$$f(x) = c x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n} \tag{E.7}$$

where $c \geq 0$ and $a_i \in \mathbb{R}$. Note that this is not to be confused with the standard definition of a monomial in algebra, in which the exponents must be nonnegative integers.

A *posynomial* is a sum of monomials. That is

$$f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}} \tag{E.8}$$

where $c_j \geq 0$.

E.4.2 Geometric Programming using the Linear Communication Model

We start the derivation by assuming the linear node-level communication model proposed in Equation 2.3 without a cutoff communication radius R_{c_i} for simplicity. This means we are assuming that $r_i < R_{c_i}$, or that all nodes are within one-hop radio communication range of $\mathbf{x^p}$.

Let $p_i = 1 - \frac{r_i}{R_{c_i}}$ instead of the piecewise linear function in Equation 2.3. Then Equation E.3 becomes

$$\begin{aligned}
1 - \prod_{i=1}^{N_s}(1 - \check{p}_{path}^i p_i) &\geq \gamma_c \\
1 - \prod_{i=1}^{N_s}(1 - \check{p}_{path}^i (1 - \frac{r_i}{R_{c_i}})) &\geq \gamma_c \\
1 - \prod_{i=1}^{N_s}(C_i + \check{p}_{path}^i \frac{r_i}{R_{c_i}}) &\geq \gamma_c \qquad \text{where} \qquad C_i = 1 - \check{p}_{path}^i \\
1 - \gamma_c &\geq \prod_{i=1}^{N_s}(C_i + \check{p}_{path}^i \frac{r_i}{R_{c_i}}) \\
1 &\geq \frac{1}{1-\gamma_c}\prod_{i=1}^{N_s}(C_i + \frac{\check{p}_{path}^i}{R_{c_i}} r_i)
\end{aligned}$$

Note that $\frac{1}{1-\gamma_c}\prod_{i=1}^{N_s}(C_i + \frac{\check{p}_{path}^i}{R_{c_i}} r_i)$ is a posynomial in r_i, meaning this is a valid geometric programming constraint.

Unfortunately, the full piecewise linear communication model in Equation 2.3 changes the posynomial to

$$\frac{1}{1-\gamma_c}\prod_{i \in \mathcal{N}_c(\mathbf{x^p})}(C_i + \frac{\check{p}_{path}^i}{R_{c_i}} r_i).$$

Because the terms in the product depend on the location of $\mathbf{x^p}$, this expression/constraint changes depending on the control input! Unlike the SINR model to be discussed in Section E.4.3, we cannot ignore the presence of a cutoff communication radius or we will be dealing with negative probabilities for nodes outside of one-hop communication range of $\mathbf{x^p}$. So in conclusion, the linear decay model does not convert nicely to a geometric program.

E.4.3 Geometric Programming using the SINR Communication Model

We assume the SINR node-level communication model in Equation 2.4. Again, we do not use a cutoff communication radius R_{c_i} to avoid the issues encountered in Section E.4.2.

First of all,

$$1 - \check{p}_{path}^i p_i = 1 - \check{p}_{path}^i \frac{\beta}{\beta + r_i^\alpha} = \frac{(\beta + r_i^\alpha) - \check{p}_{path}^i \beta}{\beta + r_i^\alpha} = \frac{(1 - \check{p}_{path}^i)\beta + r_i^\alpha}{\beta + r_i^\alpha}$$

$$= \frac{C_i + r_i^\alpha}{\beta + r_i^\alpha} \qquad \text{where} \qquad C_i = (1 - \check{p}_{path}^i)\beta.$$

Using this, the constraint becomes

$$\begin{aligned}
1 - \prod_{i=1}^{N_s}(1 - \check{p}_{path}^i p_i) &\geq \gamma_c \\
1 - \prod_{i=1}^{N_s} \frac{C_i + r_i^\alpha}{\beta + r_i^\alpha} &\geq \gamma_c \\
1 - \gamma_c &\geq \prod_{i=1}^{N_s} \frac{C_i + r_i^\alpha}{\beta + r_i^\alpha} \qquad \text{(E.9)} \\
0 &\geq \prod_{i=1}^{N_s}(C_i + r_i^\alpha) - (1 - \gamma_c)\prod_{i=1}^{N_s}(\beta + r_i^\alpha) \\
1 &\geq 1 + \prod_{i=1}^{N_s}\Big((1 - \check{p}_{path}^i)\beta + r_i^\alpha\Big) - (1 - \gamma_c)\prod_{i=1}^{N_s}(\beta + r_i^\alpha) \\
1 &\geq 1 + \sum_{k=1}^{2^{N_s}} c_k r_1^{\alpha I(k,1)} r_2^{\alpha I(k,2)} r_3^{\alpha I(k,3)} \dots r_{N_s}^{\alpha I(k,N_s)} \qquad \text{(E.10)} \\
&\quad \text{where} \qquad I(k,j) = \begin{cases} 1, & \text{if } \lfloor \frac{k}{2^j} \rfloor > 0 \\ 0, & \text{if } \lfloor \frac{k}{2^j} \rfloor = 0 \end{cases} \\
&\quad \text{and} \qquad c_k = \Big((1 - \check{p}_{path}^i)^n - (1 - \gamma_c)\Big)\beta^n \\
&\qquad\quad \text{where} \qquad n = N_s - \sum_{i=1}^{N_s} \lfloor \frac{k}{2^i} \rfloor
\end{aligned}$$

The right side of Equation E.10 is a posynomial if c_k are all positive. But if all c_k are positive, then the terms are all positive and the inequality is only satisfied if all the terms are zero. This problem cannot be converted to a geometric program. This difficultly arises because in Equation E.9, $\prod_{i=1}^{N_s}(C_i + r_i^\alpha)$ is a posynomial and $\prod_{i=1}^{N_s}(\beta + r_i^\alpha)$ is a posynomial but their quotient is not necessarily a posynomial.

