ABSTRACT

CHANG, CHIH-CHIEH GEOFF. Secure Localization and Tracking in Sensor Networks. (Under the direction of Professor Wesley E. Snyder.)

Localization and tracking of objects of interest are two canonical issues in sensor networks research. When the object of interest is static, we use localization algorithms to identify its location. When the object of interest is moving, we use tracking algorithms to estimate its path over time. Since sensor networks are often deployed in remote or hostile terrains, however, security becomes another critical issue. Hence the localization or tracking accuracy would go down as a result of the presence of malicious nodes.

The objective of this dissertation is to correctly identify the malicious nodes during the localization and tracking processes. A novel algorithm based on relaxation labeling is presented to achieve this objective. Our approach provides a different perspective from the existing literature on secure localization and tracking. Current literature uses statistical measures to perform localization and tracking as accurately as possible given the influence of malicious nodes. Instead, those malicious nodes are isolated first, and use only data from benign nodes to perform localization and tracking. Both simulations and field experiments are used to demonstrate the performance of our algorithm.

Novel contributions of this dissertation are as follows: 1. Extension of the classic relaxation labeling algorithm to contain higher-order consistency functions defined on triples of objects, rather than pairs. 2. Solution of the secure localization problem by detecting malicious nodes. 3. Definition of the secure tracking problem in the presence of malicious nodes, and solution of that problem combining conventional tracking with relaxation labeling.

# Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **2008**	2. REPORT TYPE	3. DATES COVERED **00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE **Secure Localization and Tracking in Sensor Networks**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **North Carolina State University,Department of Electrical and Computer Engineering,Raleigh,NC,27695**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S)
	11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**157**	

Secure Localization and Tracking in Sensor Networks

by

Chih-Chieh Geoff Chang

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina

2008

APPROVED BY:

Dr. Griff L. Bilbro	Dr. John E. Franke
Dr. Wesley E. Snyder Chair of Advisory Committee	Dr. Cliff Wang

Dedication

To my parents, Yurng-Der Chang and Shiu-Mei Lee

Biography

Chih-Chieh Geoff Chang was born and raised in a beautiful resort town, Hualien, Taiwan, Republic of China. In 1993, he passed the high-school entrance exam with the highest score in the county, and was admitted to Hualien Senior High School. Before graduation in 1996, he enjoyed a happy childhood in the less-developed Eastern Taiwan.

Beginning in 1996, Chang left Hualien and was enrolled in National Taiwan University (NTU) in Taipei, Taiwan, the Capitol and the biggest city. He studied Electrical Engineering at NTU, the most prestigious university in Taiwan, and served as President of the Chinese Institute of Engineers Student Chapter at NTU from 1998 to 1999.

Upon graduation from NTU in 2000, Chang worked as a summer intern in Nortel Networks in Taipei, Taiwan for three months. Beginning in October 2000, he served (compulsory) military service as a communication lieutenant. After the initial five months of training, he was dispatched to the Spratly Islands on the South China Sea for about a year. He completed the military service in June 2002.

Chang started his graduate study at North Carolina State University in Raleigh, North Carolina in August 2002. He was awarded his master degree in December 2003, also in Electrical Engineering. His research interests include wireless sensor networks, radio frequency identification (RFID), and all aspects of digital image processing and digital signal processing. He has published more than four papers on internationally peer-reviewed journals and conferences. He is also a certified RFID professional by the Computing Technology Industry Association.

Acknowledgments

There is a Chinese proverb which can be loosely translated into English as: “One day as a teacher; one lifetime as a father.” This saying demonstrates the traditional Chinese way of respecting teachers, and as a result, one should respect his or her teacher like his or her father. Dr. Snyder is exactly this kind of fatherly figure to his students. He is genuinely smart, gregarious, and has an insatiable appetite for knowledge. What is more, he treats his students as families, not subordinates. He is probably the best advisor you can find, and I am apparently a major beneficiary of that. Without his guidance, I could not have finished this dissertation.

I would also like to thank Dr. Cliff Wang at the United States Army Research Office for providing the funding to my research and suggestions for possible research directions.

A special thanks goes to Dr. Peng Ning at the Computer Science Department and his students. They provide the experimental data for use in this dissertation. Another special thanks goes to Dr. Ping-Tung Hsiao at the Marine, Earth and Atmospheric Sciences Department for hiring me as a research assistant for one summer.

Last but not least, I would like to thank my parents for their unconditional love and care. This dissertation is dedicated to them. Without their love I could not have accomplished it. I would also like to thank my sister and brother-in-law who can accept the fact that I miss their wedding in Taiwan due to the preparation of this dissertation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 What Are Sensor Networks?

Recent advances in microelectronic technology have enabled a new generation of sensor networks which holds the potential to revolutionize our economy and society [23, 16, 58, 50]. Large numbers of tiny and smart sensor nodes will be sprayed onto roads, bridges, factories and forests. These "sensor nodes" collaboratively monitor all kinds of information, including light, sound, temperature, humidity, acceleration, voltage, motion, radiation, etc. Using the analogy of the human nervous system, the numerous sensor nodes are just like our digital receptors to monitor the physical world. Moreover, the information collected by the sensor networks will be transmitted over the networks, analogous to our digital spinal cord and digital neural networks. Finally, the information will be processed by digital computers, analogous to our digital brain. Hence a "digital nervous system" can be created for us to instrument the physical world at a global scale.

A sensor node is made up of four basic components: a *sensing unit*, a *processing unit*, a *transceiving unit* and a *power unit* [1]. The sensing unit may have any number of sensors which can detect different types of signals (light, sound, temperature, etc.). The sensing unit also may include analog-to-digital converters to convert the measurements into digital data. The processing unit, which usually contains a small data-storage unit, coordinates the cooperation with other sensor nodes and manages the activation/hibernation schedule. The transceiving unit may contain wired or wireless communication components in order to report sensor measurements. Finally, the power unit may use power sources like battery or solar energy [59], depending on the cost of manufacturing and application needs. Currently, companies like Arch Rock, Crossbow, Dust Networks, Millennial Net, and Moteiv offer various types of sensor nodes, and [28] gives an overview of the hardware platform of sensor nodes.

Sensor networks can be used in many different applications, ranging from scientific to educational to military. For example, in environmental applications, sensor networks can be used to track wild birds on a deserted island [56]. Before the advent of sensor networks, estimating the number of wild birds was a daunting task because birds migrate over a large area. Deploying a sensor network can easily solve this problem by covering the entire area with low-cost sensor nodes that are equipped with sound sensors to keep track of the specific bird chirp. Another example is to embed sensor networks in the physical environment to construct a developmental problem-solving environments for early childhood education [54]. This allows “person to physical world communications” as opposed to traditional “person to person” or “person to computer” communications. This is a natural application of sensor networks as young children learn by exploring and interacting with objects such as toys in their environment. Finally, a military application is the VigilNet system [27]. He et al. [27] describe the design and implementation of a sensor network system, VigilNet, specifically for military surveillance purposes. In the VigilNet project, 70 sensor nodes are deployed in a 280-foot long perimeter in a grassy field that would typically represent a critical military “choke point”. The VigilNet system can track moving vehicles in the surveillance area using the sensor network. These three example applications of sensor networks are among the new applications which continue to be discovered and envisioned by scientists around the world.

1.2 Localization and Tracking in Sensor Networks

In order to realize the immense potential of sensor networks, one of the critical capabilities of a sensor network is to keep track of the specific objects of interest. If the object of interest is *static*, that is, its spatial coordinates do not change in time, we denote it as an *event*. The algorithm that the sensor network uses to identify the spatial position of the static event is denoted as an *event-localization* algorithm, or more briefly, a *localization* algorithm. After the event has been detected and localized, it may move or be moved by an external force. We denote a moving object of interest as a *target*.

In an event-localization scenario, three or more sensor nodes will be turned on at all times (we will explain the details in Chapter 2). Once an event occurs, those sensor nodes which have detected the event will report their measurements. Based on the measurements, we can run localization algorithms to find out the location of the event.

When the target is moving, we may use *target-tracking* algorithms (or simply *tracking* algorithms) to continuously estimate its current position at real-time rates.

Figure 1.1 gives illustrations of localization and tracking scenarios.

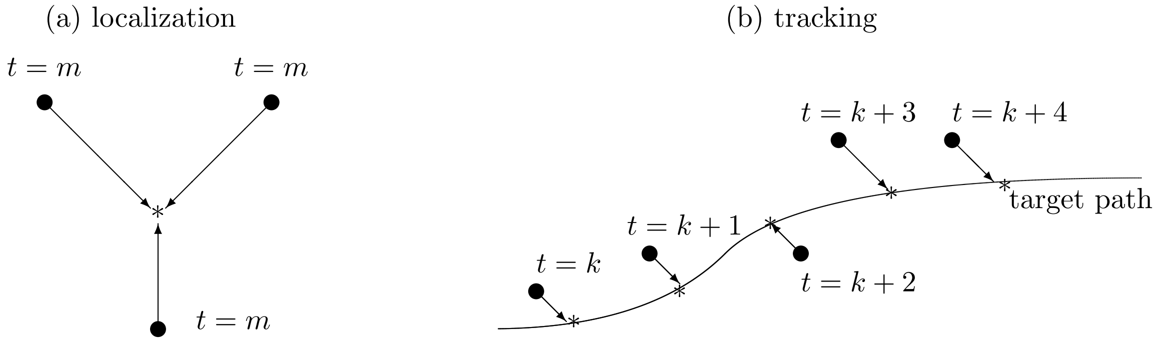


Figure 1.1: Illustration of localization and tracking. The filled circles represent sensor nodes. In (a), the star represents the static event, and 3 nodes are active at the same time. In (b), the stars represent the target positions at each time instant along the target path, and one new node is activated at each successive time instant. Precise definition of "successive time instant" will be defined later.

Although the purposes of both localization and tracking algorithms are to estimate the spatial position of the object of interest, subtle differences between the two algorithms exist. We will explain the details of localization and tracking later in this dissertation. To give the reader an overview, the differences between a localization algorithm and a tracking algorithm, as defined in this dissertation, are:

- The localization process happens at the same time instant; while the tracking process takes many time steps.

- A localization algorithm usually requires three or more sensor nodes activated at any time; while a tracking algorithm may require as few as one node [47]. Details of localization and tracking will be given in later sections.

In [44], Li et al. provide a general framework to detect an event using localization algorithms and subsequently track its movement using tracking algorithms, as illustrated in Figure 1.2. In Figure 1.2, the monitored area is divided into different cells. A cell is defined as a local collection of nodes, all of which are active at a given time. In order to preserve battery power, not all cells are simultaneously activated. Note that although Figure 1.2 shows disjoint cells, this definition does not necessarily require disjoint, non-overlapping cells. They also assume (as does this dissertation) that there is a *central processing unit* (CPU) that collects the information from all of the sensor nodes in a specific collection of cells. The basic approach for localization and tracking a single target is reproduced and summarized as follows [44] (for complete details, please refer to [44]):

1. Some and perhaps all of the nodes in cell A detect the target. These nodes are the "active nodes", and cell A is the "active cell". The active nodes report their measurements to the CPU.

2. At each time instant, the CPU determines the location of the target from the measurements of the active nodes.

3. The CPU uses locations of the target at the N successive time instants to predict the location of the target at M future time instants.

4. The predicted positions of the target are used by the CPU to activate new cells that the target is likely to enter. This is illustrated in Figure 1.2(b) where cell B represents the region that the target is likely to enter after the current active cell (cell A in Fig 1.2(a)).

5. Once the target is detected in the new cell, it is designated as the new active cell and the nodes in the original active cell (cell A in Fig 1.2(a)) may be put in the standby state to conserve energy.

The steps 1 - 5 are repeated for the new active cell, and this forms the basis of a sensor network monitoring system. Chu et al. [13] also provides a similar mechanism for the localization and tracking of targets.

Note that once the target is moving, it is possible to activate all of the sensor nodes in the network and use those nodes who detect the target to perform localization at each successive time step. In other words, we could activate all nodes in the network, and performing “tracking” by localization at each time step without using the actual tracking algorithm. However, activating all of the sensor nodes in the network will consume much more power than needed since at each time step only those nodes near the current target position can detect it. Those nodes who are not in the vicinity of the target are turned on unnecessarily. Hence tracking algorithms allow us to only activate necessary nodes. Those nodes not in the active cell can be put in the standby state to conserve energy.

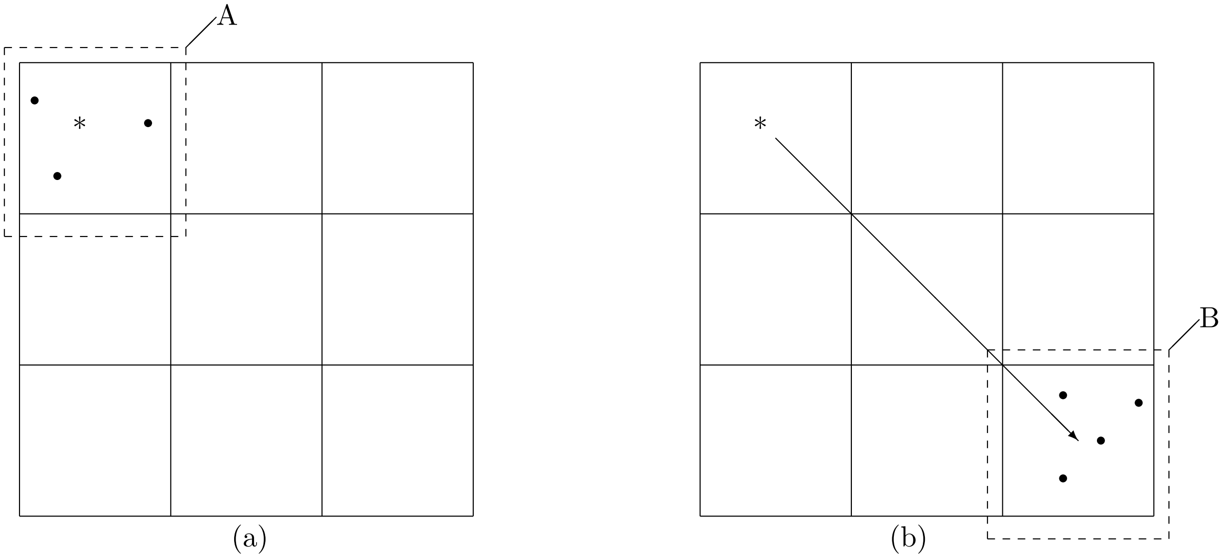


Figure 1.2: A schematic illustrating localization and tracking of a single target [44]. The target is shown as an asterisk. (a) The target enters cell A, and cell A becomes the active cell. Nodes in cell A are activated, and report their measurements to the CPU. The CPU calculates the position of the target. (b) The localization information when the target is inside A is used by the CPU to predict the future position of the target. Cell B is the new cell that the target is likely to enter, hence all nodes in cell B will be activated.

1.3 Security in Sensor Networks

Sensor networks are often deployed in remote or hostile terrains, hence they are often susceptible to various attacks. Ilyas et al. [33] provide several survey articles on different types of security issues in sensor networks. For the purpose of this dissertation, we define two types of sensor nodes: *malicious* and *benign*. The purpose of malicious nodes is to lower the accuracy of the localization and tracking processes. The adversary may physically capture some unknown number of existing benign nodes, and replace them with malicious ones. Or, the adversary may remotely login to the sensor nodes and turn them into malicious nodes, assuming that such network connection is available.

Furthermore, we define malicious nodes to be able to authenticate correctly with the sensor network. The malicious nodes also use the same type of encryption codes to communicate with the other nodes in the network. Malicious nodes can inject false localization or tracking reports into the network without being detected using authentication or encryption methods. In this dissertation, we will provide novel algorithms to detect those malicious nodes, which are assumed to be less than half of the total nodes inside the sensor network, based solely on their behavior.

1.4 Overview of the Dissertation

In dissertation, two interrelated problems are presented: secure localization and secure tracking. In both localization and tracking, there is an unknown number of malicious nodes in the network (we generally assume that the number of malicious nodes are fewer than benign nodes). The malicious nodes will attempt to lower the accuracy of localization and tracking by reporting that the target is at a fictitious location which is different from the true target location. Furthermore, we assume that malicious nodes will agree (collude) on that fictitious position. Our objective is to detect those malicious nodes.

We propose a novel algorithm to detect malicious nodes, in both localization and tracking scenarios [11, 9, 10]. The rest of this dissertation is to explain our algorithm, and it is organized as follows:

In Chapter 2, we will provide the background on secure localization. Similarly, background on the secure tracking problem is provided in Chapter 3.

Chapter 4 gives the background on the *relaxation labeling* algorithm. The relaxation labeling algorithm is a prominent method to reduce ambiguity, and we will extend it in Chapter 5 and Chapter 6.

Chapter 5 is our proposed algorithm to solve the secure localization problem. It corresponds to the problem defined in Chapter 2. Chapter 6 is our proposed algorithm to solve the secure tracking problem, and it corresponds to the problem defined in Chapter 3.

Finally, Chapter 7 is the conclusion and discussions on future work.

Chapter 2

Background on Localization

2.1 Sensor Models

In order to estimate the location of events, an adequate model of the sensors is needed. Given an adequate sensing model, we may derive location estimation algorithms for such sensors. We denote the time-dependent sensor measurement from sensor node i as $z_i(t)$. We denote $\mathbf{x}(t)$ as the unknown event-related parameters that we would like to estimate, and in this dissertation $\mathbf{x}(t)$ is the unknown event location in a two-dimensional space. One general model to relate $z_i(t)$ to $\mathbf{x}(t)$ is given as [13]

$$z_i(t) = h(\mathbf{x}(t), w_i), \qquad i = 1, 2, \cdots, n \tag{2.1}$$

where $h : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}$ is a (possibly) non-linear function which depends on $\mathbf{x}(t)$ and w_i. w_i is additive, signal-independent Gaussian noise whose variance σ^2 is assumed to be known. Existing literature on sensor networks commonly uses two special cases of the sensor model given in (2.1): acoustic amplitude sensors and acoustic array sensors. We do not specifically define the form of h here because they could be different in both types of sensors. Please see details in the next sections.

2.1.1 Acoustic Amplitude Sensors

Acoustic amplitude sensors provide a range estimate as follows [40]: First, we assume that acoustic signals propagate isotropically. We denote the known sensor node position as $\boldsymbol{\xi}_i$ for the ith sensor node. For acoustic amplitude sensors, (2.1) becomes

$$z_i = \frac{a}{\|\mathbf{x} - \boldsymbol{\xi}_i\|^{\frac{\alpha}{2}}} + w_i, \tag{2.2}$$

where a is the given amplitude of the signal at the event position $\mathbf{x}$, α is a known attenuation coefficient, and $\|\|$ is the Euclidean norm. Equation (2.2) means that z_i should go up as a goes up, except for the influence of noise. Similarly, z_i will be smaller if the distance from the node to the event is farther, except for the influence of noise. Note that $\mathbf{x}$ and $\boldsymbol{\xi}_i$ are both vectors denoting spatial locations, where $\mathbf{x}$ is the (unknown) spatial location of the event, and $\boldsymbol{\xi}_i$ is the known spatial location of the sensor node i. For the purpose of this dissertation, both $\mathbf{x}$ and $\boldsymbol{\xi}_i$ are in the two-dimensional space (more details will be given in Figure 2.1).

Note that the model in (2.2) is also adopted in modeling types of sensors other than acoustic amplitudes. For example, [37] uses this model in radiation sensors.

In this dissertation, we assume that the sensor node has a limited sensing range. When the event is out of the sensing range of the node, or when the event amplitude is too small, the sensor node will not be able to detect such an event. Hence we do not consider the exceptional case when $\frac{a}{\|\mathbf{x}-\boldsymbol{\xi}_i\|^{\frac{\alpha}{2}}} \simeq 0$ and the received signal consists only of the noise term, $z_i \simeq w_i$.

2.1.2 Acoustic Array Sensors

Acoustic array sensors are essentially arrays of microphones, and signal processing algorithms such as beam-forming [20, 12] may be used to estimate the location of the event. Acoustic array sensors can measure the *Direction of Arrival* information of the target, and are another example of the general sensor model in (2.1). However, in this dissertation, we assume that each node has a single sensor, not an array. Readers interested in array sensors should refer to [61] and the references therein.

2.2 Localization Algorithms

The sensor model in (2.2) is adaptable to different types of sensors since it has been used to model acoustic amplitude sensors [40] and radiation sensors [37]. As mentioned in

Chapter 1, the sensor in a sensor node could be a sound sensor, a pressure sensor, a light sensor, and so on. In order to model as many different types of sensors as possible, the sensor model in (2.2) is the most suitable one. Equation (2.2) simply states that an event, which has an associated event amplitude, decays as the distance gets larger (to the power of the attenuation constant). This can be used to model many physical phenomena. Therefore, we will use (2.2) as the sensor model.

2.2.1 Localization of a Single Event

The most common localization algorithm is *triangulation* [61]. Similar concepts of triangulation by solving a linear system can also be found in *Global Positioning Systems (GPS)* [62] to locate a GPS module or *Time of Arrival (TOA)* techniques in cellular phone systems [63]. In triangulation, we form a system of linear equations of the unknown $\mathbf{x}$. We may estimate $\mathbf{x}$ by inverting the system matrix, as described below.

We assume that a single target exists and is detected by sensor i. We introduce a term *range* for each sensor node, and it is denoted as δ_i which is a measurement of $\|\mathbf{x}-\boldsymbol{\xi}_i\|$. Letting $\alpha = 4$ in (2.2), δ_i is defined by

$$z_i = \frac{a}{\delta_i^2} \tag{2.3}$$

Or equivalently,

$$\delta_i = \sqrt{\frac{a}{z_i}} \tag{2.4}$$

Note that in (2.3) and (2.4), the noise term in (2.2) is implicit in the measurement z_i

$$\delta_i = \sqrt{\frac{a}{z_i}} = \sqrt{\frac{a}{\frac{1}{\|\mathbf{x}-\boldsymbol{\xi}_i\|^2} + w_i}} \tag{2.5}$$

Finding the unknown location of the event, $\mathbf{x}$, can then be posed as minimizing the following objective function

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{argmin} \sum_{i=1}^{n} \left(\delta_i - \|\mathbf{x} - \boldsymbol{\xi}_i\|\right)^2 . \tag{2.6}$$

Triangulation is a method to approximate the solution of (2.6). Since we assume sensor nodes are deployed in a two-dimensional space, we denote the known location of each sensor node as $\boldsymbol{\xi}_i = [x_i \quad y_i]^T$ and the unknown location of the event as $\mathbf{x} = [x_0 \quad y_0]^T$. Then

$$(x_i - x_0)^2 + (y_i - y_0)^2 = \delta_i^2, \qquad i = 1, 2, \cdots, n \tag{2.7}$$

Equation (2.7) basically says that using the location of the sensor node as the center and the range δ_i as the radius, we can plot a circle. The location of the event lies somewhere on the circle. In (2.7), if we subtract the $i = 1$ equation from the the other equations, we obtain a system of linear equations in x_0 and y_0 of the form

$$2(x_i - x_1)x_0 + 2(y_i - y_1)y_0 = (x_i^2 - x_1^2) + (y_i^2 - y_1^2) + \delta_1^2 - \delta_i^2, \qquad i = 2, \ldots, n \tag{2.8}$$

Defining

$$\mathbf{A} = \begin{bmatrix} 2(x_2 - x_1) & 2(y_2 - y_1) \\ \vdots & \vdots \\ 2(x_n - x_1) & 2(y_n - y_1) \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} (x_2^2 - x_1^2) + (y_2^2 - y_1^2) + \delta_1^2 - \delta_2^2 \\ \vdots \\ (x_n^2 - x_1^2) + (y_n^2 - y_1^2) + \delta_1^2 - \delta_n^2 \end{bmatrix},$$

we get a linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{2.9}$$

where $\mathbf{A}$ is referred to as the "system matrix". If $\mathbf{A}$ is invertible, the location of the event can be estimated by

$$\hat{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{b}. \tag{2.10}$$

In general, we need at least 2 sensors in a one-dimensional space to resolve ambiguity. Similarly, we need at least 3 sensors in a 2D problem, as illustrated in Figure 2.1. Without loss of generality, we will assume that sensor nodes are deployed in a two-dimensional space from this point on.

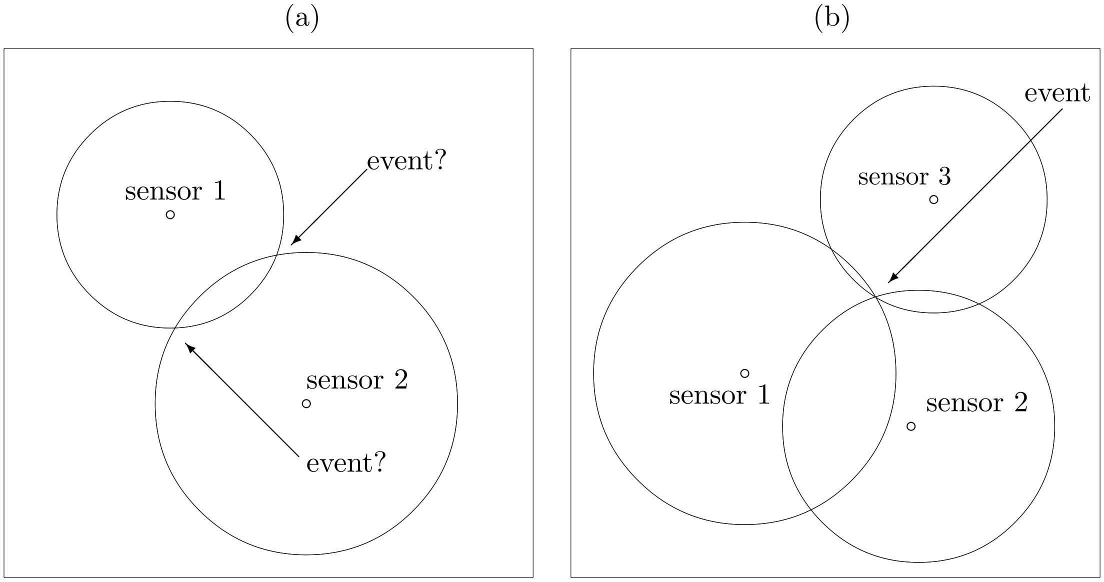


Figure 2.1: Illustration of how at least 3 sensors are needed to resolve ambiguity in a 2D problem. In (a), two sensor nodes create ambiguity; while in (b), a unique solution can be found.

In the cases when we have more than enough sensors ($n > 3$), it becomes an over-determined problem, and a mean-squared-error estimate of the location of the event can be determined using the pseudoinverse

$$\hat{\mathbf{x}} = \left[\mathbf{A}^T\mathbf{A}\right]^{-1}\mathbf{A}^T\mathbf{b}. \tag{2.11}$$

under the assumptions that the inverse of $\mathbf{A}^T\mathbf{A}$ exists. Geometrically, since (2.7) is drawing a circle, solving (2.9) is equivalent to estimating the common intersection of all of the circles. What is notable is that one sensor alone cannot determine the location of the event; it requires collaboration of multiple sensors to solve the event-localization problem. Hence event-localization is a typical example of *collaborative signal processing* [61].

2.2.2 Localization of Multiple Events

It is possible to extend the localization algorithm to estimate the locations of multiple simultaneous events. In this case, the sensor model becomes a mixture

$$z_i = \sum_j \frac{a_j}{\|\mathbf{x}_j - \boldsymbol{\xi}_i\|^{\frac{\alpha}{2}}} + w_i, \tag{2.12}$$

in which a_j is the amplitude of each event. In this dissertation, we do not consider multiple events in secure localization problems. Please refer to [8] for a discussion of localization of multiple events.

2.3 Related Localization Literature

In this dissertation, localization refers to finding the location of the event, not the location of the sensor node itself. There exists literature in which a sensor network is deployed with only a portion of the nodes knowing their locations [32]. In that literature, localization refers to using those nodes who know their own locations to determine the locations of the rest of the nodes in the network [32]. Although related to this dissertation, the objective (determining $\boldsymbol{\xi}_i$) is distinctive from our objective, determining $\mathbf{x}$.

In the robotics literature, there is a topic on *Simultaneous Localization and Mapping* (SLAM) [14]. SLAM refers to the process in which a mobile robot explores and maps the current environment while keeping track of its own current position. What a mobile robot localizes is the current position of itself, and this is also distinctive from what we mean by localization in this dissertation.

Also note from our sensor model in (2.2) that we do not have time information from the event since we cannot know when the event occurred in advance. Hence we cannot measure how long it takes for the "signal" to propagate from the event position to the node position. There exists localization techniques to determine the location of a mobile subscriber in a cellular or wireless local area network (WLAN) environments [53]. Those techniques include *Time of Arrical*, *Time Difference of Arrival* and *Angle of Arrival*, but because of our assumptions, they will not be discussed or used in this dissertation.

2.4 Security Problem in Localization

2.4.1 Problem Statement

Consider the case when we have deployed a sensor network, and all of the node positions have been determined and calibrated. All of the nodes are active and listen to the environment for possible event occurrences, and those who have detected the event will report it. An unknown number of benign nodes are replaced with malicious nodes by the adversary. The problem of *secure localization* [43] is to correctly identify the location of the event under the influence of malicious nodes. Further detail follows.

2.4.2 Problem Definition

For the n nodes that have detected some particular event, each of their locations, (x_i, y_i), and range reports δ_i are known. We denote a set of 3 nodes as a *triple*. Since $n > 3$, we have $\left(\begin{array}{c} n \\ 3 \end{array} \right) = \frac{n!}{(n-3)!3!} = \frac{(n-2)\cdot(n-1)\cdot n}{1\cdot 2\cdot 3}$ triples. For any triple, we can readily obtain an estimate of the event location, $\hat{\mathbf{x}} \equiv (\hat{x_0}, \hat{y_0})$, using triangulation.

There is an unknown number of malicious nodes inside the network. We generally assume that malicious nodes are fewer than benign nodes in the newtork. We define a malicious node to have the following properties

1. The objective of the malicious node is to lower the accuracy of the localization of events by injecting false localization reports into the network.
2. All of the malicious nodes collude on a "*fictitious event*". The fictitious event location is the location of the event, as reported by the malicious nodes. It is almost always different from the true location of the event.
3. Malicious nodes can successfully authenticate with any other node or the central server of the network, and they also have obtained the encryption key used by existing nodes. Hence malicious nodes can successfully communicate with the network (and each other), and any effort to use encryption or authentication to detect them will be futile.

For a triple whose nodes are located at

$$\boldsymbol{\xi}_a, \quad \boldsymbol{\xi}_b, \quad \boldsymbol{\xi}_c, \qquad a, b, c \in \{1, 2, \cdots, n\}, \qquad a \neq b, \qquad b \neq c, \qquad c \neq a,$$

we denote the locations of the nodes of that particular triple by

$$\boldsymbol{\xi}_l, \qquad l \in \{a, b, c\}$$

and similarly the subscript l is used for other data or parameters belonging to the nodes of that particular triple.

Within a particular triple, we have the ranges δ_l, $l \in \{a, b, c\}$, calculated using the measurements and (2.4). We also have the estimated location of the target, $\hat{\mathbf{x}} = (\hat{x_0}, \hat{y_0})$, determined by (2.10)

$$\hat{\mathbf{x}} = (\hat{x_0}, \hat{y_0}) = \mathbf{A}^{-1}\mathbf{b}.$$

Then, for each node in the triple, we may define a "*discrepancy*" as [46]

$$\epsilon_l = |\delta_l - \sqrt{(\hat{x_0} - x_l)^2 + (\hat{y_0} - y_l)^2}|, \qquad l \in \{a, b, c\} \tag{2.13}$$

since the location of each sensor node, (x_l, y_l), is known. In (2.13), the discrepancy has two major contributors: noise and (possibly) false δ_l generated by malicious nodes. Hence the problem is to classify which sensor nodes are malicious using our currently-available information:

- (x_l, y_l), the location of the sensor nodes
- δ_l, the range estimate reported by the sensor node (or the measurement, z_l)
- $(\hat{x_0}, \hat{y_0})$, the estimated event location based on z_l
- ϵ_l, the error term

Note that we allow malicious nodes to know their exact location, (x_l, y_l), by either physically capturing existing nodes which hold location information or breaking into our system to obtain the system coordinates.

2.4.3 Supplemental Properties

The event is static

In this section, we assume that the source of the event detected by the sensor network is not moving, or moving slowly enough that its motion is negligible relative to the measurement process.

Redundancy of the sensor measurements

We assume that when an event occurs, there are more than enough ($n > 3$) nodes that have detected the event. The reason why the required number of nodes is greater than three is that we need more than one triples. If $n = 3$, there exists only one triple and we cannot use the consistency among triples to detect malicious nodes.

Fictitious event

The fictitious event location has to fall within the sensing range of all the active nodes. Otherwise, the benign nodes can simply tell which node is lying.

Centralized model

Since we are considering a secure localization scheme, we assume a centralized system in which a central processor will collect information from all active nodes and calculate the event location. This is due to the fact that some of the nodes may be malicious, hence we need a central processor to make the final decision. Secure localization methods in ad hoc systems [43] are not covered in this dissertation.

2.4.4 Problem Analysis

The challenge of this problem is that we do not know which sensor is malicious in the first place, so $(\hat{x_0}, \hat{y_0})$ is possibly erroneous. Hence the discrepancy ϵ_l depends on whether or not the node is malicious. Therefore, the problems of classification of sensors and event localization are like a "chicken and egg" problem - whichever comes first will solve the other. If we knew which sensors are malicious in the first place, we could use only benign sensor measurements to do the localization and find the correct event location. If we knew where the event is located, (x_0, y_0), we could use it to immediately tell which node is lying by verifying δ_i and $\sqrt{(x_0 - x_i)^2 + (y_0 - y_i)^2}$.

The malicious nodes in the sensor network could be either *non-colluding* or *colluding*. For non-colluding attacks, the malicious nodes would independently generate random location estimation reports into the network; while colluding nodes could uniformly report a new, fictitious event. Current research have proposed methods including *mean-squared-error* [46] and *statistical filtering* [60, 65, 17] to detect malicious nodes. When the attackers are non-colluding, using mean-squared-error and statistical filtering can detect those attackers who are outliers to the community. However, when the attackers are colluding, they become a consistent group which is harder to detect. This dissertation focuses on colluding attacks and uses relaxation labeling to poll every triple in the network, and can detect even a large number of malicious nodes in the network.

Before we discuss the new algorithm in detail, we would like to point out that the philosophy of relaxation labeling algorithms is to use “local information” to achieve global consistency. Classical relaxation labeling algorithms use pairs of objects (local information) to resolve ambiguity of the entire system (global consistency). Due to the nature of our problem, we extend classical relaxation labeling algorithms to use triples. However, we do not need quadruples or any higher-order compatibility functions because the philosophy of relaxation labeling is to strive to use as local information as possible. Classical relaxation labeling algorithms will be reviewed in Chapter 4. A new relaxation labeling algorithm will be proposed based on triples of sensor nodes, instead of pairs, in Chapter 5.

Chapter 3

Background on Tracking

3.1 Target Tracking

As discussed in Section 1.2, the use of tracking algorithms allows us to estimate the current position of the target using fewer sensor nodes. As a result, less power is consumed. In this chapter, we will describe the concept of target tracking and propose the problem of secure tracking.

3.1.1 System Models

In target tracking, we aim to estimate the current *state* of the target(s) based on available information, including past target states and measurements. The state of the target can be location, velocity, orientation, etc. In our sensor model, the nodes can get only a range estimate of the location of the target(s). Hence in this dissertation, we will refer to the location and velocity of the target(s) in a two-dimensional space as the state of the target(s). We extend the notation of the previous chapter to denote $\{\mathbf{x}^k, k \in \mathbb{N}\}$ as the state of the target(s) at time k, where $\mathbf{x}^k = \begin{bmatrix} x^k & y^k & \dot{x}^k & \dot{y}^k \end{bmatrix}^T$.

In target tracking using sensor nodes as described in the previous chapter, two models form the foundation of all algorithms: the *motion model* of the target positions and the *measurement model* of the sensor nodes. A good motion model of the target will certainly facilitate the extraction of the information about the target states from sensor-node observations [45]. Consider the evolution of state sequence $\{\mathbf{x}^k, k \in \mathbb{N}\}$ of a target given by

$$\mathbf{x}^k = \mathbf{f}^k(\mathbf{x}^{k-1}, \mathbf{v}^{k-1}) \tag{3.1}$$

where $\mathbf{x}^k$ is the state of the target at time k, $\mathbf{f}^k : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}^4$ is a possibly nonlinear function, and $\mathbf{v}^{k-1}$ is a noise sequence. $\mathbf{v}^{k-1}$ models the unpredictable disturbances [4] while the target is moving. The disturbances are assumed independent of the target motion, and we do not assume any relation between the elements in the noise sequence. Hence $\mathbf{v}^{k-1}$ is an independently and identically distributed (i.i.d.) noise sequence. Note that in (3.1), $\mathbf{x}^k$ depends only on $\mathbf{x}^{k-1}$, but not on any of the previous states $\{\mathbf{x}^i, i = 1, \cdots, k-2\}$. Hence, for the purpose of this dissertation, (3.1) defines a Markov process of order one.

The other model of interest is the measurement model of the sensor nodes, and it is given by

$$z_i^k = h(\mathbf{x}^k, w_i^k) \tag{3.2}$$

where z_i^k is node i's measurement from the target at time k, $h : \mathbb{R}^4 \times \mathbb{R} \to \mathbb{R}$ is a possibly nonlinear function, and w^k is an i.i.d. noise sequence. Similar to $\mathbf{v}^{k-1}$ in (3.1), we assume that each w^k is identically distributed, and each w^k is independent from each other. Hence w^k is i.i.d. w^k is also assumed to have no relation to the target state, $\mathbf{x}^k$. In this dissertation, we consider only single-target tracking problems, hence in (3.2), $\mathbf{x}^k$ represents the state of *the* target at time k. We use the amplitude sensor model in Section 2.1.1 since it models a general amplitude measurement and can be applied to many different types of signals. Hence (3.2) follows (2.2) as

$$z_i^k = \frac{a}{|\mathbf{x}^k - \boldsymbol{\xi}_i|^{\frac{\alpha}{2}}} + w_i^k \tag{3.3}$$

where a is the amplitude of the signal emitted from the target and it is assumed to be known. In (3.3), $\boldsymbol{\xi}_i$ is the location of node i, and it is also assumed to be known. Note that the measurement model in (3.3) is nonlinear and the motion model in (3.1) could also be nonlinear.

3.1.2 Tracking Algorithm

In a tracking problem, our purpose is to estimate the *probability density function* (pdf) $p(\mathbf{x}^k|z^{1:k})$ that the target is in state $\mathbf{x}^k$, based on measurements $z^{1:k} = \{z^i, i = 1, \cdots, k\}$. We have two stages at each time step k: *prediction* and *update*. Suppose that the pdf $p(\mathbf{x}^{k-1}|z^{1:k-1})$ is available. In the prediction stage, we obtain the prior pdf of $\mathbf{x}^k$ via the Chapman-Kolmogorov equation [2]

$$p(\mathbf{x}^k|z^{1:k-1}) = \int p(\mathbf{x}^k|\mathbf{x}^{k-1})p(\mathbf{x}^{k-1}|z^{1:k-1})d\mathbf{x}^{k-1}. \tag{3.4}$$

In (3.4), the probabilistic model of $p(\mathbf{x}^k|\mathbf{x}^{k-1})$ is defined by (3.1) and the known statistics of $\mathbf{v}^{k-1}$. Moreover, in (3.4), the pdf $p(\mathbf{x}^{k-1}|z^{1:k-1})$ is known from the previous time step, $k-1$. Hence in (3.4) we obtain $p(\mathbf{x}^k|z^{1:k-1})$, the belief that the state at time k is $\mathbf{x}^k$ given past measurements.

