



PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO



Localization Heuristic in Mobile Wireless Networks

Thais Webber, Adelcio Biazi, Thomas Volpato de Oliveira,
Matheus Senna de Oliveira, Plauto de Abreu Neto,
Letícia B. Poehls, César Marcon

Technical Report N° 080

Porto Alegre, Setembro de 2014.

Localization Heuristic in Mobile Wireless Networks

ABSTRACT

The improvement of WSN technologies boosts the development of several new applications. For many of these applications, the use of not only static nodes, but also mobile nodes - with the capability to change their position over time, can greatly expand the node coverage area, reducing the required number of nodes and the network costs. The movement of a sensor node may impose extra challenges for the WSN, such as the necessity to know the current node location as well as the ability of dynamically connecting to other nodes. Localization techniques have benefits and limitations, but by combining some of them in a proper way, it may be possible to develop more reliable and flexible node localization system, that can take advantage of the flexibility of an inertial technique, for example, and the simplicity of anchor node based techniques. This work proposes a localization heuristic that combines an anchor-based technique and the inertial measurement technique, exploring aspects regarding precision, reliability and flexibility.

Keywords: Wireless Sensor Networks, Mobility, Localization, Anchor-based technique, Inertial Measurement, Simulation.

Summary

1	Introduction.....	5
1.1	Objectives	6
1.2	Organization	6
2	Theoretical background.....	8
2.1	Anchor Point Localization Techniques	8
2.2	Connectivity	8
2.2.1	Received Signal Strength (RSS)	9
2.2.2	One-Way Propagation Time Measurement (OWPT)	10
2.2.3	Time-Difference of Arrival (TDOA).....	11
2.2.4	Angle Of Arrival (AOA)	11
2.3	Inertial Navigation System	12
2.3.1	Accelerometer	14
2.3.2	Gyroscope	14
2.3.3	Combining inertial navigation and other localization techniques	15
2.4	Kalman filters	16
2.4.1	Kalman filter in continuous time.....	17
2.4.2	Kalman filter in discrete time.....	19
2.4.3	Kalman filter for IMU signals in MatLab	20
3	Simulation and other tools	22
3.1	Discrete Event Simulation.....	22
3.2	WiNeS.....	22
3.3	OMNet++	22
3.4	Random-based mobility models for nodes	23
3.4.1	Random Waypoint model	23
3.4.2	Random Direction model	24

3.4.3	Manhattan model.....	25
3.5	Considerations about simulators and WSN evaluation	26
3.5.1	Sensor nodes characteristics.....	26
3.5.2	Nodes energy consumption	27
3.5.3	Energy models for simulation	27
4	Proposed localization technique	29
4.1	Algorithm description	29
4.2	Preliminar hardware tests	31
4.2.1	The proposed test bed.....	31
4.2.2	Procedures for the Arduino platform tests	34
4.2.3	Magnetometer tests.....	35
4.3	Matlab and Arduino experiments	36
4.3.1	Filters and algorithms	36
4.3.2	Results obtained with the IMU data and filter	37
4.3.3	Comments on the Arduino experiments	38
4.4	Localization simulation scenario using WiNeS.....	39
4.4.1	Module implementation on the WiNeS environment	39
4.4.2	Class diagram.....	41
4.4.3	Algorithms description	42
4.4.4	Output Logs	43
5	Final considerations.....	45
6	References	46
	Appendix I - Arduino platform tests	50

1 INTRODUCTION

A Wireless Sensor Network (WSN) typically consists of hundreds or thousands of distributed nodes, each one equipped with a wireless communication module, a microcontroller, a power source and sensors with the ability to monitor parameters like temperature, pressure, vibration, movement and many other physical characteristics (1). Additionally, WSN technologies are easy to deploy, low-cost and high scalable. However, this versatility comes with inherent limitations, such as low storage capacity, low communication bandwidth, and limited battery energy. Due to the limited battery energy being usually the primary limitation, components with reduced energy consumptions are always desirable.

