Range Based Indoor Localization in Wireless Sensor Networks with Telos

Examensarbete utfört i Kommunikationssystem vid Tekniska högskolan i Linköping
av

Petter Pehrson Skidén

LiTH-ISY-EX--13/4138--SE

Linköping 2013
Range Based Indoor Localization in Wireless Sensor Networks with Telos

Examensarbete utfört i Kommunikationssystem vid Tekniska högskolan i Linköping av

Petter Pehrson Skidén

LiTH-ISY-EX--13/4138--SE

Handledare: Josep Paradells Aspas
UPC, Barcelona, Spain

Examinator: Mikael Olofsson
ISY, Linköpings universitet

Linköping, 26 April, 2013
Title
Range Based Indoor Localization in Wireless Sensor Networks with Telos

Author
Petter Pehrson Skidén

Abstract
Localization of individual nodes in a wireless network is useful in many applications, e.g. for tracking patients in hospitals. Using the Received Signal Strength Indicator, RSSI, for this purpose has been explored in numerous studies. It is energy efficient and rarely requires customised hardware of configuration. The possibility to use pre-configured, off-the-shelf products is especially important in large scale sensor network deployments. Using RSSI has, however, many drawbacks, since the radio signal is heavily affected by the surrounding environment. Most studies in this area discuss the impact of multipath effects. Our study on range based distance estimations, using the Telos hardware, shows that individual profiling requirements and antenna quality are equally challenging.

Still, RSSI based indoor localization systems remains an active field of research. A multitude of approaches and algorithms have been proposed to gain accuracy in position estimations. The most common of these techniques are explored in this report. Based on previous work at The Polytechnic University of Catalonia, the Telos hardware has been integrated successfully with existing software to form local wireless sensor networks for indoor localization. We present applications developed on top of TinyOS, an operating system for embedded systems. These applications serve as a platform for related future work at The Polytechnic University of Catalonia and elsewhere.

Keywords
RSSI, WSN, Telos, Indoor Localization, Trilateration, nesC, TinyOS
In memory of Joakim Öhnell
Abstract

Localization of individual nodes in a wireless network is useful in many applications, e.g. for tracking patients in hospitals. Using the Received Signal Strength Indicator, RSSI, for this purpose has been explored in numerous studies. It is energy efficient and rarely requires customised hardware of configuration. The possibility to use pre-configured, off-the-shelf products is especially important in large scale sensor network deployments. Using RSSI has, however, many drawbacks, since the radio signal is heavily affected by the surrounding environment. Most studies in this area discuss the impact of multipath effects. Our study on range based distance estimations, using the Telos hardware, shows that individual profiling requirements and antenna quality are equally challenging.

Still, RSSI based indoor localization systems remains an active field of research. A multitude of approaches and algorithms have been proposed to gain accuracy in position estimations. The most common of these techniques are explored in this report. Based on previous work at The Polytechnic University of Catalonia, the Telos hardware has been integrated successfully with existing software to form local wireless sensor networks for indoor localization. We present applications developed on top of TinyOS, an operating system for embedded systems. These applications serve as a platform for related future work at The Polytechnic University of Catalonia and elsewhere.

Sammanfattning

Lokalisering av individuella noder i trådlösa sensornätverk är användbart i många applikationer, till exempel för att söka upp patienter i sjukhusbyggnader. Att utnyttja Received Signal Strength Indicator, RSSI för detta ändamål har undersökts i ett flertal tidigare arbeten och anses i allmänhet som en attraktiv lösning eftersom det är energisnålt och oftast innebär att ingen specialutvecklad hårdvara behövs. Möjlichten att kunna använda vanliga off-the-shelves-produkter blir en viktig faktor om man sätter upp storskaliga sensornätverk. RSSI har dock signifikanta nackdelar eftersom radiosignaler påverkas mycket av saker i omgivningen. Mängder av tidigare publikationer presenterar olika påverkan från effekten av bland annat flervägsfel. I det här examensarbetet visar avståndsmätningar med sensornoden Telos att ytterligare utmaningar består av individuell profilering och kvalitet på antenner.
Acknowledgments

First of all I would like to thank my examiner Mikael Olofsson for your thoroughness, valuable advice and patience. The last of which appears to be infinite. I would also like to thank my supervisor Josep Paradells and his staff at UPC for welcoming me into their lab in Barcelona. A special thanks to Jordi Vilaseca Posa who provided lots of insight into the existing localization system.

I would also like to thank my family for the love and support I have always received as well as for all the adventures we have done, and will keep doing, together.

Thanks to my friends around the world, who have taught me so many things and with whom I have had some of the best moments in my life. I am grateful to have met all of you.

Finally, I would like to thank my lovely Lisa who always inspires me with new ideas and supports me no matter what I do or prioritize in this world. I love you.

San Francisco, April 2013
Petter Skidén
# Contents

1 Background and Work Description 3
   1.1 Localization in Wireless Sensor Networks .................. 4
   1.2 Our Field of Interest ....................................... 5
      1.2.1 Existing Localization System with Cricket Motes .... 5
   1.3 Thesis Objectives ........................................... 7
      1.3.1 Literature Study ........................................ 7
      1.3.2 Generate Path-Loss Models for Distance Estimation with
            RSSI on the Telos Platform .................................. 7
      1.3.3 Porting the Localization System to Telos and Applying a
            New Localization Algorithm ................................ 8
      1.3.4 Ad-hoc Correction for Automatic Adjustment .......... 8
   1.4 Thesis Outline ................................................ 8

2 Literature Study of Range Based Localization Methods 11
   2.1 Common Ranging Techniques in WSNs ......................... 12
      2.1.1 Time of Arrival .......................................... 12
      2.1.2 Time Difference of Arrival ................................ 12
      2.1.3 Received Signal Strength Indicator, RSSI ............. 13
   2.2 Centralized Localization and Position Estimation .......... 15
      2.2.1 Localization Algorithm Of The Existing System At UPC . 15
      2.2.2 Localization Algorithms In Other Work .................. 16
   2.3 Summary and Conclusions from Literature Study .......... 19

3 Telos and TinyOS 21
   3.1 Introduction ................................................... 21
   3.2 Developing Applications for the Telos in TinyOS .......... 22
   3.3 Communication Between Telos Motes with TinyOS .......... 23
      3.3.1 Routing Capabilities ..................................... 23
   3.4 Radio Emitting Power .......................................... 24
   3.5 Antenna Characteristics ....................................... 24

4 Measuring RSSI for Range Based Localization with Telos 27
   4.1 Introduction ................................................... 27
   4.2 NesC Application on Telos .................................... 28
   4.3 The Java Listener and Logger ................................ 29
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AODV</td>
<td>Ad hoc On-Demand Distance Vector Routing</td>
</tr>
<tr>
<td>AM</td>
<td>Active Message. Message type in TinyOS</td>
</tr>
<tr>
<td>GDoP</td>
<td>Geometric Dilution Of Precision</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>i2CAT</td>
<td>IT Foundation in Catalonia, Spain</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>INLR</td>
<td>Iterative Nonlinear Regression</td>
</tr>
<tr>
<td>LMS</td>
<td>Least Median of Squares</td>
</tr>
<tr>
<td>LQI</td>
<td>Link Quality Indicator</td>
</tr>
<tr>
<td>Mote</td>
<td>Sensor node in a wireless sensor network</td>
</tr>
<tr>
<td>NesC</td>
<td>Network Embedded Systems C</td>
</tr>
<tr>
<td>NLR</td>
<td>Nonlinear regression</td>
</tr>
<tr>
<td>RANSAC</td>
<td>Random Sample Consensus</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RSSI</td>
<td>Received Signal Strength Indicator</td>
</tr>
<tr>
<td>TDoA</td>
<td>Time Difference of Arrival</td>
</tr>
<tr>
<td>TIX</td>
<td>Triangular Interpolation and eXtrapolation</td>
</tr>
<tr>
<td>ToA</td>
<td>Time of Arrival</td>
</tr>
<tr>
<td>ToF</td>
<td>Time of Flight</td>
</tr>
<tr>
<td>UPC</td>
<td>Universitat Politècnica de Catalunya</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
</tbody>
</table>
Chapter 1

Background and Work Description

A wireless sensor network, WSN for short, consist of geographically distributed autonomous devices equipped with sensors for monitoring physical or environmental conditions. Common applications for wireless sensor networks are measuring of temperature, sound, vibration, pressure and light at different locations. [47]

A WSN often uses a mesh network layout, which means that the nodes themselves decide how to route packets to other nodes in the network and no traditional routers are needed in between. Figure 1.1 shows two typical networks; star and mesh. In a star layout traffic is always routed through a central node in the network. This is a typical layout in a residential network where the router also serves as a gateway to a local internet provider. In a mesh network there is no central point. In such a network all nodes have to take routing responsibility. [8]

Traditionally, the deployment of WSNs were mainly of interest for military applications such as target detection, battlefield surveillance and tracking of military vehicles. However, during the last few years, it has become common to use wireless sensor networks in industrial and civilian applications such as industrial process monitoring, healthcare and traffic control. [19]

A sensor device, also known as a mote, is usually the actual hardware that forms a node in the network. A mote performs tasks individually and can forward data to the rest of the network by some wireless communication technique often radio communication but for certain applications light or ultrasound devices are used. [26] A more advanced device as a laptop or stationary computer could also
form part of the network, as a gateway to the internet for example. Such a node, on the edge of the network, is normally called a sink node. [13, c. 12.2]

In addition to one or more sensors, a mote usually consists of a radio transceiver, or some other wireless communications device, a microcontroller, and some kind of energy source, for example a battery. The size of a sensor node can vary widely depending of technical requirements such as energy life time, computational speed, memory and bandwidth. In [47] an interval between shoebox and a grain of dust serves as an example although science has not yet developed motes of microscopic dimensions.

For a sensor network to achieve its purpose it is often essential to know the origin of the information sensed. In applications where motes are fixed at a constant location it is easy to record the location and get a complete map of the WSN at question. Unfortunately, this is not always the case. Many WSNs contain moving nodes where positions need to be updated continuously and often without the possibility of manual measurements and calibration. Localization of sensor nodes is therefore an inevitable challenge in some WSN applications and the field has been studied for many years. [43]

1.1 Localization in Wireless Sensor Networks

The problem of localization in a WSN is defined by [43] as;

*Estimating the position or spatial coordinates of the sensor nodes according to some system of coordinates, global or local.*
In some WSN applications the nodes of the network can be placed manually in fixed configurations that allows for the system to know the exact coordinates of each sensor. For a dynamic network where nodes can move or if nodes are placed into the network without accuracy (e.g. thrown from airplanes over a forest) there has to be some kind of localization mechanism for the gathered sensor data to mean anything. Nodes can be equipped with a Global Positioning System (GPS), but this is a costly solution in terms of size, money and power consumption. For sensors used indoors there are even more reasons not to use GPS as a mean of localization as it requires line-of-sight to satellites. [45]

Localization indoors is often more complex than outdoors because of the shifting environment. Objects such as walls, floors, ceilings and furniture blocking line-of-sight between sensor devices and can make for dramatic errors in estimation of distances, angles and signal power. [43]

A common approach in indoor localization is to assume that some sensor nodes already know their own positions. Those types of nodes are often called beacon or beacon nodes. [6]

Beacons are used to estimate the positions of non-beacon nodes either by measurements of direct distances, called a range based technique explained in [7], or by a range free technique, discussed in [54]. A range-free strategy would be counting number of hops in the routing of communication between the nodes for example. A localization algorithm can then be applied to calculate a best fit from several measurements between different combinations of nodes.