As mentioned in the previous section, $\mathbf{v}^{k-1}$ models the unpredictable disturbances of the target motion [4]. Although $\mathbf{v}^{k-1}$ is a random variable, its statistics have to be known in order to derive solutions to the tracking problems.

Our eventual goal is to estimate $p(\mathbf{x}^k|z^{1:k})$, hence we need the update stage.

The update stage starts as the measurement z^k becomes available. We may obtain the desired $p(\mathbf{x}^k|z^{1:k})$ using Bayes rule as

$$p(\mathbf{x}^k|z^{1:k}) = \frac{p(z^k|\mathbf{x}^k)p(\mathbf{x}^k|z^{1:k-1})}{p(z^k|z^{1:k-1})} \tag{3.5}$$

where the normalizing constant

$$p(z^k|z^{1:k-1}) = \int p(z^k|\mathbf{x}^k)p(\mathbf{x}^k|z^{1:k-1})d\mathbf{x}^k \tag{3.6}$$

depends on the likelihood function $p(z^k|\mathbf{x}^k)$. The likelihood function, $p(z^k|\mathbf{x}^k)$, is determined by the measurement model and the known statistics of $\mathbf{w}_i^k$ in (3.2). The other term in (3.6), $p(\mathbf{x}^k|z^{1:k-1})$, is already obtained using (3.4). Hence we can calculate $p(z^k|\mathbf{z}^{1:k-1})$ in (3.6). Similarly, when we look at (3.5), we have the likelihood function $p(z^k|\mathbf{x}^k)$ from (3.2), $p(\mathbf{x}^k|z^{1:k-1})$ from (3.4), and $p(z^k|z^{1:k-1})$ from (3.6), so the desired pdf $p(\mathbf{x}^k|z^{1:k})$ is obtained.

The recursive prediction and update relations of (3.4) and (3.5) form the optimal Bayesian solution to the tracking problem. They are only a conceptual solution in general since they usually cannot be determined analytically [2]. Solutions do exist under strict conditions, which we will cover in the next section.

Note that the initial pdf $p(\mathbf{x}^0|z^0) \equiv p(\mathbf{x}^0)$ is also assumed to be given. In our simulations, $p(\mathbf{x}^0)$ is a given unit-variance Gaussian distribution centered at $\mathbf{x}^0$.

Kalman Filter

There exists an optimal solution to the tracking problem, the Kalman filter [38, 39], which was derived based on (3.4) and (3.5). The Kalman filter assumes that the system models are Gaussian and linear [38, 39]. That is, (3.1) becomes

$$\mathbf{x}^k = \mathbf{F}^k \mathbf{x}^{k-1} + \mathbf{v}^{k-1} \tag{3.7}$$

where $\mathbf{F}^k$ is a known system matrix. Meanwhile, (3.2) becomes

$$z^k = \mathbf{H}^k \mathbf{x}^k + \mathbf{n}^k \tag{3.8}$$

where $\mathbf{H}^k$ is also a known measurement matrix[1]. $\mathbf{v}^{k-1}$ and $\mathbf{n}^k$ are zero-mean Gaussian noises and have covariances of $\mathbf{Q}^{k-1}$ and $\mathbf{R}^k$, respectively. Also, $\mathbf{v}$ is independent from $\mathbf{n}$. Note that $\mathbf{n}^k$ is different from the w_i^k in (3.2) because $\mathbf{n}^k$ is strictly zero-mean Gaussian with known covariance $\mathbf{R}^k$ (for details of the derivation of the Kalman filter please refer to Appendix A).

Under such Gaussian and linear conditions, the Kalman filter can be derived as a recursive relationship involving $\mathbf{F}^k$, $\mathbf{H}^k$ and the statistics of $\mathbf{v}^{k-1}$ and $\mathbf{n}^k$. Relation of the Kalman filter to our target tracking problem will be presented in the following sections.

The restrictions of Gaussianity and linearity of the Kalman filter algorithm is often impractical. Hence there exists two major types of nonlinear tracking algorithms: the *extended Kalman filter* (EKF) [35] and *particle filters* [18]. They are also referred to as

[1] We observe that our measurement model, eq. (3.2), cannot be written in this way

suboptimal algorithms since the Kalman filter is the optimal solution under Gaussian and linear assumptions.

Extended Kalman Filters

The EKF [35] allows the motion model in (3.1) and the measurement model in (3.2) to become nonlinear. The concept of EKF is to use local linearization of the equations, (3.1) and (3.2). In the EKF algorithm, we use the notation of $\mathbf{m}^{t_1|t_2}$ to denote the expected value $\mathbf{m}$ at time t_1 based on the measurement that was obtained at time t_2. Assuming that the expected value of the target position, $\mathbf{m}^{k-1|k-1}$, and the covariance of its error, $\mathbf{P}^{k-1|k-1}$, are both given from the previous time step ($t = k - 1$), and the pdf at $t = k - 1$ is Gaussianly distributed

$$p(\mathbf{x}^{k-1}|z^{1:k-1}) \approx \mathcal{N}(\mathbf{x}^{k-1}; \mathbf{m}^{k-1|k-1}, \mathbf{P}^{k-1|k-1}) \tag{3.9}$$

where $\mathcal{N}(\mathbf{x}, \mathbf{m}, \mathbf{P})$ denotes a Gaussian distribution of variable $\mathbf{x}$ whose mean is $\mathbf{m}$ and whose covariance is $\mathbf{P}$. The EKF solution to the tracking problem can be shown to be [2]

$$p(\mathbf{x}^{k}|z^{1:k-1}) \approx \mathcal{N}(\mathbf{x}^{k}; \mathbf{m}^{k|k-1}, \mathbf{P}^{k|k-1}) \tag{3.10}$$

$$p(\mathbf{x}^{k}|z^{1:k}) \approx \mathcal{N}(\mathbf{x}^{k}; \mathbf{m}^{k|k}, \mathbf{P}^{k|k}) \tag{3.11}$$

where

$$\mathbf{m}^{k|k-1} = \mathbf{f}^{k}\left(\mathbf{m}^{k-1|k-1}\right) \tag{3.12}$$

$$\mathbf{P}^{k|k-1} = \mathbf{Q}^{k-1} + \hat{\mathbf{F}}^{k}\mathbf{P}^{k-1|k-1}\left(\hat{\mathbf{F}}^{k}\right)^{T} \tag{3.13}$$

$$\mathbf{m}^{k|k} = \mathbf{m}^{k|k-1} + \mathbf{K}^{k}\left(\mathbf{z}^{k} - \mathbf{h}^{k}\left(\mathbf{m}^{k|k-1}\right)\right) \tag{3.14}$$

$$\mathbf{P}^{k|k} = \mathbf{P}^{k|k-1} - \mathbf{K}^{k}\hat{\mathbf{H}}^{k}\mathbf{P}^{k|k-1} \tag{3.15}$$

and where $\mathbf{f}^k(\cdot)$ and $\mathbf{h}^k(\cdot)$ are the nonlinear functions from (3.1) and (3.2), $\hat{\mathbf{F}}^k$ and $\hat{\mathbf{H}}$ are local linearizations of these nonlinear functions

$$\hat{\mathbf{F}}^k = \left.\frac{d\mathbf{f}^k(x)}{dx}\right|_{x=\mathbf{m}^{k-1|k-1}}, \tag{3.16}$$

the Jacobian of $\mathbf{f}^k(x)$;

$$\hat{\mathbf{H}}^k = \left.\frac{d\mathbf{h}^k(x)}{dx}\right|_{x=\mathbf{m}^{k|k-1}}, \tag{3.17}$$

the Jacobian of $\mathbf{h}^k(x)$; and where

$$\mathbf{S}^k = \hat{\mathbf{H}}^k\mathbf{P}^{k|k-1}\left(\hat{\mathbf{H}}^k\right)^T + \mathbf{R}^k \tag{3.18}$$

$$\mathbf{K}^k = \mathbf{P}^{k|k-1}\left(\hat{\mathbf{H}}^k\right)^T \mathbf{S}^{k^{-1}}. \tag{3.19}$$

Particle Filters

Particle filter algorithms [26, 34, 7, 15], known collectively as *Sequential Monte Carlo* algorithms [18], represent the density function by a set of particles. The particle filter algorithm described in [26] was demonstrated to have better performance than the EKF, hence that algorithm was chosen for use in our dissertation.

In the particle filter algorithm, the key idea is to represent the required density function by a set of particles. A particle, $(\mathbf{x}(i), q(i))$, has two components: a sample of the state vector, $\mathbf{x}(i)$, and each sample is also associated with a weight, $q(i)$. To initiate the particle filter algorithm, N_s samples $\mathbf{x}^0(i), i = 1, \cdots, N_s$ are drawn from the known prior, $p(\mathbf{x}^0)$. Each sample is associated with a weight, $q^k(i)$, and $\sum_i^{N_s} q^k(i) = 1$. Then the desired density at iteration k can be approximated as

$$p(\mathbf{x}^k|z^{1:k}) = \sum_{i=1}^{N_s} q^k(i)\delta\left(\mathbf{x}^k - \mathbf{x}^k(i)\right) \tag{3.20}$$

which is a discrete weighted approximation of the desired pdf $p(\mathbf{x}^k|z^{1:k})$ and $\delta()$ is

$$\delta(t) = \left\{ \begin{array}{ll} 1, & t = 0 \\ 0, & t \neq 0 \end{array} \right. . \tag{3.21}$$

Recall that in a tracking problem, we have two stages: prediction and update. We now explain the particle filter algorithm proposed by Gordon et al. [26] at these two stages.

- Prediction

Recall that for a target tracking problem, we are given two models: the system model in (3.1) and the measurement model in (3.2). At the prediction stage, each sample, $\mathbf{x}^{k-1}(i)$, is passed through the system model in (3.1) to obtain a corresponding new sample, $\mathbf{x}^{k^*}(i)$ [26]

$$\mathbf{x}^{k^*}(i) = \mathbf{f}^k(\mathbf{x}^{k-1}(i), \mathbf{v}^{k-1}(i)), \qquad i = 1, \cdots, N_s \tag{3.22}$$

where $\mathbf{v}^{k-1}(i)$ is a sample drawn from the given pdf of the noise $p(\mathbf{v}^{k-1})$ in (3.1). Intuitively, this is a fast way of obtaining the new samples at $t = k$ since the system model (3.1) and the statistics of the noise are both known.

- Update

At the update stage, recall that the sensor node will make a measurement of the current target state. On receipt of the measurement z^k, we evaluate the likelihood of each sample, $p(z^k|\mathbf{x}^{k^*}(i))$, based on the known measurement noise properties in (3.2). Then, it can be derived (see [2]) that we can calculate the weights at $t = k$, $q^k(i)$, according to

$$q^k(i) = \frac{p(z^k|\mathbf{x}^{k^*}(i))}{\sum_{j=1}^{N_s} p(z^k|\mathbf{x}^{k^*}(j))} \tag{3.23}$$

where the denominator is the normalization to ensure $\sum_i^{N_s} q^k(i) = 1$ [26]. We summarize the particle filter algorithm in Table 3.1.

Table 3.1: Summary of the particle filter algorithm [2]

```
Given:
A set of particles $\{\mathbf{x}^{k-1}(i), q^{k-1}(i)\}_{i=1}^{N_s}$ and measurement $z^k$

Objective:
Calculate the updated particles $\{\mathbf{x}^k(i), \frac{1}{N_s}\}_{i=1}^{N_s}$

Algorithm:
1. (prediction)
    for i=1:$N_s$ {
        Generate a new sample by passing the old sample through the system model
        $\mathbf{x}^{k^*}(i) = \mathbf{f}^k(\mathbf{x}^{k-1}(i), \mathbf{v}^{k-1}(i))$
    }
2. (update)
    for i=1:$N_s$ {
            Calculate $q^k(i) = \frac{p(z^k|\mathbf{x}^{k^*}(i))}{\sum_{j=1}^{N_s} p(z^k|\mathbf{x}^{k^*}(j))}$
    }
3. (resampling)
    Resample $\{\mathbf{x}^{k^*}(i), q^k(i)\}_{i=1}^{N_s}$ using the resampling algorithm in Table 3.2
    to obtain $\{\mathbf{x}^k(i), \frac{1}{N_s}\}_{i=1}^{N_s}$
```

Using (3.22) and (3.23), we now have a set of particles $(\mathbf{x}^{k^*}(i), q^k(i))$, $i = 1, \cdots, N_s$. However, the update stage is not complete yet. There is a "degeneracy phenomenon" which is a common problem with particle filter algorithms [48]. The degeneracy problem occurs when after a few iterations, all but one sample will have negligible weights [2]. To avoid the degeneracy problem, Gordon et al. propose to perform a *resampling* procedure [26]. In the resampling procedure, we are given $(\mathbf{x}^{k^*}(i), q^k(i))$, $i = 1, \cdots, N_s$, and the output is a new set of particles $(\mathbf{x}^k(i), \frac{1}{N_s})$, $i = 1, \cdots, N_s$. After the resampling procedure, the update stage is complete. Note that $\mathbf{x}^{k^*}(i), i = 1, \cdots, N_s$ are only intermediate samples (hence the asterisk sign). Rather, the new $\mathbf{x}^k(i), i = 1, \cdots, N_s$ (after resampling) are what will be used and passed on to the next time step.

- Resampling

The basic idea of resampling is to eliminate samples that have small weights and concentrate on samples with large weights [2]. We begin by building a cumulative distribution function (CDF) of $q^k(j)$, and we denote it as c_j, $j = 1, \cdots, N_s$. We will also build another CDF of a uniformly distributed random variable, and we denote it as t_i, $i = 1, \cdots, N_s$. After we have these two CDF functions (both of equal length N_s), we begin at the bottoms of them, $i = 1$ and $j = 1$. Next we will iterate through c_j. As we traverse along c_j, we compare c_j with t_i. There are two cases:

1. If $c_j \geq t_i$, we will set $\mathbf{x}^k(i) = \mathbf{x}^{k^*}(j)$ and keep incrementing i (while setting $\mathbf{x}^k(i) = \mathbf{x}^{k^*}(j)$) until $c_j < t_i$.

2. If $c_j < t_i$, we will skip this $\mathbf{x}^{k^*}(j)$.

In other words, the resampling procedure is saying that when we encounter a weight, $q^k(j)$, that is larger than that which would result from a uniform distribution ($c_j > t_i$), we will make multiple copies of the $\mathbf{x}^{k^*}(j)$, corresponding to the same $q^k(j)$, to multiple $\mathbf{x}^k(i)$ (by incrementing i until $c_j < t_i$ again). When we encounter a weight, $q^k(j)$, that is less than what a uniform distribution would produce ($c_j < t_i$), the $\mathbf{x}^{k^*}(i)$ corresponding to $q^k(i)$ is skipped. This follows the basic idea of resampling.

Readers interested in other variations of the particle filter algorithms should refer to [18] and the references therein.

The particle filter algorithm in Table 3.1, in essence, is a process of producing the pdf $p(\mathbf{x}^k|z^k)$ given the previous pdf, $p(\mathbf{x}^k|z^{k-1})$, and the current measurement z^k. At every time step, only one measurement (of the target state) is required. Hence in a sensor network scenario, it is feasible to only activate one sensor node to make the measurement. This is the beauty of target tracking algorithms. Armed with the known models in (3.1) and (3.2), and also the known statistics of the noises in them, target tracking algorithms produce an estimate of the current target location, based only on one measurement.

Table 3.2: Procedure for resampling from the particles [2]

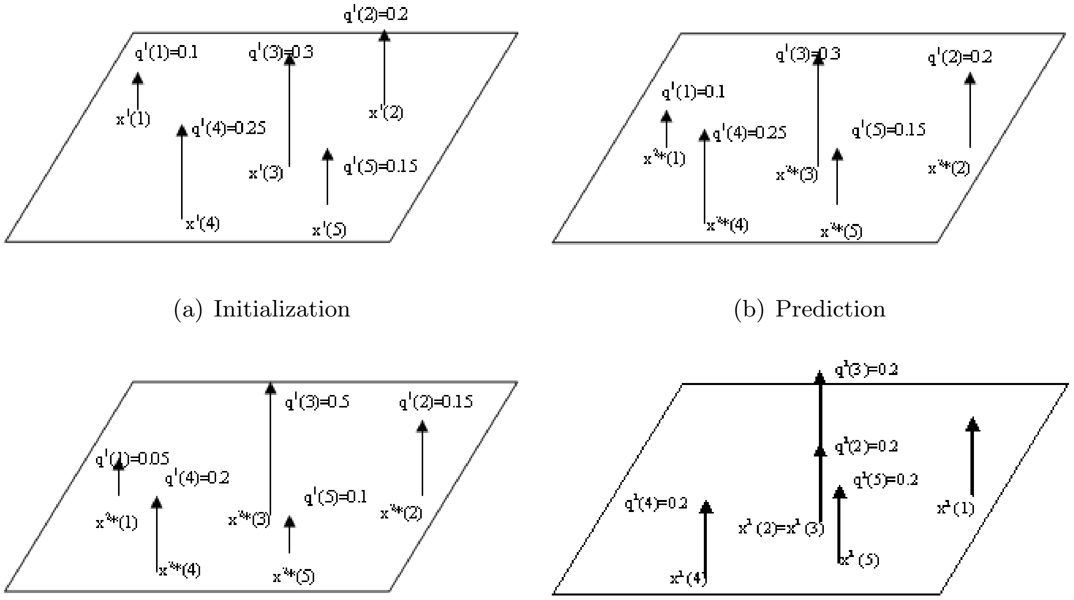
```
Given:
A set of particles ${\mathbf{x}^{k^*}(i), q^k(i)\}_{i=1}^{N_s}$

Objective:
Calculate the new particles $\{\mathbf{x}^k(i), \frac{1}{N_s}\}_{i=1}^{N_s}$

Algorithm:
(1)
Initialize the Cumulative Distribution Function(CDF) of $q^k(j)$: $c_1 = 0$
for $j = 2 : N_s$ {
    Construct the CDF: $c_j = c_{j-1} + q^k(j)$
}
(2)
Initialize another CDF of a uniformly distributed random variable: $t_i$
Draw a value: $t_1 = \mathbb{U}\left[0, \frac{1}{N_s}\right]$
for $i = 2 : N_s$ {
    $t_i = t_1 + \frac{1}{N_s}(i-1)$
}
(3)
Start at the bottom of the CDF of $q^k(j)$: $j = 1$
Start at the bottom of the CDF of $t$: $i = 1$
for $(i = 1 : N_s)$ {
    while $(t_i > c_j)$ {
        j = j + 1;
    }
    Assign sample: $\mathbf{x}^k(i) = \mathbf{x}^{k^*}(j)$
    Assign weight: $q^k(i) = \frac{1}{N_s}$
}
```

For better illustration, we also provide a toy example of the particle filter process in Figure 3.1. We illustrate this example in a two-dimensional space. In Figure 3.1(a), we are at time step $k = 1$, and we have a set of $q^1(i)$ and $\mathbf{x}^1(i)$, where $i = 1, 2, \ldots, 5$. Note that in Figure 3.1(a), the sequence order, $i = 1, 2, \ldots, 5$, for both $q^1(i)$ and $\mathbf{x}^1(i)$ is determined at the initialization of the particle filter, and will not be altered at any later time step. In Figure 3.1(b), we will calculate new samples and denote them as $\mathbf{x}^{2*}(i)$, $i = 1, 2, \ldots, 5$. Note that the weights $q^1(i)$ are unchanged from Figure 3.1(a) to Figure 3.1(b).

At the update stage, we will recalculate the new weights, $q^2(i)$. Note that from Figure 3.1(b) to Figure 3.1(c), the samples, $\mathbf{x}^{2*}(i)$, are unchanged. Finally, after the update stage, we will perform a resampling step in Figure 3.1(d). After resampling, all weights are set to equal to $\frac{1}{N_s}$, where $N_s = 5$ in this toy example, and $N_s = 500$ in our simulations in Chapter 6.



(a) Initialization

(b) Prediction

(c) Update

(d) Resampling

Figure 3.1: Illustration of the particle filter process. In (a), we are given a set of samples $\mathbf{x}^1(i)$ and weights $q^1(i)$. In this toy example, we set $N_s = 5$. From (a) to (b) is what we call the update stage. We can use (3.22) to calculate the new samples in (b). Note that from (a) to (b), the weights are unchanged. From (b) to (c), we calculate new weights using (3.23). Note that from (b) to (c), the samples are unchanged. Finally, to overcome the degeneracy problem, we perform a resampling to obtain the new set of weights $q^2(i)$ and samples $\mathbf{x}^2(i)$.

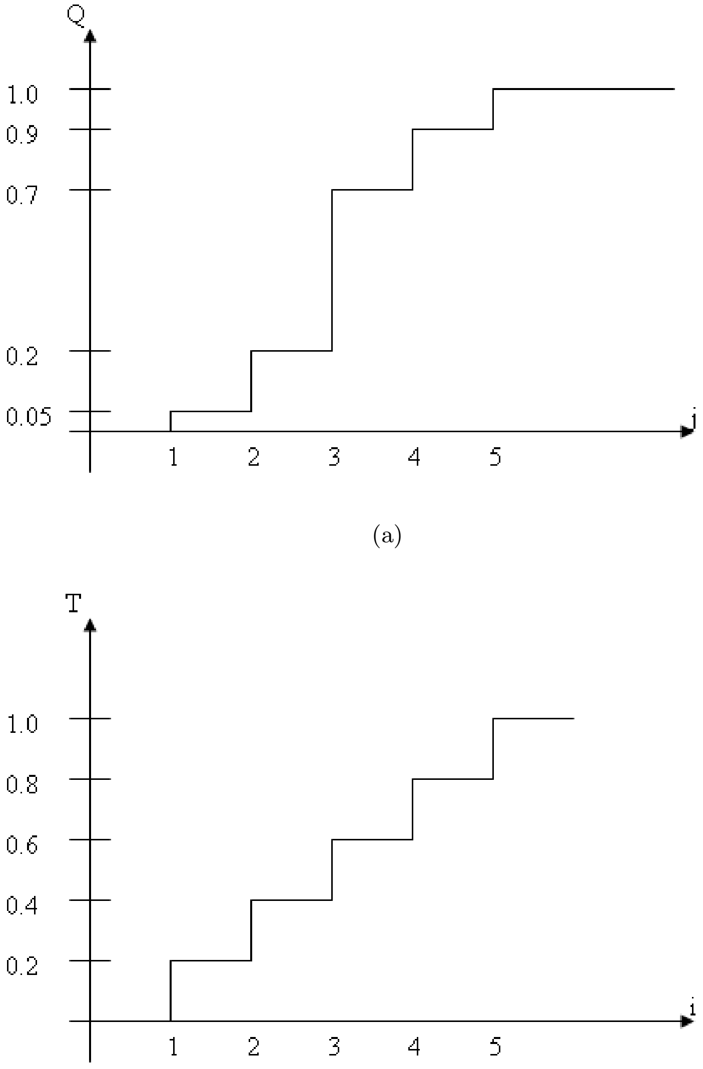
From Figure 3.1(c) to Figure 3.1(d), the resampling process is not shown. We present an illustration of the resampling process in Figure 3.2. We use the weights to build a cumulative distribution function, Q, in Figure 3.2(a). We can also build a cumulative distribution function for a uniform random variable, and we denote it as T in Figure 3.2(b). At the beginning, the index for Q, j, and the index for T, i, are both set to 1. We then increment i and j based on the resampling algorithm in Table 3.2 as follows:

1. $i = 1$, $j = 1$, $c_1 = 0.05 < t_1 = 0.2$
2. $i = 1$, $j = 2$, $c_2 = 0.2 \geq t_1 = 0.2$, $\mathbf{x}^2(1) = \mathbf{x}^{2*}(2)$
3. $i = 2$, $j = 2$, $c_2 = 0.2 < t_2 = 0.4$
4. $i = 2$, $j = 3$, $c_3 = 0.7 > t_2 = 0.4$, $\mathbf{x}^2(2) = \mathbf{x}^{2*}(3)$
5. $i = 3$, $j = 3$, $c_3 = 0.7 > t_3 = 0.6$, $\mathbf{x}^2(3) = \mathbf{x}^{2*}(3)$
6. $i = 4$, $j = 3$, $c_3 = 0.7 < t_4 = 0.8$
7. $i = 4$, $j = 4$, $c_4 = 0.9 > t_4 = 0.8$, $\mathbf{x}^2(4) = \mathbf{x}^{2*}(4)$
8. $i = 5$, $j = 4$, $c_4 = 0.9 < t_5 = 1.0$
9. $i = 5$, $j = 5$, $c_5 = 1.0 \geq t_5 = 1.0$, $\mathbf{x}^2(5) = \mathbf{x}^{2*}(5)$

After the above procedure, the resampling is complete. The new set of sample and weight pairs are shown in Figure 3.1(d). The basic idea of resampling can be seen in this toy example, where $\mathbf{x}^{2*}(1)$ is ignored because $q^1(1) = 0.05$ is too small. On the other hand, $\mathbf{x}^{2*}(3)$ is duplicated, because $q^1(3) = 0.5$ is larger.

The new set of samples, $\{\mathbf{x}^2(i)\}$, $i = 1, \ldots, 5$, will be passed to (3.22) to begin the next iteration of particle filters.

To illustrate the performance of the particle filter, we provide an example tracking problem where the measurement model is nonlinear in Appendix B.



(a)

(b)

Figure 3.2: Illustration of the resampling process. After we build two cumulative distribution functions, Q in (a) and T in (b), we begin with $i = 1$ and $j = 1$. As we increment j, we compare $c(j)$ and $t(i)$. If $c(j) \geq t(i)$, we will increment i until $t(i) \geq c(j)$ again. Every time we increment i, we will set $\mathbf{x}^2(i) = \mathbf{x}^{1*}(j)$ and $q^2(j) = \frac{1}{N}$. On the other hand, if $c(j) < t(i)$, we will do nothing except incrementing j until $c(j) \geq t(i)$ again.

3.1.3 Collaborative Tracking Using Sensor Networks

For sensor networks, Liu et al. [47] propose a special treatment for the tracking problem. At any time k, only one sensor node will be activated to perform the tracking task. This strategy is referred to as leader-based tracking. We will not reproduce all of the details in [47] here. Instead, we describe how the tracking algorithm is performed and how to activate the sensor node in the next time step.

Liu et al. [47] assume that sensor nodes are resource-limited, hence the tracking task itself is done by approximating the two equations in (3.4) and (3.5) numerically by nonparametric representations of $p(\mathbf{x}^k|z^k)$. This approach may reduce the computing cost on the sensor nodes, however, tracking accuracy is not guaranteed. Instead of restricting the capability of the nodes, in this dissertation we will instead use the particle filter tracking algorithm which will perform at least as well at the cost of computing resources.

As for the selection of sensor node, Liu et al. [47] propose to use *mutual information* to select the next sensor node to activate. Let U and V be two random variables having a joint pdf $p(u, v)$. The mutual information between U and V is defined as

$$I(U;V) \equiv E\left[log \frac{p(u,v)}{p(u)p(v)}\right] \tag{3.24}$$

At time step $k-1$, Liu et al. [47] propose to select the sensor node s that maximizes the mutual information between the next target location, $\mathbf{x}^k$, and the new measurement z^k

$$s = \underset{s \in \mathcal{S}}{argmax} \quad I_s(\mathbf{x}^k|z^{k-1}; z^k|z^{k-1}) \tag{3.25}$$

where $\mathcal{S}$ is the collection of sensors that the current node can talk to, namely the current node's neighborhood. Using the definition of mutual information in (3.24), we have

$$I_s(\mathbf{x}^k|z^{k-1}; z^k|z^{k-1}) = E\left[log \frac{p(\mathbf{x}^k, z^k|z^{k-1})}{p(\mathbf{x}^k|z^{k-1})p(z^k|z^{k-1})}\right] \tag{3.26}$$

In summary, the node activation algorithm is that at time step $k-1$, we select a collection of sensor nodes, $\mathcal{S}$, as the candidates to be activated in the next time step, k. For each node in $\mathcal{S}$, we will compute the mutual information according to (3.26). The node in $\mathcal{S}$ with the highest mutual information will be the next node to be activated at step k.

Let us look at how to compute (3.26) in detail. Recall that in the prediction stage of the particle filter algorithm, we pass the set of samples through the known motion model in (3.1). In a similar manner, at time step $k-1$, the term $p(\mathbf{x}^k|z^{k-1})$ can be "predicted" by passing the current $p(\mathbf{x}^{k-1}|z^{k-1})$ through the motion model in (3.1).

The predicted $p(\mathbf{x}^k|z^{k-1})$ can be represented by a discrete approximation, say, a set of particles. Recall from the previous section that a particle has two components: $(\mathbf{x}(i), q(i))$, $i = 1, \ldots, N_s$. The set of $\mathbf{x}(i)$, $i = 1, \ldots, N_s$ is the samples of the target state. We also choose a set of real values to quantify the possible range of the sensor measurement z^k. For example, we can use $[1, 2, \ldots, M_s]$ to quantify the range of the possible values of z^k. We denote this set as $\mathcal{Z}^k$, and $\mathcal{Z}^k \in \mathbb{R}$. Then for every element in the set of $\mathbf{x}(i)$, $i = 1, \ldots, N_s$ and for every element in the set $\mathcal{Z}^k$, we can calculate

$$p(\mathbf{x}^k, z^k|z^{k-1}) = p(z^k|\mathbf{x}^k)p(\mathbf{x}^k|z^{k-1}) \tag{3.27}$$

We already have the second term in (3.27), $p(\mathbf{x}^k|z^{k-1})$, by passing the current $p(\mathbf{x}^{k-1}|z^{k-1})$ through the motion model. The first term in (3.27), $p(z^k|\mathbf{x}^k)$, can be calculated using the known statistics of the measurement noise in (3.3). Note that if we have N_s samples $\mathbf{x}(i)$, and there are M_s values in $\mathcal{Z}^k$, then $p(\mathbf{x}^k, z^k|z^{k-1})$ in (3.27) is a $N_s \times M_s$ matrix.

After we have the joint density function $p(\mathbf{x}^k, z^k|z^{k-1})$, we can calculate the partial density function $p(z^k|z^{k-1})$ by

$$p(z^k|z^{k-1}) = \sum_{\mathbf{x}^k(i)} p(\mathbf{x}^k, z^k|z^{k-1}) \tag{3.28}$$

Note that $p(z^k|z^{k-1})$ is now a vector of length M_s.

Now that we have $p(\mathbf{x}^k|z^{k-1})$, $p(\mathbf{x}^k, z^k|z^{k-1})$ and $p(z^k|z^{k-1})$, we can calculate the $N_s \times M_s$ matrix

$$D = log\frac{p(\mathbf{x}^k, z^k|z^{k-1})}{p(\mathbf{x}^k|z^{k-1})p(z^k|z^{k-1})} \tag{3.29}$$

and the mutual information can be calculated using

$$I_s = \sum_{\mathbf{x}^k(i)} \sum_{\mathcal{Z}^k} D \cdot p(\mathbf{x}^k, z^k|z^{k-1}) \tag{3.30}$$

Note that the scalar value of I_s is calculated for every sensor node in the collection $\mathcal{S}$. The node with the maximal I_s will be activated in the next time step.

We follow this node activation scheme as part of our secure tracking algorithms. However, if we only activate only one sensor node at a time, the sensor network is susceptible to attacks. Once the active node is replaced with a malicious node, the tracking result will be inaccurate. Let us consider the following security problem in target tracking.

3.2 Security in Tracking

3.2.1 Problem Statement

Consider the case when we have deployed a sensor network, and all of the node positions have been determined and calibrated. At each time step, one or more sensor nodes will be activated, depending on battery resources and security needs. Following [47], we assume that sensor nodes have adequate processing power, and can calculate the current pdf $p(\mathbf{x}^k|z^k)$ after making measurements of their ranges to the current target location. The current pdf is reported to the central processor. The central processor will select the next sensor node(s) to activate, and send $p(\mathbf{x}^k|z^k)$ to the next node(s) in order to calculate $p(\mathbf{x}^{k+1}|z^{k+1})$. In the sensor network, an unknown number of the nodes are malicious and attempt to lower the accuracy of target tracking. The problem of *secure tracking*, as defined in this dissertation, is to correctly estimate the current target location by removing malicious nodes.

3.2.2 Problem Definition

Our objective is to detect the malicious nodes in the network during the tracking process. The basic assumptions given in Section 2.4 about the objective and behavior of the malicious nodes still hold. At any time step, all the malicious nodes will agree that the current target location is the fictitious event location in Section 2.4, and we refer to it as *fictitious target location.* In other words, at any time step, all the malicious nodes will report a pdf $p(\mathbf{x}^k|z^k)$ as if the target is at the fictitious target location. If we connect the fictitious locations over time, we form a *fictitious path.* We illustrate a fictitious path in

Figure 3.3. The fictitious path will mislead our defense mechanism and allow the enemy to avoid detection. The colluding behavior is also another feature that distinguishes malicious nodes from ordinary malfunction nodes.

In this section, localization is not performed at each time step, hence the discrepancy described in (2.13) no longer exists. Also, in tracking we do not activate as many nodes as in localization at each time step. In localization, there are multiple nodes active at one moment in time. In tracking, only one node is necessary since the motion model, the measurement model and past tracking history are known (we will activate more than one node in our secure tracking algorithm in Chapter 6).

What is notably different from Section 2.4 is that in Section 2.4, each sensor node only reports a range estimate δ_i. Here in target tracking, each sensor node can independently estimate the current target location using particle filters. We summarize what a particular sensor node does at each time step k as follows[2]:

1. The sensor node is given $p(\mathbf{x}^{k-1}|z^{k-1})$.
2. Next, the sensor node makes a measurement of the target using the measurement model given in (3.2). Hence z^k is obtained.
3. Using z^k and $p(\mathbf{x}^{k-1}|z^{k-1})$, calculate the current belief $p(\mathbf{x}^k|z^k)$

Based on the current belief $p(\mathbf{x}^k|z^k)$ from the sensor node, we estimate the current location of the target using the *posterior mean*

$$\hat{\mathbf{x}^k} = \int \mathbf{x}^k p(\mathbf{x}^k|z^k) d\mathbf{x}^k \tag{3.31}$$

or the *posterior max*

$$\hat{\mathbf{x}^k} = \underset{\mathbf{x}^k}{argmax} \quad p(\mathbf{x}^k|z^k) \tag{3.32}$$

From this point on, we will use the expected value as in (3.31) to illustrate the estimated current location of the target in the figures. However, between nodes and the central server, the entire density function $p(\mathbf{x}^k|z^k)$ is transmitted.

[2] Note that since we are describing the behavior of one node, we omit the subscript denoting which node this is

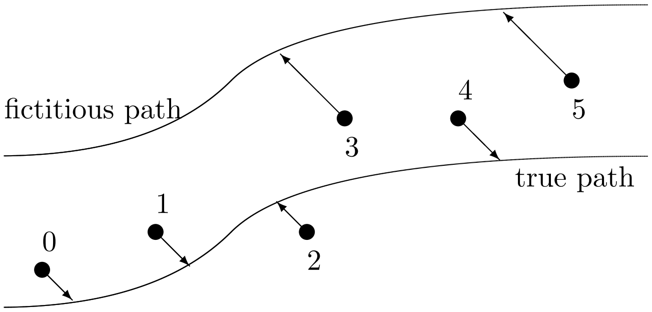


Figure 3.3: We have activated 6 sensor nodes consecutively, which are denoted as 0 through 5. The lower path is the true target path; while the upper one is fictitious. In this scenario, two malicious nodes report that the target is most likely on the upper path, i.e. Nodes 3 and 5 are malicious. Nodes 0, 1, 2 & 4 report correctly the range to the actual target position, lying on the true path.

3.2.3 Supplemental Properties

Discrete time steps

We assume that time is discrete in this dissertation. Also, the measurement process is fast enough that at the same time step, the target is (relatively) stationary, from the beginning to the end of the measurement process.

Sensor measurements

At each time step, we may activate more than one sensor node, if necessary. For those nodes active at the same time, we assume that their clocks are *synchronized*.

Fictitious target location

Similar to Section 2.4, the fictitious target location has to fall within the sensing range of all the active nodes. Otherwise, the benign nodes can simply tell which node is lying.

Centralized model

Although sensor nodes can perform tracking independently, the current pdf will be collected by the central processor. The central processor also determines which node(s) to activate in the next time step.

No communication delay

We do not consider any communication delay between the sensor nodes and the central processor. So even if the target is moving, we can still keep track of it using the tracking algorithm.

Chapter 4

Relaxation Labeling

The problems defined in Section 2.4 and Section 3.2 are critical to a secure sensor network environment. Before we could solve these two problems, however, we need to review the relaxation labeling algorithm which will be the foundation of our solution in the next chapters.

Relaxation labeling was proposed in the seminal paper of [51] in the area of *computer vision* - teaching computers to recognize the content of digital images. The original purpose of designing the relaxation labeling algorithm was to reduce the ambiguity in identifying the objects in a scene. Given the different relationships among the objects in the scene, the relaxation labeling algorithm will give a higher weight to a more likely solution and vice versa. The ambiguity is said to be removed when the objects in a scene can each be uniquely labeled with a consistent label.

Reducing ambiguity is also important in many other areas of science. For example, shape and stereo matching, image enhancement (edges, features) and high-level interpretation of image content [42] all can be posed as problems of reducing ambiguity. Over the years, relaxation labeling has become a prominent algorithm in solving them.

Let us begin with a toy example in image content interpretation. In Figure 4.1, we have an image which has three objects. Suppose for now that we have used some image segmentation algorithm to obtain this image. Based on the result, we found three objects inside this image. Assume that we somehow know that one of them is a circle, one of them is a triangle, and one of them is a square, except that we do not know which is which. How do we correctly label the right object inside the image to have the correct shape(label) ?

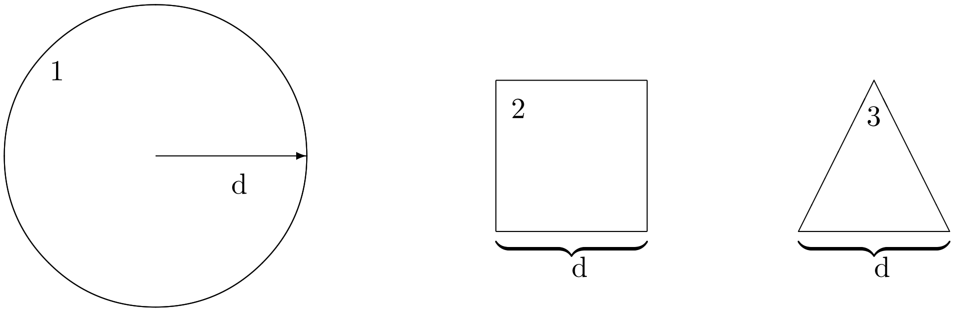


Figure 4.1: An scene labeling example to illustrate the relaxation labeling process. In the scene we have three objects: object 1 is a circle, object 2 is a square and object 3 is a triangle.

The solution lies in relaxation labeling. Introduced in 1976 [51], the formulation of "nonlinear relaxation" presented here has become the de facto standard method for solving consistent labeling problems. We review the method in this section to ensure the reader understands the method, before extending it to triples and applying the extension to sensor networks in Chapter 5. Both the extension and application are novel.

