A PARTICLE FILTER BASED FRAMEWORK FOR INDOOR WIRELESS LOCALIZATION USING CUSTOM IEEE 802.15.4 NODES

By

Vijay Vasant Dixit

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

Of the Requirements for the Degree of

MASTER OF SCIENCE IN COMPUTER ENGINEERING

UNIVERSITY OF TEXAS AT ARLINGTON

December 2008

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family and friends for their support and encouragement that made all this possible.

I would like to offer special thanks to my supervising professor Dr. Gergely Záruba for his consistent help, support, and funding throughout this endeavor. His patience, experience, and knowledge have been invaluable throughout my research, and I am truly grateful for this. I would also like to thank Dr. Manfred Huber, for explaining me the concepts behind a Particle Filter and Mr. David Levine for his encouragement during the course of the project.

Furthermore, I would like to thank Mr. Bito Irie for his continuous support and encouragement during the course of my Masters at UTA, without which this endeavor may have not been possible. I would also like to thank Zhou Lanjiang, for his help with "Tag Orientation/Step Detection", Umeshkumar Keswani for his help with the "Distance/Orientation Lookup Table", "Collection of RSSI readings" and Fawaz Bokhari, for his help with the "Tag Routing" algorithm.

November 25, 2008

ABSTRACT

A PARTICLE FILTER BASED FRAMEWORK FOR INDOOR WIRELESS LOCALIZATION USING CUSTOM IEEE 802.15.4 NODES

Vijay Vasant Dixit, M.S.

The University of Texas at Arlington, 2008

Supervising Professor: Gergely Zaruba

Locating people close to real-time and with acceptable precision has always been an important part of any organization or industry, especially in law enforcement, manufacturing, healthcare, and logistics. Technologies that have the ability to locate objects or people are called Real Time Location Systems (RTLS). They typically use small low-power transmitters called tags attached to assets (or worn by people) as well as sets of readers that map the location of these tags. Systems that map the longitude and attitude of an object are geo-location systems and generally use the Global Positioning System (GPS) for location mapping. GPS could be used as the location determination portion of an RTLS system (relaying that information would have to rely on another system); unfortunately, GPS signals do not penetrate buildings well and thus GPS will in general not

work inside buildings and in dense urban areas. Thus, there is a need for RTLS systems that work in GPS-denied environments.

Several technologies have been proposed to create Real Time Location Systems. Some use dedicated tags and readers while others use existing WLAN networks and add RTLS ability to those networks. We propose a probabilistic approach to localization, based upon Received Signal Strength (RSSI) and inertial information coming from tags (e.g., accelerometer and rotational sensor readings). Global localization is a flavor of localization in which the device is unaware of its initial position and has to determine the same from scratch.

The first step to localize tags in this work involves building a wireless measurement model of the tag with respect to some anchor nodes (access points). The model is built by measuring the RSSI readings of the mobile node relative to the access point at various distances and orientations (rotation away from the access point). These readings form a sample set for sequential Monte-Carlo sampling. Next, a posterior probability distribution for the location of the wireless device is computed over the entire area using Monte-Carlo sampling based Bayesian filtering, also known as *particle filters*. Location estimates may then be determined from this distribution using the maximum density point or other parameters depending on the estimate needed.

We discuss theory and research leading to the proposed method and describe the experimental hardware/firmware/software system built for the purposes of this work and provide results of real-life experiments.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1
INTRODUCTION

The rapid proliferation of wireless networks over the past few years has given rise to a large number of mobile applications. There is an inherent nature in wireless networks that enables mobility of devices. These features create a compelling reason to build applications that solve the problem of location awareness among these mobile devices. On the other hand considering the cost and effort involved, it is questionable to utilize the services of existing wireless technologies like 3G, Wi-Fi etc. solely for the purpose of providing location based services. With regard to popular technologies meant for localization like GPS (Global Positioning System), the problem is that it is highly unsuitable for supporting indoor based localization for e.g., within a shopping mall. Furthermore, most of the GPS devices behave as *receivers*, that is, rarely is it that a GPS device would enable *tracking* by relaying knowledge of its location to a third party localization application.

There are many scenarios (e.g., patient tracking in hospitals) where real-time localization using simple inexpensive devices is beneficial. There is a need for technologies with which location of people or equipment wearing simple wireless devices can be located in real time inside buildings. The availability of inexpensive low power microcontrollers, System on Chip, and radio on chip technologies forms for an even more compelling reason to develop cost-effective and architecturally

flexible wireless devices (nodes) which can be specifically used for the purpose of *tracking*.

In order to localize nodes there needs to be a way to calculate distances and/or distance travelled between/for nodes. Such information could be derived from different types of sensory data, including received signal strength indication (RSSI) readings, rotational sensor (gyroscope) readings, accelerometer readings, and time of flight measurements. As digital radio technologies became inexpensive, their signal propagation properties are good candidates for such distance measurements. Most wireless digital radios are able to measure RSSI based readings as a part of their standard operation. In general, the more sensors provide with distance indicating data, the better a location estimate can be. Thus, there is an incentive to use additional means for distance travelled measurement, i.e., rotational and acceleration sensors. Using such sensors has proven to provide good results in inertial navigation systems [1] used for several decades in the aviation industry, or in dead-reckoning systems [2]. But, modifications need to be made in order to derive the remaining types. This thesis outlines our research towards a system of custom radio nodes that use a combination of sensor readings as inputs to a *Particle Filter,* to help determine the location of a device and relay such information to a central processing station using a mesh network of nodes [3]. More specifically, the aim of the thesis is to present a proof of concept, system setup that shows that reasonable location estimates may be achieved by using a combination of RSSI readings from multiple

access points, rotational and acceleration sensor readings at a mobile node, relaying such raw location indicating information for processing to a central node.

1.1. Motivation

The reasons behind research and development of localization techniques that do not rely on the GPS can be broadly categorized into the following:

- Issues related to line of sight (LoS) needs.
- Improvement in wireless security.
- Pervasive computing – location awareness, asset tracking.

We will discuss in detail each of the above, in the following sub-sections.

1.1.1. Issues Related to Line of Sight (LoS) Needs

Many popular location estimation techniques like RADAR (Radio Detection and Ranging) and GPS (Global Positioning Systems) have been in service for several years. Both [4] these techniques rely on measuring the time taken by radio waves to propagate, and a fairly precise location estimate (relative to the location of the tracking device) may easily be determined by calculating differences in propagation times. However, they require that the targets to be detected (e.g., flying objects, GPS receivers) are more or less within the receivers' line of sight. Consequently, the inability of technologies like GPS to function indoors prevents, or at least restricts, the use of such systems in populated areas where such a line of sight may not be available. This limitation severely reduces the feasibility of successful deployment in

an environment where mobile devices (targets) may not remain within the receivers' line of sight.

RSSI readings from a radio transceiver device, on the other hand, are available from the device without the need for a line of sight. While RF waves are capable of penetrating many physical boundaries or structures, these do affect the RSSI reading, albeit in a somewhat predictable fashion. This predictability it possible to map RSSI readings from a fixed access point to a region, and thus enables localization using RSSI readings. In addition, these techniques can be easily combined with other mobility based sensory data like measurement of local displacement and orientation, to achieve the best possible estimates. RSSI is not the only technology that can be used to infer distance from the properties of RF waves. Recently techniques using time of flight of UWB [5] data and Fourier descriptors of signals [6] have been shown to be feasible.

1.1.2. Improvement in Wireless Security

Location estimation would play an important role in the information security area, with regard to physically tracking unauthorized access to private wireless networks [7]. It would be easy to locate an unauthorized user relative to an accesspoint nearest to him.

If we consider a wired network, physical authentication is inherently provided by the physical access to the network socket. On the other hand in case of a wireless sensor network, signal propagation is not limited by a fixed boundary, and

unauthorized access from outside the security perimeter is possible. Wireless localization is one solution to such a problem which is able to identify intruders based on their location, and thus successfully defend a "parking lot" attack. During the localization phase, the RSSI of the mobile nodes may be measured by multiple accesspoints that are positioned to provide coverage of the area. Considering a server-based approach to support the above network for processing data, it would be possible to enable security application.

1.1.3. Pervasive Computing

Another strong motivating factor is the application area of the proposed technique. Knowledge of the physical location data from mobile nodes would allow many software packages to provide a whole range of features that support applications ranging from location awareness to asset tracking.

Location aware applications would refer to a situation where, for e.g., a person would be able to walk across the house with TV screens in each room switching to the channel he/she was watching, getting directions on a PDA to a store within a large mall.

Tracking would be another important application that would take advantage of the inherent nature of such a system to relaying location based information to third parties. This feature would enable a number of applications like keep track of patients within a hospital, tracking trapped miners in a mine etc.

CHAPTER 2
APPROACHES TO WIRELESS LOCALIZATION

Real Time Locating Systems (RTLS, incorrectly named Real Time Location Systems), according to international standards, are used to track and identify the location of objects in real time using simple, inexpensive nodes (badges/tags) attached to or embedded in objects and devices that receive the wireless signals from these tags to determine their locations. RTLS typically refers to systems that provide (automatic) collection of location information without user intervention.

Several methods for performing wireless localization are possible, depending on the type of data derived from each of them. Some of them are as follows:

- Time of Arrival (ToA)
- Time of Flight (ToF)
- Angle of Arrival (AoA)
- Received Signal Strength Indicator(RSSI)

We will discuss each of the above in detail.

2.1. Time of Arrival (ToA)

The Time of Arrival, or ToA, is a method based on the [8] measurement of the propagation delay of the radio signal between tag and one or more readers. The delay $t_i - t_0$, is the time lag of the departure of a signal from a source station to a

destination station; in other words, it is the amount of time required to for a signal to travel from the transmitter to the receiver, as shown in Figure 2.1.

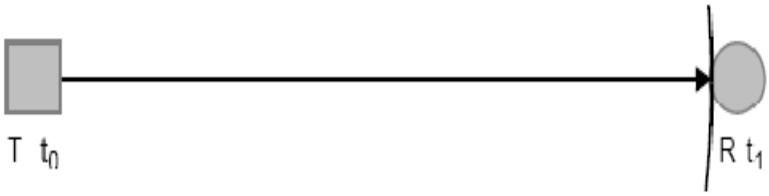

Figure 2.1. ToA from a Tag to a Reader

When the propagation time $t_i - t_0$, is multiplied by the propagation speed of the signal, the propagation delay can be converted into a distance between the tag and the reader. To determine the tag position in a 2D plane, at least three readers are required to take ToA measurements. Determining the tags position in a 2D plane, at least three receivers (readers) are required to take ToA measurements.

To determine the tag position in 3D space, at least four readers are required to take ToA measurements. In a 2D plane, the location of a tag can be seen as an intersection of circles, while in 3D space, the location of the tag can be seen as an intersection of spheres.

The Time of Arrival method for 2D range calculations can be illustrated as shown in Figure 2.2. Location Approximation through intersection., where the tag is denoted as *T,* while R_1, R_2, and R_3 are the readers. The signal is transmitted at the time moment t_0 and received by readers at the time moments t_1, t_2, and t_3 respectively.

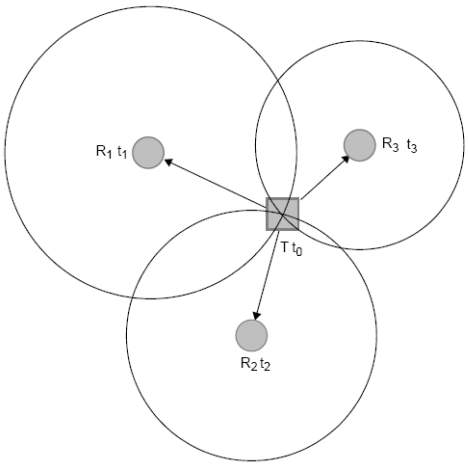

Figure 2.2. Location Approximation through intersection.

One of the main requirements of this approach is the synchronization of the clocks present on the tag and the reader. To attain a more precise distance measurement a timing precision up to the nanosecond scale is a requirement, which results in a more elaborate clock synchronization system.

2.2. Time of Flight (ToF)

The Time of Flight method [8] uses measured elapsed time for a transmission between a tag and a reader based on the estimated propagation speed of a typical signal through a medium. As this method is based on a time value, clock accuracy becomes significantly more important than in previous methods.

Readers R with highly accurate clocks is used which transmit signals with known departure time values to tags T (or other readers) also with highly accurate clocks. The departure time t_1 is compared to the arrival time t_2, and using an estimating the propagation speed of the signal S, the distance D between the devices

can be determined with accuracy within 1 or 2 meters. Using three readers, an algorithm can determine the location of the tag in 3D space.

The method is as shown in Figure 2.3, does not add additional hardware overhead to the system as it can use the same hardware that would be used for data communication and signal processing.

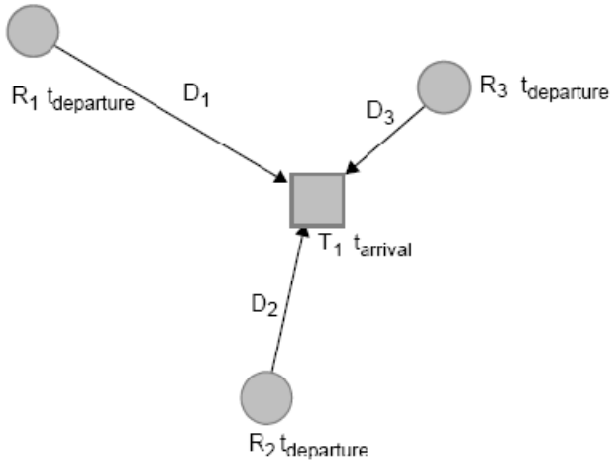

Figure 2.3. Determining tag position with ToF

An ideal ToF system requires costly accurate clocks. In reality, the clock offset and clock drift corrupt ranging accuracy. Also, the signal can be affected by interference from other signals, noise, and multipath propagation. Yet, ToF has an advantage over other systems as the cost of additional hardware is minimal. It is also reasonably successful in indoor environments, such as with concrete walls and floors, and it has a relatively high accuracy compared to other methods. Furthermore, ToF is considered to be a secure method for RTLS.

2.3. Angle of Arrival (AoA)

The Angle of Arrival, or AoA, is a method for determining the direction of propagation of an RF signal received from a tag at a reader. Using direction sensitive antennas on a reader, the direction to the mobile node can be obtained.

By measuring the angle between a line that runs from the reader to the tag and a line from the reader with a predefined direction, the AoA can be determined. This method can be illustrated as in Figure 2.4, where R_1 is the reader and T denotes the tag.

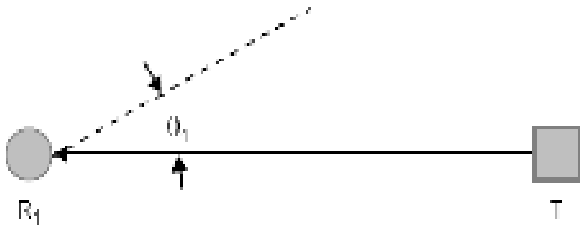

Figure 2.4. Angle of Arrival (AoA) method

The position of a tag transmitted to both readers can be determined using simple triangulation, using the positions of two readers at known locations. For each reader, the angle of arrival of the signal received from the same tag is calculated and then an algorithm is used by the location engine to determine the position of the tag. This method is illustrated in Figure 2.5, where two readers, R_1 and R_2, are used and where T denotes the tag whose position is being determined.