## 1.2 Our Field of Interest

In the fall of 2007 a research group formed by staff at i2CAT and UPC in Barcelona, Spain made a demonstration of a localization mechanism for a WSN they had developed. The idea was to investigate the possibility to build a self-configuring network that was able to localize another node simply by knowing the coordinates of the reference nodes. The focus was however not on localization efficiency but rather on developing a functional system, a framework, that permitted later improvements and new functionality. The main part of this work was documented and published in Catalan in [40] by Jordi Vilaseca and following sub chapter 1.2.1 is a summary of that work.

### 1.2.1 Existing Localization System with Cricket Motes

The localization system consists of a number of Cricket [11] motes sold by Crossbow Technology Inc. The Cricket is a low power processor module
equipped with a 433 MHz-band radio and an ultrasound device. In the working system a handful of Crickets were used as beacons with known coordinates forming a mesh network. Each beacon has a unique identifier and can receive commands from a computer outside the network through a Cricket working as a gateway, a sink node, to the regular network. The responsibility of the beacons in the network is to estimate the position of a non-beacon placed nearby. Figure 1.2 displays a Cricket mote.

![Cricket mote](image)

**Figure 1.2.** Cricket mote equipped with ultrasound device and serial port connector. Image from [11]

The localization system at UPC is range based. It combines ultrasonic signals with radio signals to estimate distances between the motes in the network. A localization query can be sent from a PC, acting as a sink node, to a specific non-beacon node. The non-beacon responds to this query by sending a broadcast message together with an ultrasonic pulse to all other nodes in the network.

The listening beacons first receives the radio message and then counts the time until the arrival of the ultrasonic pulse. This technique is widely known as Time Difference of Arrival, TDoA, discussed in the next chapter. The beacons also measures the strength of the radio signal sent from the non-beacon. The data is then sent back to the PC to be used in a localization algorithm.

A Java program was developed for communication and calculation of localization data on the PC. It contains some classes where the data collected by the beacons can be transformed to distance estimates. Under the assumption that the propagation speed of the radio signals is infinite and the ultrasound waves travel with the speed of sound at some fixed temperature the program calculates a distance from the measured TDoA value. Distance estimation from the measured signal attenuation is done according to an empirical formula deducted from limited testing in the lab at UPC.

Potentially, the program can work with two distance estimates between non-beacons and beacons. The TDoA measurement and the power reading from the radio signal in the incoming message. However, because of the higher precision of the TDoA estimation the Java application will prefer to use that value if the ultrasonic pulse reaches the reference node. To calculate a position
estimate for a non-beacon node, the system uses a trilateration algorithm.

Distance values from at least three separate beacons are needed to calculate an estimation of position. The program trilaterates data from every group of three reference nodes that have returned distance values but at the end it returns the position estimate from the group of references with the lowest distance estimates under the assumption that references closer to the mobile node return the most precise measurements.

1.3 Thesis Objectives

The localization system formed by the Cricket motes worked successfully in terms of communication and data gathering. Many possible additions and improvements were identified. This thesis work is intended to cover some of those tasks. The research group has been involved in a couple of projects where another sensor device called Telos [33] has been used. It uses a IEEE 802.15.4 [23] compliant radio and communicates over the 2.4GHz band.

It is desired to port the localization system to work for the Telos hardware and to do some tasks explained in the following subsections. Porting the localization mechanism to the Telos means rewriting software for the new hardware and possibly some further development of the Java application. A literature study will be done to gather some information for the different tasks. The work will be done for i2CAT in the laboratory shared at UPC in Barcelona and will serve as a source of knowledge for future work done by the research team at UPC.

1.3.1 Literature Study

To gain some knowledge about range based localization a study of previous work in the indoor localization will be made. It will include information about the most common ranging techniques and also cover some localization algorithms possible to implement in the existing localization system. It should also explore the Telos platform and TinyOS, the operating system that will be used on Telos. An introduction to TinyOS has been done by Philip Levis in [27].

1.3.2 Generate Path-Loss Models for Distance Estimation with RSSI on the Telos Platform

Part of the thesis work is to try to create a more accurate propagation model for the distance estimation between the sensor nodes with the signal attenuation
technique. Calculating distances by measuring the attenuation of signal strength of packets sent over radio can be done by using the Received Signal Strength Indicator, RSSI further discussed in 2.1.3.

Partly this task is also intended for learning to use TinyOS on the Telos motes for later activities. Software for measuring the radio signal strength in arriving packets, logging the data and compute adjusted curves for a range of distances is also part of this task. A brief investigation of antenna characteristics could possibly be useful for additional understanding about how precision is affected by the placement of the motes.

1.3.3 Porting the Localization System to Telos and Applying a New Localization Algorithm

As mentioned earlier the system is to be used with Telos motes. Software is to be developed for the sensor nodes and modifications in the Java application will follow. The focus in this subtask is on making the existing localization mechanism work with Telos motes and then to modify the Java code to make it easier to add new algorithms. A localization algorithm from the literature survey is to be chosen and implemented.

1.3.4 Ad-hoc Correction for Automatic Adjustment

A general problem with the distance measurements using RSSI is loss of signal strength because of walls and other obstacles. If additional time exists the task is to attempt to implement a calibration mechanism between reference nodes to make automatic adjustments to sudden offsets in signal attenuation. Knowing the actual distance between the references should make it possible to discover walls and correct position estimates in a post processing sense.

1.4 Thesis Outline

The outline of this thesis is laid out as follows: Chapter 2 is a literature study of work in the area of indoor localization exploring the most common ranging techniques and localization algorithms. Chapter 3 describes the Telos sensor node and the TinyOS operating system.

That is followed by chapter 4 in which some RSSI experiments with Telos are presented. Chapter 5 covers the development of a Telos based WSN built for
localization queries from the existing Java application at UPC. Finally section 6 contains conclusions from chapter 4 and 5 together with suggested future work.
Chapter 2

Literature Study of Range Based Localization Methods

Localization in WSNs can be classified as either being range-based or range-free. The thesis work focused solely on range-based localization and therefore an extended comparison between the two techniques will not be included in this document. However, the topic is discussed in [10] and [12].

The principle in range-based localization is to use absolute point-to-point measurements between nodes to calculate coordinates. [7] Estimating the absolute distance between two sensor nodes require a ranging technique, that is, some way of measuring.

There are numerous of ranging techniques to estimate the distance between two points. Time of Arrival, ToA and Time Difference of Arrival, TDoA are examples of deducting distance estimates from measuring propagation time between two nodes, described in 2.1.1 and 2.1.2. RSSI, the method of calculating distance from the attenuation of signal strength is discussed in 2.1.3.

Often the unit of the obtained from measurements is time or loss of signal strength. A model is therefore necessary to convert the results to an estimated distance. If multiple distance measurements are available a localization algorithm can be applied to get more precise results. An example of a node localization technique is trilateration or in the general case with more sources, multilateration. [30][58] Multilateration can be done with many different algorithms, further mentioned in chapter 2.2.
2.1 Common Ranging Techniques in WSNs

2.1.1 Time of Arrival

Assuming the environment is coherent enough for signals to travel at constant propagation speed, distance estimation between two sensor nodes can be calculated by measuring the time it takes for the signal to travel between two nodes. This is generally called *Time of Arrival* or sometimes *Time of Flight*. This technique however, requires that the two points have synchronized clocks. It works well in the case of the Global Positioning System which uses synchronized atomic clocks in the satellites thus giving very high accuracy. [59]

It is also used in mobile communication outdoors where there are very long distances between endpoints. However with the hardware limitations and the constraints of energy consumption for sensor networking devices it is a very costly and inaccurate technique. [52] Using radio signals propagating at the speed of light, require clocks with nanosecond precision to archive accuracy within a meter.

The off-the-shelves products used at i2CAT, like the Telos and Cricket motes, use time stamping with millisecond accuracy in their radio packets. [11][33] This yields a very poor accuracy and makes the ToA technique not at all suitable for an indoor localization system like the one used at UPC. Using acoustic signals, like ultrasound, with significantly lower propagation speed will of course increase accuracy. Clocks with 1 ms precision allows for about 35 centimeter accuracy. Ultrasonic signals have the advantage of being less vulnerable to multipath disturbance than RF signals but has a poorer range and requires more power. [40]

2.1.2 Time Difference of Arrival

Another way of measuring distances using propagation time is the Time Difference of Arrival method. [22] A sending node transmits a radio and ultrasonic signal at the same time. The radio signal arrives at the receiving end essentially instantaneously and the ultrasonic signal takes much longer and the distance between the two nodes can be calculated from the time difference. This is a way of avoiding the clock synchronization problem and offers an accuracy level of a couple of centimeters for distances between nodes in the range of 3-4 meters. [60]

One of the pioneers of the Time Difference of Arrival approach in indoor localization is [42]. The localization system at i2CAT implements the same technique with Cricket motes. [40] For this work however it is not a viable option as the scenario considered for this work is a WSN consisting of purely Telos
2.1 Common Ranging Techniques in WSNs

motes without ultrasonic capabilities.

2.1.3 Received Signal Strength Indicator, RSSI

A popular way of estimating distances is by using the Received Signal Strength Indicator, RSSI. A few examples of such work are [56][45] and [36]. RSSI is taking advantage of the fact that energy of an electromagnetic wave, like a radio signal, decreases as it propagates in space. [51, p. 68-73]

If a radio signal is sent between two nodes and the initial power of the transmitted signal is known the receiving end can compare that value with the measured strength of the received signal and estimate the distance based on the signal attenuation. Estimating distances with RSSI is an attractive alternative in a WSN as the sensor nodes do not need any extra hardware which means using less energy. [7]

The main disadvantage with RSSI is that the errors in this type of distance measurement can be quite high. This is concluded in numerous work like [40] [56] and [45]. Obstacles and multipath effects of the environment especially in harsh indoor environments densely populated by objects and people makes it impossible to define a globally valid relation between loss of signal power and physical distance. The errors grow even larger in a dynamic environment as things move and fixed models or empirical tables lose validity. [14]

In [24] Krishnan et. all point out advantages such as low power consumption and low cost because of mass produced hardware but they also discuss problems with multipath propagation and signal attenuation due to obstacles such as walls etc.

The measured signal attenuation is often obtained in dBm from the hardware and can be transformed to a distance estimate according to a propagation model [51, p. 214-215]. The model could be empirically known or theoretical or a combination of the two. In an empirical model measurements have been made with the existing hardware and every power reading maps to a certain distance value. It is well known that the power of the received radio signal decays with increasing distance depending on the obstacles that surround the transmitter and the receiver. The environment characteristics vary greatly between indoor and outdoor. [40]

An often used theoretical model for indoor propagation is the Log-Distance Path Loss Model [21][44] shown in 2.1.

\[ P_L(d) = P_L(d_0) + 10n\log\left(\frac{d}{d_0}\right) + X_\rho \]  

(2.1)
\( P_L(d) \) is the total path loss expressed in decibels, \( d_0 \) is a reference distance, \( n \) is a path loss exponent depending on the surrounding environment and \( d \) is the path between transmitter and receiver. \( X_p \) is a Gaussian random variable with zero mean and standard deviation of \( \sigma \) dB. It is explained in more detail in [51]. [40] presents a table with proposed values for \( n \) in different indoor environments.

