Position Estimation of Nodes Moving in a Wireless Sensor Network

by

Lingchen Zhou

TR14-229, February 28, 2014

This is an unaltered version of the author's MCS thesis

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
E-mail: fcs@unb.ca
http://www.cs.unb.ca

Abstract

This thesis investigates the use of measured radio signal strength indicator (RSSI) in wireless sensor networks (WSNs) to provide an estimated distance. Estimated distances are arrived at using experimental calibration of a free space model to find the best free space model exponent and transmission power level over distances up to 7.78 m. We use these distances to estimate the position of a moving wireless sensor node. A new application called MobiPos for indoor distance determination using TelosB module has been developed and tested. In our MobiPos application, the moving node collects RSSI values from TestFtsp messages sent by up to eight stationary nodes. The moving node, in turn, collects valid RSSI observations from the same set, and formulates a special MULTI_RSSI message that is sent to a stationary gateway node. The gateway node collects and time stamps received MULTI_RSSI messages for post processing using a least squares position estimation process.

We experimentally verified our research on a 23.92 m long mobile wireless sensor network testbed with up to six stationary nodes and one moving node.

With an average velocity of 0.19 m/s, and 131 received `MULTI_RSSI` messages over a test period of 127 seconds , we found that 64 observation sets (49%) converged with reasonable position estimates compared to approximate true positions. The average position difference for these 64 estimated positions compared to their approximate true positions was 5.03 m.

Acknowledgements

I thank my supervisor, Prof. Bradford G. Nickerson, for many insightful conversations during the development of the ideas in this thesis, and for his wisdom, friendship, understanding, for teaching me how to be a good researcher, and for helpful comments on the text.

I would like to thank Dr. Brent Petersen, my instructor for the EE6514 Wireless Communications course. I appreciated his boundless patience and encouragement.

My parents have been an inspiration throughout my life. They have always supported my dreams and aspirations. I'd like to thank them for all they are, and all they have done for me.

Table of Contents

List of Tables

List of Figures

List of Symbols, Nomenclature or Abbreviations

a	The moving node
AOA	Angle of Arrival
AP	Access Point
b	difference between the estimated and approximate true positions
$\mathbf{B}$	identity matrix in the least squares solution
$\mathbf{C}_{\hat{\mathbf{x}}}$	covariance matrix
d_i	The observed distance from moving node to stationary node
$\bar{d}_i$	The average observed distance from moving node to stationary node
$\hat{d}_i$	estimated distances
df	degree of freedom
δ	Interval between `TestFtsp` messages
Δ	Interval between two sets of `TestFtsp` messages
$\Delta\mathbf{x}$	correction vector
$\boldsymbol{\ell}$	The vectors of observations
$F(\mathbf{x}, \ell) = 0$	A nonlinear function relating $\mathbf{x}$ and $\boldsymbol{\ell}$
FSPM	Free Space Propagation Model
FTSP	Flooding Time Synchronization Protocol
GPS	Global Positioning System
GMHLD	GPS/MEM Hybrid Location-Detection
GLONASS	Global Navigation Satellite System
JAMA	Java Matrix Package
k	Number of stationary nodes
$\boldsymbol{\ell}$	vector of observations
LNSM	Log-Normal Shadowing Model

LOS	Line of Sight
m	The number of observed distances from moving node to stationary node i
ms	millisecond
n	Path loss exponent
$node_i$	The stationary nodes
NLOS	Non Line of Sight
NRND	Not Recommended for Newdesign
$\mathbf{P}$	Weight Matrix
$\mathbf{p_i}$	Position of the stationary node i.
p_j	The position of the stationary node i on the inner track
p'_j	The position of the stationary node i on the outer track
$\hat{\mathbf{r}}$	The estimated vector of residuals
$\bar{r}_i$	The average of residuals for each stationary node
RF	Radio Frequency
RFID	Radio Frequency Identication
RSS	Radio Signal Strength
RSSI	Radio Signal Strength Indicator
RTT	Round Trip Time of Flight
$\sigma^2_{i,j}$	The variance for the distance from moving node to stationary node
$\hat{\sigma}^2_0$	The estimated reference variance of the observations
s_i	Rssi observations
S	The set of rssi observations
SSDOF	Signal Strength Difference of Arrival
t_1	The time the moving node started to move
t_2	The time the moving node at any positions on the tracks
t_3	The time the moving node stoped
TDOA	Time Difference of Arrival
TOF	Time of Flight
UWB	Ultra Wide Band
$\mathbf{w}$	closure vector
WSN(s)	Wireless Sensor Network(s)
$\mathbf{x}$	The position of the moving node a
$(\bar{x}, \bar{y}, \bar{z})$	The approximate true position of the moving node
$(\Delta x, \Delta y, \Delta z)$	The difference of two positions
$(\hat{x}, \hat{y}, \hat{z})$	The estimated position of the moving node

Chapter 1

Introduction

Determining the position of moving objects is of great interest in many fields, including navigation, air traffic control, military operation and tracking of purchased goods. Many companies have been working on indoor positioning for years. Google, for example, released its indoor positioning application "My Location" [40] for Google Maps in November, 2011. My Location can be only used in some selected retail stores and airports, and is only supported by Android systems. The tracking technology maps a user's location based on cell towers, Global Positioning System (GPS) and publicly broadcast Wi-Fi signals [11]. Besides Google, Microsoft and Nokia may launch the same service sometime in 2012.

Compared with indoor positioning systems, outdoor positioning of moving objects can be provided by satellite systems, such as the GPS, Global Navigation Satellite System (GLONASS) or European Galileo System [17]. Since satellite signals are not strong enough to penetrate buildings, other systems such as cell phone triangulation [1] must be used indoor. Radio-frequency identification (RFID) [22] are wireless devices used to track object movement, and are typically used in an inventory control system.

In this thesis we will investigate the use of measured radio signal strength (RSS) in wireless sensor networks, to provide an estimate of distance. These estimated distances will be used to estimate the position of a wireless sensor node moving in a network of three or more stationary wireless sensor nodes.

1.1 Indoor Positioning Methods

Wireless indoor positioning systems have been widely investigated in recent years and many positioning methods have been developed [38]. There are two typical location estimation methods; line of sight (LOS) [22], and non line of sight (NLOS) [22]. RSS-based methods fall into the category of a NLOS methodology. The NLOS method calculates the signal path loss due to propagation and then translates the difference between the transmitted signal strength and the received signal strength into a range estimate, as shown in Fig. 1 [22].

A wireless sensor network (WSN) consists of distributed autonomous sensor nodes to monitor physical or environmental conditions. Localization of moving nodes is a main function in WSNs and this function has been used in several sensor network operations and applications [8, 21, 28, 30]. There

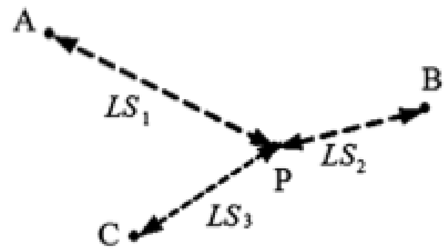


Figure 1.1: Positioning based on RSS, where LS_1,LS_2,LS_3 stand for the measured path loss (from[22]).

are several positioning methods proposed, for WSNs, including those given in the following sections.

1.1.1 Signal Strength Difference of Arrival

Signal attenuation follows certain rules in wireless transmission, based on the difference between transmitted signal strength and received signal strength. In Papadakis and Traganitis's research [28], an experiment was performed in an IEEE 802.11-based infrastructure network. Using two stationary nodes to measure the distance difference to a moving node, a hyperbolic model of distance difference is arrived at. The intersection of two hyperbolae gives the position of the moving node. The simulated result indicates a position error below 2.4m for lower measurement noise, and below 4.2m for a case of high measurement noise.

1.1.2 Angle of Arrival

In this method [37], an angle is formed by the propagation direction of an incident wave and some reference directions. To measure the angle, an expensive and complicated antenna is needed at the stationary nodes. Wong et al. [37] simulated the positioning method in a 30m×30m square, where they placed 4 access points (AP) in the corners, a mobile node sent signals to each AP and then the actual position of the mobile is computed. Based on the result, they concluded that the AOA-based positioning's accuracy is better than 2 m, which is a good positioning method.

1.1.3 Time Difference of Arrival

In this method [8], the only parameter we need to identify is the time of arrival of two or more signals at the base station. Special signal feature division difference transmitters are used. In order to enhance the accuracy of this method, clock synchronization is key. This method is widely used in both outdoor and indoor distance estimation. For example, a GPS system is a typical outdoor position estimate method, employing very precise atomic clocks on moving satellites. For indoor use, a location system called Cricket [29] also used time difference of arrival (TDoA). Cricket uses a combination of RF and ultrasonic energy to determine distances and estimate positions. It provides distance ranging and positioning accuracy between 1 and 3 cm.

1.1.4 Radio Signal Strength Indicator

The radio signal strength indicator (RSSI) does not require additional hardware devices [16]. Due to the multiple paths of a radio frequency (RF) signal and NLOS , the transmission of the signal is easily affected by the environment. Minghui and Huiqing [27] report a distance estimate accuracy of 3 m (with Gaussian filtering) using RSSI over a distance of 30m. The RSSI value for wireless transmission is readily available, so RSSI has been used in several applications [6, 20, 25].

Another example of using the RSSI method is the location engine [5]. Chipcon released the CC2431 System on Chip solution with a built-in location engine which uses the RSSI signal. The location engine was used to track soccer players on a soccer pitch [25] but because of different types of "noise", the results indicated more than 20 meter distance errors in the worst case. This was too imprecise for tracking soccer players' movement, so the researchers reverted to using GPS for position estimation. Chipcon has recently decided to move the CC2431 location engine to NRND (not recommended for new design) status.

1.1.5 Ultra Wide Band

Ultra wide band (UWB) is defined as an intentional radiator with an absolute bandwidth greater than 500 MHz having a fractional bandwidth greater than 20% [15]. Compared with other wireless communication technologies, UWB has many advantages including low power spectral density, high transfer rate, large system capacity, robustness to interference and multipath, low power consumption and low cost.

The Ubisense company uses UWB technology to build their own real-time positioning system [19]. The UWB approach uses both TDoA and AoA, along with special hardware to filter very short duration (<1ns) impulse signals spread across a very wide bandwidth.

1.1.6 Time of Flight

The main idea of Time of Flight (ToF) is to measure the propagation time (the time a radio signal travel from a transmitter to a receiver), and then multiply by the speed of light c to get the distance d, so the main problem is to measure the propagation time as accurate as possible. In the Will et al. paper [36], they used MSB-A2 nodes with a CC1101 transceiver to measure the round trip time of flight (RTT), which is the back and forth ToF from the transmitter to the receiver. A *sync word* is used to start and stop a timer on the CC1101. When the time required for responding to a TOF packet is subtracted, they obtained RTT. Will et al. also model the jitter in the signal to estimate the actual RTT more accurately. As a result, this method gives at worst 9m error within 30m distance, with an average of 3m error over 30m.

Table 1 summarizes several indoor position estimation methods.

Table 1.1: Accuracy of different methods for indoor position or distance estimation.

Method	**Range of Use**	**Accuracy**	**Reference(s)**
SSDoA	20m (simulated)	below 2.4m for the lower noise below 4.2m for the high noise	[28]
AoA	30m x 30m square (simulated)	better than 2m	[37]
RSSI	30m	distance error of 10% of range	[27]
Cricket(TDoA)	indoor area	1cm to 3cm	[8, 29]
My Location	indoor area	several meters	[24]
Ubisense (UWB)	$400m^2$for 4 stationary nodes	tens of centimeters	[19]
ToF	30m	at worst 9m, 3m average	[36]

Chapter 2

Mathematical Model for Position Estimation

The least squares method [12, 26, 33, 35] finds the best fit that minimizes the sum of squared residuals, a residual being the difference between an observed value and the fitted value provided by a mathematical model.

A moving node receives radio signals from k stationary nodes, based on k stationary nodes at positions $(x_1,\ y_1,\ z_1)$, $(x_2,\ y_2,\ z_2)$, $\ldots$, $(x_k,\ y_k,\ z_k)$. The distances from a moving node to the stationary nodes are denoted d_1, d_2, $\ldots$, d_k. The least squares process determines an estimate $\hat{\mathbf{x}} = (\hat{x}_a, \hat{y}_a, \hat{z}_a)$ of the moving node a's position, along with a $(3, 3)$ covariance matrix $\mathbf{C}_{\hat{\mathbf{x}}}$ that estimates the accuracy of the estimated position $\hat{\mathbf{x}}$.

The least squares method starts with a mathematical model $F(\mathbf{x}, \boldsymbol{\ell}) = 0$, where $\mathbf{x}$ is the vector of patameters, $\boldsymbol{\ell}$ is the vectors of observations, and

$F(\mathbf{x}, \ell) = 0$ is a nonlinear function relating $\mathbf{x}$ and $\boldsymbol{\ell}$. For distance observations.

$$F(\mathbf{x}, \boldsymbol{\ell}) = \sqrt{(x_a - x_i)^2 + (y_a - y_i)^2 + (z_a - z_i)^2} - d_i = 0 \quad for\ i = 1, 2, \ldots, k \tag{2.0.1}$$

where d_i is the observed distance, $\mathbf{x} = (x_a, y_a, z_a)$ is the position of the moving node, and $\mathbf{p_i} = (x_i, y_i, z_i)$ is the position of the stationary node i.

For least squares estimation, we compute the partial derivatives of $F(\mathbf{x}, \boldsymbol{\ell})$, with respect to the unknown parameters $\mathbf{x}$, and evaluate them at the parameter estimates, i.e.:

$$\mathbf{A}_{k,n} = \frac{\partial F(\mathbf{x}, \boldsymbol{\ell})}{\partial \mathbf{x}} \Big|_{\mathbf{x}^0, \boldsymbol{\ell}} = \begin{bmatrix} \frac{\partial F}{\partial x_a}, & \frac{\partial F}{\partial y_a}, & \frac{\partial F}{\partial z_a} \end{bmatrix} |_{\mathbf{x}^0, \boldsymbol{\ell}} \tag{2.0.2}$$

where,

$$\frac{\partial F}{\partial x_a} \Big|_{\mathbf{x}^0, \boldsymbol{\ell}} = \frac{1}{2} \frac{2(\hat{x}_a - x_i)}{\sqrt{(\hat{x}_a - x_i)^2 + (\hat{y}_a - y_i)^2 + (\hat{z}_a - z_i)^2}} = \frac{\hat{x}_a - x_i}{d_i} \tag{2.0.3}$$

$$\frac{\partial F}{\partial y_a} \Big|_{\mathbf{x}^0, \boldsymbol{\ell}} = \frac{1}{2} \frac{2(\hat{y}_a - y_i)}{\sqrt{(\hat{x}_a - x_i)^2 + (\hat{y}_a - y_i)^2 + (\hat{z}_a - z_i)^2}} = \frac{\hat{y}_a - y_i}{d_i} \tag{2.0.4}$$

$$\frac{\partial F}{\partial z_a} \Big|_{\mathbf{x}^0, \boldsymbol{\ell}} = \frac{1}{2} \frac{2(\hat{z}_a - z_i)}{\sqrt{(\hat{x}_a - x_i)^2 + (\hat{y}_a - y_i)^2 + (\hat{z}_a - z_i)^2}} = \frac{\hat{z}_a - z_i}{d_i} \tag{2.0.5}$$

With $\hat{\mathbf{x}} = (\hat{x}_a, \hat{y}_a, \hat{z}_a)$, we have the estimated distance $\hat{d}_i$

$$\hat{d}_i = \sqrt{(\hat{x}_a - x_i)^2 + (\hat{y}_a - y_i)^2 + (\hat{z}_a - z_i)^2} \tag{2.0.6}$$

The partial derivative of $F(\mathbf{x}, \boldsymbol{\ell})$ with respect to $\boldsymbol{\ell}$ is

$$\mathbf{B} = \frac{\partial F(\mathbf{x}, \boldsymbol{\ell})}{\partial l}\Big|_{\mathbf{x},\boldsymbol{\ell}} = -1 \tag{2.0.7}$$

For the Euclidean distance model of eq. (2.0.6), the $\mathbf{B}$ matrix acts as the identity matrix in the least squares solution.

The observation weight matrix $\mathbf{P}$ and closure vector $\mathbf{w}$ are defined as follows:

$$\mathbf{w}_{k,1} = \hat{\boldsymbol{\ell}} - \boldsymbol{\ell} = \hat{\mathbf{d}} - \boldsymbol{\ell} = \begin{bmatrix} \hat{d}_1 \\ \hat{d}_2 \\ \vdots \\ \hat{d}_k \end{bmatrix} - \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix} \tag{2.0.8}$$

and

$$\mathbf{P}_{k,k} = \begin{bmatrix} 1/\sigma_1^2 & 0 & 0 & \dots & 0 \\ 0 & 1/\sigma_2^2 & 0 & \dots & 0 \\ 0 & 0 & 1/\sigma_3^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1/\sigma_k^2 \end{bmatrix} \tag{2.0.9}$$

where σ_i^2 is the variance of the observation ℓ_i.

The calculated correction vector $\Delta\mathbf{x}$ is computed iteratively using least squares estimation as follows:

$$\Delta\mathbf{x} = (\mathbf{A^T P A})^{-1}\mathbf{A^T P w} \tag{2.0.10}$$

We compute the least squares solution as follows:

$$\hat{\mathbf{x}}^j = \hat{\mathbf{x}}^{j-1} + \Delta\mathbf{x} \tag{2.0.11}$$

When $|\Delta\mathbf{x}| \leq \epsilon$, the iteration stops, and the estimated position of the moving node is $\hat{\mathbf{x}}^j$.

The least squares estimation process also provides a covariance matrix $\mathbf{C}_{\hat{\mathbf{x}}}$ of the estimated parameters. $\mathbf{C}_{\hat{\mathbf{x}}}$ is computed as follows [26]:

$$\mathbf{C}_{\hat{\mathbf{x}}} = \hat{\sigma}_0^2(\mathbf{A^T P A})^{-1} \tag{2.0.12}$$

where $\hat{\sigma}_0^2$ is the estimated reference variance of the observations $\boldsymbol{\ell}$ computed as

$$\hat{\sigma}_0^2 = \frac{\hat{\mathbf{r}}^{\mathbf{T}}\mathbf{P}\hat{\mathbf{r}}}{df} \tag{2.0.13}$$

for df = degree of freedom, and $\hat{\mathbf{r}}$ is the estimated vector of residuals defined as

$$\hat{\mathbf{r}} = \mathbf{A}\Delta\mathbf{x} + \mathbf{w} \tag{2.0.14}$$

In our experiment, we have four estimated distances, i.e.:

$$F(\mathbf{x}, \ell) = \sqrt{(x_a - x_i)^2 + (y_a - y_i)^2 + (z_a - z_i)^2} - d_i = 0 \ \ for\ i = 1, 2, 3\ and\ 4 \tag{2.0.15}$$

$$\mathbf{A}_{4,3} = \begin{bmatrix} \frac{\hat{x}_a - x_1}{d_1} & \frac{\hat{y}_a - y_1}{d_1} & \frac{\hat{z}_a - z_1}{d_1} \\ \frac{\hat{x}_a - x_2}{d_2} & \frac{\hat{y}_a - y_2}{d_2} & \frac{\hat{z}_a - z_2}{d_2} \\ \frac{\hat{x}_a - x_3}{d_3} & \frac{\hat{y}_a - y_3}{d_3} & \frac{\hat{z}_a - z_3}{d_3} \\ \frac{\hat{x}_a - x_4}{d_4} & \frac{\hat{y}_a - y_4}{d_4} & \frac{\hat{z}_a - z_4}{d_4} \end{bmatrix} \tag{2.0.16}$$

The closure vector $\mathbf{w}$ and weight marix $\mathbf{P}$ are computed using equations (4.8) and (4.9), as follows:

$$\mathbf{w}_{4,1} = \hat{\boldsymbol{\ell}} - \boldsymbol{\ell} = \hat{\mathbf{d}} - \boldsymbol{\ell} = \begin{bmatrix} \hat{d}_1 \\ \hat{d}_2 \\ \hat{d}_3 \\ \hat{d}_4 \end{bmatrix} - \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} \tag{2.0.17}$$

$$\mathbf{P}_{4,4} = \begin{bmatrix} 1/\sigma_1^2 & 0 & 0 & 0 \\ 0 & 1/\sigma_2^2 & 0 & 0 \\ 0 & 0 & 1/\sigma_3^2 & 0 \\ 0 & 0 & 0 & 1/\sigma_4^2 \end{bmatrix} \tag{2.0.18}$$

Algorithm 1 summarizes the least squares position estimation process for a moving node with coordinates $\mathbf{x}$, and distances from $\mathbf{x}$ to stationary nodes $d_1, d_2, \cdots, d_k$.

Algorithm 1: Compute the least squares position estimation $\hat{\mathbf{x}}$ of a moving node.

Input:

Distances between the moving node to the stationary nodes $d_i,\ i = 1, 2, \cdots, k$, along with their variances σ_i^2;
The initinal position estimate of the moving node $\hat{\mathbf{x}}^0$;
Size of correction vector limit ϵ to stop iteration;
Positions of the stationary nodes, $\mathbf{P}_i, \quad i = 1, 2, \cdots, k$;

Output:

$\hat{\mathbf{x}}$ at iteration j: $\hat{\mathbf{x}}^j$;
Covariance matrix of the estimated parameters $C_{\hat{x}}$;
The estimated vector of residuals $\hat{\mathbf{r}}$;
The estimated reference variance $\hat{\sigma}_0^2$ of the observations $\boldsymbol{\ell}$;

```
Use equation (2.0.9) to compute P;
repeat
    Compute d̂_i  i = 1, 2, ··· , k using equation (2.0.6);
    Compute w using equation (2.0.8);
    Compute A using equation (2.0.2), which in turn, uses equations
    (2.0.3), (2.0.4) and (2.0.5);
    j ← j + 1;
    Compute Δx using equation (2.0.10), and x̂^j using equation (2.0.11);
until |Δx| ≤ ε;
Compute C_x̂ and σ̂_0^2 using equations (2.0.12) and (2.0.13);
```

Chapter 3

Distance from Signal Strength

3.1 Distance Measurement Model

3.1.1 Free Space Propagation Model

In our experiment, we use the free space propagation model as follows to determine the distance from the RSSI values [9, 18]. A free-space model is applicable to the following occasions:

1) the transmission distance is much larger than the antenna size and the carrier wavelength λ;

2) there are no obstacles between the transmitters and the receivers. Suppose the transmission power of wireless signal is P_t. The power of received signals of nodes located of a distance of d can be determined by the following formula:

$$P_r = \frac{\lambda^2}{4\pi} G_r \frac{1}{4\pi d^n} G_t P_t \tag{3.1.1}$$

where P_r is the received power in watts, λ is the carrier wave length in meters, G_r is the gain of the receiver antenna, G_t is the gain of transmitter antenna, P_t is the transmitter power in watts, d is the distance between transmitter and receiver in meters and n is the path loss exponent (based on the propagation environment), whose value is in the range of two to four.

The distance can now be determined as

$$d = \sqrt[n]{\frac{\lambda^2 G_r G_t P_t}{16\pi^2 P_r}} \tag{3.1.2}$$

The value of path loss exponent n is affected by the attenuation, reflection, multipath and other interference occurring during a radio signal transmission in the air. If the interference is smaller, then the value of n is smaller.

RSSI values are reported in dBm, which cannot be used in the equation above, so we convert to watts with equation (3.1.3), where $RSSI$ stands for the measured RSSI value (in dBm) at the node. Equation (3.1.1) is from [18].

$$P_r = 10^{\frac{RSSI}{10} - 3} \tag{3.1.3}$$

In [41], we measured distance errors of -4m over 10m, -2m over 6m, and +1m over 5m. These observations were obtained using a free space propagation model, with $n = 2.7$ and 3.0.

3.1.2 Log-Normal Shadowing Model

Distance errors in the free space propagation model are sometimes too large for accurate position estimation. We plan to investigate the log-normal shadowing model (LNSM) [18]. This model suits both indoor and outdoor situations and is defined as follows:

$$PL(d) = \overline{PL(d_0)} + 10n \log_{10}\left(\frac{d}{d_0}\right) + X_\sigma \tag{3.1.4}$$

where PL is the path loss in dB, d_0 is the near-earth reference distance in m, n is a path loss exponent depending on the surroundings, and X_σ is zero-mean Gaussian random variable in dB. The distance estimated can be computed as:

$$d = 10^{\frac{PL(d) - \overline{PL(d_0)} - X_\sigma}{10n}} d_0 \tag{3.1.5}$$

Besides the LNSM, the LNSM with dynamic variance (LNSM-DV) model [39] holds promise as an improved RSSI-based distance measurement approach. LNSM-DV has a better self-adaptability to different experimental environments, and could give more accurate distances estimates.

3.2 Experimental Design

The experiment was done in ITB214 (see Figure 3.3). The development computer runs CentOS version 6.0 and has TinyOS [4] development stack installed. The development computer is also used as the base station, where it can display RSSI values received from the stationary node. We used two Crossbow's TelosB sensor nodes (CC2420) as the sending node and the stationary node. The sending node communicates with the stationary node through the Wireless Sensor Network while the stationary node connects with the base station directly through USB connector.

3.2.1 CC2420

CC2420 [10] has a much higher rate than older radios which operates in 2.4 GHz ISM band with an effective data rate of 256 kbps. In the 2.4 GHz band, it has 16 channels in total from No. 11 to No. 26. Every channel occupying a 3 MHz bandwidth with a center frequency separation of 5 MHz for adjacent channels. CC2420 uses an encoding scheme that encodes 32 chips for a symbol of 4 bits.

In CC2420, RSSI is the estimate of the signal power and is calculated over 8 symbol periods and stored in the $RSSI\_VAL$ register. Chipcon specifies the following formula to compute the received signal power (P) in dBm:

$$P = RSSI\_VAL + RSSI\_OFFSET \tag{3.2.1}$$

where $RSSI\_OFFSET$ is about -45 dBm. The RSSI value can be used to determine the radio frequency (RF) signal power with reasonable accuracy.

3.2.2 RssiDemo

During the distance from RSSI experiment, we used the application *RssiDemo* [2] to collect data. Two TelosB (see Figure 3.7b) nodes with a CC2420 transceiver were used for this application; one is used as a gateway node and the other one is used as a sending node. The gateway node is connected to the development (base station) computer via a USB connector for both the application installation process and the entire experiment execution time. The sending node is a wireless sensor node. The sending node is connected to the development computer via a USB connector only during its application installation process. After installing the application, the sending node communicates with the gateway node through the wireless sensor network by sending TOS messages of 20 bytes every 100 ms. Figure 3.1 shows the architecture of the *RssiDemo* application and Figure 3.2 shows the wiring of the *RssiDemo* application. In Figure 3.2, a single box is a module, a double box is a configuration and a dashed border line denotes that a component is generic [32].

In our experiment, we set the sending mote's power to 0dBm, and send a packet every 100 ms. The stationary node receives the packet and then shows the RSSI value on the screen of the base station. We recorded the RSSI at 11 different locations from 1 meter to 20 meters as illustrated in Figure 3.3. The sending node was placed on the floor while the stationary node was about 80 centimeters high above the floor. We used the pythagorean theorem to measure the approximate straight path between the sending node and the

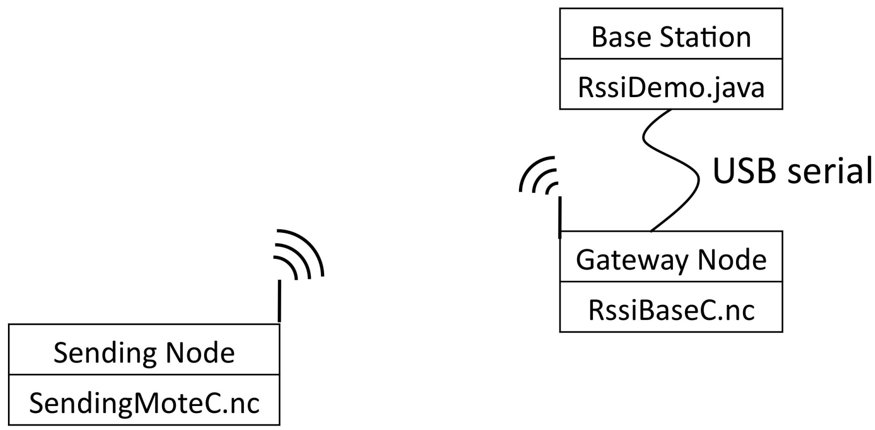


Figure 3.1: Experimental design architecture of the *RssiDemo* application.

stationary node. In each location, we recorded 50 RSSI values in 5 seconds, and converted these RSSI values to distance using equation (3.1.2).
The parameters we used in the experiment are shown in Figure 3.4.

3.3 Results

We conducted initial distance from RSSI experiments with CC2420 radio nodes to collect over 50 RSSI measurements for model parameter estimation. Then we choose 50 consecutive measurements from relatively stable measurements.
Our experimental results are shown in Tables 3.1 and 3.2. In the two tables, d is the actual measured distance, $\hat{x}_{RSSI}$ is the average (N=50) RSSI received from the sending node, S_{RSSI} is the standard deviation RSSI values, $\hat{d}$ is the estimate of distance and $S_{\hat{d}}$ is the standard deviation of estimated distance.