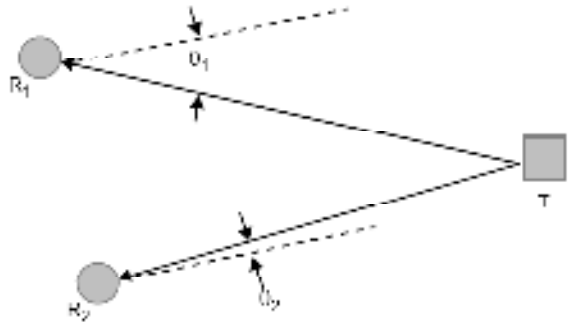

Figure 2.5. Determining tag position with AoA

The AoA method does have its set of drawbacks, taking measurements using this method often requires a complex set of antenna arrays or other complicated antenna solutions. The approach can be considered to be more of a theoretical feasibility rather than practical. To increase the accuracy of this method the number of antenna arrays used is usually increased. Also, multipath propagation common in building environments tends to affect the accuracy of the final result. Furthermore, the Angle of Arrival method is also susceptible to security threats as attackers can easily reflect or retransmit from a different location.

2.4. Received Signal Strength Indicator (RSSI)

Received Signal Strength Indication method uses several accesspoints (AP) simultaneously to track the location of a device. The signal strength of received signals from at least three AP's are used to determine the location of the object or person being tracked. In an RSSI system, the distance between a tag (object or person) and a reader (AP) is determined by converting the value of the signal strength at the reader into a distance measurement based on the known signal output

power at the tag (transmitter) and on a particular path-loss model. A location server running a localization algorithm estimates the location of the tag using the computed distances between the tag and several readers

The RSSI method can be illustrated as shown in Figure 2.6, where the tag is denoted as T and R1, R2, and R3 are the readers. The signal strength for each reader is denoted as S1, S2, and S3 respectively.

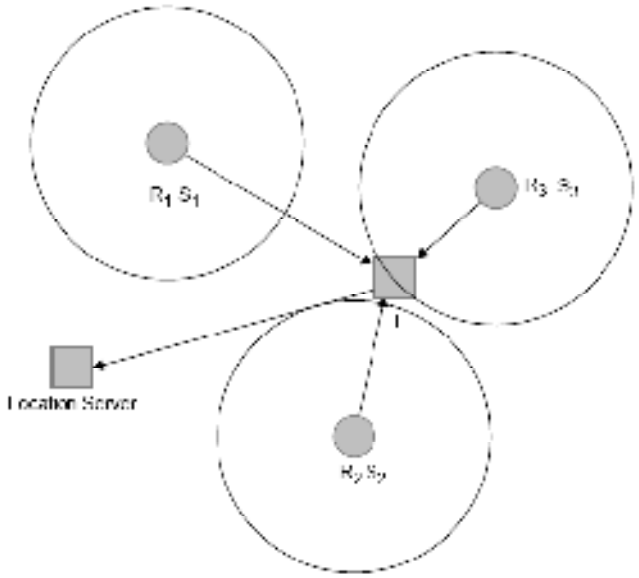

Figure 2.6. Determining tag position with RSSI

To be effective, RSSI requires a dense deployment of Access Points. However, the key problem related to RSSI based systems is that an adequate underlying path-loss model must be found for both non-line-of-sight and non-stationary environments. Consequently, in practice, estimated distances are somewhat unreliable.

Furthermore, several approaches using RSSI (e.g., the approach this Thesis is extending [4]) are quite cumbersome as they require a significant amount of human labor when collecting RSSI fingerprints. The approach used in the past for coming up with a model, relied on recording/logging the RSSI at every *cell*, specific to the indoor environment in which localization needs to work. Thus, the flexibility of using the same system in any other indoor environment is compromised, as RSSI model would have to be reestablished all over again.

CHAPTER 3
BAYESIAN FILTERS

We attempt a probabilistic based approach to localization, where the system determines what the probability is with which a target may be located at any point within the environment. After determining these probabilities across the area, the most likely location of the target may be determined by additional computing.

The technique described in the thesis is a probabilistic approach using a recursive Bayesian filter referred to as *sequential Monte Carlo sampling* (a.k.a. particle filters). The proposed technique computes a posterior distribution of the target's location using sequential Monte Carlo sampling, which is capable of using an arbitrary a-priori distribution to compute the posterior distribution. This method is less computationally intensive compared to a full mathematical Bayesian model and can be applied to an indoor wireless enabled environment where standardized distributions of RSSI readings may not be available.

3.1. Recursive Bayesian Filters

Bayesian filtering is the general term used to discuss the method of using a predict/update cycle. It is used to estimate the state of a dynamical system from sensor measurements.

We model the localization problem as a stochastic process in which estimates of location and orientation are represented as probability distributions. This means if s_n and d_n denote the state of the system and the RSSI measurement at time $t = n\Delta t$, respectively, where Δt is the sampling interval, we are trying to estimate the *evolving* state s_n given $(d_1, d_2, \ldots, d_n)$. A Bayesian filter is an algorithm that produces such an estimate, $p(s_n|d_1,....d_n)$ given a model of the measurements, $p(d_n|s_n)$, and the previous estimate, $p(s_{n-1}|d_1,....d_{n-1})$. This posterior distribution can then be used to compute any statistic of s_n.

A Recursive Bayesian filter algorithm imposes the constraint that the estimate of $p(s_n|d_1,....d_n)$ has to be generated using only:

- The previous posterior density $p(s_{n-1}|d_1,....d_{n-1})$.
- The most recent measurement d_n.

This way we conveniently avoid storing the entire measurement sequence and reduces the amount of computation performed. To achieve the desired posterior distribution $p(s_n|d_1,....d_n)$ an elaborate two step process is invoked that takes $p(s_{n-1}|d_1,....d_{n-1})$ and d_n as inputs. In the following sections we discuss the two step process in more detail.

3.1.1. Prediction (Model Update)

Using the new RSSI measurement, here we attempt to predict the exact location which the node has moved. This step maps the previous posterior distribution $p(s_{n-1}|d_1,....d_{n-1})$ into a prediction density $p(s_n|d_1,....d_{n-1})$.

$$p(s_n|\, d_1, \dots, d_{n-1}) = \int_{n-1} p(s_n|s_{n-1}, d_1, \dots, d_{n-1}) \,.\, p(s_{n-1}|\, d_1, \dots, d_{n-1})\, ds_{n-1} \quad \textbf{(3.1)}$$

This step is also known as *model update* since it updates the state space model based on prediction.

3.1.2. Measurement Update

This step combines a new observation d_n with the prediction density $p(s_n|d_1,....d_{n-1})$ to compute the desired posterior density $p(s_n|d_1,....d_n)$.

$$p(s_n|\, d_1, \dots, d_n) = \frac{p(d_n|s_n, d_1, \dots, d_{n-1}).p(s_n|\, d_1, \dots, d_{n-1})}{p(d_n|\, d_1, \dots, d_{n-1})} \quad \textbf{(3.2)}$$

The denominator in the update step is determined by integrating the numerator:

$$p(d_n|\, d_1, \dots, d_{n-1}) = \int p(d_n|s_n, d_1, \dots, d_{n-1}).\, p(s_n|\, d_1, \dots, d_{n-1})\, ds_n$$

Upon examining the equations 3.1 and 3.2, we can notice the presence of the terms $p(s_n|s_{n-1}\ ,\ d_1,....d_{n-1})$ and $p(d_n|s_n\ ,\ d_1,....d_{n-1})$ that require all the previous measurements that were collected. Thus, to perform Bayesian filtering recursively, we must have some constraints, such that eliminating previous measurements from equations 3.1 and 3.2. The next subsection explains such constraints.

3.1.3. Constraints

- **Constraint 1:** Assume that the state sequence is Markovian, i.e.,

$$p(s_n|\, s_1, \dots, s_{n-1}) = p(s_n|\, s_{n-1})$$

Any random process is said to be Markovian if the future of the process given the present is independent of the past, i.e. the present state contains the entire past.

- **Constraint 2:** Memory less channel, is a second assumption that we make, i.e., observations have to occur through a memory less channel (d_n is assumed to be independent of all states but s_n and also all other measurements). This translates to:

$$p(d_n|S_{1\dots n}) = \prod_{i=1}^{n} p(d_i|S_i)$$

Applying the first constraint the *Prediction Step,* equation 3.1 is reduced to:

$$p(s_n|\, d_1, \dots, d_{n-1}) = \int_{n-1} p(s_n|s_{n-1}) \,.\, p(s_{n-1}|\, d_1, \dots, d_{n-1})\, ds_{n-1} \qquad \textbf{(3.3)}$$

Applying the second constraint the *Measurement Update Step,* equation 3.2 can be rewritten as:

$$p(s_n|\, d_1, \dots, d_n) = \frac{p(d_n|s_n).p(s_n|\, d_1,\dots,d_{n-1})}{p(d_n|\, d_1,\dots,d_{n-1})} \qquad \textbf{(3.4)}$$

The prediction model that has been discussed so far is of the form:

$$I(f) = \int f(s) \cdot p(s|d) ds \qquad \textbf{(3.5)}$$

The solution to this is easily rendered computationally infeasible for non-standard distributions *p(s|d)* (such as the distributions that could be used to describe a propagation environment). Therefore, instead of seeking an analytic solution, we propose the use of Monte Carlo integration techniques, which sample the state space at random, independent of the number of dimensions. Using these techniques,

samples are drawn from a proposal distribution $g(s|d)$ rather than the original $p(s|d)$, where $g(s|d) \propto p(s|d)$. The integration technique used is *sequential importance sampling*, a.k.a. particle filtering, a brief discussion of which is in the following chapter.

CHAPTER 4
MONTE CARLO SAMPLING AND BAYESIAN FILTERING

4.1. Sequential Importance Sampling

$$I(f) = \int f(s) \cdot \mathrm{p}(s|d) ds \quad \textbf{(4.1)}$$

Importance sampling permits the derivation of *I(f)* by sampling from an arbitrary proposal distribution, *g(s|d)*, given that *g(s|d) > 0* whenever *p(s|d) > 0* (to guarantee that samples can be drawn for all states for which *p(s|d)* is non-zero). The discussion in this section draws upon work presented in [9]. We can rewrite equation 4.1 as:

$$I(f) = \int f(s) \cdot \mathrm{w(s)}. g(s|d) ds \quad \textbf{(4.2)}$$

Where: $w(s) = \frac{p(s|d)}{g(s|d)}$

Now, N_p independent samples $\{s^{(i)}\}$ can be drawn according to *g(s|d)* to approximate *I(f)* using Monte Carlo integration as:

$$I_{N_p}(f) \triangleq \frac{1}{N_p} \sum_{i=1}^{N_p} f(s^{(i)}) . w^{(i)}$$

Where, $w^{(i)} \equiv w(s^{(i)})$, This set is referred to as importance weights. From above rewriting the equation we get:

$$I(f) = \int f(s) \sum_{i=1}^{N_p} (w^{(i)}.\delta_{s^{(i)}}(s))\, ds$$

Where, $\delta_{s^{(i)}}(s) = \delta(s - s^{(i)})$ is the Dirac Delta function. If the empirical measure that is generated by samples $s^{(i)}$ drawn from $g(s|d)$ is denoted by $p_{N_p}(s|d)$ then:

$$p_{N_p}(s_n|d_{1\ldots n}) \triangleq \sum_{i=1}^{N_p} w^{(i)}.\delta_{s^{(i)}}(s) \qquad \textbf{(4.3)}$$

From equations 4.2 and 4.3 we can imply:

$$I_{N_p}(f) = \int f(s) \cdot p_{N_p}(s|d) ds \approx \int f(s) \cdot \mathrm{p}(s|d) ds$$

Where, the approximation improves as $N_p \to \infty$, i.e., as the number of samples chosen increase. The generated random measure represented by equation 4.3, not only contains a set of random values or *support points*, but also the *importance (weight)* of each support point. The complete specification of the distribution in equation 4.3 can be represented by the set $\{s^{(i)}, w^{(i)}\}_{i=1}^{N_p}$ and each such support point, denoted by $s_{0\ldots n}^{(i)}$, is a randomly generated *sequence of states* for our Bayesian filtering context. Given the previous state of the system, $\{s_{0\ldots n-1}^{(i)}, w_{n-1}^{(i)}\}_{i=1}^{N_p}$, we need to derive the current state of the system, $\{s_{0\ldots n}^{(i)}, w_n^{(i)}\}_{i=1}^{N_p}$ where the weights $w^{(i)}$ are computed as:

$$w_n^{(i)} = \frac{p\left(d_n \middle| s_n^{(i)}\right).p\left(s_n^{(i)} \middle| s_{n-1}^{(i)}\right)}{g\left(s_n^{(i)} \middle| s_{n-1}^{(i)}.d_n\right)} * w_{n-1}^{(i)} \qquad \textbf{(4.4)}$$

4.2. Degeneracy and Resampling

A common problem with sequential importance sampling is *degeneracy*, where, after a few iterations, all but one support point will have negligible weights. Consequently, a significant amount of computation is spent on updating particles that do not contribute to the approximation of $p(s_n|d_{n-1})$ and the quality of the approximation decreases over time. To reduce the effects of degeneracy, it is necessary to remove particles with insignificant weights and concentrate on those with significant weights. A pseudo code description for such an algorithm is as shown in Figure 4.1.

$$\left[\{s_n^{(i)}, w_n^{(i)}\}_{i=1}^{N_p}\right] = PF\left[\{s_{n-1}^{(i)}, w_{n-1}^{(i)}\}_{i=1}^{N_p}, d_n\right]$$

FOR i=1 TO N_s

Draw $s_n^{(i)} \approx g(s_n|s_{n-1}^{(i)}, d_n)$

Assign the particle a weight, $w_n^{(i)}$ according to equation 4.4

END FOR

Calculate total weight $t = \Sigma\left[\{w_n^{(i)}\}_{i=1}^{N_p}\right]$

FOR i=1 TO N_p

Normalize $w_n^{(i)} = t^{-1}.w_n^{(i)}$

END FOR

Calculate N_{eff} using $N_{eff} = \frac{1}{\sum_{i=1}^{N_p}(w_n^{(i)})^2}$

IF $N_{eff} < N_s$

Resample according to $w_n^{(i)}$

END IF

Figure 4.1. Particle Filter Algorithm

This is accomplished, through a technique known as *resampling*, by drawing N_p independent samples from $p_{N_p}(s_n|d_{1\ldots n})$ whenever degeneracy falls below some threshold N_τ. Because resampling draws from a true posterior (in an approximate sense), the resampled weights remain uniform throughout. The support point set $\{s_n^{(i)}\}_{i=1}^{N_p}$ will be referred to as *"particles"*.

CHAPTER 5
WIRELESS LOCALIZATION USING PARTICLE FILTERS

As shown in the previous chapter, a particle filter allows for a high degree of flexibility as far as the state model is concerned. The filter is capable of handling substantially complex state models including those that are non-linear and non-Gaussian. This allows the use of arbitrary process models, which is of particular interest since a model of wireless signal strength distributions should be neither linear nor Gaussian.

In the following sections we will give a high level view of particle filters, where the discussion will more oriented towards utilizing the properties of a particle filter for the purpose of wireless localization.

5.1. Reference frames for Localization

Before starting out, we should define which parameters define the location of a mobile node. The location of a node within a system utilizing a particle filter would correspond to the state of that object. Since, our application is restricted solely for the purpose of indoor wireless localization; we define the physical structure of the indoor environment as a reference frame. A specific point in the environment is specified as the origin and the location of the mobile node is specified in terms of

cartesian coordinates **(X, Y)** with respect to this origin. These, coordinates form the first two parameters that define the location of our mobile node.

A second reference frame is necessary to specify orientation $\boldsymbol{\theta}$ of the mobile node. To specify the orientation completely, we first need to specify an internal reference frame that changes orientation with respect to an external reference frame, (i.e., the infrastructure or "Earth"). The orientation between the mobile node and AP will be considered as the internal reference (This means that the 0 degree direction coincides with one of the coordinate-axis).

5.2. Particle Filter Models

We are primarily interested in tracking the location of a wireless mobile node or device in an indoor environment and thus need to develop valid measurement and mobility models, which will in turn be used by the *measurement update* and *system update* respectively.