Appendix F

Background: Differential Games

F.1 Solving Pursuit-Evasion Games of Kind

This section is a summary/overview from [26] chapters 8 and 4, assuming some knowledge of Games of Degree.

The Lifeline game is known as a *Game of Kind*, where the value of the game V is -1 for capture or 1 for escape. Thus, the pursuer wishes to minimize the value of the game while the evader wishes to maximize the value of the game. Although the value of the game V is not differentiable, we can view a Game of Kind as a game embedded in another *Game of Degree* with differentiable V. This V would have only a terminal cost and no integral cost. $V \geq 0$ is interpreted to mean escape while $V < 0$ means capture.

A solution to a Game of Kind involves finding regions in the state space guaranteeing victory for the pursuer or the evader should they play using optimal policies. These regions are known as the *Escape Zone*, EZ, and the *Capture Zone*, CZ. This involves finding *barriers* $\mathcal{S}$ between these regions, called *semipermeable surfaces* or SPS. They are called semipermeable surfaces because by the unilateral action of the evader (pursuer), the state of the game cannot penetrate the surface in the direction favoring the pursuer (evader).

F.1.1 Solving for the Barrier

Assume the Kinematic Equations for the game are:

$$\dot{x}_j = f_j(\mathbf{x}, \phi, \psi) \tag{F.1}$$

Finding the barrier of the game involves solving equations similar to the Main Equation and Path Equations for a Game of Degree, but with a change of interpretation. Instead of a derivative of the value of the game V, we use a vector $v = (v_1, \ldots, v_n)$ that is normal to the barrier surface. The length of v is unimportant so long as it does not equal 0, but v must point toward the escape zone EZ. The modified Main Equation becomes:

$$\min_{\phi} \max_{\psi} \sum_{i=1}^{n} v_i f_i(\mathbf{x}, \phi, \psi) = 0 \tag{F.2}$$

Using an argument similar to those used to derive the original Path Equations, we get:

$$\dot{v}_j = \sum_i v_i f_{ij} \qquad \text{where} \quad f_{ij} = \frac{\partial f_i}{\partial x_j}$$

and the retrogressive path equations (RPE) become

$$\begin{aligned} \mathring{x}_j &= -f_j(\mathbf{x}, \bar{\phi}, \bar{\psi}) && \text{(F.3)} \\ \mathring{v}_j &= \sum_i v_i f_{ij}(\mathbf{x}, \bar{\phi}, \bar{\psi}) && \text{(F.4)} \end{aligned}$$

where $\mathring{x} = \frac{dx}{d\tau}$, $\tau = -t$, and $\bar{\phi}$ means the optimal control for ϕ (and similarly $\bar{\psi}$ is the optimal control for ψ).

The general procedure for solving Differential Games is to start from the terminal surface of the game, $\mathcal{C}$ (the end conditions, where the value of the game is known), and use the RPE to integrate backwards in time and find the value of the game throughout the state space of the game.

The terminal surface $\mathcal{C}$ can usually be parameterized as an $(n-1)$ dimensional surface:

$$x_i = h_i(s_1, \ldots, s_{n-1}) \tag{F.5}$$

Note that there are parts of the terminal surface $\mathcal{C}$ that will never be reached under optimal play of both players. For instance, let's assume it is advantageous for the evader to avoid termination. If at points infinitesimally close to a subset of $\mathcal{C}$, the evader can force

$$\min_{\phi} \max_{\psi} \sum_{i=1}^{n} v_i f_i(\mathbf{x}, \phi, \psi) > 0 \tag{F.6}$$

where v is the normal to $\mathcal{C}$ pointing into the playing space, then the game can never end. These parts of $\mathcal{C}$ are called the *nonuseable part*, or NUP. Similarly, the points on $\mathcal{C}$ that *can* be reached under optimal play are called the *useable part*, or UP, and the $(n-2)$-manifold separating the two regions is called the *boundary of the useable part*, or BUP.

We are interested in integrating backwards from the BUP to get a barrier separating the CZ (region that reaches the UP under optimal play) from the EZ. Let's parameterize the BUP as

$$\mathcal{D} : x_i = h_i(s_1, \ldots, s_{n-2}) \tag{F.7}$$

to get the n equations describing the initial conditions of x_i. We only need $n-1$ equations describing the initial conditions of v, the normal vector to $\mathcal{C}$, because the magnitude of v does not matter. The orientation of v can be deduced from where the playing region of the game lies with respect to $\mathcal{C}$. The normality of v with respect to $\mathcal{C}$ requires that

$$\sum_i v_i h_{ij} = 0 \quad \text{for} \quad j = 1, \ldots, n-2 \qquad \text{where} \quad h_{ij} = \frac{\partial h_i}{\partial s_j} \tag{F.8}$$

This yields $n-2$ equations, which together with the Main Equation yields the $n-1$ equations neede to describe v.