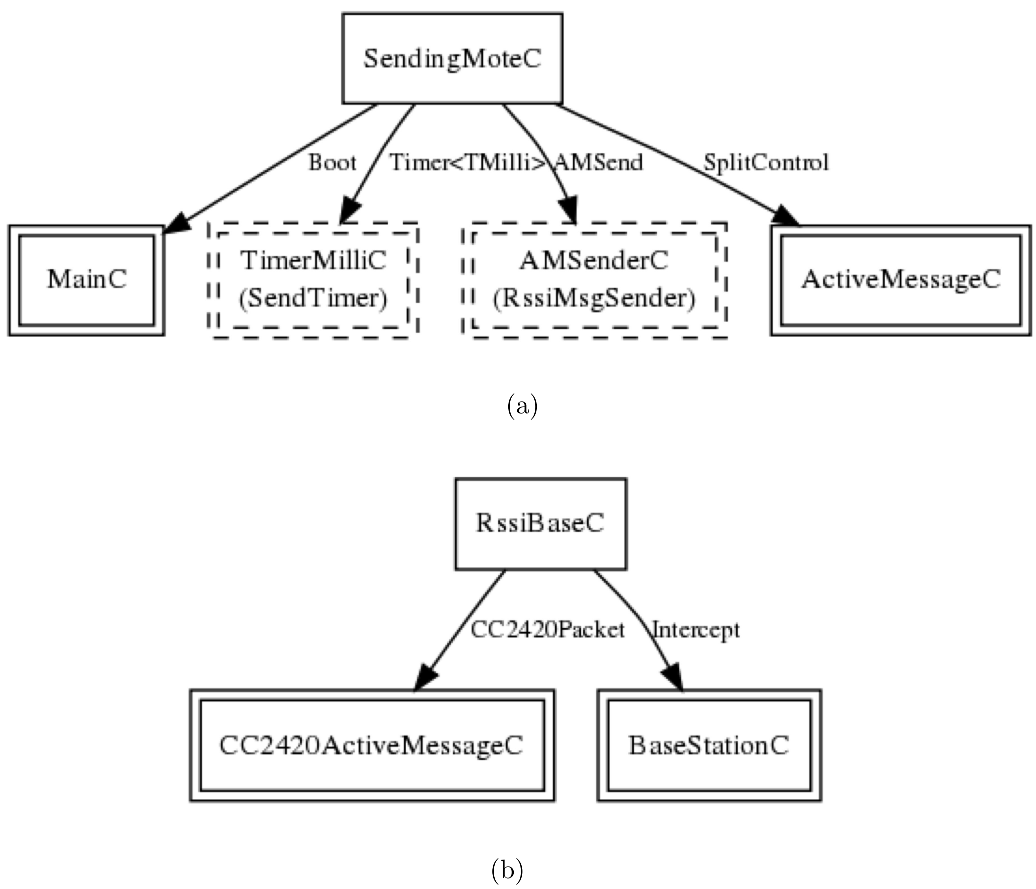


Figure 3.2: Wiring of the `RssiDemo` application (from [2]).

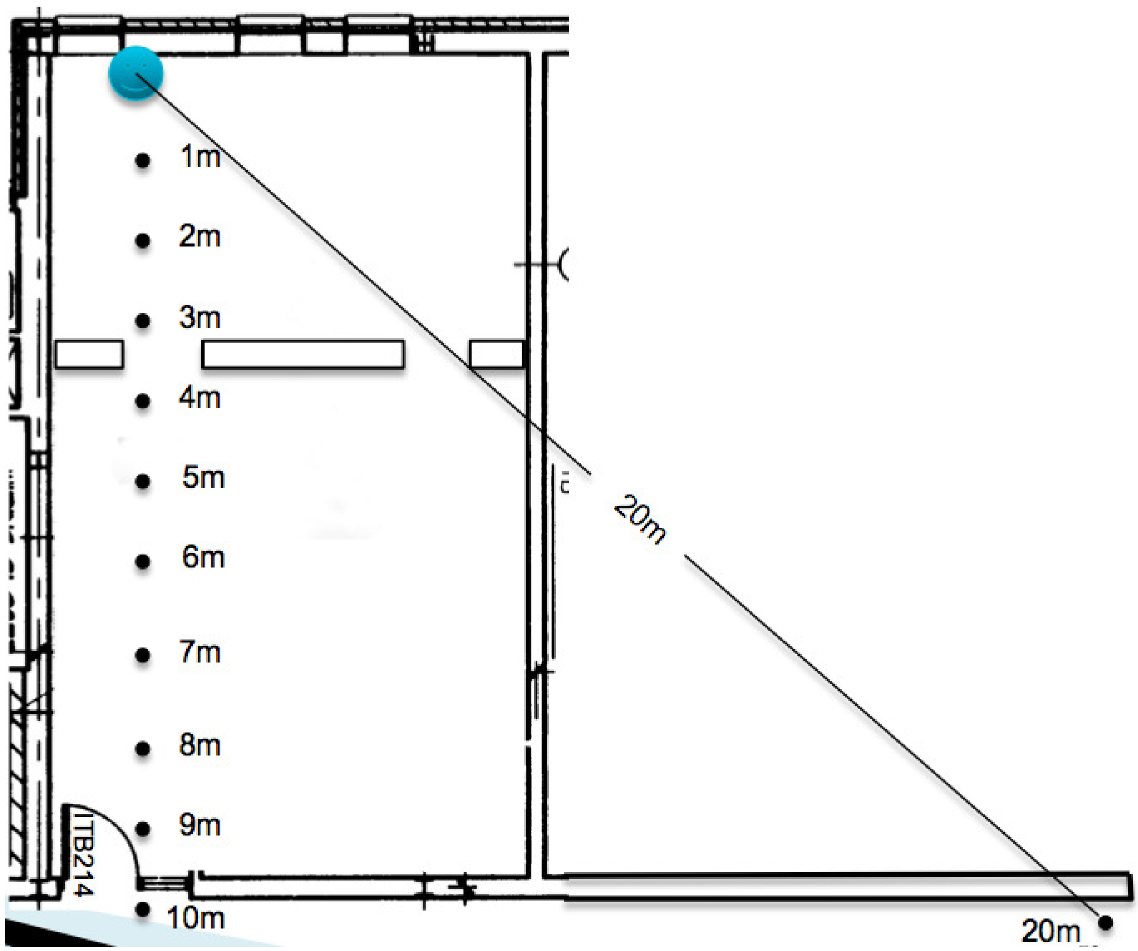


Figure 3.3: Sketch of Sending Mote Locations in ITB214.

Model Parameters		
Category	Parameter	value
Observation Points	Number	11
	Maximum Distance *d*	20 m
	Placement	indoor
TelosB Mote (CC2420)	Transmit Power	0 dBm (maximum)
	Frequency	2.048 GHz
	Antenna Gain	1
	Sending speed	250ms
Path Loss	Model	$P_r = \frac{\lambda^2}{4\pi} G_r \frac{1}{4\pi d^n} G_t P_t$
	Exponent *n*	2.7 and 3.0

Figure 3.4: Parameters of the experiment.

We can see from the tables that the standard deviation of estimated distance and estimated RSSI values have a distinct change at the distance of 5m, 6m and 20m. This means in the ITB214 lab, the signal strength is affected strongly by the environment.

Figure 3.5 shows the two groups of difference between $\hat{d}$ (estimated distance) and d (actual measured distance) when we chose different propagation exponents. It is clear that from 1 to 5 meters when $n = 3.0$ the estimated distance is closer to the real distance than when $n = 2.7$, but from 6 to 10 meters there is a quite opposite situation, and $n = 2.7$ is a more suitable path loss exponent in this area.

Figure 3.6 shows the estimated distance when $n = 3.0$, and the blue line represents the actual measured distances. The points which are far from the line are the less accurate ones. We can tell from Figure 3.5 that, when the

Table 3.1: Distance concerned for the RSSI measurement using free space model.

d(m)	$\hat{x}_{RSSI}(dBm)$	$\hat{x}_{RSSI}(W)$	S_{RSSI}	$\hat{d}(n=2.7)$	$\hat{d}(n=3.0)$	$S_{\hat{d}}$(n=3)
1	-39.56	1.11E-7	0.50	1.08	1.07	0.041
2	-52.92	5.11E-9	0.40	3.37	2.98	0.084
3	-55.36	2.91E-9	0.56	4.15	3.60	0.16
4	-57.76	1.67E-9	0.48	5.09	4.32	0.16
5	-62.12	6.14E-10	0.85	7.39	6.05	0.39
6	-54.22	3.78E-9	1.11	3.77	3.30	0.29
7	-53.90	4.07E-9	0.61	3.66	3.22	0.15
8	-58.04	1.57E-9	0.20	5.21	4.41	0.069
9	-60.18	9.59E-10	0.39	6.25	5.20	0.16
10	-59.70	1.07E-9	0.46	6.00	5.02	0.18
20	-83.66	4.31E-12	1.29	46.55	31.69	3.24

Table 3.2: Distance concerned for the RSSI measurement using Log-normal shadow model.

d(m)	$\hat{x}_{RSSI}(dBm)$	$\hat{x}_{RSSI}(W)$	S_{RSSI}	$\hat{d}(n=2.7)$	$\hat{d}(n=3.0)$	$S_{\hat{d}}$(n=3)
1	-39.56	1.11E-7	0.50	0.99	0.99	0.090
2	-52.92	5.11E-9	0.40	3.21	2.812	0.20
3	-55.36	2.91E-9	0.56	3.84	3.30	0.28
4	-57.76	1.67E-9	0.48	4.73	4.01	0.35
5	-62.12	6.14E-10	0.85	6.97	5.69	0.55
6	-54.22	3.78E-9	1.11	3.50	3.06	0.38
7	-53.90	4.07E-9	0.61	3.42	3.03	0.38
8	-58.04	1.57E-9	0.20	4.85	4.24	0.34
9	-60.18	9.60E-10	0.39	5.86	4.94	0.43
10	-59.70	1.072E-9	0.46	5.66	4.68	0.44
20	-83.66	4.31E-12	1.29	44.64	30.02	3.79

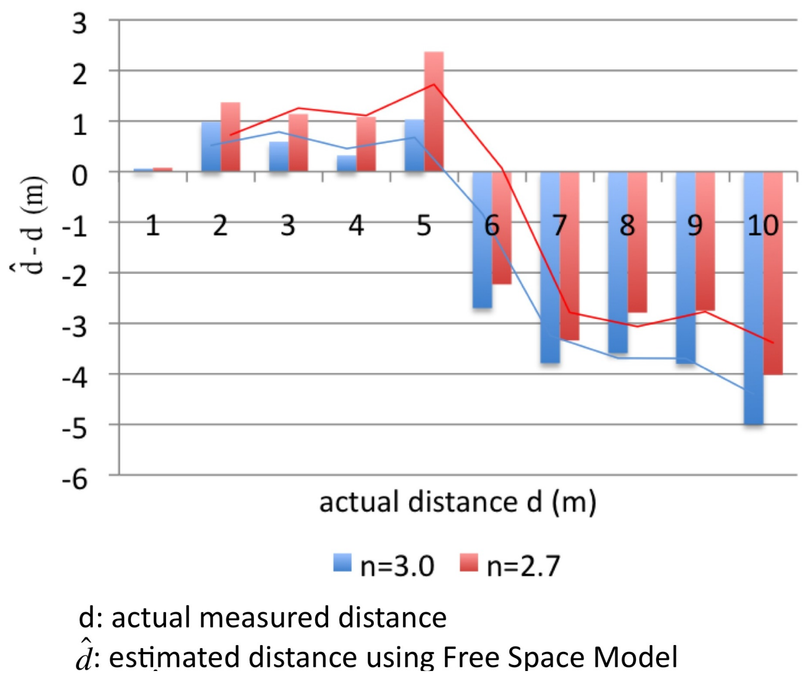


Figure 3.5: Difference between d and $\hat{d}$ when $n = 3.0$ and $n = 2.7$.

distance is over 5 meters, the accuracy decreases significantly. Based on this phenomenon we can assume that the model we used is probably only suitable in the range less than 5 meters.

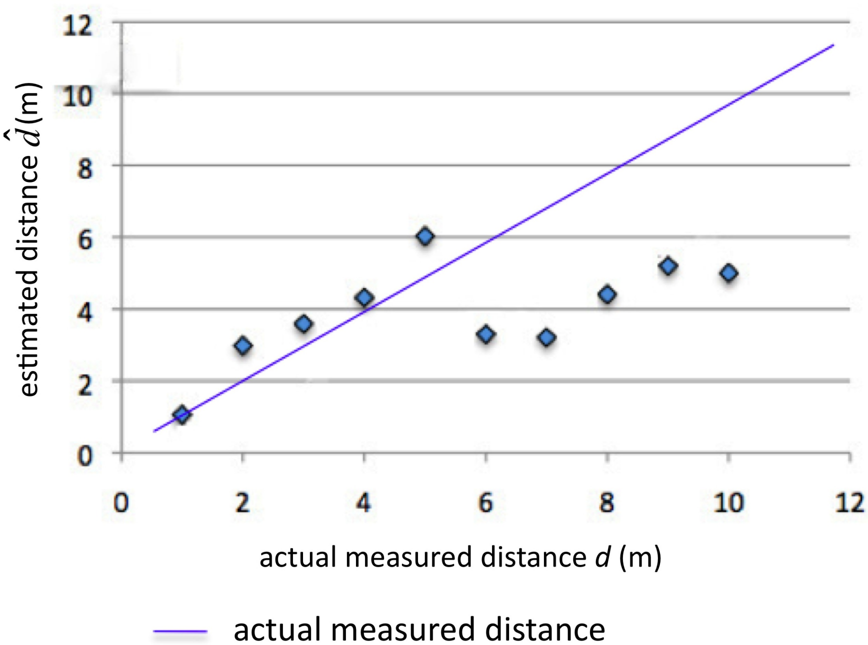


Figure 3.6: Estimated distance vs. actual measured distance.

3.4 Power Level Experiment

The transmission power of CC2420 is programmable. Table 3.3 shows the connection between different programmable settings and their corresponding output power. The default **PA_LEVEL** is 31, which is approximately 0 dBm. The experimental results in the previous section were all carried out with this default power level.

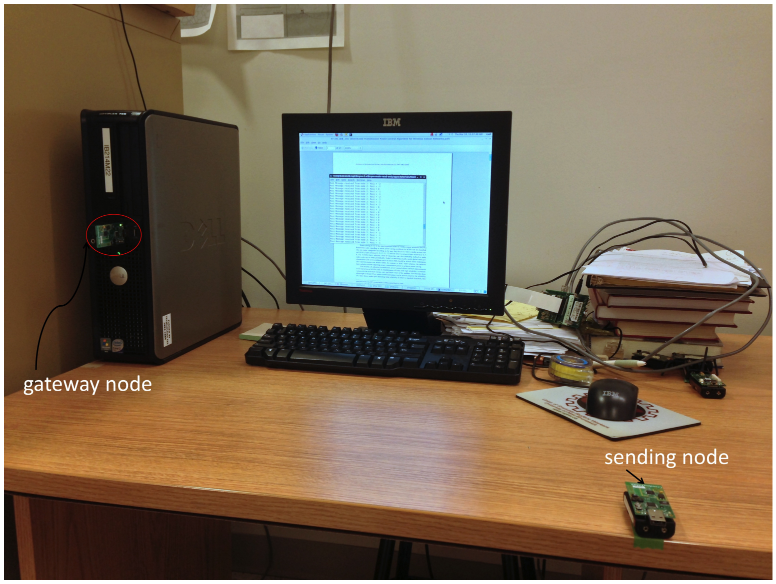
Table 3.3: Programable CC2420 output power settings and typical current consumption at 2.45GHz (from [10]).

PA_LEVEL	Output Power [dBm]	Current Consumption [mA]
31	0	17.4
27	-1	16.5
23	-3	15.2
19	-5	13.9
15	-7	12.5
11	-10	11.2
7	-15	9.9
3	-25	8.5

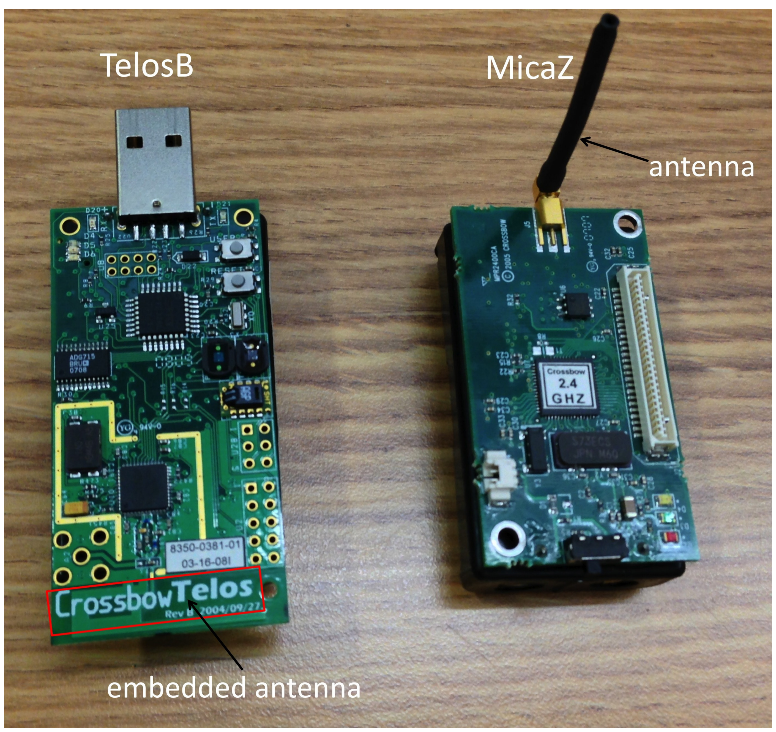
3.4.1 Short Distance Testing

This section used the *RssiDemo* application to determine how different output power levels affect the received RSSI value. We also compared the performance of the TelosB and MicaZ nodes. Figure 3.7a shows the locations of the sending node and gateway node for this power level experiment; the distance between the sending and gateway nodes is 0.79m. For this power level experiment, we used equation (3.1.2) for the free space (FS) model, and (3.1.5) for the log normal shadowing model (LNSM). Once average RSSI values are determined, power levels in dBm are computed using equation (3.2.1).

In Tables 3.4 and 3.5 we calculated the standard deviation (in parentheses) of 50 observed RSSI values, as well as the difference between real distance and estimated distances. Table 3.4 shows that the MicaZ node has very large



(a) Location of two nodes.



(b) TelosB node and MicaZ node.

Figure 3.7: Power level experiment setup with two types of CC2420 sensor nodes.

Table 3.4: Distances and distance differences from the sending node (MicaZ) to the gateway node (TelosB) using the *RssiDemo* application. Here, FS = free space model, LNSM = log normal shadowing model, n is the propagation exponent and distances are in m. The real distance is 0.79m, and number of observations N = 50.

Node	distance (standard deviation)		distance difference	
PA_LEVEL	3	31	3	31
average RSSI value	-75.68 (0.76)	-58.40 (0.81)		
LNSM ($n = 3.0$)	15.88 (15.09)	4.27 (3.48)	15.09	3.48
LNSM ($n = 2.7$)	21.38 (20.59)	4.97 (4.18)	20.59	4.18
FS ($n = 2.7$)	23.48 (22.69)	5.38 (4.59)	22.69	4.59
FS ($n = 3.0$)	17.13 (16.34)	4.54 (3.75)	16.34	3.75

Table 3.5: Distances and distance differences from the sending node (TelosB) to the gateway node (TelosB) using the *RssiDemo* application. Here, FS = free space model, LNSM = log normal shadowing model, n is the propagation exponent and distances are in m. The real distance is 0.79m, and N = 50.

Node	distance (standard deviation)		distance difference	
PA_LEVEL	3	31	3	31
average RSSI value	-67.90 (0.30)	-37.76 (0.48)		
LNSM ($n = 3.0$)	8.88 (8.09)	0.86 (0.07)	8.09	0.07
LNSM ($n = 2.7$)	11.23 (10.44)	0.87 (0.08)	10.44	0.08
FS ($n = 2.7$)	12.07 (11.28)	0.92 (0.14)	11.28	0.13
FS ($n = 3.0$)	9.41 (8.62)	0.93 (0.15)	8.62	0.14

errors using the highest and lowest transmission power. For the maximum power level 31 (around 0 dBm), the error is about five times smaller compared to the lowest power level 3 (around -25 dBm). This is the expected behavior as a large RSSI value implies a shorter distance. Note that the MicaZ antenna has a gain of 0 dBi (standard deviation of 0.8 dB) [7] compared to the embedded TelosB antenna (see Figure (3.7b)) for which the gain is not known. For the TelosB node, when we use the highest transmission power at 0 dBm, the estimated distances are reasonable, with the difference between the real distance and the estimated distance always less than 0.15m. The low power level distance estimate is significantly worse, i.e. giving between 61 and 130 times larger distance difference. In addition, the maximum power level gives a more accurate distance estimate and is heavily dependent on the hardware configuration. This experimental result shows that the free space and log normal shadowing model give approximately the same result for short estimated distances.

3.4.2 Longer Distance Testing

The purpose of this experiment is to test the performance of TelosB and MicaZ nodes for longer distances i.e. > 5 m. We placed the TelosB node and MicaZ node separately on the floor as sending nodes. A TelosB node connected via a USB cable acts as a gateway node to read the received RSSI value. The MicaZ mote signal was not received by the gateway node when placed as far away as the TelosB mote, so we moved the MicaZ mote 1.52m closer to the gateway node. Tables 3.6 and 3.7 show the experimental results.

Table 3.6: Distances and distance differences from the sending node (MicaZ) to the gateway node (TelosB) using the *RssiDemo* application. Here, FS = free space model, LNSM = log normal shadowing model, n is the propagation exponent and distances are in m. The real distance is 6.26 m, and N = 50.

Node	distance (st. deviation, N = 50)		distance difference	
PA_LEVEL	3	31	3	31
average RSSI value	-81.2 (0.61)	-54.98 (0.14)		
LNSM ($n = 3.0$)	24.73 (2.29)	3.26 (0.25)	18.47	-3
LNSM ($n = 2.7$)	33.94 (3.14)	3.71 (0.36)	27.68	-2.55
LNSM ($n = 2.0$)	118.46 (15.18)	5.93 (0.75)	112.2	-0.33
FS ($n = 3.0$)	26.14 (1.23)	3.49 (0.037)	19.88	-2.77
FS ($n = 2.7$)	37.56 (1.97)	4.01 (0.046)	31.3	-2.25
FS ($n = 2.1$)	105.90 (7.15)	5.96 (0.088)	99.64	-0.3

We note that the estimated distances are less than the real distance with maximum transmission power. Changing to a lower value exponent n would reduce the distance difference. The extra rows with $n = 2.0$ and 2.1 (see Tables 3.6 and 3.7) show the effect of this change. We note that the best estimate is obtained for high power setting in all four cases. The best distance estimate for longer distance (> 1m) is obtained for $n = 2.1$ for the free space (FS) model, and $n = 2.0$ for the log normal shadowing model (LNSM).

Table 3.7: Distances and distance differences from the sending node (TelosB) to the gateway node (TelosB) using the *RssiDemo* application. Here, FS = free space model, LNSM = log normal shadowing model, n is the propagation exponent and distances are in m. The real distance is 7.78 m, and N = 50.

Node	distance (st. deviation, N = 50)		distance difference	
PA_LEVEL	3	31	3	31
average RSSI value	-80.86 (0.57)	-57.86 (1.36)		
LNSM (n = 3.0)	23.84 (1.79)	4.04 (0.54)	16.06	-3.74
LNSM (n = 2.7)	34.44 (3.54)	4.83 (0.70)	26.66	-2.95
LNSM (n = 2.0)	119.96 (14.98)	8.33 (1.48)	112.18	0.55
FS (n = 3.0)	25.46 (1.12)	4.37 (0.42)	17.68	-3.14
FS (n = 2.7)	36.48 (1.78)	5.16 (0.55)	28.7	-2.62
FS (n = 2.1)	102.00 (6.40)	8.26 (1.11)	94.22	0.48

Chapter 4

Static Testing

To test the least squares algorithm, we connected the moving node to the base station, so that it was stationary, then used the least squares method in Algorithm 1 to estimate the position of the stationary "moving node".

4.1 Accurate Distance Observations

We first estimated (using Algorithm 1) the position of the moving node with accurately measured distances. These measured distances are computed from the positions of the stationary nodes and known position of the moving node, using equation (4.1.1), as follows:

$$\hat{d}_{i,a} = \sqrt{(x_i - x_a)^2 + (y_i - y_a)^2 + (z_i - z_a)^2},\ i = 1, 2, 3, 4 \tag{4.1.1}$$

4.1.1 Static Test Architecture

As shown in Figure 4.1, we chose the northwest floor corner in the lab as the origin. Using a metal tape measure, we obtained the positions of the four stationary nodes and the moving node `a` (which is actually stationary in this case). The tape measure we used has an accuracy of around 1mm for measuring the distance along a Cartesian axis.

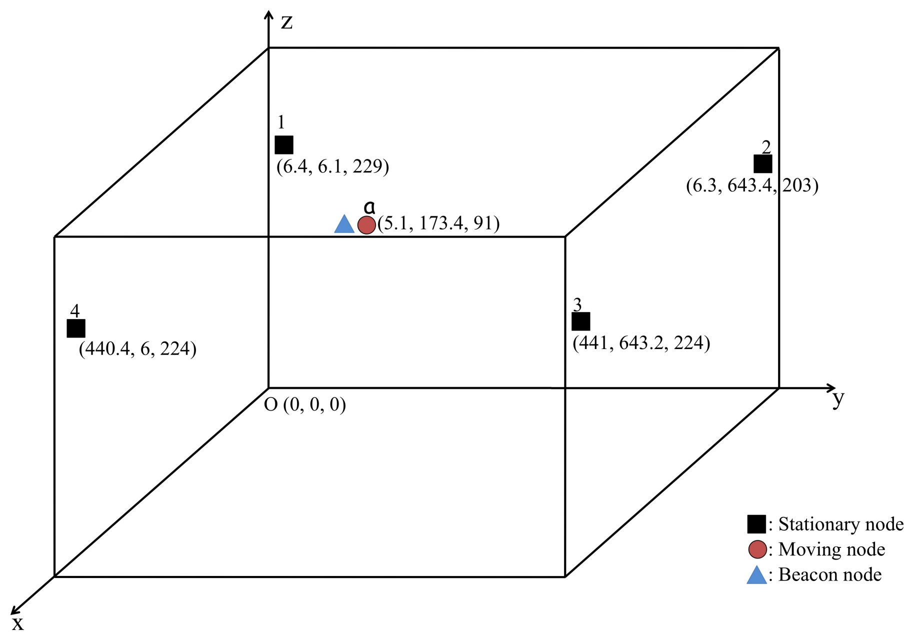


Figure 4.1: TelosB modules deployment for static testing in ITB214. Positions of nodes are shown in cm with respect to the origin.

Figure 4.2 shows the architecture of software used for static testing. As described in more detail in Section 5.3, the beacon node is a separate node running a program to transmit Flooding Time Synchronization Protocol (FTSP) packets. All static testing are performed using Tinyos-2.1.1 and two sensor nodes, the TelosB [13] and MicaZ [14]. Both TelosB and MicaZ use the CC2420 radio chip as shown in Figure 3.7b.

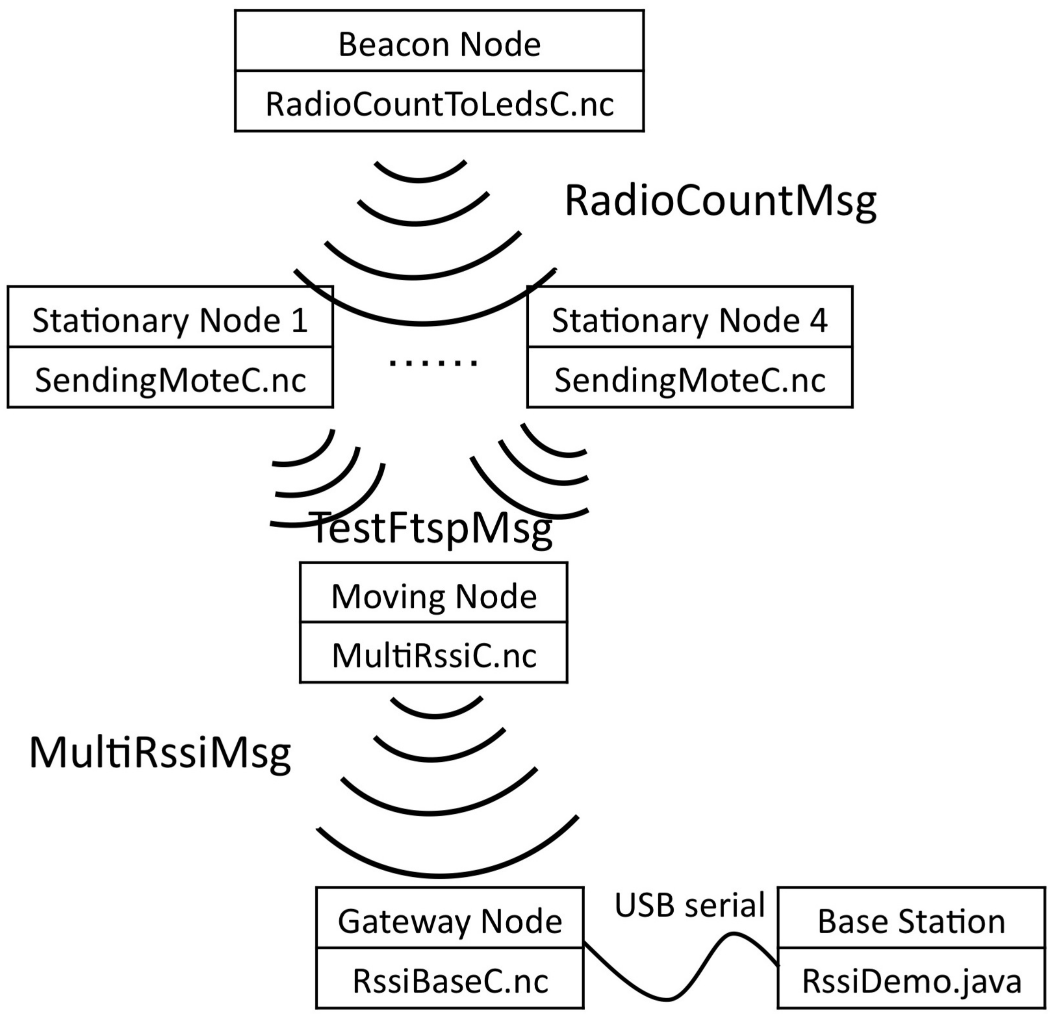


Figure 4.2: Experimental Design Architecture.