In the existing localization system at i2CAT however, the model in 2.2 is used. [40]

\[
P(d) = \frac{P_0}{d^n}
\]

In the model \( P(d) \) is the received signal strength at a distance \( d \) from the transmitting node and \( P_0 \) is the incoming power at \( d = 1m \). \( P_0 \) is a calibrated value and depends on the original signal power from the transmitting side, signal frequency, wave length and antenna gain.[40] The power losses are proportional to \( d^{-n} \) where \( n \) is a parameter that depends on the environment. \( n \) depends on if there is line-of-sight or not, signal frequency used, density of objects in the environment and type of material used in the building construction among other things.

Path-loss in free-space yields a value of \( n = 2 \) but in indoor environments this value will vary. A greater value means effectively that the signal attenuation is greater, normally caused by objects in the way of the signal. A lower value than 2 is also possible in certain environments, for example hallways, where reflections from parallel walls or other objects could increase the received signal power. [21]

According to [40] the value of \( n \) should be \( 1 < n < 5.22 \) but the lower bound is likely to be higher. [5] presents a table with recorded values of \( n \) from measurements in some typical indoor environments. The lowest value of \( n \) from their measurements was 1.6, found in a metalworking environment.

Experiments with empirical values for this module have been done in [44] and [5] and values for \( n \) are suggested for different scenarios. In the experimental setup at i2CAT the path loss exponent can be changed manually. A general conclusion in the field of indoor propagation is that there does not exist a universally valid propagation model. [21]
2.2 Centralized Localization and Position Estimation

As mentioned in the beginning of this chapter a distinction is made between estimating absolute distances (ranging) and using the ranging data to estimate a position (node localization). Ranging can be done with numerous of different techniques, some of them discussed above and then a node localization algorithm can work with any of them. Node localization can be classified as either distributed or centralized.[55]

In distributed localization computations are made by the whole network. A node estimates its own position in the network relative other nodes and there is no central unit involved. Some examples of work in distributed localization are [25][41]. In this thesis centralized localization is the focus and further discussions of distributed localization is excluded. In continuation a summary of the existing centralized localization algorithm and some algorithms used in other work are presented.

Centralized localization algorithms require the gathering of connectivity data from the network to a more computationally powerful device. This generates more traffic on the network than distributed localization algorithms. A drawback with distributed computing however is that more energy is consumed by computations on the motes.[55] That often means a restriction on the complexity of the localization algorithms because of battery life-time requirements. As the system developed at UPC is using centralized localization the problem of restricted energy sources can be disregarded.[40]

2.2.1 Localization Algorithm Of The Existing System At UPC

During the localization project with Cricket motes that resulted in the existing system at UPC the focus was not on making advanced centralized localization algorithms but to investigate the possibilities to create a basic mesh network of communicating sensors that had the ability to make positioning estimates with both RSSI and ultrasound techniques.[40]

The Java application which was used on a PC contains some calculation classes that implement a localization algorithm based on Trilateration described in greater detail by [30]. The application collects information from every available beacon in the network and trilateration is then used to calculate node positions from groups of three beacons.

The input data to this algorithm consists of a set of coordinates from every
beacon \( x_i, y_i \) and estimated distances to the mobile node, \( d_i \). The localization algorithm then finds the best possible fit to these input parameters and returns an estimated position.

For each beacon a residual is defined as in equation 2.3. The system assumes that the altitude of the mobile node is known and calculates a 2-dimensional estimate with the nonlinear least squares method also called Nonlinear regression, NLR. To make the system over determined for a 2-dimensional plane at least three nodes with known coordinates are required.\[40\]

\[
e(i) = \sqrt{(x_0 - x_i) - (y_0 - y_i) - d_i}
\]  

(2.3)

We define the sum square error \( E_{ss} \) as the sum of all residuals in 2.4.

\[
E_{ss} = \sum_{i=1}^{n} e(i)^2
\]  

(2.4)

The least squares solution is the unique solution \((x_0, y_0)\) that minimizes the sum of the squares of the residuals. The Newton Gradient method is used to find this global minimum.\[48\] This is a numerical method and requires an initial starting point i.e. a guess at the solution. It then iterates a determined number of times with an applied correction from the previous solution to reduce the sum of squared residuals.

This calculation is done for every possible group of three beacon nodes but at the end the position estimate from the three beacons with the strongest RSSI values is used. This is done under the assumption that the measurements from the three nodes with the strongest signals are the most reliable and accurate. \[40\]

### 2.2.2 Localization Algorithms In Other Work

The most common technique for range based localization systems is to use different algorithms based on trilateration as described above. Some examples of systems using this technique are [7][34][20]. Also numerous evaluations of different algorithms for trilateration and multilateration have been published. The following sections explores some of those algorithms.
Iterative Nonlinear Regression (INLR)

A single application of nonlinear regression requires reliable data to perform well according to [46] and [61]. The solution of a single NLR analysis gives the best fit based on all input parameters but has no mechanism to reject outliers. To use all measurement data and to incorporate a way of rejecting outliers is to iteratively form NLR models reducing the data size for every iteration. This means starting off with calculating a single NLR solution based on all available data from the all beacons and then discard the measurement that disagree with the fit the most.

The algorithm then starts over without the residual associated with the identified measurement. This process is repeated until a chosen threshold is reached or if there are insufficient measurements to compute a fit. The final sum square error is used as the predicted error and the computed coordinates in the last iteration is the position estimate of the mobile node.

This algorithm was evaluated in [46] and considered the most accurate of the five localization algorithms deployed. In the paper they also conclude that INLR had a higher computation cost than the alternatives. This however is not a big concern in a centralized localization system like the one developed at i2CAT where calculations are made by a PC.

Least Median Of Squares, LMS

In [28] and [9] the robust Least Median of Squares is proposed as applicable in localization for wireless sensor networks. It is a robust variant to the Least Squares algorithm. The LMS method was first introduced by Rousseeuw and described in [49]. The algorithm uses all available measurements from the beacons and create subsets of three.

The sum square error is computed for all possible triplets and for every solution the median is identified from the three residuals in the equation. After all triplets have been minimized. The equation with the lowest median of the residuals is chosen instead of the equation with the smallest total error. This approach does not try to identify bad measurements (outliers), rather it is a method to try to live with them but without giving them a lot of weight in the results.

Random Sample Consensus, RANSAC

Another algorithm capable of identifying outliers is Random Sample Consensus, RANSAC for short introduced by [15]. The classic algorithm uses trilateration with groups of the minimal number of beacons (in our case three) to calculate a
position estimate \((x_0, y_0)\). This triplet is randomly selected and all other calculated estimates are used to either support or not support the results based on some given error tolerance.

A calculated position is used if sufficient data readings in the rest of the data set supports the results. If rejected, another triplet is randomly picked and compared in the same manner. If there is no supported result to find the algorithm could either return a fail or use the most supported estimation.

**Triangular Interpolation and eXtrapolation, TIX**

In [18] the authors introduce a calibration-free location algorithm (TIX) based on trilateration. It has a different approach to the multilateration technique used in the existing application but with the same input parameters i.e. the known positions of the reference nodes and distance measurements to the mobile node. The algorithm requires exactly three beacons to form a triangle whose corners are the known coordinates of the beacons. The TIX uses dividers of the sides derived from the distance measurements to compute a centroid of the triangle as the final location estimate as in figure 2.1.

![Figure 2.1](image)

**Figure 2.1. To left: Interpolation** Centroid formed by internal dividers \(D_1, D_2, D_3\). **To right: Extrapolation** Centroid formed by internal divider \(D_2\) and external dividers \(D_1\) and \(D_3\).

If more beacons are available this method can be used iteratively for all possible beacon combinations to filter outliers. The authors in [18] conclude that while their TIX algorithm does eliminate the need to calibrate their distance readings it does not achieve as high precision as trilateration algorithms where calibrated distance values are used.[18]

**Geometric Dilution Of Precision (GDoP)**

The geometry set-up for the beacons alone can affect localization accuracy. This is a known effect called Geometric Dilution of Precision and is presented in [50].
2.3 Summary and Conclusions from Literature Study

They suggest a method where a classification of all position estimations, regardless of algorithm used, makes it possible to reject some combinations of beacon node positions. Savvides et al also present results from experiments with different beacon node configurations concluding that the geometric setup affects the localization results regardless of algorithm used.

2.3 Summary and Conclusions from Literature Study

This chapter has explored some of the most commonly used ranging techniques such as ToA, TDoA and RSSI for estimating absolute distances between motes, ubiquitous sensor devices, forming part as nodes in a WSN. Numerous work has been done in the ranging are referenced throughout chapter 2.1.

This literature study concluded that ToA and TDoA is not suitable for estimating distances as the focus of this work at UPC is to use Telos in its off-the-shelves configuration. This leaves only RSSI as a viable technique for distance estimations.

In chapter 2.2 the trilateration in the existing localization system at UPC was presented as well as some different trilateration algorithms from other work. As the existing system calculates estimated positions at a central point, in a Java application on a PC all of these algorithms could be potential candidates to be implemented in later tasks or in future work at UPC.
Chapter 3

Telos and TinyOS

This chapter is intended to summarize the investigation of the Telos sensor mote that is going to be used in future work revolving wireless sensor networks at UPC and i2CAT. For specific and fully detailed information about the Telos hardware a datasheet is available here at the vendors website [33].

Figure 3.1. Telos Rev. B presented in [33]

3.1 Introduction

The Telos, developed at UC Berkeley, is an ultra low power wireless module aimed for experimentation and research in wireless sensor networks. It is equipped with a Chipcon CC2420 IEEE 802.15.4-compliant radio that uses the 2.4 GHz band [39]. Each device has, apart from the radio module, a MSP430 microcontroller from Texas Instruments with 10kB RAM, a set of integrated sensors for humidity, temperature and light and an integrated antenna with up
to 100 meters of radio range outdoors. It also carries a USB connector for serial communication with for example an ordinary PC. The devices are powered with a supply voltage between 2.1 and 3.6 volts that can be provided either via the USB connector or by connecting a battery pack with batteries as a mobile power source.

The operating system used on the hardware is TinyOS, a small, open source, energy-efficient operating system developed by UC Berkeley. [57] TinyOS has a component based architecture that makes for easy application development. It comes with a software library composed of network protocols, distributed services, data acquisition tools and drivers for the most common sensors. TinyOS is under constant development and has been ported to many platforms. The latest current release is version 2.1.2 (August 20 2012).

3.2 Developing Applications for the Telos in TinyOS

To get started developing in TinyOS the principal forum is the TinyOS Documentation Wiki website referenced in [16]. It offers tutorials, open source software contributions, mailing lists and a development forum. Additionally it contains a number of easy to follow beginner lessons to introduce the application structure used.

The programming language for TinyOS is called Network Embedded Systems C, NesC for short. It is closely related to C, with similar syntax, but designed for embedded systems. The programming model includes event-driven execution and component-oriented application design. [17]

Applications in TinyOS consists of components, wired together during compilation. A component is made up from one or more tasks and its behavior is defined by interfaces. The interfaces can be implemented (provided) by the component or just used by it and provided from elsewhere. An interface is used to describe what operations the component can offer and it also describes which functions other components have to implement to use the operations offered.[27]

NesC code is cross-compiled on a PC with the TinyOS environment installed and then flashed to the mote in a binary including both the application and the operating system. It works under with all the common operating systems like Linux, MacOS X and Windows (Cygwin). Once loaded, the mote runs its application independently. Software compiling and deployment is covered step by step in the TinyOS Getting started guide [4].

To interact with a WSN from an ordinary computer one or more motes in the
3.3 Communication Between Telos Motes with TinyOS

TinyOS provides a number of interfaces to use the underlying radio communication services and a number of components that provide (implement) these interfaces. All of these interfaces and components use a common message buffer called TOSMsg which is implemented as a nesC struct (similar to a C struct). Chris Merlin at University of Rochester has some good examples of how to send and receive data packets over radio in [32].