The improvement of WSN technologies boosts the development of several new applications. For many of these applications, the use of not only static nodes, but also mobile nodes - with the capability to change their position over time, can greatly expand the node coverage area, reducing the required number of nodes and the network costs (2). Moreover, there are applications such as vehicle tracking, where the primary objective of the sensor nodes is to keep track of its position. The movement of a sensor node may impose extra challenges for WSN, such as the necessity to know the node location as well as the ability of dynamically connecting to other nodes. When a sensor node is able to move around, its localization is essential information, since having sensed data such as temperature, humidity and pressure without the location knowledge may be useless (3).

Many techniques can be employed in order to determine the sensor node (i.e. node of interest) position in space. Usually, these techniques require the prior knowledge of the location of a few special nodes (i.e. anchor nodes) combined with the ability to calculate the distances between the node of interest and these anchor nodes. There are quite a few ways to estimate the distance between two nodes, from simple connectivity or Received Signal Strength (RSS) up to more sophisticated like time or angle of arrival, each approach with its advantages and limitations (3) (4).

Besides anchor node techniques being able to achieve quite good results, all of them require that the node of interest is in communication range of one or a group of anchor nodes, i.e. if the node is outside communication range, it loses its ability to know its position. This problem can be solved with the use of a GPS system, although, it is usually unpractical alternative since GPS modules are normally expensive and consume a great amount of energy, i.e. two undesirable characteristics for WSNs. Moreover, for indoor

applications, the GPS may not work properly, because GPS signals are too weak to penetrate buildings or go underground (3).

Another technology that can be used to estimate position without the need of reference points is known as inertial measurement. This technology is based on combining inertial measurement sensors such as gyroscopes, accelerometers and magnetometers in order to estimate acceleration, angular velocity and orientation, respectively, and from that, it integrates linear velocity and finally integrates position. Inertial Measurement Units (IMUs) are cheaper than GPS, and can work in most diverse environments such, indoor, outdoor, dense forests or even underground. The biggest challenge of this technique is that it is not well suited to estimate position. The nature of the double integral needed to derive the positions from acceleration values introduces a lot of error over time. By itself, this technique is only useful when the time interval are very short and error do not sum up to unbearable values.

Each of these localization techniques has its benefits and limitations, but by combining some of them in the proper way, it may be possible to develop more reliable and flexible node localization system, that can take advantage of the flexibility of the inertial technique, the precision of the GPS system, and the simplicity of the anchor node based techniques.

1.1 Objectives

This technical report discusses a localization algorithm that combines the anchor node based technique and the inertial measurement technique, exploring aspects regarding precision, reliability and flexibility. In addition, this work approaches the development of a test bed containing multiple nodes equipped with a micro-controller, an IMU [4] and an IEEE 802.15.4 [5] radio transceiver.

1.2 Organization

This work is structured as follows. Section 2 introduces the theoretical background focusing on anchor point localization techniques, connectivity, inertial navigation system, and Kalman filters. Section 3 exploits concepts of simulation such as discrete event simulation, and tools (frameworks WiNeS and OMNet++). In addition it is introduced random-based mobility for simulation as well as some considerations about simulators and WSN evaluation. Section 4 describes the proposed localization technique focusing on

algorithm description, preliminar hardware tests, experiments on Matlab and Arduino. Using WiNeS a localization simulation scenario is also proposed and modeled. Section 5 presents final considerations about this ongoing research.

2 THEORETICAL BACKGROUND

2.1 Anchor Point Localization Techniques

Anchor point localization techniques are by far the most usual way to estimate the location of a WSN node (5), they are all “relative positioning” techniques; this means it will provide the location of a node in relation to other nodes in the system. All these techniques require the previous knowledge of the positions of one or more of these specific sensor nodes called anchor nodes. The position of the anchor nodes must be acquired previously by other techniques such as GPS (for mobile anchors) or by simply installing the anchor node in a known coordinate position.