In relaxation labeling, three key components of the algorithm are defined [51]:

- $P_i(\lambda)$: the "confidence" for object i to have label λ
- $q_i(\lambda)$: the "support" for object i having label λ
- $r(i, j, \lambda, \lambda')$: the compatibility function between object i having label λ and object j having label λ'

Let us begin the explanation of relaxation labeling with $P_i(\lambda)$. $P_i(\lambda)$ is the final, desired result that we are seeking. The value of $P_i(\lambda)$ will eventually determine the labeling of each object. After we introduce $P_i(\lambda)$, we will introduce $q_i(\lambda)$ because $P_i(\lambda)$ depends on $q_i(\lambda)$. Finally, $r(i, j, \lambda, \lambda')$ will be introduced because $q_i(\lambda)$ depends on $r(i, j, \lambda, \lambda')$.

Each object i will be assigned a different label λ, and we assume that the types of labels are known. For example, in Figure 4.1, there are three labels: circle, triangle and square. The confidence that one can say object i has label λ is called $P_i(\lambda)$ [51]. $P_i(\lambda)$ is a number between 0 and 1, and the higher it is, the more confidence we have in saying that object i is correctly labeled as λ. Again using our example in Figure 4.1, we want $P_1(circle) \to 1$ over the iterations. Since we assume the labels are mutually exclusive and collectively exhaustive, $\sum_j P_i(\lambda_j) = 1$ (at any iteration) [51]. Hence $P_1(circle) + P_1(square) + P_1(triangle) = 1$ in Figure 4.1. In general, a relaxation labeling process works with a set of labels $\Lambda = \{\lambda_1, \lambda_2, \cdots, \lambda_d\}$ and $\sum_j P_i(\lambda_j) = 1$.

Now that we have defined $P_i(\lambda)$ and its properties, let us look at how to calculate it. Relaxation labeling is an iterative process. For iteration $t+1$, $P_i^{t+1}(\lambda)$ is updated as follows [51]

$$P_i^{t+1}(\lambda) = \frac{P_i^t(\lambda)\left[1 + q_i^t(\lambda)\right]}{\sum\limits_j P_i^t(\lambda_j)\left[1 + q_i^t(\lambda_j)\right]}, \tag{4.1}$$

In (4.1), the numerator means that P_i^{t+1}, the new confidence in the next iteration, will be equal to the current confidence, P_i^t, multiplied by $\left[1 + q_i^t(\lambda)\right]$. $q_i^t(\lambda)$ is the support of object i having label λ, as provided by other labelings, and $-1 \leq q_i^t(\lambda) \leq 1$ [51]. Hence if the support is larger, $\left[1 + q_i^t(\lambda)\right]$ should be higher, and as a result the confidence in the next iteration will be higher. If the support is negative, it will drive the confidence $p_i(\lambda)$ in the next iteration down. We can also see that in (4.1), the denominator acts as a normalization factor that scales $P_i^{t+1}(\lambda)$ to ensure $0 \leq P_i(\lambda) \leq 1$.

In (4.1), if $q_i^t(\lambda)$ is large (i.e., close to 1), it will eventually drive $P_i^{t+1}(\lambda)$ to be close to 1. On the other hand, if $q_i^t(\lambda)$ is negative, it will drive $P_i^{t+1}(\lambda)$ toward 0. Hence $q_i^t(\lambda)$ is essential to how we will end up getting $P_i(\lambda)$. Rosenfeld et al. [51] defines $q_i^t(\lambda)$ as follows

$$q_i^t(\lambda) = \sum_j C_{ij}\left[\sum_{\lambda'} r(i,j,\lambda,\lambda')P_j^t(\lambda')\right] \tag{4.2}$$

where C_{ij} is an optional coefficient. The purpose of C_{ij} is to be a normalization coefficient to make sure that $-1 \leq q_i^t(\lambda) \leq 1$. It may also be used to constrain the relationship between two objects. For example, if object i and object j have no relation at all, then $C_{ij} = 0$.

Equation (4.2) answers the question: given object i having label λ, how is that compatible with all the possible cases of the other objects having different labels? For example, in Figure 4.1, if we want to calculate $q_1(circle)$, we need to examine the compatibilities of

1. object 1 is *circle* and object 2 is *triangle*
2. object 1 is *circle* and object 2 is *square*
3. object 1 is *circle* and object 3 is *triangle*
4. object 1 is *circle* and object 3 is *square*

After we calculate the four compatibilities, we multiply them each with its respective confidence and sum them up. We observe that (4.2) is a sum of a function multiplied by a "probability" and some readers may find it helpful to think of it as an expected value.

The actual design of the compatibility function $r(i, j, \lambda, \lambda')$ is problem-dependent. It should be a function that returns a higher value when both labeling object i as λ and labeling object j as λ' make sense. Specifically, in the design of $r(i, j, \lambda, \lambda')$, the return value is required to satisfy

$$-1 \leq r(i, j, \lambda, \lambda') \leq 1 \tag{4.3}$$

Using our toy example in Figure 4.1, we assume that we somehow know that all the objects in the image are drawn with a constant size, i.e. the radius of the circle and the sides of the triangle and square are all d. Then using our knowledge of geometry, we know that the area of a circle of radius d is πd^2, which is larger than the area of a square of size d. The area of the square, d^2, is again larger than the area of a regular triangle of size d, which is $\frac{\sqrt{3}}{4}d^2$. Hence one way of designing a compatibility function for the problem in Figure 4.1 is to compare the area of object i and object j. If we label object i as circle and object j as square, then the area of object i should be larger than j. Similarly, if we label object i as square and object j as triangle, then the area of object i should also be larger than object j. If we find the area of object i to be much bigger than the area of object

j, then labeling i as *circle* and labeling j as *square* makes sense, so $r(i, j, circle, square)$ should return a positive value close to 1.

We give a simple example design of the compatibility function here. Suppose that we use a certain image processing algorithm to estimate the area of object i in the image, and we denote it as A_i. Similarly, the area of another object j is estimated to be A_j. Following our previous assumption that the objects should be of the same side length, we know that if we label object i as circle while labeling object j as square, then we are expecting $A_i > A_j$. We also know that given $A_i > A_j$, the number $\frac{A_i - A_j}{A_i}$ should be between 0 and 1 since both A_i and A_j are positive numbers. To meet the requirement of returning a value between -1 and 1 when we design a compatibility function, we can choose $r(i, j, circle, square)$ to be $2\frac{A_i - A_j}{A_i} - 1$ when we label i as circle and j as square. Similarly, when we expect $A_i > A_j$, we can also use this form. In the cases where $A_i < A_j$, we can use the form $2\frac{A_j - A_i}{A_j} - 1$. Then one possible design of the compatibility function can be

$$\begin{cases} r(i, j, circle, square) = r(i, j, circle, triangle) = r(i, j, square, triangle) = 2\frac{A_i - A_j}{A_i} - 1 \\ r(i, j, square, circle) = r(i, j, triangle, circle) = r(i, j, triangle, square) = 2\frac{A_j - A_i}{A_j} - 1 \end{cases} \tag{4.4}$$

It is important to understand the distinction between the terms “compatible” and “correct”. It is possible for a labeling of node i by λ to be completely compatible with labeling node j by λ' and both be wrong. We hope to always set up the iterative labeling problem so that the most globally compatible labeling will also be the correct one.

In Appendix C, we will explain why relaxation labeling is a “relaxation” process and derive how to minimize its objective function.

Chapter 5

A New Relaxation Labeling Architecture for Secure Localization

5.1 Related Work

The secure localization problem was first proposed by Lazos et al. [43] who formulated the problem in an a distributed system. Also, there is no direct measurement of range in [43]. Instead, the distance between sensor nodes are calculated by counting how many intermediate hops between nodes there are, the so-called “hop counts”. For example, if a message is sent by node 1, and it goes through node 2 to reach node 3, then the hop count is two. As another example, if a message is sent by node 1, and goes through nodes 2 through 4 to reach node 5, then the hop count is 4. Since our localization algorithms are centralized and range-based, the distributed methods are not furthered discussed.

A recent work was proposed by Liu et al. [46], which uses a thresholding technique to detect malicious nodes. If n sensor nodes report the same event, each with a different range estimate δ_i, $i = 1, \cdots, n$, we can calculate the discrepancy associated with each node as in (2.13). Liu et at. [46] set a threshold on the discrepancy, and set those nodes with discrepancies higher than the threshold as malicious nodes. The performance of [46] decreases as the number of malicious nodes increases in the network.

Statistical filtering methods based on authentication are proposed in [60, 65, 17] to detect false range reports made by malicious nodes. In [60], when an event occurs, those nodes which have detected the event collectively calculate the location of the event, and endorse it with a *Message Authentication Code* (MAC). As a report is forwarded through multiple hops toward the central processing unit, each forwarding node verifies the correctness of the MAC with certain probability. If the probability of some MAC being correct is too low, the report will be dropped. The probability of detecting incorrect MACs increases with the number of hops it travels. The success of the algorithm in [60] depends on the design of the MAC key for other nodes to authenticate. Our algorithm is not based on key authentication, and Ye et al. [60] also does not formulate the problem assuming the sensor nodes are colluding.

Although we assume that malicious nodes can successfully authenticate with the network, earlier works propose methods to securely distribute keys to nodes inside the network. For example, Eschenauer et al. propose a key management scheme for key distribution, revocation, re-eying and incremental additions of nodes [22].

Our work provides a different perspective to the secure localization problem [9, 11]. We propose a new relaxation labeling architecture to detect colluding malicious nodes first [9]. After removing malicious nodes, we can use only data from benign nodes to perform localization [11].

5.2 The New Relaxation Labeling Architecture

We assume that the sensor nodes are densely deployed, i.e., for every event there will be n nodes that have detected it, and $n > 3$. Besides, as illustrated in Figure 2.1, only three nodes are needed to localize the event. We will denote a set of 3 nodes as a *triple*. Since $n > 3$, we have $\left(\begin{array}{c} n \\ 3 \end{array} \right) = \frac{n!}{(n-3)!3!} = \frac{(n-2)\cdot(n-1)\cdot n}{1\cdot 2\cdot 3}$ triples. The estimated event locations from each triple may vary, since there may be 1, 2 or 3 malicious nodes in each set. If a triple has malicious nodes in it, the ranges reported by the 3 nodes will usually be inconsistent (unless all 3 are malicious), and such inconsistency will be shown in ϵ_i, $i = 1, 2, 3$.

We would like to exploit such inconsistency and use relaxation labeling algorithms to detect malicious nodes. In Chapter 6, we will activate more than one node at each time step, and the relationship between those active nodes at successive time steps can be used to examine inconsistency. More details on the tracking part will be described in Chapter 6.

In classical relaxation labeling algorithms [51], the compatibility function is defined over two objects. The idea of extending the compatibility function to three or more objects, the so-called “higher-order” compatibility functions, was first conceived in [31]. However, very little literature has really designed and implemented higher-order relaxation labeling. The reason is explained in [21], as Eklundh et al. experimented with both classical and higher-order relaxation labeling algorithms on edge enhancement applications. The conclusion in [21] is that the performance improvement gained from using higher-order compatibility functions is not very significant. Hence the performance gain does not justify the increased computing cost of higher-order compatibility functions.

There does exist evidence showing that higher-order relaxation labeling algorithms, when designed for certain applications, outperform classical ones. One example is graph matching [25, 41]. The compatibility function in [25] is specified in terms of the face-units of the graphs under match, and it requires only knowledge of the number of nodes, edges and faces in the model graph. The work in [25] shows that higher-order relaxation labeling algorithms, when designed well, can provide better performance. In this dissertation, however, it is *necessary* to use higher-order compatibility functions since localization of an event requires a triple of nodes.

We define *label* λ for each sensor node as assigning it to be malicious or benign. A “labeling” is the association of a node, i, with a label, λ, and this association is denoted by the ordered pair (i, λ). Specifically, if a node has label $\lambda = m$, it is considered to be malicious, while having label $\lambda = b$ denotes benign. For each sensor node, we define a *confidence* $P(\lambda)$. The confidence of node i having label λ is denoted as $P_i(\lambda)$. The confidence $P_i(\lambda_j)$ has probability-like properties:

$$0 \leq P_i(\lambda_j) \leq 1 \quad \forall i, j \qquad \sum_j P_i(\lambda_j) = 1. \tag{5.1}$$

Note that in (5.1), we only have two labels, hence $\lambda_j \in \{m, b\}$. Following [51], we iteratively update the confidence of node i having label j as

$$P_i^{t+1}(\lambda_j) = \frac{P_i^t(\lambda_j)\left[1 + q_i^t(\lambda_j)\right]}{D_i^t}, \tag{5.2}$$

where $D_i^t = \sum_k P_i^t(\lambda_k)\left[1 + q_i^t(\lambda_k)\right], \lambda_k \in \{m, b\}$ is a normalization required to ensure that $P_i(\lambda_j)$ sums to 1, and t stands for iteration.

Since we need at least 3 nodes to perform the localization task, we define $q_i^t(\lambda)$ to be a measure of how consistent the labeling of node i as having label λ is with the labeling of both the other nodes:

$$q_i^t(\lambda) = \frac{1}{N} \sum_j \sum_k \sum_{\lambda'} P_j(\lambda') \sum_{\lambda''} P_k(\lambda'') r(i, j, k, \lambda, \lambda', \lambda''), \tag{5.3}$$

where $N = (n-1)(n-2)$, n is the number of nodes in the network, $j = 1, ..., n$, $k = 1, ..., n$, $j \neq i$, $k \neq i$, $j \neq k$, and $r(i, j, k, \lambda, \lambda', \lambda'')$ is a "compatibility function" to be defined and explained later. Note that the summation are over all the nodes, not just a single triple. The power of $q_i^t(\lambda)$ is that it considers the consistency of node i having label λ with all other *pairs* of nodes j, k. If (i, λ) is not consistent with other labelings, $q_i^k(\lambda)$ will be negative, hence driving $P_i(\lambda)$ down.

From (5.1), it is evident that we only need to calculate one probability ($P_i(b)$ or $P_i(m)$) for each node since the two probabilities sum up to 1. That probability is further dependent on $q_i^t(\lambda_j)$, as shown in (5.2). The computational complexity of $q_i^t(\lambda_j)$, as shown in (5.3), is $O(n^2)$, where n is the number of nodes in the network.

5.2.1 Design of the Compatibility Function

To implement (5.3), we need to design a compatibility function $r(i, j, k, \lambda, \lambda', \lambda'')$ which will be higher when nodes i, j, k having labels $\lambda, \lambda', \lambda''$, respectively, is consistent, and vice versa. Specifically, $-1 \leq r(\cdot) \leq 1$. For example, if we have 10 nodes, and we are examining nodes 3, 5 and 7. If node 3 is malicious, and nodes 5 and 7 are benign, then $r(3, 5, 7, m, b, b)$ should be close to 1.

For each triple containing nodes (i, j, k), we may calculate the respective discrepancies ϵ_i, ϵ_j, and ϵ_k by equation (2.13). Define $\epsilon = \epsilon_i + \epsilon_j + \epsilon_k$ as the total discrepancy. Each node of the triple could be either benign or malicious, so we ought to have $2^3 = 8$ different compatibility functions for each triple

1. $r(i, j, k, b, b, b)$: node i benign, node j benign, node k benign
2. $r(i, j, k, b, b, m)$: node i benign, node j benign, node k malicious
3. $r(i, j, k, b, m, b)$: node i benign, node j malicious, node k benign
4. $r(i, j, k, b, m, m)$: node i benign, node j malicious, node k malicious
5. $r(i, j, k, m, b, b)$: node i malicious, node j benign, node k benign
6. $r(i, j, k, m, b, m)$: node i malicious, node j benign, node k malicious
7. $r(i, j, k, m, m, b)$: node i malicious, node j malicious, node k benign
8. $r(i, j, k, m, m, m)$: node i malicious, node j malicious, node k malicious

Due to the fact that the malicious nodes are colluding, we only need to design two types of compatibility functions. Case 1 and Case 8 use one type of the compatibility function. Case 2, Case 3, Case 4, Case 5 and Case 6 use the other type of the compatibility function. Let us look at them in detail.

Case 1 and Case 8

We assume that the malicious nodes are colluding, i.e. all the malicious nodes will report that a fictitious event occurred at a particular location (which is, of course, different from the true event location). In this regard, the case where all three nodes are malicious is as compatible as the case where all three nodes are benign. Let us consider the 3-node benign case. Since all of the 3 nodes are benign, the total discrepancy should be small. So what we need is a function with the following properties:

- Given a small ϵ, the compatibility function returns a value close to 1
- Given a large ϵ, the compatibility function returns a value close to -1

- Given any intermediate value of ϵ, the compatibility function returns a value between -1 and 1
- The compatibility function is monotonic and (preferably) smooth

Hence for the 3-node benign case, we can define

$$r(i, j, k, b, b, b) = 1 - \frac{2}{1 + e^{-\alpha_1(\epsilon - T_1)}} \tag{5.4}$$

where T_1 is a threshold on the error term ϵ. In equation (5.4), ϵ is the total discrepancy of the triple. The larger ϵ is, the less likely that this triple has all benign nodes, because a large ϵ indicates that the nodes in the triple do not agree with each other. In equation (5.4), the function of the form $1/(1+e^{-\alpha(x-\beta)})$ is generally referred to as a *sigmoid function* [57]. The advantage of using a sigmoid function is that the compatibility function will be a smooth curve between -1 and 1. We show some example parameters of equation (5.4) in Figure 5.1.

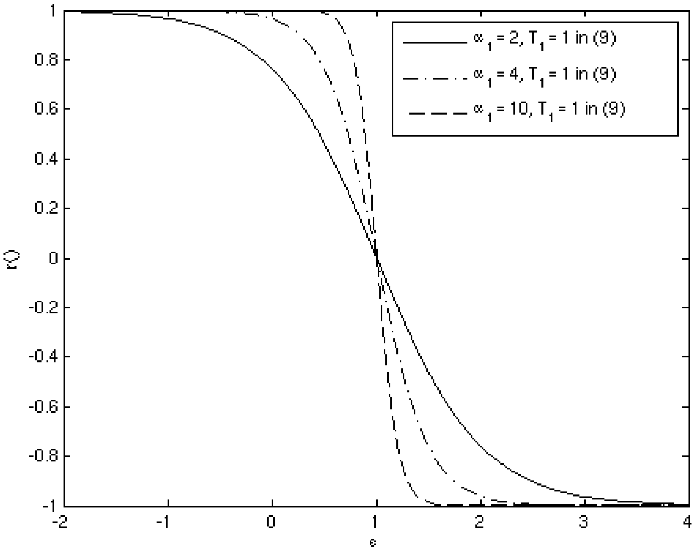


Figure 5.1: Some example parameter settings of equation (5.4)

The compatibility function for the 3-node malicious case is identically

$$r(i,j,k,m,m,m) = 1 - \frac{2}{1+e^{-\alpha_1(\epsilon - T_1)}} \tag{5.5}$$

since the malicious nodes are assumed to be colluding. Values for α_1 and T_1 and sensitivity to their choices will be discussed in Section 5.4.

Case 2 to Case 6

If there is only one malicious node in the triple, the other two nodes must be benign. This one-node malicious case is exactly the same as the one-node benign case, because in the one-node benign case, the other two malicious nodes are colluding.

Let us look at the one-node malicious case first. For one-node malicious case, the single malicious node in the triple reports an incorrect event location. Hence the localization result will be shifted due to the "contamination" from the malicious node. As a result, inconsistency among the three nodes exists. Since the localization result is incorrect, every node in this triple will have a discrepancy, ϵ_l, $l \in \{a,b,c\} \in \{1,2,\cdots,n\}$. Since the malicious node is the minority among the three, it will tend to contribute a higher error. We can design our compatibility function based on this concept:

Given a particular triple, (i,j,k), and considering the labelings (i,m), (j,b) and (k,b), we then introduce a new variable x which measures the relative contribution of node i to the total discrepancy

$$x = \frac{\epsilon_i}{\epsilon} = \frac{\epsilon_i}{\epsilon_i + \epsilon_j + \epsilon_k} \tag{5.6}$$

We also need to check if the total discrepancy, ϵ, of this triple is small. In (5.5), T_1 was used to indicate a significant inconsistency, and we use it again here for the same purpose. If $\epsilon < T_1$, then this triple should be either all-benign or all-malicious. Therefore, we want a compatibility function which has the following properties

- First check if ϵ is small. If yes, the compatibility function should return a negative value[1]. Otherwise, check x as follows:

[1] If ϵ is quite small or quite large, we are confident that our labeling is consistent, and r is allowed to go to ±1. However, in the 1-node different case, we have less confidence overall. Fortunately, relaxation labeling provides a mechanism for expressing this lack of confidence; we simply bound r to be in the range $-0.5 < r < 0.5$.

- Given a small x, the compatibility function should return a negative value
- Given a large x, the compatibility function should return a positive value
- Given any intermediate value of x, the compatibility function is a smooth curve

The compatibility function for one-node malicious (or one-node benign) case is then defined as

$$r(i, j, k, m, b, b) = -0.5 \qquad if \qquad \epsilon \leq T_1 \tag{5.7}$$

Otherwise, if $\epsilon > T_1$, we use the following form for the compatibility function for one-node malicious (one-node benign) case

$$r(i, j, k, m, b, b) = \frac{1}{1 + e^{-\alpha_2(x - T_2)}} - 0.5 \tag{5.8}$$

where $x = \epsilon_i/\epsilon$, the ratio of the discrepancy by the malicious node i divided by the total discrepancy, ϵ. In equation (5.8), the larger x is, the larger r is. That is to say, if we find that for a particular triple (i, j, k), the discrepancy coming from node i is a large component of the total discrepancy ϵ, then we have a high confidence to claim that node i is malicious and nodes j and k are benign. Some example parameter choices of (5.8) are illustrated in Figure 5.2.

The other one-node benign/malicious compatibility functions are defined in the same manner as (5.8). We summarize all 8 compatibility functions in Table 5.1. Note that in Table 5.1, $x_\zeta = \epsilon_\zeta/\epsilon$, where $\zeta = i, j, k$.

We have observed that the concept behind the compatibility function in (5.4) almost always matches with real data (see Section 5.4). In other words, what we have found from the real data is that for all benign nodes, the total discrepancy, ϵ, is always small. However, the design concept of the compatibility function in (5.8) is less robust on real data. That is to say, some small portion of the triples having one malicious node and two benign nodes, will violate the rule of $x = \frac{\epsilon_m}{\epsilon} \leq \frac{1}{3}$, where ϵ_m denotes the discrepancy from the malicious node. For this reason, we scale the compatibility functions to be between -0.5 and 0.5 in (5.8). We have discovered that, empirically, this makes the convergence of the confidences more stable.

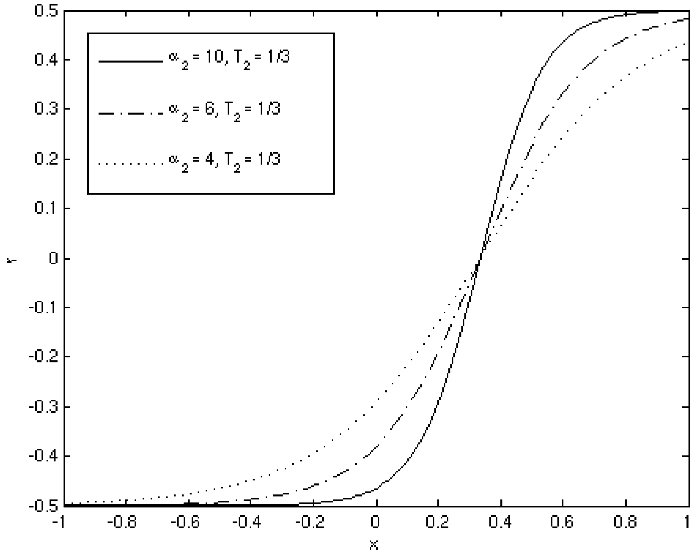


Figure 5.2: Some example parameter settings of equation (5.8)

Table 5.1: Compatibility functions

$r(i,j,k,m,m,m)$	$1-2/(1+e^{-\alpha_1(\epsilon-T_1)})$	

$r(i,j,k,m,m,b)$	-0.5
	$1/(1+e^{-\alpha_2(x_k-T_2)})-0.5$
		if $\epsilon \leq T_1$
		if $\epsilon > T_1$

$r(i,j,k,m,b,m)$	-0.5
	$1/(1+e^{-\alpha_2(x_j-T_2)})-0.5$
		if $\epsilon \leq T_1$
		if $\epsilon > T_1$

$r(i,j,k,m,b,b)$	-0.5
	$1/(1+e^{-\alpha_2(x_i-T_2)})-0.5$
		if $\epsilon \leq T_1$
		if $\epsilon > T_1$

$r(i,j,k,b,m,m)$	-0.5
	$1/(1+e^{-\alpha_2(x_i-T_2)})-0.5$
		if $\epsilon \leq T_1$
		if $\epsilon > T_1$

$r(i,j,k,b,m,b)$	-0.5
	$1/(1+e^{-\alpha_2(x_j-T_2)})-0.5$
		if $\epsilon \leq T_1$
		if $\epsilon > T_1$

$r(i,j,k,b,b,m)$	-0.5
	$1/(1+e^{-\alpha_2(x_k-T_2)})-0.5$
		if $\epsilon \leq T_1$
		if $\epsilon > T_1$

$r(i,j,k,b,b,b)$	$1-2/(1+e^{-\alpha_1(\epsilon-T_1)})$	

We summarize the algorithm of relaxation labeling in Table 5.2. The pseudo code for calculation of $q_i(\lambda_j)$ in equation (5.3) is listed in Table 5.3. In our simulations in Section 5.3.1, we specifically removed the exceptional cases where some sensor nodes are equidistant to both true event and fictitious event locations. For example, we show one sensor network consisting of four nodes in Figure 5.3. In that figure, we have two malicious nodes and two benign nodes. The two benign nodes are equidistant to both the true event and the fictitious event. Hence they appear to be consistent, but the result is incorrect (all agreeing on the fictitious event in Figure 5.3). For the purpose of simulations, we have excluded such ill-conditioned cases where sensor nodes are equidistant to both the true and fictitious events.

Table 5.2: Algorithm for relaxation labeling

```
Use range estimates $\delta_i$ to calculate $\epsilon_i$, $\epsilon_j$, $\epsilon_k$ for all triples;
Initialize $P_i(m) \simeq P_i(b) \simeq 0.5$;
Set $\kappa$ to be a small positive number;
while (for all nodes $\kappa < P_i(m) < 1 - \kappa$) {
for each node {
  Calculate $q_i(m)$ and $q_i(b)$ using Table 5.3;
  Calculate $D_i(m) = P_i(m)(1 + q_i(m)) + P_i(b)(1 + q_i(b))$;
  Calculate $P_i(m) = P_i(m) \times (1 + q_i(m))/D_i$;
  Calculate $P_i(b) = 1 - P_i(m)$;
} }
```

Table 5.3: Subroutine for calculation of $q_i(\lambda_j)$

```
Given node number $i$ and $\lambda_1$;
$total = 0.0$;
$n$ is the total number of nodes;
for (k from 0 to *number of nodes*) {
  if ( k $\neq$ i) {
    for (l from 0 to *number of nodes*) {
      if ( (i $\neq$ l and k $\neq$ l)) {
        for ($\lambda_1$ from $m$ to $b$) {
          $tot = 0.0$;
          for ($\lambda_2$ from $m$ to $b$) {
            Calculate r(i, k, l, $\lambda_1$, $\lambda_2$, $\lambda_3$) using Table 5.1;
            $tot$ += r(i, k, l, $\lambda_1$, $\lambda_2$, $\lambda_3$) $\times$ $P_l(\lambda_2)$;
          }
        $total$ += $tot$ $\times$ $P_k(\lambda_1)$;
} } } } }
return $total/(n-1)/(n-2)$;
```

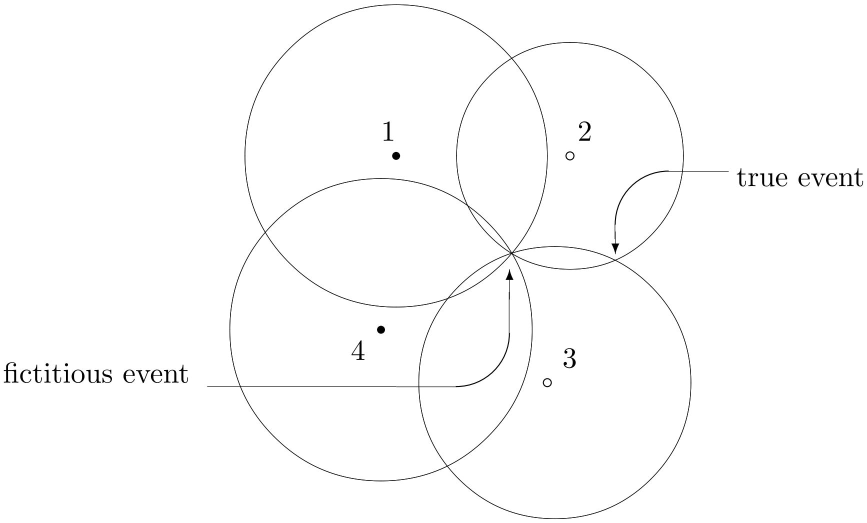


Figure 5.3: A sensor network consisting of four nodes. Node 1 and node 4 are malicious; while node 2 and node 3 are benign. In this example, node 2 and node 3 are equidistant to both the true event and the fictitious event. Hence they appear to be consistently reporting on the fictitious event. Ill-conditioned cases like this have been excluded in our simulations.

5.3 Experimental Results of Detecting Malicious Nodes Using Relaxation Labeling

5.3.1 Simulation

We randomly generate the positions of n nodes in a test field of $[0, 10] \times [0, 10]$. Among the n nodes, n_m are malicious. We simulate the sensor measurements z_i using the sensor model in (2.2) by setting $\alpha = 4$

$$z_i = \frac{a}{d_i^2} + w_i, \tag{5.9}$$

where a is the amplitude of the event, d_i is the actual distance from sensor i to the event, and w is additive Gaussian noise. Without loss of generality, we set $a = 1.0$, and equation (5.9) becomes

$$z_i = \frac{1}{d_i^2} + w_i \tag{5.10}$$

For benign nodes, d_i is the distance from node i to (x_b, y_b), the location of the true event. Similarly, for malicious nodes, d_i is the distance from node i to (x_m, y_m), the fictitious event location that the malicious nodes report. To get an estimate of how large we should choose the noise variance in our simulations, consider the position of a sensor node, (x_i, y_i). x_i and y_i are both uniform random variables, $x_i \sim \mathbb{U}[0, 10]$ and $y_i \sim \mathbb{U}[0, 10]$. Consider that the event is positioned at $(0, 0)$, hence $d_i^2 = \left(x_i^2 + y_i^2\right)$. The expected value of x_i^2 is $\frac{100}{3}$, so the expected value of d_i^2 is roughly $\frac{200}{3}$. Hence a typical value of $\frac{1}{d_i^2}$ is $\frac{3}{200} = 0.015$. So noise variance $\sigma^2 = 0.01$ is large as compared to $\frac{1}{d_i^2}$.

To obtain range estimates from each sensor node, we use (2.4) to calculate

$$\delta_i = \sqrt{\frac{1}{z_i}} = \sqrt{\frac{1}{\frac{1}{d_i^2} + w_i}} = \sqrt{\frac{d_i^2}{1 + d_i^2 w_i}} \tag{5.11}$$

In our first experiment, we set $(x_b, y_b) = (3.0, 3.0)$ and $(x_m, y_m) = (7.0, 7.0)$. All of the probabilities are initialized at $P_i(\lambda_j) \simeq 0.5$. The number of nodes, n, is 7, and the first two nodes are always chosen to be malicious, i.e. $n_m = 2$. The parameters are determined experimentally (see Section 5.4) as $\alpha_1 = 4.0$, $T_1 = 0.1$, $\alpha_2 = 2.0$, $T_2 = 1/3$. Note that α_1 and α_2 merely control the slope of the sigmoid function in Equation (5.4) and (5.8), and small changes of α_1 and α_2 generally do not have any influence on the simulation outcome.

We choose α_1 to be larger than α_2 because for a triple to be consisting of all benign or all malicious nodes is a more strict condition than one node benign or one node malicious[2]. T_2 is also almost always set to 1/3 since we assume that each node in a triple contributes equally. Hence, the two parameters that impact the performance are T_1 and the noise variance σ^2, where T_1 can be controlled by the user but σ^2 is determined by the system environment.

To verify the correctness of our relaxation labeling algorithm, we set $\sigma^2 = 1.0 \times 10^{-6}$, essentially noise free. The relaxation labeling process took 22 iterations to converge, and the time it took was less than 1 second on a Dual 1.8GHz PowerPC G5 computer. We plot an instance of the (random) locations of the 7 nodes in Figure 5.4. Also, based on δ reported from each sensor, we plot a circle for each node in Figure 5.4. In Figure 5.5, we show the convergence of $P_i(b)$, $i = 1, ..., 7$. Since $P_0(b)$ and $P_1(b)$ converged to 0, node 0 and node 1 are found to be malicious.

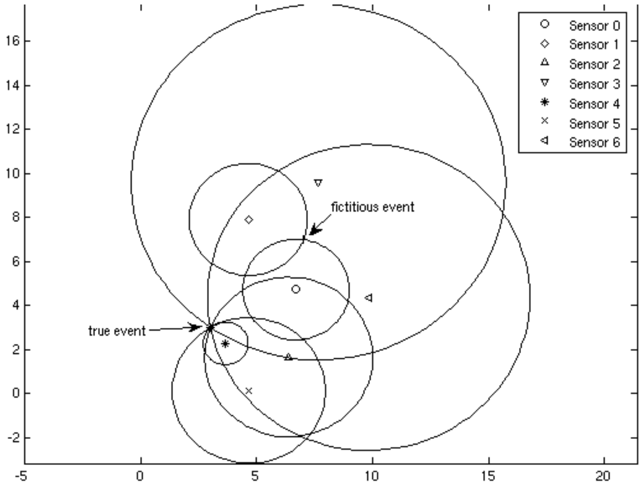


Figure 5.4: Simulation Example of 7 nodes. Two of the nodes are malicious.

[2] Recall that for a triple to be all benign or all malicious, its ϵ has to be small. Choosing α_1 larger will make the sigmoid function more like a threshold function

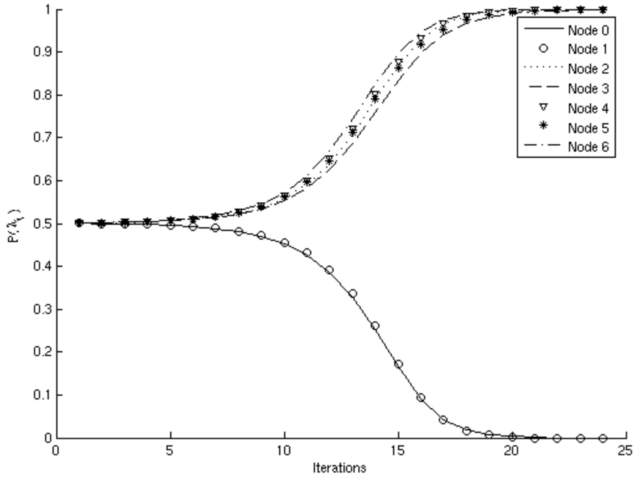


Figure 5.5: Convergence of $P(b)$

Next, to examine if the relaxation labeling process is correct, we repeat the same experiment using $\sigma^2 = 1.0 \times 10^{-6}$ and $T_1 = 0.01$ for 10,000 times. In all of the $10,000$ experiments, the locations of the nodes are different. The result from the $10,000$ experiments are all correct.

Next, we look at the impact of parameters on the system performance. At a particular T_1 and noise variance σ^2, we repeat the relaxation labeling experiment (2 malicious nodes out of 7 nodes) for 100 times, each time the nodes are positioned randomly. For any experiment out of 100 times, we will consider the entire experiment a failure if ANY of the 7 nodes is misclassified. Note that this is a very strict failure definition. We then perform another 100-times experiment at different T_1 and noise variance σ^2 values. The failure rate (out of 100 experiment) is shown in Figure 5.6 as a 3D graph. Note that in Figure 5.6, the y-axis is log-scale (base 10), hence noise variance σ^2 ranges from 1.0 to 1.0×10^{-6}. We can see that at $1.0 \times 10^{-5} \leq \sigma^2 \leq 1.0 \times 10^{-6}$ and $0.1 \leq T_1 \leq 0.75$, the error rates are almost 0. The error percentage goes up as T_1 goes up. As σ^2 goes up, the failure rate goes up considerably. These two phenomena are expected because as σ^2 increases, more and more of the localizations done by each triple would be inaccurate. Hence the overall system error rate would go up. There should a suitable range for T_1 because, as in (5.4), it controls threshold for determining a triple to be all-benign (all-malicious) or not. If T_1 is set too high, then no matter how a triple behaves, it will always be regarded as an all-benign (all-malicious) triple. This will create error for the overall system performance. Hence if T_1 is set too high and as it increases, the error rate goes up. The number of iterations required to reach convergence corresponding to the T_1 and σ^2 values in Figure 5.6 are illustrated in Figure 5.7. We can see that as σ^2 increases, the number of iterations required to converge also increases. This is because that higher σ^2 leads to more inaccurate localizations, and there are more inconsistencies in the system. Increasing T_1 also makes the required number of iterations to go up a bit, although the trend is less obvious. If we increase T_1 too large, there will be more triples incorrectly regarded as one-benign (one-malicious). Such inconsistencies lead to the slight increase in the number of iterations in Figure 5.7. In Figure 5.6, the average time to perform any 100-times experiment is 3.489 seconds on a Dual 1.8GHz PowerPC G5 computer.