We noticed that when test program RssiDemo is initiated, the RSSI values changed significantly (e.g. ± 10), but settle down to steady values after 2 to 3 seconds. RSSI values are also very sensitive to objects moving in the lab. For example, when I move my hand within 5 cm of the gateway node, the RSSI values changed by e.g. -5. The 50 readings used to compute the average RSSI values in Table 4.2 and Table 4.3 were obtained after the RSSI values have settled down.

Figures 4.3 and 4.4 shows the deployment of the four stationary nodes, the beacon node and the moving node `a`.

Table 4.1 shows the measured stationary nodes' positions, and the accurately determined distance from the moving node a to each stationary node.

Table 4.1: Accurate input data for static testing. All units are in cm.

Node	**x**	**y**	**z**	$\tilde{d}_{i,a}$	**estimated** σ
a_0	223	325	200	N/A	N/A
1	6.4	6.1	229	221.4	0.1
2	6.3	643.4	203	485.2	0.1
3	441	643.2	224	624.9	0.1
4	440.4	6	224	444.2	0.1

After running the data in Table 4.1 through Algorithm 1 with $\epsilon = 0.1$ cm, we obtained the following results: $\hat{\mathbf{x}} = [50.01, 173.40, 91.03]$ cm,

$\hat{\mathbf{r}} = [0.003, -0.159, 0.016, -0.015]$ cm.

Computing $\hat{\sigma}_0^2$ using equation (2.13), with $df = 1$, we obtain $\hat{\sigma}_0^2 = 0.007$ and

Figure 4.3: Figure (top) is the TelosB nodes deployment for static testing in ITB214 facing West and Figure (bottom) is the TelosB nodes deployment for static testing in ITB214 facing East.

Figure 4.4: TelosB nodes deployment for static testing in ITB214 facing North.

$$\mathbf{C}_{\hat{\mathbf{x}}} = \hat{\sigma}_0^2(\mathbf{A}^{\mathbf{T}}\mathbf{PA})^{-1} = \begin{bmatrix} 6.83E-4 & -9.17E-5 & -3.26E-4 \\ -9.17E-5 & 3.50E-4 & 1.61E-4 \\ -3.26E-4 & 1.61E-4 & 0.0015 \end{bmatrix} \tag{4.1.2}$$

is computed using equation 2.12. Equation 4.1.2 indicates that the error in the z coordinate is $\sqrt{0.0015} = 0.039$ cm.

We use the coefficients of the covariance matrix $C_{\hat{x}}$ to compute the error ellipsoid, as follows:

$$\sigma_{axis} = \sqrt{\lambda_i}, i = 1, 2, 3 \tag{4.1.3}$$

where σ_{axis} is the length of the semi-axis (u, v, w) of a rotated coordinate framework, and λ_i (i = 1, 2, 3) are the three eigenvalue of $C_{\hat{x}}$. The orientation of the axies (u, v, w) are given by the normalized eigenvectors of $C_{\hat{x}}$.

Using the Java Matrix Package (JAMA) we obtain the eigenvalue for each axis as $\lambda_1 = 0.00057$, $\lambda_2 = 0.0016$, $\lambda_3 = 0.00032$, so we obtain:

$$\sigma_x = \sqrt{0.00057} = 0.24 \text{ cm} \tag{4.1.4}$$

$$\sigma_y = \sqrt{0.0016} = 0.04 \text{ cm} \tag{4.1.5}$$

$$\sigma_z = \sqrt{0.00032} = 0.018 \text{ cm} \tag{4.1.6}$$

4.1.2 Statistical Test of the Reference Variance

We use the method from [34] to test the quadratic form of the residuals with the following equation:

$$\frac{\nu\hat{\sigma}_0^2}{\chi^2_{\nu,\frac{\alpha}{2}}} \leq \sigma_0^2 \leq \frac{\nu\hat{\sigma}_0^2}{\chi^2_{\nu,1-\frac{\alpha}{2}}} \tag{4.1.7}$$

where ν is degree of freedom which is equal to 1 in our case, and $\chi^2_{\nu,\frac{\alpha}{2}}$ is the chi-square statistical value corresponding to ν degrees of freedom and confidence level $\frac{\alpha}{2}$. This tests the null hypothesis $\hat{\sigma}_0^2 = \sigma_0^2$.
For $\alpha = 0.01$, $1-\frac{\alpha}{2} = 0.995$, $\frac{\alpha}{2} = 0.005$, so $\chi^2_{\nu,\frac{\alpha}{2}}$ is equal to 7.87944 and $\chi^2_{\nu,1-\frac{\alpha}{2}}$ is equal to 0.00004, with these assumptions, equation (4.1.7) becomes:

$$\frac{4.72}{7.87944} \leq 1 \leq \frac{4.72}{0.00004} \tag{4.1.8}$$

which is

$$0.60 \leq 1 \leq 118000 \tag{4.1.9}$$

Equation (4.1.9) is true, so we cannot reject the null hypothesis, and we can assume that equation (4.1.2) is a reasonable estimate (at the 99% confidence level) of our position error.

4.2 Signal Strength Distance Observations

We measured the signal strength from each stationary node to the moving node a. The 8-bit signal strength RSSI of RssiMsg (see section 5.1) is determined by the moving node RssiBase.nc program (see section 5.4). The static test setting is as shown in Figure 4.1. Using the free space propagation model (see section 3.1.1) and the log normal shadowing model (see section 3.1.2), we obtained the distances in Table 4.2 and Table 4.3. The observed RSSI values are translated using equation (3.2.1) in Table 4.2 and Table 4.3, and $\tilde{d}_{i,a}$ are the accurate distances from Table 4.1. The distances $d_{i,a}$ are the average of N = 50 observations within 10 seconds, and the standard deviation (s) are computed for this sample of size 50.

Table 4.4 shows the estimated positions $\hat{\mathbf{x}}$ for the moving node computed using Algorithm 1, the distances from Table 4.2 and the stationary node positions in Table 4.1. Table 4.4 also shows the difference $\tilde{c} - c_F$, where $\tilde{c}$ is the accurate coordinate value and c_F is the coordinate value estimated using distances computed with the free space model. Similarly, the difference $\tilde{c} - c_L$ is also shown, where c_L is the coordinate value estimated using distances computed with the log normal shadowing model.

4.2.1 Statistical Test of the Reference Variance

Based on the input data in Tables 4.2 and 4.4, using the least squares method of chapter 2 we obtain the results shown below:

Table 4.2: TelosB distances from nodes 1, 2, 3, 4 to node a determined using the RSSI measurement (N = 50) and two models. Here FS = free space model, LNSM = log normal shadowing model, RSSI is the signal strength average, with the sample standard deviation (s) in parentheses. The first entry of two entries in each row is for $n = 2.7$, while the second entry is for $n = 3.0$. Distances are in m.

From node	$\tilde{d}_{i,a}$	RSSI (s)	$d_{i,a}$ FS	$d_{i,a} - \tilde{d}_{i,a}$	$d_{i,a}$ LNSM	$d_{i,a} - \tilde{d}_{i,a}$
1	2.21	-50.92(1.03)	2.84(0.24) 2.56(0.20)	0.63 0.35	2.66(0.34) 2.44(0.27)	0.45 0.23
2	4.85	-44.88(0.39)	1.70(0.055) 1.61(0.05)	-3.15 -3.24	1.56(0.10) 1.49(0.11)	-3.29 -3.36
3	6.25	-56.02(0.68)	4.39(0.26) 3.78(0.20)	-1.86 -2.47	4.01(0.47) 3.55(0.32)	-2.24 -2.7
4	4.44	-48.38(0.97)	2.29(0.19) 2.11(0.16)	-2.15 -2.33	2.12(0.22) 1.94(0.18)	-2.32 -2.5
avg.			-1.63 -1.92		-1.85 -2.08	

Table 4.3: MicaZ distances from nodes 1, 2, 3, 4 to node a determined using the RSSI measurement (N = 50) and two models. Here FS = free space model, LNSM = log normal shadowing model, RSSI is the signal strength average, with the sample standard deviation (s) in parentheses. The first entry of two entries in each row is for $n = 2.7$, while the second entry is for $n = 3.0$. Distances are in m.

From node	$\tilde{d}_{i,a}(s)$	RSSI (s)	$d_{i,a}$ FS	$d_{i,a} - \tilde{d}_{i,a}$	$d_{i,a}$ LNSM	$d_{i,a} - \tilde{d}_{i,a}$
1	2.21	-62.58(1.37)	7.71(0.94) 6.29(0.69)	5.5 4.08	7.22(1.05) 5.97(0.90)	5.01 3.76
2	4.85	-65.48(1.68)	9.92(1.51) 7.88(1.07)	5.07 3.03	9.21(1.91) 7.35(1.26)	4.36 2.5
3	6.25	-55.16(0.71)	4.07(0.24) 3.54(0.19)	-2.18 -2.71	3.79(0.38) 3.32(0.30)	-2.455 -2.93
4	4.44	-63.72(2.41)	8.66(2.45) 6.96(1.71)	4.22 2.52	8.11(2.28) 6.53(1.80)	3.67 2.09
avg.			3.15 1.73		2.65 1.36	

Table 4.4: Table of estimated static position.

Coordinate	Accurate Position $\tilde{c}$	c_F	$c_F - \tilde{c}$	c_L	$c_L - \tilde{c}$
x	50.62	273.21	222.59	260.11	209.49
y	173.07	280.24	107.17	291.36	118.29
z	90.72	251.67	160.95	354.36	263.64
length	201.86	465.31	294.85	527.37	356.91

With the free space propagation model, we obtained

$\hat{\mathbf{x}} = [273.21, 280.24, 251.67]$ cm, $\hat{\mathbf{r}} = [-259.80, -227.27, -240.38, -302.19]$ cm, $\hat{\sigma}_0^2 = 18.74$,

and

$$\mathbf{C}_{\mathbf{x}} = \begin{bmatrix} 522318.56 & 58912.02 & -341336.47 \\ 58912.02 & 149940.42 & -16435.82 \\ -341336.47 & -16435.82 & 430012.38 \end{bmatrix} \text{cm}^2 \tag{4.2.1}$$

We use equation (4.1.3) to compute the error ellipsoid, and find the eigenvalue for each axis is $\lambda_1 = 815966.78$, $\lambda_2 = 16933.02$ and $\lambda_3 = 169371.56$ so we obtain:

$$\sigma_x = \sqrt{\lambda_1} = 903.31 \text{ cm} \tag{4.2.2}$$

$$\sigma_y = \sqrt{\lambda_2} = 130.13 \text{ cm} \tag{4.2.3}$$

$$\sigma_z = \sqrt{\lambda_3} = 411.55 \text{ cm} \tag{4.2.4}$$

We use the same method as in equation (4.1.1) to test the quadratic form of the residuals with equation (4.2.5), i.e.

$$\frac{\nu\hat{\sigma}^2}{\chi^2_{\nu,\frac{\alpha}{2}}} \leq \sigma_0^2 \leq \frac{\nu\hat{\sigma}^2}{\chi^2_{\nu,1-\frac{\alpha}{2}}} \tag{4.2.5}$$

where ν is degree of freedom which is equal to 1 in our case. For $\alpha = 0.01$, $1 - \frac{\alpha}{2} = 0.995$, $\frac{\alpha}{2} = 0.005$, so $\chi^2_{\nu,\frac{\alpha}{2}}$ is equal to 7.87944 and $\chi^2_{\nu,1-\frac{\alpha}{2}}$ is equal to 0.00004, and equation (4.2.5) becomes:

$$\frac{18.74}{7.87944} \leq 1 \leq \frac{18.74}{0.00004} \tag{4.2.6}$$

which is

$$2.38 \leq 1 \leq 468500 \tag{4.2.7}$$

Equation (4.2.7) failed due to inaccurate estimation of σ^2. As long as we obtain accurate estimates for σ^2, then $\hat{\sigma}_0^2$ will be close to 1, so that equation (4.2.5) will be true. The same situation happened when we used the log normal shadowing model.

Using the log normal shadowing model, we obtain:

$\hat{\mathbf{x}} = [260.11, 291.36, 354.36]$ cm, $\hat{\mathbf{r}} = [-194.96, -167.84, -201.24, -207.16]$ cm, $\hat{\sigma}_0^2 = 9.91$,

and

$$\mathbf{C_x} = \begin{bmatrix} 229154.96 & 18224.62 & 302115.36 \\ 18224.62 & 80640.34 & 88677.80 \\ 302115.36 & 88677.80 & 746412.78 \end{bmatrix} \text{cm}^2$$

Then we use the same equation (4.1.3) to compute the error ellipsoid: Where the eigenvalue for each axis is $\lambda_1 = 58081.55$, $\lambda_2 = 103093.38$ and $\lambda_3 = 895033.15$ so we obtain:

$$\sigma_1 = \sqrt{\lambda_1} = 241.00 \text{ cm} \tag{4.2.8}$$

$$\sigma_2 = \sqrt{\lambda_2} = 321.08 \text{ cm} \tag{4.2.9}$$

$$\sigma_z = \sqrt{\lambda_3} = 946.06 \text{ cm} \tag{4.2.10}$$

We use the same method in equation (4.1.3) to test the quadratic form of the residuals with equation (4.2.11).

$$\frac{\nu\hat{\sigma}^2}{\chi^2_{\nu,\frac{\alpha}{2}}} \leq \sigma_0^2 \leq \frac{\nu\hat{\sigma}^2}{\chi^2_{\nu,1-\frac{\alpha}{2}}} \tag{4.2.11}$$

where ν is degree of freedom which is equal to 1 in our case. For $\alpha = 0.01$, $1 - \frac{\alpha}{2} = 0.995$, $\frac{\alpha}{2} = 0.005$, so $\chi^2_{\nu,\frac{\alpha}{2}}$ is equal to 7.87944 and $\chi^2_{\nu,1-\frac{\alpha}{2}}$ is equal to 0.00004, we obtain equation (4.2.12) as follows:

$$\frac{9.91}{7.87944} \leq 1 \leq \frac{9.91}{0.00004} \tag{4.2.12}$$

which is

$$1.26 \geq 1 \leq 247750 \tag{4.2.13}$$

$\mathbf{C_x}$ is the covariance matrix, it represents the estimate of errors in (x, y, z). For the free space model, the estimated errors in (x, y, z) are (722.7, 387.2, 655.75) cm, and for the log normal shadowing model, the estimated errors in (x, y, z) are (478.70, 283.97, 863.95) cm.

The large errors occurred because of the inaccuracy of distances determined from RSSI values based on the propagation models we used.

In the next chapter, we investigate computing the position of a moving node.

4.3 Two Points Testing

In this test, the moving node is placed twice on a line parallel to the y-axis. The first position is $\mathbf{x_1}$ = (71, 173.4, 76.2) cm, and the second position is $\mathbf{x_2}$ = (71, 295.3, 76.2) cm. Figure 4.5 shows the architecture of our two points test.

Table 4.5: Table of single RSSI values received at $\mathbf{x_1}$ and $\mathbf{x_2}$, along with their estimated distances using the FSPM with $n = 2.1$.

$\mathbf{x_1}$ (cm)			$\mathbf{x_2}$ (cm)		
node	RSSI value	distances	node	RSSI value	distances
1	-2	203	1	-3	221
2	-10	402	2	-8	339
3	-10	402	3	-6	286
4	-12	476	4	-11	438

After processing using Algorithm 1, Table 4.6 shows the estimated position of $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$ and their difference with the known positions $\mathbf{x_1}$ and $\mathbf{x_2}$.

The straight line connecting the two estimated points $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ passes

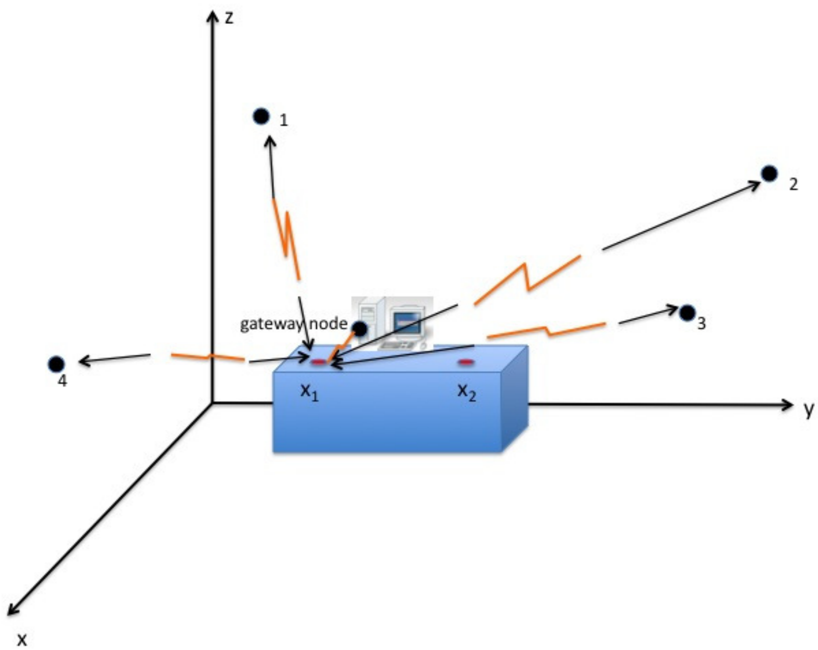


Figure 4.5: Architectural diagram for the two points test.

Table 4.6: Table of estimated position1 and position2, unit in this table are all in cm.

Coordinate	$\mathbf{x_1}$	$\hat{\mathbf{x}}_1$	$\mathbf{x_1} - \hat{\mathbf{x}}_\mathbf{1}$	$\mathbf{x_2}$	$\hat{\mathbf{x}}_2$	$\mathbf{x_2} - \hat{\mathbf{x}}_\mathbf{2}$
x	71	-441.25	512.25	71	113.11	-42.11
y	173.4	-56.57	229.97	295.3	278.73	16.57
z	76.2	481.28	-405.28	76.2	103.66	-27.66

through the plane formed by the x-axis and y-axis. The line connecting the actual measured points $\mathbf{x_1}$ and $\mathbf{x_2}$ is parallel to the y-axis. The large error in distance clearly results in large estimated position error.

Chapter 5

Dynamic Testing

Chapter 4 presented the results of testing our position estimation method in a static situation. In this chapter, we used a model train to carry the moving node that moves around a mobile wireless sensor network testbed track, which tests the ability of our positioning method for mobile objects. The train can go both forward and backward, and we can control the speed of the train velocity. Up to six stationary nodes were positioned outside the testbed track in the corners of the lab. The detailed layout of the test track is explained in section 5.4.

5.1 Time Synchronization

To make the stationary nodes communicate with the moving node one after the other, we synchronized the stationary nodes using a beacon node, so that stationary nodes can send messages to the moving node at an interval of δ seconds (e.g. 10 to 50 ms).

5.1.1 Flooding Time Synchronization Protocol

The Flooding Time Synchronization Protocol (FTSP) is one of the most widely used time synchronization methods in wireless sensor network applications. Delays in radio message transmission in WSNs will affect the precision of time synchronization. These delays [23] are:

(1) Send Time: the time used to assemble the packet on the transmit node.

(2) Access Time: the time the packet waits for access to the transmit channel.

(3) Transmission Time: the time for the transmit node to send the whole packet.

(4) Propagation Time: the time for the transmitted packet to reach the receive node.

(5) Reception Time: the time for the receive node to receive the whole packet.

(6) Receive Time: the time for the receive node to process the packet.

(7) Interrupt Handling Time: the time required for the radio chip to send a request until the microcontroller gives a response. This delay may be less than a few microseconds at the beginning, but it can grow larger in later processing.

(8) Encoding Time: the time for the radio chip to encode the the binary data into electromagnetic waves, and then send them to the microcontroller and

have the microcontroller raise an interrupt.

(9) Decoding Time: the time for a radio chip to decode the received massage until it raises an interrupt indicating it finishes.

(10) Byte Alignment Time: the delay that occurred because of different byte alignment in the sender and the receiver.

Figures 5.1 and 5.2 show all the delays mentioned above during the process to transmit a packet.

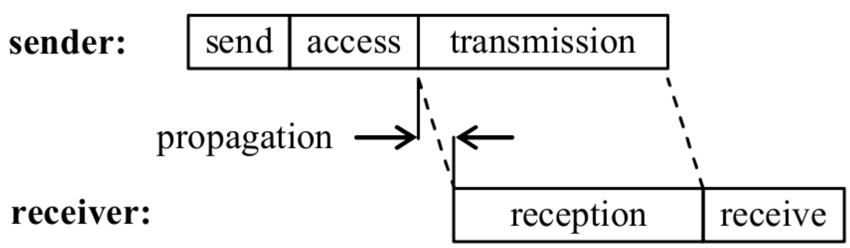


Figure 5.1: Decomposition of the message delivery delay over a wireless link (from [23]).

Time at each node (without FTSP) is based on the natural frequency of the node's crystal clock. The FTSP effectively reduces the interrupt handing and encoding/decoding times by using a MAC layer time-stamped massage at both send and receive sides. A beacon node acts as a root, and broadcasts its local time as global time to synchronize all stationary nodes. Based on the experiment in [23], the message delivery delay of FTSP in a typical WSN is less than 1μs for 300 meters. Figure 5.3 shows the wiring of the `TestFtsp` application.

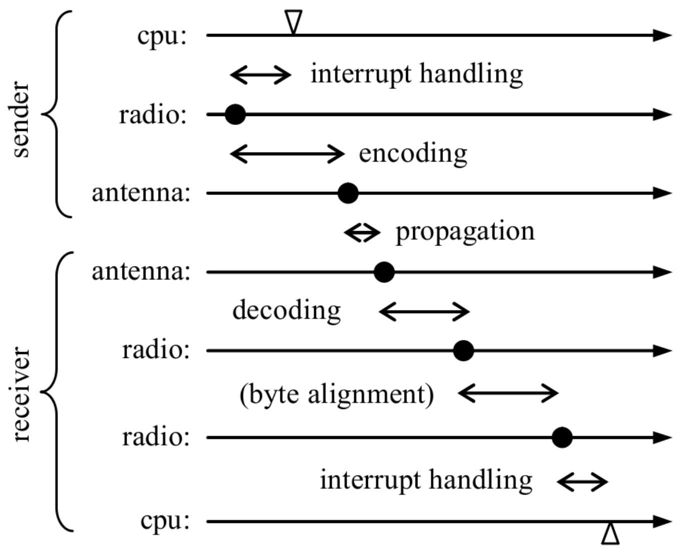


Figure 5.2: An idealized point (such as the end of one byte of a massage) transmits in the software, hardware and physical layer on both sides. Each line represents a time line, the dots represent the time instance and the triangles on two cpu time lines represent the time when the cpu makes the time-stamps (from [23]).

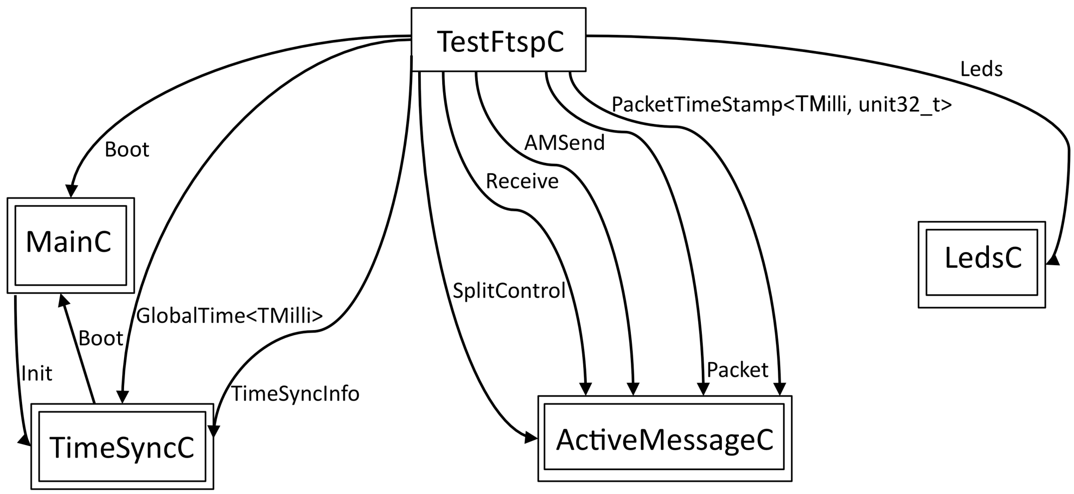


Figure 5.3: Wiring of the `TestFtsp` application (from [2]).

5.1.2 `TestFtsp` Message Structure

The data rate of the CC2420 chip is 250 Kbps. We used the `TestFtsp` message structure [23] to determine the RSSI value from each stationary node at the moving node. The message structure for the broadcast message is in file `TestFtsp.h` as shown in Appendix A.3. `TestFtsp` messages are sent by stationary nodes to the moving node. The data format for the `TestFtsp` message is shown in Figure 5.4. The `counter` in a `TestFtsp` message comes from the `RadioCountToLeds` message, which is sent by the beacon node to each stationary node. Each `TestFtsp` message is 39 bytes, or 312 bits long, which requires at least 1.248 ms to transmit at 250 Kbps.

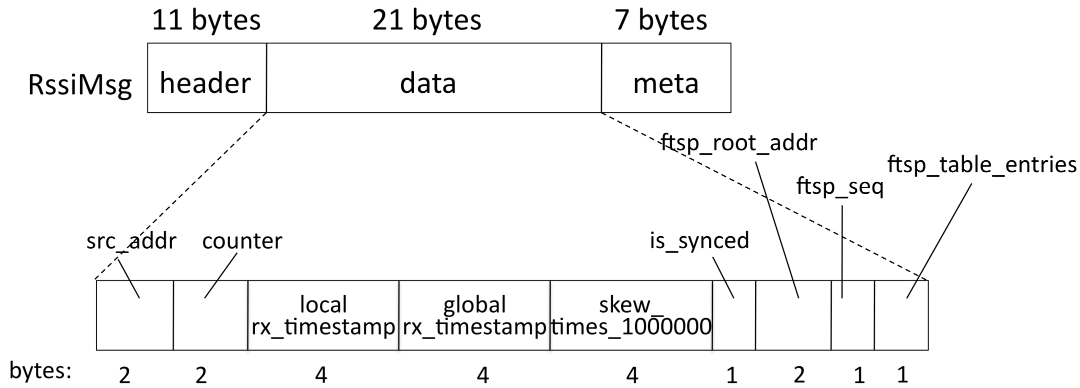


Figure 5.4: Structure of the TestFtsp message.

5.1.3 Beacon Node Algorithm

The beacon node, as explained in section 5.1, is an important device for time synchronization of the stationary nodes. Here we give a detailed description of the beacon node algorithm.

5.1.3.1 Radio Count Message Structure

`RadioCount` messages are sent from the beacon node to stationary nodes. Figure 5.5 shows the detailed `RadioCount` message structure that contains 20 bytes of data, with the payload being a two byte `counter`. The `RadioCountToLeds.h` file is given in Appendix A.1. The program running on the beacon node is called `RadioCountToLedsC.nc`, and is given in Appendix A.2. Algorithms 2 and 3 summarize the main logic of the `RadioCountToLeds` program.

The beacon node sends `RadioCount` messages to the stationary nodes one after the other in the order of the stationary node id with an interval of δ

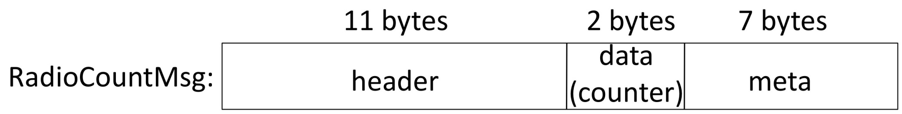


Figure 5.5: Structure of the `RadioCount` message.

ms. Figure 5.6 shows the pattern of sending up to `K` `RadioCount` messages. The interval is controlled by a countdown timer, which is a binary precision millisecond level timer [31], where one second equals to 1024 binary milliseconds. `K` is the maximum number of stationary nodes, and `k` is the number of existing stationary nodes.

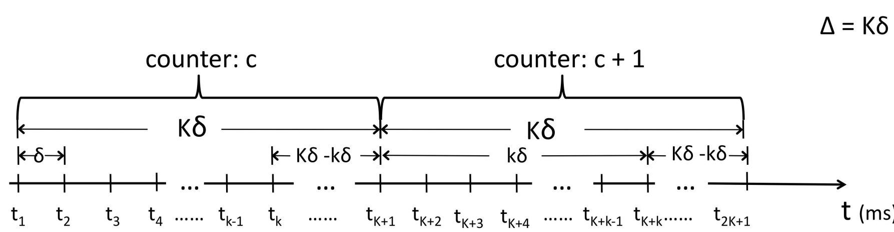


Figure 5.6: `RadioCount` message pattern transmitted by the beacon node.