F.1.2 Finding the Optimal Control

Note that since the value of the game is binary, there is technically no notion of "optimal control" for any points not on the barrier $\mathcal{C}$. A player can delay "playing optimally" until he comes arbitrarily close to the barrier without crossing it.

If we embed games of kind into an equivalent game of degree, then we can talk about calculating the optimal control with respect to the value function in the game of degree.

F.1.2.1 Equations for Games of Degree

Value Function V

$$\int G(\mathbf{x}, \phi, \psi) dt + H(s) \tag{F.9}$$

Main Equation

Version 1 (ME_1)

$$\min_{\phi} \max_{\psi} \sum_{j=1}^{n} [V_j f_j(\mathbf{x}, \phi, \psi) + G(\mathbf{x}, \phi, \psi)] = 0 \tag{F.10}$$

$$\text{where } V_j = \frac{\partial V}{\partial x_j}$$

Version 2 (ME_2)

$$\sum_j V_j f_j(\mathbf{x}, \bar{\phi}, \bar{\psi}) + G(\mathbf{x}, \bar{\phi}, \bar{\psi}) = 0 \tag{F.11}$$

$$\text{where } \bar{\phi} = \bar{\phi}(\mathbf{x}, V_x) \text{ and } \bar{\psi} = \bar{\psi}(\mathbf{x}, V_x)$$

Path Equations (includes Kinematic Equations)

$$\dot{V}_k = -\left\{ \sum_i V_i f_{ik}(\mathbf{x}, \bar{\phi}, \bar{\psi}) + G_k(\mathbf{x}, \bar{\phi}, \bar{\psi}) \right\} \tag{F.12}$$

$$\dot{x}_k = f_k(\mathbf{x}, \bar{\phi}, \bar{\psi}) \tag{F.13}$$

$$\text{where } f_{ik} = \frac{\partial f_i}{\partial x_k} \text{ and } G_k = \frac{\partial G}{\partial x_k}$$

Retrogressive Path Equations

$$\mathring{V}_k = \sum_i V_i f_{ik}(\mathbf{x}, \bar{\phi}, \bar{\psi}) + G_k(\mathbf{x}, \bar{\phi}, \bar{\psi}) \tag{F.14}$$

$$\mathring{x}_k = -f_k(\mathbf{x}, \bar{\phi}, \bar{\psi}) \tag{F.15}$$

F.1.2.2 Optimal Control for Games of Degree

Instead of integrating the RPE from the BUP (Eq. F.7), we integrate the RPE from the entire terminal surface (Eq. F.5). Also, we substitute in

$$\frac{\partial H}{\partial s_k} = \sum_i V_i \frac{\partial h_i}{\partial s_k}, \quad k = 1, \ldots, n-1 \tag{F.16}$$

for Equation F.8.

Integrating the RPE gives us $2n$ functions for x_i and V_i in terms of the n arguments $\tau, s_1, \ldots s_{n-1}$. We can then invert the first n of the functions and solve for τ and $s_1, \ldots s_{n-1}$ in terms of x_i. This can then be substituted into the second n functions for V_i, giving us $V_i(x_1, \ldots, x_n)$. This can then be integrated to get V within an additive constant, which is fixed by the known value of V on $\mathcal{C}$.

Finally, we can get the optimal strategies by substituting x_i and V_i into $\bar{\phi} = \bar{\phi}(\mathbf{x}, V_x)$ and $\bar{\psi} = \bar{\psi}(\mathbf{x}, V_x)$ that accompanied ME_2.

F.2 Step-by-step Solution of the Classic Lifeline Game

Continued from Section 3.1. Again, all this material is from Isaacs [26].

Taking the derivative of $\mathcal{K}$ given in Equation 3.4 with respect to the parameter s yields

$$\begin{aligned} \frac{dy_1}{ds} &= -l \sin s \\ \frac{dy_2}{ds} &= 0 \\ \frac{dx}{ds} &= l \cos s \end{aligned} \tag{F.17}$$

We know that the normal vector v should be perpendicular to the barrier. Therefore, $v \cdot \frac{d\mathbf{x}}{ds} = 0$ (again, $\mathbf{x} = (y_1, y_2, x)$). This yields

$$-v_1 l \sin s + v_3 l \cos s = 0$$

$$-v_1 \sin s + v_3 \cos s = 0$$

$$\text{which gives} \quad v_1 = \cos s, \quad v_3 = \sin s \tag{F.18}$$

Given the ME in Equation 3.2, we know that the optimal controls are given by

$$\cos\bar{\phi} = -\frac{v_1}{\rho_1}, \qquad \sin\bar{\phi} = \frac{v_3}{\rho_1} \tag{F.19}$$

$$\cos\bar{\psi} = \frac{v_2}{\rho_2}, \qquad \sin\bar{\psi} = \frac{v_3}{\rho_2} \tag{F.20}$$

$$\text{where } \rho_1 = \sqrt{v_1^2 + v_3^2}, \quad \rho_2 = \sqrt{v_2^2 + v_3^2}$$

Therefore, an alternate form of the ME, ME_2, is

$$-\rho_1 + w\rho_2 = 0 \tag{F.21}$$

Combining the ME_2 with the values of v_1 and v_2 in Equation F.18 yields:

$$\begin{aligned}
\rho_1 &= \sqrt{v_1^2 + v_3^2} = 1 \\
\rho_2 &= \frac{1}{w} = \sqrt{v_2^2 + \sin^2 s} \\
v_2 &= \pm\sqrt{(1/w^2) - \sin^2 s}
\end{aligned}$$

where $\pm$ is $+$ to make v_2 point in the right direction (towards escape, the direction which the evader wishes to maximize).