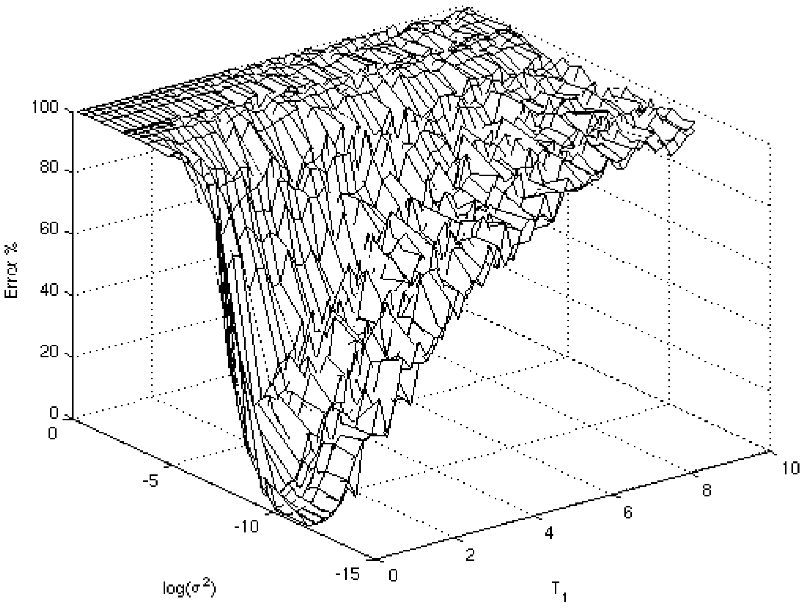


Figure 5.6: The effect of T_1 and σ^2 on the system performance

Finally, we fix the number of malicious nodes at 2, and increase the number of total nodes in the network. We also fix $T_1 = 0.5$ and repeat the experiments for 100 times, each time with different random node locations. The failure rate (out of 100 times) is shown in Figure 5.8 at three different noise levels. In Figure 5.8, we can see that the failure rate slightly goes down as the number of nodes gradually increases. This agrees with intuition

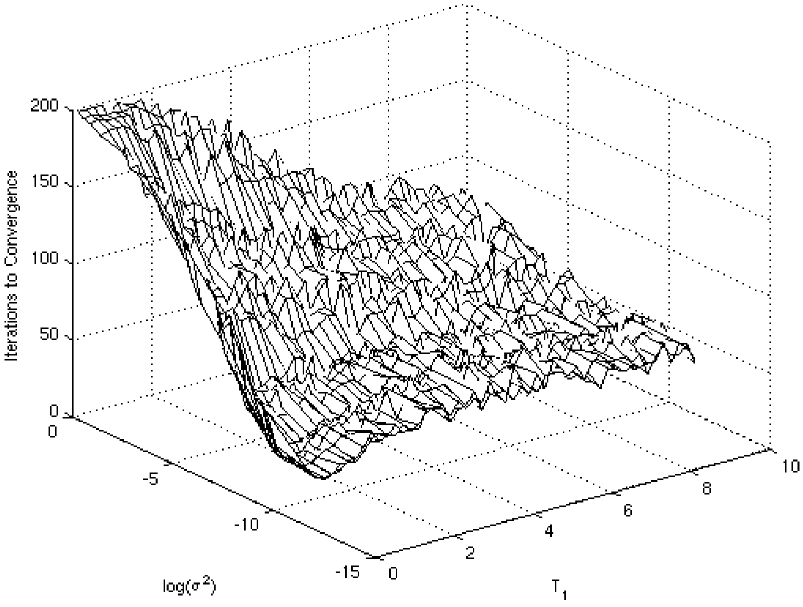


Figure 5.7: The number of iterations required to reach convergence corresponding to the T_1 and σ^2 values in Figure 5.6

because we are detecting the malicious nodes. As the number of total nodes increases, there are more consistency in the system because the number of benign nodes go up. Note that in Figure 5.8 , under the assumption of dense deployment, even if we misclassify a small number of benign nodes, we can still use the remaining benign nodes to correctly localize events.

5.3.2 Field Experiment

In this section, we use an existing set of field experiment data to test our relaxation labeling algorithm [46]. The field experiment was conducted in an outdoor environment using MICA2 motes [29] running TinyOS [28]. 8 nodes are deployed in a 40 × 40 feet field, as illustrated in Figure 5.9. The node positioned at $(5, 5)$ of Figure 5.9 does not know its position, hence the original purpose of the experiment [46] is to use the other nodes (node 0, node 1, $\cdots$, node 6) to localize its position.

The node positioned at $(5, 5)$ is denoted as the "event" node. The reader might find it helpful to think of it as the event. The rest of the nodes are sensor nodes (since they know their respective positions), and they localize the event node based on the received

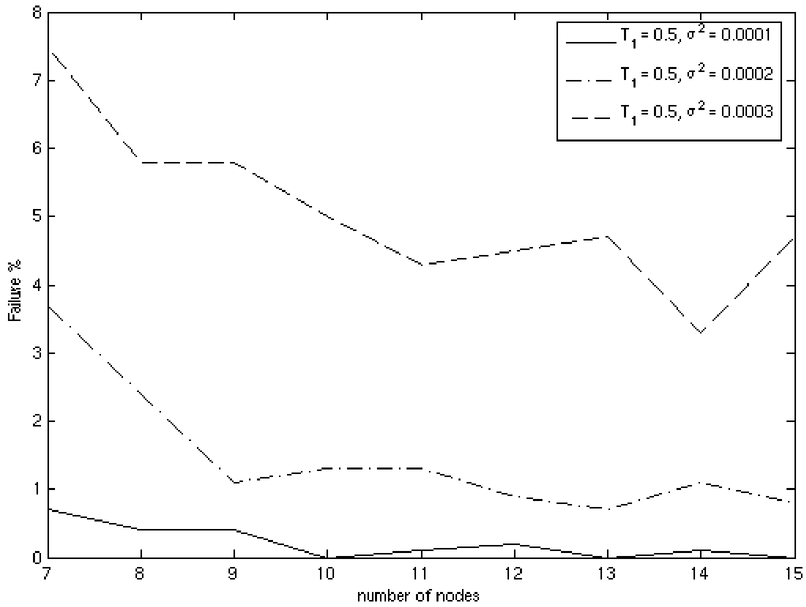


Figure 5.8: Failure rate for a network of various number of nodes. At each network size, only 2 nodes are malicious.

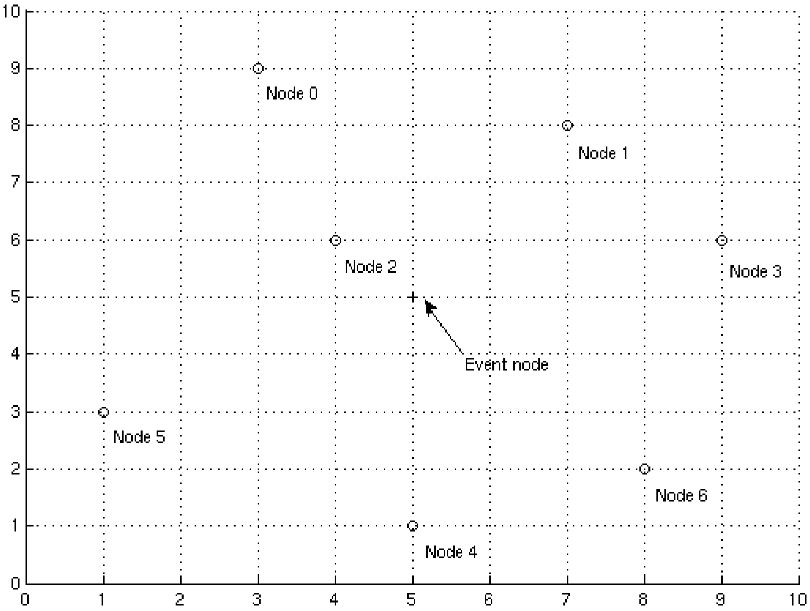


Figure 5.9: Deployment of sensor nodes in the field (Unit: 4 feet). The sensor nodes are denoted as circles, while the event node, positioned at (5,5), is denoted as a cross.

signal from it. This experiment is isomorphic to our problem formulation in Chapter 3 since the event node plays the role of the event.

At each sensor node location, we plot circles using the range estimates δ, and the result is illustrated in Figure 5.10. We observe from Figure 5.10 that due to noise, some nodes report longer δ; while some report shorter δ. However, the intersections of most of the circles fall on or near the event node location, $(5, 5)$, in Figure 5.10.

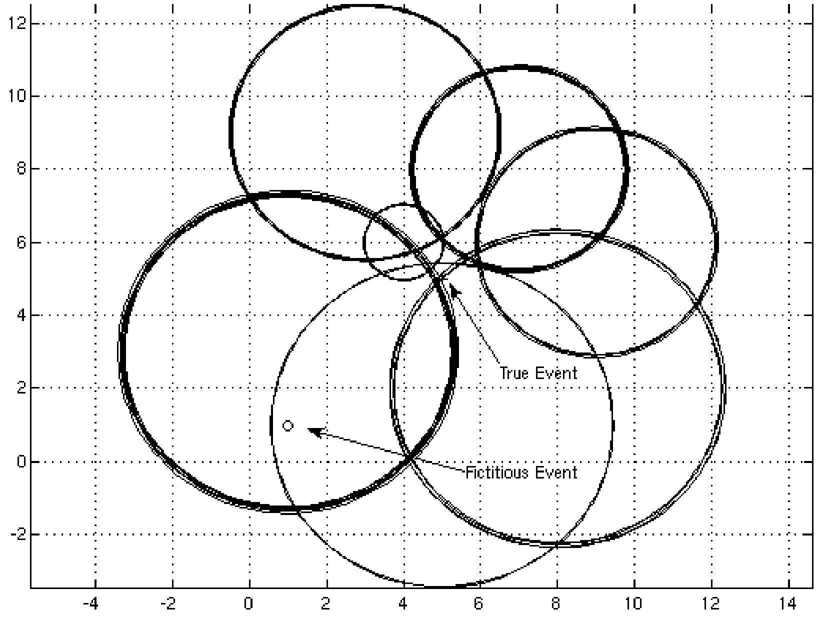


Figure 5.10: For each measurement at sensor nodes, we plot a circle using the range estimates. Note that the event node is at the center of the field, $(5, 5)$. The intersections of most of the circles are either on or near the event node due to noise.

To create colluding attacks, we chose node 0 and node 1 to be malicious and replace their range estimates with the distance from each node to $(1, 1)$, plus some noise disturbances. The noise variance chosen for the malicious node is small, $\sigma^2 = 0.000001$, since these nodes are colluding. That is to say, node 0 and node 1 report that the event occurs at $(1, 1)$. The range estimates for nodes 2, ..., 6 are chosen from the field measurements we have. In future work, malicious nodes could be modeled as having randomness in their behavior.

Using our relaxation labeling algorithm, we set the threshold in Equation (5.4) to 1.5. This threshold is higher than the T_1 in Section 5.3.1 because the range of the sensor field is 40 by 40 feet, larger than our simulations. Recall that T_1 is the threshold on the total discrepancy. As the scale of the system in this field experiment is larger, we are expecting a larger total discrepancy. Hence T_1 is larger. All the other parameters are unmodified. The probabilities converged at iteration = 104. We show the convergence of the probabilities in Figure 5.11, and we successfully identify node 0 and node 1 to be malicious and nodes 2, ..., 6 to be benign.

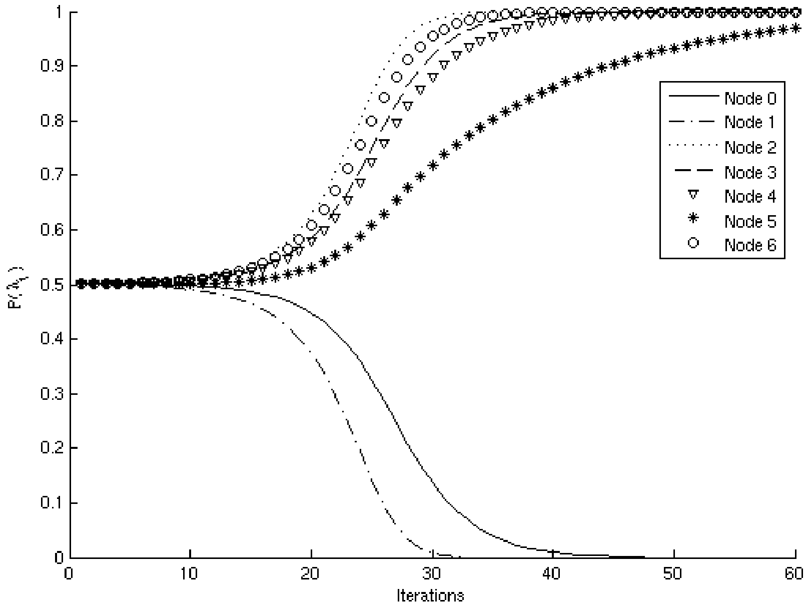
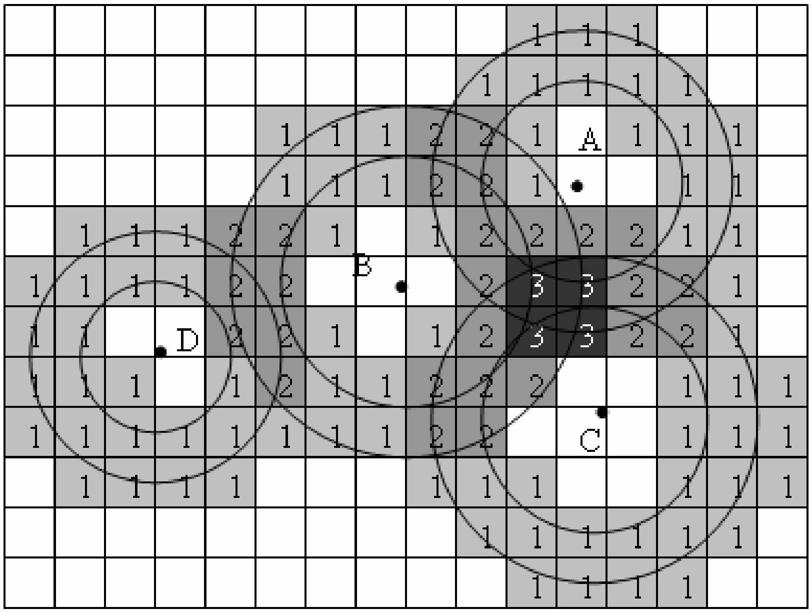


Figure 5.11: The probability of each sensor node being benign, $P(b)$. $P_0(b)$ and $P_1(b)$ go down to 0, which means that these two nodes cannot be benign.

In the field measurement data that we obtained, each node has several range estimates of the event. We randomly pick two range estimates from every node, so we have $2^7 = 128$ different experiments. Again, we pick the first 2 nodes to be malicious, and we replace their range estimates δ with the distance to $(1, 1)$. We successfully identified node 0 and node 1 to be malicious nodes in all 128 experiments.

5.3.3 Comparison with an Existing Algorithm

Liu et al. propose a voting-based secure localization algorithm [46]. We illustrate the voting algorithm in Figure 5.12. The sensor field is divided into different non-overlapping cells, and each cell has a counter. At the beginning, all counters are reset to zero. At each node location, we plot a circle using the range estimate, δ_i, as the radius. Those cells on the circle are incremented by one. Note that Liu et al. also consider a measurement error term, φ, such that the circles are actually rings [46]. After all circles have been drawn, we pick the cell having the highest count as the event location. The choice of cell sizes and other statistical measures to make the voting algorithm more robust are also discussed in [46]. Note that this is also a Hough-transform [19, 30, 3] like approach, where consistent solutions get high increment values.

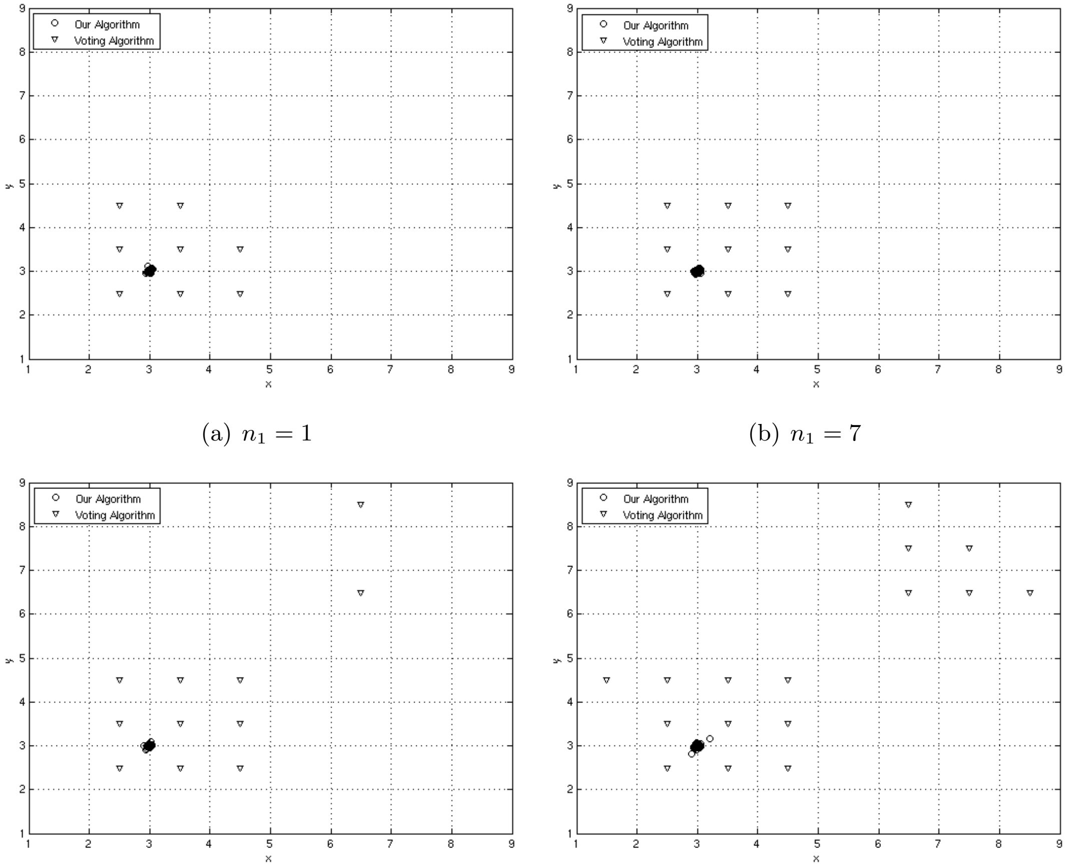


										1	1	1			
									1	1	1	1	1		
					1	1	1	2	2	1	A	1	1	1	
					1	1	1	2	2	1	●		1	1	
	1	1	1	2	2	1		1	2	2	2	2	1	1	
1	1	1	1	2	2		B ●		2	3	3	2	2	1	
1	1		● D	2	2	1		1	2	3	3	2	2	1	
1	1	1		1	2	1	1	2	2	2			1	1	1
1	1	1	1	1	1	1	1	2	2		● C		1	1	1
	1	1	1	1				1	1	1			1	1	1
									1	1	1	1	1	1	
										1	1	1	1		

Figure 5.12: Illustration of the voting algorithm (Reproduced from [46]).

For our relaxation labeling algorithm, we identify the malicious nodes first, and remove them from the network. The localization is then done by collecting the δ_i from the benign nodes and using (2.11).

Here we are demonstrating that the voting algorithm produces more incorrect location estimates as the number of malicious nodes increases. For the voting algorithm, we choose the cell size to be 1.0×1.0, and we denote the center of the cell as the event location, according to [46]. We repeat the localization experiments for 100 times, each time using a different (and random) set of 20 node locations. The estimated event locations using each algorithm are shown in Figure 5.13 under various number of malicious nodes. In Figure 5.13(a), there is only one malicious node present. The number of malicious nodes increase from seven to nine in Figure 5.13(b), Figure 5.13(c) and Figure 5.13(d). The relaxation labeling algorithms can correctly localize the event location, $(3.0, 3.0)$, with a high number of malicious nodes. Hence the estimated event locations (marked as circles) in Figure 5.13 are fairly close to each other and to the true event location. However, as the number of malicious nodes increases, a higher percentage of estimates using voting algorithm (marked as triangles) is close to the fictitious event location. The choice of cell size of 1.0×1.0 is not related to this trend, since we have observed the same phenomenon at cell size of 0.5×0.5.



(a) $n_1 = 1$

(b) $n_1 = 7$

(c) $n_1 = 8$

(d) $n_1 = 9$

Figure 5.13: Comparison of the performance of the relaxation labeling algorithm and the voting algorithm in a 20-node network. In (a), only one node is malicious. In (b), (c) and (d), the number of malicious nodes increase from seven to nine. Note that both algorithms perform equally well when there are 2 - 6 malicious nodes. Since the experiments are repeated for 100 times, each with a different set of random node locations, overlapping results are marked at the same spots in (a) - (d). The voting algorithm occasionally concludes that the event is actually at the fictitious event location, $(7.0, 7.0)$. However, our algorithm, relaxation labeling, correctly localize the event at $(3.0, 3.0)$ in every case.

5.4 Choice of Parameters

In Figure 5.6, we can see that the system performance is effected by two factors: the noise in the measurement process, which is not controlled by the user, and the parameters T_1 and T_2 that can be controlled by the user. In this section, we would like to develop ways to choose useful values of parameters.

5.4.1 The Sigmoid Function

Before we can do that, we would like to remind the reader of the sigmoid function in (5.4). Given a set of (i, j, k, b, b, b), (5.4) is essentially

$$r(\epsilon) = 1 - \frac{2}{1 + e^{-\alpha_1(\epsilon - T_1)}}$$

which is in the form of

$$r : \mathbb{R}^1 \to \mathbb{R}^1 \tag{5.12}$$

We know from (2.13) that ϵ is the sum of three discrepancies, $\epsilon_l, l \in \{a, b, c\} \in \{1, 2, \cdots, n\}$, within a triple, hence $\epsilon > 0$. So the *Domain* of (5.12) is

$$\{\epsilon \in \mathbb{R}^1 | \epsilon > 0\}$$

The *Range* of (5.12) is

$$\{r(\epsilon) \in \mathbb{R}^1 | r(\epsilon) \in [-1, 1]\}$$

since

$$\begin{aligned} \mathop{lim}_{\epsilon \to \infty} r &= -1 \\ \mathop{lim}_{\epsilon \to -\infty} r &= 1 \end{aligned}$$

The sigmoid function in (5.4) is also a solution to the differential equation

$$\frac{\partial r}{\partial \epsilon} = \frac{\alpha_1}{2}(r^2 - 1). \tag{5.13}$$

To see this, we take derivative of $r(i,j,k,b,b,b)$ with respect to ϵ, and we get

$$\begin{aligned}\frac{\partial r}{\partial \epsilon} &= 2(1+e^{-\alpha_1(\epsilon-T_1)})^{-2}e^{-\alpha_1(\epsilon-T_1)}(-\alpha_1)\\ &= 2\alpha_1(\frac{1-r}{2})^2(1-\frac{2}{1-r})\\ &= 2\alpha_1(\frac{1-r}{2})(\frac{1-r}{2}-1)\\ &= -2\alpha_1(\frac{1-r}{2})(\frac{1+r}{2})\\ &= \frac{\alpha_1}{2}(r^2-1) \qquad (5.14)\end{aligned}$$

The parameter T_1 is the inflection point of the function $r(\epsilon)$. That is to say, $\frac{\partial^2 r}{\partial \epsilon^2} = 0$ at $\epsilon = T_1$. To see this, we again take the derivative of (5.14) and set it equal to 0, and we have

$$\begin{aligned}\frac{\partial^2 r}{\partial \epsilon^2} &= \frac{\alpha_1}{2}(2r)\frac{\partial r}{\partial \epsilon}\\ &= \frac{\alpha_1}{2}(2r)\frac{\alpha_1}{2}(r^2-1)\\ &= \frac{\alpha_1^2}{2}r(r^2-1)\\ &= 0 \qquad (5.15)\end{aligned}$$

Therefore, we have $r=0$, $r=-1$ or $r=1$ as the solution to (5.15). When $r=-1$, it is clear that $r=-1$ is the asymptotic limit of $\epsilon \to \infty$, hence $r=-1$ is merely a trivial solution ($\frac{\partial r}{\partial \epsilon}=0$). Similarly, $r=1$ is another trivial solution to (5.15). The interesting solution is when $r=0$, we have

$$\begin{aligned}r(i,j,k,b,b,b) = 1-\frac{2}{1+e^{-\alpha_1(\epsilon-T_1)}} &= 0\\ \frac{2}{1+e^{-\alpha_1(\epsilon-T_1)}} &= 1\\ 1+e^{-\alpha_1(\epsilon-T_1)} &= 2\\ e^{-\alpha_1(\epsilon-T_1)} &= 1\\ \alpha_1(\epsilon-T_1) &= 0 \qquad (5.16)\end{aligned}$$

Hence $\epsilon = T_1$. The property that $\epsilon = T_1$ is an inflection point will be used in the following sections.

Analysis of the sigmoid function in (5.8) can be derived in a similar manner.

Recall from equations (5.4) and (5.8), four parameters are of interest

1. T_1: controls the shift in (5.4). See Figure 5.1
2. α_1: controls the slope in (5.4). See Figure 5.1
3. T_2: controls the shift in (5.8). See Figure 5.2
4. α_2: controls the slope in (5.8). See Figure 5.2

We now examine each of them as follows.

5.4.2 Choice of T_1

We would like to choose suitable values of T_1 so that the system error rate in terms of misclassifying nodes into wrong labels would be minimal. However, it is difficult to write any analytic equation that directly relates the system error rate to T_1.

One alternative is to think of $q_i(\lambda)$ as the objective function, and relate $q_i(\lambda)$ to T_1. We call this approach the optimization perspective. Another alternative is to simply learn from the experimental data. Our design objective in (5.4) is to choose T_1 as a threshold for the total discrepancy ϵ. By modeling the experimental data, we can choose a suitable T_1 that matches our design objective in (5.4). We call this approach data dependency perspective.

Optimization perspective

In relaxation labeling, our objective is to calculate $P_i(\lambda)$, the probability of object i having label λ. What drives $P_i(\lambda)$ is $q_i(\lambda)$. Hence we can begin the optimization approach by focusing on $q_i(\lambda)$. We denote the *correct* value for $q_i(\lambda)$ to be $G_i(\lambda)$. When node i is indeed of type λ, $G_i(\lambda)$ is 1.0. In other words, $q_i(\lambda)$ *should* be 1.0 when node i actually has label λ. On the other hand, $G_i(\lambda') = -1.0 \quad \forall \quad \lambda' \neq \lambda$ because we are labeling node i with a wrong label λ'.

Since we expect $q_i(\lambda)$ to be $G_i(\lambda)$, we can design the following objective function

$$\sum_i \sum_\lambda [q_i(\lambda) - G_i(\lambda)]^2 \tag{5.17}$$

where $i = 1, \ldots, n$ are all nodes in the sensor network, and $\lambda \in \{m, b\}$. We will take derivative of this objective function, and set it to zero.

Using chain rule, we have

$$\begin{aligned}
\frac{\partial}{\partial T_1} \sum_i \sum_\lambda [q_i(\lambda) - G_i(\lambda)]^2 &= \sum_i \sum_\lambda 2\,[q_i(\lambda) - G_i(\lambda)]\, \frac{\partial\,[q_i(\lambda) - G_i(\lambda)]}{\partial T_1} && (5.18) \\
&= \sum_i \sum_\lambda 2\,[q_i(\lambda) - G_i(\lambda)]\, \frac{\partial q_i(\lambda)}{\partial T_1} && (5.19)
\end{aligned}$$

From the definition of $q_i(\lambda)$, we can expand the term $\frac{\partial q_i(\lambda)}{\partial T_1}$ in (5.19) as

$$\begin{aligned}
q_i(\lambda) &= \frac{1}{N} \sum_j \sum_k \sum_{\lambda'} P_j(\lambda') \sum_{\lambda''} P_k(\lambda'') r(i, j, k, \lambda, \lambda', \lambda'') && (5.20) \\
\frac{\partial q_i(\lambda)}{\partial T_1} &= \frac{1}{N} (\sum_j \sum_k P_j(\lambda) P_k(\lambda) \frac{\partial r(i, j, k, \lambda, \lambda, \lambda)}{\partial T_1}) && (5.21)
\end{aligned}$$

Note that now we are treating T_1 as a parameter of r. That is to say, (5.4) is treated as $r(T_1)$ by holding the ϵ in (5.4) as a constant. Take derivative of $r(i, j, k, \lambda, \lambda, \lambda)$ with respect to T_1, we get

$$\begin{aligned}
\frac{\partial r(i, j, k, \lambda, \lambda, \lambda)}{\partial T_1} &= 2(1 + e^{-\alpha_1(\epsilon - T_1)})^{-2} e^{-\alpha_1(\epsilon - T_1)} \alpha_1 && (5.22) \\
&= 2\alpha_1 (\frac{1-r}{2})^2 (\frac{2}{1-r} - 1) && (5.23) \\
&= 2\alpha_1 (\frac{1-r}{2})(1 - \frac{1-r}{2}) && (5.24) \\
&= 2\alpha_1 (\frac{1-r}{2})(\frac{1+r}{2}) && (5.25) \\
&= \frac{\alpha_1}{2}(1 - r^2) && (5.26)
\end{aligned}$$

This arrives at a similar differential equation as (5.14).

Substituting (5.26) into (5.21), we get

$$\begin{aligned}
\frac{\partial q_i(\lambda)}{\partial T_1} &= \frac{1}{N} (\sum_j \sum_k P_j(\lambda) P_k(\lambda) \frac{\partial r(i, j, k, \lambda, \lambda, \lambda)}{\partial T_1}) \\
&= \frac{1}{N} \left[\sum_j \sum_k P_j(\lambda) P_k(\lambda) \frac{\alpha_1}{2} (1 - r^2(i, j, k, \lambda, \lambda, \lambda)) \right] && (5.27)
\end{aligned}$$

where $j = 1, \cdots, n$, $k = 1, \cdots, n$, $j \neq i$, $k \neq i$.

Hence (5.18) becomes

$$\begin{array}{l} \frac{\partial}{\partial T_1} \sum_i \sum_\lambda \left[q_i(\lambda) - G_i(\lambda)\right]^2 = \\ \sum_i \sum_\lambda \left[q_i(\lambda) - G_i(\lambda)\right] \frac{1}{N} \left[\sum_j \sum_k P_j(\lambda) P_k(\lambda) \alpha_1 (1 - r^2(i, j, k, \lambda, \lambda, \lambda))\right] \end{array} \tag{5.28}$$

Setting (5.28) to 0, we have

$$\sum_i \sum_\lambda (q_i(\lambda) - G_i(\lambda)) \sum_j \sum_k P_j(\lambda) P_k(\lambda)(1 - r^2(i, j, k, \lambda, \lambda, \lambda)) = 0 \tag{5.29}$$

The next step is to relate (5.29) to T_1. We can achieve this by using the Taylor series

$$e^x = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \cdots \simeq 1 + x \tag{5.30}$$

and we have

$$\begin{aligned} r &= 1 - \frac{2}{1 + e^{-\alpha_1(\epsilon - T_1)}} && (5.31) \\ &= 1 - \frac{2}{1 + 1 - \alpha_1(\epsilon - T_1)} && (5.32) \\ &= 1 - \frac{2}{2 - \alpha_1(\epsilon - T_1)} && (5.33) \end{aligned}$$

Using (5.33), we can have

$$\begin{aligned} 1 - r^2 &= (1 - r)(1 + r) && (5.34) \\ &= (\frac{2}{2 - \alpha_1(\epsilon - T_1)})(2 - \frac{2}{2 - \alpha_1(\epsilon - T_1)}) && (5.35) \\ &= \frac{4}{2 - \alpha_1(\epsilon - T_1)} - \frac{4}{[2 - \alpha_1(\epsilon - T_1)]^2} && (5.36) \\ &= 4\left[\frac{2 - \alpha_1(\epsilon - T_1)}{(2 - \alpha_1(\epsilon - T_1))^2} - \frac{1}{(2 - \alpha_1(\epsilon - T_1))^2}\right] && (5.37) \\ &= \frac{4(1 - \alpha_1(\epsilon - T_1))}{(2 - \alpha_1(\epsilon - T_1))^2} && (5.38) \end{aligned}$$

Substituting (5.38) into (5.29), we have

$$\sum_i \sum_\lambda (q_i(\lambda) - G_i(\lambda)) \sum_j \sum_k P_j(\lambda) P_k(\lambda) \frac{4(1-\alpha_1(\epsilon - T_1))}{(2-\alpha_1(\epsilon - T_1))^2} = 0 \tag{5.39}$$

One condition that (5.39) will hold is $q_i(\lambda) = G_i(\lambda) \quad \forall \quad i, \lambda$. However, it is impossible to derive an analytic solution of T_1 from this condition. The other condition that (5.39) will hold is $4(1 - \alpha_1(\epsilon - T_1)) = 0$, that is

$$\hat{T_1} = \epsilon - \frac{1}{\alpha_1} \tag{5.40}$$

The analytic solution in (5.40) is still difficult to be applied to real conditions since ϵ is a variable that is different for every triple in the sensor network. To obtain a useful solution for choosing T_1, we take the expected value of (5.40)

$$\begin{aligned} T_1 &= E[\epsilon] - \frac{1}{\alpha_1} && (5.41) \\ &\simeq E[\epsilon] && (5.42) \end{aligned}$$

where $E[u]$ is the expected value of u and we have assumed that in practical cases, α_1 can be chosen to be large enough that $\frac{1}{\alpha_1}$ is negligible as compared to ϵ. Hence we arrive at the analytic solution that T_1 can be chosen by $E[\epsilon]$. The probability density function of ϵ can be approximated by a histogram (see the following section for examples) and hence obtain the expected value of ϵ.

Data dependency perspective

Based on our analysis in the previous two sections, we know that T_1 controls the location where $r(\epsilon) = 0$. When $\epsilon = T_1$, it is also the location of an inflection point. Hence T_1 lies in the domain of $r(\epsilon)$, not in the range of $r(\epsilon)$. To choose a suitable value for T_1, we need to model ϵ well.

In other words, the suitable choice of T_1 depends on ϵ, the total discrepancy of a triple. To choose T_1 is a data-dependent issue.

From (2.13), we know that ϵ is the sum of three respective discrepancies in a triple. There are two terms in (2.13), namely δ_l and $\sqrt{(\hat{x_0} - x_l)^2 + (\hat{y_0} - y_l)^2}$. δ_l is the reported range from the node to the event, hence it is proportional to the scale of our coordinate system. $\sqrt{(\hat{x_0} - x_l)^2 + (\hat{y_0} - y_l)^2}$ is the distance from the node to the estimated event location, hence it is also proportional to the scale of our coordinate system. Therefore, as the scale of our coordinate system gets larger, T_1 needs to be larger (and vice versa) since it is the inflection point of ϵ. For example, in our simulations, the scale of our coordinate system is chosen to be $[0, 10] \times [0, 10]$. If we set it instead to be $[0, 100] \times [0, 100]$, then ϵ will also be scaled by 10 times, and T_1 will need to be similarly scaled.

The other factor of importance to T_1 is the measurement noise, w_i, in (2.2). ϵ measures the difference between the reported range, δ_l, and the calculated distance from the node to the event, $\sqrt{(\hat{x_0} - x_l)^2 + (\hat{y_0} - y_l)^2}$. Regardless of whether the node is malicious or benign, noise contributes to part of ϵ. Hence the suitable choice of T_1 should also take noise into account.

How do we model T_1 based on the real data? In most situations, prior calibrations can be used to find suitable parameters for the algorithm. As an example calibration, we simulate a seven-node system with two malicious nodes whose experimental settings are the same as those in Section 5.3.1. After repeating the simulation for 100 times, we show the histogram of all ϵ in Figure 5.14. We denote the ϵ for triples which are all benign or all malicious as ϵ_a. Similarly, we denote the ϵ for one benign or one malicious triples as ϵ_b. In Figure 5.14, the histogram of ϵ_a is shown in red, while part of the histogram of ϵ_b is shown in blue. The histogram of ϵ_b larger than 1 is not shown because the maximal ϵ_b has a much larger scale than any of ϵ_a.

We can see from Figure 5.14 that most of the ϵ from all-benign (or all-malicious) triples are smaller than the ϵ of one-benign (or one-malicious) triples. This is reasonable since there is some disagreement among nodes for one-benign or one-malicious triples.

In a later section, we will need the largest values of ϵ_a, therefore, for notational clarity, we denote ϵ_a in descending order as

$$\epsilon_{a1} \geq \epsilon_{a2} \geq \cdots \geq \epsilon_{a6000} \tag{5.43}$$

since there are 6000 ϵ_a samples from a seven-node simulation which is repeated for 100 times.

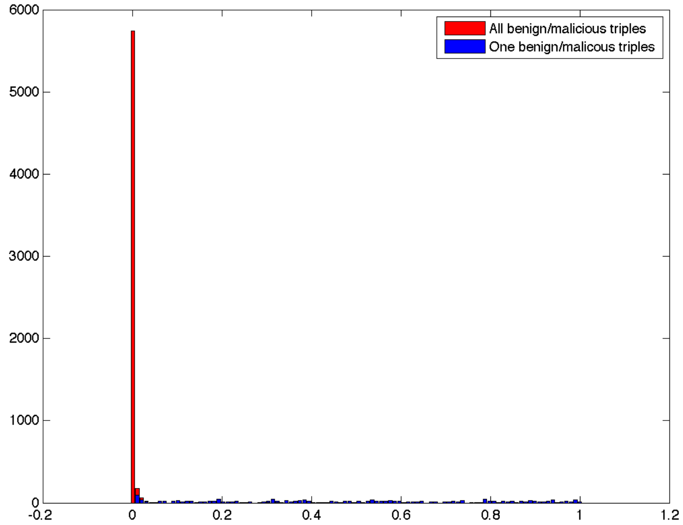


Figure 5.14: Histogram of ϵ for all triples in a seven-node simulation environment. Scale is $[0, 10] \times [0, 10]$, noise variance $\sigma^2 = 10^{-6}$. The histogram of ϵ_a is shown in red, while part of the histogram of ϵ_b is shown in blue. Note that the histogram of ϵ_b larger than 1 is not shown because the maximal ϵ_b has a much larger scale than any of ϵ_a.

We show the histogram of $\{\epsilon_{a1}, \cdots, \epsilon_{a6000}\}$ at $\sigma^2 = 10^{-6}$ in Figure 5.15.

Similarly, histogram of $\{\epsilon_{a1}, \cdots, \epsilon_{a6000}\}$ at $\sigma^2 = 10^{-5}$ in Figure 5.16. Histogram of $\{\epsilon_{a1}, \cdots, \epsilon_{a6000}\}$ at $\sigma^2 = 10^{-4}$ is shown in Figure 5.17.

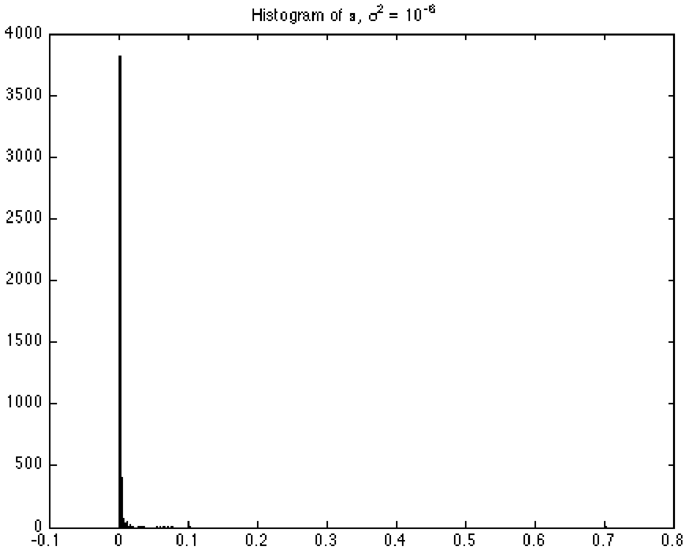


Figure 5.15: Histogram of $\{\epsilon_{a1}, \cdots, \epsilon_{a6000}\}$, $\sigma^2 = 10^{-6}$. The maximal value on the horizontal axis is chosen relative to the maximal value of ϵ_a for clearer presentation.

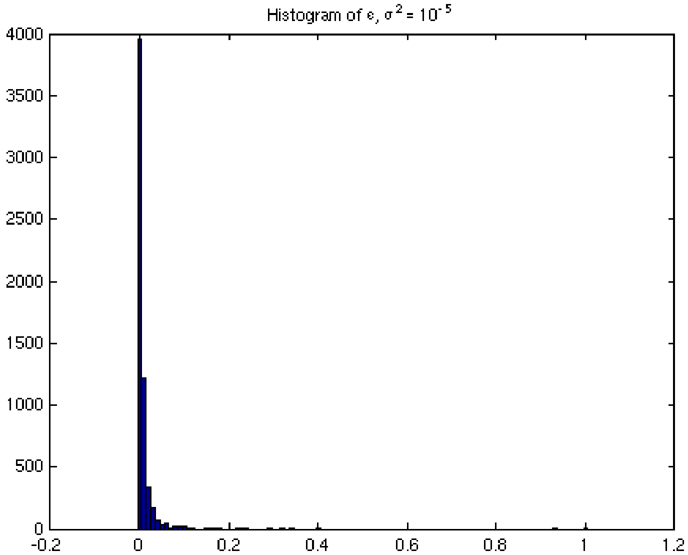


Figure 5.16: $\{\epsilon_{a1}, \cdots, \epsilon_{a6000}\}$, $\sigma^2 = 10^{-5}$. The maximal value on the horizontal axis is chosen relative to the maximal value of ϵ_a for clearer presentation.