The TOSMsg struct is available for the user to define messages to be sent to other nodes. Radio communication in TinyOS follows the Active Message (AM) model. Each packet consists of two parts; the fixed 10 byte AM header, defined in the AM header file in the TinyOS library, and the data payload containing the actual message. The payload size is of course dependent of the application used and the amount of data necessary. These structures are covered by the Mote to Mote Communication Tutorial in [2].

3.3.1 Routing Capabilities

For building mesh networks with our sensor nodes at UPC we are using a routing protocol called Ad hoc On-Demand Distance Vector. It is specified in RFC 3561 by Internet Engineering Task Force, IETF [38]. The AODV implementation for TinyOS 1.x were made at UPC by former students. [35]

It is a reactive protocol which means that it establishes a path to a destination only on demand. That makes the traffic on the network less and energy is saved. For a software developer that wants to use the protocol a number of interfaces are provided. Packets in the communication will have an additional AODV header encapsulating the application data under the AM header. [40]
3.4 Radio Emitting Power

The signal output power on the Telos is coded by numbers between 1 to 31 and it is possible to change this power level in runtime. The Telos data sheet supplies a mapping table to dBm for some of the values displayed in the following table.

<table>
<thead>
<tr>
<th>Power level</th>
<th>dBm</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>-1</td>
</tr>
<tr>
<td>23</td>
<td>-3</td>
</tr>
<tr>
<td>19</td>
<td>-5</td>
</tr>
<tr>
<td>15</td>
<td>-7</td>
</tr>
<tr>
<td>11</td>
<td>-10</td>
</tr>
<tr>
<td>7</td>
<td>-15</td>
</tr>
<tr>
<td>3</td>
<td>-25</td>
</tr>
</tbody>
</table>

The CC2420 radio module on the Telos provides a RSSI value for each packet that may be read any time. Upon packet reception the first eight bits are sampled and an error rate is calculated. From that value a link quality indication (LQI) value is produced for each received packet. [53]

\[ RF_{power}(dBm) = -45 + RSS_{measured}(\text{decimal}) \]  

(3.1)

Both the RSSI and LQI values are coded on one byte. The RSSI is mapped to dBm according to a chart in the Telos data sheet [33]. Like the data sheet describes the power in dBm is computed as the received signal strength on decimal form minus an offset of roughly 45 as in eq. 3.1

3.5 Antenna Characteristics

The Telos mote is shipped with an integrated antenna with an indoor range of 20-30 meters. There is also an option to use an external antenna connected through a SMA connector. [33] According to the data sheet the internal antenna is not completely omnidirectional. Something that will affect distance estimation as the orientation of the motes would become a factor. The data sheet also supplies two figures with horizontal and vertical radiation pattern shown in figures 3.2 and 3.3.
Figure 3.2. Horizontal radiated pattern for the internal antenna on Telos. Image taken from [33]

Figure 3.3. Vertical radiated pattern for the internal antenna on Telos. Image taken from [33]
Chapter 4

Measuring RSSI for Range Based Localization with Telos

4.1 Introduction

To compute distances between two nodes using loss of strength in RF signals as indicator it is necessary to have some kind of path-loss model as mentioned in chapter 2. In the existing localization application at UPC an empirical model was used for the Cricket motes. It was derived from measurements at one occasion in one of the labs at the university. For more information regarding that work see [40]. That work however, primarily focused on distance measurements using ultrasound.

It was desired to develop an application for measuring signal attenuation and to obtain empirical path-loss models also for the Telos mote. Partly to investigate if better estimation accuracy could be achieved by analyzing the collected data and identify possible distortion due to hardware effects on the Telos. Another reason was to get an easy-to-use tool for measuring connectivity on different set-up sites e.g. hospitals, trains etc for other possible applications outside the scope of this thesis.

The data assembly tools created consisted of two main components:

- A nesC application to run on the Telos motes for communication and
measuring of signal attenuation

- A Java application running on a PC to receive data from the motes via serial communication over USB.

For analyzing data and drawing graphs, to display results, Matlab was used. The applications were then to be used to perform a number of experiments on the Telos hardware, further discussed in this chapter.

4.2 NesC Application on Telos

The application for the Telos motes is written in nesC and will be called RssiCollector from now on. The source code is attached in appendix A. Two Telos motes are used when measuring RSSI with RSSICollector and they both run the same version of the application.

It was decided to collect RSSI measurements from the Telos at different levels of transmission power. This would give some understanding about the range possibilities for a certain level of signal strength on the Telos and could potentially be used as a range indicator. The application automatically collects data from all transmitting power levels.

At startup the motes are put in a waiting state. To begin the data gathering one of the Telos motes is initialized with the user button on the hardware, see product datasheet [33] for complete overview of buttons on the Telos. The activated mote will then start to send a fixed amount of messages to the other mote at the highest signal power level, 31 (0 dBm). After all packets have been received it sends an equal amount of packets at power level 27 and continuing down in steps of four to power level 3 (-25 dBm).

When all packets have been sent from the first Telos mote, the receiving mote requests to send messages the opposite way. The motes change roles and the process is done all over again. Figure 4.1 shows the different set-ups. Data is saved by direct forwarding to the PC via USB cable where the recordings are stored in a text file.

Each packet consists of the fixed 10 byte AM header followed by the payload. The payload size is of course dependent of the application used and the amount of data necessary. There are 6 data fields used in rssiCollector; txid, rxid, rl, cnt, strength, lqi. All of them with a size of 1 byte.

- \textbf{txid} - The identification number of the mote acting as beacon.
4.3 The Java Listener and Logger

In the Java packages that comes with the TinyOS installation there are classes for sending and receiving packets from the UART serial connection over USB between PC and motes. The listener application in the TinyOS installation [16] was modified to receive the measurements and store them in text files.

The signal strength and lqi value are computed at the time of reception automatically by the hardware and stored in the TOSMsg structure. The values are not however included in the AM header of the packet. Therefore it has to be added to the payload in the application code when the message event is triggered. The values can be reached by fetching the values ‘strength’ and ‘lqi’ from the TOSMsg structure. The code snippet below shows how this is done.

```c
pack = (struct ChirpMsg *)tosmsg.data;
pack -> rssi = m -> strength;
pack -> lqi = m -> lqi;
```

$m$ is a pointer pointing at the first element of the received message. The pointer $pack$ is pointing to the data field of a new message that is sent to the PC and the values are stored in the payload in the new message.

Figure 4.1. Set-up for data gathering. The application can be used in two set-ups.

- **rxid** - The identification number of the mote acting as mobile client.
- **rl** - Level of transmission power used when sending the packet.
- **cnt** - Packet count for a certain signal power level. Together with the transmission power it forms a unique id for the message.
- **strength** - The signal strength of the received packet.
- **lqi** - The link quality value. Stored for possible future use.

The signal strength and lqi value are computed at the time of reception automatically by the hardware and stored in the TOSMsg structure. The values are not however included in the AM header of the packet. Therefore it has to be added to the payload in the application code when the message event is triggered. The values can be reached by fetching the values ‘strength’ and ‘lqi’ from the TOSMsg structure. The code snippet below shows how this is done.

```c
pack = (struct ChirpMsg *)tosmsg.data;
pack -> rssi = m -> strength;
pack -> lqi = m -> lqi;
```

$m$ is a pointer pointing at the first element of the received message. The pointer $pack$ is pointing to the data field of a new message that is sent to the PC and the values are stored in the payload in the new message.

4.3 The Java Listener and Logger

In the Java packages that comes with the TinyOS installation there are classes for sending and receiving packets from the UART serial connection over USB between PC and motes. The listener application in the TinyOS installation [16] was modified to receive the measurements and store them in text files.

The signal strength and lqi value are computed at the time of reception automatically by the hardware and stored in the TOSMsg structure. The values are not however included in the AM header of the packet. Therefore it has to be added to the payload in the application code when the message event is triggered. The values can be reached by fetching the values ‘strength’ and ‘lqi’ from the TOSMsg structure. The code snippet below shows how this is done.

```c
pack = (struct ChirpMsg *)tosmsg.data;
pack -> rssi = m -> strength;
pack -> lqi = m -> lqi;
```

$m$ is a pointer pointing at the first element of the received message. The pointer $pack$ is pointing to the data field of a new message that is sent to the PC and the values are stored in the payload in the new message.
4.4 Post-processing

On the PC side some Matlab functions were written to read the RSSI values from the data files and to compute mean values and variance for each measurement batch. The RSSI values are adjusted according to the original transmission strength to get the attenuation value. A mean value of all collected data for each distance measurement can be calculated for every measurement point. The data can then be fitted with different models using the Curve Fitting Toolbox in Matlab and constants are calculated. More information about the Curve Fitting Toolbox can be found in [31].

4.5 RSSI Experiments

4.5.1 Measuring Setup

The physical measurement setup was divided in two scenarios. The first setup was having all motes at the same level of 1 meter. One mote acting as a beacon mounted on the wall and another mote acting as a mobile client placed on a pedestal on top of a desk.

The normal placement of a beacon mote however is on a wall at around 2.5 meters. This is the altitude used from previous projects [40] and it was considered to be a place where the nodes are not in danger of being harmed physically from people or objects that are being moved around. A mote is also more likely to have line-of-sight to other motes if it is at a high altitude. The mote acting as mobile client was always put at an altitude of 1 meter which was considered a possible height of a person wearing the mote as a wristband. This scenario was used as a second setup.

Both motes were connected to a computer with USB cables. The receiving mote on each measurement forwarded the measured RSSI to the logging application in the PC.

The mote acting as a mobile client started sending 100 packets for every power level to the beacon mote. All 8 power levels shown in table 3.4 were used meaning 800 packets in total. This procedure was then repeated in the opposite direction with the mobile client mote as receiver. Everything was logged in files with a modified version of the Listener application included in the TinyOS installation.

The RSSI measurements between the motes were done every meter from 1 to 6 meters. Figure 4.2 and 4.3 shows the two set-ups.
4.5 RSSI Experiments

The measuring was done in the lab at UPC with some people working nearby but with a direct line-of-sight between the motes during transmissions. For these measurements line-of-sight was assumed as the signal attenuation would be significantly impacted by a person or object in the way. There were objects like electronic equipment, furniture and walls nearby which was considered part of the normal indoor environment. They will however also affect in terms of multipath and reflection disturbances.

4.5.2 Applying a Simple Model

A simple power function 4.1 was applied as a first attempt to model the signal attenuations and visualise the data from the RSSI experiments.

\[ P(d) = ax^b \]  

(4.1)

Using the curve fitting toolbox in Matlab you get the constants \(a\) and \(b\) computed from the applied data set. Models were generated from each of the
levels of measured transmission power and also a combined model was calculated with all measurements. The resulting model curves were then plotted together with the mean values and the variance at each distance from 1 to 6 meters.

### 4.5.3 Results and Observations Beacon at 1 meter

The first experiment was to collect RSSI measurements between two nodes at the same altitude, one meter above floor level as shown in figure 4.2. Plots from these measurements are shown in figures 4.4-4.11.

The collected data presented in the plots clearly show that the received signal strength at each point is not declining as expected. Even though motes are placed at the same altitude, within line-of-sight and antennas facing each other there is a consistent pattern across all measurements where for example the node at 2 meters receives a weaker signal on the incoming packets than the node at 3 meters. In all measurements the mote at a distance of one meter shows by far the lowest RSSI which of course is expected. Points between 2-6 meters however, are not consistent in returning RSSI values in declining order.