Integrating the RPE given these values for v using $\mathcal{K}$ as the starting condition yields Equations 3.5. Note that in this problem, the integration is easy because $\mathring{v}_i = 0$ and the state variables never appear on ther right hand side of the equation.

Appendix G

Background: MCMCDA Multiple-Target Tracking

This section is a summary/exerpt from [58] with some additional information on multi-sensor fusion from Songhwai Oh.

In [40], Markov chain Monte Carlo data association (MCMCDA) is presented. The MCMCDA tracking algorithm can track an unknown number of targets in real-time and is an approximation to the optimal Bayesian filter. It has been shown that MCMCDA is computationally efficient compared to the multiple hypothesis tracker (MHT) [65] and outperforms MHT under extreme conditions, such as tracking when there is a large number of targets in a dense environment with low detection probabilities and high false alarm rates [40].

MCMCDA has many features that make it appropriate for tracking targets using sensor networks. Unlike some tracking algorithms like Joint Probabilistic Data Association (JPDA) [66], MCMCDA can autonomously initiate and terminate tracks. MCMCDA is robust against transmission failures because transmission failures are just another form of missing observations, which is handled by the algorithm. Also, MCMCDA performs data association using both current and past observations, so delayed observations, *i.e.*, out-of-sequence measurements, can be easily combined with previously received observations to improve the accuracy of the estimates. Furthermore, MCMCDA requires less memory than other tracking algorithms since it maintains only the current hypothesis and the hypothesis with the highest posterior. It does not require the enumeration of all or some of the hypotheses as in [65], [67]. Last, the algorithm is scalable because it can be extended in a hierarchical manner.

To our knowledge, the algorithm presented below is the first general multiple-target tracking algorithm for sensor networks which can systematically track an unknown number of targets in the presence of false alarms and missing observations while being robust against transmission failures, communication delays and sensor localization error.

G.1 Multiple-Target Tracking

G.1.1 Problem Formulation

Let $T \in \mathbb{Z}^+$ be the duration of surveillance. Let K be the number of objects that appear in the surveillance region $\mathcal{R}$ during the surveillance period. Each object k moves in $\mathcal{R}$ for some duration $[t_\mathrm{i}^k, t_\mathrm{f}^k] \subset [1, T]$. Notice that the exact values of K and $\{t_\mathrm{i}^k, t_\mathrm{f}^k\}$ are unknown. Each object arises at a random position in $\mathcal{R}$ at t_i^k, moves independently around $\mathcal{R}$ until t_f^k and disappears. At each time, an existing target persists with probability $1 - p_\mathrm{z}$ and disppears with probability p_z. The number of objects arising at each time over $\mathcal{R}$ has a Poisson distribution with a parameter $\lambda_\mathrm{b} V$ where λ_b is the birth rate of new objects per unit time per unit volume, and V is the volume of $\mathcal{R}$. The initial position of a new object is uniformly distributed over $\mathcal{R}$.

Let $F^k : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ be the discrete-time dynamics of the object k, where n_x is the dimension of the state variable, and let $x^k(t) \in \mathbb{R}^{n_x}$ be the state of the object k at time t. The object k moves according to

$$x^k(t+1) = F^k(x^k(t)) + w^k(t), \qquad \text{for } t = t_\mathrm{i}^k, t_\mathrm{i}^k + 1 \ldots, t_\mathrm{f}^k - 1, \tag{G.1}$$

where $w^k(t) \in \mathbb{R}^{n_x}$ are white noise processes. The white noise process is included to model non-rectilinear motions of targets. The noisy observation (or measurement[1]) of the state of the object is measured with a detection probability p_{d}. Notice that, with probability $1-p_{\text{d}}$, the object is not detected and we call this a *missing observation.* There are also false alarms and the number of false alarms has a Poisson distribution with a parameter $\lambda_{\text{f}} V$, where λ_{f} is the false alarm rate per unit time per unit volume. Let $n(t)$ be the number of observations at time t, including both noisy observations and false alarms. Let $y^j(t) \in \mathbb{R}^{n_y}$ be the j-th observation at time t for $j = 1, \dots, n(t)$, where n_y is the dimension of each observation vector. Each object generates a unique observation at each sampling time if it is detected. Let $H^j : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ be the observation model. Then the observations are generated as follows:

$$y^j(t) = \begin{cases} H^j(x^k(t)) + v^j(t) & \text{if } j\text{-th measurement is from } x^k(t) \\ u(t) & \text{otherwise,} \end{cases} \tag{G.2}$$

where $v^j(t) \in \mathbb{R}^{n_y}$ are white noise processes and $u(t) \sim \text{Unif}(\mathcal{R})$ is a random process for false alarms. We assume that targets are indistinguishable in this paper, but if observations include target type or attribute information, the state variable can be extended to include target type information. The multiple-target tracking problem is to estimate K, $\{t_{\text{i}}^k,\ t_{\text{f}}^k\}$ and $\{x^k(t) : t_{\text{i}}^k \le t \le t_{\text{f}}^k\}$, for $k = 1, \dots, K$, from observations.