With the information of the position of the anchor nodes is just a matter of estimating the distance of the node of interest in relation to these anchor nodes. Few techniques can be used to estimate this relative position. We briefly describe three techniques: Connectivity, Received Signal Strength (RSS), One-Way propagation time (OWPT) and Time Difference of Arrival (TDOA). We also comment the Angle of Arrival (AOA) technique that measures angles instead of distance.

2.2 Connectivity

The connectivity is probably the simplest technique that can be used for node localization. In its simpler form, if a node of interest is in connectivity range of an anchor node, the node of interest location can be considered as being equal to the anchor node location. The main advantage of this technique is its simplicity, any WSN running any protocol should be able to perform it, and it only requires a single anchor node to work properly. For example, if we have an application that only requires knowing if a node is inside a room, we could use a single anchor node with a range limitation confined inside this room. As soon as any node of interest enters the room, it will be able to connect with the anchor node and when it does it, the node position can be defined as “inside the room”.

Unfortunately, this simplicity implies some hard limitations. For instance, the localization precision is inversely proportional to the nodes communication range, when the nodes communication range increases, nodes from farther away are able to connect to the anchor nodes, therefore the location precision decreases. In our example, if we use a

node with enough power to transmit through multiple rooms, we would be unable to define in which room the node of interest is located. Another limitation is the fact that if a node is in connection range of multiple anchor nodes at the same time, this technique is unable to choose in which location the node actually is. To solve this last problem, a common approach is to calculate the centroid of the anchor nodes that are in connection range of the node of interest, and define that point as the node location.

2.2.1 Received Signal Strength (RSS)

RSS techniques estimate the distance between two nodes by measuring the difference in the signal strength from the transmitter to the receiver. The capability of measuring the signal strength must be provided by the wireless device; fortunately, most of them come with this functionality.

The wireless signal strength received by a sensor node is a monotonic decreasing function of their distance (5), and they are commonly related to each other according to the following function:

$$P_r(d) = P_0(d_0) - 10n_p \log_{10} \left(\frac{d}{d_0} \right) + X_\sigma$$

where $P_0(d_0)$ is a power reference in dB milliwatts at a distance reference d_0 from the transmitter, n_p is the path loss exponent that measures the rate at which the received signal strength decreases with distance and X_σ is a zero mean Gaussian distributed random variable with standard deviation σ and it accounts for the random effect caused by shadowing. The n_p and σ are both environment dependent. By using this model and knowing *a priori* (by measuring) the required system parameters, the inter-sensor distance can be estimated from the RSS measurements, and location algorithms applied over these distances can define the node relative position.

This technique provides some advantages over the simpler connectivity technique, since with a fixed position provided by a single anchor node, the technique may define a radius of possible positions instead of an area of possible positions. In the previous example, this could mean that a single anchor node can now determine if a node of interest is in one of multiple rooms, the only pre-requisite is that these rooms should be at distinct radius distance from the anchor node position (an example of this scenario can be seen in Figure 1). In addition, by using multiple anchor nodes within communication range

of the node of interest, trilateration [19] techniques can be used to identify a position with much more accuracy than the centroid approach used in the connectivity technique.

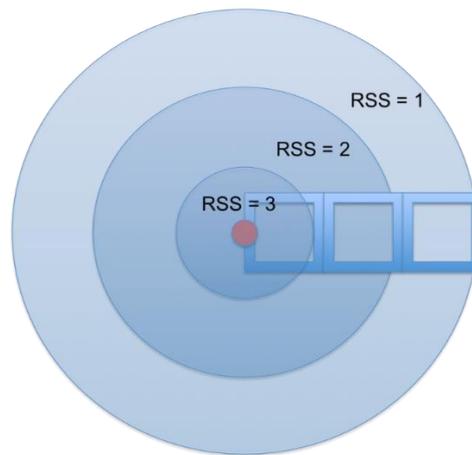


Figure 1 - RSS based room localization.