We define the location of the mobile node in terms of its position in space and its orientation relative to a reference frame. The movement model depends on the agent carrying the mobile node (e.g. the movement model for a human carrying a laptop computer would be considerably different from that of a wireless enabled robotic vacuum cleaner). The measurement model, on the other hand, is largely a function of the environment and of the location of the access points.

5.2.1. Mobility Model

A typical movement model representing the motion of a mobile node generally consists of velocity and/or acceleration parameters. Considering, a more detailed view, each particle is defined by a set of three dimensions which are updated based on a zero-mean Gaussian model. Consequently, based on the accelerometer and gyro sensor data, we pick a normally distributed random variable to update the location of each particle during the system update step.

5.2.2. Measurement Model

A typical wireless communication system consists of at least two nodes exchanging information with each other. For the proposed localization approach to work, it is necessary that one of these nodes be at a fixed location at all times and we require a setup where the mobile node, or target, communicates with one or more wireless access points.

In our setup we extract the RSSI readings from *control packets* transmitted by access points. To estimate the location of a mobile node from such readings a statistical representation of the RSSI reading corresponding to distances is necessary. Building such a representation involves collecting a number of RSSI readings from an access point using a mobile tag at various distances and various orientations (angles of rotation) towards the access point. An experimental setup showing an access point mounted on a tripod and a rotational template is shown in Figure 5.1.

Figure 5.1. An *Orientation Board* lined up with an AP.

Essentially, the above collection of readings gives us a measurement model for the particle filter, represented by a matrix $M_{i,j}$ of measured two-tuples (average reading and standard deviation of readings), where I is an index corresponding to distance and j is an index corresponding to orientation (0 to 360 degrees).

Since it is infeasible in practice to collect readings at every possible location the mobile node may be in, it is necessary to come up with the above mentioned approach which allows for a flexible and common measurement model irrespective of the reference area. Recording such measurement model would ideally be a onetime routine. During run time, based on the surroundings (presence of walls/ obstacles, recorded in the map) one can vary the mean or standard deviation of the values within $M_{i,j}$.

5.3. Particle Filter Phases

The main objective of particle filtering is to "track" a variable of interest (the location of the mobile node) as it evolves over time, typically with a non-Gaussian and potentially multi-modal probability density function (pdf). The basis of the method is to represent the entire pdf [10] by the aid of samples. Based on a model a series of actions are taken, each one modifying the state of the variable of interest. The variable is fine-tuned by observations that arrive periodically.

Multiple representations (particles) [10] of the variable of interest are used, each one associated with a weight that signifies the quality of that specific particle. The particle filter algorithm is recursive in nature and as described in Chapter 4, operates in two phases: *prediction* and *update*.

In the *prediction stage* after each action, properties of particles (i.e., their coordinates) are modified according to the movement of the tag. The mobility model of the tag includes a perceived randomness representing the uncertainty of the mobility model itself (see Chapter 4 for a more detailed description). In the following *measurement update stage* the weight of each particle is updated based on the latest sensory information which in our case would be RSSI based [10]. Finally, resampling will eliminate particles with small weights.

The variable of interest [10] in our case the position of the mobile node at time t=k may be represented as $X^k = \left[x^k, y^k, \hat{\theta}^k\right]^T$. This in turn may also be represented as a set of M samples/particles($S_i^k = \left[X_j^k, w_j^k\right] : j = 1 \ldots . M$), where j

denotes the index of the particle. Each particle represents some probability that the variable of interest has the same values as the particle with a weight w_j^k (that defines the importance of this particle to the overall estimate of the variable).

If at time *k* we know the probability distribution of the system at the previous instant time *k-1* then we model the effect of the action to obtain the a priori of the probability distribution at time *k* (prediction) [10]. In other words, the prediction (system update) phase uses a model in order to simulate the effect an action (with a particular level of uncertainty) has on the particle set. The update phase uses the observations made by the sensors to update the weights of the particles to have a more accurate reasoning on the measurements.

Algorithmically, we can sum up the above discussion as follows:

- To begin, sample from initial probability distribution. If we have no prior data about the mobile node's location then we can use a uniform distribution of spreading particles over the target area.
- For each iteration, execute the following three processes:
    1) Prediction
    2) Update
    3) Resample (optional)

The, following subsections describe each of the above steps in detail and how they are used in conjunction with the sensory data.

5.3.1. System Update (Prediction)

For the purpose of predicting the probability distribution of the position of the mobile node after a change in the node's position, we need to have a model of the effect of noise on the resulting position *(System Model)*, which we will discuss later.

Based, on what has been discussed in section 5.1 every particle is represented by a three-tuple: $\{X,Y,\theta\}$, corresponding to a certain pre-defined reference axis. In each iteration we receive a mobility descriptor from the mobile node containing information related to recent position and angular displacement $\{\Delta x,\Delta y,\Delta\theta\}$. During system update, the $\{\Delta x,\Delta y,\Delta\theta\}$ values are used as a basis to calculate a probabilistic displacement for each of the particles.

One important note here is that in our approach, $\Delta x,\Delta y$ correspond to a co-ordinate system in which the mobile node starts, which may or may not be the same as the internal Cartesian reference that defines the location of our mobile node. So it is important that initially, the mobile node matches its Cartesian axis with the internal reference axis and will wait for the filter to correct this assumption.

Considering, a more detailed view, each particle is defined by a set of three dimensions which are updated based on a zero-mean Gaussian model. The standard deviations of these models are specified within a configuration file (described in Chapter 7) and may be fine tuned.

During the system update, one important issue is that of the dimensions of a particle being logically incorrect. Since, it is not possible for a mobile node to pass

through a wall, the same would hold true for the particles as well. Therefore after updating the particle dimensions, we check to make sure that the new displacement related dimensions do not make a particle seem, as though it passed through a wall. If a particle falls under this case, its weight is reassigned to zero and it is as good as non-existent in the posterior distribution. The number of particles is kept constant throughout the experiments.

5.3.2. Measurement Update

After an action (the movement of the mobile node) the RSSI readings from access points are employed in order to evaluate the distance of the node from the access points. This information is very noisy, and one of the tasks of the filter is to compensate for this noise. Each particle will have to be reevaluated (re-weighed) based on this new evidence.

In order to determine the weight of a particle, we first compute the density value for the current reading from the measurement model. Based on previous research carried out [4], where RSSI readings at a particular point were determined to follow a Gaussian distribution, we use the Gaussian probability density function.

The measurement update in our approach can be further broken down into four major steps. The first step deals with assigning weights to particles based on the RSSI readings received from various AP's. For a particular AP RSSI reading, the weight of the particle is calculated by the following steps:

- Calculate the distance and orientation of the particle relative to the AP.
- Use these values to look-up the measurement model matrix, which returns the Mean ($RSSI_{mean}$) and Std-Dev ($RSSI_{stdev}$) for the RSSI reading corresponding to distance, and orientation input.
- Since, the measurement model is Gaussian based, we can find the weight for a particular AP RSSI ($RSSI_{input}$) by the following equation:

$$Density = \frac{1}{RSSI_{stdev} \cdot \sqrt{2\pi}} * e^{-\frac{(\mathrm{RSSI_{input}} - \mathrm{RSSI_{mean}})^2}{2 \cdot RSSI_{stdev}^2}}$$

On receiving multiple RSSI readings, for a particular particle we calculate the weight for every reading, multiply these weights and then assign it to the particle. The second step would deal with generating a Cumulative Distribution based on the weights assigned earlier, followed by normalizing them. Lastly a uniformly distributed random variable is used as the threshold N_τ during the resampling step.

5.3.3. Resampling

One of the problems, with the use of Particle Filters is the degeneration of the population after a few iterations. Many particles may receive weights that become insignificant to contribute to the location probability representation of the mobile node. Figure 5.2 visualizes the *importance resampling* process to avoid degeneration.

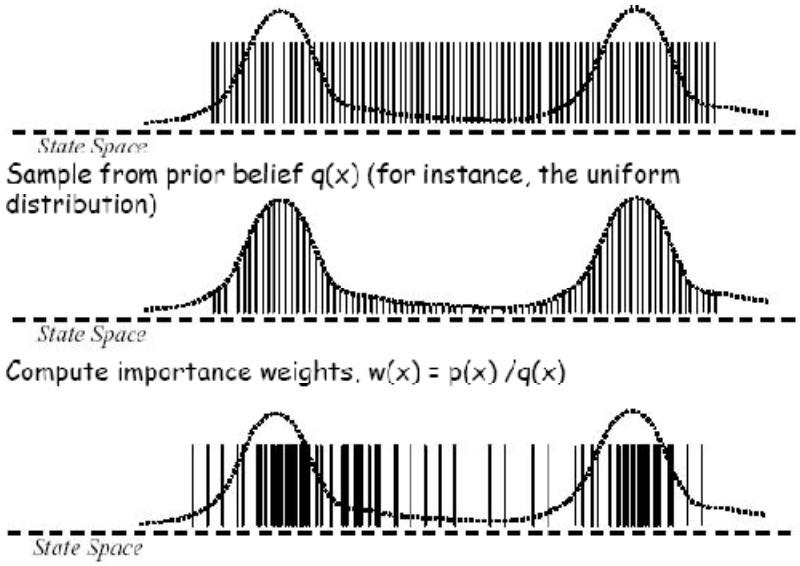

Figure 5.2. Resample particles according to importance weights to get p(x)
Samples with high weights chosen many times; density reflects pdf

Let us consider a situation where the posterior to be represented, is a bimodal distribution (i.e., the "two-humped" dotted line in Figure 5.2). Resampling provides a computationally much more feasible solution by rearranging the distribution of the particles (as seen in Figure 5.2) [11]. First the prior (the uniform distribution) distribution is sampled. For each of those samples, we find the value of the posterior p(x). So for each sample, we assign that sample a weight, w(x), equal to p(x)/q(x). At this point, when the particles are weighted, we can use the highest-weighted (highest probability) sample as the best-guess state, or we can use the weighted sum of particles to get a mean-equivalent (which may be misleading), or we can use the location with the highest density of particles for a more intelligent best-guess.

Resampling within our setup is performed only on receiving a certain number of RSSI readings. This is a value that can be set within the configuration file described in Chapter 7. The idea behind performing the resample only on receiving a certain number of RSSI readings is to remove any *bias* within the algorithm. Bias, would refer to a scenario where resampling of the particles is performed immediately after the *measurement update* which was carried out based on partial RSSI readings. For example, if the tag is closer to AP1, but in a report receives RSSI readings only from AP2 which is quite farther away, due to the immediate resampling step following a measurement update based on the AP2 RSSI reading alone, we may observe a movement of the particles towards AP2. This would be incorrect because in reality the user would be near AP1. To account for such a case the algorithm waits for a certain number of RSSI readings before carrying out a resample.

CHAPTER 6
DESCRIPTION OF THE CUSTOM IEEE 802.15.4 - SENSOR NODES

In this section we are going to describe the wireless nodes that were designed and developed for the system. These nodes communicate with a chipset designed for the wireless standard IEEE 802.15.4 [12].

6.1. Node variants

Based on functionality we have three types of wireless nodes:

- Basestation (Sink)
- Accesspoint (Relay node)
- Tag (Mobile node)

In our experimental setup all the above devices have the same basic hardware layout with some small differences. The *Tag* is outfitted with a 3D rotational sensor and a 3D accelerometer for collecting displacement (mobility) related data. Each node communicates through an XBee-Pro [13] 802.15.4 based radio transceiver module manufactured by Digi and is controlled by a Texas Instruments 16-bit low power microcontroller (MSP430) [14].

The firmware for each of the hardware modules can be easily updated. The XBee-Pro transceivers can be updated with various versions of firmware as provided by the manufacturer using the XCTU software [15]. The MSP430 MCU's firmware

is developed by us in C and is flashed onto the microcontroller using a JTAG interface. In the following sections we briefly describe the role and functionalities of each of the node types.

6.1.1. Basestation (BS)

The base station is the sink of the communication. All *data packets* coming from tags are relayed to this node, which in turn relays them to a host computer using a USB interface. The BS is thus:

- Responsible for collecting all relevant data and *control packets* being transmitted by access points and tags.
- Transmits periodic *control packets* for the purpose of route establishments.
- Is serially connected (USB) to a computer, running the *localization engine*.
- Receives data and *control packets* being broadcasted to help with diagnostic trace data while testing.

6.1.2. Accesspoint (AP)

The access points are static transceivers statically and strategically placed in the target environment. Their tasks are:

- Relaying (forwarding) *data packets* from a tag or another AP to the BS.
- Transmitting periodic *control packets* within the network based on which tags record their corresponding RSSI, which in turn is relayed to the sink to be used by the particle filter.

6.1.3. Tag (Mobile Node)

Tags are wireless transceivers that can be used to tag personnel or equipment. The goal of the RTLS is to localize these tags within the target environment. Tasks of the tags are:

- Attached to entities, which need to be located within a target area.
- Keeping track of the attached entities mobility using inertial sensors.
- Recording the RSSI readings of broadcast packets received from access points relaying such information along with mobility related data to the access points.

6.2. Node Interaction

The above described three types of devices together coordinate and provide for a well knit resilient wireless infrastructure. We will consider the most basic scenario where there is a single BS, multiple AP's and one tag (that we intend to localize).

At the network level, it is important that every node has in some way knowledge, at least about its immediate *one-hop neighborhood*. This, in turn supports the ability of the network, to incorporate some level of resiliency within itself. Resiliency would essentially refer to providing a level of flexibility and fault tolerance within a network which leads us to the function of routing. Routing encompasses the task of building routing tables within access points that are consulted when making forwarding decisions to relay tag data towards the sink.

Based, on the requirements that have been discussed above, routing within a typical network is achieved by defining two types of packets:

- Control Packet
- Data Packet

The system also defines a third type, a "Comment Packet" which is primarily used for data logging and debugging. The flexible nature of the packet format allows for adding any new types in the future. In the following sections we will discuss the role of each of the above mentioned packets and how they are used within the network. The firs payload byte within each packet corresponds to the type of the packet.

6.2.1. Control Packet

A *control packet* is *broadcasted* by the AP's and BS. When received by an AP or Tag, the payload information is utilized to update their corresponding next-hop table. The packet type is "**0xFF**" and the format of the packet is shown in Figure 6.1.

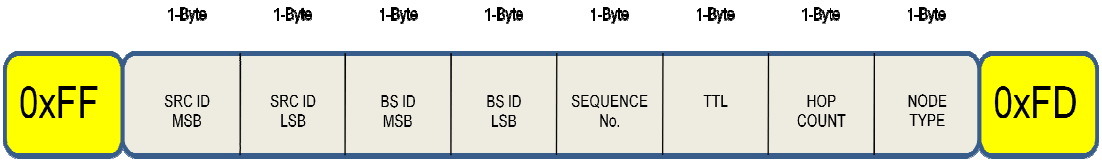

Figure 6.1. Format of a *control packet*

Additional fields as seen in the figure are:

- **Source ID:** This field is two bytes long and is used to indicate the 16-bit source address of the node broadcasting the *control packet*.
- **BS ID:** A two byte field indicates the 16-bit address of the BS present within the network.
- **Sequence Number:** One byte in length, this field is used to indicate the packet count being broadcasted by the BS. It is a particularly important field for the receiver node, which uses it to determine if it needs to probe the packet for the hop-count. This field is specific to the BS and once transmitted by the BS it is piggybacked on the remaining *control packets* within the network.
- **TTL (Time to Live):** This field is 1 byte in length, and is updated on receiving a new *control packet* and decremented by every node prior to transmission. This field determines for how long a BS specific SEQ NO and TTL field should exist within a network. This field is specific to the BS and once transmitted by the BS it is piggybacked on the remaining *control packets* within the network.
- **Hop Count:** This is a 1 byte field, and is used to determine the Next-Hop node for a particular node that is probing this field. This field holds the cumulative count of number of nodes it's away from a BS. The field is incremented by a node prior to transmission of a *control packet*.
- **Node Type:** A byte long field**,** indicating the type of node transmitting the *control packet* **'**T' for Tag, 'B' for BS and 'A' for an AP.