Algorithm 3 shows how the `counter` remains the same for `K` messages in a row. The `RadioCount` message `rcm` is sent to the stationary node id `addr`.

Algorithm 2: Beacon Node Boot Event E_B

Trigger: E_B invoked when the beacon node boots up.
Result: Initialize the node's index
count ← 0;
addr ← 0;
initialize `RadioCountToLeds` message (counter ← 0);

Algorithm 3: Beacon Node (`RadioCount` Message) Send Event E_S

Trigger: E_S invoked by a countdown timer with interval of δ ms.
Result: Send `RadioCount` message to stationary nodes.

```
count++;
addr ← count % K; //stationary node id to send to
if (addr == 0) then
|  counter ++;
end
addr ← addr + offset;
rcm.counter ← counter;
if (call AMSend.send(addr, rcm) == SUCCESS) then
|  locked ← TRUE;
end
```

5.1.4 Stationary Node Algorithm

The `TestFtspC.nc` program (see Appendix A.4) sends `TestFtsp` messages when a beacon node `RadioCount` message (see section 5.1.3) is received. The beacon node algorithm is discussed in section 5.1.3. Algorithm 4 shows how the stationary nodes synchronize with the `GlobalTime` from the beacon node. The main method `local2Global(uint32_t *time)` (see [3]) converts the local time into the corresponding global time using the following equation:

$$\text{globalTime} = \text{localTime} + \text{offset} + \text{skew} * (\text{localTime} - \text{syncPoint}) \quad (5.1.1)$$

where `offset`, `skew` and `syncPoint` are computed using the `TimeSyncInfo` interface povided by the TinyOS 2.x FTSP library.

Algorithm 4: Stationary node (`TestFtsp` Message) Send Event E_S

Trigger: E_S invoked by receipt of a `RadioCount` message from the beacon node.
Result: Send `TestFtsp` message to moving node.
//stationary node synchronized with beacon node
tfm.src_addr ← TOS_NODE_ID;
tfm.counter ← rcm.counter;
tfm.local_rx_timestamp ← rxTimestamp;
tfm.is_synced ← GlobalTime.local2Global(&rxTimestamp);
tfm.global_rx_timestamp ← rxTimestamp;
tfm.skew_times_1000000 ←
(uint32_t)TimeSyncInfo.getSkew()*1000000UL;
tfm.ftsp_root_addr ← TimeSyncInfo.getRootID();
tfm.ftsp_seq ← TimeSyncInfo.getSeqNum();
tfm.ftsp_table_entries ← TimeSyncInfo.getNumEntries();
if (AMSend.send(tfm) == SUCCESS) **then**
| locked ← TRUE;
end

5.2 Real-Time Message Passing Architecture

5.2.1 `MULTI_RSSI` Message Structure

In our experiment, the moving node collects RSSI values from TestFtsp messages sent from up to eight stationary nodes, and combines them into one `MULTI_RSSI` message that is then sent to the gateway node. Figure 5.7 shows the structure of the `MULTI_RSSI` message. Appendix A.5 gives the `MultiRssi.h` file that defines this message structure.

The data fields of the `MULTI_RSSI` message contain (a) one byte `no_of_vals` indicating the numbers of RSSI values contained in this message, (b) two byte `counter` indicating the sequence number of this message, (c) one byte for each `nodeID` indicating which node the RSSI value is from, (d) two bytes to store each `rssi value`, and (e) four bytes for `rssi time` which indicates the local time that the `TestFtsp` messages arrived at the moving node. The `rssi time` is the local time on the moving node which is not synchronized with the stationary nodes. Recording `rssi time` permits the calculation of the difference in arrival times for messages from each stationary node.

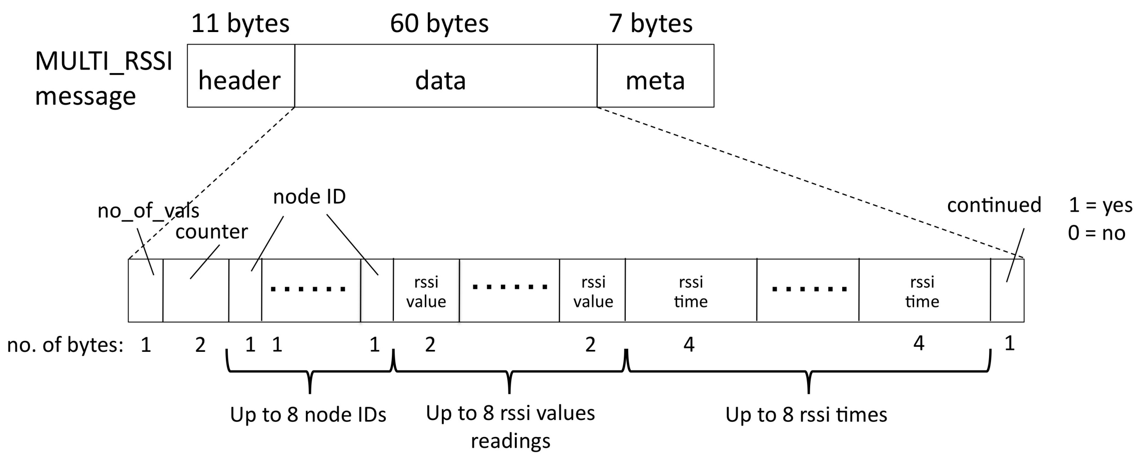


Figure 5.7: Structure of the `MULTI_RSSI` message.

5.2.2 Moving Node Algorithm

Algorithms 5 and 6 summarize the operation of the moving node algorithm. Appendix A.6 gives the complete `MultiRssiC.nc` program corresponding to these algorithms.

Figure 5.8 shows a state diagram indicating how the `MULTI_RSSI` message is constructed and sent in a state diagram. Variable `current` (initially 0)

Algorithm 5: Moving Node Boot Event E_B

Trigger: E_B invoked when the moving node boots up and received a `TestFtsp` message
Result: Initalize the node's index
current ← 0;
no_of_vals ← 0;
initialize `MULTI_RSSI` message (index ← 0);
continued ← 0;
rssi_time_save ← 0;
rssi_val_save ← 0;
nodeid_save ← 0;

keeps track of the `current set` of RSSI observations. The state `accumulate` represents the fact that we are adding RSSI observations s_i to the set S. Once we see a message with different sequence number, we enter the `check` $|S|$ state to check if we have seen at least three `TestFtsp` messages from different nodes. If we have three or more messages from different nodes, we send a `MULTI_RSSI` message containing the set S of RSSI observations and save the first message s_0 from the new set. Otherwise, we reset S to the empty set $\varnothing$, save the first message s_0 from the new set and return to the `accumulate` state.

5.3 Test Track

As part of the mobile WSN testbed, a short test track (143 cm long) was constructed (see Figure 5.9). With regard to the origin (the northwest floor corner in the lab), we obtained the positions of the test track ends. The

Algorithm 6: Moving Node Message Receive and Send Event E_R

```
Trigger: E_R invoked by arrival of a TestFtsp message
Data: TestFtsp message (see Figure 5.4)
Result: Push RSSI values into MULTI_RSSI message
get counter and src_addr from TestFtsp message;
rssi_time ← LocalTime.get( );
//accumulate state
if (counter == current) then
    if (rssi_time_save != 0 && rssi_val_save != 0 && nodeid_save !=
    0) then
        rssi_time[index] ← rssi_time_save;
        rssi_val[index] ← rssi_val_save;
        nodeID[index] ← nodeid_save;
        index ++; no_of_vals ← index;
        rssi_time_save ← 0;
        rssi_val_save ← 0;
        nodeid_save ← 0;
    end
    rssi_time[index] ← LocalTime.get( );
    rssi_val[index] ← getRssi( );
    nodeID[index] ← src_addr;
    index ++; no_of_vals ← index;
end
// check |S| state
else if index >=3 then
    seq ++; continued ← 0;
    if (call AMSend.send(MultiRssi message) == SUCCESS) then
        index ← 0; no_of_vals ← 0;
    end
    rssi_time_save ← LocalTime.get( );
    rssi_val_save ← getRssi( );
    nodeid_save ← src_addr;
    current ← counter;
end
else
    current ← counter;
    index ← 0; rssi_time_save ← LocalTime.get( );
    rssi_val_save ← getRssi( );
    nodeid_save ← src_addr;
end
```

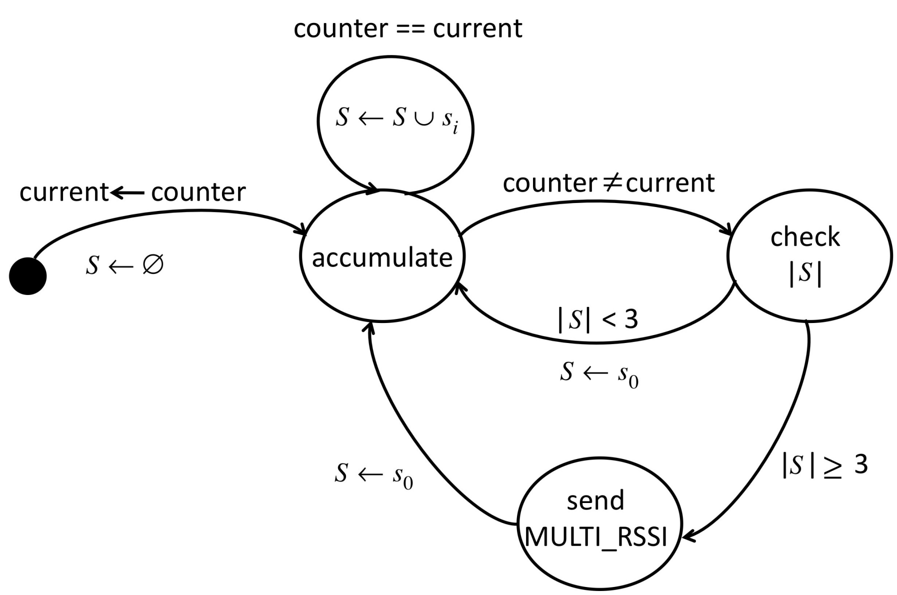


Figure 5.8: State diagram for the Moving node algorithm 6.

start is located at $[303.34, 569.64, 76.50]$, and the end of the test track is $[409.24, 569.64, 76.5]$.

Figure 5.9: Test track for initial testing.

The train with the flatbed trailer is 39 cm long, so the distance that the train can move is 104 cm. We placed a moving node (TelosB mote with two AA batteries) on the trailer. Using the maximum speed, it took 3.6s for the train to run from one end to the other; the maximum speed of the model train carrying the moving node is 0.289m/s.

As discussed below, we determined that 250 ms is the smallest interval between moving node position estimate calculations. With this rate of 4 position estimations per second, the train moves 0.07225m in 250 ms.

We designed an RSSI message transmission pattern to work in a dynamic environment with one moving node and k stationary nodes. The interval between RSSI messages is δ ms (e.g. 10 to 25 ms), and a group of k RSSI messages is repeated every Δ ms (e.g. 250 ms). The pattern is shown in Figure 5.10.

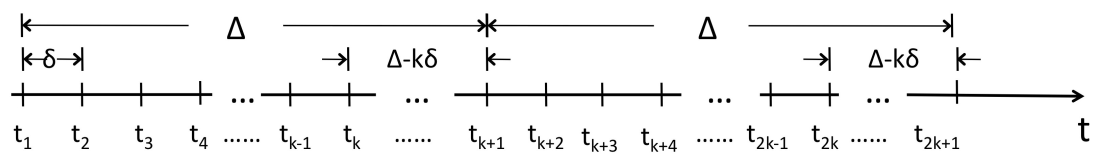


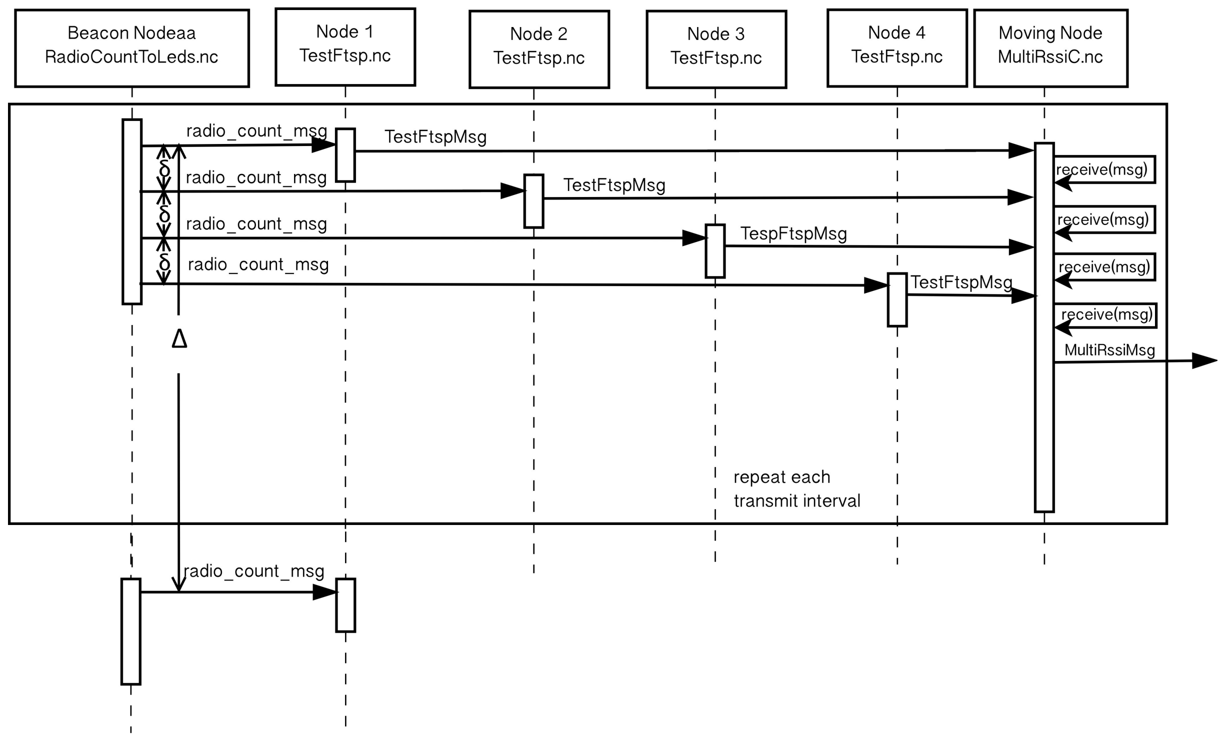
Figure 5.10: RssiMsg message transmit pattern.

Figure 5.11 shows how we combined the time synchronization messages with the position estimation for four stationary nodes. The beacon node broadcasts a Flooding Time Synchronization Protocol (FTSP) packet every second. Then, the stationary nodes send RssiMsg messages to the moving node in a synchronized fashion.

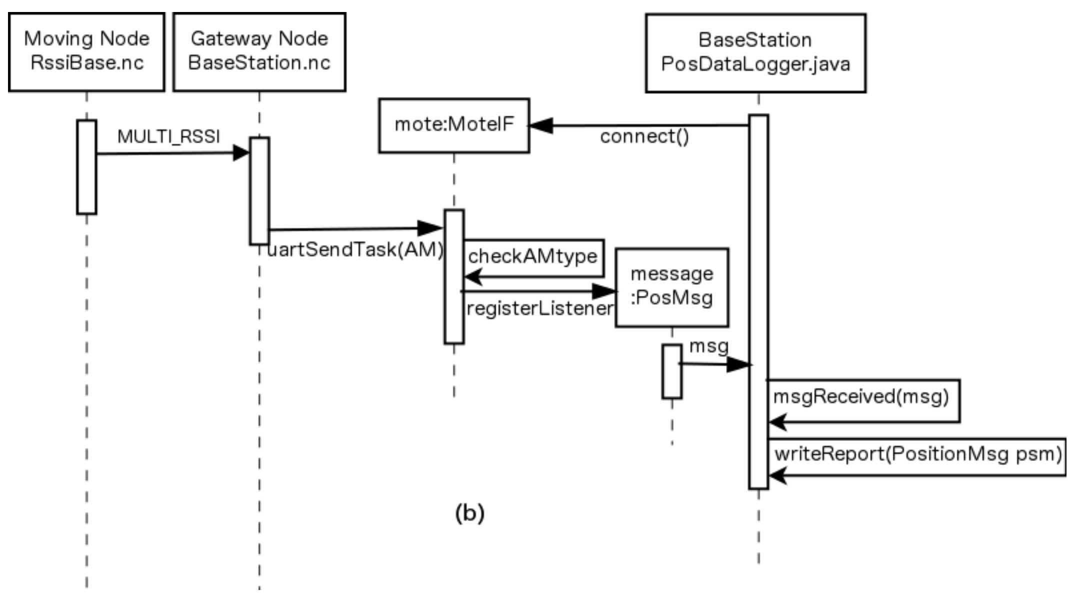
5.4 Test Architecture

We have an entire model train track near the ceiling in the Wireless Communication and Sensor Network lab (ITB214), as shown in Figure 5.12. Table 5.1 shows the coordinates of the stationary nodes.

We measured the precise location of eight points $p_1, \cdots, p_8$ on the inner



(a)



(b)

Figure 5.11: Sequence diagram for time synchronization and sending RSSI messages. (a) stationary nodes send messages to the moving node, (b) moving node sends message to the gateway and base station.

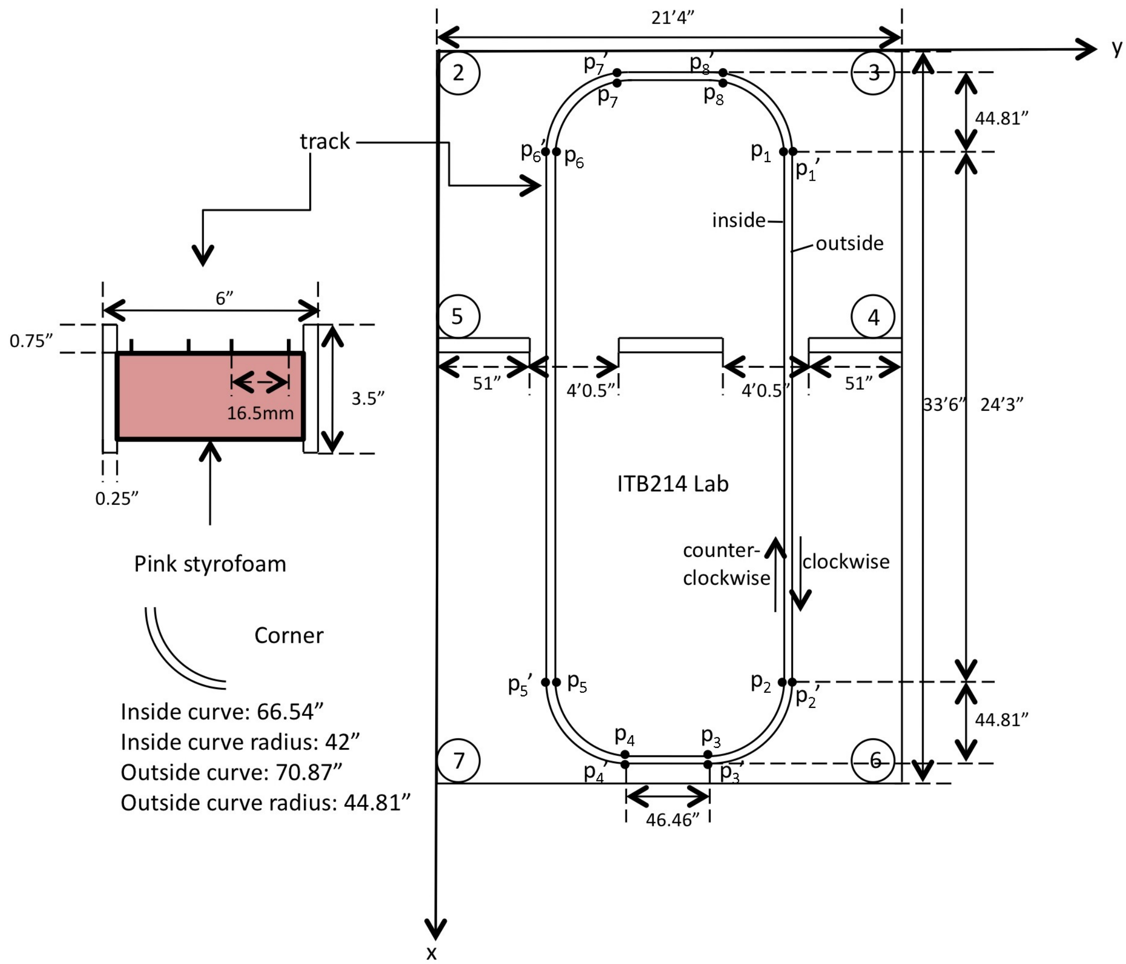


Figure 5.12: Physical layout of the wireless sensor network testbed.

Table 5.1: Coordinates of the stationary nodes.

Point	Coordinates (cm)
2	(6.4, 6.1, 229)
3	(6.3, 643.4, 203)
4	(441, 643.2, 224)
5	(440.4, 6, 224)
6	(1008.8, 643.2, 225.5)
7	(1008.8, 6.1, 223)

track, and eight locations $p'_1, \cdots, p'_8$ on the outer track. The coordinates of these 16 points are shown in Tables 5.2 and 5.3. Positions p_1 and p'_1 are the start and end points for one circuit around the track. A Java application was written called `mouseListener` that records the start time t_1 and end time t_3 of one round trip through a mouse right key click. The variable t_2 is the time the train is at any other position except p_1 and p'_1, so we have $t_1 \leq t_2 \leq t_3$. Based on the length of the track, the time to finish one circuit and eight known positions at the track joints, the Java application `posOnTrack` (see Appendix A.7) is used to get the position of the train at any time t_2 between t_1 and t_3 time. It should be noted that we assume that the velocity during one round trip is constant. The outer track length is 24.36m, and the inner track length is 23.92m.

Table 5.2: Coordinates of points on the inner track.

Point	Coordinates(cm)
p_1	(139, 490, 250)
p_2	(872, 491, 251)
p_3	(989, 383, 252)
p_4	(987, 261, 251)
p_5	(871, 162, 250)
p_6	(136, 162, 252)
p_7	(30, 267, 252)
p_8	(29, 386, 252)

Table 5.3: Coordinates of points on the outer track.

Point	Coordinates(cm)
p'_1	(139, 496, 250)
p'_2	(872, 497, 251)
p'_3	(995, 383, 252)
p'_4	(993, 261, 251)
p'_5	(871, 156, 250)
p'_6	(136, 156, 252)
p'_7	(24, 267, 252)
p'_8	(23, 386, 252)

5.4.1 Software Architecture

Figures 5.13 and 5.14 show an overall architecture diagram of our moving node position estimation (`MobiPos`) system. `MobiPos` comprises four Tinyos (.nc) programs and four Java programs. The four object classes shown in Figure 5.14 are JAVA programs.

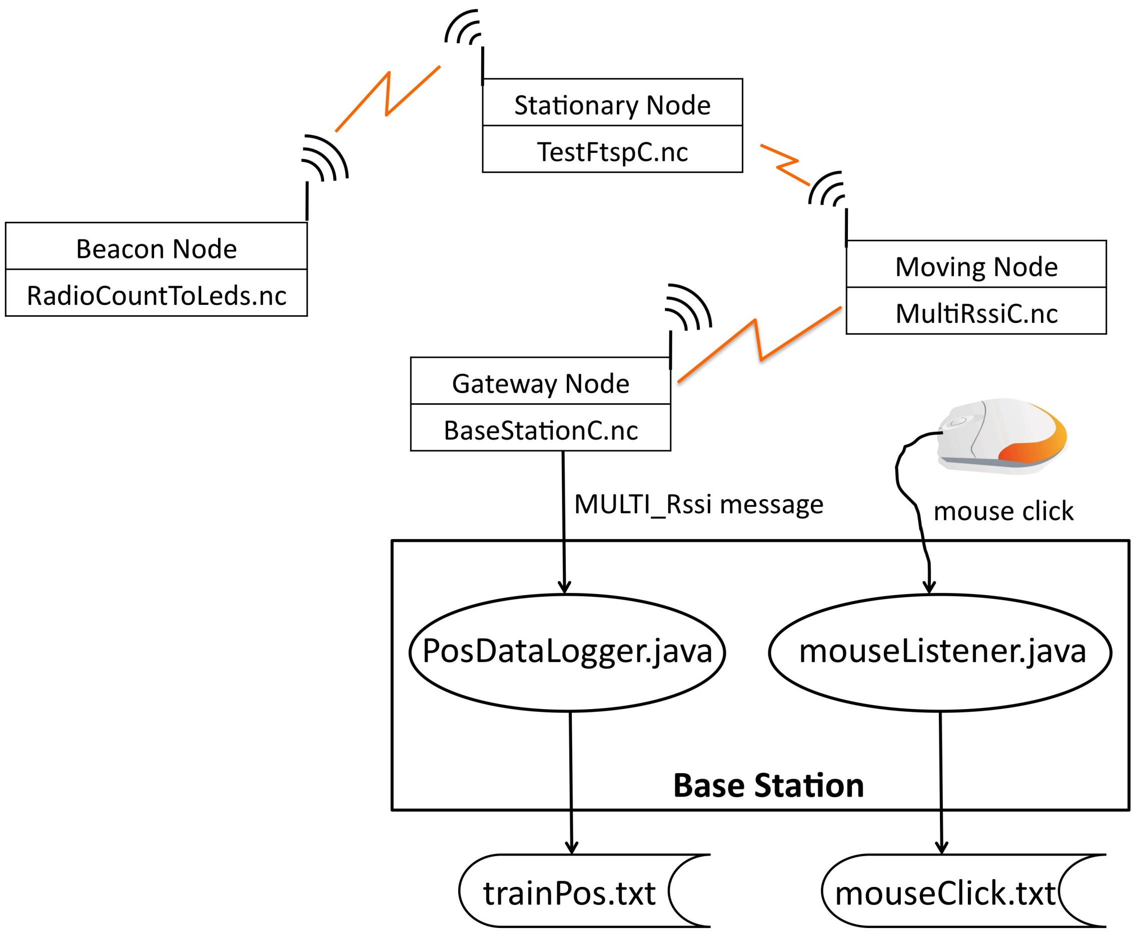


Figure 5.13: `MobiPos` software architecture diagram.

Figure 5.14 shows the data flow in the `MobiPos` application.

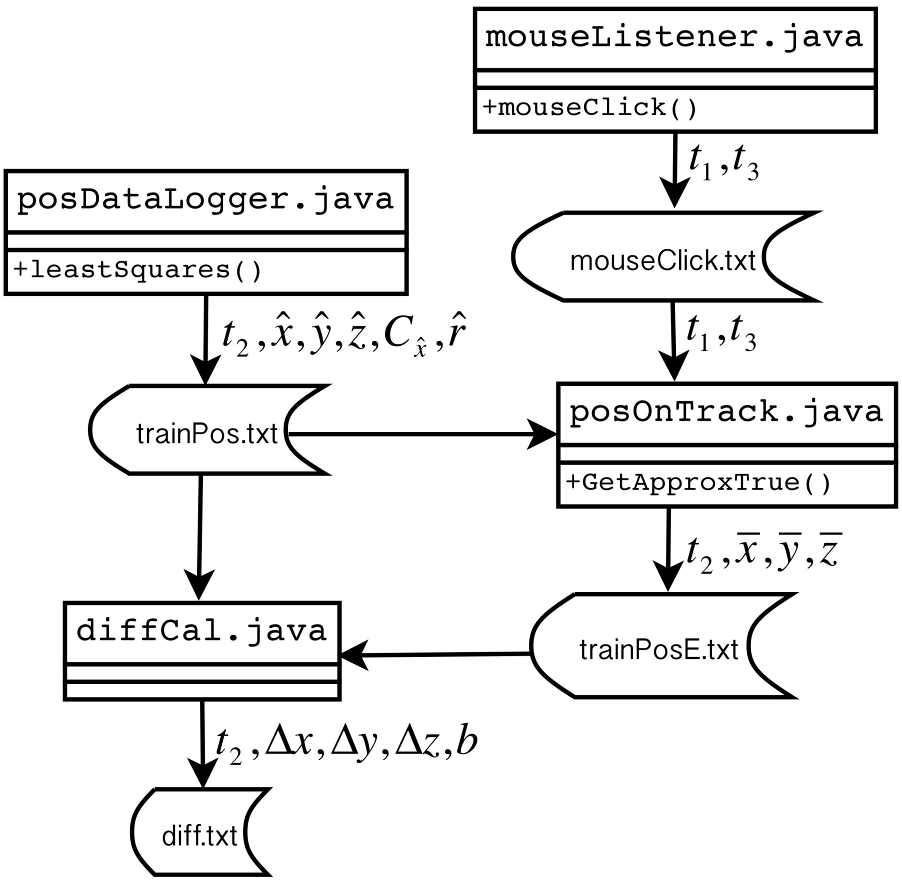


Figure 5.14: Data flow on the base station for computing estimated positions of the moving node at the time the `MULTI_RSSI` message was received. All four object classes shown here are JAVA programs.