G.1.2 Solutions to the Multiple-Target Tracking Problem

Let $y(t) = \{y^j(t) : j = 1, \dots, n(t)\}$ be all measurements at time t and $Y = \{y(t) : 1 \le t \le T\}$ be all measurements from $t = 1$ to $t = T$. Let Ω be a collection of partitions of Y such that, for $\omega \in \Omega$,

1. $\omega = \{\tau_0, \tau_1, \dots, \tau_K\}$;
2. $\bigcup_{k=0}^K \tau_k = Y$ and $\tau_i \cap \tau_j = \emptyset$ for $i \ne j$;
3. τ_0 is a set of false alarms;
4. $|\tau_k \cap y(t)| \le 1$ for $k = 1, \dots, K$ and $t = 1, \dots, T$; and

[1]Note that the terms *observation* and *measurement* are used interchangeably.

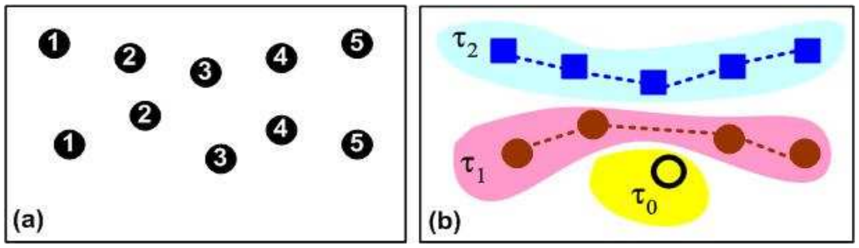


Figure G.1. (a) An example of measurements Y (each circle represents a measurement and numbers represent measurement times); (b) an example of a partition ω of Y (associations are indicated by dotted lines and hollow circles are false alarms).

5. $|\tau_k| \geq 2$ for $k = 1, \ldots, K$.

An example of a partition is shown in Figure G.1 and ω is also known as a *joint association event* in literature. Here, K is the number of tracks for the given partition $\omega \in \Omega$ and $|\tau_k|$ denotes the cardinality of the set τ_k. We call τ_k a track when there is no confusion although the actual track is the set of estimated states from the observations τ_k. We assume there is a deterministic function that returns a set of estimated states given a set of observations, so no distinction is required. The fourth requirement says that a track can have at most one observation at each time, but, in the case of multiple sensors with overlapping sensing regions, we can easily relax this requirement to allow multiple observations per track. A track is assumed to contain at least two observations since we cannot distinguish a track with a single observation from a false alarm, assuming $\lambda_\mathrm{f} > 0$. For special cases, in which $p_\mathrm{d} = 1$ or $\lambda_\mathrm{f} = 0$, the definition of Ω can be adjusted accordingly.

Let $e(t-1)$ be the number of targets from time $t-1$, $z(t)$ be the number of targets terminated at time t and $c(t) = e(t-1) - z(t)$ be the number of targets from time $t-1$ that have not terminated at time t. Let $a(t)$ be the number of new targets at time t, $d(t)$ be the number of actual target detections at time t and $g(t) = c(t) + a(t) - d(t)$ be the number of undetected targets. Finally, let $f(t) = n(t) - d(t)$ be the number of false alarms. It can be shown that the posterior of ω is:

$$P(\omega|Y) \propto P(Y|\omega) \prod_{t=1}^{T} p_\mathrm{z}^{z(t)} (1-p_\mathrm{z})^{c(t)} p_\mathrm{d}^{d(t)} (1-p_\mathrm{d})^{g(t)} \lambda_\mathrm{b}^{a(t)} \lambda_\mathrm{f}^{f(t)} \tag{G.3}$$

where $P(Y|\omega)$ is the likelihood of measurements Y given ω, which can be computed based on the chosen dynamic and measurement models.

There are two major approaches to solving the multiple-target tracking problem [68]: the *maximum a posteriori* (MAP) approach and the Bayesian (or *minimum mean square error* (MMSE)) approaches. The MAP approach finds a partition of observations such that $P(\omega|Y)$ is maximized and estimates states of targets based on the partition which maximizes $P(\omega|Y)$. The MMSE approach seeks the conditional expectations such as $\mathbb{E}(x_t^k|Y)$ to minimize the expected (square) error. However, when the number of targets is not fixed, a unique labeling of each target is required to find $\mathbb{E}(x_t^k|Y)$ under the MMSE approach. In this paper, we take the MAP approach to the multiple-target tracking problem for its convenience.