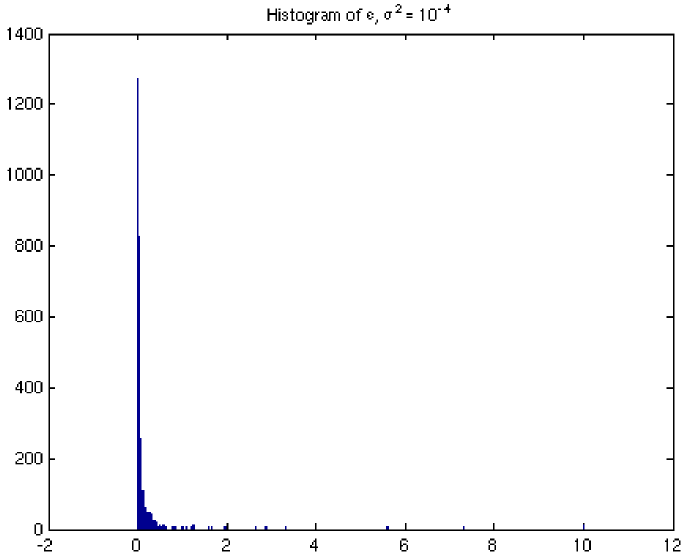


Figure 5.17: $\{\epsilon_{a1}, \cdots, \epsilon_{a6000}\}$, $\sigma^2 = 10^{-4}$. The maximal value on the horizontal axis is chosen relative to the maximal value of ϵ_a for clearer presentation.

We have observed that as the noise variance of our system increases, $\{\epsilon_{a1}, \cdots, \epsilon_{a6000}\}$ also increases. First, we plot $\frac{1}{10}\sum_{i=1}^{10}\epsilon_{ai}$ versus noise variance σ^2 in Figure 6.4(a). We choose the ten largest ϵ_a to represent the maximal values of ϵ_a. Next, we plot $\frac{1}{6000}\sum_{i=1}^{6000}\epsilon_{ai}$, the mean of ϵ_a, versus noise variance σ^2 in Figure 6.4(b).

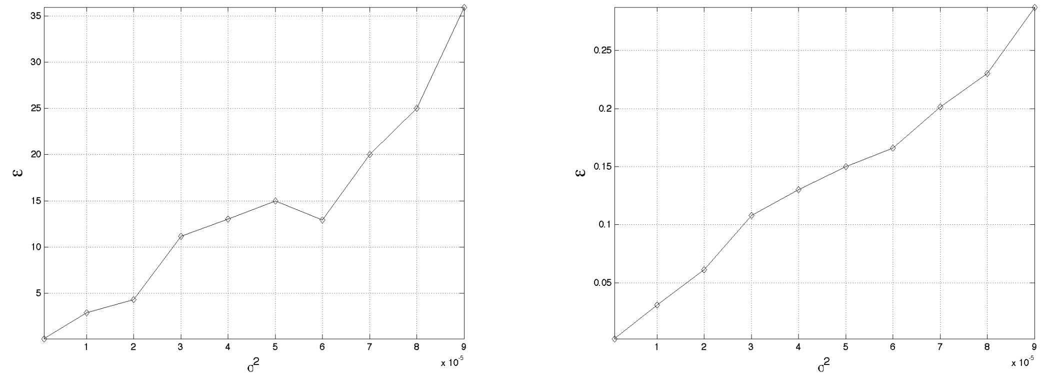
In order to construct a parametric form of ϵ, we observe that the histogram of ϵ_a is close to the pdf of an exponential random variable in Figure 5.15, Figure 5.16 and Figure 5.17. The probability density function of an exponential random variable is

$$P(x) = \kappa e^{-\kappa x} \tag{5.44}$$

where κ here is a free parameter pertaining to the exponential random variable. We show some example PDFs of an exponential random variable in Figure 5.19.

To derive T_1 as a function of ϵ_a, consider that the mean (expected value) of an exponential pdf is

$$\frac{1}{\kappa} \tag{5.45}$$



(a) $\frac{1}{10}\sum_{i=1}^{10}\epsilon_{ai}$ versus noise variance σ^2

(b) $\frac{1}{6000}\sum_{i=1}^{6000}\epsilon_{ai}$ versus noise variance σ^2

Figure 5.18: (a) The maximum (as represented by $\frac{1}{10}\sum_{i=1}^{10}\epsilon_{ai}$) of ϵ_a versus noise variance σ^2 (b) The mean (as represented by $\frac{1}{6000}\sum_{i=1}^{6000}\epsilon_{ai}$) of ϵ_a versus noise variance σ^2

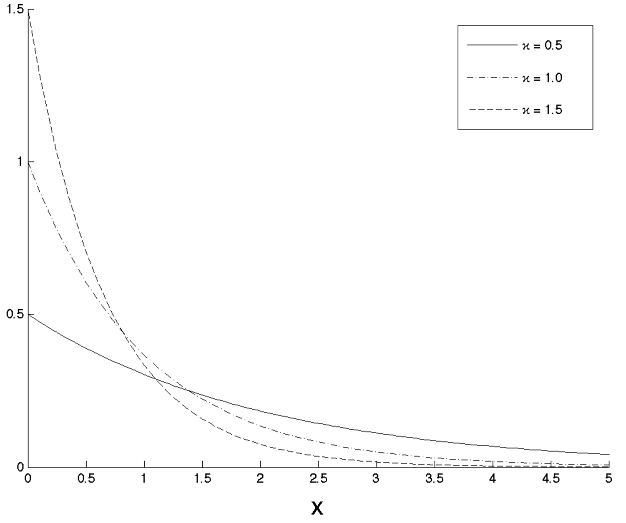


Figure 5.19: Some example PDFs of an exponential random variable.

The moment-generating function of an exponential pdf is

$$\begin{aligned}\theta(t) &= \int_0^\infty e^{tx}\kappa e^{-\kappa x}dx \\ &= \kappa\int_0^\infty e^{(t-\kappa)x}dx \\ &= \kappa\int_{-\infty}^0 e^{(\kappa-t)x}dx \\ &= \kappa\frac{e^{(\kappa-t)x}}{\kappa-t}\Bigg|_{-\infty}^0 \\ &= \frac{\kappa}{\kappa-t} \qquad (5.46)\end{aligned}$$

If we model the histogram of ϵ_a as an exponential pdf, we can use the Chernoff bound

$$\begin{aligned}P[X\geq a] &\leq e^{-at}\theta(t) \\ &= \frac{\kappa e^{-at}}{\kappa-t} \qquad (5.47)\end{aligned}$$

as a tool to choose a suitable T_1. In (5.47), the Chernoff bound means that the probability for an exponential random variable X to be larger than a certain value a is less than or equal to its moment-generating function $\theta(t)$ multiplied by e^{-at} [55]. Also, in (5.47), the moment-generating function is a function of a variable, t. To have the tightest bound, take

$$\begin{aligned}\frac{d}{dt}(\frac{\kappa e^{-at}}{\kappa-t}) &= 0 \\ \frac{-a\kappa e^{-at}(\kappa-t)-\kappa e^{-at}(-1)}{(\kappa-t)^2} &= 0 \\ \kappa e^{-at} &= a\kappa e^{-at}(\kappa-t) \\ t &= \kappa-\frac{1}{a} \qquad (5.48)\end{aligned}$$

Substituting the t in (5.47) with $t=\kappa-\frac{1}{a}$, the tightest Chernoff bound is

$$P[X\geq a]\leq a\kappa e^{(1-a\kappa)} \qquad (5.49)$$

Comparing with (5.47), (5.49) is the tightest Chernoff bound because $t=\kappa-\frac{1}{a}$ is the place where $\frac{\kappa e^{-at}}{\kappa-t}$ is minimized.

Example

When $\sigma^2 = 10^{-6}$, we can calculate the $\kappa = 377.1124$ by simply calculating the inverse of the mean of the $\epsilon_{a_1}, \cdots, \epsilon_{a_{6000}}$. Then we have

- Let $T_1 = 0.01$, $P[X \geq T_1] \leq 0.2360$
- Let $T_1 = 0.02$, $P[X \geq T_1] \leq 0.0109$
- Let $T_1 = 0.03$, $P[X \geq T_1] \leq 3.75 \times 10^{-4}$
- Let $T_1 = 0.04$, $P[X \geq T_1] \leq 1.15 \times 10^{-5}$
- Let $T_1 = 0.05$, $P[X \geq T_1] \leq 3.31 \times 10^{-7}$

This means that at the particular scale and noise variance ($\sigma^2 = 10^{-6}$), if we choose $T_1 = 0.01$, it is not a good upper bound because there is a probability of 0.236 that some ϵ_a will be greater than T_1. However, at the same scale and noise variance, $T_1 = 0.05$ would be a suitable choice because the probability of $\epsilon_a > T_1$ is fairly small ($\simeq 3.31 \times 10^{-7}$). However, it does not imply that choosing a large T_1 will always be good, as illustrated in Figure 5.6. We have to choose a T_1 that is large enough for $P[X \geq T_1]$ to be small, but not too large to raise the overall system error rate higher. We could choose a reasonably small $P[X \geq T_1]$ so that we are confident in saying that T_1 is almost at the top of all ϵ_a. In this example, we choose $P[X \geq T_1] \leq 1.0 \times 10^{-3}$, and calculate the corresponding $T_1 = 0.0262$ using (5.49). We present an algorithm for choosing T_1 in Table 5.4.

Table 5.4: An algorithm to choose T_1

1. Simulate a sensor network localization process, including malicious nodes

2. Collect all ϵ for all triples

3. Calculate κ in (5.49) by inverting the mean of the ϵ_a: $\kappa = \frac{1}{\frac{1}{N_a}\sum \epsilon_a}$
where N_a is the number of ϵ_a available

4. Set a reasonably small probability of $P[X \geq T_1]$. For example, $P[X \geq T_1] \leq 1.0 \times 10^{-3}$

5. Calculate the T_1 corresponding to $P[X \geq T_1] \leq 1.0 \times 10^{-3}$ using (5.49)

5.4.3 Choice of T_2

Similar to the role T_1 plays, T_2 controls the place where the sigmoid function in (5.8) equals 0. Hence we can choose T_2 as a function of x in (5.8). However, note that x in (5.8) measures the relative contribution from this particular node i to the total discrepancy. Since there are three nodes in a triple, and there is no reason for us to assume that one node is more important to another, setting T_2 to $\frac{1}{3}$ makes the most sense, because choosing any value other than $T_2 = \frac{1}{3}$ is equal to saying that node i (which is malicious) is more important than nodes j or k.

5.4.4 Choise of α_1 and α_2

We perform secure localization in a seven-node system with two malicious nodes using the following parameters

1. $\sigma^2 = 1.0 \times 10^{-6}$
2. $T_1 = 0.5$
3. $T_2 = \frac{1}{3}$

α_1 and α_2 merely control the slope of the sigmoid functions, and have little impact on the performance of the relaxation labeling algorithm. This can be demonstrated by the failure rate of detecting malicious nodes. We adjust different values of α_1, and the result demonstrates that with a suitable choice of σ^2, T_1 and T_2, the failure rate is consistently 0 at low noise level. The same outcome is obtained for α_2. (The graphs are not shown since they are both 0).

For completeness, we report the effect of α_1 and α_2 at higher noise variances. Again, we perform secure localization in a seven-node system with two malicious nodes using the following parameters

1. $\sigma^2 = 1.0 \times 10^{-4}$
2. $T_1 = 0.5$
3. $T_2 = \frac{1}{3}$

Figure 5.20 shows the system error rate with respect to different values of α_1 and α_2. As we have expected, α_1 and α_2 do not effect the system performance.

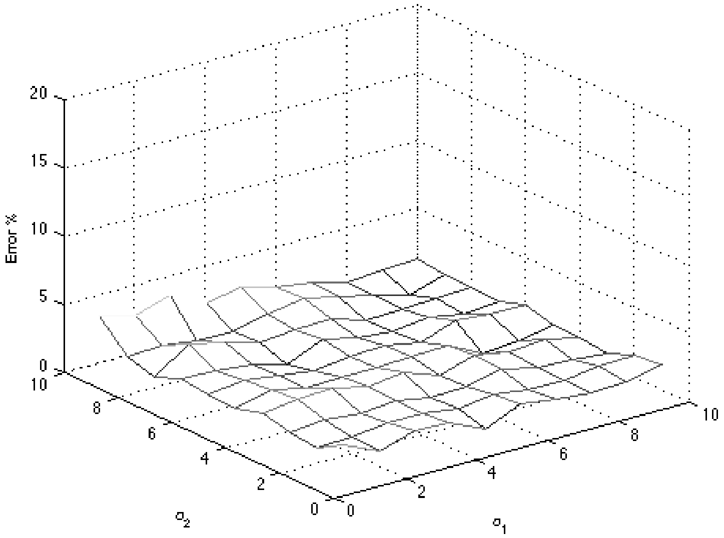


Figure 5.20: Effect of α_1 and α_2 on the system performance. We choose a higher noise variance, $\sigma^2 = 1.0 \times 10^{-4}$ in this figure. There is little or no effect on the system performance while the values of α_1 and α_2 are being changed.

5.5 Discussions on the Speed of Convergence

In this section, we aim to address the following question. How is the speed of convergence of a relaxation labeling process affected by the total number of nodes in the network?

One way to address this issue is from Appendix C of this dissertation, which formulates relaxation labeling as a gradient descent method. We briefly review the related parts of Appendix C here.

Classical relaxation labeling updates the probability as follows

$$P_i^{t+1}(\lambda_j) = \frac{P_i^t(\lambda_j)\left[1 + q_i^t(\lambda_j)\right]}{\sum_k P_i^t(\lambda_k)\left[1 + q_i^t(\lambda_k)\right]} \tag{5.50}$$

We define $Q_i(\lambda) = \left[1 + q_i^t(\lambda)\right]$, and (5.50) becomes

$$P_i^{t+1}(\lambda_j) = \frac{P_i^t(\lambda_j)Q_i^t(\lambda_j)}{\sum_k P_i^t(\lambda_k)Q_i^t(\lambda_k)} \tag{5.51}$$

The relaxation labeling process in (5.51) can be expressed as a gradient method

$$P_i^{t+1}(\lambda) = P_i^t(\lambda) + \phi s_i^t(\lambda) \tag{5.52}$$

where

$$s_i^t(\lambda) = \frac{P_i^t(\lambda)\left[Q_i^t(\lambda) - \sum_k P_i^t(\lambda_k)Q_i^t(\lambda_k)\right]}{\sum_k P_i^t(\lambda_k)Q_i^t(\lambda_k)} \tag{5.53}$$

and $\phi = 1$.

The term $s_i^t(\lambda)$ in (5.52) is indeed the rate of change for $P_i(\lambda)$

$$s_i^t(\lambda) = P_i^{t+1}(\lambda) - P_i^t(\lambda) \simeq \frac{dP_i(\lambda)}{dt} \tag{5.54}$$

Therefore, if $s_i^t(\lambda)$ is larger, $P_i(\lambda)$ will converge faster (and vice versa).

5.5.1 Speed of Convergence

We will define the number of nodes in the network as n, and the number of malicious nodes in the network as n_m. Hence the ratio of malicious nodes in the network is n_m/n. In this section, we relate $s_i^t(\lambda)$, the term that controls the speed of convergence, to n.

In this dissertation, we have only two labels, m and b. Without loss of generality, we look at $s_i(b)$ first. Equation (5.53) becomes

$$\begin{aligned}
s_i^t(b) &= \frac{P_i^t(b)\left[Q_i^t(b) - P_i^t(b)Q_i^t(b) - P_i^t(m)Q_i^t(m)\right]}{P_i^t(b)Q_i^t(b) + P_i^t(m)Q_i^t(m)} && (5.55)\\
&= \frac{P_i^t(b)Q_i^t(b) - P_i^t(b)\left[P_i^t(b)Q_i^t(b) + P_i^t(m)Q_i^t(m)\right]}{P_i^t(b)Q_i^t(b) + P_i^t(m)Q_i^t(m)} && (5.56)\\
&= \frac{P_i^t(b)Q_i^t(b)}{P_i^t(b)Q_i^t(b) + P_i^t(m)Q_i^t(m)} - P_i^t(b) && (5.57)\\
&= \frac{1}{1 + \frac{P_i^t(m)Q_i^t(m)}{P_i^t(b)Q_i^t(b)}} - P_i^t(b) && (5.58)
\end{aligned}$$

In (5.58), we need to identify those terms related to n. The terms $P_i^t(m)$ and $P_i^t(b)$ change with respect to time steps, hence we assume them to be constants, not functions of n. Rather, $Q_i^t(m)$ and $Q_i^t(b)$ will be changed once n changes.

5.5.2 Speed of Convergence vs. the Total Number of Nodes

In order to analyze $s_i^t(b)$ (or $s_i^t(m)$, for that matter) as n changes, we consider a specific case of the sensor network. Consider a sensor network of n nodes, and each node is labeled as $1, 2, \ldots, n$. Node 1 is a malicious node, while nodes $2, \ldots, n$ are all benign nodes.

Now we add an extra benign node into the network. That is, node 1 is malicious, nodes $2, \ldots, n+1$ are all benign. We denote the $Q_i^t(b)$ when the network size is n as $Q_i^t(b)\Big|_n$.

Consider node 1 (which is malicious) first. The rate of convergence for $P_1^t(b)$ is controlled by $s_1^t(b)$, which is controlled by $\frac{Q_1^t(m)}{Q_1^t(b)}$. Since we know in advance that node 1 is malicious, we know that, as $P_1^t(b)$ converges, $P_1^t(b) \leq P_1^t(m)$. Hence we have

$$Q_1^t(b)\Big|_n \leq Q_1^t(m)\Big|_n \tag{5.59}$$

By definition, we have the following equation

$$\begin{aligned} Q_1^t(m)\Big|_{n+1} &= 1 + \frac{1}{n(n-1)}\Big\{(n-1)(n-2)(Q_1^t(m)\Big|_n - 1) \\ &+ \sum_i \sum_{\lambda'} P_j(\lambda') \sum_{\lambda''} P_{n+1}(\lambda'') r(1, i, n+1, m, \lambda', \lambda'') \\ &+ \sum_i \sum_{\lambda'} P_{n+1}(\lambda') \sum_{\lambda''} P_j(\lambda'') r(1, n+1, i, m, \lambda'', \lambda')\Big\} \end{aligned} \tag{5.60}$$

In (5.60), we know that node 1 is malicious, and the rest of the nodes are benign. Hence as the system approaches convergence, only $P_1(m)$ and $P_i(b)$, $i = 2, \ldots, n+1$ are closer to 1. Other probabilities for other possible labels are small, or even close to 0. For the purpose of analysis, we will omit those terms (this is, in spirit, similar to Taylor series where we drop higher-order terms which are much smaller). Hence (5.60) can be approximated as

$$\begin{aligned} Q_1^t(m)\Big|_{n+1} &\simeq 1 + \frac{1}{n(n-1)}\Big\{(n-1)(n-2)(Q_1^t(m)\Big|_n - 1) \\ &+ \sum_i P_j(b)P_{n+1}(b)r(1,i,n+1,m,b,b) \\ &+ \sum_i P_{n+1}(b)P_j(b)r(1,n+1,i,m,b,b)\Big\} \end{aligned} \tag{5.61}$$

The terms $r(1,i,n+1,m,b,b)$ in (5.61) are positive numbers $\forall i \neq 1,(n+1)$. Equation (5.61) can be simplified in the form of $Q_1^t(m)\Big|_{n+1} \approx aQ_1^t(m)\Big|_n + b$, where a and b are both positive numbers. Hence we have

$$Q_1^t(m)\Big|_{n+1} \geq Q_1^t(m)\Big|_n \tag{5.62}$$

In a similar manner, we can have

$$Q_1^t(b)\Big|_{n+1} \leq Q_1^t(b)\Big|_n \tag{5.63}$$

Moreover, the fact the node 1 is malicious still holds, hence we have

$$Q_1^t(b)\Big|_{n+1} \leq Q_1^t(m)\Big|_{n+1} \tag{5.64}$$

From (5.59), (5.62), (5.63) and (5.66), we have

$$s_1^t(m)\Big|_{n+1} \geq s_1^t(m)\Big|_n \tag{5.65}$$

which means that the speed of convergence should be faster as we introduce one more benign node. Let us take a numerical example, suppose that $\frac{Q_1^t(b)}{Q_1^t(m)} = \frac{0.4}{0.6}$. As we introduce more benign nodes, let us say that $\frac{Q_1^t(b)}{Q_1^t(m)}$ becomes $\frac{0.2}{0.8}$. Since $\frac{0.4}{0.6} > \frac{0.2}{0.8}$, we have $\frac{1}{1+\frac{0.4}{0.6}} < \frac{1}{1+\frac{0.2}{0.8}}$. Hence we know that the speed of convergence will be be faster. We illustrate the following equation in Table 5.5

$$s_1^t(m) \simeq \frac{1}{1 + \frac{Q_1^t(b)}{Q_1^t(m)}} \tag{5.66}$$

We only select a few decreasing values of $Q_1^t(b)$ and increasing values of $Q_1^t(m)$ in Table 5.5. As Table 5.5 illustrates, $r_1^t(m)$ will go up as $Q_1^t(b)$ decreases and $Q_1^t(m)$ increases.

For the purpose of analysis, we consider only the case where there is one malicious node inside the network. When the sensor network has a different number of malicious

Table 5.5: Example values of (5.66).

$Q_1^t(b)$	$Q_1^t(m)$	$s_1^t(m)$
0.5	0.5	0.5
0.4	0.6	0.6
0.3	0.7	0.7
0.2	0.8	0.8
0.1	0.9	0.9

nodes, it becomes difficult to analyze. $Q_i^t(\lambda)$ is a collection of compatibility functions, $r_i^t(\lambda)$. The value that $r_i^t(\lambda)$ gives depends on the actual triples and how they perform the localizations. It is difficult to predict, as n gets larger, whether a $Q_i^t(\lambda)$ will be larger or smaller simply because it has more $r_i^t(\lambda)$ in the summation.

5.5.3 Experiments

We simulate a sensor network of n nodes, in which only the first node is malicious. We then record the number of iterations required to converge. After we repeat such experiments for 10 times, we average the number of iterations and show them in Figure 5.21. We can see that the system converges faster as n gets larger. This agrees with our analytical solution in (5.65).

However, the speed of convergence will not get larger as long as the network grows to a certain size. Recall that, by definition, $0 \leq Q_i^t(\lambda) \leq 1$. Hence in this example, $Q_1^t(m)$ is getting closer to 1, while $Q_1^t(b)$ is getting smaller and closer to 0. Let us consider a numerical example. Let us say that $\frac{Q_1^t(b)}{Q_1^t(m)} = \frac{0.1}{0.9}$. As the network size gets larger, $\frac{Q_1^t(b)}{Q_1^t(m)}$ becomes $\frac{0.05}{0.95}$, which is not much different from $\frac{0.1}{0.9}$. Hence the speed of convergence will not indefinitely increase as the network size grows.

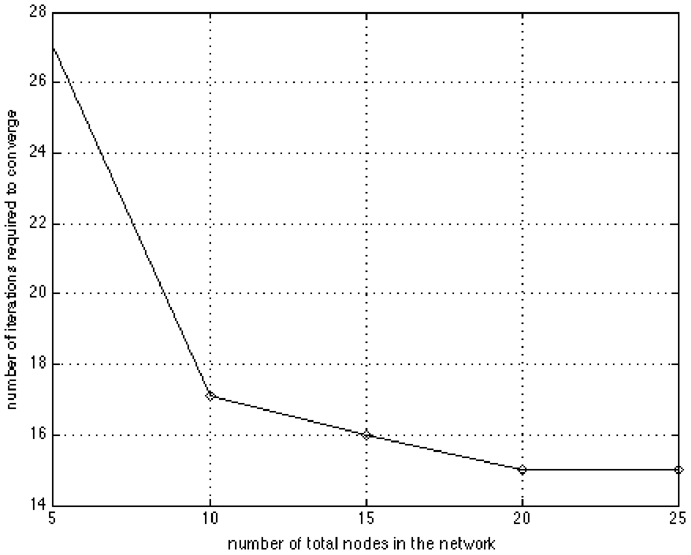


Figure 5.21: Number of iterations required to reach convergence at various number of network sizes. We can see that the system converges faster as n gets larger. However, the speed of convergence does not get any larger as the network reaches a certain size (20 in this experiment).

Chapter 6

Secure Tracking Using Relaxation Labeling

Our algorithm can successfully detect malicious nodes in event localization scenarios. Now we design and extend the relaxation labeling algorithm for secure tracking issues. For background on target tracking, please refer to Chapter 3.

6.1 Related Work

There is little or no literature on the topic of secure tracking in sensor networks. Capkun et al. [6] propose a protocol to securely verify the time of encounters in multi-hop networks. However, Capkun et al. [6] did not address Bayesian tracking as defined by (3.1) and (3.2). Besides, the work in [6] is based on an ad hoc network, which is different from our centralized setting.

Our work uses relaxation labeling to identify the malicious nodes in the sensor network, and use only information from benign nodes to build the particle filter to perform target tracking [10]. Both the problem definition and the proposed algorithm are novel [10].

6.2 Algorithm to Detect Malicious Nodes while Tracking

We propose a secure tracking algorithm based on relaxation labeling algorithms [51, 66, 67, 52, 31, 42]. Whereas for localization, at least three active nodes are required to localize an event; for tracking, fewer may be used. For example, in [47], Liu et al. propose to perform target tracking by activating one node at a time. However, for security purposes, we activate two or more nodes at each time step k in the sensor networks. With such redundancy, any inconsistency in the behavior of nodes can be exploited. Besides, activating more nodes at each time step allows us to remove some portion of them that are deemed malicious.

We begin the explanation of our algorithm by considering the case where we activate two nodes at each time step, as illustrated in Figure 6.1. The cases in which we activate three or more nodes will be discussed later in this section. As defined in Section 3.2, we assume that each sensor node has adequate processing power, and can calculate the current estimate of the target location based on previous estimates. Let us begin with time step $t = 0$, when the initial position of the target, $p(\mathbf{x}^0)$, is assumed to be known. $p(\mathbf{x}^0)$ is passed to the two nodes (nodes 1 and 2) activated[1] at time step $t = 1$. After calculating $p(\mathbf{x}^1|z^1)$ using particle filter algorithms, each of the two nodes at $t = 1$ will pass $p(\mathbf{x}^1|z^1)$ to both of the two nodes at $t = 2$ (nodes 3 and 4). Based on the two different $p(\mathbf{x}^1|z^1)$ from $t = 1$ given, the two nodes at $t = 2$ will each produce two different $p(\mathbf{x}^2|z^2)$. For example, in Figure 6.1, node 3 receives $p(\mathbf{x}^1|z^1)$ from both nodes 1 and 2, so it can calculate two different $p(\mathbf{x}^2|z^2)$ based on them. What if, in Figure 6.1, that node 1 is malicious, and node 2 is benign (assuming that node 3 is benign)? Then the two estimates of $p(\mathbf{x}^2|z^2)$ calculated by node 3 could be drastically different since the $p(\mathbf{x}^1|z^1)$ reported by node 1 is already different from the $p(\mathbf{x}^1|z^1)$ by node 2.

Following such logic, we design our relaxation labeling algorithm based on sets of three nodes, which we again denote as *triples*. There are three types of triples

1. Type I : a triple consisting of two predecessor nodes and one successor nodes. For example, nodes (1,2,3) and (1,2,4) in Figure 6.1
2. Type II: a triple consisting of one predecessor node and two successor nodes. For example, nodes (1,3,4) and (2,3,4) in Figure 6.1
3. Type III: a triple consisting solely of nodes at the same time instant. For example, nodes (1,2,3) and (4,5,6) in Figure 6.5

[1] We use the node activation algorithm discussed in Section 3.1.3.

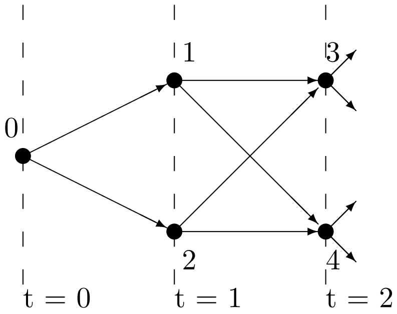


Figure 6.1: At $t = 0$, we assume $P(\mathbf{x}^0)$ is known, and this information is passed to the two nodes activated at $t = 1$. Using the particle filter algorithm, node 1 and node 2 can each calculate $p(\mathbf{x}^1|z^1)$, and those information is passed to node 3 and node 4. Both node 3 and node 4 have two inputs, hence they will each produce two distinctive $p(\mathbf{x}^2|z^2)$. Note that in this model, each node (e.g. node 3) reports BOTH of its estimates to the central processor.

In our algorithm, the nodes in a Type III triple do not pass information to each other, so there is no way to exploit their inconsistency. We will only use Type I and Type II triples in our algorithm. As a result, we will design two different compatibility functions for these two types of triples.

Note that the three Types are mutually exclusive and collectively exhaustive for triples. However, a single node could belong to one or more Types. For example, in Figure 6.1, node 1 belong to both $(1, 2, 3)$, a Type I triple, and $(1, 3, 4)$, a Type II triple.

When malicious nodes collude in a localization scenario, as in Chapter 2 and 5, they report ranges which are distances from the fictitious event position to the node positions. In a tracking scenario, nodes report a PDF, hence the way they collude is to report PDF's whose means lies on the *fictitious target location*, as defined in Chapter 3.

6.2.1 Type I Triples

In this section, we focus on Type I triples and how we define the compatibility function for them. In Figure 6.2, we illustrate a Type I triple, and we denote the predecessor nodes as $g1$ and $g2$; and the successor node as s. For clarity of notation, explicit references to time are omitted unless required by context.

Both node $g1$ and node $g2$ pass information to node s, hence node s can examine

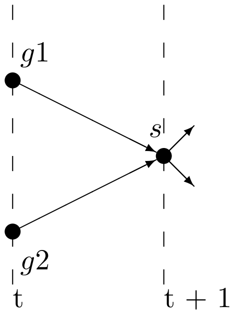


Figure 6.2: Two predecessor nodes passing information to one successor node.

the difference between the two beliefs that it produces and reports. We denote the belief that node s calculates based on the belief of node $g1$ as $p_1(\mathbf{x}|z)$. Similarly, we denote the other belief that node s calculates as $p_2(\mathbf{x}|z)$. We can quantify their difference by comparing the expected values of $p_1(\mathbf{x}|z)$ and $p_2(\mathbf{x}|z)$

$$\begin{aligned} d &= || < p_1(\mathbf{x}|z) > - < p_2(\mathbf{x}|z) > || && (6.1) \\ &= || \int \mathbf{x}[p_1(\mathbf{x}|z) - p_2(\mathbf{x}|z)]d\mathbf{x}|| && (6.2) \end{aligned}$$

where $||\cdot||$ denotes the 2-norm.

Let us consider whether $g1$, $g2$ or s is malicious or benign here. Since each node in a Type I triple could have two possible labels: malicious or benign, there ought to be $2^3 = 8$ combinations.

If node s is malicious, then, regardless of what were passed to it from its predecessors, node s will report two $p(\mathbf{x}|z)$, both of which have means that fall on the fictitious target location. Hence d should be close to 0 if node s is malicious.

If node s is benign, there are three cases:

1. One of its predecessor is malicious (the other benign)
2. Both nodes $g1$ and $g2$ are malicious
3. Both nodes $g1$ and $g2$ are benign

If node s is benign, and one of its predecessors is malicious, the two outputs from node s would disagree on each other. Hence we expect d to be large in case 1. However, cases 2 and 3 are equivalent, since in case 2 the two malicious nodes are colluding and both point to the fictitious location. Hence in both case 2 and case 3, we expect d to be small. We list all the possible cases for a Type I triple in Table 6.1. Note that in Figure 6.1, the behaviors of Table 6.1 apply to nodes $(1, 2, 3)$ and $(1, 2, 4)$. A formulation for "small" and "large" will be presented in Section 6.2.3.

Table 6.1: Expected behaviors in d for a Type I triple

Predecessor 1	Predecessor 2	Successor	Behavior
malicious	malicious	malicious	$d \simeq 0$
malicious	malicious	benign	d small
malicious	benign	malicious	$d \simeq 0$
malicious	benign	benign	d large
benign	malicious	malicious	$d \simeq 0$
benign	malicious	benign	d large
benign	benign	malicious	$d \simeq 0$
benign	benign	benign	d small

6.2.2 Type II Triples

In a Type II triple, one predecessor node will pass its $p(\mathbf{x}|z)$ to two different successors. We highlight such scenario in Figure 6.3. We call the predecessor node g, and the two successors node $s1$ and $s2$ in this section. Assuming that node g is benign, and one of the successors is malicious (and the other benign), the outputs from the two successor nodes would be different. Again, we can exploit the inconsistency whenever the nodes are behaving differently.

We denote (for Type II triples) the belief that node $s1$ calculates as $p_1(\mathbf{x}|z)$, and it has a different meaning from the $p_1(\mathbf{x}|z)$ defined for Type I triples. Similarly, the belief that node $s2$ calculates is denoted as $p_2(\mathbf{x}|z)$. Then we can calculate the difference between $p_1(\mathbf{x}|z)$ and $p_2(\mathbf{x}|z)$ using (6.1).

Regardless of whether node g is malicious or not, as long as one successor is

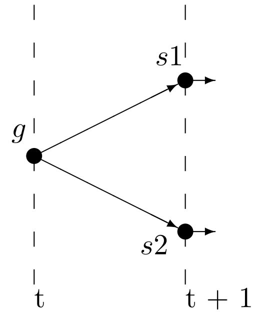


Figure 6.3: One predecessor node passing information to two successor nodes.

malicious and the other is not, we expect d to be large. What if both of node $s1$ and node $s2$ are benign? Since they are given the same input from node g, their outputs should be similar because they are both benign nodes. Hence we expect d to be small when both of node $s1$ and node $s2$ are benign. If both of node $s1$ and node $s2$ are malicious, we expect $d \simeq 0$, since no matter what their inputs are, they will collude in their reports. We list all 8 possible behaviors in Table 6.2.

Table 6.2: Expected behaviors in d for a Type II triple

Predecessor	Successor 1	Successor 2	Behavior
malicious	malicious	malicious	$d \simeq 0$
malicious	malicious	benign	d small
malicious	benign	malicious	d small
malicious	benign	benign	d small
benign	malicious	malicious	$d \simeq 0$
benign	malicious	benign	d large
benign	benign	malicious	d large
benign	benign	benign	d small

6.2.3 Algorithm Details

The expected behaviors in Table 6.1 and 6.2 are the core of our relaxation labeling algorithm. We define a compatibility function, r, based on d. In Table 6.1 and 6.2, if d is expected to be small or close to 0, we have

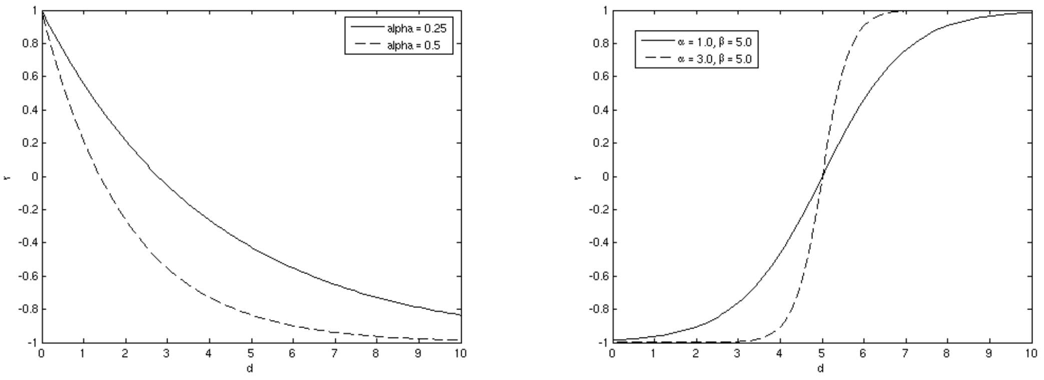
$$r = 2e^{-\alpha d} - 1 \tag{6.3}$$

On the other hand, if d is expected to be large, we will use this compatibility function

$$r = \frac{2}{1 + e^{-\alpha(d-\beta)}} - 1 \tag{6.4}$$

where in (6.3) and (6.4), α and β are parameters. Note that the compatibility function r will always return a value between -1 and 1. The higher the value that r returns, the more compatible the 3 nodes are. For example, in Figure 6.1, if we assume node 1 is malicious, node 2 is benign, and node 3 is malicious, then we expect $d \simeq 0$, as in Table 6.1. We then calculate r using $2e^{-\alpha d} - 1$. If the assumption that node 1 is malicious, node 2 is benign, and node 3 is malicious is indeed true, then the empirical data, d, should lead r to return a value close to 1. Otherwise, it should return a negative number (which is larger than -1).

We illustrate some parameter settings of (6.3) and (6.4) in Figure 6.4. In Figure 6.4(a), we expect d to be small for a correct labeling, hence a small d will lead r to return a value close to 1. On the other hand, we expect d to be large in Figure 6.4(b), hence a large d will make r return a value close to 1.



(a) (6.3)

(b) (6.4)

Figure 6.4: Illustration of some parameter settings of (6.3) and (6.4)

So if we assume node 1 is malicious, how is that compatible with, say, node 2 being malicious and node 3 benign? How about node 1 being malicious, node 2 and 3 both being benign? In each possible case, we can calculate the compatibility function r using (6.3) or (6.4). We denote malicious as m and benign as b, and we define a function q which accumulates all the effects of labeling node i as λ

$$q_i^t(\lambda) = \frac{1}{N} \sum_j \sum_k \sum_{\lambda'} P_j(\lambda') \sum_{\lambda''} P_k(\lambda'') r(\cdot), \tag{6.5}$$

where $N = (n-1)(n-2)$, n is the number of nodes in the network, $j = 1, ..., n$, $k = 1, ..., n$, $j \neq i$, $k \neq i$, $j \neq k$, and $P(\cdot)$ is a confidence function to be defined and explained later. As before, in (6.5), $q_i^t(\lambda)$ is a way to tally all the possibilities when we label node i as λ, label node j as λ' and label node k as λ''. Note that in (6.5), t is the iteration number, and this superscript on the right-hand side has been omitted for clarity. Furthermore, in (6.5), we exclude all the cases that node i, j and k are at the same time step in the tracking process.

Finally, we define *confidence* $P(\lambda)$ as in Chapter 5. The confidence of node i having label λ_j is denoted as $P_i(\lambda_j)$, and

$$0 \leq P_i(\lambda_j) \leq 1 \qquad \forall i, j \tag{6.6}$$

and

$$\sum_j P_i(\lambda_j) = 1. \tag{6.7}$$

remain the same.