Object class mouseListener uses any click of the mouse to record the start and end times of the moving node for one round trip. The posOnTrack object class uses the t_1, t_2 and t_3 times to compute the approximate true train position $(\bar{x}, \bar{y}, \bar{z})$ on the track at time t_2. Appendix A.7 contains the complete source code for the getApproxTrue() method of the posOnTrack object class. Object class posDataLogger implements the LeastSquares method given in Chapter 2. The complete source code for the leastSquares() method is given in Appendix A.8. Method getArrayD() reads the MULTI_RSSI messages to extract the RSSI values, converts them to distance observations $(d_1, \cdots, d_k)$, and passes these observations to the LeastSquares method. The LeastSquares method computes the estimated position $(\hat{x}, \hat{y}, \hat{z})$ of the moving node, along with the associated covariance matrix $\mathbf{C}_{\hat{\mathbf{x}}}$ and residuals $\hat{\mathbf{r}}$. Finally, we use diffCal to compute the position difference$(\Delta x, \Delta y, \Delta z)$ = $(\bar{x}, \bar{y}, \bar{z})$ - $(\hat{x}, \hat{y}, \hat{z})$, and the difference b between the estimated and approximate true positions as follows:

$$b = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{5.4.1}$$

The complete diffCal program is given in Appendix A.9.

Chapter 6

Dynamic Testing Experimental Results

This chapter shows the experimental result of four different dynamic tests, as illustrated in Table 6.1.

Table 6.1: Four different dynamic tests.

Test	no. of stationary nodes	packets received	time duration
test 1	4	131	128 s
test 2	6	131	127 s
test 3	6	282	276 s
test 4	6	64,443	17.5 h

6.1 Data Noise

In our experimental testing, we found a large variation in observed RSSI values. Time variations of 2.4 GHz wireless signal noise in ITB214 seemed to affect the results significantly. Thus, we need to obtain reasonable estimates for the variance σ_i^2 of distance observations d_i used in the weight matrix $\mathbf{P}$.

6.1.1 Stationary Train Noise Situation

With the moving node held stationary, we first estimated the variation in RSSI observations using the following process:

1) Place the moving node at p_1, and observe rssi_vals for the six stationary nodes for at least 50 observations. We used equation (3.13) to compute the distance d_i to each stationary node. This gives the average $\bar{d}_i$ and variance $\sigma_{i,1}^2$ for the distance from p_1 to each stationary node i.

2) Place the moving node at p_5, and observe at least 50 times the `rssi_val`. This gives the average $\bar{d}_i$ and variance $\sigma_{i,5}^2$ for the distance from p_5 to each stationary node i.

3) Take the average $\sigma_i^2 = \frac{\sigma_{i,1}^2 + \sigma_{i,5}^2}{2}$ as the weight matrix $\mathbf{P}$ values for $\frac{1}{\sigma_i^2}$.

Table 6.2 shows the average standard deviation of the estimated distance at each stationary node. These average variances were used to compute the weight matrix $\mathbf{P}$ value as shown in equation (2.0.9).

Table 6.2: Variance of the estimated distance at each stationary node.

node	$\bar{d}_1$	$\sigma_{i,1}^2$	$\bar{d}_5$	$\sigma_{i,5}^2$	σ_i^2 (cm^2)
2	618	1180	874	33262	17221
3	89	24	402	1366	695
4	328	1963	1187	582	1273
5	677	1882	266	715	1299
6	785	1883	1130	17372	9628
7	479	1445	221	173	809

6.1.2 Using Actual Residuals

The values in Table 6.2 are far smaller than the residuals $\hat{\mathbf{r}}$ computed using equation (2.0.14). To give a more reasonable estimate of the observed distance variances, we used the average residual $\bar{r}_i$ (equation 6.1.2) of up to 50 residuals $\hat{r}_i$ from test 2 (Table 6.10) for stationary node i. We estimate the weight matrix $\mathbf{P}$ values $\frac{1}{\sigma_i^2}$ by computing σ_i^2 as follows:

$$\sigma_i^2 = \frac{\sum_{j=1}^{m}(\hat{r}_{i,j} - \bar{r}_i)^2}{m-1} \tag{6.1.1}$$

where m is the number of observed distances to stationary node i in test 2, and

$$\bar{r}_i = \frac{\sum_{j=1}^{m}\hat{r}_{i,j}}{m} \tag{6.1.2}$$

The results of the above computations for the six stationary nodes in test 2 are shown in Table 6.3.

Table 6.3: Average residual $\bar{r}_i$ from test 2 for each stationary node.

node	m	$\bar{r}_i$	σ_i^2 (cm^2)
2	64	-369.43	8912.39
3	64	-285.99	676.93
4	64	-142.57	2517.13
5	64	-93.65	72.60
6	64	-256.77	2271.24
7	64	-339.77	358.62

6.2 Position Estimation with Four Stationary Nodes

We refer to the test with four stationary nodes as test 1. We tested the train moving clockwise (looking from on top) on the inside track. To estimate the average speed, we observed one circuit of the moving node start time (java time) of 1384889721693ms (11/19/2013 3:35:21 PM, 2013 GMT-4) and end time of 1384889849797ms (11/19/2013 3:37:29 PM, 2013 GMT-4). This gives an average speed of 0.19m/s with an inside track length of 23.92m. We received 131 valid `MULTI_RSSI` messages for a single circuit of the train around the track. As designed, Algorithm 6 only sends `MULTI_RSSI` messages with three or more RSSI values. All 131 messages sent during this 128 second test period were received as no sequence numbers are missing. As shown in Tables 6.4 and 6.5, we chose 10 `MULTI_RSSI` messages (every 13th one) to check the accuracy of our method.

`JAVA_TIME` represents the time the gateway node received the MULT_RSSI message from the moving node. `node_id` shows the stationary node number

Table 6.4: Data in the `MULTI_RSSI` message for test 1.

[JAVA_TIME]	[SEQ_NUM]	pos	[node_id]	[rssi_val]
1377603342649	186	1	[2 4 3 5]	[-10 -9 4 -16]
1377603355341	198	2	[2 4 3 5]	[-13 -12 2 -17]
1377603367050	210	3	[2 4 3 5]	[-12 0 -25 -12]
1377603378771	222	4	[2 4 3 5]	[-10 -14 -15 -17]
1377603390498	234	5	[2 4 3 5]	[-3 -11 -15 -6]
1377603402220	246	6	[2 4 3 5]	[-8 -11 -15 -8]
1377603413934	258	7	[2 4 3 5]	[-12 -6 -14 1]
1377603425677	270	8	[4 3 5 2]	[-22 -3 -3 3]
1377603437519	282	9	[4 3 5 2]	[-11 -16 -11 5]
1377603450070	294	10	[2 3 5]	[-11 -5 -6]

sending the `TestFtsp` message in the order `TestFtsp` messages were received.

Table 6.5: Estimated and approximate true positions for test 1.

[SEQ_NUM]	pos	$(\hat{x}, \hat{y}, \hat{z})$ cm	$(\bar{x}, \bar{y}, \bar{z})$ cm
186	1	N/A (Matrix is singular)	[134, 489, 249]
198	2	N/A (Matrix is singular)	[375, 490, 250]
210	3	N/A (Matrix is singular)	[598, 490, 250]
222	4	[-490, 158, -294]	[821, 490, 250]
234	5	[245, -165, 50]	[988, 385, 251]
246	6	[409, -48, -179]	[954, 180, 251]
258	7	N/A (Matrix is singular)	[740, 162, 250]
270	8	N/A (Matrix is singular)	[517, 162, 250]
282	9	[-26, -205, 352]	[292, 162, 251]
294	10	[471, 518, 400]	[62, 192, 252]

Figure 6.1 shows a plot of the estimated and approximate true positions computed using the least squares method given in Chapter 2. The weight matrix **P** values given in Table 6.3 were used for estimating all positions. Table 6.6 shows the computed estimate of residuals, along with the square

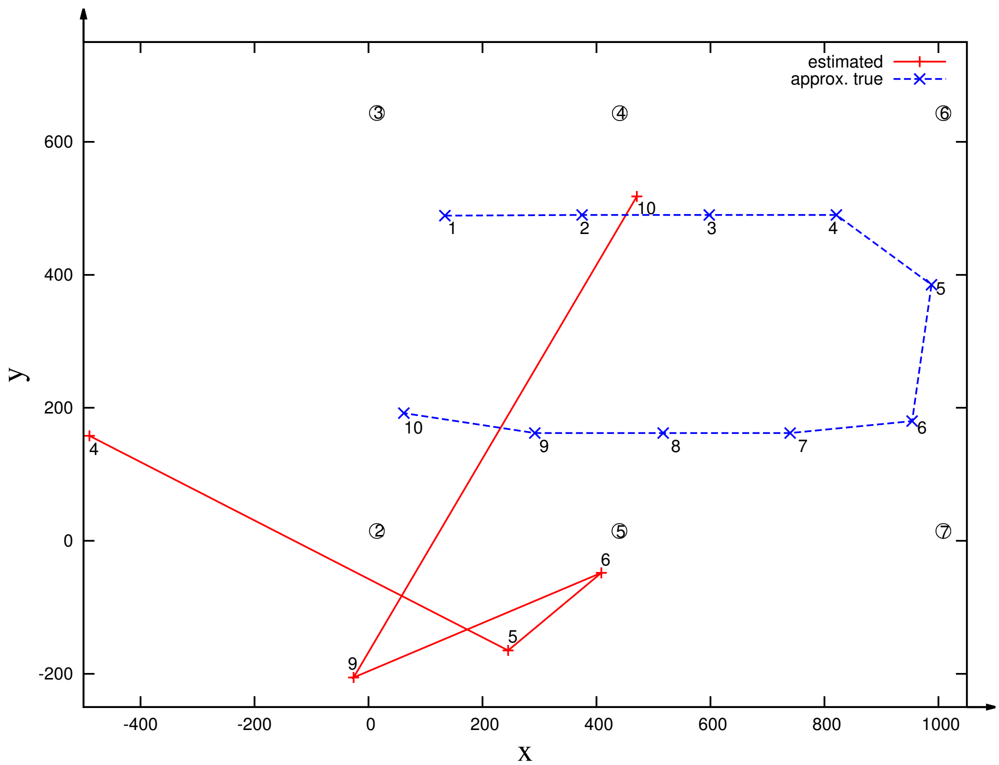


Figure 6.1: The positions from Table 6.5 plotted in 2d for test1.

root of the diagonal of covariance matrix $\mathbf{C}_{\hat{\mathbf{x}}}$ (see equation (2.0.12)). If these error estimates are reasonable, they should match the actual difference shown in Table 6.7.

Table 6.7 shows the calculated distance between the least squares estimated position $(\hat{x}, \hat{y}, \hat{z})$ and the estimated true position$(\bar{x}, \bar{y}, \bar{z})$ at the same time, using equation (5.4.1). For point 9, the difference b is smaller than 500 cm, for points 5, 6 and 10 b is smaller than 1000 cm, and for point 4, b is greater

Table 6.6: Data in diffCal.txt for test 1.

[SEQ_NUM]	pos	[node_id]	$\hat{r}(cm)$	$\hat{\sigma}_0^2$	$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$(cm)
186	1	[2 4 3 5]	N/A	N/A	N/A
198	2	[2 4 3 5]	N/A	N/A	N/A
210	3	[2 4 3 5]	N/A	N/A	N/A
222	4	[2 4 3 5]	[-139, -244, 180, 212]	2.5	[898, 568, 1170]
234	5	[2 4 3 5]	[-67, -183, 177, 73]	1.19	[273, 269, 507]
246	6	[2 4 3 5]	[-95, 134, 147, 72]	0.86	[325, 248, 226]
258	7	[2 4 3 5]	N/A	N/A	N/A
270	8	[4 3 5 2]	N/A	N/A	N/A
282	9	[4 3 5 2]	[-394, 127, 129, -342]	4.88	[729, 680, 1390]
294	10	[2 3 5]	[-111, -541, -527]	1.69	[301, 292, 3702]

than 1000 cm.

Table 6.7: Difference between trainPos and trainPosE for test 1.

[JAVA_TIME]	[SEQ_NUM]	pos	$\bar{x}-\hat{x}$	$\bar{y}-\hat{y}$	$\bar{z}-\hat{z}$	b (cm)
1377603342649	186	1	N/A	N/A	N/A	N/A
1377603355341	198	2	N/A	N/A	N/A	N/A
1377603367050	210	3	N/A	N/A	N/A	N/A
1377603378771	222	4	1311	332	544	1458
1377603390498	234	5	743	550	201	946
1377603402220	246	6	545	228	430	731
1377603413934	258	7	N/A	N/A	N/A	N/A
1377603425677	270	8	N/A	N/A	N/A	N/A
1377603437519	282	9	318	367	-101	496
1377603450070	294	10	-409	-326	-148	543

For the complete test 1 data, i.e. all 131 `MULTI_RSSI` messages, we found that 52 messages (40%) provided estimated positions, and 80 messages (60%) did not converge, (i.e. gave a "Matrix is singular." error message).

6.3 Position Estimation with Six Stationary Nodes

We refer to the first test with six stationary nodes as test 2. We tested the train moving clockwise (viewed from the top) on the inside track. To estimate the average speed, we observed one circuit of the moving node start time (java time) of 1384889721889ms (11/19/2013 3:35:21 PM, 2013 GMT-4) and end time of 1384889848846ms (11/19/2013 3:37:28 PM, 2013 GMT-4). This gives an average speed of 0.19m/s with an inside track length of 23.92m. We received 131 valid `MULTI_RSSI` messages for a single circuit of the train around the track. As designed, Algorithm 3 only sends `MULTI_RSSI` messages with three or more RSSI values. All 131 messages, with sequence numbers 203 to 333, sent during this 126.957 seconds test period were received as no sequence numbers are missing. We used the least squares approach described in Chapter 2, Algorithm 1 to compute the estimated position $(\hat{x}, \hat{y}, \hat{z})$ for each `MULTI_RSSI` message, we used $\sigma_i^2 = 62500$ for test 2. The time interval Δ was set to 1000 binary ms (1.024 s) with $\delta = 50$ binary ms ($= 50 * 0.001024$ s $=$ 0.0512 s). As shown in Tables 6.8 and 6.9, we chose 10 `MULTI_RSSI` messages (every 12th one) to illustrate of the test 2 results.

Figure 6.2 shows a plot of the five successful estimates compared to the approximate true positions.

Table 6.11 shows the calculated distance between the least squares estimated position $(\hat{x}, \hat{y}, \hat{z})$ and the estimated true position$(\bar{x}, \bar{y}, \bar{z})$ at the same time,

Table 6.8: A subset of 10 `MULTI_RSSI` messages in trainPos.txt for test 2.

[JAVA_TIME]	[SEQ_NUM]	[node_id]	[rssi_val]
1384889721889	203	[2 3 4 5 6 7]	[-10 5 3 -13 -14 0]
1384889733611	215	[2 3 4 5 6 7]	[-18 1 -10 -12 -2 -6]
1384889745329	227	[2 3 4 5 6 7]	[-8 -13 -12 -14 -3 -7]
1384889757051	239	[2 3 4 5 6 7]	[-14 -11 -3 -8 -9 -10]
1384889768771	251	[2 3 4 5 6 7]	[-9 -14 -9 4 2 -6]
1384889780485	263	[2 3 4 5 6 7]	[-9 -9 -7 -2 -11 -23]
1384889792213	275	[2 3 4 5 6 7]	[-10 -8 -1 3 -10 5]
1384889803927	287	[2 3 4 5 6 7]	[-18 -8 -5 0 0 2]
1384889815635	299	[2 3 4 5 6 7]	[4 -23 -7 -10 -21 -13]
1384889827363	311	[2 3 4 5 6 7]	[2 -8 -7 -2 -9 -10]

Table 6.9: Estimated and approximate true positions for test 2.

[SEQ_NUM]	pos	$(\hat{x}, \hat{y}, \hat{z})$	$(\bar{x}, \bar{y}, \bar{z})$
203	1	(330.77 482.50 214.62)	(139.00 490.00 250.00)
215	2	Matrix is singular	(361.91 490.30 250.30)
227	3	(826.32 370.76 -21.81)	(584.75 490.61 250.61)
239	4	(631.04 467.02 24.53)	(807.66 490.91 250.91)
251	5	Matrix is singular	(986.28 399.06 251.86)
263	6	(-157.61 570.99 -23.38)	(963.28 190.71 250.59)
275	7	Matrix is singular	(754.59 162.00 250.32)
287	8	Matrix is singular	(531.87 162.00 250.92)
299	9	Matrix is singular	(309.25 162.00 251.53)
311	10	(355.04 133.67 99.18)	(88.06 173.46 252.00)

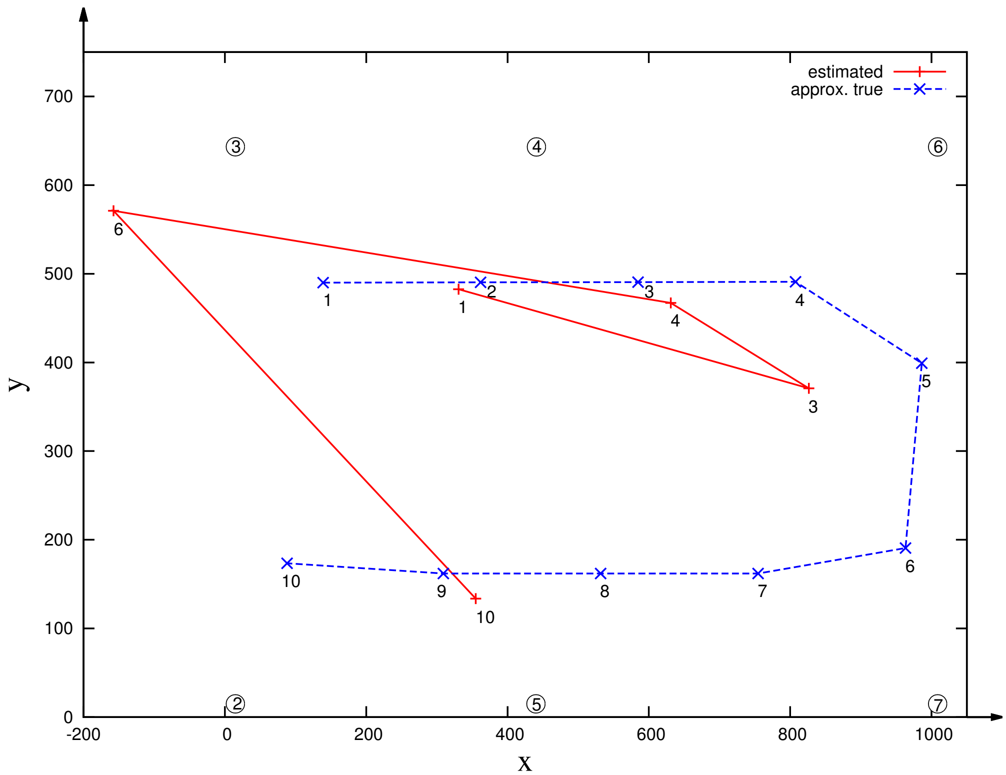


Figure 6.2: Positions in 2d for test2 (Table 6.9).

Table 6.10: Data in trainPosE.txt for test 2.

pos	$\hat{\mathbf{r}}$(cm)	$\hat{\sigma}_0^2$	$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm)
1	[-173, -366, -170, 167, 245, -696]	4.09	[311, 338, 495]
2	NA	NA	NA
3	[-451, -62, 211, 341, -133, -45]	2.08	[223, 259, 356]
4	[123, -6, -49, -57, 72, -30]	0.15	[57,66, 101]
5	NA	NA	NA
6	[-104, 246, -221, -610, -528, 1166]	11.37	[710, 738, 1252]
7	NA	NA	NA
8	NA	NA	NA
9	NA	NA	NA
10	[-1257, -835, -883, -1149, -906, -898]	32.06	[1351, 1613, 724]

using equation (5.4.1). For positions (pos) 3, 4 and 10, the difference b is smaller than 400 cm, for positions (pos) 1 and 6, b is greater than 1000 cm.

Table 6.11: Difference between trainPos and trainPosE for test 2.

[JAVA_TIME]	pos	$\bar{x}-\hat{x}$	$\bar{y}-\hat{y}$	$\bar{z}-\hat{z}$	b (cm)
1384889721889	1	-358.18	117.99	-1381.03	1431.59
1384889733611	2	NA	NA	NA	NA
1384889745329	3	-242	120	272	383.32
1384889757051	4	-177	-24	-226	288.12
1384889768771	5	NA	NA	NA	NA
1384889780485	6	1121	-380	274	1214.93
1384889792213	7	NA	NA	NA	NA
1384889803927	8	NA	NA	NA	NA
1384889815635	9	NA	NA	NA	NA
1384889827363	10	-267	40	152	309.88

For the complete test 2 data, i.e. all 131 `MULTI_RSSI` messages, we found that 72 messages (55%) provided estimated positions, and 59 messages (45%) did not converge, (i.e. gave a “Matrix is singular.” error message), the result is better than we use four stationary nodes (40% messages are converged). In addition, we discarded 8 "unsuccessful" estimated positions as these estimated residuals $\hat{\mathbf{r}}$ were always very large (>1000 m or <-1000 m), which gives a very large $\hat{\sigma}^2$ (always >5000). The complete input test 2 data is given in Appendix B.1. Appendix B.2 plots the individual distances observed for all 131 `MULTI_RSSI` messages. The noisy nature of distance observations derived solely from `RSSI` values is clearly evident in the Appendix B.2 plots.

For the 64 successful estimates, the average `b` was 503.34 cm, the average $\hat{\sigma}_0^2$ was 18.99, and the average $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) was [697.42, 855, 799.81] (Table 6.10). Table 6.12 shows the average `b` (cm), $\hat{\sigma}_0^2$ and $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) value for the 64 successful estimates from test 2.

Table 6.12: Average `b` (cm), $\hat{\sigma}_0^2$ and $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) for the 64 successful estimates from test 2.

	average value, N = 64
`b`	503.34
$\hat{\sigma}_0^2$	18.99
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[1,1]}$	697.42
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[2,2]}$	855
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[3,3]}$	799.81

6.3.1 Test 3

Test 2 used all apriori estimates of the variance of a single distance observation as $\sigma_i^2 = 62500$ cm^2. Test 3 uses the computed average residuals from Test 2 to define the apriori variance estimate σ_i^2 of a distance observation to stationary node i. These apriori estimates are given in Table 6.3. We tested the train moving clockwise (looking from on top), and we ran two sircuits around the inside track, receiving 282 valid `MULTI_RSSI` messages during the time period java time 1387404238298 ms (12/18/2013 6:03:58 PM GMT-4) to java time 1387404514450 ms (12/18/2013 6:08:34 PM GMT-4), with sequence numbers from 150 to 432, and sequence number 387 was missing. We found that 134 messages (48%) provided estimated positions and 148 (52%) did not converge. In addition, we discarded 10 “unsuccessful” estimated positions as these estimated residuals $\hat{\mathbf{r}}$ were always very large (>1000 m or <-1000 m), which gives a very large $\hat{\sigma}^2$ (always >5000).

For the 124 successful estimates, the average `b` was 548.28 cm, the average $\hat{\sigma}_0^2$ was 19.37, and the average $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) was [730.24, 893.94, 922.01]. Table 6.13 shows the average `b` (cm), $\hat{\sigma}_0^2$ and $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) value for the 124 successful estimates from test 3. These results are slightly worse than test 2.

Table 6.13: Average `b` (cm), $\hat{\sigma}_0^2$ and $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) for the 124 successful estimates from test 3.

	average value, N = 124
`b`	548.28
$\hat{\sigma}_0^2$	19.37
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[1,1]}$	730.24
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[2,2]}$	893.94
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[3,3]}$	922.01

6.3.2 Test 4

During test 2 testing, we noticed that the moving node stopped transmitting `MULTI_RSSI` messages after about four minutes. We traced this down to a bug in the code that initialized an integer counter using 8 bits instead of 16 bits. Once this was fixed, we ran another experiment called test 4 which used the same experimental setup as test 3. For test 4, we started the `MobiPos` application at java time 1387404174124 (12/18/2013 6:02:54 PM GMT-4) and stopped it at java time 1387467544986 (12/19/2013 11:39:04 AM GMT-4). The `MobiPos` application ran 17.5 hours straight with continuous receipt of `TestFtsp` messages. We received 64,443 `MULTI_RSSI` messages, with sequence numbers from 84 to 64981, and 455 (0.7%) sequence numbers missing. Of these 64,443 received `MULTI_RSSI` messages, 3593 (5.6%) `MULTI_RSSI` had five valid distances with the remaining 60,850 (94.4%) having six valid distances. We found that 29,632 messages (46%) provided estimated positions and 34,811 (54%) did not converge. In addition, we discarded 2137 "unsuccessful" estimated positions as these estimated residuals $\hat{\mathbf{r}}$ were always very large (>1000 m or <-1000 m), which gives a very large $\hat{\sigma}^2$ (always >5000). For the 27495 successful estimates, the average `b` was 617.39 cm, the average $\hat{\sigma}^2$ was 62.51, and the average $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) was [1748.51, 2190.39, 1631.31]. Table 6.14 shows the average `b` (cm), $\hat{\sigma}_0^2$ and $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) value for the 27495 successful estimates from test 4.

Running the post processing part of `MobiPos` (as shown in Figure 5.14) took 498.49 seconds to process the 64,443 `MULTI_RSSI` messages, including the

Table 6.14: Average `b` (cm), $\hat{\sigma}_0^2$ and $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) for the 27495 successful estimates from test 4.

	average value, N = 27495
`b`	617.39
$\hat{\sigma}_0^2$	62.51
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[1,1]}$	1748.51
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[2,2]}$	2190.39
$\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[3,3]}$	1631.31

time to determine the "unsuccessful" estimated positions and the average `b`, $\hat{\sigma}_0^2$ and $\sqrt{\mathbf{C}_{\hat{\mathbf{x}}}[i,i]}$ (cm) values. The post processing was done on a MacBook Air computer with 1.86 GHz Intel Core 2 Duo and 2 GB RAM.

Chapter 7

Summary and Conclusions

7.1 Summary

To obtain a reasonable position of a moving node in an indoor environment, we measured distances using two motes (TelosB and MicaZ) and two propagation models (Free Space Propagation Model (FSPM) and Log-normal Shadowing Model (LNSM)). Our experimental calibration indicated a better performance for the TelosB mote and the FSPM as they gave a distance estimate that was 0.5 m closer to the actual distances (as measured with a tape measure) compared to the LNSM with MicaZ or with the TelosB. The best experimental results for the FSPM were achieved for a path loss exponent n = 2.1.

In our `MobiPos` system, the moving node collected RSSI values from `TestFtsp` messages sent (at a default power level of 0 dBm) by up to six stationary nodes within a short period (i.e. 300ms). All six `TestFtsp` messages were received 95% of the time, and five `TestFtsp` messages were received for the remaining 5%. A stationary gateway node received all `MULTI_RSSI` messages with no lost packets. For the two tests (test 2 and test 3) with six stationary nodes, we found that 49% and 44% converged with reasonable position estimates. Compared to the approximate true position (as estimated assuming constant velocity), the average position difference for the 49% estimated positions from test 2 is 503.34 cm. Thus, we can say that using `RSSI` signals in the ITB214 lab with TelosB motes and a free space propagation model gives a position accurate to approximately 5m on average. `RSSI` signals are noisy (e.g. due to multiple reflections), resulting, on average, in around half of them giving unreasonable position estimates. Our approach of computing a position estimate using a least squares approach with more than the minimum of three distances to compute the estimated position allows us to effectively detect and eliminate the unreasonable position estimates. In addition, this approach provides a good estimate of the error in the estimated position.

7.2 Future Work

To obtain more accurate distances, we may need to use e.g. cellular telephone (e.g. Google patent: GPS/MEM Hybrid Location-Detection (GMHLD) [40]). Cellular telephones consume more energy than TelsoB motes, but cellular telephones give reasonable distances (e.g. GMHLD request GPS signals once every 100 seconds (0.01 Hz) to gain precision accuracy within 1 meter [40]). Position estimation accuracy can probably be improved by incorporating other devices (e.g. accelerometers to measure change in position) combined with a more robust navigation algorithm such as Kalman filtering.

Another option to investigate would be to filter RSSI measurements observed at a higher rate (e.g. $\sigma = 10$ ms), to smooth out the noise. Can the `MobiPos` algorithm be implemented directly on a moving node so that the moving node can know it's own position without first transmitting `MULTI_RSSI` messages to a gateway node?

References

[1] *Cell-Loc Location Technologies Inc.*, http://www.cell-loc.com/how_tech.html.

[2] *Interfaces and components of TINY OS*, http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosb/.

[3] *Interface: tos.lib.ftsp.GlobalTime*, August 2008, http://www.tinyos.net/tinyos-2.x/doc/nesdoc/micaz/ihtml/tos.lib.ftsp.GlobalTime.html#global2Local.

[4] *Tinyos documentation wiki*, August 2011, http://docs.tinyos.net/tinywiki/index.php/MainPage.

[5] K. Aamodt, *CC2431 Location Engine Application Note AN042* , Texas Instruments. July 2006.

[6] J. Arias, J. Lazaro, A. Astarloa, J. Jimenez, and A. Zuloaga, *Location algorithm for wireless sensor networks in industrial applications*, Proc. IEEE Int. Conf. Industrial Technology IEEE ICIT '04, vol. 2, 2004, pp. 757–762.

[7] J. A. R. Azevedo and F. E. Santos, *Signal Propagation Measurements with Wireless Sensor Nodes*, (2007).

[8] Timothy TJ Brooks, Huub HC Bakker, Ken A Mercer, and Wyatt H Page, *A Review of Position Tracking Methods*, November 2005, pp. 21–23.