G.2 Multi-Sensor Fusion Algorithm

In order to obtain finer position reports from binary detections, we use spatial correlation among detections from neighboring sensors. The idea behind the fusion algorithm is to compute the likelihood given detections assuming there is a single target. This is an approximation since there can be more than one target. However, any inconsistencies caused by this approximation are fixed by the tracking algorithm using temporal correlation.

For each sensor i, let R_i be the sensing region of i. R_i can be in an arbitrary shape but we assume that it is known to the system in advance. Let $y_i \in \{0, 1\}$ be the detection made by sensor i, such that sensor i reports $y_i = 1$ if it detects a moving object in R_i, and $y_i = 0$ otherwise. Let p_i be the detection probability and q_i be the false detection probability of sensor i. Let x be the position of an object. For the purpose of illustration, suppose that there are two sensors, sensor 1 and sensor 2, and $R_1 \cap R_2 \neq \emptyset$ (see Fig. G.2 (a)). The overall sensing region $R_1 \cup R_2$ can be partitioned into a set of non-overlapping segments. The likelihoods of points in each segment are the same and they can be computed as follows.

$$\begin{aligned} P(y_1, y_2|x \in S_1) &= p_1^{y_1}(1-p_1)^{1-y_1} q_2^{y_2}(1-q_2)^{1-y_2} \qquad \text{(G.4)} \\ P(y_1, y_2|x \in S_2) &= q_1^{y_1}(1-q_1)^{1-y_1} p_2^{y_2}(1-p_2)^{1-y_2} \\ P(y_1, y_2|x \in S_3) &= p_1^{y_1}(1-p_1)^{1-y_1} p_2^{y_2}(1-p_2)^{1-y_2}, \end{aligned}$$

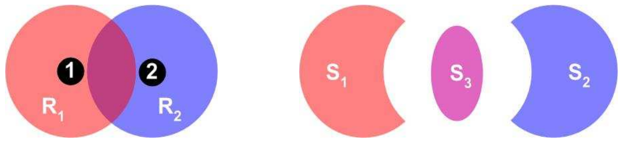


Figure G.2. (left) Sensing regions of two sensors 1 and 2. R_1 is the sensing region of sensor i and R_2 is the sensing region of sensor 2. (right) Non-overlapping partition of sensing regions from (left). $S_1 = R_1 \setminus R_2$, $S_2 = R_2 \setminus R_1$ and $S_3 = R_1 \cap R_2$.

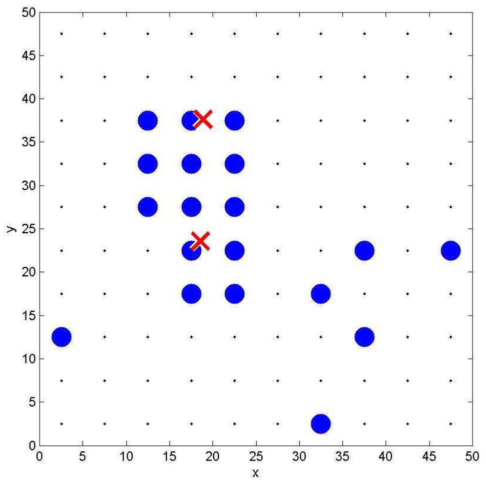


Figure G.3. Detections of two targets by a 10×10 sensor grid (targets in $\times$ (red), detections in (blue) disks, and sensor positions in small dots).

where $S_1 = R_1 \setminus R_2$, $S_2 = R_2 \setminus R_1$ and $S_3 = R_1 \cap R_2$ (see Fig. G.2 (b)). Hence, for any deployment we can first partition the surveillance region into a set of non-overlapping segments. Then, given detection data, we can compute the likelihood of each segment as shown in the previous example.

An example of detections of two targets by a 10×10 sensor grid is shown in Figure G.3. In this example, the sensing region is assumed to be a disk with a radius of 7.62m (10 ft). We have assumed $p_i = 0.7$ and $q_i = 0.05$ for all i. From the detections shown in Figure G.3, its likelihood can be computed efficiently as follows (see Figure G.4). First, for each non-overlapping segment S_j, pick a point in S_j and compute its likelihood similar to (G.4). Then assign the computed likelihood to the whole segment S_j.

There are two parts in this likelihood computation: the detection part (terms involving

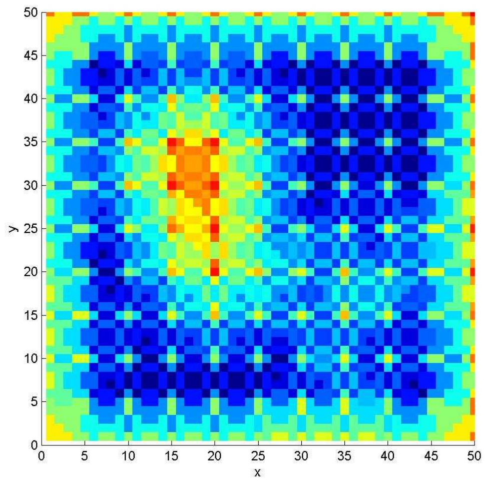


Figure G.4. Likelihood of detections from Figure G.3. (This figure is best viewed in color.)