Note that in (6.7), we only have two labels, hence $\lambda_j \in \{m, b\}$. Being labeled as m means that this node is malicious, and being labeled λ_b means benign. Following [51], we will iteratively update the confidence of node i having label j as

$$P_i^{t+1}(\lambda_j) = \frac{P_i^t(\lambda_j)\left[1 + q_i^t(\lambda_j)\right]}{D_i^t}, \tag{6.8}$$

where $D_i^t = \sum_k P_i^t(\lambda_k)\left[1 + q_i^t(\lambda_k)\right]$ is a normalization required to ensure that $P_i(\lambda_j)$ sums to 1, and t stands for iteration. If $P_2^t(m)$ converges to 1, while $P_2^t(b)$ converges to 0, then node 2 is found to be malicious.

In Figure 6.1, we apply the relaxation labeling process to nodes 1, 2, 3 and 4. If, say, $P_1(m)$ approaches 1 after certain iterations in the relaxation labeling process, then we determine node 1 to be malicious. After the relaxation labeling process, we remove malicious nodes, and average the $p(\mathbf{x}|z)$ of benign nodes at $t = 2$. The averaged $p(\mathbf{x}|z)$ is passed on to the two nodes active at $t = 3$. The two nodes at $t = 3$ will each calculate $p(\mathbf{x}|z)$, and pass them on to the two nodes at $t = 4$. We then use the four nodes at $t = 3, 4$ to perform relaxation labeling. After the malicious nodes are detected, we average the correct results from benign nodes, and produce a single $p(\mathbf{x}|z)$ to transmit to the two nodes at $t = 5$. The process repeats afterwards.

It is also possible to activate more nodes at each time step. What is more, if we have more than 3 nodes, we can not only do tracking, but also localization of the target. This will provide added security mechanism using the secure localization algorithms. Figure 6.5 illustrates the case when we activate three nodes at each time step in tracking. We will examine any combination of 3 nodes in the relaxation labeling process, except the combination of nodes $(1, 2, 3)$ or nodes $(4, 5, 6)$. If we select two predecessor nodes and one successor node, we should use the compatibility functions as in Table 6.1. Similarly, if the 3 nodes that we choose have one predecessor and two successors, we will use the compatibility function in Table 6.2. The algorithm is summarized in Table 6.3 and in Figure 6.6.

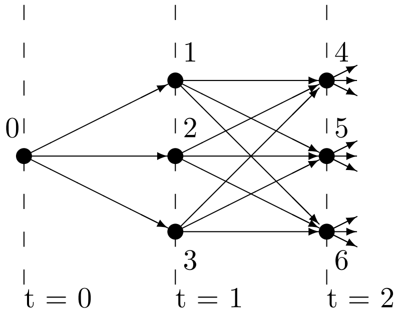


Figure 6.5: We can activate three nodes at each time step during the tracking process. Each successor node will have three inputs, hence it can produce three different outputs. The inconsistency between its three outputs can be used in the relaxation labeling process.

Table 6.3: Secure tracking algorithm using relaxation labeling

time t = k	Receive $p(\mathbf{x}_{k-1}\|z_{k-1})$
	(if $k == 1$, $p(\mathbf{x}_0\|z_0) \equiv p(\mathbf{x}_0)$ is known)
	Activate n nodes
	Node i calculates $p^i(\mathbf{x}_k\|z_k)$
time t = $k+1$	Activate n nodes
	Each node receives $p^i(\mathbf{x}_k\|z_k)$, $i = 1, \ldots, n$
	Node j calculates $p^j(\mathbf{x}_{k+1}\|z_{k+1})$, $j = 1, \ldots, n$
	Use relaxation labeling algorithm
	Remove malicious nodes
	Average the results to obtain $\hat{p}(\mathbf{x}_{k+1}\|z_{k+1})$
time t = $k+2$	Go to time t = k; use the same algorithm except
	k is replaced with $(k+2)$
	$p(\mathbf{x}_{k-1}\|z_{k-1})$ is replaced with $\hat{p}(\mathbf{x}_{k+1}\|z_{k+1})$

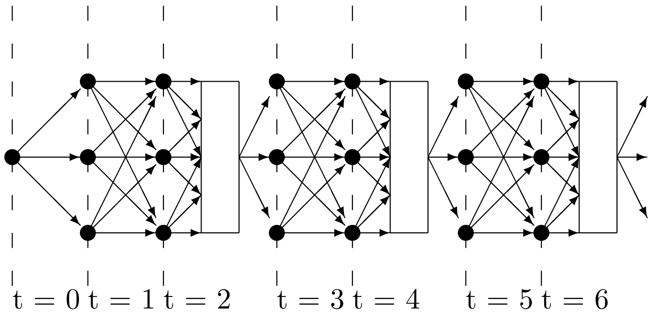


Figure 6.6: Illustration of the relaxation labeling algorithm. The rectangular box stands for relaxation labeling algorithm. At each time step (except time 0), we activate 3 nodes. For every 3 nodes (except time 0), we perform relaxation labeling algorithm to detect malicious nodes. After removing malicious nodes and average the results from benign nodes, the relaxation labeling algorithm produces a correct result and pass it on to the next time step.

6.3 Experimental Results

In this section, we demonstrate results of relaxation labeling for secure tracking. The target in our experiments is traveling in a two-dimensional space. That is, the state vector is $\mathbf{x} = [x, y, \dot{x}, \dot{y}]^T$. We begin our experiments with a linear motion model of the target as

$$\mathbf{x}^k = \mathbf{x}^{k-1} + [0.25, 0.25, 0, 0]^t + \mathbf{v}^{k-1} \tag{6.9}$$

where $\mathbf{v}^{k-1}$ is the process noise and the initial state is $\mathbf{x} = [0.1, 0.2]^t$. In (6.9), we assume that the target is traveling at a constant velocity. This assumption is for the simplicity of our simulations, and such restrictions can be relaxed. For example, in [64], Zhou et al. proposed a linear motion model in which both the velocity and noise variance are updated at each time step. Another issue worth of note is that, although (6.9) is linear and looks simple, in modeling real-world applications such as tracking objects in video sequences [64] a linear model does a fairly good job.

Our sensor model is

$$z^k = \frac{50}{\|\mathbf{x}^k - \boldsymbol{\xi}_i\|} + w^k \tag{6.10}$$

w^t is the measurement noise.

To choose suitable variances for the process and measurement noises, consider the evolution of the state sequences, $\mathbf{x}$, over a certain period of time steps. For example, here we choose time steps $t = 0, 1, \cdots, 50$. At $t = 0$, the target starts at the origin of our coordinate system, that is, it is given an initial position of $\mathbf{x} = [0.1, 0.2]^t$. We confine our observations to be within this 50-time-step window, because such a window is wide enough to demonstrate the dynamics of the target maneuvering and to show the effectiveness of our algorithm. The assumption of the initial position, $\mathbf{x} = [0.1, 0.2]^t$, is also reasonable because although the target is constantly maneuvering, we can always choose any time step (during the path of the target) as $t = 0$ and *that* particular target location as the origin of our coordinate system.

The initial position of the target is $\mathbf{x} = [0.1, 0.2]^t$, and it is traveling with a constant velocity of $\dot{\mathbf{x}} = [0.25, 0.25]$. After 50 time steps, therefore, it will be at a position around $\mathbf{x} = [12.6, 12.7]^t$. Adding the influence of noise, we expect that the position of the target, at $t = 50$, will generally have a range of $0 \leq \mathbf{x}_1, \mathbf{x}_2 \leq 15$. Hence we choose an isotropic covariance matrix for the disturbance, $\mathbf{v}$, to be $\mathbf{Q} = \begin{bmatrix} 0.15 & 0 \\ 0 & 0.15 \end{bmatrix}$. To choose an appropriate noise variance, we also observe from (6.10) that the measurement z^k is obtained by $\frac{50}{\|\mathbf{x}^k - \boldsymbol{\xi}_i\|}$, where $\|\mathbf{x}^k - \boldsymbol{\xi}_i\|$ is the distance from the current target position to the node position. We use the node activation algorithm in Section 3.1.3 to activate nodes, hence the activated nodes are fairly close to the latest target position. Hence we can assume that $0 < \|\mathbf{x}^k - \boldsymbol{\xi}_i\| < 2$, and $0 < \frac{50}{\|\mathbf{x}^k - \boldsymbol{\xi}_i\|} < 30$. So we choose the variance of the measurement noise w^t to be 3.0, although it is trivial to vary in simulations.

6.3.1 Tracking with Multiple Sensor Nodes

We begin our experiments with target tracking by activating multiple (redundant) sensor nodes. The tracking result in a two-dimensional space using three sensor nodes is shown in Figure 6.7. Note that the three sensor nodes act independently. We denote the nodes as s_1, s_2 and s_3. The node at $t = k$, s_1^k, will pass its estimate, $p(\mathbf{x}^k|z^k)$ to node s_1^{k+1} only. Similarly, s_2^k will pass information to s_2^{k+1} only, and s_3^k will interact with s_3^{k+1} only. In other words, the three activated nodes at any time step act independently, as shown in Figure 6.7. The central processor collects all of the data from each active sensor node, and it performs the relaxation labeling algorithm to detect malicious nodes before activate future nodes.

We can also activate more sensor nodes to perform tracking. For example, we activate four sensor nodes in Figure 6.8 and five nodes in 6.9. Note that in Figure 6.7, Figure 6.8 and Figure 6.9, there is no malicious node present.

In the simulations illustrated in Figure 6.8 and Figure 6.9, all of the nodes can sense the target, and the only difference between them is the number of nodes activated. Hence the node positions are not shown. Node selection is shown in the next section.

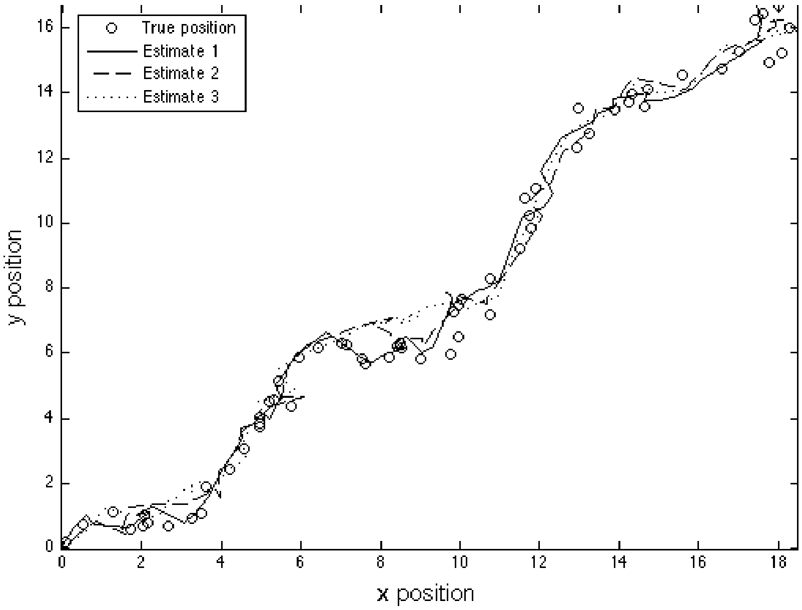


Figure 6.7: Tracking of the target in a two-dimensional space, **x**, by using three sensor nodes. We activate three sensor nodes at each time. The three sensor nodes act independently to perform tracking.

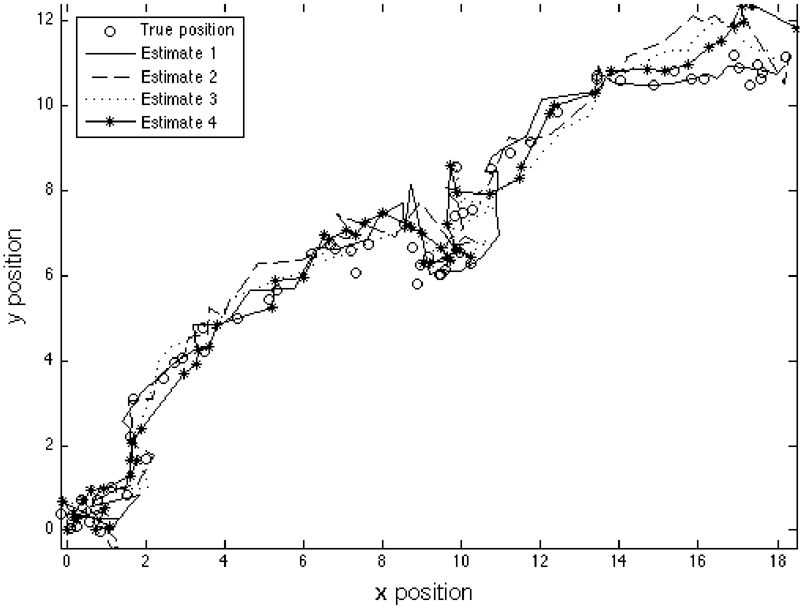


Figure 6.8: Tracking of the target by using four sensor nodes. The experimental setup is identical to Figure 6.7. The only difference is that we activate four nodes.

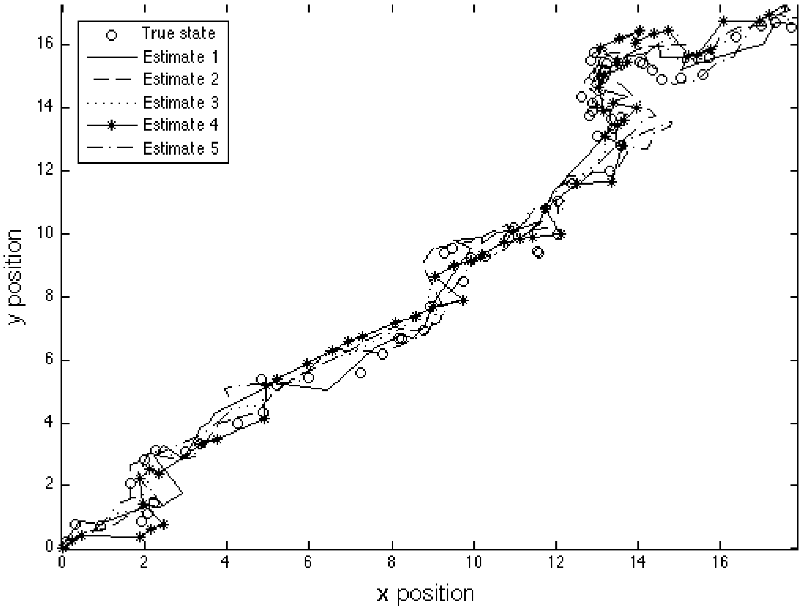


Figure 6.9: Tracking of the target by using five sensor nodes. This experiment is similar to Figure 6.7 except that we activate five nodes.

6.3.2 Node Selection Algorithm

We follow the node selection algorithm proposed in [47] to calculate the mutual information between $p(\mathbf{x}^{k+1}|z^k)$ and $p(z^{k+1}|z^k)$[2]. Here we demonstrate a target traveling in a two-dimensional space, as shown in Figure 6.10. In the area that the target is likely to pass, we have deployed 500 sensor nodes which are randomly located within $[0, 15] \times [0, 15]$. At every time step, we choose 20 nodes which are closest to the current (known) target location, $\mathbf{x}^k$. Then for each of the 20 nodes, we can calculate the mutual information between $p(\mathbf{x}^{k+1}|z^k)$ and $p(z^{k+1}|z^k)$ [47], and pick the 3 nodes that have the highest mutual information values. Figure 6.10 shows the three nodes that have the highest mutual information at $t = 20$, while Figure 6.11 shows the three nodes at $t = 30$. Recall from Section 3.1.3 that the calculation of mutual information (for activation of the next nodes) is equivalent to calculating velocity implicitly.

We enlarge the sensor field to $[0, 15] \times [0, 15]$ because we are dealing with a maneuvering target in Chapter 6. The target is maneuvering with a certain unpredictable disturbance, so it is better to enlarge the sensor field to $[0, 15] \times [0, 15]$.

[2] This is explained in Section 3.1.3

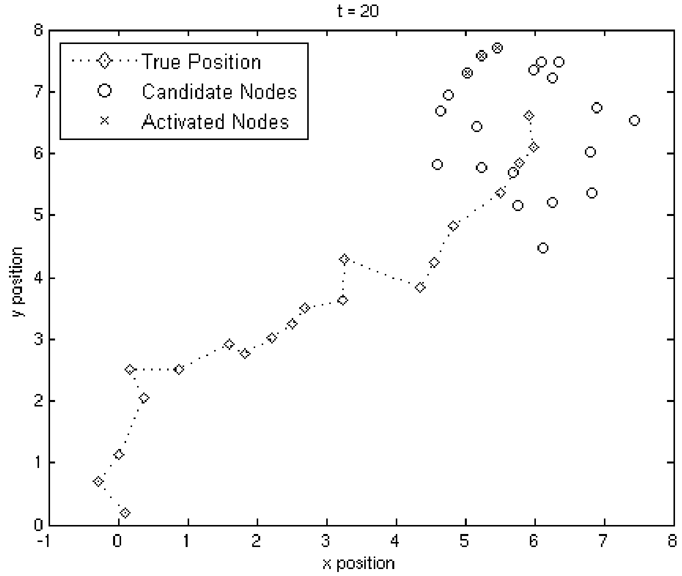


Figure 6.10: Illustration of the node activation results. The states (only positions are shown) of the target are shown as diamonds. At $t = 20$, 20 nodes are chosen to calculate the mutual information, which are shown as circles. Among them, three nodes that have the highest mutual information will be selected as active nodes at $t = 21$, which are shown as filled circles. We can see from the trajectory of the target at $t = 19, 20$ that the current best estimate of velocity is in the northwest direction. Hence the three nodes which are in the northwest direction of the target position at $t = 20$ are activated. This agrees with the highest mutual information that we calculated.

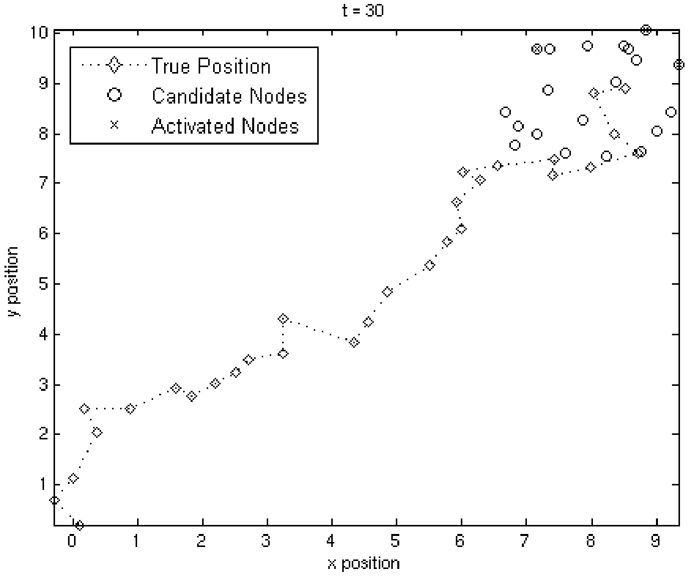


Figure 6.11: Illustration of the node activation algorithm. This is from the same experiment on Figure 6.10, except that the result here is extracted at $t = 30$. Unlike Figure 6.10, it is harder to see where the target is heading based on its trajectory at $t = 28, 29, 30$. Hence the three nodes activated at $t = 30$ do not appear to fall on one particular spot that the target is likely heading.

6.3.3 Secure Tracking Results

In this section, we demonstrate the performance of relaxation labeling in secure tracking, as compared to averaging the results from the multiple sensor nodes. Recall that we have activated several multiple sensor nodes at each time step, and they act independently to perform target tracking. One method of secure tracking is simply to average the calculated $p(\mathbf{x}|z)$ over the sensor nodes at each time step. We denote this method as *averaging*. Our algorithm, however, detect malicious nodes first, and average the results from only benign nodes.

First, we activate three sensor nodes in Figure 6.12. Denoting the three nodes at each time steps as s_1^k, s_2^k and s_3^k, we choose one node to be malicious at every ten time steps. In Figure 6.12, we have s_1^{10}, s_2^{20}, s_3^{30}, s_1^{40} and s_2^{50} as malicious nodes. Over the 60 time steps, we can average the result of the three paths, and calculate the mean-squared error (MSE) with respect to the true target path. The MSE of such averaging algorithm is 1.2349 in Figure 6.12.

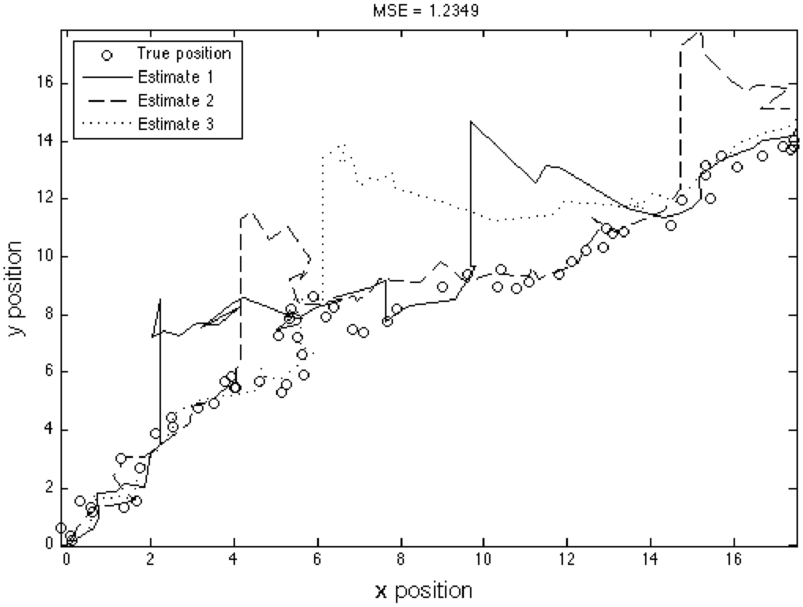


Figure 6.12: Adding malicious nodes to the sensor network. We choose sensor nodes s_1^{10}, s_2^{20}, s_3^{30}, s_1^{40} and s_2^{50} to be malicious. In other words, there is one malicious nodes (out of three) at $t = 10, 20, 30, 40, 50$.

Next, we compare the performance of using relaxation labeling and averaging (without relaxation labeling). Figure 6.13 shows the result of activating three sensor nodes. By using relaxation labeling to remove malicious nodes, the MSE decreases to 0.4938 in Figure 6.13.

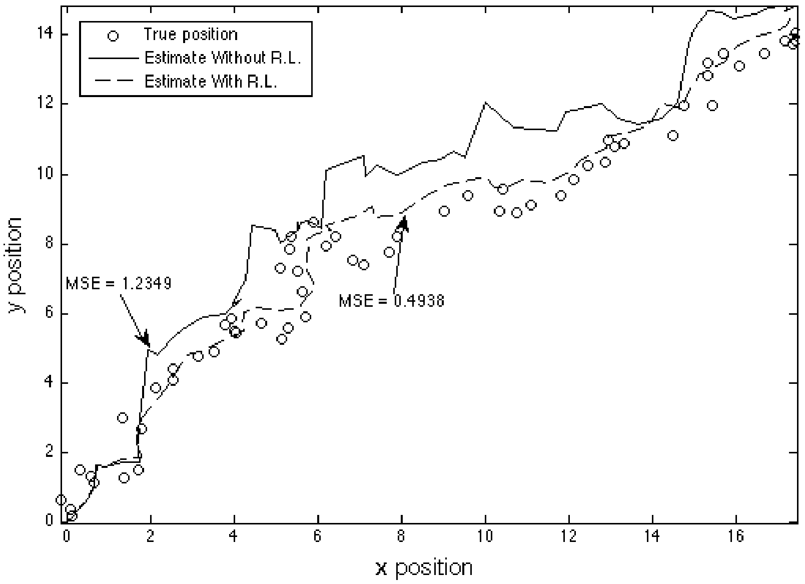


Figure 6.13: Comparison of relaxation labeling and averaging. The solid line is obtained by averaging the three paths in Figure 6.12. The dotted line is obtained by removing malicious nodes using relaxation labeling.

Next, we activate four nodes at each time step, as shown in Figure 6.14. The malicious nodes are s_1^{10}, s_2^{20}, s_3^{30}, s_4^{40} and s_1^{50}. The MSE over the 60 time steps is 0.7090, which is better than the MSE in Figure 6.12. This comes at the price of activating more sensor nodes and utilizing more system resources. The result using relaxation labeling to remove malicious nodes is shown in Figure 6.15. We can see that the MSE by using relaxation labeling has been decreased to 0.3669.

In a similar manner, we activate five nodes in Figure 6.16. The result of using relaxation labeling to remove malicious nodes is shown in Figure 6.17. Figure 6.16 and Figure 6.17 have identical experimental setup as the previous one, except that we active five sensor nodes. It is worth noting that since we only have one malicious node present

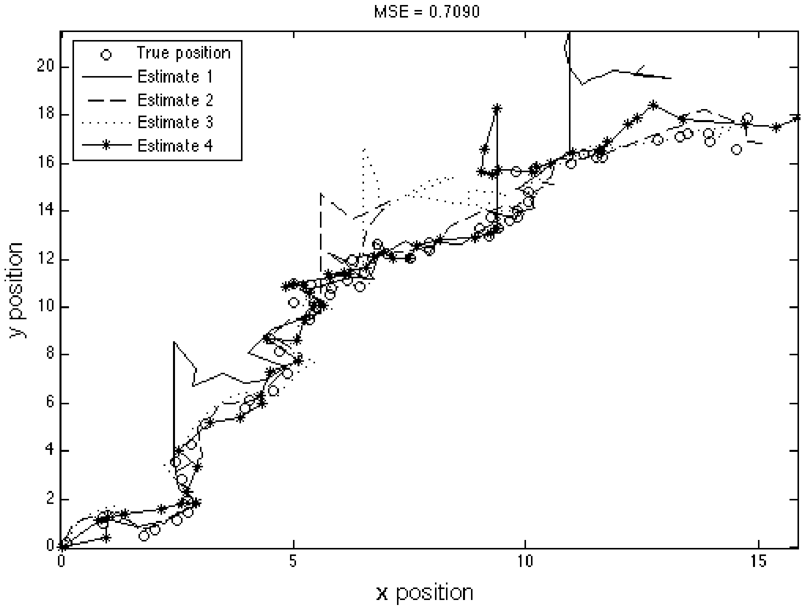


Figure 6.14: Tracking result with malicious nodes. We activate four sensor nodes, and there is one malicious node (which can sense the target) at $t = 10, 20, 30, 40, 50$, and that node remains active (malicious) for only one time step

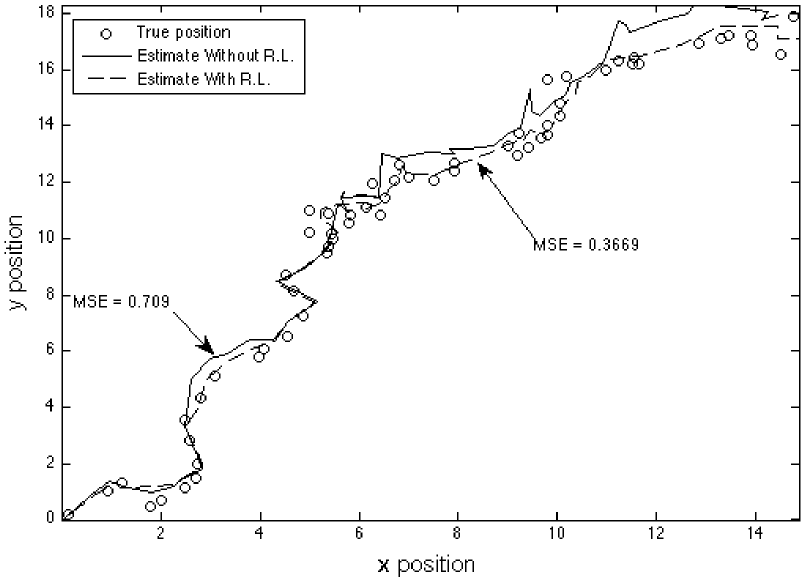


Figure 6.15: Comparison of the tracking performance. The solid line is obtained by averaging the result in Figure 6.14, and its MSE is 0.7090. The dashed line is obtained by using relaxation labeling to remove malicious nodes. Its MSE is 0.3669, which is smaller than the MSE of averaging.

and we activate five sensor nodes, averaging has relatively good results. This is because that the malicious node is a minority among the five nodes. Using relaxation labeling still provides a better MSE, 0.4081.

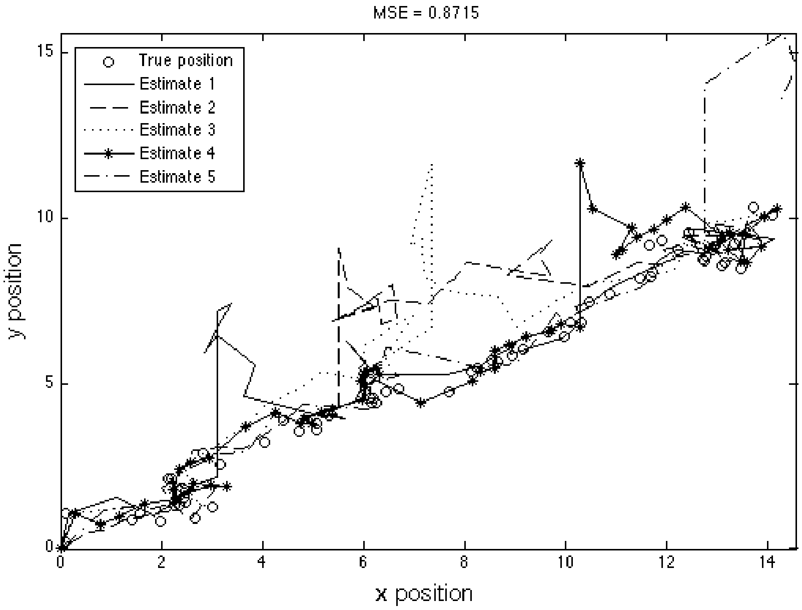


Figure 6.16: Tracking result with malicious nodes by activating five sensor nodes.

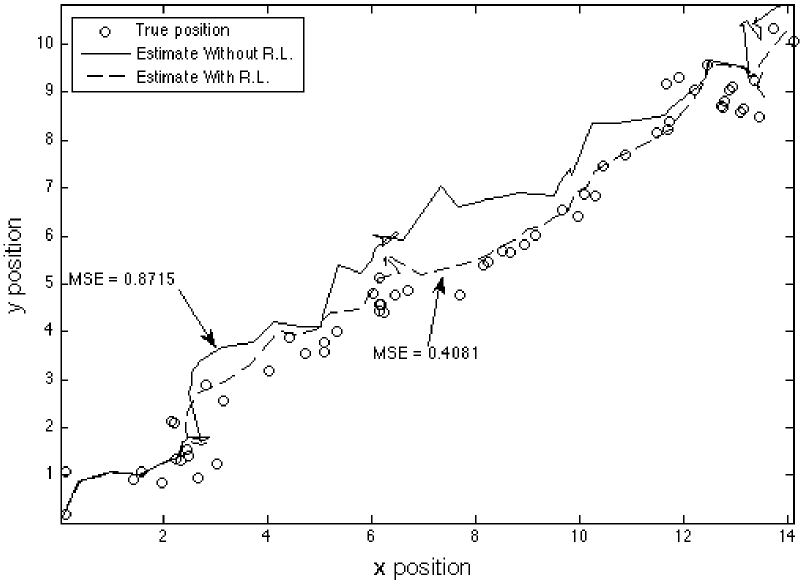


Figure 6.17: Comparison of the tracking performance for five active nodes.

As a final demonstration, we activate five nodes in Figure 6.18. In this experiment, we choose one malicious node at two consecutive time steps. That is, the malicious nodes are

1. s_1^{10} and s_2^{11}
2. s_2^{20} and s_3^{21}
3. s_3^{30} and s_4^{31}
4. s_4^{40} and s_5^{41}
5. s_5^{50} and s_1^{51}

For example, when we apply the relaxation labeling algorithm to $t = 10$ and $t = 11$, we will have one malicious node at both time steps. We further examine the probability of each node being malicious at $t = 10$ and $t = 11$ in Figure 6.19. Figure 6.19 shows that we correctly identify s_1^{10} and s_2^{11} to be malicious nodes, which is a correct result. The relaxation labeling algorithm took about 20 iterations to converge, which is very fast. There are other malicious nodes at $t = 20, 30, 40, 50$. The probabilities of being malicious for those nodes at $t = 20, 30, 40, 50$ are listed in Appendix D for the ease of reading. The final tracking result using relaxation labeling is shown in Figure 6.20. Figure 6.20 shows that detection of malicious nodes are successful for $t = 10, 20, 30, 40, 50$.

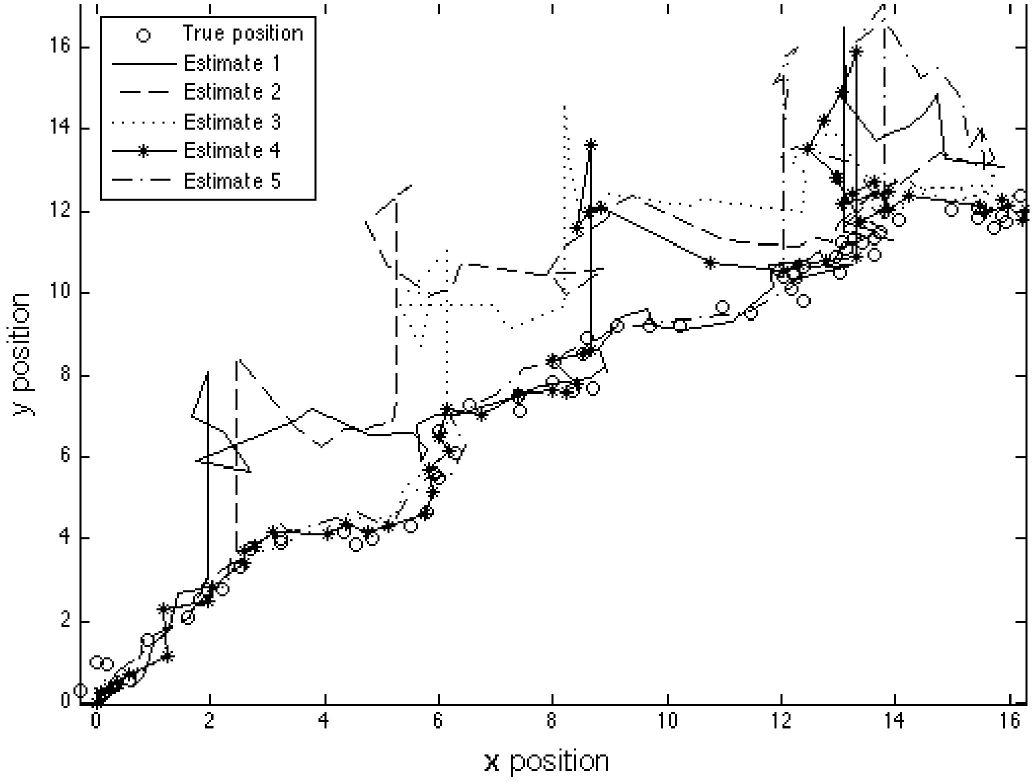


Figure 6.18: Tracking performance under the influence of malicious nodes. Note that no secure tracking algorithm is performed in this figure.

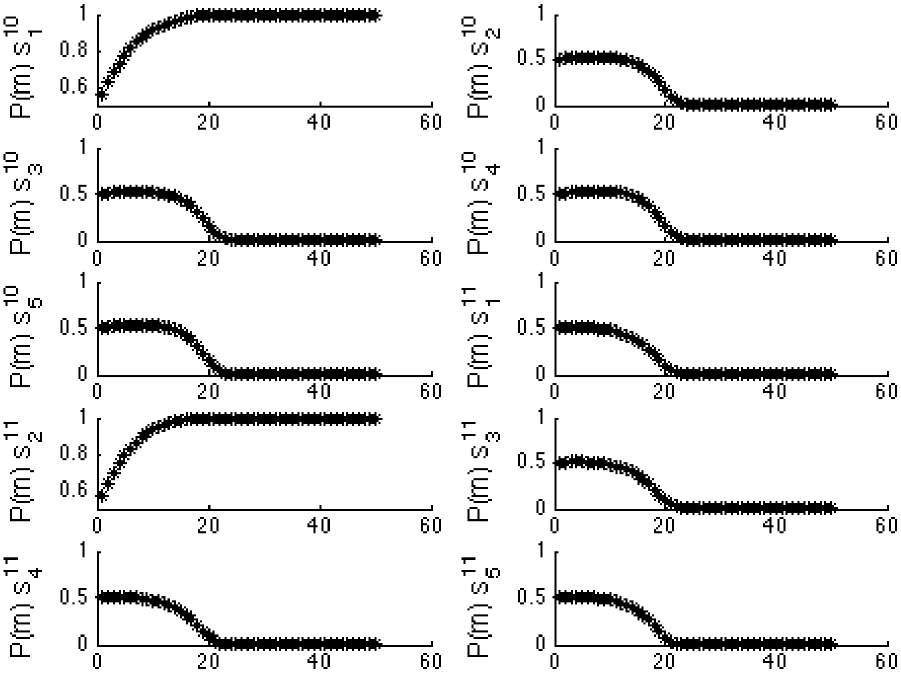


Figure 6.19: Probability of being malicious nodes for the five nodes at $t = 10$ and another five nodes at $t = 11$. Hence we have $5 \times 2 = 10$ $P(m)$ here. The first five probabilities are for $P(\lambda)$ at $t = 10$. The last five are for $t = 11$. We can see that at $t = 10$, the first node is found to be malicious, while at $t = 11$, the second node is found to be malicious. This agrees with the actual data.

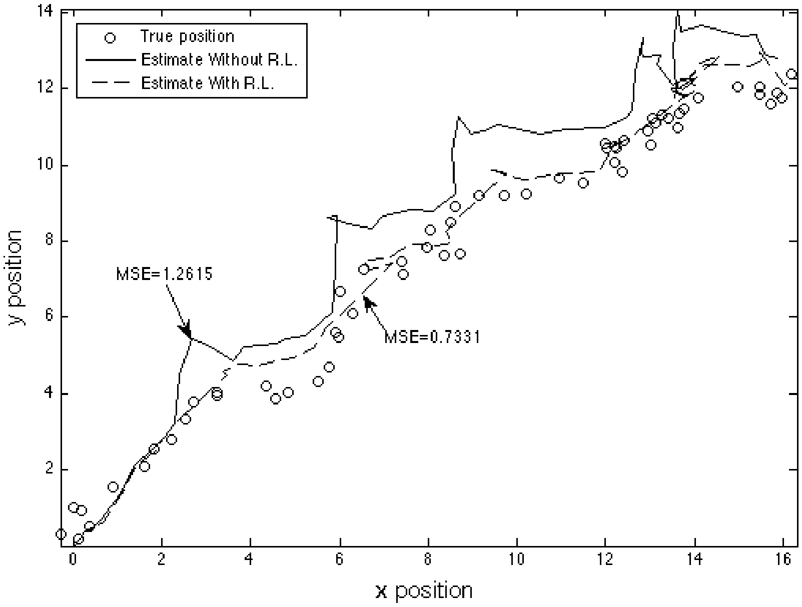


Figure 6.20: Tracking performance under the influence of malicious nodes. Two secure tracking results are shown in this figure. One is averaging (shown in solid line), and the other is relaxation labeling (shown as the dashed line). The MSE for averaging is 1.2615. The MSE for relaxation labeling is 0.7331.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this dissertation, the problems of secure localization and secure tracking for sensor networks were investigated. Malicious nodes are assumed to be colluding, and the only way to detect them is by their behaviors. A new relaxation labeling algorithm was proposed which employs higher-order compatibility functions to detect malicious nodes, and then uses the report from benign nodes to perform localization and tracking. The performance of the relaxation labeling algorithm has been demonstrated with both simulations and field experiments. The result of secure localization has also been compared with an existing algorithm based on majority voting. Our algorithm is shown to be able to detect malicious nodes, even when they are colluding. Choice of suitable parameters for the relaxation labeling algorithm was also discussed.