[9] K. Bullington, *Radio Propagation at Frequencies above 30 Megacycles*, Proceedings of the IRE **35** (1947), no. 10, 1122–1136.

[10] Chipcon, *CC2420 2.4Hz IEEE 802.15.4 ZigBee -ready RF document SWRS041B*, (rev.b) ed., March 2007.

[11] Alex Chitu, *How Google Collects WiFi Data*, April 2010, http://googlesystem.blogspot.com/2010/04/how-google-collects-wifi-data.html.

[12] J.L. Crassidis and J.L. Junkins, *Optimal estimation of dynamic systems*, Chapman & Hall/CRC applied mathematics and nonlinear science series, Chapman & Hall/CRC, 2004.

[13] Crossbow, *TelosB Mote Platform Datasheet Document Part Number: 6020-0094-01 Rev B*, http://www.willow.co.uk/TelosB_Datasheet.pdf.

[14] Crossbow Technology, Inc., *MicaZ Datasheet*, December 2004.

[15] C.M. De Dominicis, A. Flammini, S. Rinaldi, E. Sisinni, A. Cazzorla, A. Moschitta, and P. Carbone, *High-precision UWB-based timestamping*, Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on, September 2011, pp. 50 –55.

[16] Liang Dong and F. L. Severance, *Position Estimation With Moving Beacons in Wireless Sensor Networks*, Proc. IEEE Wireless Communications and Networking Conf. WCNC 2007, 2007, pp. 2317–2321.

[17] European Union (EU) and European Space Agency (ESA), *Galileo (satellite navigation) wiki*, May 2003, http://en.wikipedia.org/wiki/Galileo_(satellite_navigation).

[18] Andrea Goldsmith, *Wireless Communications Principles and Practice*, Cambridge, MA, USA: Cambridge University Press, 2005.

[19] Yanying Gu, A. Lo, and I. Niemegeers, *A survey of indoor positioning systems for wireless personal networks*, Communications Surveys Tutorials, IEEE **11** (2009), no. 1, 13 –32.

[20] E. Guizzo, *Bugged balls for tough calls [ball tracking systems]*, Spectrum, IEEE **42** (2005), no. 6, 14–15.

[21] Loukas Lazos, Radha Poovendran, and Srdjan Čapkun, *ROPE: robust position estimation in wireless sensor networks*, Proceedings of the 4th international symposium on Information processing in sensor networks (Piscataway, NJ, USA), IPSN '05, IEEE Press, 2005.

[22] Hui Liu, H. Darabi, P. Banerjee, and Jing Liu, *Survey of Wireless Indoor Positioning Techniques and Systems*, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE **37** (2007), no. 6, 1067–1080.

[23] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi, *The flooding time synchronization protocol*, Proceedings of the 2nd international conference on Embedded networked sensor systems (New York, NY, USA), SenSys '04, ACM, 2004, pp. 39–49.

[24] Brian McClendon, *A new frontier for Google Maps: mapping the indoors*, November 2011, `http://googleblog.blogspot.com/2011/11/new-frontier-for-google-maps-mapping.html`.

[25] Daniel Michel and David Toggenburger, *Diploma Thesis WS 2006 Sensor Network Soccer*, Ph.D. thesis, University of Applied Sciences Rapperswil, 2006.

[26] Edward M. Mikhail, *Observations and Least Squares*, Purdue University, June 1983.

[27] Zhu Minghui and Zhang Huiqing, *Research on model of indoor distance measurement based on receiving signal strength*, Computer Design and Applications (ICCDA), 2010 International Conference on, vol. 5, June 2010, pp. V5–54 –V5–58.

[28] S. Papadakis and A. Traganitis, *Wireless positioning using the Signal Strength Difference on Arrival*, Proc. IEEE 7th Int Mobile Adhoc and Sensor Systems (MASS) Conf, 2010, pp. 674–681.

[29] Nissanka Bodhi Priyantha, *The Cricket Indoor Location System*, Ph.D. thesis, Massachusetts Institute of Technology, 2005.

[30] P Rong and M. L. Sichitiu, *Angle of Arrival Localization for Wireless Sensor Networks*, Proc. 3rd Annual IEEE Communications Society Sensor and Ad Hoc Communications and Networks SECON '06, vol. 1, 2006, pp. 374–382.

[31] Cory Sharp, Martin Turon, and David Gay, *TinyOS 2.x Timer interface nesdoc*, December 2011, http://www.tinyos.net/tinyos-2.x/doc/html/tep102.html.

[32] TinyOS, *Getting Started with TinyOS*, April 2010, http://docs.tinyos.net/tinywiki/index.php/Getting_Started_with_TinyOS.

[33] P. Vanicek, *Introduction to Adjustment Calculus (LN35)*, Department of Geodesy and Geomatics Engineering, University of New Brunswick, 1973.

[34] Petr Vanek, *Geodesy, the concepts*, North Hollan, June 1982.

[35] D. E. Wells and E. J. Krakiwsky, *The Method of Least Squares (LN18)*, Department of Geodesy and Geomatics Engineering, University of New Brunswick, 1971.

[36] Heiko Will, Stefan Pfeiffer, Stephan Adler, Thomas Hillebrandt, and Jochen Schiller, *Distance Measurement in Wireless Sensor Networks with Low Cost Components*, Internation Conference on Indoor Positioning and Indoor Navigation(IPIN), September 2011.

[37] Carl Wong, R. Klukas, and G.G. Messier, *Using WLAN Infrastructure for Angle-of-Arrival Indoor User Location*, Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th, September 2008, pp. 1 –5.

[38] Z. Xiang, S. Song, J. Chen, H. Wang, J. Huang, and X. Gao, *A wireless LAN-based indoor positioning technology*, IBM Journal of Research and Development **48** (2004), no. 5, 617–626.

[39] Jiuqiang Xu, Wei Liu, Fenggao Lang, Yuanyuan Zhang, and Chenglong Wang, *Distance Measurement Model Based on RSSI in WSN*, Wireless Sensor Network **2** (2010), 606–611.

[40] Qingxuan Yang, Edward Chang, and Guanfeng Li, *United States Patent No: 8,362,949 GPS and MEMS hybrid location-detection architecture*, January 29 2013.

[41] Lingchen Zhou, *Indoor Distance Determination Based on Received Signal Strength*, University of New Brunswick Project Report for Course EE6514 Wireless Communications, December, 2011, 18 pages.

Appendix A

Code for the MobiPos Application

A.1 RadioCountToLeds.h

Message structure of `RadioCount` message.

```
#ifndef RADIO_COUNT_TO_LEDS_H
#define RADIO_COUNT_TO_LEDS_H

enum
{
    K = 20,
    astNoOffset = 2,
    interval = 50,
};

typedef nx_struct radio_count_msg {
  nx_uint16_t counter;
} radio_count_msg_t;

enum {
  AM_RADIO_COUNT_MSG = 6,
};

#endif
```

A.2 RadioCountToLedsC.nc

Codes for stationary nodes to synchronize time and send `TestFtsp` message.

```

#include "Timer.h"
#include "RadioCountToLeds.h"

/**
 * Implementation of the RadioCountToLeds application. RadioCountToLeds
 * maintains a 4Hz counter, broadcasting its value in an AM packet
 * every time it gets updated. A RadioCountToLeds node that hears a counter
 * displays the bottom three bits on its LEDs. This application is a useful
 * test to show that basic AM communication and timers work.
 *
 * @author Philip Levis
 * @date   June 6 2005
 */

module RadioCountToLedsC @safe() {
  uses {
    interface Leds;
    interface Boot;
    interface Receive;
    interface AMSend;
    interface Timer<TMilli> as MilliTimer;
    interface SplitControl as AMControl;
    interface Packet;
  }
}
implementation {

  message_t packet;

  bool locked;
  uint16_t count = 0;
  uint16_t counter = 0;
  uint8_t addr = 0;

  event void Boot.booted() {
    call AMControl.start();
  }
```

```
  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
      call MilliTimer.startPeriodic(interval);

    }
    else {
      call AMControl.start();
    }
  }

  event void AMControl.stopDone(error_t err) {
    // do nothing
  }

  event void MilliTimer.fired() {
    count++;
    call Leds.led0Toggle();
    dbg("RadioCountToLedsC", "RadioCountToLedsC: timer fired, counter is %hu.\n", counter);
    if (locked) {
      return;
    }
    else {
      radio_count_msg_t* rcm = (radio_count_msg_t*)call Packet.getPayload(&packet, sizeof(radio_count_msg_t));
      if (rcm == NULL) {
return;
      }

      addr = count%K; // addr = stationary node id to send to
      if(addr == 0) {
         counter ++;
      }
      rcm->counter = counter;
      addr = addr + astNoOffset;   // node id to send to += astNoOffset(e.g. 2)
      if (call AMSend.send(addr, &packet, sizeof(radio_count_msg_t)) == SUCCESS) {
dbg("RadioCountToLedsC", "RadioCountToLedsC: packet sent.\n", counter);
locked = TRUE;
      }
    }
  }
```

```
  event message_t* Receive.receive(message_t* bufPtr,
       void* payload, uint8_t len) {
    dbg("RadioCountToLedsC", "Received packet of length
       %hhu.\n", len);
    if (len != sizeof(radio_count_msg_t)) {return bufPtr;}
    else {
      radio_count_msg_t* rcm = (radio_count_msg_t*)payload;
      if (rcm->counter & 0x1) {
 call Leds.led0On();
      }
      else {
 call Leds.led0Off();
      }
      if (rcm->counter & 0x2) {
 call Leds.led1On();
      }
      else {
 call Leds.led1Off();
      }
      if (rcm->counter & 0x4) {
 call Leds.led2On();
      }
      else {
 call Leds.led2Off();
      }
      return bufPtr;
    }
  }

  event void AMSend.sendDone(message_t* bufPtr, error_t error) {
    if (&packet == bufPtr) {
      locked = FALSE;
    }
  }

}
```

A.3 TestFtsp.h

Message structure of `TestFtsp` message.

```
#ifndef TEST_FTSP_H
#define TEST_FTSP_H

typedef nx_struct test_ftsp_msg
{
  nx_uint16_t    src_addr;
  nx_uint16_t    counter;
  nx_uint32_t    local_rx_timestamp;
  nx_uint32_t    global_rx_timestamp;
  nx_int32_t     skew_times_1000000;
  nx_uint8_t     is_synced;
  nx_uint16_t    ftsp_root_addr;
  nx_uint8_t     ftsp_seq;
  nx_uint8_t     ftsp_table_entries;
} test_ftsp_msg_t;
enum
{
 AM_TEST_FTSP_MSG = 137
};

#endif
```

A.4 TestFtspC.nc

Codes for stationary nodes to synchronize time and send `TestFtsp` message.

```
#include "TestFtsp.h"
#include "RadioCountToLeds.h"

module TestFtspC
{
    uses
    {
        interface GlobalTime<TMilli>;
        interface TimeSyncInfo;
        interface Receive;
        interface AMSend;
        interface Packet;
        interface Leds;
        interface PacketTimeStamp<TMilli,uint32_t>;
        interface Boot;
```

```
        interface SplitControl as RadioControl;
// interface Timer<TMilli> as MilliTimer;

    }
}

implementation
{
    message_t msg;
    bool locked = FALSE;

    event void Boot.booted() {
        call RadioControl.start();
    }


    event message_t* Receive.receive(message_t* msgPtr, void*
       payload, uint8_t len)
    {
        call Leds.led0Toggle();
        if (!locked && call PacketTimeStamp.isValid(msgPtr)) {
            radio_count_msg_t* rcm = (radio_count_msg_t*)call
               Packet.getPayload(msgPtr,
               sizeof(radio_count_msg_t));
            test_ftsp_msg_t* report = (test_ftsp_msg_t*)call
               Packet.getPayload(&msg, sizeof(test_ftsp_msg_t));

            uint32_t rxTimestamp = call
               PacketTimeStamp.timestamp(msgPtr);

            report->src_addr = TOS_NODE_ID;
            report->counter = rcm->counter;
            report->local_rx_timestamp = rxTimestamp;
            report->is_synced = call
               GlobalTime.local2Global(&rxTimestamp);
            report->global_rx_timestamp = rxTimestamp;
            report->skew_times_1000000 = (uint32_t)call
               TimeSyncInfo.getSkew()*1000000UL;
            report->ftsp_root_addr = call
               TimeSyncInfo.getRootID();
            report->ftsp_seq = call TimeSyncInfo.getSeqNum();
            report->ftsp_table_entries = call
               TimeSyncInfo.getNumEntries();
```

```
            if (call AMSend.send(AM_BROADCAST_ADDR, &msg, sizeof(test_ftsp_msg_t)) == SUCCESS) {
            call Leds.led2Toggle();
              locked = TRUE;
            }
        }

        return msgPtr;
    }

    event void AMSend.sendDone(message_t* ptr, error_t success) {
        locked = FALSE;
        return;
    }

    event void RadioControl.startDone(error_t err) {}
    event void RadioControl.stopDone(error_t error){}
}
```

A.5 MultiRssi.h

`MultiRssi` message structure sending by the moving node.

```
#ifndef MULTI_RSSI_H
#define MULTI_RSSI_H

enum
{
    TOTAL_NODE = 6,
    MOVE_NODE = 1,
    BEACON_NODE = 100,
    GATEWAY_NODE = 99,
};

typedef nx_struct multi_rssi_msg
{
  nx_uint8_t      no_of_vals;
  nx_uint16_t     counter; //sequence number
  nx_uint8_t      nodeid[TOTAL_NODE];   //node IDs
  nx_int16_t      rssi_val[TOTAL_NODE];   //rssi values
```

```
  nx_int32_t      rssi_time[TOTAL_NODE]; //record when Ftsp
      message arrived to the Moving node
  nx_uint8_t      continued;  //1 = yes, 0 = no
} multi_rssi_msg_t;

enum
{
 AM_MULTI_RSSI_MSG = 133
};


#endif
```

A.6 MultiRssiC.nc

Codes running on the moving node.

```
#include <Timer.h>
#include "MultiRssi.h"
#include "TestFtsp.h"


module MultiRssiC
{

    provides interface Init;
    uses
    {
        interface Receive;
        interface AMSend;
        interface Packet;  //for MULTI_RSSI message
        interface Leds;
        interface Boot;
        interface SplitControl as RadioControl;
        interface CC2420Packet;  //for getRssi() call to return
            rssi via TestFtsp program
        interface LocalTime<TMilli>;
    }
}

implementation
```

```
{
    message_t msg2;  // MULTI_RSSI message to send
    bool locked = FALSE;
    uint8_t no_of_vals = 0;
    uint16_t current = 0;  //current set of rssi observation
    uint8_t i = 0;
    uint8_t index = 0;
    uint16_t seq = 0; //squence number of rssi message
    uint8_t count[TOTAL_NODE];
    uint8_t nodeid[TOTAL_NODE];
    int16_t rssi_val[TOTAL_NODE];
    uint32_t rssi_time_save = 0;
    uint16_t rssi_val_save = 0;
    uint8_t nodeid_save = 0;

    uint16_t getRssi(message_t *msg)
    {
     return (uint16_t) call CC2420Packet.getRssi(msg);
    }

    event void Boot.booted() {
        call RadioControl.start();
    }


    command error_t Init.init() {
    for (i = 0; i < TOTAL_NODE; i++) {
      count[i] = 0;
      nodeid[i] = 0;
      rssi_val[i] = 0;
    }
    return SUCCESS;
  }

     event message_t* Receive.receive(message_t* msg1, void* payload, uint8_t len)
     {
     if (len != sizeof(test_ftsp_msg_t)) {return msg1;}
     else {
            test_ftsp_msg_t* tfm = (test_ftsp_msg_t*)call Packet.getPayload(msg1, sizeof(test_ftsp_msg_t)); //message received from stationary nodes
            multi_rssi_msg_t* mrm = (multi_rssi_msg_t*)call Packet.getPayload(&msg2,
```

```
        sizeof(multi_rssi_msg_t)); //message sent from moving node

    uint32_t localTime = call LocalTime.get();
    uint16_t counter = tfm->counter; //sequence number

    if(counter == current)
    {
      call Leds.led0Toggle();
      if ( rssi_time_save != 0 && rssi_val_save != 0 && nodeid_save != 0 )
      {
        mrm->rssi_time[index] = rssi_time_save; //get the time when TestFtsp msg arrived
        mrm->rssi_val[index] = rssi_val_save; //push rssi value to multi_rssi msg
        mrm->nodeid[index] = nodeid_save; //push node id to multi_rssi msg
        mrm->no_of_vals = ++index;
        rssi_time_save = 0;
        rssi_val_save = 0;
        nodeid_save = 0;
      }
      mrm->rssi_time[index] = localTime; //get the time when TestFtsp msg arrived
      mrm->rssi_val[index] = getRssi(msg1); //push rssi value to multi_rssi msg
      mrm->nodeid[index] = tfm->src_addr; //push node id to multi_rssi msg
      mrm->no_of_vals = ++index;
    }

    else if (index >= 3)
    {
      call Leds.led1Toggle();
      mrm->counter = ++ seq;
      mrm->continued = 0; //1 = yes, 0 = no

      /*send MULTI_RSSI*/
      if (call AMSend.send(AM_BROADCAST_ADDR, &msg2, sizeof(multi_rssi_msg_t)) == SUCCESS)
      {
        call Leds.led2Toggle();
        locked = TRUE;
        index = 0;
```

```
                }
                rssi_time_save = localTime;  //get the time when
                    TestFtsp msg arrived
                rssi_val_save = getRssi(msg1);  //push rssi
                    value to multi_rssi msg
                nodeid_save = tfm->src_addr;  //push node id to
                    multi_rssi msg
                index = 0;    // start a new set S
                current = counter;
            }
            else // < 3 rssi messages; throw them away, but
                keep new message
            {
                current = counter;
                index = 0;  // start a new set S
                rssi_time_save = localTime;  //get the time when
                    TestFtsp msg arrived
                rssi_val_save = getRssi(msg1);  //push rssi
                    value to multi_rssi msg
                nodeid_save = tfm->src_addr;  //push node id to
                    multi_rssi msg
            }
          }
         return msg1;

     }


    event void AMSend.sendDone(message_t* msg, error_t success)
    {
        locked = FALSE;
        return;
    }

    event void RadioControl.startDone(error_t err) {}
    event void RadioControl.stopDone(error_t error){}
}
```

A.7 GetApproxTrue()

Method GetApproxTrue() in `posOnTrack.java`

```
/**
 * Using Least Squares Method to calculate the position of
    moving node.
 *
 * @param t1  start time of a loop.
 * @param t2  end time of a loop.
 * @param t3 get the position of moving node at t3, t1<t3<t2.
 * @param dir 1 = clockwise from the top, 0 = counter
    clockwise from the top.
 * @param track 1 = outside track, 0 = inside track.
 *
 * @return
 */

public double[] GetApproxTrue(double t1, double t2, double
   t3, int dir, int track) {
 dir = 1; // 1 = clockwise from top, 0 = ccw from top
 track = 0; // 1 = outside, 0 = inside
 double v; // Velocity of train for circuit
 double s = 0.0; // Distance from p0 to the moving node
 double p = 0.0; // Function way around track
 double X[] = {0.0, 0.0, 0.0};

    double[] p0 = {139, 490, 250};
    double[] p1 = {872, 491, 251};
    double[] p2 = {989, 383, 252};
    double[] p3 = {987, 261, 251};
    double[] p4 = {871, 162, 250};
    double[] p5 = {136, 162, 252};
    double[] p6 = {30, 267, 252};
    double[] p7 = {29, 386, 252};

    double[] p0o = {139, 496, 250};
    double[] p1o = {872, 497, 251};
    double[] p2o = {995, 383, 252};
    double[] p3o = {993, 261, 251};
    double[] p4o = {871, 156, 250};
    double[] p5o = {136, 156, 252};
    double[] p6o = {24, 267, 252};
    double[] p7o = {23, 386, 252};

    double insideLength = 2414.27;
    double outsideLength = 2436;
    double L; // Length of track
    if( track == 1 ) {
```

```
        L = outsideLength;
       }
       else {
        L = insideLength;
       }

    double[] Si = {0.0, 733.0,  916.78, 1038.78, 1220.89,
        1955.99, 2122.49, 2241.49, 2414.27};
    double[] So = {0.0, 740.0, 920.0, 1038.0, 1218.0, 1958.0,
        2138.0, 2256.0, 2436.0 };
    double[] SiCounter = {0.0, 169.0, 287.0, 456.0, 1196.0,
        1365.0, 1483.0, 1652.0, 2392.0};
    double[] SoCounter = {0.0, 180.0, 298.0, 478.0, 1218.0,
        1398.0, 1516.0, 1696.0, 2436.0};


    v = L / (t2 - t1); // velocity in cm/sec
    System.out.println("Moving node is moving at speed of: ");
    System.out.print(v);

    s = v * (t3 - t1);

       /** When moving node is on outside track, cw, at S0 to
           S1 area, return its position
       @return  X
       */
    if( (dir == 1) && (track == 1) && s < So[1]  ) {
     p = s / So[1];
     for(int j = 0; j<=2; j++){
      X[j] = p0[j] + p * (p1o[j] - p0o[j]);
     }
     System.out.println("Moving node is between p0o and p1o");
     System.out.print("The position of moving node is :");
     System.out.println(" X  = [ " + X[0] + " ," + X[1] + " ,"
         + X[2] + " ]" );
    }

       /** When moving node is on inside track, cw, at S0 to S1
           area, return its position
       @return  X
       */
    else if( (dir == 1) &&  (track == 0) && s < Si[1]  ) {
     p = s / Si[1];
     for(int j = 0; j<=2; j++){
      X[j] = p0[j] + p * (p1[j] - p0[j]);
```

```
        }
        System.out.println("Moving node is between p0 and p1");
        System.out.println(" X1  = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }


        /** When moving node is on outside track, cw, at S1 to
            S2 area, return its position
        @return   X
        */
      else if( (dir == 1) && (track == 1) && ( s > So[1]) && (s
        <= So[2])){
        double r = Math.abs(p2o[0] - p1o[0]);
        double alpha = 180 * (s - So[1]) / (Math.PI*r);
        p = alpha / 90;
        X[0] = p1o[0] + Math.sin(alpha);
        X[1] = p1o[1] - (r - Math.cos(alpha));
        X[2] = p1o[2] + p * (p2o[2] - p1o[2]);
        System.out.println("Moving node is between p1o and p2o");
        System.out.println(" X2  = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }


        /** When moving node is on inside track, cw, at S1 to S2
            area, return its position
        @return   X
        */
      else if( (dir == 1) && (track == 0) && ( s > Si[1]) && (s
        <= Si[2])){
        double r = Math.abs(p2[0] - p1[0]);
        double alpha = 180 * (s - Si[1]) / (Math.PI*r);
        p = alpha / 90;
        X[0] = p1[0] + Math.sin(alpha);
        X[1] = p1[1] - (r - Math.cos(alpha));
        X[2] = p1[2] + p * (p2[2] - p1[2]);
        System.out.println("Moving node is between p1 and p2");
        System.out.println(" X2  = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }


        /** When moving node is on outside track, cw, at S2 to
            S3 area, return its position
```

```
    @return   X
    */
else if( (dir == 1) && (track == 1) && ( s > So[2]) && (s
    <= So[3]) ){
  p = (s - So[2]) / (So[3] - So[2]);
  for(int j = 0; j <= 2; j++){
  X[j] = p2o[j] + p * (p3o[j] - p2o[j]);
  }
  System.out.println("Moving node is between p2o and p3o");
  System.out.println(" X3  = [ " + X[0] + "  ," + X[1] + "  ,"
      + X[2] + "  ]" );
}

    /** When moving node is on inside track, cw, at S2 to S3
        area, return its position
    @return   X
    */
else if( (dir == 1) && (track == 0) && ( s > Si[2]) && (s
    <= Si[3]) ){
  p = (s - Si[2]) / (Si[3] - Si[2]);
  for(int j = 0; j <= 2; j++){
  X[j] = p2[j] + p * (p3[j] - p2[j]);
  }
  System.out.println("Moving node is between p2 and p3");
  System.out.println(" X3  = [ " + X[0] + "  ," + X[1] + "  ,"
      + X[2] + "  ]" );
}


    /** When moving node is on outside track, cw, at S3 to
        S4 area, return its position
    @return   X
    */
else if( (dir == 1) && (track == 1) && ( s > So[3]) && (s
    <= So[4]) ){
  double r = Math.abs(p4o[0] - p3o[0]);
  double alpha = 180 * (s - So[3]) / (Math.PI*r);
  p = alpha / 90;
  X[0] = p3o[0] - (r - Math.cos(alpha));
  X[1] = p3o[1] - Math.sin(alpha);
  X[2] = p3o[2] + p * (p4o[2] - p3o[2]);
  System.out.println("Moving node is between p3o and p4o");
  System.out.println(" X4  = [ " + X[0] + "  ," + X[1] + "  ,"
      + X[2] + "  ]" );
}
```

```java

        /** When moving node is on inside track, cw, at S3 to S4 area, return its position
        @return  X
        */
      else if( (dir == 1) && (track == 0) && ( s > Si[3]) && (s <= Si[4]) ){
       double r = Math.abs(p4[0] - p3[0]);
       double alpha = 180 * (s - Si[3]) / (Math.PI*r);
       p = alpha / 90;
       X[0] = p3[0] - (r - Math.cos(alpha));
       X[1] = p3[1] - Math.sin(alpha);
       X[2] = p3[2] + p * (p4[2] - p3[2]);
       System.out.println("Moving node is between p3 and p4");
       System.out.println(" X4  = [ " + X[0] + " ," + X[1] + " ," + X[2] + " ]" );
      }


        /** When moving node is on outside track, cw, at S4 to S5 area, return its position
        @return  X
        */
      else if( (dir == 1) && (track == 1) && ( s > So[4]) && (s <= So[5]) ){
       p = (s - So[4])/(So[5] - So[4]);
       for(int j = 0; j <= 2; j++){
        X[5] = p4o[5] + p * (p5o[5] - p4o[5]);
       }
       System.out.println("Moving node is between p4o and p5o");
       System.out.println(" X5  = [ " + X[0] + " ," + X[1] + " ," + X[2] + " ]" );
      }


        /** When moving node is on inside track, cw, at S4 to S5 area, return its position
        @return  X
        */
      else if( (dir == 1) && (track == 0) && ( s > Si[4]) && (s <= Si[5]) ){
       p = (s - Si[4])/(Si[5] - Si[4]);
       for(int j = 0; j <= 2; j++){
        X[j] = p4[j] + p * (p5[j] - p4[j]);
       }
```

```
        System.out.println("Moving node is between p4 and p5");
        System.out.println(" X5  = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }


        /** When moving node is on outside track, cw, at S5 to
            S6 area, return its position
        @return   X
        */
      else if( (dir == 1) && (track == 1) && ( s > So[5]) && (s
        <= So[6]) ){
        double r = Math.abs(p6o[0] - p5o[0]);
        double alpha = 180 * (s - So[5]) / (Math.PI*r);
        p = alpha / 90;
        X[0] = p5o[0] - Math.sin(alpha);
        X[1] = p5o[1] + (r - Math.cos(alpha));
        X[2] = p5o[2] + p * (p6o[6] - p5o[6]);
        System.out.println("Moving node is between p5o and p6o");
        System.out.println(" X6  = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }

        /** When moving node is on inside track, cw, at S5 to S6
            area, return its position
        @return   X
        */
      else if( (dir == 1) && (track == 0) && ( s > Si[5]) && (s
        <= Si[6]) ){
        double r = Math.abs(p6[0] - p5[0]);
        double alpha = 180 * (s - Si[5]) / (Math.PI*r);
        p = alpha / 90;
        X[0] = p5[0] - Math.sin(alpha);
        X[1] = p5[1] + (r - Math.cos(alpha));
        X[2] = p5[2] + p * (p6[2] - p5[2]);
        System.out.println("Moving node is between p5 and p6");
        System.out.println(" X6  = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }

        /** When moving node is on outside track, cw, at S6 to
            S7 area, return its position
        @return   X
        */
```

```
else if( (dir == 1) && (track == 1) && ( s > So[6]) && (s
    <= So[7]) ){
  p = (s - So[6])/(So[7] - So[6]);
  for(int j = 0; j <= 2; j++){
   X[7] = p6o[7] + p * (p7[2] - p6[2]);
  }
  System.out.println("Moving node is between p6o and p7o");
  System.out.println(" X7  = [ " + X[0] + " ," + X[1] + " ,"
      + X[2] + " ]" );
}

    /** When moving node is on inside track, cw, at S6 to S7
        area, return its position
    @return   X
    */
else if( (dir == 1) && (track == 0) && ( s > Si[6]) && (s
    <= Si[7]) ){
  p = (s - Si[6])/(Si[7] - Si[6]);
  for(int j = 0; j <= 2; j++){
   X[j] = p6[j] + p * (p7[j] - p6[j]);
  }
  System.out.println("Moving node is between p6 and p7");
  System.out.println(" X7  = [ " + X[0] + " ," + X[1] + " ,"
      + X[2] + " ]" );
}


    /** When moving node is on outside track, cw, at S7 to
        S8 area, return its position
    @return   X
    */
else if( (dir == 1) && (track == 1) && ( s > So[7]) && (s
    <= So[8]) ){
  double r = Math.abs(p0o[0] - p7o[0]);
  double alpha = 180 * (s - So[7]) / (Math.PI*r);
  p = alpha / 90;
  X[0] = p7o[0] + (r - Math.cos(alpha));
  X[1] = p7o[1] + Math.sin(alpha);
  X[2] = p7o[2] + p * (p0o[2] - p7o[2]);
  System.out.println("Moving node is between p7o and p0o");
  System.out.println(" X8  = [ " + X[0] + " ," + X[1] + " ,"
      + X[2] + " ]" );
}
```

```
    /** When moving node is on inside track, cw, at S7 to S8
       area, return its position
    @return  X
    */
  else if( (dir == 1) && (track == 0) && ( s > Si[7]) && (s
    <= Si[8]) ){
   double r = Math.abs(p0[0] - p7[0]);
   double alpha = 180 * (s - Si[7]) / (Math.PI*r);
   p = alpha / 90;
   X[0] = p7[0] + (r - Math.cos(alpha));
   X[1] = p7[1] + Math.sin(alpha);
   X[2] = p7[2] + p * (p0[2] - p7[2]);
   System.out.println("Moving node is between p7 and p0");
   System.out.println(" X8  = [ " + X[0] + "  ," + X[1] + "  ,"
      + X[2] + "  ]" );
  }

    /** When moving node is on outside track, ccw, at
       SoCounter0 to SoCounter1 area, return its position
    @return  X
    */
  else if( (dir == 0) && (track == 1) && ( s > SoCounter[0])
    && (s <= SoCounter[1]) ){
   double r = Math.abs(p0o[0] - p7o[0]);
   double alpha = 180 * (s - SoCounter[0]) / (Math.PI*r);
   p = alpha / 90;
   X[0] = p0o[0] - Math.sin(alpha);
   X[1] = p0o[1] - (r - Math.cos(alpha));
   X[2] = p0o[2] + p*(p7o[2] - p0o[2]);
   System.out.println("Moving node is between p0o and p7o");
   System.out.println(" X  = [ " + X[0] + "  ," + X[1] + "  ,"
      + X[2] + "  ]" );
  }

    /** When moving node is on inside track, ccw, at
       SiCounter0 to SiCounter1 area, return its position
    @return  X
    */
  else if( (dir == 0) && (track == 0) && ( s > SiCounter[0])
    && (s <= SiCounter[1]) ){
   double r = Math.abs(p0[0] - p7[0]);
   double alpha = 180 * (s - SiCounter[0]) / (Math.PI*r);
   p = alpha / 90;
   X[0] = p0[0] - Math.sin(alpha);
   X[1] = p0[1] - (r - Math.cos(alpha));
```

```java
      X[2] = p0[2] + p*(p7[2] - p0[2]);
      System.out.println("Moving node is between p0 and p7");
      System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
          + X[2] + " ]" );
    }

      /** When moving node is on outside track, ccw, at
          SoCounter1 to SoCounter2 area, return its position
      @return  X
      */
    else if( (dir == 0) && (track == 1) && ( s > SoCounter[1])
      && (s <= SoCounter[2]) ){
      p = (s - SoCounter[1])/(SoCounter[2] - SoCounter[1]);
      for(int j = 0; j <= 2; j++){
       X[j] = p7o[j] + p * (p6o[j] - p7o[j]);
      }
      System.out.println("Moving node is between p7o and p6o");
      System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
          + X[2] + " ]" );
    }

      /** When moving node is on inside track, ccw, at
          SiCounter1 to SiCounter2 area, return its position
      @return  X
      */
    else if( (dir == 0) && (track == 0) && ( s > SiCounter[1])
      && (s <= SiCounter[2]) ){
      p = (s - SiCounter[1])/(SiCounter[2] - SiCounter[1]);
      for(int j = 0; j <= 2; j++){
       X[j] = p7[j] + p * (p6[j] - p7[j]);
      }
      System.out.println("Moving node is between p7 and p6");
      System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
          + X[2] + " ]" );
    }


      /** When moving node is on outside track, ccw, at
          SoCounter2 to SoCounter3 area, return its position
      @return  X
      */
    else if( (dir == 0) && (track == 1) && ( s > SoCounter[2])
      && (s <= SoCounter[3]) ){
      double r = Math.abs(p6o[0] - p5o[0]);
      double alpha = 180 * (s - SoCounter[2]) / (Math.PI*r);
```

```
        p = alpha / 90;
        X[0] = p6o[0] + (r - Math.cos(alpha));
        X[1] = p6o[1] - Math.sin(alpha);
        X[2] = p6o[2] + p*(p5o[2] - p6o[2]);
//      System.out.println("Moving node is between p0o and p1o");
        System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }

        /** When moving node is on inside track, ccw, at
            SiCounter2 to SiCounter3 area, return its position
        @return X
        */
      else if( (dir == 0) && (track == 0) && ( s > SiCounter[2])
        && (s <= SiCounter[3]) ){
        double r = Math.abs(p6[0] - p5[0]);
        double alpha = 180 * (s - SiCounter[2]) / (Math.PI*r);
        p = alpha / 90;
        X[0] = p6[0] + (r - Math.cos(alpha));
        X[1] = p6[1] - Math.sin(alpha);
        X[2] = p6[2] + p*(p5[2] - p6[2]);
        System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }

        /** When moving node is on outside track, ccw, at
            SoCounter3 to SoCounter4 area, return its position
        @return X
        */
      else if( (dir == 0) && (track == 1) && ( s > SoCounter[3])
        && (s <= SoCounter[4]) ){
        p = (s - SoCounter[3])/(SoCounter[4] - SoCounter[3]);
        for(int j = 0; j <= 2; j++){
          X[j] = p5o[j] + p * (p4o[j] - p5o[j]);
        }
        System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
            + X[2] + " ]" );
      }

        /** When moving node is on inside track, ccw, at
            SiCounter3 to SiCounter4 area, return its position
        @return X
        */
      else if( (dir == 0) && (track == 0) && ( s > SiCounter[3])
        && (s <= SiCounter[4]) ){
```

```
        p = (s - SiCounter[3])/(SiCounter[4] - SiCounter[3]);
        for(int j = 0; j <= 2; j++){
         X[j] = p5[j] + p * (p4[j] - p5[j]);
        }
        System.out.println(" X  = [ " + X[0] + "  ," + X[1] + "  ,"
           + X[2] + " ]" );
      }

        /** When moving node is on outside track, ccw, at
            SoCounter4 to SoCounter5 area, return its position
        @return   X
        */
      else if( (dir == 0) && (track == 1) && ( s > SoCounter[4])
          && (s <= SoCounter[5]) ){
        double r = Math.abs(p3o[0] - p4o[0]);
        double alpha = 180 * (s - SoCounter[4]) / (Math.PI*r);
        p = alpha / 90;
        X[0] = p4o[0] + Math.sin(alpha);
        X[1] = p4o[1] + (r - Math.cos(alpha));
        X[2] = p4o[2] + p*(p3o[2] - p4o[2]);
        System.out.println(" X  = [ " + X[0] + "  ," + X[1] + "  ,"
           + X[2] + " ]" );
      }

        /** When moving node is on inside track, ccw, at
            SiCounter4 to SiCounter5 area, return its position
        @return   X
        */
      else if( (dir == 0) && (track == 0) && ( s > SiCounter[4])
          && (s <= SiCounter[5]) ){
        double r = Math.abs(p3[0] - p4[0]);
        double alpha = 180 * (s - SiCounter[4]) / (Math.PI*r);
        p = alpha / 90;
        X[0] = p4[0] + Math.sin(alpha);
        X[1] = p4[1] + (r - Math.cos(alpha));
        X[2] = p4[2] + p*(p3[2] - p4[2]);
        System.out.println(" X  = [ " + X[0] + "  ," + X[1] + "  ,"
           + X[2] + " ]" );
      }

        /** When moving node is on outside track, ccw, at
            SoCounter5 to SoCounter6 area, return its position
        @return   X
        */
```

```java
        else if( (dir == 0) && (track == 1) && ( s > SoCounter[5])
            && (s <= SoCounter[6]) ){
         p = (s - SoCounter[5])/(SoCounter[6] - SoCounter[5]);
         for(int j = 0; j <= 2; j++){
          X[j] = p3o[j] + p * (p2o[j] - p3o[j]);
         }
         System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
             + X[2] + " ]" );
        }

          /** When moving node is on inside track, ccw, at
              SiCounter5 to SiCounter6 area, return its position
          @return  X
          */
        else if( (dir == 0) && (track == 0) && ( s > SiCounter[5])
            && (s <= SiCounter[6]) ){
         p = (s - SiCounter[5])/(SiCounter[6] - SiCounter[5]);
         for(int j = 0; j <= 2; j++){
          X[j] = p3[j] + p * (p2[j] - p3[j]);
         }
         System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
             + X[2] + " ]" );
        }

          /** When moving node is on outside track, ccw, at
              SoCounter6 to SoCounter7 area, return its position
          @return  X
          */
        else if( (dir == 0) && (track == 1) && ( s > SoCounter[6])
            && (s <= SoCounter[7]) ){
         double r = Math.abs(p1o[0] - p2o[0]);
         double alpha = 180 * (s - SoCounter[6]) / (Math.PI*r);
         p = alpha / 90;
         X[0] = p2o[0] - (r - Math.cos(alpha));
         X[1] = p2o[1] + Math.sin(alpha);
         X[2] = p2o[2] + p*(p1o[2] - p2o[2]);
         System.out.println(" X = [ " + X[0] + " ," + X[1] + " ,"
             + X[2] + " ]" );
        }

          /** When moving node is on inside track, ccw, at
              SiCounter6 to SiCounter7 area, return its position
          @return  X
          */
```

```
else if( (dir == 0) && (track == 0) && ( s > SiCounter[6])
    && (s <= SiCounter[7]) ){
 double r = Math.abs(p1[0] - p2[0]);
 double alpha = 180 * (s - SiCounter[6]) / (Math.PI*r);
 p = alpha / 90;
 X[0] = p2[0] - (r - Math.cos(alpha));
 X[1] = p2[1] + Math.sin(alpha);
 X[2] = p2[2] + p*(p1[2] - p2[2]);
 System.out.println(" X  = [ " + X[0] + " ," + X[1] + " ,"
     + X[2] + " ]" );
}


   /** When moving node is on outside track, ccw, at
       SoCounter7 to SoCounter8 area, return its position
   @return   X
   */
else if( (dir == 0) && (track == 1) && ( s > SoCounter[7])
    && (s <= SoCounter[8]) ){
 p = (s - SoCounter[7])/(SoCounter[8] - SoCounter[7]);
 for(int j = 0; j <= 2; j++){
  X[j] = p1o[j] + p * (p0o[j] - p1o[j]);
 }
 System.out.println(" X  = [ " + X[0] + " ," + X[1] + " ,"
     + X[2] + " ]" );
}

   /** When moving node is on inside track, ccw, at
       SiCounter7 to SiCounter8 area, return its position
   @return   X
   */
else if( (dir == 0) && (track == 0) && ( s > SiCounter[7])
    && (s <= SiCounter[8]) ){
 p = (s - SiCounter[7])/(SiCounter[8] - SiCounter[7]);
 for(int j = 0; j <= 2; j++){
  X[j] = p1[j] + p * (p0[j] - p1[j]);
 }
 System.out.println(" X  = [ " + X[0] + " ," + X[1] + " ,"
     + X[2] + " ]" );
}

 return X;
}
```