6.2.2. Data Packet

A data packet is *unicast* by a tag to an access point and is relayed (unicast) by the AP-s until it reaches the BS. When received by a BS, the payload information is utilized by the Particle Filter. The packet type is "**0xFE**" and the packet is depicted in Figure 6.2.

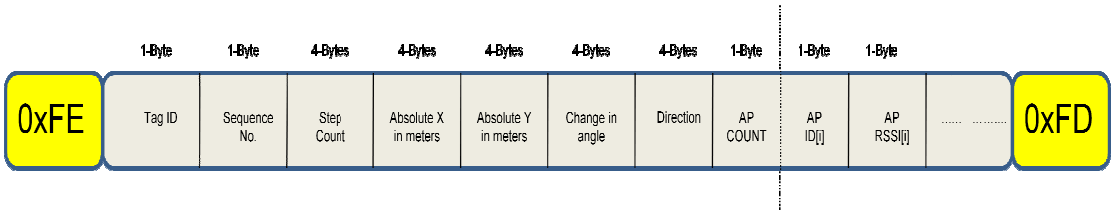

Figure 6.2. Format of a *data packet*

We will describe the fields in detail:

- **Tag ID:** one byte long field stores the lowest eight bits of the tag's address.
- **Sequence Number:** one byte long; varying from 0-127. This field is used to indicate the packet count being unicast by the tag.
- **Step Count: A** four byte field which holds the number of steps taken by the user holding a tag.
- **Position X:** A four byte field indicating the displacement of the tag in "X direction".
- **Position Y:** A four byte field indicating the displacement of the tag in "Y direction".
- **Change in Angle:** A four byte field indicating the change in tag's orientation.

- **Direction:** A four byte field indicating the absolute direction covered by the tag.
- **AP Count:** A one byte field indicating the number of AP RSSI readings present in the packet.
- **AP ID:** A one byte field indicating the ID of the AP from which the RSSI has been received.
- **AP RSSI:** A one byte field indicating the RSSI reading corresponding to an AP.

6.2.3. Comment Packet

A Comment Packet can be *broadcast* by any of the AP-s, BS-s and tags. The payload information is utilized for debugging and information purposes. The packet type is "**0xFC**". The payload data can be a generic sequence of characters.

6.3. Node Features

Together, the three types of nodes co-operate using the various packet types and provide a set of features which aid the localization algorithm. The APs, BS and tags together can co-operate to provide the network with the capability of routing signaling packets. Furthermore, the tag by itself has the ability to determine two dimensional displacement and change in orientation, which are used to build the system model for the *particle filter.*

6.3.1. Routing

This is a feature required by the system to work when the tag is not within the signal range of the Base-Station running the localization algorithm. The Tag unicasts *data packets* including raw location information to its next hop AP or BS node. If the next hop is an AP, the AP in turn relays (unicasts) this *data packet* to its corresponding next hop node. This sequence is followed until the *data packet* reaches a BS. Thus AP's form a wireless mesh network over the target area.

Next, we will discuss the sequence of steps for a certain scenario, which best describes the routing feature. Observing the four diagrams in Figure 6.3 it is clearly visible that in our example the tag is out of range of the BS. We describe the above example in three sections, dealing with the state of the system, node interaction and lastly the criteria based on which the nodes interact.

1. **State Description:**
    - All AP's and BS-s are periodically transmitting *control packets*.
    - The AP's within the range of the BS are AP1 and AP3 (in the Figure).
    - The AP's within the range of AP1 are AP2, AP4 and the BS.
    - The node in the range of AP4 is the tag which needs to get across a *data packet* to the BS.

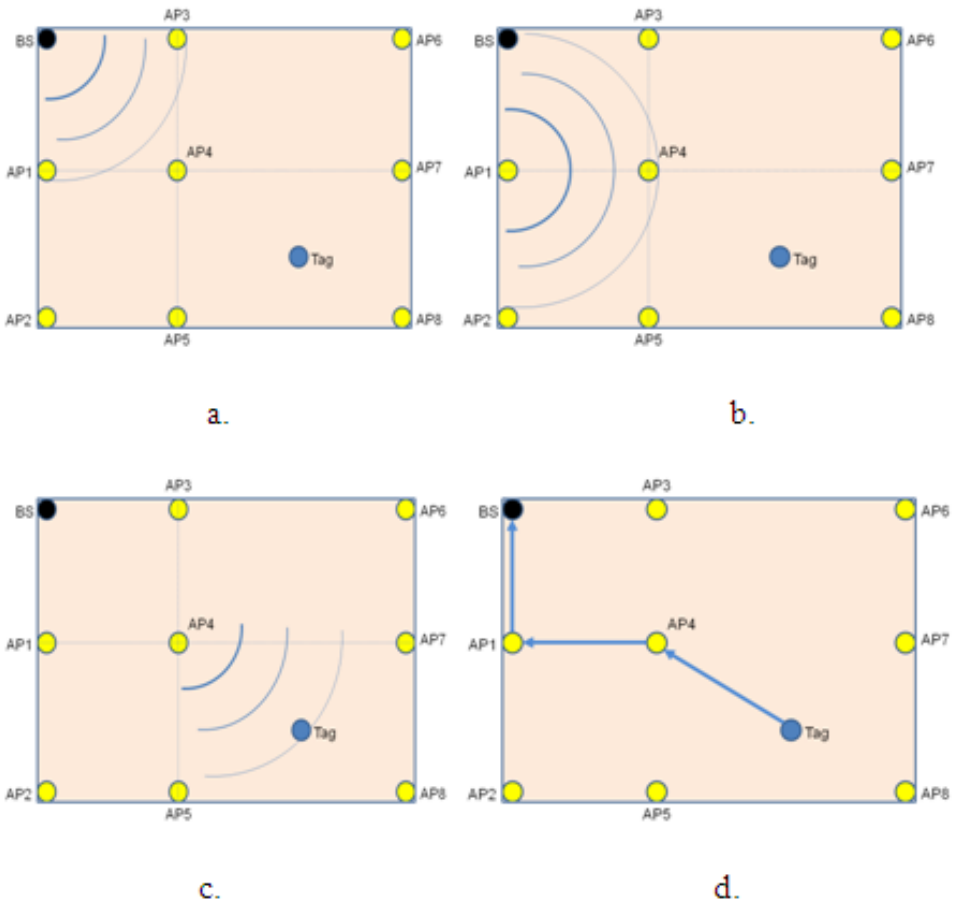

Figure 6.3. Routing steps a) AP4 not in the range of BS b) AP4 in the range of AP1 c) Tag in range of AP4 d) Tag unicasting *data packet* to BS.

2. **Interaction Description:** Based on the above state description, the following sequence of events takes place:

   - The BS broadcasts a *control packet* with a *HOP_COUNT* field set to 1. On receiving this *control packet* AP1 and AP3 nodes update their local *NEXT_HOP* fields to BS, indicating that the BS is just one hop away from each of them.

- AP1 in turn broadcasts an updated *control packet* with a *HOP_COUNT* field set to 2. On receiving this *control packet* AP4 updates its local *NEXT_HOP* fields to AP1, indicating that the BS is two hops away.
- Next, AP4 starts to broadcast an updated *control packet* with a *HOP_COUNT* field set to 3. On receiving this *control packet*, the tag updates its local *NEXT_HOP* field to AP4, indicating that the BS is three hops away.
- Now, when the tag has a *data packet* to send to the BS its sends it to its immediate *NEXT_HOP*, which in its case is AP4. An AP on receiving a *data packet* relays or unicasts this *data packet* to its corresponding *NEXT_HOP*. This continues till the *data packet* is received by the BS.

3. **Update Criteria:** The updates described above are based on a certain criteria, dependent on the values of the field present within the received *control packet*. We represent the update sequence by means of pseudo-code shown in the consecutive figures.

```
/*Called only by Basestation on receiving a Control or Data Packet*/
if (LOCAL_NODE_TYPE == 'BS')
{
        //RF data within a Control Packet is serially passed on to the PC
        apiout_transmitdata_usb();
}

/*Called only by Accesspoint and Tag on receiving a Control or Data Packet*/
else if(LOCAL_NODE_TYPE == 'Tag'|| LOCAL_NODE_TYPE == 'AP')
{
        apiin_process_rxframe();
}
```

Figure 6.4. Condition steps when a *control* or *data packet* is received.

- Updates occur only on an AP or tag, only if the *control packet* is transmitted by an AP or BS, as shown in Figure 6.5.

```
apiin_process_rxframe()
{
        if (RECV_NODE_TYPE == 'BS' || RECV_NODE_TYPE == 'AP')
        {
                if (Condition 1)
                if (Condition 2)
                else if (Condition 3)
        }
        if (LOCAL_NODE_TYPE == 'Tag')
        {
                //Store the RSSI reading only if received from an Accesspoint.
                if (RECV_NODE_TYPE == 'AP')
                {
                        insert_RSSI(Received RSSI);
                }
        }
}
```

Figure 6.5. Processing steps on receiving a *control packet*.

- Another scenario is when a tag or AP does not receive a *control packet* from the chosen *NEXT_HOP* node for more than a parameter amount of time, in which case the table is reset. This is shown in Figure 6.6.

Condition 1: If a tag or AP does not receive a gossip packet from the chosen next hop device for more than GOSSIP_NOTRECEIVED_LIMIT times continuously. Change the LOCAL_HOP_COUNT to max value.

```
if (RECV_SRC_ID != UNICAST_ID)
{
        NEXTHOP_GOSSIP_NOTRECEIVED_COUNT++;
        if (NEXTHOP_GOSSIP_NOTRECEIVED_COUNT == GOSSIP_NOTRECEIVED_LIMIT)
        {
                //Reset the update condition variables
                LOCAL_SEQ_NUM = 0;
                LOCAL_HOP_COUNT = 99;
                NEXTHOP_GOSSIP_NOTRECEIVED_COUNT = 0;
        }
}
else
{
        NEXTHOP_GOSSIP_NOTRECEIVED_COUNT = 0;
}
```

Figure 6.6. Condition 1 of processing a *control packet*.

- As shown in Figure 6.7 if a tag or AP receives a *control packet* from a previously chosen *NEXT_HOP*, then the *NEXT_HOP* update will occur only if the received *SEQ_NUM* is greater than the already stored *SEQ_NUM* and the received *HOP_COUNT* is less than the local *HOP_COUNT*.

Condition 2: If a Tag or AP receives a packet from the previously chosen next hop node.

```
if (RECV_SRC_ID == UNICAST_ID)
{
        if ((RECV_SEQ_NUM>LOCAL_SEQ_NUM) && (RECV_HOP_COUNT < LOCAL_HOP_COUNT))
        {
                LOCAL_SEQ_NUM = RECV_SEQ_NUM;
                LOCAL_TTL = RECV_TTL;
                LOCAL_HOP_COUNT = RECV_HOP_COUNT;
                //Updating the next nearest hop address
                NEXTHOP_ID = RECV_SRC_ID;
                UNICAST_ID = NEXTHOP_ID;
                LOCAL_BS_ID = RECV_BS_ID;
        }
}
```

Figure 6.7. Condition 2 of processing a *control packet.*

- As shown in Figure 6.8 if a Tag or AP receives a packet from a node other than the previously chosen *NEXT_HOP* node, then the update occurs only if the received *HOP_COUNT* is less than the local *HOP_COUNT*.

Condition 3: Else if a Tag or AP receives a packet from a different node other than the previously chosen next hop node

```
else if (RECV_HOP_COUNT < LOCAL_HOP_COUNT)
{
        LOCAL_SEQ_NUM = RECV_SEQ_NUM;
        LOCAL_TTL = RECV_TTL;
        LOCAL_HOP_COUNT = RECV_HOP_COUNT;
        //Updating the next nearest hop address
        NEXTHOP_ID = RECV_SRC_ID;
        UNICAST_ID = NEXTHOP_ID;
        LOCAL_BS_ID = RECV_BS_ID;
}
```

Figure 6.8. Condition 3 of processing a *control packet*.

- As shown in Figure 6.9 if an AP receives a *data packet* from another AP or Tag it then calls an API function to relay this *data packet* to its corresponding *NEXT_HOP* node.

```
apiin_process_rxframe()
{
        if (LOCAL_NODE_TYPE == 'AP')
        {
                apiout_insert_relaytxframe();
        }
}
```

Figure 6.9. Processing steps on receiving a *data packet*.

6.3.2. Detecting a Change in the Tag's Orientation

To detect the change in the orientation of the tag we utilize the A/D readings from the onboard analog rotational sensor. Based on the data sheet of the employed chip [16] and empirical observations, there is a linear relationship between the AD readings and the angular rate (degrees/second).

To calibrate the sensor we used a turn-table (vinyl record player) with well defined angular speeds of 33 and 45 revolutions/minute respectively. By placing the tag on the turntable and running it at each of the above mentioned speeds, we can calibrate the A/D readings for the tag. Figure 6.10, depicts the linear response which allows us to find the angular rate for any A/D value at run time.

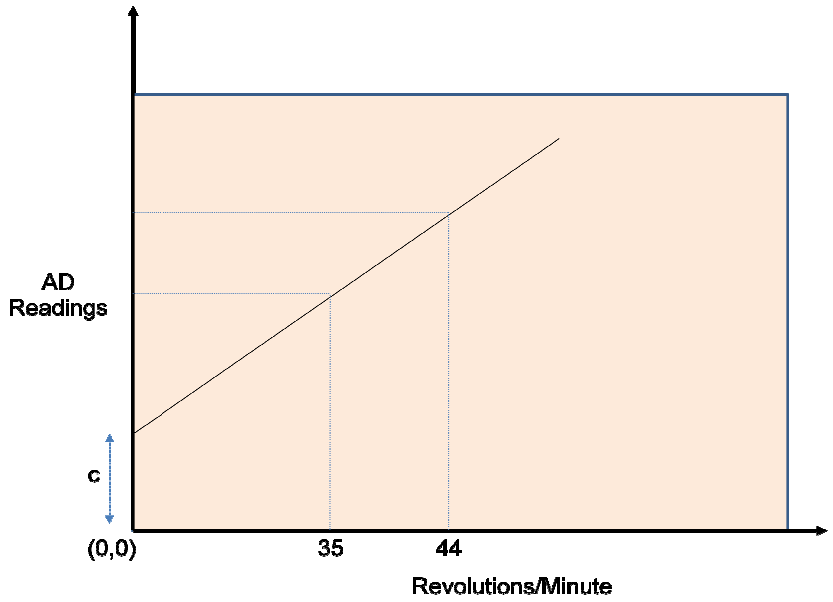

Figure 6.10. Response plot of Gyro AD readings vs the Angular rate.

We will now derive a generic equation for determining the angular change based on an AD sample. If **Y** is the A/D reading, **X** is the Angular rate (AR), **C** is the observed AD reading when the tag is stationary, **m** is the slope of the line which is calculated from the known two points.

$$AR = \frac{AD - C}{m}$$

Since we sample the angular rate sensor 40 times a second, we end up calculating the angular rate for $\frac{1}{40^{th}}$ of a second. Therefore, the current angular rate is:

$$AR = \frac{AD-C}{m} * \left(\frac{1}{40}\right) * \left(\frac{360}{60}\right) \text{ degrees/second}$$

6.3.3. Step Detection

Step detection (i.e., human step detection) is carried out with the aid of the 3D accelerometer. Detecting a step is important to calculate the relative displacement of the tag with respect to the internal reference axis.

The analysis of the A/D values returned when a person takes a step reveals that the A/D values in the vertical axis vary between a *peak value* and *bottom value.* So, the idea is to keep a lookout within the sampled A/D values for a *peak* followed by a *bottom.* The presence of this two the values, the *peak* first followed by the *bottom* would be the detection of one step.