7.2 Future Work

In Chapter 6, we considered the target tracking problem for sensor networks. Target tracking, as explained in Chapter 3, consists of two parts: prediction and update. The update stage hinges on the measurement made by the sensor nodes - that is, we can only correctly estimate the current location of the target if sensor nodes make measurement of the target range.

A future research direction is to consider that, due to the difficulty of the terrain, some parts of our monitored area do not have sensor nodes deployed. However, the target may go through these areas. In other words, our sensor network will inevitably lose track of the target for several time steps when the target is within the difficult domain. For example, consider the scenario depicted in Figure 7.1 which has a shaded area where no sensor node can be deployed. We denote the shaded area as the *unavailable area.* Therefore, when the target passes through the unavailable area, we cannot make measurement of the range between the target and any sensor node. The problem is then to decide which sensor nodes to the right of the unavailable area (assuming that the target is traveling from left to right) should we activate in order not to lose track of the target?

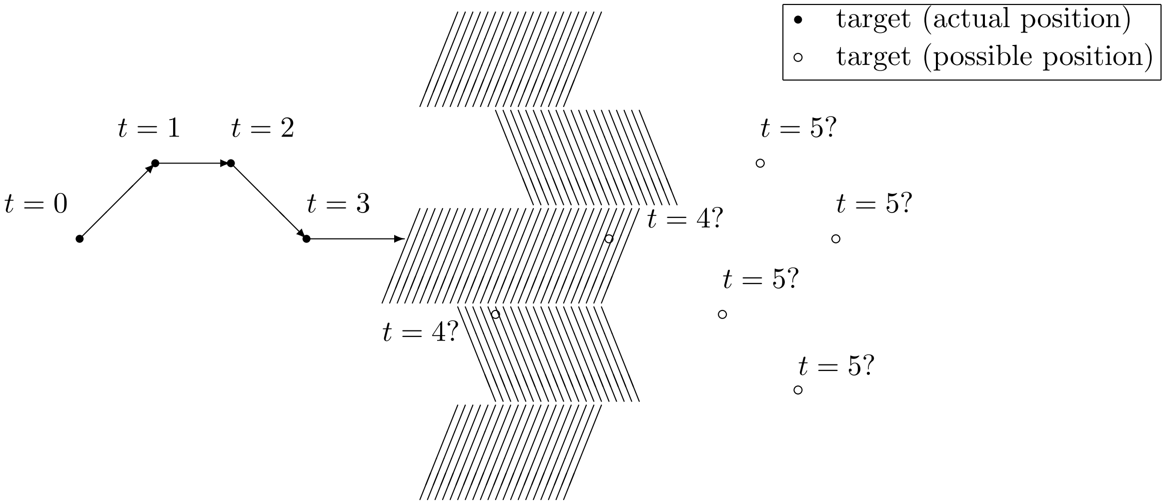


Figure 7.1: A possible scenario where a target travels through some obstacles where no sensor nodes are deployed. In this scenario, the target is traveling from the left to the right. The shaded area is where there is no sensor nodes are deployed. For example, consider that the shaded area is a river. The target is a tank that we are tracking. There are no sensor nodes deployed in the river, however, the tank can successfully pass through the river. Assume that it takes one time step for the tank to pass the river, we cannot determine the location of the tank at $t = 4$. The problem is to determine which nodes to the right of the unavailable area should we activate, at $t = 5$, in order not to miss the tank?

One possible approach to this research problem is to consider a possible radius and a possible angle for the target to move, based on the available motion model of the target. Based on the last available target position, we may predict a *candidate area* wherein the target will be, at the immediately following time step. Depending on the number of time steps that the target will not be measured, we may expand this candidate area based on the available motion model of the target. Hence the nodes within the expanded candidates area which are located beyond the unavailable area should be activated. Figure 7.2 shows an example of a fan-shaped candidate area. Based on the linear motion model of the target, we can calculate the radius of the fan-shaped area based on the speed of the target. The angle of the fan-shaped area is also determined according to the past trajectory of the target and known bounds on accelerations. Hence we may expand the candidate area for all the necessary time steps that we have no measurement of the target. The nodes to be activated will be within the expanded candidate areas.

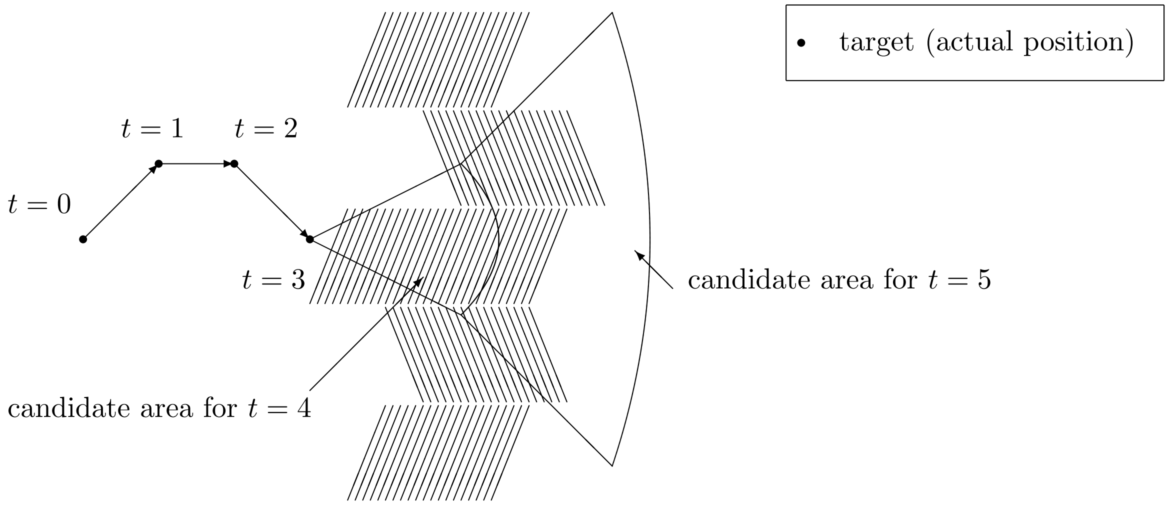

• target (actual position)
$t = 0$
$t = 1$
$t = 2$
$t = 3$
candidate area for $t = 4$
candidate area for $t = 5$

Figure 7.2: Candidate areas for node activations. The candidate area is calculated based on an estimated speed and an estimated turning angle of the target. First, the candidate area for $t = 4$ is calculated. Then the candidate area for $t = 5$ is calculated based on the candidate area for $t = 4$. Those nodes inside the candidate area for $t = 5$ will be activated to detect possible target appearances.

Obviously, the candidate area will enlarge as time progresses. If the target is undetected for several time steps (within the unavailable area), the number of nodes required to be activated will be large. Another possible research approach is to predict a maximum likelihood position of the target. In other words, we try to extend the target trajectory. For example, Jonker et al. provide an algorithm for path extension [36]. After we arrive at the maximum likelihood estimate of the target location beyond the unavailable area, we may turn on only the nodes close to the maximum likelihood target location. In this way, we do not have to activate a large number of nodes and waste precious system resources. One example is illustrated in Figure 7.3. In Figure 7.3, we try to extend the target trajectory and arrive at the maximum likelihood target location at $t = 5$. We only activate nodes within the neighborhood of the estimated target location at $t = 5$. The size of the neighborhood is increased or decreased based on our confidence of the estimated target location at $t = 5$.

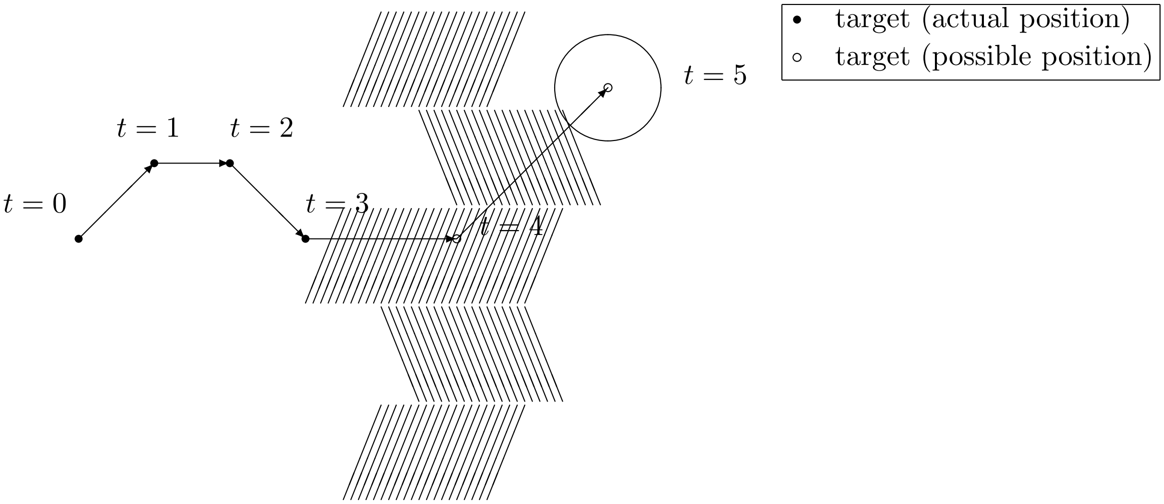


Figure 7.3: Maximum likelihood estimates of the target locations. We predict the maximum likelihood target location of the target at $t = 4$. Based on the estimated target location at $t = 4$, we predict the maximum likelihood location of the target at $t = 5$. We only activate the nodes within a certain neighborhood of the estimated target location at $t = 5$. The size of the neighborhood can be adjusted according to our confidence of the prediction of the likely target position.

Bibliography

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

[2] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174 – 188, February 2002.

[3] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111 – 122, 1981.

[4] Yaakov Bar-Shalom and Xiao-Rong Li. *Estimation and Tracking.* Artech House, 685 Canton Street, Norwood, MA 02062, 1993 1993.

[5] B. Bhanu and O. D. Faugeras. Segmentation of images having unimodal distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(4):408 – 419, July 1982.

[6] S. Capkun, L. Buttyan, and J.-P. Hubaux. Sector: secure tracking of node encounters in multi-hop wireless networks. In *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, pages 21 – 32, 2003.

[7] J. Carpenter, P. Clifford, and P. Fearnhead. Improved particle filter for nonlinear problems. *IEE Proceedings - Radar, Sonar and Navigation*, 146(1):2 – 7, February 1999.

[8] C. G. Chang, W. E. Snyder, and C. Wang. Robust localization of multiple events in sensor networks. In *IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006*, volume 1, pages 168 – 177, June 2006.

[9] C. G. Chang, W. E. Snyder, and C. Wang. A new relaxation labeling architecture for secure localization in sensor networks. In *IEEE International Conference on Communications*, pages 3076 – 3081, June 2007.

[10] C. G. Chang, W. E. Snyder, and C. Wang. Secure tracking in sensor networks. In *IEEE International Conference on Communications*, pages 3082 – 3087, June 2007.

[11] C. G. Chang, W. E. Snyder, and C. Wang. Secure target localizaiton in sensor networks using relaxation labeling. *International Journal of Sensor Networks*, 2008.

[12] J. C. Chen, K. Yao, and R. E. Hudson. Source localization and beamforming. *IEEE Signal Processing Magazine*, 19:30 – 39, 2002.

[13] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *International Journal of High Performance Computing Applications*, 16(3), Fall 2002.

[14] J. Craig. *Introduction to Robotics: Mechanics and Control*. Prentice Hall, 3 edition, October 2003.

[15] D. Crisan, P. Del Moral, and T. J. Lyons. Non-linear filtering using branching and interacting particle systems. *Markov Processes Related Fields*, 5(3):293 – 319, 1999.

[16] D. Culler, D. Estrin, and M. Srivastava, editors. *Special Issue on Sensor Networks*, volume 37 of *Computer*. IEEE Computer Society, August 2004.

[17] J. Deng, R. Han, and S. Mishra. Sensor networks: Defending against path-based dos attacks in wireless sensor networks. In *the 3rd ACM workshop on Security of ad hoc and sensor networks*, 2005.

[18] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential monte carlo methods in practice*. Springer-Verlaag New York, Inc, 2001.

[19] R. O. Duda and P. E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15:11 – 15, January 1972.

[20] D. E. Dudgeon and R. M. Mersereau. *Multidimensional digital signal processing*. Prentice-Hall, Inc., 1984.

[21] J.-O. Eklundh and A. Rosenfeld. Some relaxation experiments using triples of pixels. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-10(3):150 – 153, 1980.

[22] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 41 – 47, 2002.

[23] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the physical world with pervasive networks. *IEEE Pervasive Computing*, 1(1):59 – 69, Jan - March 2002.

[24] O. D. Faugeras and M. Berthod. Improving consistency and reducing ambiguity in stochastic labeling: an optimization approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(4):412 – 424, July 1981.

[25] A. M. Finch, R. C. Wilson, and E. R. Hancock. Matching delaunay triangulations by probabilistic relaxation. In V. Hlavac and R. Sara, editors, *Proceedings of the Sixth International Conference on Computer Anaylsis of Images and Patterns*, volume 970/1995, pages 350 – 358, 1995.

[26] N. Gordon, D. Salmond, and A. F. M. Smith. Novel approach to nonlinear and non-gaussian bayesian state estimation. *IEE Proceedings-F (Radar and Signal Processing)*, 140(2):107 – 113, April 1993.

[27] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, G. Zhou, J. Hui, and B. Krogh. Vigilnet: an integrated sensor network system for energy-efficient surveillance. *ACM Transactions on Sensor Networks*, 2(1):1 – 38, February 2006.

[28] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy. The platforms enabling wireless sensor networks. *Communications of the ACM*, 47(6):41 – 46, 2004.

[29] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architecture Support for Programming Languages and Operating Systems*, pages 93 – 104, November 2000.

[30] Paul V. C. Hough. Method and means for recognizing complex patterns. U.S. Patent 3069654, March 25 1960.

[31] R. A. Hummel and S. W. Zucker. On the foundations of relaxation labeling processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):267 – 287, May 1983.

[32] A. T. Ihler, J. W. Fisher III, R. L. Moses, and A. S. Willsky. Nonparametric belief propagation for self-localization of sensor networks nonparametric belief propagation for self-localization of sensor networks. *IEEE Journal on Selected Areas in Communications*, 23(4):809 – 819, April 2005.

[33] M. Ilyas and I. Mahgoub, editors. *Handbook of sensor networks: compact wireless and wired sensing systems*. CRC Press LLC, 2005.

[34] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5 – 28, August 1998.

[35] A. H. Jazwinski. *Stochastic processes and filtering theory*. New York: Academic, 1970.

[36] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325 – 340, December 1987.

[37] D. L. Stephens Jr. and A. J. Peurrung. Detection of moving radioactive sources using sensor networks. *IEEE Transactions on Nuclear Science*, 51(5):2273 – 2278, October 2004.

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

[39] R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Transactions of the ASME - Journal of Basic Engineering*, 83:95 – 107, 1961.

[40] L. E. Kinsler, A. R. Frey, A. B. Coppens, and J.V. Sanders. *Fundamentals of Acoustics*. John Wiley and Sons, Inc., 1999.

[41] J. Kittler and E. R. Hancock. Contextual decision rule for region analysis. *Image and Vision Computing*, 5(2):145 – 153, May 1987.

[42] J. Kittler and J. Illingworth. Relaxation labeling algorithms - a review. *Image and Vision Computing*, 3(4):206 – 216, 1985.

[43] L. Lazos and R. Poovendran. Serloc: Robust localization for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 1(1):73 – 100, August 2005.

[44] D. Li, K. D. Wong, Y. H. Hu, and A. M. Sayeed. Detection, classification, and tracking of targets. *IEEE Signal Processing Magazine*, pages 17 – 29, March 2002.

[45] X. R. Li and V. P. Jilkov. Survey of maneuvering target tracking: dynamic models. In Oliver E. Drummond, editor, *Proceedings of SPIE Conference on Signal and Data Pocessing of Small Targets*, volume 4048, pages 212 – 235, July 2000.

[46] D. Liu, P. Ning, and W. K. Du. Attack-resistant location estimation in sensor networks. In *Fourth International Symposium on Information Processing in Sensor Networks (IPSN2005)*, pages 99 – 106, April 15 2005.

[47] J. Liu, J. Reich, and F. Zhao. Collaborative in-network processing for target tracking. *EURASIP Journal on Applied Signal Processing*, 4:378 – 391, March 2003.

[48] J. S. Liu and R. Chen. Sequential monte carlo methods for dynamical systems. *Journal of The American Statistical Association*, 93:1032 – 1044, 1998.

[49] S. A. Lloyd. An optimization approach to relaxation labelling algorithms. *Image and Vision Computing*, 1(2):85 – 91, May 1983.

[50] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51 – 58, May 2000.

[51] A. Rosenfeld, R. A. Hummel, and S. W. Zucker. Scene labeling by relaxation operations. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-6(6):420 – 433, June 1976.

[52] A. Rosenfeld and A. Kak. *Digital Picture Processing.* Academic Press, 2 edition, 1982.

[53] A. H. Sayed, A. Tarighat, and N. Khajehnouri. Network-based wireless location: challenges faced in developing techniques for accurate wireless location information. *IEEE Signal Processing Magazine*, 22(4):24 – 40, July 2005.

[54] M. Srivastava, R. Muntz, and M. Potkonjak. Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving enviroments. In *the 7th annual*

international conference on Mobile computing and networking, pages 132 – 138. ACM Press, 2001.

[55] H. Stark and J. W. Woods. *Probability and random processes with applications to signal processing.* Prenteice Hall, third edition, 2002.

[56] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Wireless sensor networks: Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34 – 40, June 2004.

[57] D. H. von Seggern. *CRC Standard Curves and Surfaces.* CRC Press LLC, 1992.

[58] B. Warneke, M. Last, B. Liebowitz, and K. Pister. Smart dust: communicating with a cubic-millimeter computer. *Computer*, 34(1):44 – 51, January 2001.

[59] B. A. Warneke, M. D. Scott, B. S. Leibowitz, L. Zhou, C. L. Bellew, J. A. Chediak, J. M. Kahn, B. E. Boser, and K. S. J. Pister. An autonomous 16 mm3 solar-powered node for distributed wireless sensor networks. *Proceedings of IEEE Sensors*, 2:12 – 14, June 2002.

[60] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical en-route filtering of injected false data in sensor networks. *IEEE Journal on Selected Areas in Communications*, 23(4):839 – 850, April 2005.

[61] F. Zhao and L. J. Guibas. *Wireless sensor networks: an information processing approach.* Morgan Kaufmann Publishers, 2004.

[62] Y. Zhao. *Vehicle location and navigation systems.* Artech House, 1997.

[63] Y. Zhao. Standardization of mobile phone positioning for 3g systems. *IEEE Communications Magazine*, 40(7):108 – 116, July 2002.

[64] S. K. Zhou, R. Chellappa, and B. Moghaddam. Visual tracking and recognition using appearance-adaptive models in particle filters. *IEEE Transactions on Image Processing*, 13(11):1491 – 1506, November 2004.

[65] S. Zhu, S. Setia, S. Jajodia, and P. Ning. An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In *IEEE Symposium on Security and Privacy*, 2004.

[66] S. W. Zucker, E. V. Krishnamurthy, and R. L. Haar. Relaxation processes for scene labeling: convergence, speed and stability. *IEEE Transactions on Systems, Man and Cybernetics*, 8(1):41 – 48, 1978.

[67] S. W. Zucker, Y. G. Leclerc, and J. L. Mohammed. Continuous relaxation and local maxima selection: conditions for equivalence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(2):117 – 127, 1981.

Appendices

Appendix A

Derivation of the Kalman Filter

In this section we derive the Kalman filter[1] [38, 39, 4]. The motion model of the target is

$$\mathbf{x}^k = \mathbf{F}^k \mathbf{x}^{k-1} + \mathbf{v}^{k-1} \tag{A.1}$$

where $\mathbf{F}^k$ is a known system matrix. Meanwhile, the measurement model is

$$\mathbf{z}^k = \mathbf{H}^k \mathbf{x}^k + \mathbf{w}^k \tag{A.2}$$

where $\mathbf{H}^k$ is also a known measurement matrix. $\mathbf{v}^{k-1}$ and $\mathbf{w}^k$ are zero-mean, statistically independent, and have covariances of $\mathbf{Q}^{k-1}$ and $\mathbf{R}^k$, respectively. That is

$$\mathbf{Q}^{k-1} = E\left[\mathbf{v}^{k-1}\left(\mathbf{v}^{k-1}\right)^T\right] \tag{A.3}$$

$$\mathbf{R}^k = E\left[\mathbf{w}^k\left(\mathbf{w}^k\right)^T\right] \tag{A.4}$$

We begin with the given pdf $p(\mathbf{x}^{k-1}|\mathbf{z}^{k-1})$, which is assumed Gaussian, i.e.

$$p(\mathbf{x}^{k-1}|\mathbf{z}^{1:k-1}) = \mathcal{N}(\mathbf{x}^{k-1}; \mathbf{m}^{k-1|k-1}, \mathbf{P}^{k-1|k-1}) \tag{A.5}$$

which is to say, we can parameterize $p(\mathbf{x}^{k-1}|\mathbf{z}^{k-1})$ using

$$\mathbf{m}^{k-1|k-1} = E\left[\mathbf{x}^{k-1}\right] \tag{A.6}$$

[1] Appendix A derived by C.-C. G. Chang, using notation from [2] and the approach described in [4]

and covariance

$$\mathbf{P}^{k-1|k-1} = E\left[\left(\mathbf{x}^{k-1} - \mathbf{m}^{k-1|k-1}\right)\left(\mathbf{x}^{k-1} - \mathbf{m}^{k-1|k-1}\right)^T\right] \tag{A.7}$$

Next, we calculate the intermediate pdf $p(\mathbf{x}^k|\mathbf{z}^{k-1})$ based on $p(\mathbf{x}^{k-1}|\mathbf{z}^{k-1})$. First, Kalman proposes to predict the state estimates at $t = k$ using

$$\mathbf{m}^{k|k-1} = E\left[\mathbf{x}^k|\mathbf{z}^{k-1}\right] = \mathbf{F}^k\mathbf{m}^{k-1|k-1} \tag{A.8}$$

On the other hand, the transition of the true states, $\mathbf{x}^k$, is $\mathbf{x}^k = \mathbf{F}^k\mathbf{x}^{k-1} + \mathbf{v}^{k-1}$. Therefore, we can calculate the error between the state estimates and true states using

$$\begin{aligned} e^{k|k-1} &= \mathbf{x}^k - \mathbf{m}^{k|k-1} && \text{(A.9)} \\ &= (\mathbf{F}^k\mathbf{x}^{k-1} + \mathbf{v}^{k-1}) - \mathbf{F}^k\mathbf{m}^{k-1|k-1} && \text{(A.10)} \\ &= \mathbf{F}^k e^{k-1} + \mathbf{v}^{k-1} && \text{(A.11)} \end{aligned}$$

Hence for $\mathbf{P}^{k|k-1}$, we have

$$\begin{aligned} \mathbf{P}^{k|k-1} &= E\left[e^{k|k-1}\left(e^{k|k-1}\right)\right] && \text{(A.12)} \\ &= E\left[\left(\mathbf{F}^k e^{k-1} + \mathbf{v}^{k-1}\right)\left(\mathbf{F}^k e^{k-1} + \mathbf{v}^{k-1}\right)^T\right] && \text{(A.13)} \\ &= E\left[\mathbf{F}^k e^{k-1}\left(\mathbf{F}^k e^{k-1}\right)^T\right] + E\left[\mathbf{v}^{k-1}\left(\mathbf{v}^{k-1}\right)^T\right] && \text{(A.14)} \\ &= \mathbf{F}^k\mathbf{P}^{k-1|k-1}\left(\mathbf{F}^k\right)^T + \mathbf{Q}^{k-1} && \text{(A.15)} \end{aligned}$$

The intermediate pdf $p(\mathbf{x}^k|\mathbf{z}^{k-1})$ is another Gaussian

$$p(\mathbf{x}^k|\mathbf{z}^{1:k-1}) = \mathcal{N}(\mathbf{x}^k; \mathbf{m}^{k|k-1}, \mathbf{P}^{k|k-1}) \tag{A.16}$$

Next, we will calculate the desired pdf $p(\mathbf{x}^k|\mathbf{z}^k)$ using the intermediate pdf in (A.16). We will soon derive that the desired pdf $p(\mathbf{x}^k|\mathbf{z}^k)$ is also Gaussian

$$p(\mathbf{x}^k|\mathbf{z}^{1:k}) = \mathcal{N}(\mathbf{x}^k; \mathbf{m}^{k|k}, \mathbf{P}^{k|k}) \tag{A.17}$$

whose parameters are calculated using

$$\mathbf{m}^{k|k} = \mathbf{m}^{k|k-1} + \mathbf{K}^k(\mathbf{z}^k - \mathbf{H}^k\mathbf{m}^{k|k-1}) \tag{A.18}$$

$$\mathbf{P}^{k|k} = \mathbf{P}^{k|k-1} - \mathbf{K}^k \mathbf{H}^k \mathbf{P}^{k|k-1} \tag{A.19}$$

and where

$$\mathbf{S}^k = \mathbf{H}^k \mathbf{P}^{k|k-1} \left(\mathbf{H}^k\right)^T + \mathbf{R}^k \tag{A.20}$$

is the covariance of the innovation term $\mathbf{z}^k - \mathbf{H}^k \mathbf{m}^{k|k-1}$, and

$$\mathbf{K}^k = \mathbf{P}^{k|k-1} \left(\mathbf{H}^k\right)^T \mathbf{S}^{k^{-1}} \tag{A.21}$$

is the Kalman gain.

The derivation begins with calculating $\mathbf{m}^{k|k}$ using $\mathbf{m}^{k|k-1}$. Kalman proposes that the measurement prediction is $\hat{\mathbf{z}}^k = \mathbf{H}^k \mathbf{m}^{k|k-1}$, so the measurement residual (also referred to as the innovation term) is

$$\mathbf{z}^k - \hat{\mathbf{z}}^k = \left(\mathbf{z}^k - \mathbf{H}^k \mathbf{m}^{k|k-1}\right) \tag{A.22}$$

Kalman proposes to update the state predictions $\mathbf{m}^{k|k}$ using the intermediate state predictions, $\mathbf{m}^{k|k-1}$, as

$$\mathbf{m}^{k|k} = \mathbf{m}^{k|k-1} + \mathbf{K}^k \left(\mathbf{z}^k - \mathbf{H}^k \mathbf{m}^{k|k-1}\right) \tag{A.23}$$

where $\mathbf{K}^k$ is the Kalman gain. Substituting the measurement model in (A.2) into (A.23), we have

$$\begin{aligned}\mathbf{m}^{k|k} &= \mathbf{m}^{k|k-1} + \mathbf{K}^k \left(\mathbf{z}^k - \mathbf{H}^k \mathbf{m}^{k|k-1}\right) \\ &= \mathbf{m}^{k|k-1} + \mathbf{K}^k \left(\mathbf{H}^k \mathbf{x}^k + \mathbf{w}^k - \mathbf{H}^k \mathbf{m}^{k|k-1}\right) \qquad \text{(A.24)}\end{aligned}$$

To calculate the covariance of the error term $\mathbf{P}^{k|k}$, we have

$$\begin{aligned}\mathbf{P}^{k|k} &= E\left[e^k \left(e^k\right)^T\right] \qquad \text{(A.25)} \\ &= E\left[\left(\mathbf{x}^k - \mathbf{m}^{k|k}\right)\left(\mathbf{x}^k - \mathbf{m}^{k|k}\right)^T\right] \qquad \text{(A.26)}\end{aligned}$$

Substituting (A.24) into (A.26), we have

$$\begin{aligned}
\mathbf{P}^{k|k} &= E\left[\left(\mathbf{x}^k - \mathbf{m}^{k|k}\right)\left(\mathbf{x}^k - \mathbf{m}^{k|k}\right)^T\right] \\
&= E\Big\{ \left[(I - \mathbf{K}^k\mathbf{H}^k)(\mathbf{x}^k - \mathbf{m}^{k|k-1}) - \mathbf{K}^k\mathbf{w}^k\right] \\
&\qquad \left[(I - \mathbf{K}^k\mathbf{H}^k)(\mathbf{x}^k - \mathbf{m}^{k|k-1}) - \mathbf{K}^k\mathbf{w}^k\right]^T \Big\} \qquad (A.27)
\end{aligned}$$

The error term $\mathbf{x}^k - \mathbf{m}^{k|k-1}$ is uncorrelated with the measurement noise $\mathbf{w}^k$, so we have

$$\begin{aligned}
\mathbf{P}^{k|k} &= E\Big\{ \left[(I - \mathbf{K}^k\mathbf{H}^k)(\mathbf{x}^k - \mathbf{m}^{k|k-1}) - \mathbf{K}^k\mathbf{w}^k\right] \\
&\qquad \left[(I - \mathbf{K}^k\mathbf{H}^k)(\mathbf{x}^k - \mathbf{m}^{k|k-1}) - \mathbf{K}^k\mathbf{w}^k\right]^T \Big\} \\
&= (I - \mathbf{K}^k\mathbf{H}^k)E\left[(\mathbf{x}^k - \mathbf{m}^{k|k-1})(\mathbf{x}^k - \mathbf{m}^{k|k-1})^T\right](I - \mathbf{K}^k\mathbf{H}^k)^T \\
&+ \ \mathbf{K}^k E\left[\mathbf{w}^k\left(\mathbf{w}^k\right)^T\right]\left(\mathbf{K}^k\right)^T \qquad (A.28) \\
&= (I - \mathbf{K}^k\mathbf{H}^k)\mathbf{P}^{k|k-1}(I - \mathbf{K}^k\mathbf{H}^k)^T + \quad \mathbf{K}^k R\left(\mathbf{K}^k\right)^T \qquad (A.29) \\
&= \mathbf{P}^{k|k-1} - \mathbf{K}^k\mathbf{H}^k\mathbf{P}^{k|k-1} - \mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T\left(\mathbf{K}^k\right)^T \\
&\quad + \mathbf{K}^k\left(\mathbf{H}^k\mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T + R\right)\left(\mathbf{K}^k\right)^T \qquad (A.30)
\end{aligned}$$

The sum of the diagonal elements of a matrix is the *trace* of a matrix. In the case of a error covariance matrix in (A.30), the trace is the sum of the mean squared errors. The Kalman filter is a *Minimum Mean Square Error* (MMSE) estimator. The mean squared error may be minimized by minimizing the trace of $\mathbf{P}^{k|k}$, and by minimizing the trace of $\mathbf{P}^{k|k}$ we can obtain the desired $\mathbf{K}^k$.

We differentiate $\mathbf{P}^{k|k}$ with respect to $\mathbf{K}^k$ in order to find the conditions of this minimum. Note that the trace of a matrix is equal to the trace of its transpose, hence we have

$$Tr[\mathbf{P}^{k|k}] = Tr[\mathbf{P}^{k|k-1}] - 2Tr[\mathbf{K}^k\mathbf{H}^k\mathbf{P}^{k|k-1}] + Tr\left[\mathbf{K}^k\left(\mathbf{H}^k\mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T + \mathbf{R}\right)\left(\mathbf{K}^k\right)^T\right] \qquad (A.31)$$

where $Tr[\cdot]$ is the trace of a matrix.

Differentiating with respect to $\mathbf{K}^k$ gives

$$\begin{aligned}
\frac{Tr[\mathbf{P}^{k|k}]}{d\mathbf{K}^k} &= -2\left(\mathbf{H}^k\mathbf{P}^{k|k-1}\right)^T + 2\mathbf{K}^k\left(\mathbf{H}^k\mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T + \mathbf{R}\right) && \text{(A.32)}\\
&= 0 && \text{(A.33)}
\end{aligned}$$

we have

$$\left(\mathbf{H}^k\mathbf{P}^{k|k-1}\right)^T = \mathbf{K}^k\left(\mathbf{H}^k\mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T + \mathbf{R}\right) \qquad \text{(A.34)}$$

Solving for $\mathbf{K}^k$, we have

$$\begin{aligned}
\mathbf{K}^k &= \mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T\left(\mathbf{H}^k\mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T + \mathbf{R}\right)^{-1} && \text{(A.35)}\\
&= \mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T\left(\mathbf{S}^k\right)^{-1} && \text{(A.36)}
\end{aligned}$$

(A.36) is the Kalman gain equation as previously defined in (A.21). Also, the definition of the covariance of the innovation term in (A.37) comes from (A.36)

$$\mathbf{S}^k = \mathbf{H}^k\mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T + \mathbf{R}^k \qquad \text{(A.37)}$$

Substituting (A.36) into (A.30), we obtain

$$\begin{aligned}
\mathbf{P}^{k|k} &= \mathbf{P}^{k|k-1} - \mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T\left(\mathbf{H}^k\mathbf{P}^{k|k-1}\left(\mathbf{H}^k\right)^T + \mathbf{R}\right)^{-1}\mathbf{H}^k\mathbf{P}^{k|k-1} && \text{(A.38)}\\
&= \mathbf{P}^{k|k-1} - \mathbf{K}^k\mathbf{H}^k\mathbf{P}^{k|k-1} && \text{(A.39)}
\end{aligned}$$

(A.39) proves the error covariance matrix update equation given in (A.19). We illustrate the workflow of the Kalman filter estimation process in Figure A.1.