A.8 leastSquares.java

Full implementation of the `least squares` method discussed in Chapter 2.

```
public class leastSquares {
 private double[] xHat;
 private double[] d;
 private double[][] Ps;
 private double epsilon;
 private double[] var;
 private double[] deltaX;
 private double[][] Cx;
 private double[] rHat;
 private double sigmaSquare;
 private static int m, n;
 String contents = "";
 PrintStream outReport = null;



 /**
  * Using Least Squares Method to calculate the position of
     moving node.
  *
  * @param epsilon  = parameter (cm); if length of deltaX
     vector in cm < epsilon,
  *                                  then the least squares
     iteration process ends.
  * @param d  distances between moving node to stationary nodes
     in cm.
  * @param var variances of distances in cm^2.
  * @param Ps position of stationary nodes in cm.
  * @param xHat estimated position of moving node.
  */
 public leastSquares(double epsilon, double[] d, double[] var,
   double[][] Ps, double[] xHat) {

  if (d == null || var == null || Ps == null || xHat == null
    || d.length < 3 || Ps.length < 3 || xHat.length < 3)
  throw new IllegalArgumentException("invalid parameter");
  this.epsilon = 0.1;
  this.d = d;
  this.var = var;
  this.Ps = Ps;
  this.xHat = xHat;
  this.m = Ps.length;
```

```
    this.n = Ps[0].length;


  /*
   * get estimated position.
   */
   int count = 0;  int maxcount = 1000;
   double lengthDeltaX = 100000.0;
   double[][] P = Matrix.getP(var);
   double[] dHat = new double[m];
   deltaX = new double[m];

    while (lengthDeltaX > epsilon && count < maxcount){
    // estimated distance(4.6)
    double ds = 0.0;
    for (int j = 0; j < m; j++) {
     for (int i = 0; i < n; i++) {
      ds = ds + (xHat[i] - Ps[j][i])*(xHat[i] - Ps[j][i]);
          //distance squared
     }
     dHat[j] = Math.sqrt(ds);
     ds = 0.0;
    }

    // closure vector (4.8)
    double[] w = Matrix.getClosure(dHat, d);
    double[][] A = Matrix.getA(dHat, xHat, Ps);
    double[][] ATP = new double[n][m];
    ATP = Matrix.times(Matrix.transpose(A), P);
    double[][] N = new double[n][n];
    N = Matrix.times(ATP, A);
    double[][] NI = new double[n][n];
    NI = Matrix.inverse(N);
    double[][] I = new double[n][n];
    I = Matrix.times(N, NI);

    double[] ATPW = new double[n];
    ATPW = Matrix.times21(ATP, w);
    deltaX = Matrix.times21(NI, ATPW);

    // Update estimated position (4.11)
    for (int i = 0; i < n; i++) {
     xHat[i] = xHat[i] + deltaX[i];
    }
    lengthDeltaX = 0.0;
```

```java
    double sqrSum = 0.0;
    for (int i = 0; i < n; i++) {
     sqrSum += deltaX[i] * deltaX[i];
    }
    lengthDeltaX = Math.sqrt(sqrSum);
    count++;

  /*
   * Get estimated vector of residuals.
   * @param xHat  estimated position.
   */

   double[] AbydeltaX = new double[m];
   AbydeltaX = Matrix.times21(A, deltaX);
   rHat = Matrix.vectorPlus(AbydeltaX, w);



  /*
   * Get estimated reference variance of the observations.
   * @param xHat  estimated position.
   */

   double[] rTP = new double[m];
   for (int j = 0; j < m; j++) {
    for(int i = 0; i<m; i++) {
     rTP[i] += rHat[i] * P[i][j];
    }
   }
   sigmaSquare = Matrix.times11(rTP, rHat) / (m - n);

   /*
    * Get covariance.
    * @param xHat  estimated position.
    */
   Cx = Matrix.times02(sigmaSquare,
       Matrix.inverse(Matrix.times(Matrix.times(Matrix.transpose(A),
       P), A)));

  }
}
```

A.9 diffCal.java

Used to compute the position difference$(\Delta x, \Delta y, \Delta z) = (\bar{x}, \bar{y}, \bar{z})$ - $(\hat{x}, \hat{y}, \hat{z})$, and the difference `b` between the estimated and approximate true positions.

```java

public class diffCal {
 public static void main(String[] args) {
  int i,j,u,v,t=0;
  int count = 0;
  double disadd = 0;
  double dis, disSquare = 0.0;
  double sqrtC1add = 0.0;
  double sqrtC2add = 0.0;
  double sqrtC3add = 0.0;
  double sSadd = 0.0;
  double[] rHatadd = {0.0,0.0,0.0,0.0,0.0,0.0};
  double[] sigmaSquaredif = {0.0,0.0,0.0,0.0,0.0,0.0};
  double[] sigmaSquare = new double[6];
  double[] avrHat = {-369.43, -285.99, -142.57, -93.65,
     -256.77, -339.77};

        java.text.DecimalFormat df = new
            java.text.DecimalFormat("0.00");
        df.setRoundingMode(RoundingMode.HALF_UP);
  try{
   FileReader read1 = new FileReader("src/trainPos.txt");
   FileReader read2 = new FileReader("src/trainPosE.txt");
   BufferedReader br1 = new BufferedReader(read1);
   BufferedReader br2 = new BufferedReader(read2);
   String row1,row2;

   while((row1 = br1.readLine())!=null && (row2 =
       br2.readLine())!=null ){

    if(row1.length() < 50) // check if trainPos available
    {
     row1 = null;
     row2 = null;
        System.out.println("Estimated position doesn't exsit!");
    }else
    {
     t = row1.indexOf(":");
     String Javatime = row1.substring(t,t+15);
     System.out.println("Java time at " + Javatime);
```

```
i = row1.indexOf("[");
j = row1.indexOf("]");
u = row2.indexOf("[");
v = row2.indexOf("]");
String str1 = row1.substring(i+2, j-1);
System.out.println(str1);
String str2 = row2.substring(u+2, v-1);
System.out.println(str2);
String[] arrayXHat = str1.split(" ");
double [] xHat = new double [arrayXHat.length];
int m = 0;
for (String str : arrayXHat) {
 xHat[m++] = Double.parseDouble(str);
}

String[] arrayXBar = str2.split(" ");
double [] xBar = new double [arrayXBar.length];
int n = 0;
for (String str : arrayXBar) {
 xBar[n++] = Double.parseDouble(str);
}

    double[] deltaX = new double[xBar.length];
    int a = 0;
    a = row1.indexOf("s");
    String str3 = row1.substring(a);
    int b = 0;
    int c = 0;
    b = str3.indexOf("=");
    c = str3.indexOf(" ");
    String str4 = str3.substring(b+1,c); //sqrtC[1][1]
    String str5 = str3.substring(c+2);
    int d = 0;
    d = str5.indexOf("s");
    String str6 = str5.substring(d);
    int e = 0;
    int f = 0;
    e = str6.indexOf("=");
    f = str6.indexOf(" ");
    String str7 = str6.substring(e+1,f);
    String str8 = str6.substring(f+2);
    int g = 0;
    g = str8.indexOf("s");
    String str9 = str8.substring(g);
```

```
            int h = 0;
            int l = 0;
            h = str9.indexOf("=");
            l = str9.indexOf(" ");
            String str10 = str9.substring(h+1,l);
            String str11 = str9.substring(l+15);
            String str12 = row1.substring(j+2);
            int o = 0;
            int p = 0;
            o = str12.indexOf("[");
            p = str12.indexOf("]");
            String str13 = str12.substring(o+2, p-1); //rHat
            String[] arrayrHat = str13.split(" ");
            double [] rHat = new double [arrayrHat.length];
            int q = 0;
            for (String str : arrayrHat) {
             rHat[q++] = Double.parseDouble(str);
        }

            if(xHat[0]>10000 ||xHat[1]>10000 ||
                xHat[2]>10000||xHat[0]<0)
            {
             System.out.println("Position didn't converge in the
                 right way.");
             System.out.println();
            }
            else{
             double sqrtC1 =  Double.parseDouble(str4);
             double sqrtC2 =  Double.parseDouble(str7);
             double sqrtC3 =  Double.parseDouble(str10);
             double sigmaSquared =  Double.parseDouble(str11);

            System.out.print("Position difference: xBar - xHat =
               (" );
            for(int k = 0; k < xBar.length; k ++)
            {
             deltaX[k] = xBar[k] -xHat[k];
             disSquare +=deltaX[k] * deltaX[k];
         System.out.print(df.format(deltaX[k]) + "&");
            }
            dis = Math.sqrt(disSquare);
            disSquare = 0;

            System.out.println(")");
```

```
        System.out.println("distance difference: " +
            df.format(dis));
            count ++;
            sqrtC1add = sqrtC1add + sqrtC1;
            sqrtC2add = sqrtC2add + sqrtC2;
            sqrtC3add = sqrtC3add + sqrtC3;
            sSadd = sSadd + sigmaSquared;
            for(int k = 0; k<rHat.length;k++){
             rHatadd[k] = rHatadd[k] +rHat[k];
             sigmaSquaredif[k] =(rHat[k]  - avrHat[k]) * (rHat[k]
                 - avrHat[k]);
             System.out.println("average rHat[" + k +"] = "   +
                 df.format(rHatadd[k]/count));
             System.out.println("sigmaSquared[" + k +"] = "   +
                 df.format(sigmaSquaredif[k]/(count-1)));

            }
            disadd = disadd + dis;
            }
            System.out.println("average sqrtC[1][1] = " +
                df.format(sqrtC1add/count));
            System.out.println("average sqrtC[2][2] = " +
                df.format(sqrtC2add/count));
            System.out.println("average sqrtC[3][3] = "  +
                df.format(sqrtC3add/count));
            System.out.println("average sigmaSquared = "  +
                df.format(sSadd/count));
            System.out.println("average distance difference = " +
                df.format(disadd/count));
            System.out.println(count);
            System.out.println("unsucessfull lines: " + (line -
                count));
      }
     }
   }catch(FileNotFoundException e){
    e.printStackTrace();
   }catch(IOException e){
    e.printStackTrace();
   }

  }

}
```

Appendix B

Test Result for the MobiPos Application

B.1 Full Data for Test 2

[JAVA_TIME] [No_of_VALS] [SEQ_NUM] [DISTANCES] [NODE_ID] [RSSI_VAL] [RSSI_TIME] [CONTINUED]
1384889721889 6 203 [597.48 115.36 143.64 830.2 926.41 199.59] [2 3 4 5 6 7] [-10 5 3 -13 -14 0] [659997 660051 660106 660152 660192 660249] 0
1384889722876 6 204 [430 128.72 222.72 248.53 345.33 222.72] [2 3 4 5 6 7] [-7 4 -1 -2 -5 -1] [660997 661049 661102 661158 661198 661257] 0
1384889723842 6 205 [1153.56 143.64 597.48 1602.87 277.33 830.2] [2 3 4 5 6 7] [-16 3 -10 -19 -3 -13] [662005 662052 662100 662148 662202 662254] 0
1384889724820 6 206 [3853.46 178.86 309.47 535.44 1153.56 430] [2 3 4 5 6 7] [-27 1 -4 -9 -16 -7] [662993 663049 663098 663149 663198 663247] 0
1384889725800 6 207 [666.72 597.48 430 248.53 926.41 385.35] [2 3 4 5 6 7] [-11 -10 -7 -2 -14 -6] [663999 664049 664096 664156 664194 664253] 0
1384889726783 6 208 [830.2 160.29 128.72 597.48 479.83 385.35] [2 3 4 5 6 7] [-13 2 4 -10 -8 -6] [665000 665050 665098 665143 665195 665254] 0
1384889727757 6 209 [248.53 222.72 178.86 385.35 926.41 199.59] [2 3 4 5 6 7] [-2 -1 1 -6 -14 0] [666004 666047 666097 666148 666201 666253] 0
1384889728728 6 210 [309.47 535.44 199.59 535.44 222.72 597.48] [2 3 4 5 6 7] [-4 -9 0 -9 -1 -10] [666999 667046 667107 667152 667198 667248] 0
1384889729708 6 211 [178.86 277.33 830.2 479.83 597.48 479.83] [2 3 4 5 6 7] [1 -3 -13 -8 -10 -8] [667998 668048 668100 668152 668201 668247] 0

1384889730687 6 212 [248.53 199.59 128.72 430 666.72 345.33] [2 3 4 5 6 7] [-2 0 4 -7 -11 -5] [669002 669055 669105 669146 669200 669251] 0
1384889731662 6 213 [143.64 1287.24 199.59 743.99 926.41 479.83] [2 3 4 5 6 7] [3 -17 0 -12 -14 -8] [669997 670052 670102 670154 670197 670247] 0
1384889732644 6 214 [597.48 309.47 128.72 4798.31 535.44 479.83] [2 3 4 5 6 7] [-10 -4 4 -29 -9 -8] [671001 671046 671099 671146 671198 671247] 0
1384889733611 6 215 [1436.41 178.86 597.48 743.99 248.53 385.35] [2 3 4 5 6 7] [-18 1 -10 -12 -2 -6] [672006 672046 672102 672146 672194 672251] 0
1384889734588 6 216 [385.35 385.35 143.64 597.48 926.41 597.48] [2 3 4 5 6 7] [-6 -6 3 -10 -14 -10] [673000 673054 673099 673148 673197 673250] 0
1384889735571 5 217 [535.44 199.59 597.48 385.35 345.33] [2 3 5 6 7] [-9 0 -10 -6 -5] [673997 674058 674156 674202 674254] 0
1384889736544 6 218 [666.72 926.41 830.2 926.41 535.44 1436.41] [2 3 4 5 6 7] [-11 -14 -13 -14 -9 -18] [675002 675052 675099 675143 675199 675251] 0
1384889737526 6 219 [1602.87 479.83 430 666.72 128.72 1033.77] [2 3 4 5 6 7] [-19 -8 -7 -11 4 -15] [675999 676047 676104 676145 676207 676249] 0
1384889738496 6 220 [666.72 597.48 666.72 1153.56 430 830.2] [2 3 4 5 6 7] [-11 -10 -11 -16 -7 -13] [677005 677052 677100 677156 677203 677251] 0
1384889739471 6 221 [597.48 479.83 199.59 1602.87 479.83 1033.77] [2 3 4 5 6 7] [-10 -8 0 -19 -8 -15] [677997 678049 678103 678151 678204 678249] 0
1384889740449 6 222 [248.53 597.48 385.35 830.2 430 666.72] [2 3 4 5 6 7] [-2 -10 -6 -13 -7 -11] [678997 679043 679102 679146 679200 679249] 0
1384889741427 6 223 [178.86 743.99 309.47 1033.77 345.33 597.48] [2 3 4 5 6 7] [1 -12 -4 -15 -5 -10] [680000 680055 680100 680153 680195 680248] 0
1384889742408 6 224 [385.35 479.83 743.99 830.2 1033.77 666.72] [2 3 4 5 6 7] [-6 -8 -12 -13 -15 -11] [681001 681044 681098 681146 681203 681251] 0
1384889743371 6 225 [1033.77 743.99 160.29 1033.77 535.44 830.2] [2 3 4 5 6 7] [-15 -12 2 -15 -9 -13] [681999 682051 682109 682150 682200 682259] 0
1384889744355 6 226 [666.72 597.48 743.99 535.44 199.59 1287.24] [2 3 4 5 6 7] [-11 -10 -12 -9 0 -17] [682995 683050 683099 683157 683198 683244] 0
1384889745329 6 227 [479.83 830.2 743.99 926.41 277.33 430] [2 3 4 5 6 7] [-8 -13 -12 -14 -3 -7] [684002 684047 684093 684145 684206 684251] 0
1384889746305 6 228 [277.33 7439.86 345.33 926.41 535.44 535.44] [2 3 4 5 6 7] [-3 -33 -5 -14 -9 -9] [685001 685045 685100 685150 685205 685254] 0
1384889747285 6 229 [1033.77 666.72 309.47 1033.77 199.59 385.35] [2 3 4 5 6 7] [-15 -11 -4 -15 0 -6] [685999 686049 686104 686145 686204 686260] 0
1384889748266 6 230 [666.72 430 248.53 2773.28 309.47 1602.87] [2 3 4 5 6 7] [-11 -7 -2 -24 -4 -19] [686997 687053 687093 687153 687209 687252] 0

1384889749242 6 231 [385.35 1033.77 385.35 926.41 199.59 926.41] [2 3 4 5 6 7] [-6 -15 -6 -14 0 -14] [688001 688047 688093 688156 688199 688244] 0
1384889750222 6 232 [535.44 597.48 277.33 535.44 199.59 535.44] [2 3 4 5 6 7] [-9 -10 -3 -9 0 -9] [689000 689054 689099 689152 689201 689253] 0
1384889751191 6 233 [479.83 1033.77 160.29 666.72 222.72 128.72] [2 3 4 5 6 7] [-8 -15 2 -11 -1 4] [690006 690050 690097 690152 690205 690249] 0
1384889752176 6 234 [535.44 1436.41 743.99 277.33 1995.89 1033.77] [2 3 4 5 6 7] [-9 -18 -12 -3 -21 -15] [690995 691051 691092 691152 691200 691254] 0
1384889753144 6 235 [597.48 385.35 1436.41 597.48 103.38 385.35] [2 3 4 5 6 7] [-10 -6 -18 -10 6 -6] [692003 692054 692101 692150 692199 692251] 0
1384889754126 6 236 [1033.77 597.48 926.41 597.48 160.29 309.47] [2 3 4 5 6 7] [-15 -10 -14 -10 2 -4] [692994 693046 693101 693144 693206 693250] 0
1384889755088 6 237 [926.41 926.41 666.72 597.48 385.35 743.99] [2 3 4 5 6 7] [-14 -14 -11 -10 -6 -12] [694000 694049 694101 694150 694198 694253] 0
1384889756076 6 238 [479.83 479.83 479.83 597.48 143.64 248.53] [2 3 4 5 6 7] [-8 -8 -8 -10 3 -2] [694995 695048 695103 695152 695199 695248] 0
1384889757051 6 239 [926.41 666.72 277.33 479.83 535.44 597.48] [2 3 4 5 6 7] [-14 -11 -3 -8 -9 -10] [696002 696053 696100 696148 696199 696244] 0
1384889758029 6 240 [1287.24 666.72 479.83 1788.62 926.41 479.83] [2 3 4 5 6 7] [-17 -11 -8 -20 -14 -8] [696996 697047 697094 697152 697202 697254] 0
1384889759011 6 241 [1033.77 743.99 277.33 248.53 74.4 248.53] [2 3 4 5 6 7] [-15 -12 -3 -2 9 -2] [697999 698056 698105 698145 698196 698251] 0
1384889759980 6 242 [597.48 309.47 345.33 666.72 160.29 926.41] [2 3 4 5 6 7] [-10 -4 -5 -11 2 -14] [699006 699049 699096 699150 699200 699244] 0
1384889760959 6 243 [597.48 597.48 830.2 277.33 1033.77 385.35] [2 3 4 5 6 7] [-10 -10 -13 -3 -15 -6] [699999 700055 700103 700157 700197 700255] 0
1384889761931 6 244 [1602.87 830.2 479.83 345.33 248.53 345.33] [2 3 4 5 6 7] [-19 -13 -8 -5 -2 -5] [700999 701043 701106 701145 701198 701250] 0
1384889762910 6 245 [535.44 597.48 1287.24 926.41 926.41 926.41] [2 3 4 5 6 7] [-9 -10 -17 -14 -14 -14] [702000 702049 702109 702147 702196 702247] 0
1384889763883 6 246 [830.2 430 385.35 430 178.86 830.2] [2 3 4 5 6 7] [-13 -7 -6 -7 1 -13] [703000 703054 703103 703152 703202 703255] 0
1384889764873 6 247 [535.44 743.99 385.35 535.44 597.48 199.59] [2 3 4 5 6 7] [-9 -12 -6 -9 -10 0] [704000 704048 704094 704149 704198 704250] 0
1384889765840 6 248 [535.44 535.44 743.99 160.29 479.83 248.53] [2 3 4 5 6 7] [-9 -9 -12 2 -8 -2] [705007 705051 705101 705147 705199 705243] 0
1384889766808 6 249 [535.44 479.83 830.2 430 222.72 248.53] [2 3 4 5 6 7]