As discussed in chapter 2, a major source of error when using RSSI is due to multipath effects caused by objects in the environment. In the office environment, where the tests were performed, the radio environment is likely to change between every measurement point as the room contains quite many things that could cause multipath effects.

For indoor sites like the one used here it is reasonable to assume that the characteristics of radio environment can change for every signal wavelength as desks and floor etc will affect the received signal strength. In this case with 2.4 GHz radio signals that is about 12.5 centimetres. To get a more exact model of a specific office space the measurements need to be done much more frequent than one meter and most likely from every single beacon used. In this initial work with the Telos however, the scope has been to get some understanding of the quality of a generic path-loss model without taking into account every object causing the radio environment to change. That is, it is assumed that the localization system would apply the same path-loss model on measurements from all beacons with minimal calibration. As the graphs show this assumption results in very coarse estimations.

Even though a densely populated office area affect the RSSI, the deviations at each point suggest significant disturbances that does not pertain to local variations in the radio environment. According to [37] the standard deviation of the received power in units of dB should be relatively constant with distance in a static environment. From these initial measurements the standard deviations did not show that. In figure 4.7 and 4.8 for example the RSSI values vary
significantly between measurement points without any visible pattern.

Throughout all measurements the RSSI values on the Telos could suddenly deviate slightly for a couple of samples and then go back to being more consistent. As the environment was constant during the whole measurement process one possibility is that it is related to some problem with the mote itself, either hardware or software. Experiments with the first setup was done at two other locations with similar office environment resulting in the same conclusions.

It is important to note that, in these initial path-loss measurements with the Telos, the mote is used purely as it has been shipped. No individual profiling of the radio or antenna characteristics have been made on the motes. They are regarded as "ready to use" off-the-shelves products. To be able to rapidly deploy full scale localization networks it would be very costly to profile and calibrate each and every mote. This assumption however, will of course have a negative effect on the resolution of distance estimates with RSSI when using these devices.

In [29] the authors contribute valuable insight of the challenges with using RSSI for distance measurements on low-power radio devices using the same Chipcon CC2420 radio as on the Telos. One of the conclusions is that even in an environment with minimal multipath effects the antenna orientation on simple devices with low-power radios will have significant effect on the received signal strength.

Although the motes have been mounted with the antennas facing each other with the same orientation at every point in this these experiment it is very likely that the angles are a bit different. This is a source of error that cannot be excluded from these measurements. In an application where the antenna orientation of the mote is completely unknown this would make the signal attenuation even harder to use for distance measuring.

[29] conclude for their part that RSSI for these types of devices are only feasible for measuring distances up to 3-4 feet, about one metric meter. The results in the plots in this chapter could suggest something similar as it is a significant difference in signal attenuation between 1 meter and all other measurement points but more measurements at shorter distances are required to support such a statement.

4.5.4 Results and Observations Beacon at 2.5 meters

Measurements from the second setup, displayed in figure 4.2, are presented in figures 4.12-4.19. When placing the beacon node at 2.5 meters still with a direct line-of-sight and more or less the same scenario as the first experiment the measurements not only suffers from the same disturbances as in setup 1 but also
**Figure 4.4.** Beacon altitude 1 meter. 0 dbm transmitting power.

**Figure 4.5.** Beacon altitude 1 meter. -1 dbm transmitting power.
Figure 4.6. Beacon altitude 1 meter. -3 dbm transmitting power.

Figure 4.7. Beacon altitude 1 meter. -5 dbm transmitting power.
Figure 4.8. Beacon altitude 1 meter. -7 dbm transmitting power.

Figure 4.9. Beacon altitude 1 meter. -10 dbm transmitting power.
4.5 RSSI Experiments

Figure 4.10. Beacon altitude 1 meter. -15 dbm transmitting power.

Figure 4.11. Beacon altitude 1 meter. -25 dbm transmitting power.
Figure 4.12. Beacon altitude 2.5 meter. 0 dbm transmitting power.

has an extra dive in attenuation for short distances like at 1 and 2 meters.

The main reason being that the antennas on the Telos nodes are not completely omnidirectional as mentioned in chapter 3.5. A different angle between the antennas will cause a serious decline in RSSI. This phenomena was explored rather informally by sending continuous messages between a fixed mote and a mote moving in a circle around the fixed one. When the moving mote does not immediately face the fixed beacon the received signal strength can drop dramatically.

Note that the points of measurement in this setup can not be treated as absolute distances between the motes as the motes have been placed on different altitudes. This however was considered secondary in these initial experiments as it still gives an indication of what RSSI values to expect over an increasing distance from the beacon.

An interesting observation is that the RSSI is declining as expected for measurement points between 3 and 6 meters in this setup. This raises questions about how the motes where mounted in the first setup as the RSSI in those measurements where much more inconsistent. It is possible that the desks where the motes where mounted infringes in a more negative way in the first setup as they interfere much more with the path to each measurement point in that case.
**Figure 4.13.** Beacon altitude 2.5 meter. -1 dbm transmitting power.

**Figure 4.14.** Beacon altitude 2.5 meter. -3 dbm transmitting power.
Figure 4.15. Beacon altitude 2.5 meter. -5 dbm transmitting power.

Figure 4.16. Beacon altitude 2.5 meter. -7 dbm transmitting power.
Figure 4.17. Beacon altitude 2.5 meter. -10 dbm transmitting power.

Figure 4.18. Beacon altitude 2.5 meter. -15 dbm transmitting power.
Figure 4.19. Beacon altitude 2.5 meter. -25 dbm transmitting power.
Chapter 5

Localization System using Telos

The existing localization system at UPC by [40] described in chapter 1.2.1 relies on Cricket motes as nodes in the WSN. The main ranging method is TDoA using the ultrasonic device mounted on the Cricket but there is also an option to use RSSI. The applications running on the Cricket motes are not possible to run on the Telos as they are two different hardware platforms. The Java application however could be reused for RSSI values measured by the Telos motes.

New code had to be developed for the Telos motes to be able to act on incoming commands from the Java application running on the PC. Three NesC applications were written for the Telos motes depending on what role they have in the WSN. The three roles in the network are, Sink node, Beacon node and Client node. The sink node is the gateway between the PC and the WSN. The beacon node is a reference node with a known position which measures RSSI from a client node and reports results back to the PC. The client node is the node which has an unknown position and is going to be located.

For this simplistic WSN with only 5-10 nodes static addressing was used. That means that all the nodes in the network have their addresses configured at installation time and they will never change unless the application is reinstalled on the mote.

TinyOS provides the Active Message, AM, interface mentioned in 3.3 which makes it possible to statically assign an address to each mote. This can be done when flashing the application onto the Telos motes by adding a flag with a simple integer as address. For example by running the command make install.1 when flashing the mote from command line would give a
particular mote the address equal to 1. The static address can then be used when sending AM messages between the motes.

At this time the system is restricted to only do single hop communication in the network. That means that we always send messages to the actual destination. It is thereby assumed that all nodes in the WSN can reach each other with radio communication.

Communication Overview

Figures 5.1 and 5.2 show the communication flow when a localization is triggered from the Java application on the PC. The sink node is used to relay a command message to the client which starts to broadcast a number of packets to the beacons. The beacons reads the RSSI for each packet and saves it to memory. When the client is done broadcasting the beacons send the results in one chunk to the PC via the sink node.

![Diagram showing communication flow](image)

**Figure 5.1.** Localization is started on the laptop. A localization command message is sent to the client node which in turn starts broadcasting a preconfigurable number of packets to all available beacons.
5.1 Sink Node Application

The sink node is acting as a gateway between the PC and the WSN. It is connected to the PC with a USB cable. Its responsibility is to relay messages between the Java application and WSN. Both the NesC configuration and module source code for the sink node is attached in appendix B.1.

When a search is initiated from the PC the sink node will get an event with an AM message via serial communication over USB. The sink node needs to extract the data from the AM message to get the actual information sent from the PC. Listing 5.1 shows how this was implemented guided by the tutorial in [2].

// Receive event for TelosTrackingCmdMsg from PC. Forwarding to client event TOS_MsgPtr ReceiveTelosCmdMsg.receive(TOS_MsgPtr pmsg){
    if (len == sizeof(TelosTrackingCmdMsg)) {
        trackptr = (TelosTrackingCmdMsg *)pmsg -> data; // Get the data

        // Copy the destination from the message to the address in the AM header
        client_node_address = trackptr->dest;
        call SendMsg.send(client_node_address, sizeof(TelosTrackingCmdMsg),
            &trackptr);
    }
    return pmsg;
}
Listing 5.1. Upon message reception the address for the client is copied from the data struct and put as destination in the AM header when the message is forwarded.

In TinyOS receiving a message is done by implementing the function `receive` on the `ReceiveMsg` interface which in this case is aliased as `ReceiveTelosCmdMsg` in listing 5.1. This function gets called by TinyOS when a message arrives. The messages in TinyOS are ordinary c structs and are typically defined in one or more header files. The three messages used in the system are presented in appendix B.4.

The message used to trigger a localization from the Java application is called `TelosTrackingCmdMsg` displayed in listing 5.2. It is what is put in the data field of the AM message structure. It contains 4 fields; `seqno` is a sequence number for future use. `source` is the address of the sender. `dest` is the address of the client node that is being localized and `number_of_chirps` is the number of messages the client node should send to the beacons for RSSI measurements.

```c
typedef struct TelosTrackingCmdMsg {
    int8_t seqno;
    uint16_t source;
    uint16_t dest;
    uint8_t number_of_chirps;
} TelosTrackingCmdMsg;
```

Listing 5.2. The localization query message format sent from the Java application

5.2 Client Application

The mote being localized has only one task which is to act on incoming command messages from the sink node by broadcasting a number of messages, `chirps`, to all available beacons. This is implemented by using the TinyOS interface `Timer` described in [1]. The timer is started by calling `Timer.start` with two arguments; the TinyOS enum `TIMER_REPEAT` which instructs the timer that it should trigger the event multiple times, and an integer representing the number of times the timer should repeat. To use the Timer interface the function `Timer.fired()` needs to be implemented. This will get called when the timer times out. The complete source code for the application is attached in appendix B.2. The code snippet in listing 5.3 shows the implemented event to the timer which sends the broadcasts. Just for convenience, the motes will light the red led
5.3 Beacon Application

on the Telos board while they are broadcasting.

```c
event result_t ChirpTimer.fired(){
    cnt = cnt + 1;

    call SendMsg.send(TOS_BCAST_ADDR, sizeof(ChirpMsg), &data);

    if(cnt == MAX_NUM_CHIRPS){
        call Leds.redOff();
    }
    return SUCCESS;
}
```

Listing 5.3. The fired function gets called upon timeout. TOS_BCAST_ADDR is an enum provided by TinyOS to broadcast the message to all other participants in the WSN.

5.3 Beacon Application

The source code for the beacon application is attached in appendix B.3. The beacon motes are constantly waiting for incoming ChirpMsg messages from the client mote. When all measurement messages have been received the Beacons send back all values in one chunk to the PC via the sink node. In the current version the beacons are expecting a fixed amount of messages to arrive.

The beacons receive each message and read the RSSI, implemented as demonstrated in chapter 4.2, and add it to an array. When the first message is received a timer is started that will time out and cause the beacon to send the results back even if messages are lost. The RSSI values are stored in the field `chirpValues` in message struct LocMsg attached in appendix B.4.3.