p_i) and the false detection part (terms involving q_i). Hereafter, we will call the detection part of the likelihood as the *detection-likelihood* and the false detection part of the likelihood as the *false-detection-likelihood*. Notice that the computation of the false-detection-likelihood requires measurements from all sensors. However, it is may not feasible for wireless sensor network to exchange detection data with all other sensors. In order to avoid computation of the false-detection-likelihood and distribute the likelihood computation, we use a threshold test instead. The detection-likelihood of a segment is computed if there are at least n_{d} detections, where n_{d} is a user-defined threshold. Using $n_{\mathrm{d}} = 3$, the detection-likelihood of detections from Figure G.3 can be computed as shown in Figure G.5. The computation of the detection-likelihood can be done in a distributed manner. Assign a set of non-overlapping segments to each sensor such that no two sensors share the same segment and each segment is assigned to a sensor whose sensing region includes the segment. For each sensor i, let $\{S_{i_1}, \ldots, S_{i_{n(i)}}\}$ be a set of non-overlapping segments, where $n(i)$ is the number of segments assigned to sensor i. Then, if sensor i reports a detection, it computes the likelihood of each segment in $\{S_{i_1}, \ldots, S_{i_{n(i)}}\}$ based on its report and reports from neighboring sensors. A neighboring sensor is a sensor whose sensing region intersects the sensing region of sensor i. Notice that no report from a sensor means no detection.

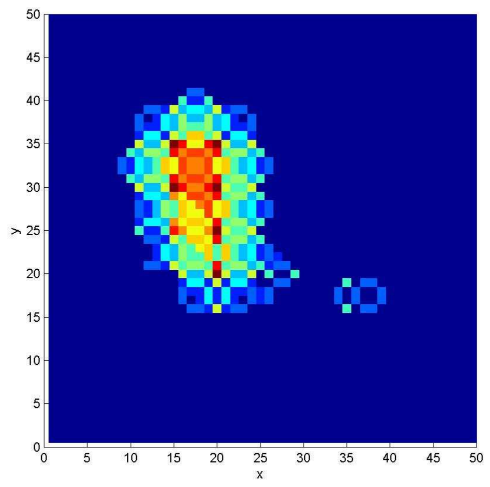


Figure G.5. Detection-likelihood of detections from Figure G.3 with threshold $n_{\mathrm{d}} = 3$. (This figure is best viewed in color.)

Finally, based on the detection-likelihood of detections, we compute position reports by clustering. Let $\mathcal{S} = \{S_1, \ldots, S_m\}$ be a set of segments whose detection-likelihoods are computed. First, randomly pick S_j from $\mathcal{S}$ and remove S_j from $\mathcal{S}$. Then cluster around S_j with the remaining segments in $\mathcal{S}$ whose set-distance to S_j is less than the sensing radius. The segments clustered with S_j are then removed from $\mathcal{S}$. Now repeat the procedure until $\mathcal{S}$ is empty. Let $\{C_k : 1 \leq k \leq K\}$ be the clusters formed by this procedure, where K is the total number of clusters. For each cluster C_k, its center of mass is computed to obtain a fused position report, *i.e.*, estimated positions of targets. An example of position reports are shown in Figure G.6.

G.3 Markov Chain Monte Carlo Data Association (MCMCDA)

This section presents an algorithm for solving the multiple-target tracking problem described in Section G.1. We develop a Markov chain Monte Carlo (MCMC) sampler to solve the multiple-target tracking problem.

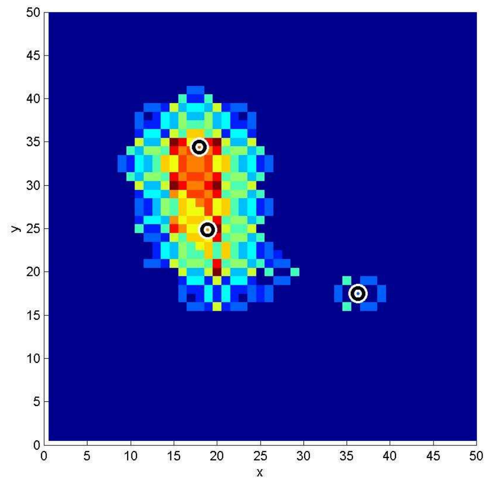


Figure G.6. Estimated positions of targets based on the detection-likelihood shown in Figure G.3 are marked by black circles. (This figure is best viewed in color.)

MCMC-based algorithms play a significant role in many fields such as physics, statistics, economics, and engineering [69]. In some cases, MCMC is the only known general algorithm that finds a good approximate solution to a complex problem in polynomial time [70]. MCMC techniques have been applied to complex probability distribution integration problems, counting problems such as #P-complete problems, and combinatorial optimization problems [69], [70].

MCMC is a general method to generate samples from a distribution π on a space Ω by constructing a Markov chain $\mathcal{M}$ with states $\omega \in \Omega$ and stationary distribution $\pi(\omega)$. We now describe a MCMC algorithm known as the Metropolis-Hastings algorithm. If we are at state $\omega \in \Omega$, we propose $\omega' \in \Omega$ following the proposal distribution $q(\omega, \omega')$. The move is accepted with an acceptance probability $A(\omega, \omega')$ where

$$A(\omega, \omega') = \min\left(1, \frac{\pi(\omega')q(\omega', \omega)}{\pi(\omega)q(\omega, \omega')}\right). \tag{G.5}$$

Otherwise the sampler stays at ω so that the detailed balance is satisfied. If we ensure that $\mathcal{M}$ is irreducible and aperiodic, $\mathcal{M}$ converges to its stationary distribution by the ergodic theorem [71].