[-9 -8 -13 -7 -1 -2] [706000 706058 706100 706151 706197 706256] 0
1384889767802 6 250 [430 2773.28 479.83 479.83 479.83 1287.24] [2 3 4 5 6 7] [-7 -24 -8 -8 -8 -17] [706995 707051 707103 707155 707204 707250] 0
1384889768771 6 251 [535.44 926.41 535.44 128.72 160.29 385.35] [2 3 4 5 6 7] [-9 -14 -9 4 2 -6] [708006 708052 708098 708148 708195 708248] 0
1384889769738 6 252 [277.33 743.99 597.48 830.2 1153.56 597.48] [2 3 4 5 6 7] [-3 -12 -10 -13 -16 -10] [709005 709059 709100 709146 709194 709255] 0
1384889770721 6 253 [277.33 1033.77 1153.56 1033.77 178.86 178.86] [2 3 4 5 6 7] [-3 -15 -16 -15 1 1] [709996 710045 710095 710154 710199 710252] 0
1384889771699 6 254 [666.72 535.44 479.83 222.72 160.29 597.48] [2 3 4 5 6 7] [-11 -9 -8 -1 2 -10] [710999 711050 711095 711150 711201 711250] 0
1384889772666 6 255 [597.48 597.48 597.48 222.72 1033.77 1033.77] [2 3 4 5 6 7] [-10 -10 -10 -1 -15 -15] [711998 712046 712103 712145 712202 712254] 0
1384889773650 6 256 [597.48 1436.41 666.72 666.72 479.83 385.35] [2 3 4 5 6 7] [-10 -18 -11 -11 -8 -6] [712994 713046 713104 713159 713202 713251] 0
1384889774632 6 257 [926.41 1287.24 926.41 199.59 345.33 430] [2 3 4 5 6 7] [-14 -17 -14 0 -5 -7] [714001 714050 714103 714159 714199 714254] 0
1384889775605 6 258 [479.83 597.48 309.47 479.83 199.59 199.59] [2 3 4 5 6 7] [-8 -10 -4 -8 0 0] [715003 715057 715101 715153 715207 715241] 0
1384889776585 6 259 [1602.87 309.47 743.99 830.2 1436.41 345.33] [2 3 4 5 6 7] [-19 -4 -12 -13 -18 -5] [716004 716046 716103 716150 716201 716257] 0
1384889777555 6 260 [430 597.48 1033.77 248.53 666.72 128.72] [2 3 4 5 6 7] [-7 -10 -15 -2 -11 4] [717002 717055 717108 717148 717203 717248] 0
1384889778539 6 261 [479.83 926.41 666.72 309.47 277.33 83.02] [2 3 4 5 6 7] [-8 -14 -11 -4 -3 8] [717996 718050 718099 718145 718202 718252] 0
1384889779505 6 262 [248.53 666.72 926.41 479.83 479.83 222.72] [2 3 4 5 6 7] [-2 -11 -14 -8 -8 -1] [719007 719046 719098 719153 719198 719250] 0
1384889780485 6 263 [535.44 535.44 430 248.53 666.72 2485.27] [2 3 4 5 6 7] [-9 -9 -7 -2 -11 -23] [719998 720043 720091 720154 720202 720244] 0
1384889781512 6 264 [1995.89 666.72 830.2 743.99 666.72 277.33] [2 3 4 5 6 7] [-21 -11 -13 -12 -11 -3] [720993 721052 721096 721148 721197 721251] 0
1384889782435 5 265 [277.33 926.41 385.35 385.35 128.72] [3 4 5 6 7] [-3 -14 -6 -6 4] [722046 722102 722144 722204 722248] 0
1384889783410 6 266 [666.72 479.83 666.72 535.44 926.41 178.86] [2 3 4 5 6 7] [-11 -8 -11 -9 -14 1] [722996 723048 723096 723145 723204 723254] 0
1384889784395 6 267 [277.33 345.33 535.44 385.35 1287.24 222.72] [2 3 4 5 6 7] [-3 -5 -9 -6 -17 -1] [723993 724058 724105 724146 724203 724250] 0

1384889785375 6 268 [345.33 535.44 345.33 199.59 1153.56 222.72] [2 3 4 5 6 7] [-5 -9 -5 0 -16 -1] [725000 725053 725096 725148 725207 725250] 0
1384889786344 6 269 [743.99 222.72 385.35 830.2 1033.77 128.72] [2 3 4 5 6 7] [-12 -1 -6 -13 -15 4] [726001 726046 726103 726142 726194 726250] 0
1384889787328 6 270 [277.33 597.48 479.83 160.29 743.99 115.36] [2 3 4 5 6 7] [-3 -10 -8 2 -12 5] [726998 727050 727105 727152 727198 727251] 0
1384889788301 6 271 [666.72 277.33 143.64 385.35 199.59 103.38] [2 3 4 5 6 7] [-11 -3 3 -6 0 6] [728003 728050 728099 728155 728194 728250] 0
1384889789275 6 272 [535.44 535.44 597.48 1788.62 666.72 199.59] [2 3 4 5 6 7] [-9 -9 -10 -20 -11 0] [729005 729054 729098 729142 729203 729247] 0
1384889790257 6 273 [743.99 385.35 479.83 830.2 535.44 535.44] [2 3 4 5 6 7] [-12 -6 -8 -13 -9 -9] [729993 730050 730107 730152 730204 730258] 0
1384889791232 6 274 [385.35 479.83 277.33 160.29 1033.77 597.48] [2 3 4 5 6 7] [-6 -8 -3 2 -15 -10] [731005 731050 731107 731144 731203 731251] 0
1384889792213 6 275 [597.48 479.83 222.72 143.64 597.48 115.36] [2 3 4 5 6 7] [-10 -8 -1 3 -10 5] [732002 732050 732099 732146 732197 732253] 0
1384889793179 6 276 [666.72 830.2 199.59 199.59 430 160.29] [2 3 4 5 6 7] [-11 -13 0 0 -7 2] [733001 733050 733100 733155 733198 733256] 0
1384889794165 6 277 [597.48 385.35 479.83 479.83 1033.77 143.64] [2 3 4 5 6 7] [-10 -6 -8 -8 -15 3] [733993 734048 734098 734152 734202 734253] 0
1384889795130 6 278 [926.41 830.2 178.86 143.64 160.29 103.38] [2 3 4 5 6 7] [-14 -13 1 3 2 6] [735006 735055 735093 735152 735197 735247] 0
1384889796114 6 279 [222.72 385.35 430 1602.87 430 128.72] [2 3 4 5 6 7] [-1 -6 -7 -19 -7 4] [735998 736043 736101 736145 736200 736247] 0
1384889797084 5 280 [666.72 1033.77 103.38 248.53 385.35] [3 4 5 6 7] [-11 -15 6 -2 -6] [737042 737100 737147 737200 737254] 0
1384889798065 6 281 [597.48 277.33 743.99 103.38 277.33 199.59] [2 3 4 5 6 7] [-10 -3 -12 6 -3 0] [737998 738053 738099 738149 738201 738246] 0
1384889799049 6 282 [597.48 309.47 222.72 160.29 479.83 430] [2 3 4 5 6 7] [-10 -4 -1 2 -8 -7] [739003 739053 739104 739149 739203 739243] 0
1384889800018 6 283 [830.2 248.53 199.59 103.38 1033.77 309.47] [2 3 4 5 6 7] [-13 -2 0 6 -15 -4] [740006 740053 740106 740150 740199 740248] 0
1384889800995 6 284 [160.29 743.99 345.33 345.33 666.72 597.48] [2 3 4 5 6 7] [2 -12 -5 -5 -11 -10] [741004 741053 741102 741148 741208 741254] 0
1384889801973 6 285 [1033.77 277.33 926.41 597.48 926.41 430] [2 3 4 5 6 7] [-15 -3 -14 -10 -14 -7] [741998 742050 742097 742146 742202 742249] 0
1384889802941 6 286 [926.41 597.48 926.41 199.59 743.99 160.29] [2 3 4 5 6 7] [-14 -10 -14 0 -12 2] [743001 743047 743104 743158 743200 743253] 0

1384889803927 6 287 [1436.41 479.83 345.33 199.59 199.59 160.29] [2 3 4 5 6 7] [-18 -8 -5 0 0 2] [743993 744046 744107 744156 744202 744244] 0
1384889804910 5 288 [178.86 535.44 535.44 830.2 597.48] [3 4 5 6 7] [1 -9 -9 -13 -10] [745045 745101 745154 745205 745249] 0
1384889805888 6 289 [178.86 345.33 248.53 597.48 597.48 597.48] [2 3 4 5 6 7] [1 -5 -2 -10 -10 -10] [746003 746051 746102 746152 746194 746254] 0
1384889806859 6 290 [666.72 666.72 1436.41 430 597.48 345.33] [2 3 4 5 6 7] [-11 -11 -18 -7 -10 -5] [747007 747046 747104 747153 747205 747256] 0
1384889807840 6 291 [103.38 430 479.83 830.2 248.53 178.86] [2 3 4 5 6 7] [6 -7 -8 -13 -2 1] [748004 748054 748110 748152 748195 748256] 0
1384889808806 6 292 [222.72 385.35 743.99 830.2 345.33 309.47] [2 3 4 5 6 7] [-1 -6 -12 -13 -5 -4] [749003 749053 749107 749158 749204 749253] 0
1384889809778 6 293 [743.99 479.83 277.33 178.86 666.72 430] [2 3 4 5 6 7] [-12 -8 -3 1 -11 -7] [749995 750046 750103 750149 750205 750251] 0
1384889810756 6 294 [103.38 743.99 830.2 53.54 535.44 535.44] [2 3 4 5 6 7] [6 -12 -13 12 -9 -9] [750997 751056 751097 751155 751201 751254] 0
1384889811740 6 295 [385.35 385.35 926.41 143.64 1153.56 1033.77] [2 3 4 5 6 7] [-6 -6 -14 3 -16 -15] [751999 752058 752096 752160 752199 752243] 0
1384889812717 6 296 [160.29 277.33 479.83 160.29 830.2 430] [2 3 4 5 6 7] [2 -3 -8 2 -13 -7] [753000 753054 753100 753150 753199 753250] 0
1384889813683 6 297 [430 160.29 535.44 160.29 1153.56 199.59] [2 3 4 5 6 7] [-7 2 -9 2 -16 0] [754002 754055 754102 754149 754205 754249] 0
1384889814663 6 298 [385.35 103.38 385.35 479.83 1995.89 830.2] [2 3 4 5 6 7] [-6 6 -6 -8 -21 -13] [754997 755050 755101 755154 755198 755255] 0
1384889815635 6 299 [128.72 2485.27 430 597.48 1995.89 830.2] [2 3 4 5 6 7] [4 -23 -7 -10 -21 -13] [756001 756054 756099 756144 756202 756248] 0
1384889816620 6 300 [143.64 309.47 597.48 160.29 597.48 830.2] [2 3 4 5 6 7] [3 -4 -10 2 -10 -13] [756993 757051 757102 757146 757203 757247] 0
1384889817599 6 301 [115.36 222.72 926.41 385.35 1153.56 277.33] [2 3 4 5 6 7] [5 -1 -14 -6 -16 -3] [757996 758042 758101 758158 758198 758255] 0
1384889818577 6 302 [92.64 222.72 597.48 115.36 743.99 926.41] [2 3 4 5 6 7] [7 -1 -10 5 -12 -14] [759002 759057 759095 759146 759196 759256] 0
1384889819549 6 303 [83.02 248.53 430 178.86 1153.56 743.99] [2 3 4 5 6 7] [8 -2 -7 1 -16 -12] [760000 760046 760100 760147 760197 760247] 0
1384889820520 6 304 [83.02 430 926.41 345.33 535.44 1436.41] [2 3 4 5 6 7] [8 -7 -14 -5 -9 -18] [760997 761058 761108 761151 761207 761250] 0
1384889821498 5 305 [103.38 222.72 385.35 743.99 385.35] [2 3 4 5 7] [6 -1 -6 -12 -6] [761996 762050 762101 762157 762252] 0

1384889822486 6 306 [128.72 248.53 926.41 597.48 1033.77 1788.62] [2 3 4 5 6 7] [4 -2 -14 -10 -15 -20] [762997 763047 763108 763160 763205 763255] 0
1384889823462 6 307 [309.47 309.47 535.44 309.47 535.44 743.99] [2 3 4 5 6 7] [-4 -4 -9 -4 -9 -12] [764005 764053 764099 764154 764199 764257] 0
1384889824437 6 308 [160.29 743.99 248.53 248.53 1788.62 1602.87] [2 3 4 5 6 7] [2 -12 -2 -2 -20 -19] [765008 765048 765106 765148 765203 765254] 0
1384889825414 6 309 [160.29 926.41 345.33 1153.56 1033.77 926.41] [2 3 4 5 6 7] [2 -14 -5 -16 -15 -14] [765998 766051 766104 766146 766200 766257] 0
1384889826390 5 310 [143.64 1287.24 199.59 830.2 2773.28] [3 4 5 6 7] [3 -17 0 -13 -24] [767048 767094 767149 767199 767254] 0
1384889827363 6 311 [160.29 479.83 430 248.53 535.44 597.48] [2 3 4 5 6 7] [2 -8 -7 -2 -9 -10] [768005 768048 768097 768151 768201 768254] 0
1384889828338 6 312 [430 535.44 385.35 535.44 926.41 666.72] [2 3 4 5 6 7] [-7 -9 -6 -9 -14 -11] [769004 769052 769100 769152 769204 769251] 0
1384889829317 6 313 [597.48 128.72 309.47 345.33 479.83 830.2] [2 3 4 5 6 7] [-10 4 -4 -5 -8 -13] [770001 770054 770104 770144 770206 770255] 0
1384889830285 6 314 [345.33 199.59 535.44 309.47 535.44 385.35] [2 3 4 5 6 7] [-5 0 -9 -4 -9 -6] [771004 771050 771105 771142 771196 771250] 0
1384889831269 6 315 [222.72 222.72 597.48 926.41 597.48 830.2] [2 3 4 5 6 7] [-1 -1 -10 -14 -10 -13] [771992 772049 772101 772151 772194 772251] 0
1384889832245 6 316 [222.72 479.83 535.44 535.44 535.44 2227.18] [2 3 4 5 6 7] [-1 -8 -9 -9 -9 -22] [773001 773051 773107 773144 773204 773255] 0
1384889833219 6 317 [143.64 248.53 479.83 1153.56 926.41 479.83] [2 3 4 5 6 7] [3 -2 -8 -16 -14 -8] [773997 774054 774097 774150 774206 774255] 0
1384889834203 5 318 [222.72 309.47 597.48 479.83 743.99] [3 4 5 6 7] [-1 -4 -10 -8 -12] [775045 775105 775156 775194 775243] 0
1384889835171 6 319 [430 160.29 535.44 479.83 535.44 1995.89] [2 3 4 5 6 7] [-7 2 -9 -8 -9 -21] [776007 776050 776109 776157 776196 776250] 0
1384889836148 6 320 [345.33 430 830.2 743.99 926.41 743.99] [2 3 4 5 6 7] [-5 -7 -13 -12 -14 -12] [776993 777053 777108 777149 777200 777245] 0
1384889837139 6 321 [277.33 535.44 277.33 830.2 430 222.72] [2 3 4 5 6 7] [-3 -9 -3 -13 -7 -1] [777998 778049 778109 778158 778205 778250] 0
1384889838103 6 322 [666.72 222.72 128.72 345.33 666.72 743.99] [2 3 4 5 6 7] [-11 -1 4 -5 -11 -12] [779005 779050 779098 779148 779203 779255] 0
1384889839076 6 323 [248.53 103.38 277.33 1033.77 1033.77 830.2] [2 3 4 5 6 7] [-2 6 -3 -15 -15 -13] [780000 780055 780097 780142 780196 780252] 0
1384889840060 6 324 [830.2 160.29 199.59 666.72 385.35 1033.77] [2 3 4 5 6 7] [-13 2 0 -11 -6 -15] [780998 781043 781102 781158 781200 781249] 0

1384889841031 6 325 [222.72 128.72 535.44 535.44 743.99 597.48] [2 3 4 5 6 7] [-1 4 -9 -9 -12 -10] [782000 782050 782104 782154 782204 782249] 0
1384889842009 5 326 [277.33 199.59 666.72 830.2 479.83] [3 4 5 6 7] [-3 0 -11 -13 -8] [783056 783105 783153 783204 783259] 0
1384889842985 6 327 [222.72 115.36 248.53 666.72 926.41 926.41] [2 3 4 5 6 7] [-1 5 -2 -11 -14 -14] [784003 784050 784103 784148 784194 784253] 0
1384889843967 6 328 [1153.56 103.38 160.29 1153.56 277.33 597.48] [2 3 4 5 6 7] [-16 6 2 -16 -3 -10] [785002 785053 785094 785151 785196 785247] 0
1384889844946 6 329 [597.48 830.2 1153.56 479.83 430 160.29] [2 3 4 5 6 7] [-10 -13 -16 -8 -7 2] [786000 786058 786095 786150 786201 786259] 0
1384889845928 5 330 [160.29 479.83 128.72 1033.77 430] [3 4 5 6 7] [2 -8 4 -15 -7] [787050 787099 787146 787203 787252] 0
1384889846893 6 331 [926.41 92.64 666.72 178.86 830.2 479.83] [2 3 4 5 6 7] [-14 7 -11 1 -13 -8] [788008 788045 788100 788148 788202 788253] 0
1384889847872 6 332 [597.48 222.72 385.35 248.53 926.41 385.35] [2 3 4 5 6 7] [-10 -1 -6 -2 -14 -6] [788999 789055 789096 789149 789203 789254] 0
1384889848846 6 333 [479.83 128.72 1033.77 143.64 1153.56 248.53] [2 3 4 5 6 7] [-8 4 -15 3 -16 -2] [790002 790050 790098 790158 790203 790256] 0

B.2 Plots of Distances for Observations in Appendix B.1

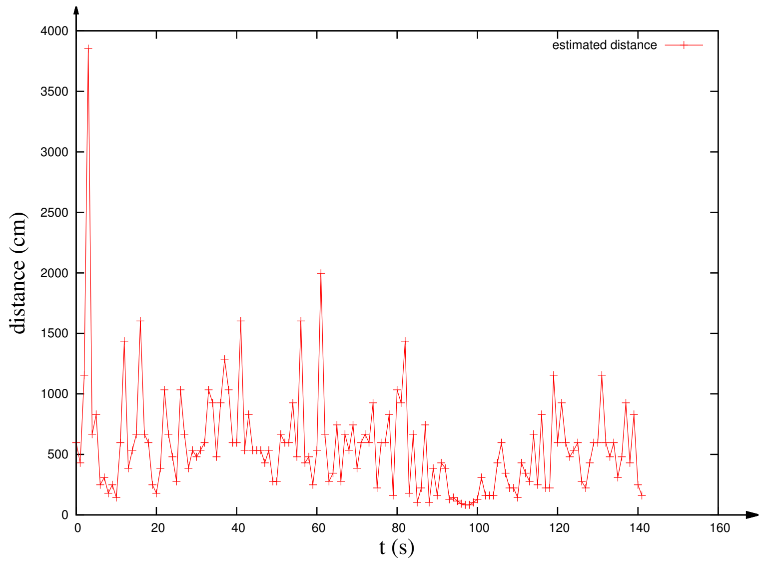


Figure B.1: Stationary node 2 distances.

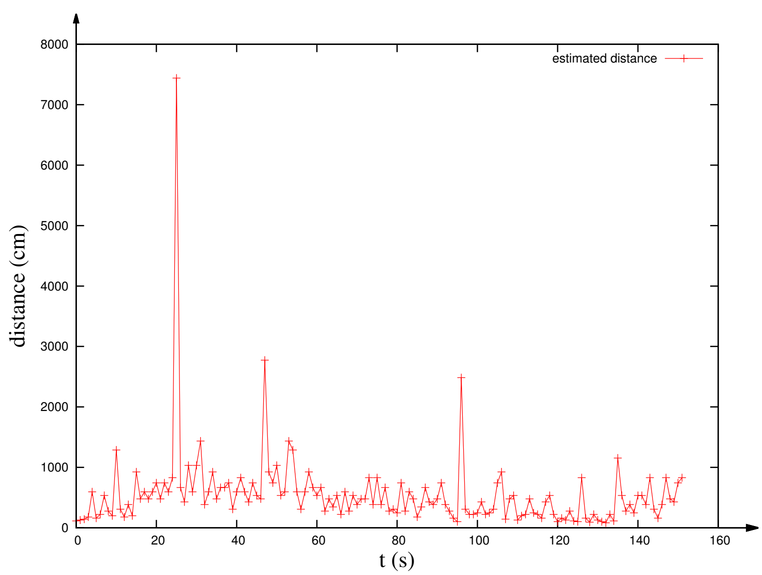


Figure B.2: Stationary node 3 distances.

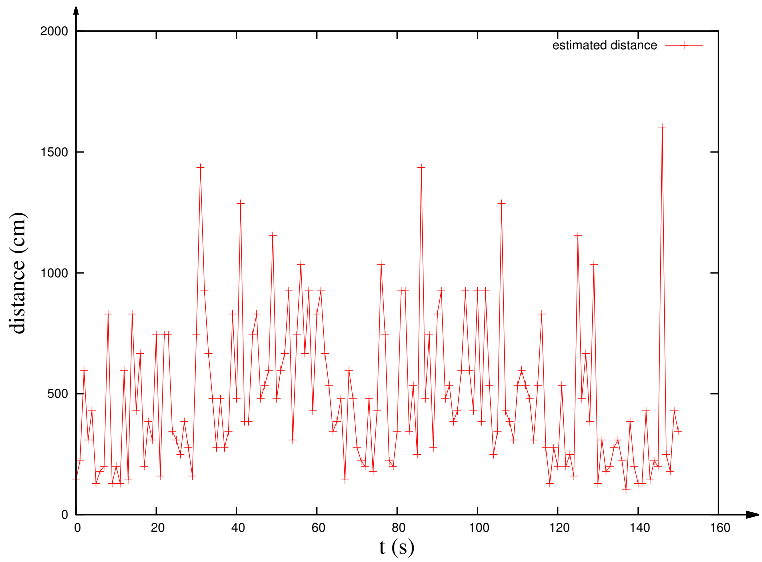


Figure B.3: Stationary node 4 distances.

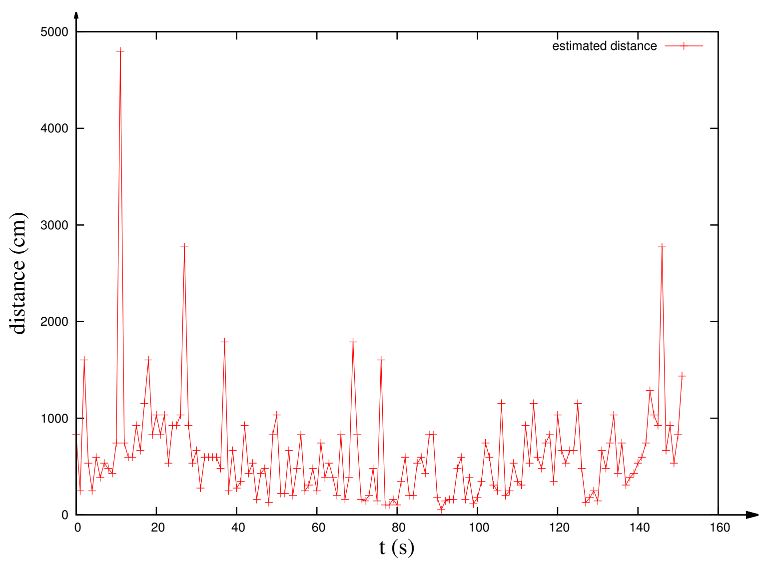


Figure B.4: Stationary node 5 distances.

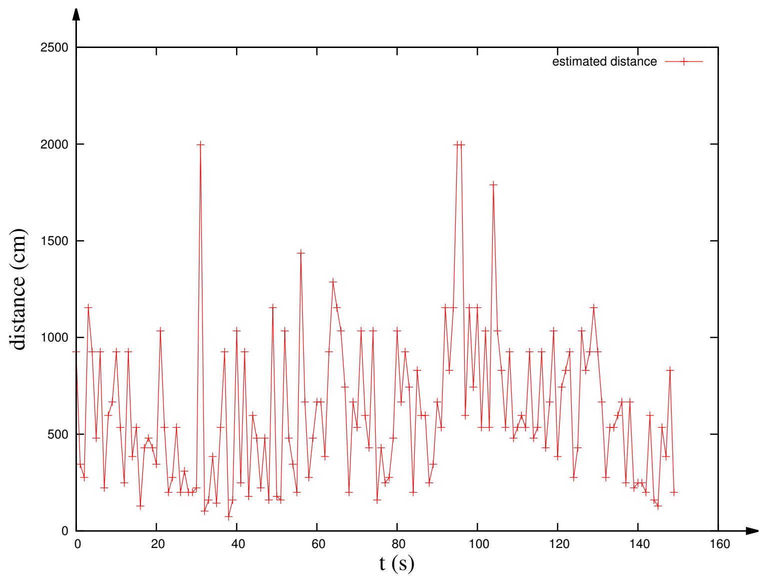


Figure B.5: Stationary node 6 distances.

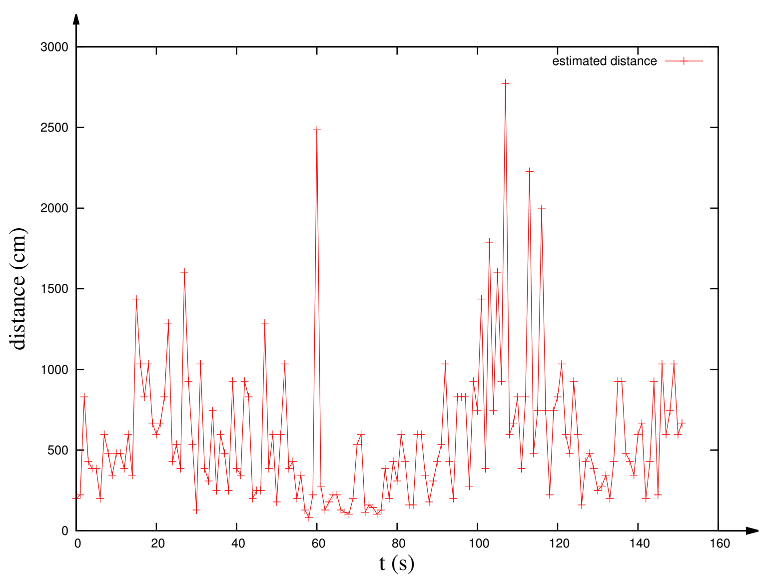


Figure B.6: Stationary node 7 distances.

Vita

Candidate's full name: Lingchen Zhou

University attended:
Aug. 2011 till now
University of New Brunswick, Fredericton, NB, Canada
Master of Computer Science

Sept. 2010 till June 2011
Southeast University, Suzhou, Jiangsu, China
Master of Software Engineering

Sept. 2005 till June 2009
Anhui University of Technology, Ma'anshan, Anhui, China
Bachelor of Computer Science

Reports:
Lingchen Zhou, Indoor Distance Determination Based on Received Signal Strength, University of New Brunswick Project Report for Course EE6514 Wireless Communications, December, 2011, 18 pages.

Posters:

Zhou, Lingchen and Nickerson, Bradford G., Position Estimation of Nodes Moving in a Wireless Sensor Network, poster at the 9th Annual UNB Faculty of Computer Science Research Exposition, Fredericton, N.B., April 10, 2012.

Zhou, Lingchen and Nickerson, Bradford G., Position Estimation of Nodes

Moving in a Wireless Sensor Network, poster at the 10th Annual UNB Faculty of Computer Science Research Exposition, Fredericton, N.B., April 12, 2013.