Typically all beacons will want to report their measured RSSI values at the same time. To avoid packet loss when reporting the data back to the sink a basic back-off mechanism was implemented. Before a beacon wants to send data, a random value is generated and used as a delay. The maximum value that was used was 2 seconds which has proved to be sufficient when running the system in the office environment at UPC.
5.4 Making Localization Queries From a Laptop

When motes have been flashed and mounted in nearby places, all human interaction with the WSN is done from the Java application written by Jordi Vilaseca and described in greater detail in [40]. The application is run on a regular laptop where the sink node is connected with a USB cable. A Linux machine has been used at the lab but there are TinyOS tool support for communicating with the Telos motes over USB on Windows as well.

The main window consists of a table where beacon coordinates in centimeters can be entered (top left), a map of the office environment at UPC (right half), some buttons to initiate localization requests and fields to adjust the propagation model (mid left) and a panel for output (bottom left). Depending on what coordinates are chosen for the beacons the application redraws the map. The beacons are represented as blue circles along with their coordinates.

![Figure 5.3. Screenshot from the Java application originally developed by Jordi Vilaseca and documented in [40].](image)

After placing the client mote somewhere in the office a location request is initiated by clicking the Locate button. The localization phase takes a couple of seconds and when results are back from the motes the map is repainted with an estimation and which of the three beacons where used in the trilateration. An example of a localization is shown in figure 5.4. Estimations from all available beacons will be displayed. The estimated point of the client is shown as a gold colored dot. The beacons with the shortest estimation is highlighted with a thick yellow circle around it.

The application applies the least sum of squares method as described in chapter
5.5 Initial Results and Observations

To verify that the Telos motes were properly integrated with the Java application a brief series of localization runs were conducted. The main observation is that the system is working with the Telos motes in the same way as with the previous Cricket motes. The motes are able to communicate, react on localization requests from the laptop and report back via the sink.

In-depth experiments to improve this Telos based localization system are left for future work. Even though these initial tests only serve as examples the results indicate the same pattern as from what was seen in chapter 4. In rare occasions when all beacons have line-of-sight to the client mote and antenna angles are favourable, localization results can be fairly accurate like in figure 5.5.

As expected, when the above conditions are not true, if for example there are obstacles blocking line-of-sight or antennas are not directly facing each other, the
error in estimation grows very big even though one of the beacons could be very close to the client. In figure 5.6 one of the beacons was placed on the other side on a thin wall and a second beacon was placed within line-of-sight but with a slightly steeper angle between the antennas. The light pink dot shows the true location of the client. As results from all beacons are weighted equally the estimated position drifts far to the right.

Figure 5.5. Example of localization with line-of-sight between client mote and all beacons. The pink dot is the true position of the client.
Figure 5.6. Example of localization without line-of-sight between one beacon and the client.
Chapter 6

Conclusions and Future Work

This chapter presents conclusions from the RSSI measurements with the Telos presented in chapter 4 and suggestions to future work in range based estimation with the Telos. In addition conclusions from chapter 5 are presented along with possible future work on the Telos based localization system at UPC.

6.1 RSSI Measurements with Telos

An application, RSSICollector, for collecting RSSI values from received packets has been developed for the Telos mote to be used in future experiments at UPC and i2CAT. Together with the Listener application included in the TinyOS tool stack it is possible to do RSSI measurements between two Telos motes and persist the data on a normal laptop. With this application a couple of initial experiments were conducted.

The main conclusion from these measurements is that it is very challenging to do reliable distance measuring with the Telos for short distances in a typical office space without knowledge about antenna direction. Previous work in the field has shown the challenges with multipath effects caused by objects in the nearby environment or directly blocking line-of-sight. The results from the measurements in chapter 4 shows that antenna orientation and lack of hardware profiling makes up for considerable additional difficulties in estimating distances with the Telos.
The Telos internal antenna used has been verified not being omnidirectional and affect the RSSI heavily depending on how the mote is facing. In a scenario where a beacon was placed at a higher altitude than the mobile clients the RSSI readings returned much lower values at the closest measurement point than the next two points further away due to a narrower angle between antennas. In a more demanding scenario where there is no knowledge about antenna orientation, which is what ultimately is aimed for at UPC, it would impossible to do any distance estimations as measurements have shown that, even with line-of-sight, a client could be located virtually anywhere in the testbed.

One way of mitigating this problem would be for UPC to use external antennas in future work. This would however make the mote considerably less ubiquitous which could make it an unfeasible solution if the mote was to be worn as a wristband at a hospital for example.

The RSSI measurements also experienced problems with seemingly random deviations of the RSSI. This highlights the importance of proper profiling of the motes. It can be very costly when deploying large scale networks but at this time it is not possible to estimate such costs.

Although experiments with the developed RSSICollector have been run in a couple of different indoor locations there is extensive work left for the future. That work could include further investigation regarding individual differences between Telos motes as well as more focus on identifying the reason behind the random deviations in RSSI readings. Discovering and discarding outliers in runtime and investigate if the link quality, lqi, also measured by RSSICollector, could be taken into consideration to improve distance estimation.

A suggestion on improvements to the RSSICollector application is to increase the transmitted power level dynamically based on previous measurements to try to discover nodes on the network. A more intelligent version of the application could for example start sending at the lowest possible level and define an area of possible ranges based on that signal strength. If no new nodes are found the transmitted power can be increased taking in to account that nothing was found using the a lower power level.

6.2 Localization System with Telos Motes

Three separate NesC applications have been developed for Telos motes to successfully perform localization estimations with RSSI and trilateration with the existing Java application at UPC. The source code is attached in appendix B.

The current system serves as a good platform for future development and
evaluation of localization algorithms. However, as more theoretic performance measurements are left for future work, conclusions regarding the accuracy of this Telos based localization system are limited at this time. As already concluded in chapter 4, using RSSI as a ranging technique with Telos is very fragile. In a scenario where one beacon antenna was turned slightly away from the client and another beacon did not have line-of-sight the estimated position of the client mote deviates greatly from the true position. This impact is very clear looking at figure 5.6.

A suggested approach for future work is to focus more on beacons returning low RSSI values, possibly by assigning different weights depending on the signal strength returned but more measurements need to confirm that low values are more reliable than higher ones.

To mitigate problems with antenna orientation and obstacles blocking line-of-sight an ad-hoc correction mechanism could be proposed. It could for example make measurements between beacons in the sensor network and calibrate the results against the known distances. This can be considered a sort of detection of walls and other large objects to calibrate the path-loss models and adjust the measuring to a specific set-up site. Figure 6.1 displays a scenario where a wall is interfering in the measurements.

![Figure 6.1. Wall detection scenario.](image)

If measurements are made only from the mobile node the estimated position for M will get an erroneous offset to the left from the beacon on the left side of the wall. By measuring the RSS between all beacons as well, this offset can be estimated.
Bibliography


Appendix A

Source Code

A.1 RSSICollector

```c
#include "RssiCollector.h"

/**
 * This configuration describes the RssiCollector application,
 * a TinyOS app that is used to collect Rssi values between two Telos
 * motes.
 * One mote is receiver and one is sender. The mote attached to a PC will
 * send all data over the UART.
 */
configuration RssiCollector { }
implementation
{
    components Main, RssiM,
        , TimerC
        , LedsC
        , GenericComm as RadioComm
        , UARTComm as Comm
        , CC2420RadioC
        , MSP430InterruptC;
    Main.StdControl -> RssiM;
    Main.StdControl -> TimerC;
    RssiCollectorM.User -> MSP430InterruptC.Port27;
    RssiCollectorM.TimerLoop -> TimerC.Timer[unique("Timer")];
    RssiCollectorM.TimerTimeout -> TimerC.Timer[unique("Timer")];
    RssiCollectorM.Leds -> LedsC;
```
** This module implements RssiCollectorM 
*/

module RssiCollectorM 
{
    provides interface StdControl;
    uses 
    {
        interface Timer as TimerLoop;
        interface Timer as TimerTimeOut;
        interface Leds;
        interface StdControl as CommControl;
        interface SendMsg as DataMsg;
        interface ReceiveMsg as ResetCounterMsg;
        interface StdControl as RadioControl;
        interface SendMsg as SendChirpMsg;
        interface ReceiveMsg as ReceiveChirpMsg;
        interface SendMsg as SendRequestMsg;
        interface ReceiveMsg as ReceiveRequestMsg;
        interface MSP430Interrupt as User;
        interface CC2420Control as CCControl; // For changing signal power
        //interface MacControl;
    }
} 

implementation 
{

    /*************** VARIABLES ************/
    uint16_t cnt;
    uint8_t radioLevel;
    uint8_t actualRL;

    RssiCollectorM.CCControl -> CC2420RadioC.CC2420Control;
    //RssiM.MacControl -> CC2420RadioC.MacControl;
    RssiCollectorM.CommControl -> Comm;

    // UART
    RssiCollectorM.ResetCounterMsg -> Comm.ReceiveMsg[AM_CHIRPMSG];
    RssiCollectorM.DataMsg -> Comm.SendMsg[AM_CHIRPMSG];

    // RADIO
    RssiCollectorM.RadioControl -> RadioComm;
    RssiCollectorM.SendChirpMsg -> RadioComm.SendMsg[AM_CHIRPMSG];
    RssiCollectorM.SendRequestMsg -> RadioComm.SendMsg[AM_REQUESTMSG];
    RssiCollectorM.ReceiveChirpMsg -> RadioComm.ReceiveMsg[AM_CHIRPMSG];
    RssiCollectorM.ReceiveRequestMsg -> RadioComm.ReceiveMsg[AM_REQUESTMSG];

}
uint8_t sendRetries;
uint8_t state;

uint8_t noMoreListening;
uint16_t readingNumber;
TOS_Msg msg[2];
uint8_t currentMsg;

ChirpMsg *chirpmsg;
RequestMsg *requestmsg;

TOS_Msg tosmsg;

void action();
void txdo();
void dataTask(TOS_MsgPtr);
void sendRequest();

/**
 * Used to initialize this component.
 */
command result_t StdControl.init() {
call Leds.init();
call Leds.yellowOff(); call Leds.redOff(); call Leds.greenOff();
call CommControl.init();
call RadioControl.init();

atomic {
    //call MacControl.enableAck();
    currentMsg = 0;
    readingNumber = 0;
    cnt = MSG_COUNT+1;
    radioLevel=31;
    actualRL=31;
    noMoreListening = 0;
}

//state = START_CHIRPING;
state = RECEIVING;

    return SUCCESS;
}

/**
 * Starts the SensorControl and CommControl components.
 * @return Always returns SUCCESS.
 */
command result_t StdControl.start() {
atomic {
    call User.disable();
    call User.clear();
    call User.edge(FALSE);
    call User.enable();
}

if (state == START_CHIRPING) {
    call TimerTimeOut.start(TIMER_ONE_SHOT, 5000);
} else {
    call TimerLoop.start(TIMER_REPEAT, INTER_MESSAGE_TIME);
}
call CommControl.start();
call RadioControl.start();
return SUCCESS;
}

/**
 * Stops SensorControl and CommControl components.
 * @return Always returns SUCCESS.
 */
command result_t StdControl.stop() {

    atomic {
        call User.disable();
        call User.clear();
    }

    call TimerLoop.stop();
call CommControl.stop();
call RadioControl.stop();
return SUCCESS;
}

/**
 * Previous packet sent.
 * @return Always returns SUCCESS.
 */
event result_t DataMsg.sendDone(TOS_MsgPtr sent, result_t success) {
    return SUCCESS;
}

/**
 * Clock tick event.
 * @return The result of calling ADC.getData().
 */
event result_t TimerLoop.fired() {
    if (state == START_CHIRPING) {
        action();
    } else if (state == IDLE) {

event result_t TimerTimeOut.fired() {
  call TimerLoop.start(TIMER_REPEAT, INTER_MESSAGE_TIME);
  return SUCCESS;
}