Based on the *Application Notes* presented in [17], it is observed that the distance covered when taking a step is given by the following equation:

$$Step\ Distance\ sd = \sqrt[4]{A_{peak} - A_{bottom}} * K$$

Where, $\boldsymbol{A_{peak}}$ is the maximum acceleration measurement in a single step, $\boldsymbol{A_{bottom}}$ is the minimum acceleration measurement in a single step, **K** is a constant for unit conversion (i.e., feet or meters travelled). From the Figure 6.11, we can see that it would be possible to derive ΔX and ΔY by using simple trigonometry, since we do possess knowledge of the relative orientation of the Tag with respect to the internal reference axis.

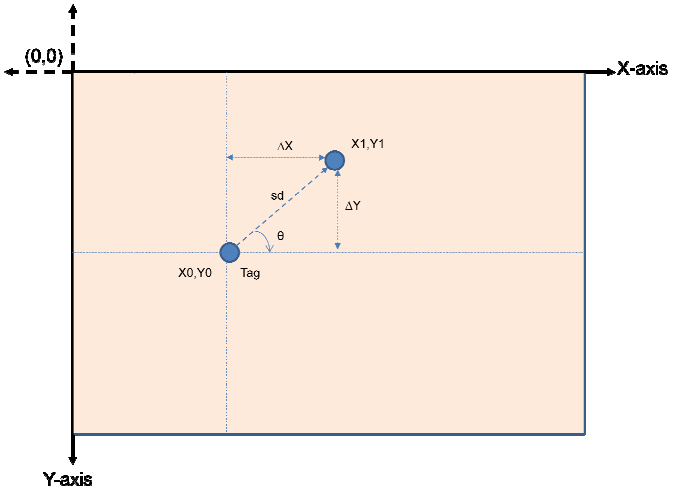

Figure 6.11. Movement of a Tag with a certain orientation

Therefore, $\Delta X = cos\,\theta * sd$ and $\Delta Y = sin\,\theta * sd$. The above sections clearly show that the three parameters, ΔX, ΔY and Δθ that are required for the prediction stage of the particle filter, can be relatively reliably derived.

6.4. Hardware Components

The hardware of the Tag, AP and BS contain two main components as shown in Figure 6.12.

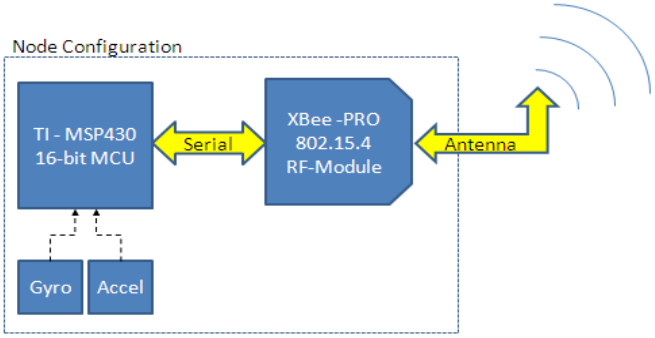

Figure 6.12. Hardware Layout of a Node

- **XBee-PRO RF module**: the RF transceiver
- **MSP430 MCU**: controls the RF transceiver and processes sensor information (in case of a tag).

6.4.1. MSP430 MCU (MSP430F1611)

The TI MSP430 is a 16-bit RISC MCU that includes a flexible clock system, using a Von-Neumann architecture. Complementing a modern CPU with modular memory-mapped analog and digital peripherals, the MSP430 offers solutions for demanding mixed-signal applications.

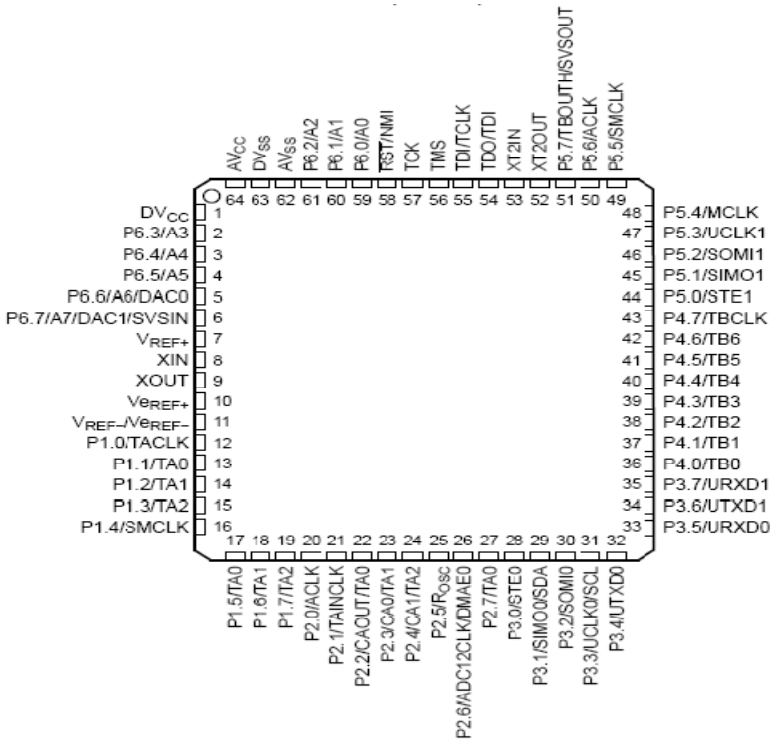

Figure 6.13. Pin layout of an MSP430

Key features of the MSP430x1xx family include:

- Ultralow-power architecture extends battery life
    - 0.1-μA RAM retention
    - 0.8-μA real-time clock mode
    - 250-μA / MIPS active
- High-performance analog ideal for precision measurement
    - 12-bit or 10-bit ADC — 200 ksps, temperature sensor, VRef
    - 12-bit dual-DAC
    - Comparator-gated timers for measuring resistive elements
    - Supply voltage supervisor
- 16-bit RISC CPU enables new applications at a fraction of the code size.
    - Large register file eliminates working file bottleneck
    - Compact core design reduces power consumption and cost
    - Optimized for modern high-level programming
    - Only 27 core instructions and seven addressing modes
    - Extensive vectored-interrupt capability
- In-system programmable Flash permits flexible code changes, field upgrades and data logging.

6.4.2. XBee-PRO RF Module

The XBee-PRO OEM RF Modules are engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor

networks. The modules require minimal power and provide reliable delivery of data between devices. Following are listed some of its features:

Data Integrity

- Indoor/Urban: up to 300’ (100 m)
- Outdoor line-of-sight: up to 1 mile (1500 m)
- Transmit Power: 100 mW (20 dBm) EIRP
- Receiver Sensitivity: -100 dBm
- RF Data Rate: 250,000 bps

Advanced Networking & Security

- Retries and Acknowledgements
- DSSS (Direct Sequence Spread Spectrum) Each direct sequence channels has over 65,000 unique network addresses available.
- Source/Destination Addressing.
- Unicast & Broadcast Communications.
- Point-to-point, point-to-multipoint and peer-to-peer topologies supported.
- Coordinator/End Device operations.

Low Power

- TX Current: 215 mA (@3.3 V)
- RX Current: 55 mA (@3.3 V)
- Power-down Current: < 10 μA

ADC and I/O line support

- Analog-to-digital conversion, Digital I/OI/O Line Passing

Easy-to-Use

- No configuration necessary for out-of box RF communications.
- Free X-CTU Software (Testing and configuration software).
- AT and API Command Modes for configuring module parameters.
- Extensive command set.
- Small form factor.

6.5. Software Components of Nodes

Logically, the software modules for a node could be divided into two categories:

- XBee-PRO RF module firmware (API Mode).
- MSP430 MCU firmware.

We will provide a detailed description of each of the above in the next subsection.

6.5.1. XBee-PRO RF module firmware (API Mode)

By default, XBee-PRO RF Modules behave as a serial cable replacement (transparent operation), where all UART data received through the DI pin is queued up for RF transmission. When the module receives an RF packet, the data is sent out the DO pin with no additional information. Inherent to *transparent operation* are the following behaviors:

- If module parameter registers are to be set or queried, a special operation is required for transitioning the module into *command mode*.
- In point-to-multipoint systems, the application sends extra data, so receiving module(s) can distinguish between data coming from different units.

As an alternative to the default *transparent operation*, API (Application Programming Interface) operating modes are available. API operation requires that communication with the module be done through a structured interface (data is communicated in frames in a defined order).

Two API modes are supported and both can be enabled using the AP (API Enable) command. Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the data is silently discarded. The API data frame format is shown in Figure 6.14.

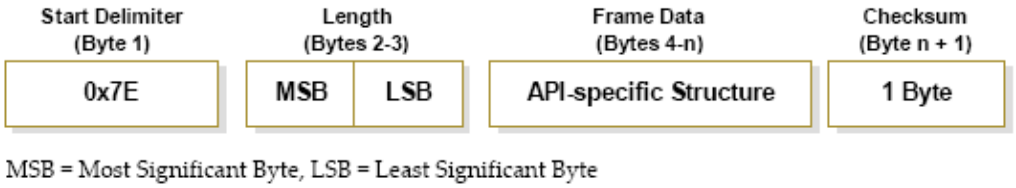

Figure 6.14. Format of an API data frame

The API specifies how commands, command responses and module status messages are sent and received from the module using a UART Data Frame.

6.5.1.1. API Types

Frame data of the UART data frame forms an API-specific structure as shown in Figure 6.15.

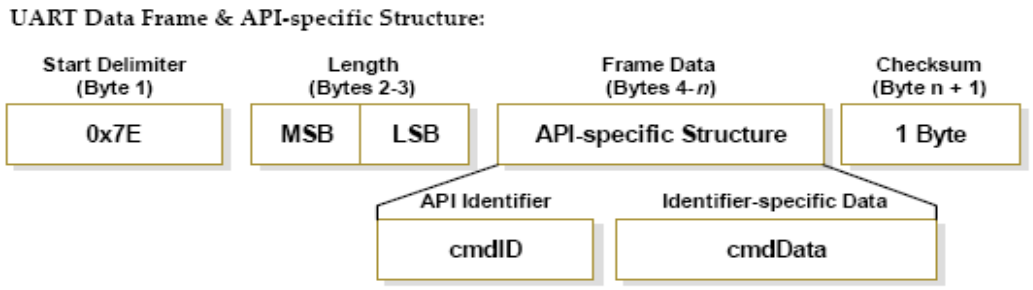

Figure 6.15. API frame format

The cmdID frame (API-identifier) indicates the type of the API messages that will be contained in the cmdData frame (Identifier-specific data). The data field values are sent in the big endian format. The following API formats have been used in the development of the nodes:

- **AT Command: (**Figure 6.16**)** The "AT Command" API type allows for module parameters to be queried or set. When using this command ID, new parameter values are applied immediately.

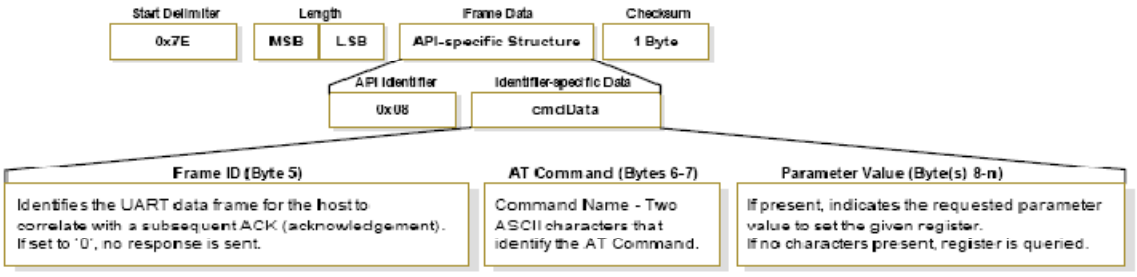

Figure 6.16. AT Command Packet Format

- **AT Command Response: (**Figure 6.17**)** In response to an AT Command message, the module will send an AT Command Response message.

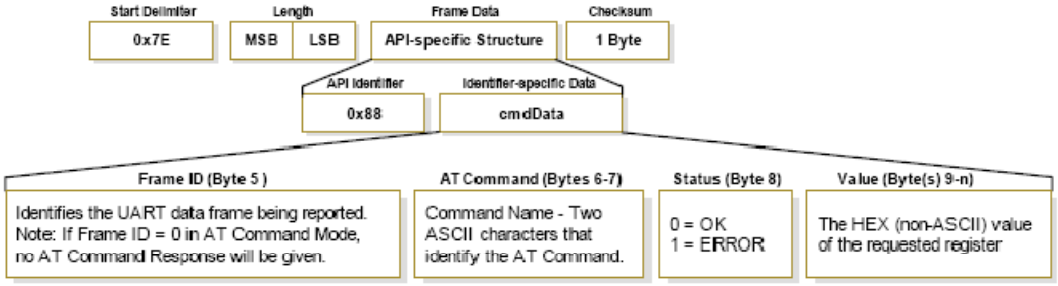

Figure 6.17. AT Command Response Packet Format

- **TX (Transmit) Request-16-bit address: (**Figure 6.18**)** A TX Request message will cause the module to send RF Data as an RF Packet.

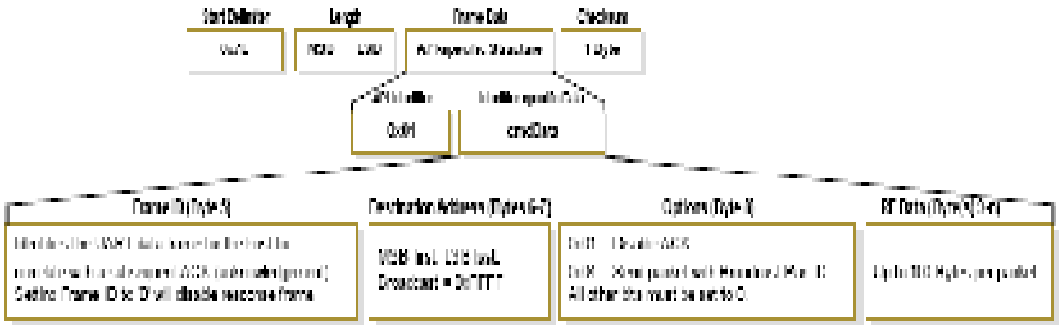

Figure 6.18. Transmission Packet Format

- **TX (Transmit) Status: (**Figure 6.19**)** When a TX Request is completed; the module sends a TX Status message. This message will indicate if the packet was transmitted successfully or if there was a failure.

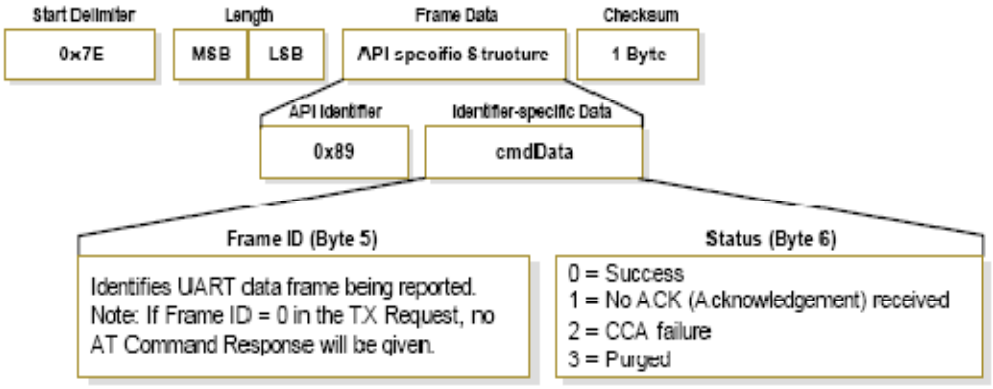

Figure 6.19. Transmission Status Packet Format

- **RX (Receive) Packet-16-bit Address: (**Figure 6.20**)** When the module receives an RF packet, it is sent out the UART using this message type.