Example. Consider an illustrative example in which the target state consists only of the target position in the two-dimensional space

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \text{(A.40)}$$

and our motion model for the target is linear

$$\mathbf{x}^{k+1} = \mathbf{F}\mathbf{x}^k + \mathbf{v}^k = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \mathbf{v}^k \qquad \text{(A.41)}$$

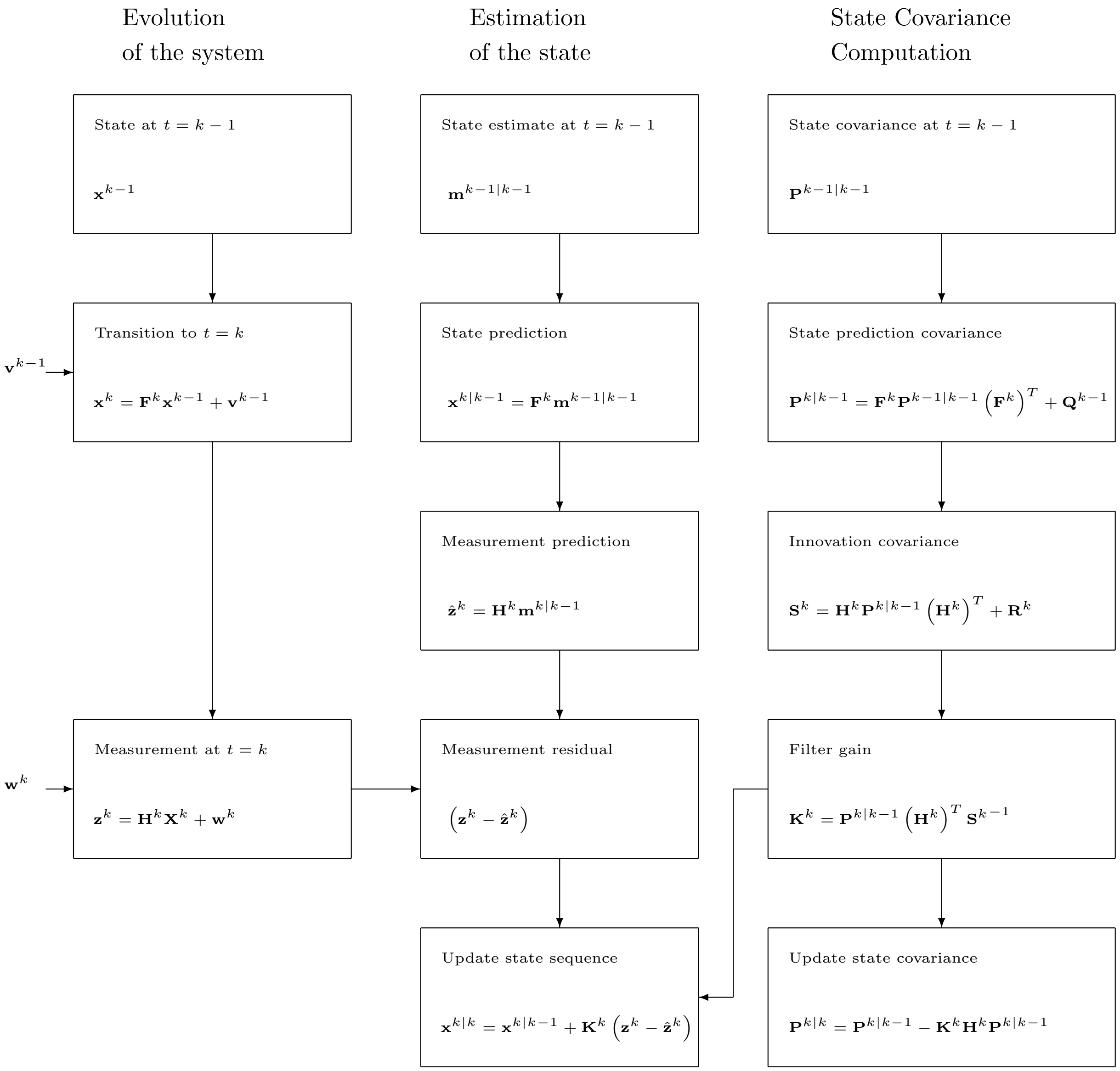

Evolution of the system
Estimation of the state
State Covariance Computation
State at $t = k - 1$
$\mathbf{x}^{k-1}$
State estimate at $t = k - 1$
$\mathbf{m}^{k-1|k-1}$
State covariance at $t = k - 1$
$\mathbf{P}^{k-1|k-1}$
$\mathbf{v}^{k-1}$
Transition to $t = k$
$\mathbf{x}^{k} = \mathbf{F}^{k}\mathbf{x}^{k-1} + \mathbf{v}^{k-1}$
State prediction
$\mathbf{x}^{k|k-1} = \mathbf{F}^{k}\mathbf{m}^{k-1|k-1}$
State prediction covariance
$\mathbf{P}^{k|k-1} = \mathbf{F}^{k}\mathbf{P}^{k-1|k-1}\left(\mathbf{F}^{k}\right)^{T} + \mathbf{Q}^{k-1}$
Measurement prediction
$\hat{\mathbf{z}}^{k} = \mathbf{H}^{k}\mathbf{m}^{k|k-1}$
Innovation covariance
$\mathbf{S}^{k} = \mathbf{H}^{k}\mathbf{P}^{k|k-1}\left(\mathbf{H}^{k}\right)^{T} + \mathbf{R}^{k}$
$\mathbf{w}^{k}$
Measurement at $t = k$
$\mathbf{z}^{k} = \mathbf{H}^{k}\mathbf{X}^{k} + \mathbf{w}^{k}$
Measurement residual
$\left(\mathbf{z}^{k} - \hat{\mathbf{z}}^{k}\right)$
Filter gain
$\mathbf{K}^{k} = \mathbf{P}^{k|k-1}\left(\mathbf{H}^{k}\right)^{T}\mathbf{S}^{k-1}$
Update state sequence
$\mathbf{x}^{k|k} = \mathbf{x}^{k|k-1} + \mathbf{K}^{k}\left(\mathbf{z}^{k} - \hat{\mathbf{z}}^{k}\right)$
Update state covariance
$\mathbf{P}^{k|k} = \mathbf{P}^{k|k-1} - \mathbf{K}^{k}\mathbf{H}^{k}\mathbf{P}^{k|k-1}$

Figure A.1: The workflow of the Kalman filter estimation(reproduced from [4])

The noise $\mathbf{v}^k$ is zero-mean, Gaussian

$$E[\mathbf{v}^k] = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad E[\mathbf{v}^k \left(\mathbf{v}^k\right)^T] = \mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.42}$$

For the measurement model, we have

$$\mathbf{z}^k = \mathbf{H}\mathbf{x}^k + \mathbf{w}^k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \mathbf{w}^k \tag{A.43}$$

which is again linear. The noise in the measurement model is also zero-mean, Gaussian

$$E[\mathbf{w}^k] = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad E[\mathbf{w}^k \left(\mathbf{w}^k\right)^T] = \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.44}$$

To initialize the system, we choose

$$\mathbf{x}^0 = \begin{bmatrix} 5 \\ 7 \end{bmatrix} \qquad \mathbf{m}^{0|0} = \begin{bmatrix} 5 \\ 7 \end{bmatrix} \qquad \mathbf{P}^{0|0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{A.45}$$

Using those initial parameters, we can simulate the target moment by generating random noise sequences. In this example, we show the tracking of the target using Kalman filter for 10 iterations. The first random noise that we generate is

$$\mathbf{v}^0 = \begin{bmatrix} -0.4326 \\ -1.6656 \end{bmatrix} \tag{A.46}$$

Hence we can calculate the first location of the target as

$$\mathbf{x}^1 = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 7 \end{bmatrix} + \begin{bmatrix} -0.4326 \\ -1.6656 \end{bmatrix} = \begin{bmatrix} 8.0674 \\ 5.3344 \end{bmatrix} \tag{A.47}$$

Next, we generate the first random measurement noise as

$$\mathbf{w}^1 = \begin{bmatrix} 1.1892 \\ -0.0376 \end{bmatrix} \tag{A.48}$$

Hence we can calculate the first measurement value as

$$\mathbf{z}^1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 8.0674 \\ 5.3344 \end{bmatrix} + \begin{bmatrix} 1.1892 \\ -0.0376 \end{bmatrix} = \begin{bmatrix} 9.2566 \\ 5.2968 \end{bmatrix} \tag{A.49}$$

If we continue this process, we can simulate more samples of the noise sequences as

$$\mathbf{v}^1 = \begin{bmatrix} -1.1465 \\ 1.1909 \end{bmatrix} \mathbf{v}^2 = \begin{bmatrix} 0.3273 \\ 0.1746 \end{bmatrix} \mathbf{v}^3 = \begin{bmatrix} -0.5883 \\ 2.1832 \end{bmatrix} \mathbf{v}^4 = \begin{bmatrix} 1.0668 \\ 0.0593 \end{bmatrix} \mathbf{v}^5 = \begin{bmatrix} 0.2944 \\ -1.3362 \end{bmatrix}$$

$$\mathbf{v}^6 = \begin{bmatrix} -0.6918 \\ 0.8580 \end{bmatrix} \mathbf{v}^7 = \begin{bmatrix} -1.4410 \\ 0.5711 \end{bmatrix} \mathbf{v}^8 = \begin{bmatrix} 0.8156 \\ 0.7119 \end{bmatrix} \mathbf{v}^9 = \begin{bmatrix} 1.1908 \\ -1.2025 \end{bmatrix} \mathbf{v}^{10} = \begin{bmatrix} -1.6041 \\ 0.2573 \end{bmatrix}$$

Similarly, repeat the process of generating measurement noise

$$\mathbf{w}^2 = \begin{bmatrix} -0.1867 \\ 0.7258 \end{bmatrix} \mathbf{w}^3 = \begin{bmatrix} -0.1364 \\ 0.1139 \end{bmatrix} \mathbf{w}^4 = \begin{bmatrix} -0.0956 \\ -0.8323 \end{bmatrix} \mathbf{w}^5 = \begin{bmatrix} 0.7143 \\ 1.6236 \end{bmatrix} \mathbf{w}^6 = \begin{bmatrix} 1.2540 \\ -1.5937 \end{bmatrix}$$

$$\mathbf{w}^7 = \begin{bmatrix} -0.3999 \\ 0.6900 \end{bmatrix} \mathbf{w}^8 = \begin{bmatrix} 1.2912 \\ 0.6686 \end{bmatrix} \mathbf{w}^9 = \begin{bmatrix} -0.0198 \\ -0.1567 \end{bmatrix} \mathbf{w}^{10} = \begin{bmatrix} -1.0565 \\ 1.4151 \end{bmatrix}$$

Using these noise sequences, we can simulate the evolution of the state sequence as

$$\mathbf{x}^2 = \begin{bmatrix} 9.5882 \\ 6.5253 \end{bmatrix} \mathbf{x}^3 = \begin{bmatrix} 13.1781 \\ 6.7 \end{bmatrix} \mathbf{x}^4 = \begin{bmatrix} 15.9398 \\ 8.8832 \end{bmatrix} \mathbf{x}^5 = \begin{bmatrix} 21.4481 \\ 8.9424 \end{bmatrix} \mathbf{x}^6 = \begin{bmatrix} 26.2138 \\ 7.6063 \end{bmatrix}$$

$$\mathbf{x}^7 = \begin{bmatrix} 29.3251 \\ 8.4643 \end{bmatrix} \mathbf{x}^8 = \begin{bmatrix} 32.1163 \\ 9.0354 \end{bmatrix} \mathbf{x}^9 = \begin{bmatrix} 37.4496 \\ 9.7473 \end{bmatrix} \mathbf{x}^{10} = \begin{bmatrix} 43.5141 \\ 8.5449 \end{bmatrix} \tag{A.50}$$

Next, we repeat the process of generating measurements using (A.43), and we have

$$\mathbf{z}^2 = \begin{bmatrix} 9.4015 \\ 7.2511 \end{bmatrix} \mathbf{z}^3 = \begin{bmatrix} 13.0417 \\ 6.8139 \end{bmatrix} \mathbf{z}^4 = \begin{bmatrix} 15.8441 \\ 8.0508 \end{bmatrix} \mathbf{z}^5 = \begin{bmatrix} 22.1625 \\ 10.5660 \end{bmatrix} \mathbf{z}^6 = \begin{bmatrix} 27.4678 \\ 6.0125 \end{bmatrix}$$

$$\mathbf{z}^7 = \begin{bmatrix} 28.9252 \\ 9.1542 \end{bmatrix} \mathbf{z}^8 = \begin{bmatrix} 33.4065 \\ 9.7040 \end{bmatrix} \mathbf{z}^9 = \begin{bmatrix} 37.4298 \\ 9.5906 \end{bmatrix} \mathbf{z}^{10} = \begin{bmatrix} 42.4576 \\ 9.96 \end{bmatrix} \tag{A.51}$$

At this point we have built a system in which the system is moving according to the state sequences in (A.50) and the sensor measurements are in (A.51).

Now we begin to perform target tracking using the Kalman filter. First, using (A.8), we update the intermediate $\mathbf{m}^{1|0}$ as

$$\mathbf{m}^{1|0} = \mathbf{F}\mathbf{m}^{0|0} = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 7 \end{bmatrix} = \begin{bmatrix} 8.5 \\ 7.0 \end{bmatrix} \tag{A.52}$$

For $\mathbf{P}^{1|0}$, we use (A.15) and we have

$$\mathbf{P}^{1|0} = \mathbf{F}\mathbf{P}^{0|0}\mathbf{F}^T + \mathbf{Q} = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2.25 & 0.5 \\ 0.5 & 2 \end{bmatrix} \tag{A.53}$$

The next two items to be calculated in the Kalman filter process are the innovation term $\mathbf{S}$ and the Kalman gain $\mathbf{K}$. We use (A.37) to calculate the innovation term and obtain

$$\mathbf{S}^1 = \mathbf{H}\mathbf{P}^{1|0}\mathbf{H}^T + \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2.25 & 0.5 \\ 0.5 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3.25 & 0.5 \\ 0.5 & 3 \end{bmatrix}$$

To calculate the Kalman gain, we use (A.21) and we have

$$\mathbf{K}^1 = \mathbf{P}^{1|0}\mathbf{H}^T \left(\mathbf{S}^1\right)^{-1} = \begin{bmatrix} 2.25 & 0.5 \\ 0.5 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.3158 & -0.0526 \\ -0.0526 & 0.3421 \end{bmatrix} = \begin{bmatrix} 0.6842 & 0.0526 \\ 0.0526 & 0.6579 \end{bmatrix}$$

The final two items that we seek are the state estimate $\mathbf{m}^{1|1}$ and its error covariance $\mathbf{P}^{1|1}$. We already have $\mathbf{K}^1$, $\mathbf{z}^1$ and $\mathbf{m}^{1|0}$, and substitute them into (A.18), we have

$$\begin{aligned} \mathbf{m}^{1|1} &= \mathbf{m}^{1|0} + \mathbf{K}^1(\mathbf{z}^1 - \mathbf{H}\mathbf{m}^{1|0}) \\ &= \begin{bmatrix} 8.5 \\ 7.0 \end{bmatrix} + \begin{bmatrix} 0.6842 & 0.0526 \\ 0.0526 & 0.6579 \end{bmatrix} \left(\begin{bmatrix} 7.3757 \\ 6.1924 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 8.5 \\ 7.0 \end{bmatrix} \right) \\ &= \begin{bmatrix} 8.9280 \\ 5.9193 \end{bmatrix} \end{aligned} \tag{A.54}$$

This is our estimate of the target state at $k = 1$. The true state of the target at $k = 1$ is (A.47). The final item that we need to calculate in the Kalman filter process is $\mathbf{P}^{1|1}$. We use (A.19) to obtain

$$\begin{aligned} \mathbf{P}^{1|1} &= \mathbf{P}^{1|0} - \mathbf{K}^1\mathbf{H}\mathbf{P}^{1|0} \\ &= \begin{bmatrix} 2.25 & 0.5 \\ 0.5 & 2 \end{bmatrix} - \begin{bmatrix} 0.6842 & 0.0526 \\ 0.0526 & 0.6579 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2.25 & 0.5 \\ 0.5 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 0.6842 & 0.0526 \\ 0.0526 & 0.6579 \end{bmatrix} \end{aligned} \tag{A.55}$$

Thus we complete one iteration of the Kalman filter process. If we continue the same process, and calculate the six items $\mathbf{m}^{k|k-1}$, $\mathbf{P}^{k|k-1}$, $\mathbf{S}^k$, $\mathbf{K}^k$, $\mathbf{m}^{k|k}$ and $\mathbf{P}^{k|k}$ in the same manner, we have

$$\mathbf{m}^{2|2} = \begin{bmatrix} 10.3420 \\ 6.615 \end{bmatrix} \mathbf{m}^{3|3} = \begin{bmatrix} 13.2682 \\ 6.7068 \end{bmatrix} \mathbf{m}^{4|4} = \begin{bmatrix} 16.1874 \\ 7.4895 \end{bmatrix}$$

$$\mathbf{m}^{5|5} = \begin{bmatrix} 21.5138 \\ 9.4754 \end{bmatrix} \mathbf{m}^{6|6} = \begin{bmatrix} 26.8649 \\ 7.4208 \end{bmatrix} \mathbf{m}^{7|7} = \begin{bmatrix} 29.5995 \\ 8.3988 \end{bmatrix}$$

$$\mathbf{m}^{8|8} = \begin{bmatrix} 33.6102 \\ 9.1764 \end{bmatrix} \mathbf{m}^{9|9} = \begin{bmatrix} 37.7248 \\ 9.3920 \end{bmatrix} \mathbf{m}^{10|10} = \begin{bmatrix} 42.4720 \\ 9.7404 \end{bmatrix} \tag{A.56}$$

We show the actual state sequence, $\mathbf{x}^k$, $k = 0, \cdots, 10$ and the estimated state sequence, $\mathbf{m}^{k|k}$, $k = 0, \cdots, 10$ in Figure A.2.

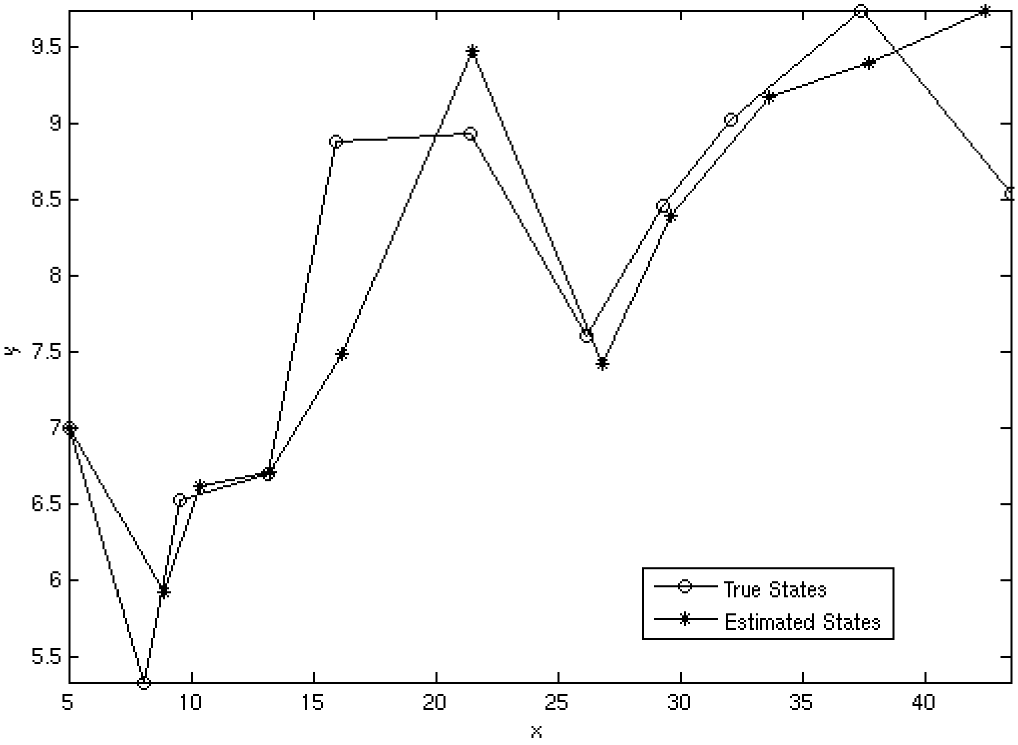


Figure A.2: Tracking example using the Kalman filter

Appendix B

Target Tracking Using Particle Filter

In this appendix, we provide a target tracking example. The state of the target, x, evolves based on the following motion model

$$x^k = x^{k-1} + 5 + v^k \tag{B.1}$$

where v^k is the unpredictable disturbances during the traversal of the target. Note that in this example, the target state, x, is one-dimensional. The variance of v^k is 1.0. At the beginning, when time step $k = 0$, $x^0 = 0.1$.

Our measurement z is related to the target state, x, according to the following *nonlinear* measurement model

$$z^k = \frac{(x^k)^2}{20} + w^k \tag{B.2}$$

where w^k is the measurement noise. Note that w^k does not necessarily model the measurement noise of a sensor node. Here, we provide a general target tracking problem, where (B.1) models *any* general target motion model and (B.2) models *any* measurement model.

The motion model is apparently similar to a straight line in (B.1). However, the measurement model in (B.2) is nonlinear. The particle filter algorithm is designed to solve nonlinear tracking problems like this. To demonstrate the *correctness* of the particle filter algorithm, we set the variance of the measurement noise, w^k, to be 1.0×10^{-5} first. The number of samples, $N_s = 1000$ in this appendix. The evolution of the target state, x^k, and the estimated target state using particle filter is shown in Figure B.1. In this experiment, the (simulated) true target states are marked as circles, and we can see that the target motion has some fluctuations, since the noise variance is nonzero. The estimated target states, using particle filter, are marked as stars. We can see from Figure B.1 that particle filter is very successful in tracking the target, even thought measurement model is nonlinear. The *mean squared error* (MSE) between the true target states and the estimated target states, over 20 time steps, is 7.5×10^{-4}.

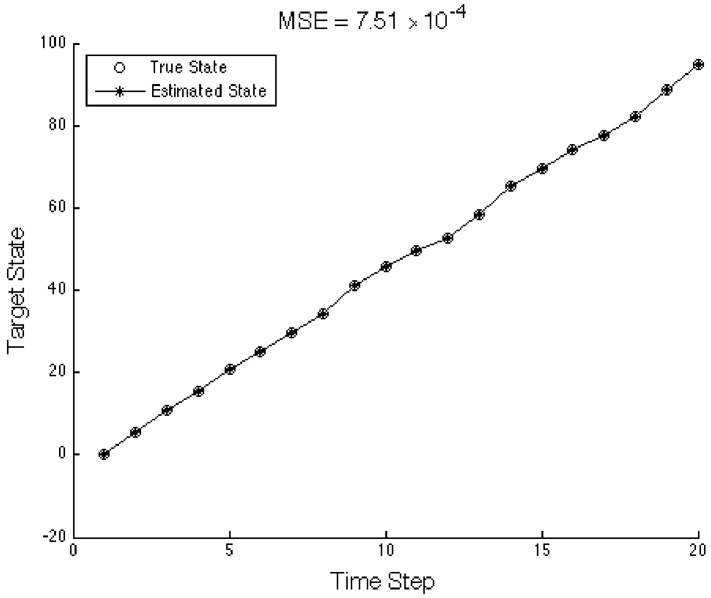


Figure B.1: Target tracking result using particle filters. The true target states are marked as circles, while the estimated target states are marked as stars. The variance of v^k is 1.0, while the variance of w^k is 1.0×10^{-5}. The tracking result is correct over 20 time steps.

As another example, we use the same motion model in (B.1) and the same measurement model in (B.2) to perform another tracking experiment with the variance of the measurement noise, w^k, equalling 1.0. Again, we illustrate the true target states (circles) and the estimated target states (stars) on the same figure in Figure B.2. The tracking result in shown in Figure B.2. The MSE is 0.1302, which is much larger than the previous experiment. We can also visibly see the mismatch between the true target states and the estimated target states.

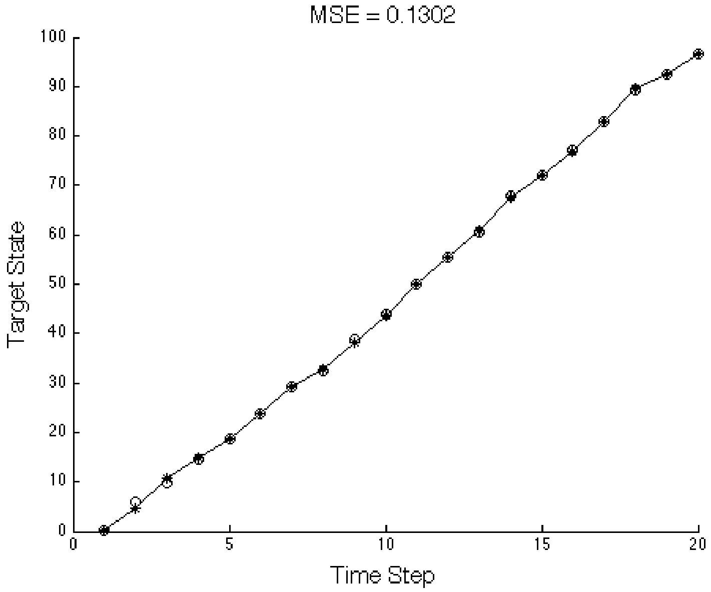


Figure B.2: Target tracking result using particle filters. The variance of v^k is 1.0, while the variance of w^k is 1.0. We can see that at time step $k = 1$ and $k = 2$, there exists some distinguishable tracking error.

Appendix C

Relaxation Labeling as an Optimization Process

C.1 Introduction

Appendix C is a revision of [49].

Consider a system of N objects and M labels. Let $P_i(\lambda_j)$ denote the "confidence" that object i has label λ_j. The confidence $P_i(\lambda_j)$ has probability-like properties:

$$0 \leq P_i(\lambda_j) \leq 1 \qquad \sum_{j=1}^{M} P_i(\lambda_j) = 1. \tag{C.1}$$

Following [51], we iteratively update the confidence of node i having label j as

$$P_i^{t+1}(\lambda_j) = \frac{P_i^t(\lambda_j)\left[1 + q_i^t(\lambda_j)\right]}{D_i^t}, \tag{C.2}$$

where $D_i^t = \sum_k P_i^t(\lambda_k)\left[1 + q_i^t(\lambda_k)\right], k = 1, \cdots, N$ is a normalization required to ensure that $P_i(\lambda_j)$ sums to 1, and t stands for iteration. $q_i^t(\lambda_j)$ is the "support" of object i having label λ_j. More details follow.

The support is defined as

$$q_i(\lambda) = \sum_j \sum_\mu R_{ij}(\lambda, \mu) P_j(\mu) \tag{C.3}$$

where the superscript for iteration is omitted for clarity. The $R_{ij}(\lambda, \mu)$ in (C.3) is the compatibility function of objects i and j having label λ and μ, respectively. The design of the compatibility function is problem-dependent, and we require that $-1 \leq R_{ij}(\lambda, \mu) \leq 1$.

For brevity, we define $Q_i(\lambda) = \left[1 + q_i^t(\lambda_j)\right]$, and (C.2) becomes

$$P_i^{t+1}(\lambda_j) = \frac{P_i^t(\lambda_j) Q_i^t(\lambda_j)}{\sum_\mu P_i^t(\mu) Q_i^t(\mu)}, \tag{C.4}$$

C.2 Why Relaxation?

Our objective is to show that the relaxation labeling algorithm can be formulated as an optimization problem, and the relaxation labeling algorithm is indeed a type of relaxation method. Relaxation is an algorithm that can be found in areas such as optimization and differential equations. We define the "feasible region", $\mathbf{P}$, to be the collection of points

$$\mathbf{P} = \left[P_1(\lambda_1), P_1(\lambda_2), \cdots, P_2(\lambda_1), P_2(\lambda_2), \cdots, P_N(\lambda_M)\right]^T \tag{C.5}$$

where each point in $\mathbf{P}$ satisfies the criteria in (C.1). For example, $0 \leq P_1(\lambda_2) \leq 1$, and $\sum_j P_1(\lambda_j) = 1$. Denote the objective function that we wish to minimize as $F(\mathbf{P})$. We iteratively replace the current point $\mathbf{P}^t$ with a more "relaxed" point $\mathbf{P}^{t+1}$, and eventually minimize $F(\mathbf{P})$. A relaxation method has two criteria:

1. $\mathbf{P}^{t+1}$ contains $\mathbf{P}^t$
2. $F(\mathbf{P}^{t+1}) \leqslant F(\mathbf{P}^t)$

Now we demonstrate that relaxation labeling can be formulated as an optimization problem. First, the objective function can be formulated as

$$F(\mathbf{P}) = -\frac{1}{2} \sum_i \sum_\lambda P_i(\lambda) Q_i(\lambda) \tag{C.6}$$

and subject to the conditions given in (C.1)

$$0 \leq P_i(\lambda_j) \leq 1 \tag{C.7}$$

$$\sum_{j=1}^{M} P_i(\lambda_j) = 1. \tag{C.8}$$

(the reason for choosing this objective function is explained in the next section). We can observe from (C.4) that relaxation labeling is an iterative process, and we can rewrite (C.4) as a gradient method

$$P_i^{t+1}(\lambda) = P_i^t(\lambda) + \phi r_i^t(\lambda) \tag{C.9}$$

where

$$r_i^t(\lambda) = \frac{P_i^t(\lambda)\left[Q_i^t(\lambda) - \sum_\mu P_i^t(\mu)Q_i^t(\mu)\right]}{\sum_\mu P_i^t(\mu)Q_i^t(\mu)} \tag{C.10}$$

and

$$\phi = 1. \tag{C.11}$$

To satisfy the first criteria of a relaxation method, we require that $P_i^{t+1}(\lambda)$ also satisfies the conditions given in (C.1). Since $P_i^{t+1}(\lambda) = P_i^t(\lambda) + r_i^t(\lambda)$, we have

$$\sum_{j=1}^{M} P_i^{t+1}(\lambda_j) = \sum_{j=1}^{M} P_i^t(\lambda_j) + \sum_{j=1}^{M} r_i^t(\lambda_j) \tag{C.12}$$

Since $\sum_{j=1}^{M} P_i^{t+1}(\lambda_j) = 1$ and $\sum_{j=1}^{M} P_i^t(\lambda_j) = 1$, we have

$$\sum_{j=1}^{M} r_i(\lambda_j) = 0 \qquad i = 1, \cdots, N \tag{C.13}$$

and

$$r_i(\lambda_j) \geqslant 0 \qquad if \qquad P_i(\lambda_j) = 0 \tag{C.14}$$

To satisfy the second criteria of a relaxation method, we have

$$F(\mathbf{P}^{t+1}) \leqslant F(\mathbf{P}^t) \tag{C.15}$$

$$\Rightarrow \quad F(\mathbf{P}^{t+1}) - F(\mathbf{P}^t) \leqslant 0 \tag{C.16}$$

$$\Rightarrow \quad \frac{F(\mathbf{P}^{t+1}) - F(\mathbf{P}^t)}{\triangle t} \leqslant 0 \tag{C.17}$$

$$\Rightarrow \quad \frac{\partial F(\mathbf{P})}{\partial t} \leqslant 0 \tag{C.18}$$

$$\Rightarrow \quad \frac{\partial F(\mathbf{P})}{\partial P_i(\lambda_j)}\frac{\partial P_i(\lambda_j)}{\partial t} \leqslant 0 \tag{C.19}$$

Denoting $\bigtriangledown F \equiv \left[\frac{\partial F(\mathbf{P})}{\partial P_i(\lambda_j)} \ldots\right]$ and observe that r in (C.12) is the change of $\mathbf{P}$ over time, we have

$$\bigtriangledown F \cdot \mathbf{r} \leqslant 0 \tag{C.20}$$

where

$$\mathbf{r} = [r_1(\lambda_1), r_1(\lambda_2), \cdots, r_2(\lambda_1), r_2(\lambda_2), \cdots, r_N(\lambda_M)]^T \tag{C.21}$$

C.3 Design of the Objective Function

In summary, the relaxation labeling process can be formulated as the following optimization problem

$$\begin{array}{lll} \textit{minimize} & F(\mathbf{P}) = -\frac{1}{2}\sum_i \sum_\lambda P_i(\lambda) Q_i(\lambda) & \text{(C.22)} \\ \textit{subject} \quad \textit{to} & 0 \leq P_i(\lambda_j) \leq 1 \qquad i = 1, \cdots, N & \text{(C.23)} \\ \textit{and} & \sum_{j=1}^{M} P_i(\lambda_j) = 1 \qquad i = 1, \cdots, N \qquad j = 1, \cdots, M & \text{(C.24)} \end{array}$$

The relaxation labeling process is a gradient method to find $\mathbf{r}$ satisfying

$$\begin{aligned} P_i^{t+1}(\lambda) &= P_i^t(\lambda) + r_i^t(\lambda) && \text{(C.25)} \\ \sum_{j=1}^{M} r_i(\lambda_j) &= 0 \qquad i = 1, \cdots, N && \text{(C.26)} \\ r_i(\lambda_j) &\geqslant 0 \qquad if \qquad P_i(\lambda_j) = 0 && \text{(C.27)} \\ \bigtriangledown F \cdot \mathbf{r} &\leqslant 0 && \text{(C.28)} \end{aligned}$$

The reason for choosing the objective function in (C.22) is that *inconsistency* is defined as the difference between $P_i(\lambda)$ and $Q_i(\lambda)$ [24]. Intuitively, $P_i(\lambda)$ is what every object "thinks" about its own labeling, and $Q_i(\lambda)$ is what its neighbors "think" about it [5]. Hence if we maximize the following function [5]

$$\frac{1}{2}\sum_i \sum_\lambda P_i(\lambda) Q_i(\lambda) \qquad \text{(C.29)}$$

the inconsistency will be minimized, since (C.29) is maximized when $P_i(\lambda) = Q_i(\lambda)$ (minimal inconsistency). To maximize (C.29) is equivalent to minimizing (C.22). The $\frac{1}{2}$ in (C.22) is to scale $Q_i(\lambda)$ to be between 0 and 1, that is, $0 \leq \frac{1}{2} Q_i(\lambda) \leq 1$.

C.4 Proof of Relaxation Labeling as an Optimization Process

We need to prove that $r_i(\lambda)$ in (C.10) satisfies (C.26), (C.27) and (C.28). Denote $\mathbf{X} = \sum_\mu P_i(\mu) Q_i(\mu)$, we have

$$\begin{aligned}
\sum_{\lambda} r_i(\lambda) &= \frac{1}{\mathbf{X}} \sum_{\lambda} P_i(\lambda) \left[Q_i(\lambda) - \mathbf{X} \right] && \text{(C.30)} \\
&= \frac{\sum_{\lambda} P_i(\lambda) Q_i(\lambda)}{\mathbf{X}} - \frac{\sum_{\lambda} \left[P_i(\lambda) \mathbf{X} \right]}{\mathbf{X}} && \text{(C.31)} \\
&= \frac{\mathbf{X}}{\mathbf{X}} - \frac{\mathbf{X} \sum_{\lambda} P_i(\lambda)}{\mathbf{X}} && \text{(C.32)} \\
&= \frac{\mathbf{X}}{\mathbf{X}} - \frac{\mathbf{X}}{\mathbf{X}} && \text{(C.33)} \\
&= 1 - 1 && \text{(C.34)} \\
&= 0 && \text{(C.35)}
\end{aligned}$$

and if $P_i(\lambda) = 0$ then $r_i(\lambda) = 0$, and thus equations (C.26) and (C.27) are satisfied.

Next we check equation (C.28). Now we use different indexes in (C.6)

$$F(\mathbf{P}) = -\frac{1}{2} \sum_{k} \sum_{\lambda_l} P_k(\lambda_l) Q_k(\lambda_l) \qquad \text{(C.36)}$$

and we have

$$\begin{aligned}
\bigtriangledown F &= \frac{\partial F}{\partial P_i(\lambda)} && \text{(C.37)} \\
&= -\frac{1}{2} \sum_{k} \sum_{l} \left[\frac{\partial P_k(\lambda_l)}{\partial P_i(\lambda)} Q_k(\lambda_l) + P_k(\lambda_l) \frac{\partial Q_k(\lambda_l}{\partial P_i(\lambda)} \right] && \text{(C.38)} \\
&= -\frac{1}{2} \sum_{k} \sum_{l} \left[Q_k(\lambda_l) + P_k(\lambda_l) \sum_{j} \sum_{\mu} \frac{\partial}{\partial P_i(\lambda)} R_{kj}(\lambda_l, \mu) P_j(\mu) \right] && \text{(C.39)} \\
&= -\frac{1}{2} \sum_{k} \sum_{l} \left[Q_k(\lambda_l) + P_k(\lambda_l) R_{ki}(\lambda_l, \mu) \right] && \text{(C.40)} \\
&= -\frac{1}{2} \mathbf{Q} - \frac{1}{2} \sum_{k} \sum_{l} \left[P_k(\lambda_l) R_{ki}(\lambda_l, \mu) \right] && \text{(C.41)} \\
&= -\frac{1}{2} \mathbf{Q} - \frac{1}{2} \mathbf{Q} && \text{(C.42)} \\
&= -\mathbf{Q} && \text{(C.43)}
\end{aligned}$$

So

$$\begin{aligned}
\bigtriangledown F \cdot \mathbf{r} &= -\sum_{i} \sum_{\lambda} r_i(\lambda) Q_i(\lambda) && \text{(C.44)} \\
&= -\sum_{i} \frac{1}{\mathbf{X}} \sum_{\lambda} P_i(\lambda) Q_i(\lambda) \left[Q_i(\lambda) - \mathbf{X} \right] && \text{(C.45)}
\end{aligned}$$

Furthermore,

$$\begin{aligned}
\sum_\lambda P_i(\lambda)Q_i(\lambda)\left[Q_i(\lambda)-\mathbf{X}\right] &= \sum_\lambda P_i(\lambda)\left[Q_i(\lambda)-\mathbf{X}\right]^2+\mathbf{X}\sum_\lambda P_i(\lambda)\left[Q_i(\lambda)-\mathbf{X}\right] && \text{(C.46)}\\
&= \sum_\lambda P_i(\lambda)\left[Q_i(\lambda)-\mathbf{X}\right]^2+\mathbf{X}\sum_\lambda P_i(\lambda)Q_i(\lambda)-\mathbf{X}\sum_\lambda P_i(\lambda)\mathbf{X} \\
&= \sum_\lambda P_i(\lambda)\left[Q_i(\lambda)-\mathbf{X}\right]^2+\mathbf{X}^2-\mathbf{X}^2\sum_\lambda P_i(\lambda) && \text{(C.47)}\\
&= \sum_\lambda P_i(\lambda)\left[Q_i(\lambda)-\mathbf{X}\right]^2+\mathbf{X}^2-\mathbf{X}^2 && \text{(C.48)}\\
&= \sum_\lambda P_i(\lambda)\left[Q_i(\lambda)-\mathbf{X}\right]^2 && \text{(C.49)}\\
&\geqslant 0 && \text{(C.50)}
\end{aligned}$$

and since $\mathbf{X} \geqslant 0$, this implies that $\triangledown F \cdot \mathbf{r} \leqslant 0$ and (C.28) is satisfied.

Appendix D

Probability of Being Malicious for Nodes in Secure Tracking

Here we have the probabilities for all the nodes in the last secure tracking experiment in Section 6.3.3.

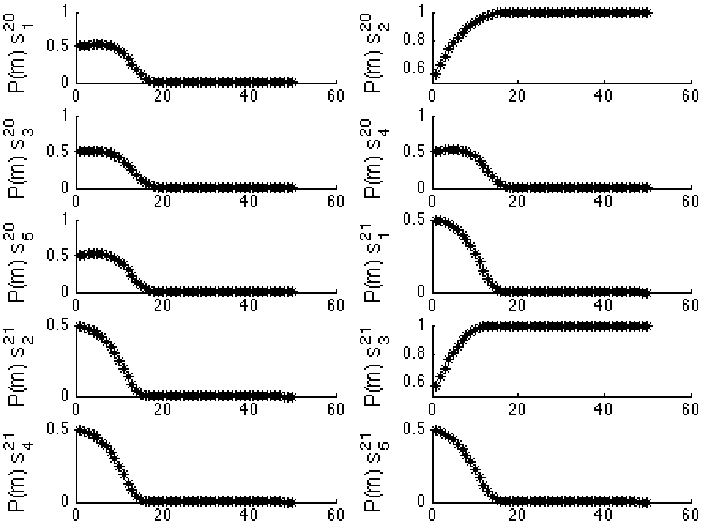


Figure D.1: Probability of being malicious nodes for the five nodes at $t = 20$ and another five nodes at $t = 21$. The malicious nodes are s_2^{20} and s_3^{21}, which agrees with what we have found here.

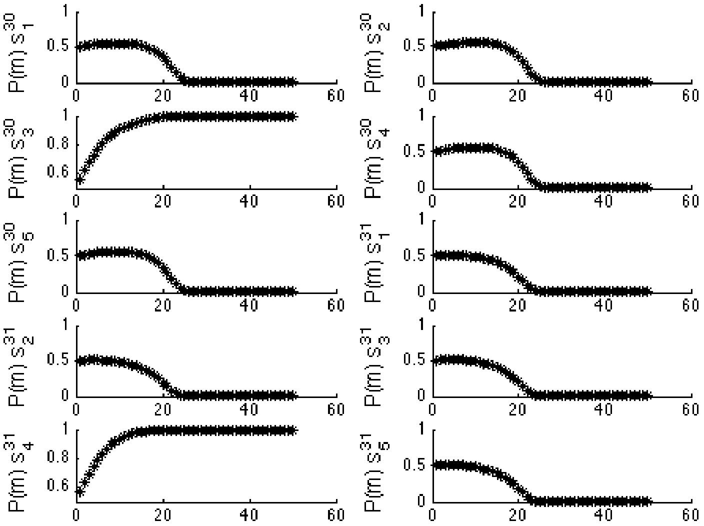


Figure D.2: Probability of being malicious nodes for the five nodes at $t = 30$ and another five nodes at $t = 31$. The malicious nodes are s_3^{30} and s_4^{31}, which agrees with what we have found here.

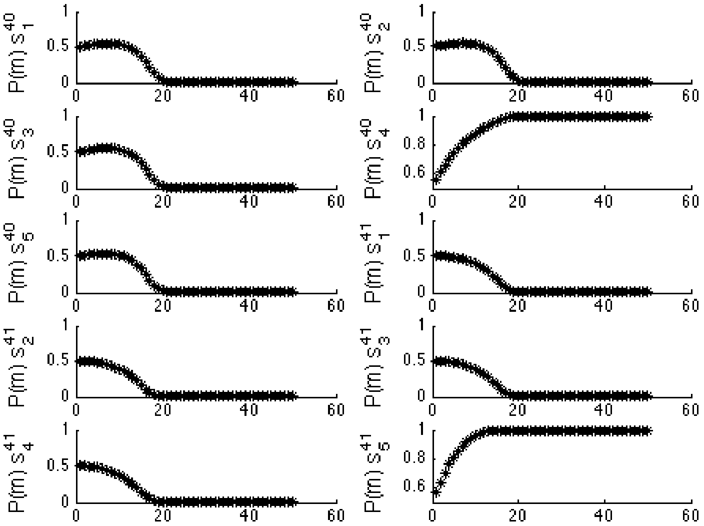


Figure D.3: Probability of being malicious nodes for the five nodes at $t = 40$ and another five nodes at $t = 41$. The malicious nodes are s_4^{40} and s_5^{41}, which agrees with what we have found here.

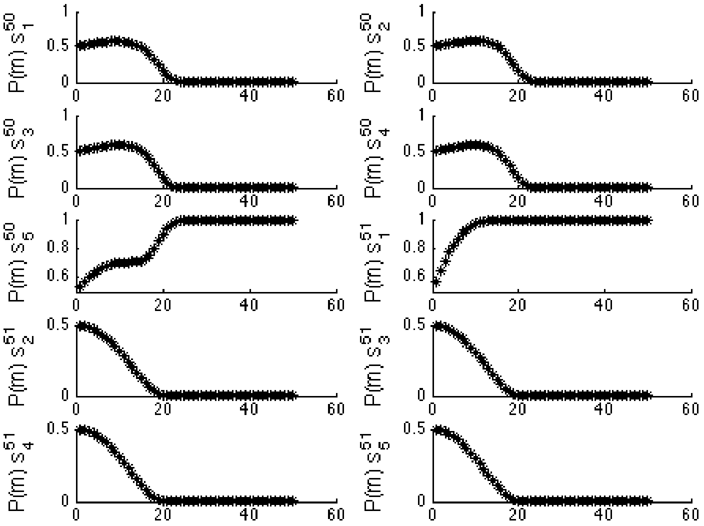


Figure D.4: Probability of being malicious nodes for the five nodes at $t = 50$ and another five nodes at $t = 51$. The malicious nodes are s_5^{50} and s_1^{51}, which agrees with what we have found here.