/**
 * Reset message counter AM is received.
 * @return The free TOS_MsgPtr.
 */
event TOS_MsgPtr ResetCounterMsg.receive(TOS_MsgPtr m) {
  atomic {
    readingNumber = 0;
  }
  return m;
}

/**
 * Previous packet has been sent.
 * @return Always returns SUCCESS.
 */
event result_t SendChirpMsg.sendDone(TOS_MsgPtr sent, result_t success) {
  atomic {
    if (success == SUCCESS){
      call Leds.redOff();
    }
    else {
      call Leds.redOn();
    }
  }
  return SUCCESS;
}

// Previous packet has been sent.
event result_t SendRequestMsg.sendDone(TOS_MsgPtr sent, result_t success) {
  return SUCCESS;
}

// Receive Chirp
event TOS_MsgPtr ReceiveChirpMsg.receive(TOS_MsgPtr m) {
  atomic {

if (state == RECEIVING) {
call Leds.greenToggle();
dataTask(m);
}
return m;

// Receive Request
event TOS_MsgPtr ReceiveRequestMsg.receive(TOS_MsgPtr m) {
   atomic {
      if (state == RECEIVING) {
         call Leds.greenToggle();
         atomic noMoreListening = 1;
         atomic state = START_CHIRPING;
      }
   }
return m;
}

// User button event
async event void User.fired() {
call TimerLoop.stop();
atomic state = START_CHIRPING;
call User.clear();
call TimerTimeOut.start(TIMER_ONE_SHOT, 5000);
}

void dataTask(TOS_MsgPtr m) {
   struct ChirpMsg *pack;
call Leds.greenToggle();
chirpmsg = (ChirpMsg *) m->data;
   atomic {
      pack = (struct ChirpMsg *)tosmsg.data;
      pack -> rssi = m -> strength;
      pack -> lqi = m -> lqi;
      pack -> txid = chirpmsg -> txid;
      pack -> cnt = chirpmsg -> cnt;
      pack -> rl = chirpmsg -> rl;
      pack -> rxid = TOS_LOCAL_ADDRESS;
   }
   if (call DataMsg.send(TOS_UART_ADDR, sizeof(struct ChirpMsg), &tosmsg) == SUCCESS) {
call Leds.greenToggle();
   }
}

void sendRequest() {
   struct RequestMsg *pack;
call Leds.greenToggle();
call Leds.yellowOn();
atomic {
    pack = (struct RequestMsg *)tosmsg.data;
    pack -> txid = TOS_LOCAL_ADDRESS;
    pack -> cnt = MSG_COUNT;
}
call CCControl.SetRFPower(radioLevel);

if (call SendRequestMsg.send(TOS_BCAST_ADDR, sizeof(struct RequestMsg), &tosmsg) == SUCCESS) {
    call Leds.greenToggle();
}
}

/**********************************
* Function action() counts down packet_cnt and controls radio level before calling txdo()
*/
void action() {
    cnt--;
    if (cnt==0x00) {
        radioLevel = radioLevel - 4;
        call Leds.yellowOn();
        cnt = MSG_COUNT;
        if (radioLevel < 1 || radioLevel > 31) {
            atomic {
                radioLevel = RADIO_MAX;
                state = RECEIVING;
                if (noMoreListening == 0) {
                    call Leds.yellowOn();
                    sendRequest();
                }
                call Leds.greenOn();
                call Leds.redOff();
            }
        }
        call Leds.greenOn();
        return;
    }
    sendRetries = RETRIES_COUNT;
    txdo();
}

// Function txdo() sets radio level and sends chirp message
void txdo() {
    atomic {
        if (radioLevel != actualRL) {
            if (call CCControl.SetRFPower(radioLevel) == SUCCESS) {
                actualRL = radioLevel;
            }
        }
    }
}
call Leds.greenOn();
} else {
    return;
}

chirpmsg = (ChirpMsg *)tosmsg.data;
atomic chirpmsg->txid = TOS_LOCAL_ADDRESS;
atomic chirpmsg->rl = radioLevel;
atomic chirpmsg->cnt = cnt;
call Leds.greenToggle();
call SendChirpMsg.send(TOS_BCAST_ADDR, sizeof(struct ChirpMsg), &tosmsg);
}

#ifndef H_TOSBase_h
#define H_TOSBase_h

enum {
    START_CHIRPING = 1,
    IDLE = 2,
    RECEIVING = 3,
    WAITING = 4,
    INTER_MESSAGE_TIME = 256, //128
    RELAY_ID = 2,
    MSG_COUNT = 100,
    RETRIES_COUNT = 3,
    START_DELAY = 8000,
    RADIO_MAX = 31
};

enum {
    AM_CHIRPMGS = 26,
    AM_REQUESTMSG = 4
};

typedef struct ChirpMsg {
    uint8_t txid;
    uint8_t rxid;
    uint8_t rl;
    uint8_t cnt;
    uint8_t rssi;
    uint8_t lqi;
} ChirpMsg;

typedef struct RequestMsg {
    uint8_t txid;
} RequestMsg;
uint8_t cnt;
} RequestMsg;

#endif
Appendix B

Localization System Source Code

B.1 Sink Mote Application

B.1.1 SinkMote.nc

includes AODV;
includes TelosTrackingCmdMsg;
includes LocMsg;
includes ChirpMsg;

classification SinkMote {
}

implementation {
    components Main, LedsC, SinkMoteM, AODV, TimerC, CC2420RadioC, FramerM;
    Main.StdControl -> SinkMoteM;
    SinkMoteM.Leds -> LedsC;
    SinkMoteM.AODV -> AODV.Control;
    SinkMoteM.Receive -> AODV.Receive[APP_ID_WSN];
    SinkMoteM.Timer -> TimerC.Timer[unique("Timer")];
    SinkMoteM.XLocSend -> AODV.SendMsg[AM_LOCMSG];
    SinkMoteM.ReceiveTelosCmdMsg ->
        SendMsg.ReceiveMsg[AM_TELOSTRACKINGCMDMSG];
    SinkMoteM.CC2420Control -> CC2420RadioC;
}
B.1.2 SinkMoteM.nc

```c
includes TelosTrackingCmdMsg;
includes LocMsg;
includes ChirpMsg;

module SinkMoteM {
  provides {
    interface StdControl as Control;
  }
  uses{
    interface Leds;
    interface Receive;
    interface SendMsg;
    interface StdControl as AODV;
    interface SendMsg as XLocSend;
    interface ReceiveMsg as ReceiveTelosCmdMsg;
    interface ReceiveMsg as ReceiveChirpMsg;
    interface CC2420Control;
    interface Timer;
  }
}

implementation {
  TOS_Msg data;
  TOS_Msg xdata;

  LocMsg *xmessage = (LocMsg *)xdata.data;
  TelosTrackingCmdMsg *trackptr;

  TOS_Msg msgbuf;
  TOS_MsgPtr msgptr;

  mensaje_a_base *ptrmsg;
  LocMsg *xptrmsg;
  uint16_t len;
  uint16_t power;
  int16_t matriz [10] [2];
  uint16_t i;
  uint16_t e;

  command result_t Control.init() {
    call AODV.init();
    call Leds.init();
    msgptr = &msgbuf;
    power = POWER;
    matriz [0] [0] = 0; matriz [0] [1] = 0;
    matriz [1] [0] = 1; matriz [1] [1] = 0;
    matriz [2] [0] = 2; matriz [2] [1] = 0;
  }
```
matriz [3] [0] = 3; matriz [3] [1] = 0;
matriz [4] [0] = 4; matriz [4] [1] = 0;
matriz [5] [0] = 5; matriz [5] [1] = 0;
matriz [6] [0] = 6; matriz [6] [1] = 0;
matriz [7] [0] = 7; matriz [7] [1] = 0;
matriz [8] [0] = 8; matriz [8] [1] = 0;
matriz [9] [0] = 9; matriz [9] [1] = 0;

return SUCCESS;
}

custom control result_t Control.start() {
call AODV.start();
call Leds.yellowOff();
call CC2420Control.SetRFPower(power);
return SUCCESS;
}

custom control result_t Control.stop() {
call AODV.stop();
call Timer.stop();
return SUCCESS;
}

// Task to send measurement data to java application
void task sendPC(){
call XLocSend.send(TOS_UART_ADDR, sizeof(LocMsg), &xdata);
}

event custom control result_t Timer.fired(){
call Leds.yellowOff();
return SUCCESS;
}

// Receive LocMsg message. Check message type and forward to PC.
event TOS_MsgPtr Receive.receive(TOS_MsgPtr msg, void *payload, uint16_t longitud) {
xptrmsg = (LocMsg *)msg->data

xmessage -> type = xptrmsg -> type;
xmessage -> reservat = xptrmsg -> reservat;
xmessage -> idBaliza = xptrmsg -> idBaliza;
xmessage -> idCliente = xptrmsg -> idCliente;
xmessage -> mean = xptrmsg -> mean;
xmessage -> variance = xptrmsg -> variance;
xmessage -> emittedPower = xptrmsg -> emittedPower;
for (i=0; i<10; i++){
xmessage -> chirpVals[i] = xptrmsg -> chirpVals[i];
}
post sendPC()
    call Leds.yellowOn();
    call Timer.start(TIMER_ONE_SHOT,1000);

    return NULL;
}

// Receive event for TelosTrackingCmdMsg from PC. Forwarding to client
event TOS_MsgPtr ReceiveTelosCmdMsg.receive(TOS_MsgPtr pmsg){
    if (len == sizeof(TelosTrackingCmdMsg)) {
        trackptr = (TelosTrackingCmdMsg *)pmsg -> data; // Get the data

        // Copy the destination from the message to the address in the
        // AM header
        client_node_address = trackptr->dest;
        call SendMsg.send(client_node_address,
            sizeof(TelosTrackingCmdMsg), &trackptr);
    } 
    return pmsg;
}

event result_t XLocSend.sendDone(TOS_MsgPtr msg, result_t success){
    return SUCCESS;
}

event result_t SendMsg.sendDone(TOS_MsgPtr sentmsg, result_t success){
    return SUCCESS;
}
Main.StdControl -> ClientMoteM;
Main.StdControl -> TimerC;
Main.StdControl -> PowerButtonC;
ClientMoteM.SubControl -> Comm;
ClientMoteM.Leds -> LedsC;
ClientMoteM.ChirpTimer -> TimerC.Timer[unique("Timer")];
ClientMoteM.SendMsg -> Comm.SendMsg[AM_ATNMSG];
ClientMoteM.CC2420Control -> CC2420RadioC;
ClientMoteM.ReceiveMsg -> Comm.ReceiveMsg[AM_TELOSTRACKINGCMDMSG];

B.2.2 ClientMoteM.nc

includes ChirpMsg;
includes TelosTrackingCmdMsg;

module ClientMoteM{
    provides{
        interface StdControl;
    }
    uses{
        interface StdControl as SubControl;
        interface SendMsg;
        interface Timer as ChirpTimer;
        interface Leds;
        interface CC2420Control;
        interface ReceiveMsg;
    }
}

implementation{

    TOS_Msg data;
    uint16_t power;
    ChirpMsg *chirpMsg = (ChirpMsg *)data.data;

    command result_t StdControl.init(){
        call SubControl.init();
        call Leds.init();
        power = POWER;
        return SUCCESS;
    }

    command result_t StdControl.start(){
        call SubControl.start();
    }
call CC2420Control.SetRFPower(power);
call Leds.greenOff();
call Leds.redOff();
call Leds.yellowOff();
return SUCCESS;
}

command result_t StdControl.stop(){
call SubControl.stop();
call ChirpTimer.stop();
return SUCCESS;
}

event result_t ChirpTimer.fired(){
aux6 = aux6 + 1;
call SendMsg.send(TOS_BCAST_ADDR, sizeof(ChirpMsg), &data);

if(aux6 == MAX_NUM_CHIRPS){
aux6 = 0;
call ChirpTimer.stop();
call Leds.redOff();
}
return SUCCESS;
}

event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success){
return SUCCESS;
}

event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m){

msgptr = (TelosTrackingCmdMsg *)m->data;