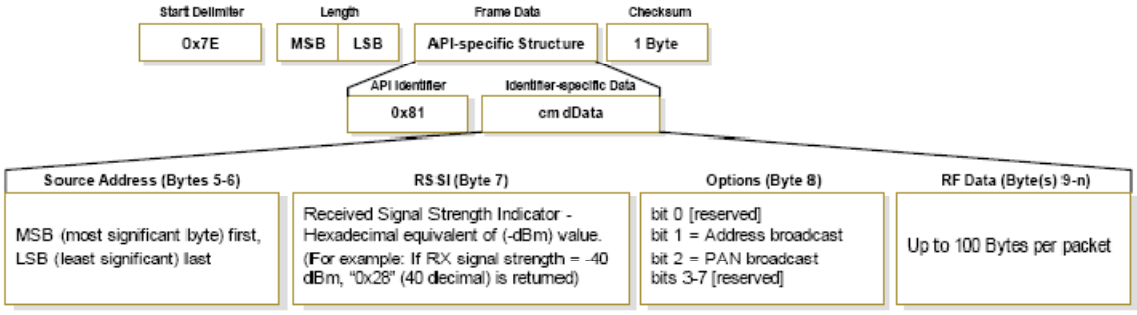

Figure 6.20. Received API Packet Format

6.5.2. MSP430 Firmware

The MSP430 firmware was developed by us. It consists of three logical layers as shown in Figure 6.21.

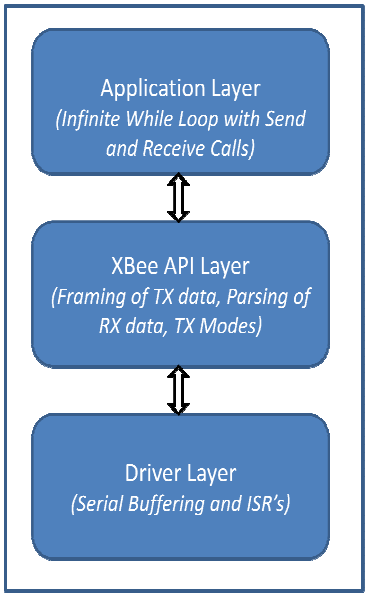

Figure 6.21. MSP430 Firmware Layers

- Application Layer
- XBee API Layer
- Driver Layer

In the subsections below we describe each of these layers in detail and identify program files associated with each of them.

6.5.2.1. Application Layer

The application layer code is a routine running in an infinite while loop, which essentially consists of API transmit and receive calls. This layer decides when to process a packet buffered in the input queue and when to transmit a packet

buffered in the output queue. The application layer is different for the three devices and in some ways defines the behavior of the particular type of node. The following files constitute the application layer:

main.c and *main.h*

Next we will discuss some subtle differences within this layer specific to each type of node:

1. **Basestation:** Based on the functionalities described above all that a BS needs to do is transmit *control packets* at periodic intervals and process (determined by the API layer) received packets and relay them to the main host.
2. **Accesspoint:** At the application level the AP seems quite similar to what the BS does but differs considerably at the API layer, with regard to the processing that is carried out.
3. **Tag:** The tag has an additional task to perform at the application layer apart from transmitting *data packets* periodically and processing buffered data. This task is the processing of the A/D converted data from the angular rate sensor and the accelerometer. The processing is carried out based on a flags set within the A/D controllers interrupt handler.

6.5.2.2. XBee API Layer

This layer is primarily responsible for taking action when receiving various packet types and providing routines to transmit and parse incoming data. This layer is common in terms of the functionalities that it provides across all the three types of

nodes. Its main role is to abstract the RF functionalities provided by the XBee RF module, thereby providing for any easy way for the MCU to interact with the RF module. As of now, it utilizes only the 16-bit addressing scheme, to address a particular node. The following files constitute the API layer:

api.c and *api.h*

Although the API layer is the same for all the three devices, it provides functionalities that are specific to each type of node. Next, we provide a brief description on the specific node type's API layer.

1. Basestation:

- The BS is always in a broadcast mode, for this purpose it utilizes the API provided by this layer to put the RF module into broadcast mode.
- On receiving a *control* or *data packet* the BS utilizes an API to transmit this received data serially to the PC, running the *localization algorithm*.

2. Tag:

- The Tag needs to unicast *data packets*. For this reason the API layer specifically provides for a function call that places the XBee RF module in a unicast mode during transmission.
- When a Tag receives a *control packet,* the API layer provides a function to update the routing table, based on the *update criteria* described in the sub-section 6.3.1.

- The API layer also provides functionality to extract the RSSI field from a *control packet* transmitted from an AP.

3. Access point:

- Apart from the broadcast/unicast and table update functionalities mentioned above, the AP utilizes an API to relay incoming *data packets* to its corresponding *NEXT_HOP* node.

6.5.2.3. Driver Layer

The *driver layer* as the name suggests, is the layer that interacts with the hardware. The layer can be divided into four modules. We will next discuss each module in detail, also listing the supporting files that correspond to each of them.

1. MCU Setup, *init.h* and *init.c.*

This module is responsible for setting up the MCU peripheral control registers based on the requirements. For example, the BS requires an additional UART to be setup to communicate with the PC, apart from the one already being used to communicate with the RF module. Therefore, these modules differ slightly across the node types.

2. Interrupt handling routines, *interrupts.h* and *interrupts.c.*

This module provides the interrupt handlers for peripherals that are invoked. These modules are types dependent, i.e., some ISR's are specific to the node type. For example, only the tag module has an A/D serving ISR, since this is the only type of device equipped with angular rate sensors and accelerometers.

3. **Serial output communication, *serialout.h* and *serialout.c*.**

   This module provides functions that buffer data destined for transmission. Functionalities include the manipulation of buffer contents, and retrieval of the state of the buffer (full or empty).

4. **Serial input communication, *serialin.h* and *serialin.c*.**

   This module implements functionalities that are required by the API layer for retrieving data from the input buffer which contains data that has been received.

CHAPTER 7
EXPERIMENTAL SETUP AND RESULTS

7.1. Experimental Setup

To validate the presented system, a test network was set up in the General Academic Classroom Building (GACB) of The University of Texas at Arlington. We will now, briefly describe some aspects of this experimental setup. Then we show some of the experimental results obtained.

7.1.1. The Environment

Our target localization area in the GACB building mainly consists of three larger corridors and two large classrooms (in addition to offices and labs) as shown in Figure 7.1.

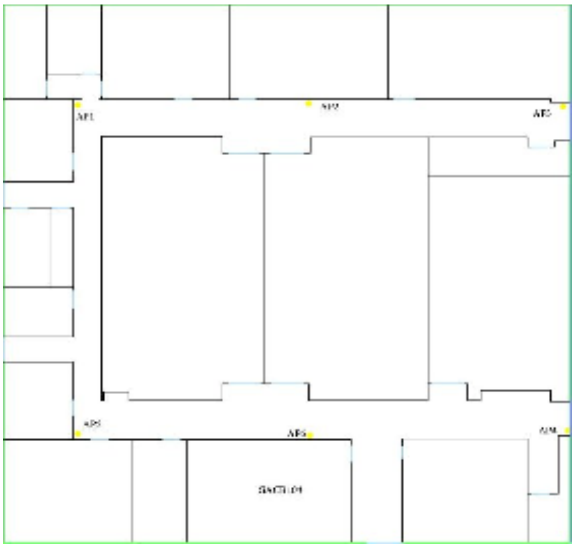

Figure 7.1. The layout of GACB (Localization Environment)

We use six AP's that have been spread somewhat evenly across the building, such that a tag is able to receive *control packets* from at least one of the AP's. The AP's are represented by yellow dots in the figure.

7.1.2. Hardware Setup

While we are working with IEEE 802.15.4 based transceivers, the proposed method should work with any other wireless technology as long as providing RSSI readings as part of their standard operation.

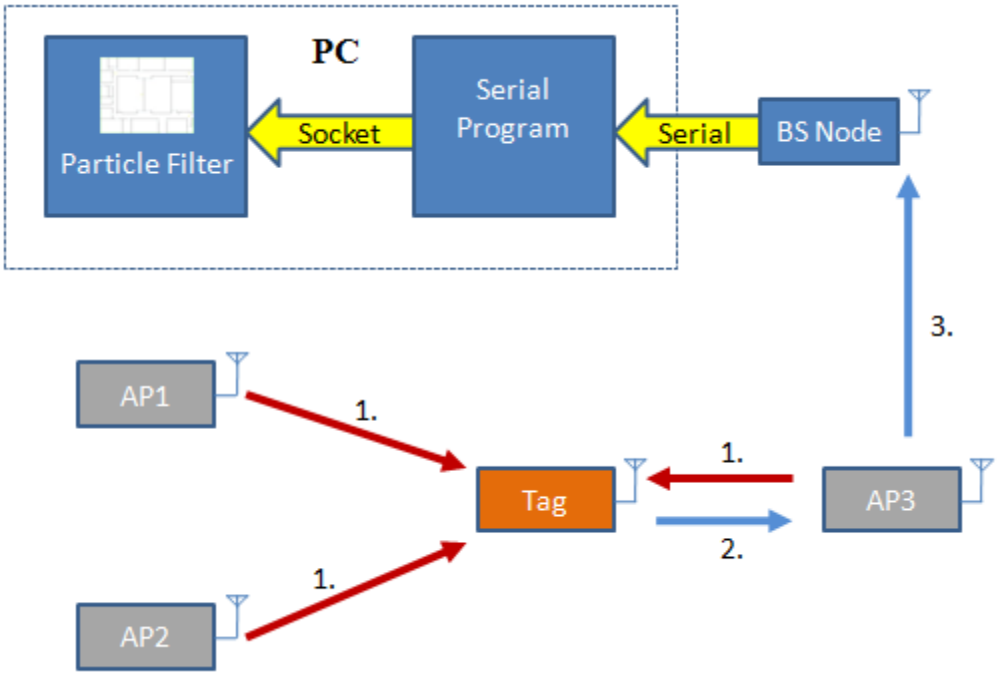

Figure 7.2. A message exchange and operation diagram of the system.

A PC is used as the host computer and thus is running the *localization algorithm (*it is serially attached to a BS for the purpose of collecting packets). AP's installed on the walls as shown in Figure 7.3, broadcast *control packets* randomly, in every half second intervals, to enable tags to determine RSSI readings.

Figure 7.3. An accesspoint installed on the wall.

An individual with a tag strapped to his/her chest, constitutes the mobile entity whose location we intend to track. The tag extracts RSSI values from received *control packets* and forwards this information together with local inertial mobility information using *data packets* to the BS.

7.1.3. Software Setup

This section discusses the various software tools utilized during the development of this project and also gives an in-depth view of the components present within the *localization engine*.

7.1.3.1. Software Requirements

We list down the various software tools used in the course of the system development:

1. **Cygwin**: [18] MS Windows based Linux environment used among others for running mspgcc and related tools.
2. **mspgcc**: [19] This is an open-source cross compiler that is specifically developed for compiling binaries for the TI MSP430 platform. It also provides utility programs for the purpose of flashing binaries onto the MCU via the JTAG interface. The package requires cygwin DLL-s.
3. **FLTK 1.1.3:** Fast Light Tool Kit [20], is an open-source C++ graphics library. This package was installed on a native Linux system running the *localization engine.*
4. **XCTU**: [15] An MS Windows based configuration software used for configuring and updating the firmware on the XBee-Pro RF modules either serially or via USB.
5. **Xfig**: [21] An object oriented graphics package for *nix systems. It is installed under Linux and is used for the purpose of generating visualization

material (e.g., building blueprints) for the *localization engine*. The so called "fig" files generated/displayed by Xfig are read by the *localization engine*; "fig" files are text files with descriptions of 2D geometric objects (such as lines, polygons and circles).

6. **Gimp**: An image manipulation framework, mainly used by us to convert and scale maps generated by Xfig.

7.1.3.2. Localization Engine

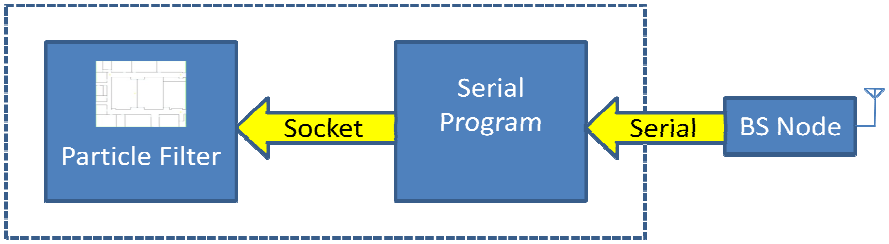

Figure 7.4. Components within the localization engine

All components of the localization engine were developed using C++. The program features a GUI using FLTK libraries, which graphically displays the location of tags (the posterior distribution generated by the Particle Filter) within the target environment. The program makes use of OOAD concepts and features a flexible design. This allows for the user to change necessary configuration parameters for the algorithm, through a single configuration file.

7.1.3.2.1. Program Initialization and Execution

In the program initialization phase, we initialize the application parameters specific to our environment. Files containing the *Measurement Model*, the

environment parameters for example: area dimensions, location of AP's etc. and the picture for the GUI are loaded, followed by the initialization of various statistical parameters like limits on various values, mean, standard deviations etc. All steps in the initialization phase are carried out by extracting information from a single configuration file, an ASCII file, which users can easily modify to setup desired parameters. We further describe the configuration file in the following section.

With regard to the particle filter running within our application, it needs a prior distribution from which to resample particle weights. When the application is running, the posterior distribution computed in the previous update cycle is used as the prior distribution. Since, our approach assigns three parameters for every particle; we initialize all the three dimensions based on uniform distribution.

For the real-time readings to serve as an input for the particle filter, we need to bring up a serial module that collects the readings from the BS and sends them over to the particle filter application via UDP sockets. The application utilizes these readings to generate a distribution for the location of the target across the environment, following which the filter performs the measurement and model update steps.

Considering that the measurement and system update are done iteratively within the *particle filter*, on the completion of the system update, we compute an estimate of the mobile nodes location and orientation. Following this the GUI is updated with new locations of each particle. The update steps are performed

approximately once every half second (the rate at which the tag transmits *data packets*).

7.1.3.2.2. Description of Configuration File

The configuration file provides for a means by which, various input parameters required by the localization algorithm can be specified. Some of the parameters are required to fine-tune the performance of the algorithm while some are needed for specifying image file paths etc. Following is a description of the configuration fields:

1. *XFIG_MAP_FILE* – This configuration parameter is used for specifying the path of the .fig file that has the layout of the localization environment.
2. *JPEG_MAP_FILE* – Specifies the path of the .jpg image file depicting the localization environment.
3. *UNITMETER* – This parameter specifies the number of Xfig units that constitute a meter.
4. *AP_COUNT* – Specifies the number of accesspoints present within the environment.
5. *AP1…APn* – This parameter specifies the list of accesspoints along with their corresponding coordinates.
6. *POW_FILE* – Specifies the path to the file containing the measurement model.
7. *ANGLE_INTERVAL* – Specifies the orientation interval within the measurement model.

8. *DISTANCE_INTERVAL* – Specifies the distance interval within the measurement model.
9. *NR_PARTICLES* – Specifies the number of particles used by the *particle filter*.
10. *NR_DIMENSIONS* – Specifies the number of dimensions associated with each particle.
11. *LIM_DIM_LOWn* – Specifies the lower limit on the value of the n^{th} dimension (n varies from 0 to NR_DIMENSIONS).
12. *LIM_DIM_UPn* - Specifies the upper limit on the value of the n^{th} dimension (n varies from 0 to NR_DIMENSIONS).
13. *DISTANCE_STDEV_MODEL* – Specifies the standard deviation for the normal function used in displacement calculations (uncertainty of the system update) of the particle filter.
14. *ANGLE_STDEV_MODEL* – Specifies the standard deviation for the normal function used in angle displacement calculations (uncertainty of the system update) of the particle filter.
15. *NR_OF_READINGS* – Specifies the number of RSSI readings that should be received before resampling of the particles is enforced.