The MCMC data association (MCMCDA) algorithm is described in Algorithm 4. MCM-

Algorithm 4 MCMCDA

```
**Input**: $Y, n_{\text{mc}}, \omega_{\text{init}}, X : \Omega \to \mathbb{R}^m$
**Output**: $\hat{\omega}, \hat{X}$
$\omega = \omega_{\text{init}}$; $\hat{\omega} = \omega_{\text{init}}$; $\hat{X} = 0$
**for** $n = 1$ to $n_{\text{mc}}$ **do**
    propose $\omega'$ based on $\omega$ (see Figure G.7)
    sample $U$ from Unif$[0, 1]$
    $\omega = \omega'$ if $U < A(\omega, \omega')$
    $\hat{\omega} = \omega$ if $p(\omega|Y)/p(\hat{\omega}|Y) > 1$
    $\hat{X} = \frac{n}{n+1}\hat{X} + \frac{1}{n+1}X(\omega)$
**end for**
```

CDA is a MCMC algorithm whose state space is the Ω described in Section G.1.2 and whose stationary distribution is the posterior (G.3). The proposal distribution for MCMCDA consists of five types of moves (a total of eight moves). They are

1. birth/death move pair;
2. split/merge move pair;
3. extension/reduction move pair;
4. track update move; and
5. track switch move.

The MCMCDA moves are graphically illustrated in Figure G.7. Each move proposes a new joint association event ω' which is a modification of the current joint association event ω. For a detailed technical description of each move, see [40], [68]. The inputs for MCMCDA are the set of observations Y, the number of samples n_{mc}, the initial state ω_{init}, and a bounded function $X : \Omega \to \mathbb{R}^n$. At each step of the algorithm, ω is the current state of the Markov chain. The acceptance probability $A(\omega, \omega')$ is defined in (G.5) where $\pi(\omega) = P(\omega|Y)$ from (G.3). The output $\hat{X}$ approximates the MMSE estimate $\mathbb{E}_\pi X$ and $\hat{\omega}$ approximates the

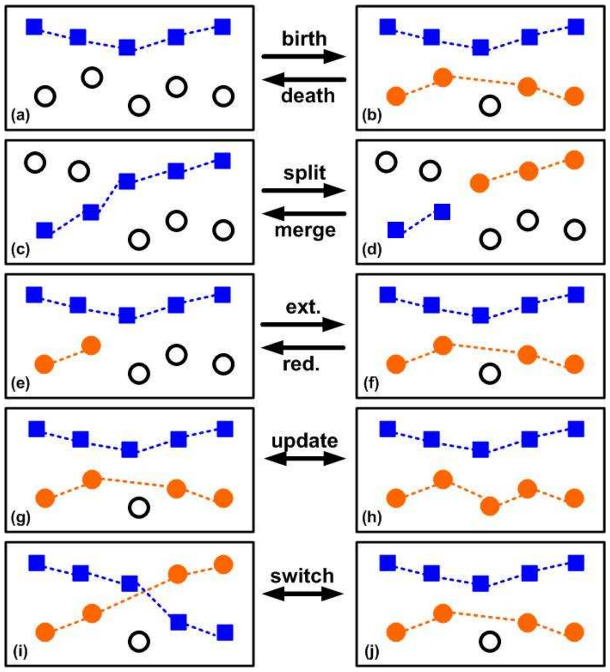


Figure G.7. Graphical illustration of MCMCDA moves (associations are indicated by dotted lines and hollow circles are false alarms). Each move proposes a new joint association event ω' which is a modification of the current joint association event ω. The birth move proposes ω' by forming a new track from the set of false alarms $((a) \rightarrow (b))$. The death move proposes ω' by combining one of the existing tracks into the set of false alarms $((b) \rightarrow (a))$. The split move splits a track from ω into two tracks $((c) \rightarrow (d))$ while the merge move combines two tracks in ω into a single track $((d) \rightarrow (c))$. The extension move extends an existing track in ω $((e) \rightarrow (f))$ and the reduction move reduces an existing track in ω $((f) \rightarrow (e))$. The track update move chooses a track in ω and assigns different measurements $((g) \leftrightarrow (h))$. The track switch move chooses two tracks from ω and switches measurement-to-track associations $((i) \leftrightarrow (j))$.

MAP estimate $\arg\max P(\omega|Y)$. Notice that MCMCDA can provide both MAP and MMSE solutions to the multiple-target tracking problem.

We have shown that MCMCDA is an *optimal Bayesian filter* in the limit, *i.e.*, given a bounded function $X : \Omega \rightarrow \mathbb{R}^n$, $\hat{X} \rightarrow \mathbb{E}_\pi X$ as $n_{\mathrm{mc}} \rightarrow \infty$ [68]. Also it has been shown that MCMCDA is computationally efficient compared to MHT and under extreme conditions outperforms MHT with heuristics such as pruning, gating, clustering, N-scan-back logic and k-best hypotheses [68].