// A localization petition is received.
if(msgptr->orden == 1){

chirpMsg->relleno = 0;
chirpMsg->flag = 1; // Flag == 1 means chirp message
chirpMsg->direccion_punto_acceso = 0;
chirpMsg->identificador_cliente = TOS_LOCAL_ADDRESS;
chirpMsg->identificador_baliza_nueva = 0;
chirpMsg->potencia_emitida = power;
chirpMsg->potencia_recibida = 0;

call Leds.redOn();
call ChirpTimer.start(TIMER_REPEAT,msgptr->number_of_chirps);
}
return m;
}
B.3 Beacon Mote Application

B.3.1 BeaconMote.nc

```verbatim
includes AODV;
includes ChirpMsg;
includes LocMsg;
includes NetProg;

configuration BeaconMote {
}

implementation {

components Main, LedsC, BeaconMoteM, AODV, CC2420RadioC, TimerC,
     PowerButtonC, RandomLFSR;

Main.StdControl    -> BeaconMoteM;
Main.StdControl    -> PowerButtonC;
BeaconMoteM.ReceiveMsg    -> Comm.ReceiveMsg[AM_CHIRPMSG];
BeaconMoteM.SendMsg     -> AODV.SendMsg[AM_LOCMSG];
BeaconMoteM.Leds       -> LedsC;
BeaconMoteM.Random     -> RandomLFSR;

// Timers
BeaconMoteM.TimerRetrySend    -> TimerC.Timer[unique("Timer")];
BeaconMoteM.TimerRandomSend    -> TimerC.Timer[unique("Timer")];
BeaconMoteM.TimeOutTimer      -> TimerC.Timer[unique("Timer")];
BeaconMoteM.TimerSendChirp    -> TimerC.Timer[unique("Timer")]; //For chirping speed
BeaconMoteM.AODV             -> AODV.Control;
BeaconMoteM.CC2420Control    -> CC2420RadioC;
}
```

B.3.2 BeaconMoteM.nc

```verbatim
includes ChirpMsg;
includes LocMsg;

module BeaconMoteM{
    provides{
        interface StdControl as Control;
    }
```
uses{
    interface StdControl as AODV;
    interface StdControl as CommControl;
    interface ReceiveMsg;
    interface Leds;
    interface SendMHopMsg;
    interface CC2420Control;
    interface SendMsg;
    interface Timer as TimerRetrySend;
    interface Timer as TimeOutTimer;
    interface Timer as TimerRandomSend;
    interface Timer as TimerSendChirp;
    interface Receive;
    interface Random;
}
}

implementation{
    enum
    {
        CHIRP_MSG = 1,
        TRACKING_MSG = 0,
        CLIENT_SEARCH_FREQ = 3000, // in ms
        MAX_NUM_CHIRPS = 10, // This is how many chirps the beacons expect to receive.
        MAX_NUM_RAND = 500, // The upper limit for the random waiting to send locData
        SINK_NODE_ADDR = 0x09
    };

    bool locActivated;
    bool locTimeOut;
    bool locDataSent;

    int randVal;
    int i;
    int chirpCount;
    uint8_t rssi;

    void computeAndSend();
    void bubbelSort(int a[], int length);

    /*
    * Math variables
    */
    float mean;
    float total;
    uint8_t_t rssiArray[MAX_NUM_CHIRPS];
    int varArray[MAX_NUM_CHIRPS];
    float vsum;
float variance;
float p;
float minimum;
float maximum;

int N, k, j, AUX;
uint16_t power;

uint16_t len;
uint16_t xlen;
uint8_t pot;
uint16_t numero;
uint16_t numvar;
uint16_t linea;
uint8_t messageType;
bool radioBusy;
uint8_t sendCounter;

TOS_Msg data;
TOS_Msg msgbuf;
TOS_Msg xlocbuf;
TOS_MsgPtr msgptr;
TOS_MsgPtr xlocptr;

ChirpMsg *mssg;

TOS_Msg chirpdata;
ChirpMsg *chirpMsgPtr = (ChirpMsg *)chirpdata.data;
LocMsg *xlocmsgptr;

command result_t Control.init() {
call Leds.init();
call AODV.init();

msgptr = &msgbuf;
msgptr = &msgbuf;

chirpCount = 0;
atomic locActivated = FALSE;
locTimeOut = FALSE;
locDataSent = FALSE;

power = POWER;
numvar = 0;
numero = NUM;
linea = 0;
messageType = 0;
radioBusy = FALSE;

return SUCCESS;
}
command result_t Control.start() {
    call AODV.start();
    call CC2420Control.SetRFPower(power);
    call ClientSearchTimer.start(TIMER_REPEAT, CLIENT_SEARCH_FREQ);
    return SUCCESS;
}

command result_t Control.stop() {
    call AODV.stop();
    call ClientSearchTimer.stop();
    call TimerRedOff.stop();
    call TimerretrySend.stop();
    call TimerYellowOff.stop();
    call TimerBatCheck.stop();
    call TimeOutTimer.stop();
    call TimerSendChirp.stop();
    return SUCCESS;
}

// Send buffert over AODV to sink node with LocMsg
void task sendLocDataAODV()
{
    if (!call
        SendMHopMsg.sendTTL(SINK_NODE_ADDR, sizeof(LocMsg), xlocptr, 0)) {
        atomic radioBusy = FALSE;
    }
}

/*
* The RandomSend timer is started after chirps have been received
* It tries to send the buffert. If it fails the retry timer is fired.
*/
event result_t TimerRandomSend.fired(){
    if (!post sendLocDataAODV()) {
        call TimerRetrySend.start(TIMER_REPEAT, 300);
    }
    return SUCCESS;
}

/*
* Retry send until locDataSent = TRUE
*/
event result_t TimerRetrySend.fired(){
    if (locDataSent == FALSE) {
        post sendLocDataAODV();
    }
    else {
        call TimerRetrySend.stop();
    }
    return SUCCESS;
event result_t TimeOutTimer.fired(){
    if (chirpCount < MAX_NUM_CHIRPS && chirpCount != 0 && locActivated == TRUE) {
        atomic computeAndSend();
    }
    return SUCCESS;
}

void task sendToSink(){
    atomic{
        if(radioBusy == FALSE){
            radioBusy = TRUE;
            if (!post mensaje_aodv()) radioBusy = FALSE;
        }
    }
}

// Chirp message from client to beacon
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m){
    mssg = (ChirpMsg *)m->data;
    numvar = mssg->identificador_cliente;
    linea = mssg->potencia_recibida;
    messageType = mssg->flag;
    pot = m->strength;

    // Receiving rssi messages
    if(messageType == CHIRP_MSG){
        locActivated = TRUE;

        if(chirpCount == 0){
            call TimeOutTimer.start(TIMER_ONE_SHOT, 4000);
        }

        rssi = pot - 45.0;
        rssiArray[chirpCount] = pot - 45.0;
        varArray[chirpCount] = (int)rssiArray[chirpCount];
        total = total + (float)rssi;
        chirpCount++;

        if (chirpCount == MAX_NUM_CHIRPS && locActivated == TRUE) {
            atomic computeAndSend();
        }
    }
B.3 Beacon Mote Application

```c
return m;
}

void computeAndSend() {
    mean = total / (float) chirpCount;
    maximum = 0;
    minimum = 0;

    for (i=0; i<MAX_NUM_CHIRPS; i++) {
        p = (float)rssiArray[i];
        if (abs(p) > abs(minimum) && abs(p) != 0) {
            minimum = p;
        }
        if (abs(p) < abs(maximum) && abs(p) != 0) {
            maximum = p;
        }
    }

    // Variance calculation
    for (i=0; i<MAX_NUM_CHIRPS; i++) {
        p = (float)rssiArray[i];
        if (p != 0) {
            vsum = vsum + ((p - mean)*(p - mean));
        }
    }
    variance = vsum / (float)chirpCount;
    bubbelSort(varArray, MAX_NUM_CHIRPS);

    xlocmsgptr -> type = 1;
    xlocmsgptr -> reservat = 0;
    xlocmsgptr -> idBaliza = (uint8_t)numero;
    xlocmsgptr -> idCliente = (uint8_t)mssg->identificador_cliente;
    xlocmsgptr -> mean = (uint16_t)mean;
    xlocmsgptr -> variance = (uint16_t)variance;
    xlocmsgptr -> emittedPower = mssg->potencia_emitida;

    for (i=0; i<MAX_NUM_CHIRPS; i++){
        xlocmsgptr -> chirpVals[i] = rssiArray[i];
        rssiArray[i] = 0;
    }

    mean = floor(0.1);
    mean = 0;
    chirpCount = 0;
    rssi = 0;
    vsum = 0;
    total = 0;
```
if (locActivated == TRUE) {
    locActivated = FALSE;
}
randVal = (call Random.rand())%MAX_NUM_RAND;
call TimerRandomSend.start(TIMER_ONE_SHOT, randVal);
call ClientSearchTimer.start(TIMER_REPEAT,CLIENT_SEARCH_FREQ);
}

// AODV Send done event. Check localization activated and if data sent.
event result_t SendMHopMsg.sendDone(TOS_MsgPtr sent, result_t success){
    if (locActivated == TRUE) {
        atomic locActivated = FALSE;
    }
    if (success == SUCCESS) {
        locDataSent == TRUE;
    }
    radioBusy = FALSE;
    return SUCCESS;
}

// Generic sendDone event.
event result_t SendMsg.sendDone(TOS_MsgPtr mg, result_t success){
    radioBusy = FALSE;
    return SUCCESS;
}

// Bubblesort an array in ascending order
void bubbelSort(int a[], int length){
    for(i=0; i<length; i++){
        for(j=length-1; j>=i+1; j--){
            if( a[j-1] > a[j] ){
                AUX = a[j-1];
                a[j-1] = a[j];
                a[j] = AUX;
            }
        }
    }
}

B.4 Message Header Files

B.4.1 ChirpMsg.h
typedef struct ChirpMsg {
    uint16_t relleno;
    uint16_t flag;
    uint16_t direccion_punto_acceso;
    uint16_t client_id;
    uint16_t identificador_baliza_nueva;
    uint8_t potencia_emitida;
    uint8_t potencia_recibida;
} ChirpMsg;

enum {
    AM_CHIRPMSG = 4
};

B.4.2 TelosTrackingCmdMsg.h

typedef struct TelosTrackingCmdMsg {
    int8_t seqno;
    int8_t address;
    uint16_t source;
    uint8_t number_of_chirps;
} TelosTrackingCmdMsg;

enum {
    AM_TELOSTRACKINGCMDMSG = 8
    LOCATE_CLIENT = 1
};

B.4.3 LocMsg.h

typedef struct LocMsg {
    uint8_t type;
    uint8_t reservat;
    uint8_t idBaliza; //id baliza
    uint8_t idCliente; //id cliente
    uint16_t mean;
    uint16_t variance;
    uint16_t emittedPower;
    uint8_t chirpVals[10];
} LocMsg;

enum {
AM_LOCMRG = 7
};