7.2. Experiments and Results

In this section, we discuss the experiments performed followed by the results we obtained. We carry out a simplistic walkthrough through the environment depicted in Figure 7.5. The blue dot indicates the start point and the orange dot the

end point. We show four cases each denoting a certain stage during the walkthrough. We judge the performance of the algorithm by visually comparing the *expected position* of the mobile node with its *estimated position*.

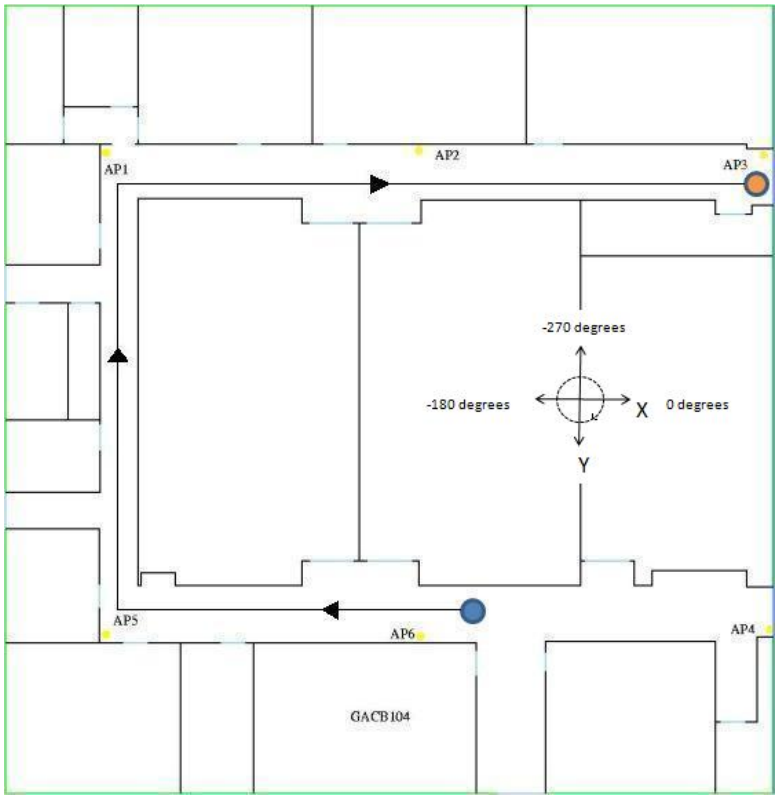

Figure 7.5. Path taken during the walkthrough.

The *expected position* is determined by the aid of carrying out an initial *dry run* along the environment. We calculate the time taken to cover these *markers* along the walkthrough. A *marker* would refer to a position during the walkthrough where there is a change in the Tag's orientation and would also include the *start* and *stop* points. This allows us to create a movement model where each entry is a tuple of the Tag's start coordinates, end coordinates, velocity of movement (assumed to be

uniform throughout), time duration to cover the distance and orientation during the displacement.

The *estimated position* is compared against the *expected position* by time-stamping each incoming report from the Tag, thereby enabling a visual plot of the *expected* and *estimated* position of the Tag with relation to time. In our work the expected and estimated positions are represented by a small *black colored* concentric circle and a large *blue colored* concentric circle respectively.

With reference to Figure 7.5, the *dry run* walkthrough within our experiment, used for determining the movement model can be described as follows:

1. Initially, the mobile node has an orientation of 0 degrees along the X-axis for around 10 seconds. Thereafter, the node turns clock-wise at the same point and has a new orientation of -180 degrees for another 10 seconds (*start marker).*
2. The node moves along the X-axis towards AP5, with a uniform velocity.
3. On reaching the point near AP5, the node stays at this point 10 seconds maintaining its orientation at -180 degrees. This will be considered as the first *marker.*
4. Thereafter, the node turns clock-wise at the same point and has a new orientation of -270 degrees for another 10 seconds.
5. The node then moves along the Y-axis towards AP1, with a uniform velocity.

6. On reaching the point near AP1, the node stays at this point 10 seconds maintaining its orientation at -270 degrees. This position will be considered as the second *marker*.
7. Thereafter, the node turns clock-wise at the same point and has a new orientation of -360 degrees for another 10 seconds.
8. The node then moves along the X-axis towards AP3, with a uniform velocity and comes to a stop at the end of the corridor (*stop marker)*.

The movement model described is utilized to determine the expected position of the mobile node at any time given *t* from the start. On bringing up the localization application, it is passed with real-time, time-stamped data. Based on the time-stamp of the data, the application plots the expected position and simultaneously plots the nodes estimated position.

The position of a node is estimated by extracting location based information from the posterior distribution of the particles generated by the filter. Considering, the work carried out in [4], the weight of each particle is computed with respect to the distance from all other particles. The weight of a particle with respect to another is derived taking the inverse of the distance between them. Such weights are computed with respect to all other particles and the sum of these weights determine the final weight of the particle. The particle with the maximum weight is considered to be the estimated position of the mobile node. We now describe a set of cases during the walkthrough:

Case 1: At the start as shown in Figure 7.6, the *expected* position is represented by the concentric black circles that overlap the blue *start* dot. Beside it, with a fair amount of accuracy is the *estimated* position of the mobile node is represented by the concentric blue circles.

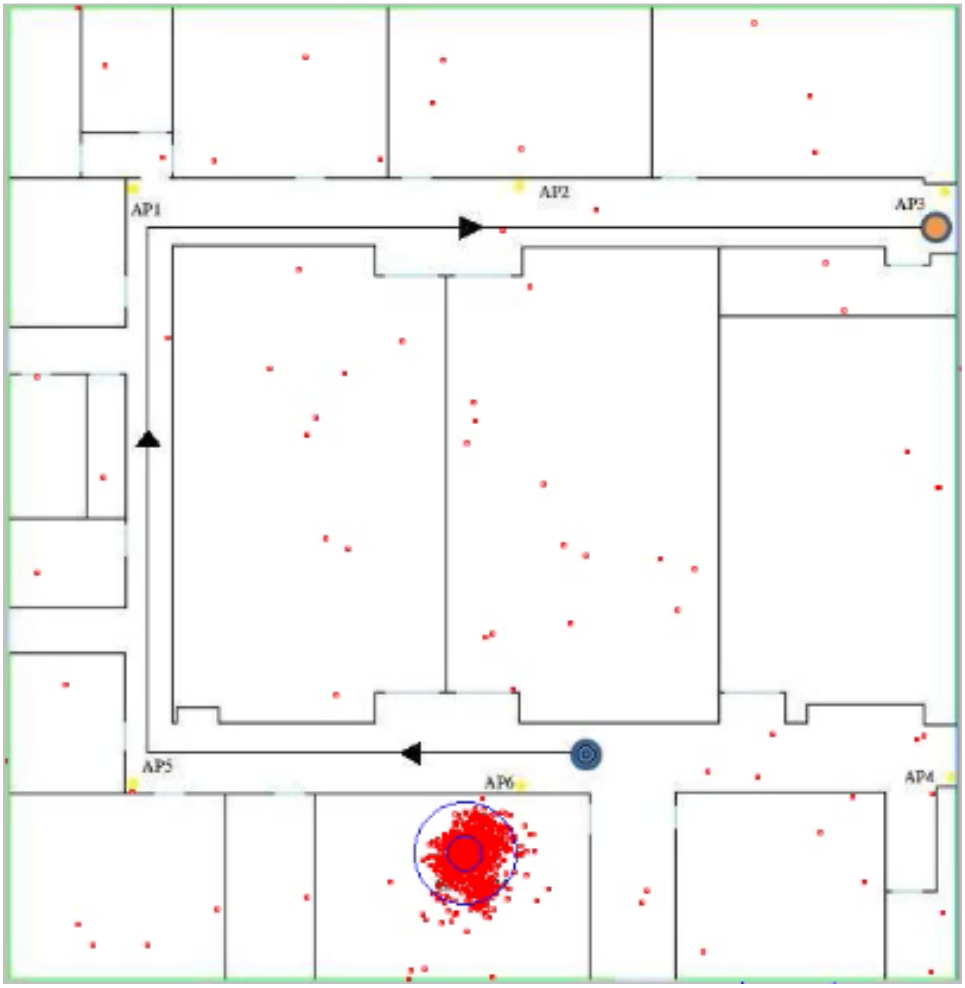

Figure 7.6. Estimated position at the start.

Case 2: Once the node starts to move along the path it reaches the first *marker* towards the end of the corridor where it changes its replace orientation from -180 degrees to -270 degrees, as shown in Figure 7.7. Since, the node remains stationary here, the estimated position is accurate to the order of a meter.

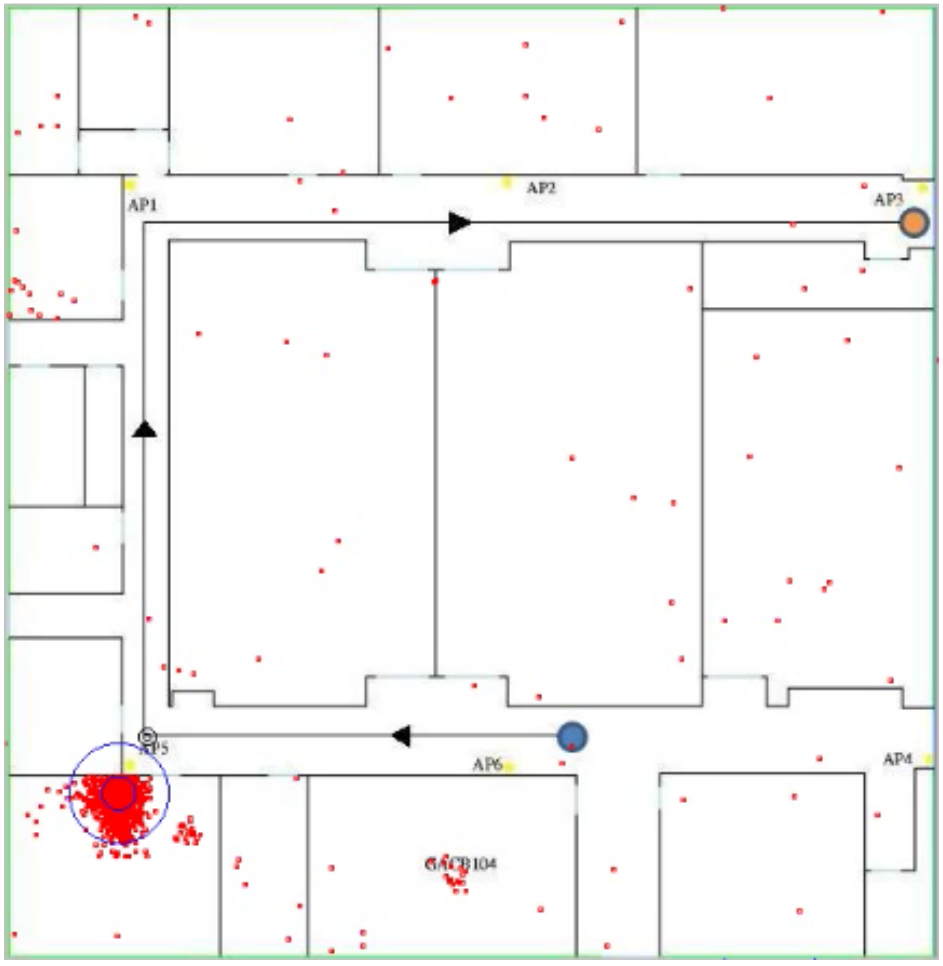

Figure 7.7. Movement of node from the *start* to the first *marker* near AP5.

Case 3: Once the node starts to move along the path from the *first* to the *second marker* we show a case where the mobile node is midway during this transition. The estimation is fairly accurate during movement as can be seen from Figure 7.8.

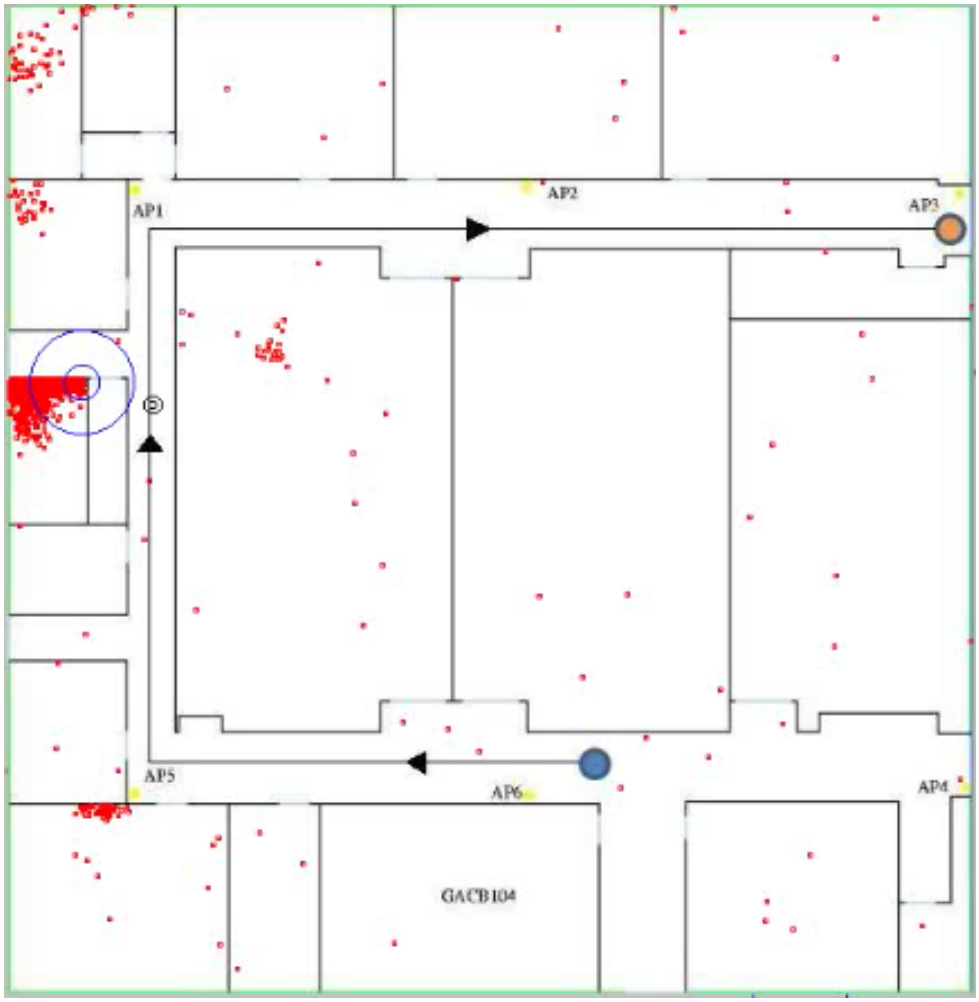

Figure 7.8. Movement from *first* to the *second marker.*

Case 4: On reaching the *second marker* (near AP1) towards the end of the corridor, the node changes its relative orientation from -270 degrees to -360 degrees, as shown in Figure 7.9. Again, since the node remains stationary here, the estimated position is accurate to the order of a meter.

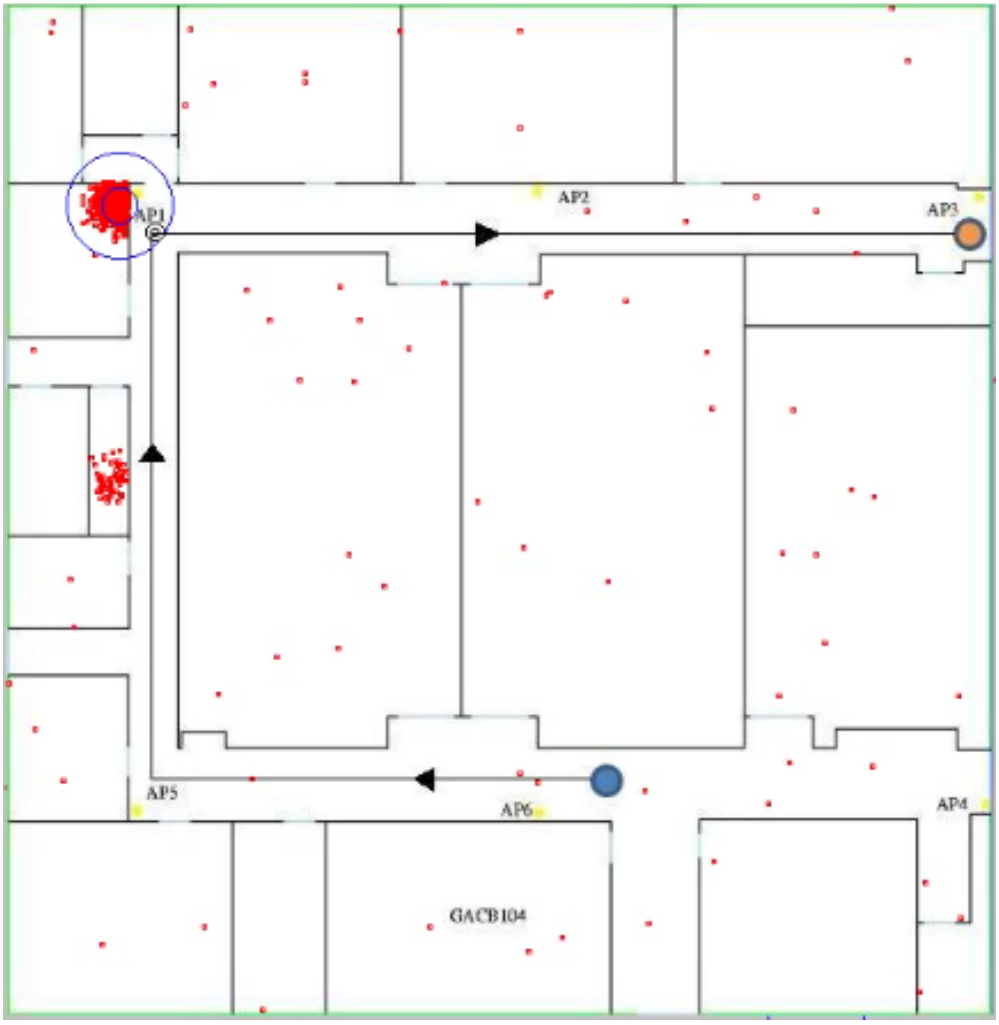

Figure 7.9. Movement of node from the *first* to the *second marker* near AP1.

Case 5: Once the node starts to move along the path from the *second* to the *stop marker* we show a case where the mobile node is midway during this transition. The estimation is fairly accurate during movement as can be seen from Figure 7.10.

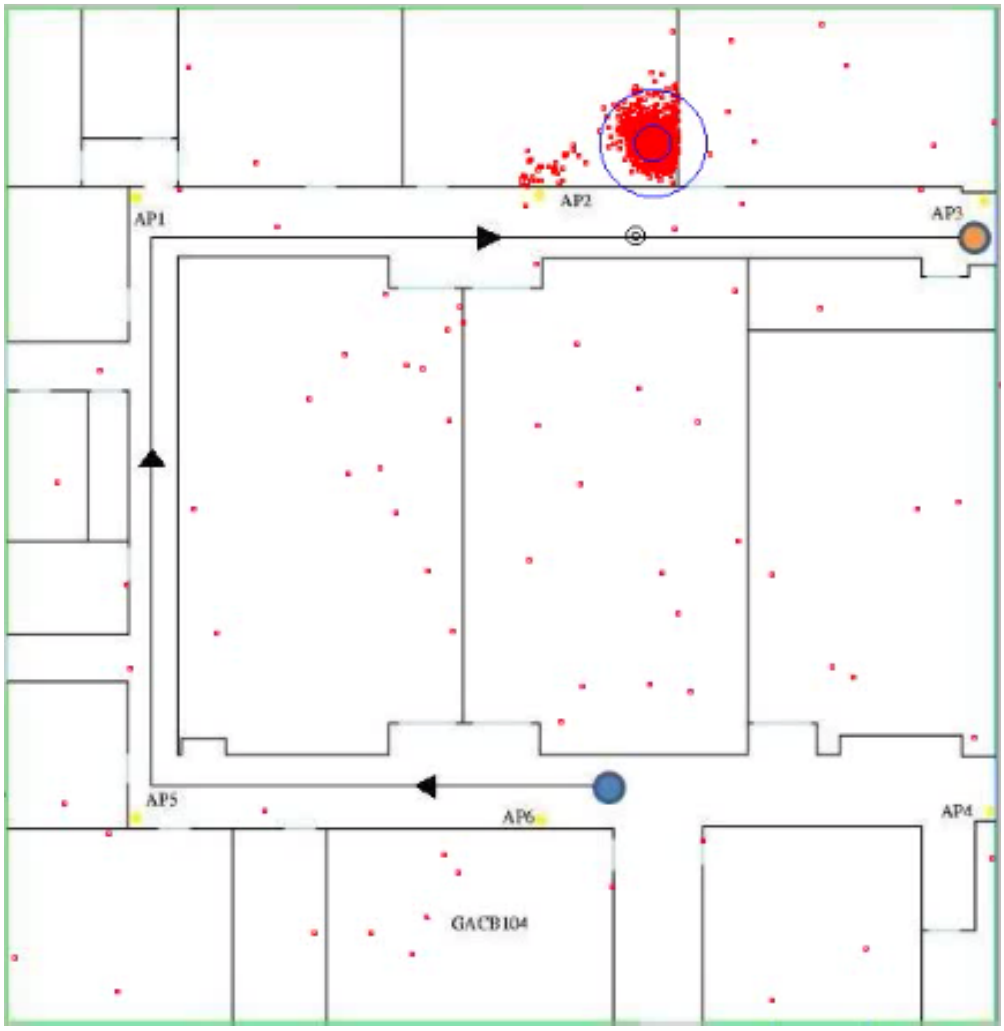

Figure 7.10. Movement from *second* to the *stop marker.*

Case 6: On reaching the *stop marker* (near AP3) towards the end of the corridor, the node comes to a stop as shown in Figure 7.11.

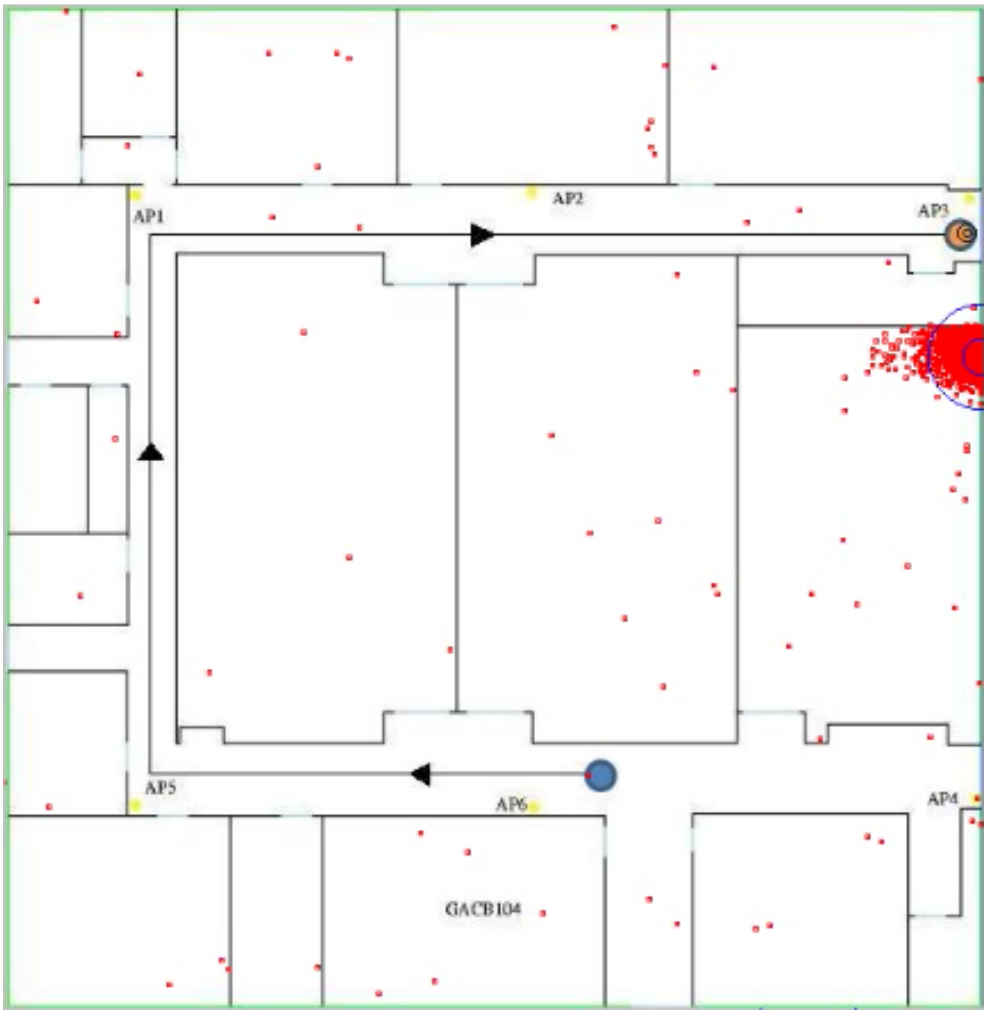

Figure 7.11. Estimation of the nodes position at the *stop marker*.

7.3. Implementation Issues Faced

In this subsection we list some of the implementation difficulties that we had to overcome:

- After the execution of an AT command, a call to a pause function (200 ms) within the MCU is required.
- Baud rate settings on the MCU, XBee and Serial program should match.
- Occasional malfunction of XBee RF modules. Recovery Steps:
    - Bring up XCTU.
    - Uncheck API mode.
    - Select the Device Type as XBP24 and firmware version as 10A5.
    - Click on Write, and wait for a RESET Window to pop-up.
    - Immediately, hit the reset button on the evaluation board.
    - Select the PC Setting Baud-rate as 9600.
    - Under Modem Configuration, configure firmware for API mode 1 and Baud-rate as 57600 bps.
    - Click on Write, “may have to” wait for RESET Window to pop-up.

CHAPTER 8
CONCLUSIONS AND FUTURE WORK

Many methods have been developed, and are being developed, to build real time location systems. These efforts are motivated by an exploding market for such systems, as they can increase productivity, reduce costs, and ensure security. Location awareness for wireless devices has a wide range of applications including personal navigation, security systems, and health care, and plays an important role in the future of pervasive computing. Consequently, this establishes a need for a simple and inexpensive approach for localization to facilitate location aware applications.

In this thesis, we have presented a probabilistic approach to localization using a custom IEEE 802.15.4 based wireless infrastructure. The availability of inexpensive, low-power microcontrollers, system on chip and radio on chip transceivers creates an even more compelling reason to develop cost-effective and architecturally flexible wireless devices (nodes) which can be specifically used for the purpose of *tracking*. The proposed system requires sensor readings pertaining to RSSI, change in orientation and change in displacement. RSSI is available in almost all digital radio transceivers; orientation and displacement based sensory information can be easily gathered by outfitting a wireless device with off-the-shelf angular rate and acceleration sensors. We have demonstrated the performance of the proposed

approach through real-time experiments performed in an indoor environment with four access points.

These experiments demonstrate that our system was successful in tracking a user with reasonable precision throughout the target area. This proves the potential of our approach in enabling location aware applications for wireless devices with the aid of an existing wireless infrastructure.

8.1. Future Work

With regard to future work within this project, it can be broadly categorized into two areas. The first area includes improvements with regard to incorporating new features both within the hardware and the localization algorithm. Utilizing low power features of the MSP430 microcontroller, developing the Tag for detecting movement sideways and backwards would be hardware related improvements while, including ToA related data from the sensor nodes would result in an improvement within the localization algorithm.

Lastly, there is a need to fine-tune many approximations and assumptions like standard-deviation in distance and orientation required by the system model, through more research. Also, consideration of more variables or factors like obstacles will aid in the creation of a better distance/orientation matrix (measurement model).

REFERENCES

1. **Ethan Leland, Kipp Bradford, Odest Chadwicke Jenkins.** *Robot Localization and Control.*

2. **Kamal, Ibrahim.** WFR, a Dead Reckoning Robot. *http://www.ikalogic.com/wfr.php.* [Online]

3. **Markus Anwander, Gerald Wagenknecht, Torsten Braun.** *Energy-efficient Management of HeterogeneousWireless Sensor.*

4. **Vinay Seshadri, M.S.** *A Bayesian Sampling Approach to Indoor Localization Of Wireless Devices Using Received Signal Strength Indication.* Arlington : UTA, 2003.

5. *Location accuracy of an UWB localization system in a multi-path environment.* **Michael Tüchler, Volker Schwarz, Alexander Huber.** Zurich : s.n., September 2005.

6. *Shooter Localization and Weapon Classification with Soldier-Wearable Networked Sensors.* **Peter Volgyesi, Gyorgy Balogh, Andras Nadas, Christopher B. Nash, Akos Ledeczi.** San Juan : s.n.

7. *Physical Authentication through Localization in Wireless Local Area Networks.* **Vishal Bhargava, Mihail L. Sichitiu.** Raleigh, NC : s.n., 2005.

8. **Nanotron.** Real Time Location Systems (RTLS). *http://www.nanotron.com/EN/SU_docs_white_papers.php.* [Online] 2007.

9. *The use of Multi-stage Sampling Schemes in Monte Carlo Computations.* **Marshall, W.** New York : s.n., 1956.

10. **Rekleitis, Ioannis M.** *A Particle Filter Tutorial For Mobile Robot Localization.*

11. **Kaijen Hsiao, Henry de Plinval-Salgues, Jason Miller.** *Particle Filters and Their Applications.*

12. **Society, IEEE Computer.** *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks.* s.l. : IEEE, 2007.

13. **Digi.** *XBee™/XBee-PRO™ OEM RF Modules Product Manual v1.06.* s.l. : MaxStream, Inc.

14. **Instruments, Texas.** *MSP430x1xx Family User's Guide.* s.l. : Texas Instruments.

15. **Digi.** *Quick Start Guide - XBee-PRO PKG-U™ USB RF Modem.* s.l. : Maxstream.

16. **Analog Devices.** ADXRS300 ±300°/s Single Chip Yaw Rate Gyro with Signal Conditioning. *http://www.analog.com/static/imported-files/data_sheets/ADXRS300.pdf.* [Online] Analog Devices.

17. **Harvey Weinberg, Analog Devices.** *Using the ADXL202 in Pedometer and Personal Navigation Applications.*

18. **Hunt, Harold L.** *Cygwin/X User's Guide.*

19. **Underwood, Steve.** *mspgcc A port of the GNU tools to the Texas Instruments MSP430 microcontrollers.*

20. **Michael Sweet, Craig P. Earls, Matthias Melcher, and Bill Spitzak.** *FLTK 1.1.8 Programming Manual.*

21. **Hiscocks, Peter.** *Using XFig.* s.l. : Ryerson Polytechnic University, August 17, 2001.

22. *A Bayesian Sampling Approach to In-door Localization of Wireless Devices Using Received Signal Strength Indication.* **Vinay Seshadri, Gergely Zaruba, Manfred Huber.** Arlington : s.n., 2005.

23. *Attack Detection in Wireless Localization.* **Yingying Chen, Wade Trappe, Richard P. Martin.** 2007.

BIOGRAPHICAL INFORMATION

Vijay Vasant Dixit was born August 1984, India. He received his Bachelor of Engineering in Electronics and Communication from Visvesvaraya Technological University, India in June 2005.

Prior to pursuing his Masters he worked at MindTree Consulting, Bangalore, India in the field of storage area networks for a period of one and a half years. In Spring of 2007, he started his graduate studies in computer engineering. He received his Masters in Computer Engineering from the University of Texas at Arlington, in December 2008. His research interests include operating systems, embedded systems, and computer networks.