Nevertheless, this technique has serious problems for indoor localization since objects, walls, persons and even humidity, can attenuate the signal strength causing wrong distance estimations. In the example being used, the sensor signal strength from a node in the room nearest to the anchor node can be misunderstood as a sensor in the second room, because there was some interference between the node of interest and the anchor node at the time of measurement and the RSS reading dropped from 3 to 2.

2.2.2 One-Way Propagation Time Measurement (OWPT)

The principle of OWPT is to measure the signal time difference from transmitter to the receiver. Using this time difference and the signal propagation speed in the medium, the distance between the transmitter and the receiver can be calculated (5).

The great advantage of this technique is the simplicity to implement an algorithm to compute the formula, and the obstacles in the transmitting path do not significantly affect the propagation time. On the other hand, this approach requires accurate synchronization between the local time at the transmitter and the local time at the receiver. Any difference between the two local times may cause huge error in the distance calculation. Since the RF signal travels at the speed of light, a synchronization error as small as 1ns can cause an error in distance of 0.3 m (5). To achieve such synchronization requirements, the nodes require a highly accurate clock or a sophisticated synchronization algorithm, increasing the node cost and the energy required above what is usually desirable for a WSN.

2.2.3 Time-Difference of Arrival (TDOA)

The TDOA is an improvement of OWPT, since it overcomes the tight requirement of a high-level synchronization between the clocks of the communicating nodes due to the high propagation speed of the RF signal, by taking advantage of the fact that sound waves travel much slower than the RF signals. Therefore, by combining RF signals and Ultrasound waves, the synchronization requirements drop considerably, allowing the use of much simpler synchronization schemes. In the TDOA approach, two signals are simultaneously generated by the transmitter: an RF signal that travels at speed of light and reach the receiver almost instantaneously, and an ultrasonic wave that will take considerably more time to reach the receiver. The algorithm then calculates the time difference between the RF signal and the sound wave, and knowing the speed of light and sound, it calculates the distance between the transmitter and the receiver. Besides this approach requiring much less synchronization, it does require an extra hardware to produce and detect sound waves. In addition, a noisy environment like a factory floor can seriously affect sound waves, turning this technique sometimes unpractical.

2.2.4 Angle Of Arrival (AOA)

AOA technique estimates the position by the knowledge of the direction from where signals are being generated. There are multiple ways to determine the direction, a common one is known as *beamforming* and it is based on the anisotropy in the reception pattern of an antenna (5). Figure 2 shows the beam pattern of a typical anisotropic antenna. In order to estimate the direction from where the transmitted signal is arriving, the receiver must rotate (usually electronically) its antenna to detect the specific direction in which the signal strength is maximum, this will be the direction from where the signal is being generated and the node is located. This technique can be very useful when the number of anchor nodes is reduced, since with only two anchor nodes in range of the node of interest is enough to identify its position in space, and the math involved is considerably simple (lines intersection point).

Using this technique, some of the limitations involved are the difficulty to deal with signal strength variations and multipath. If the transmitted signal varies in amplitude (strength), this system cannot differentiate this behavior from the loss caused by the anisotropy of the receiver antenna. Thus, the system is highly sensible to multipath, since a reflection of the main transmission can be mistaken as the transmission itself, and by

definition, the direction of the reflection will generally be completely distinct from the source of the original transmission.

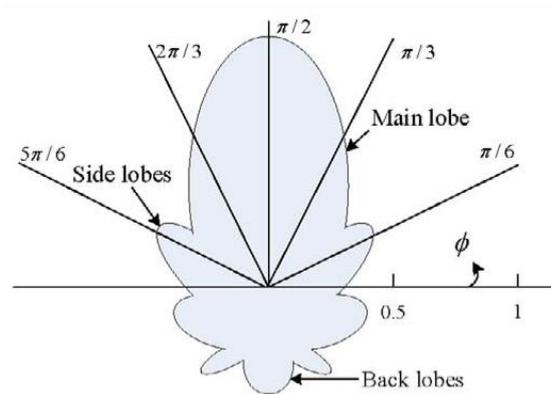


Figure 2 - Beam pattern of a typical anisotropic antenna.

2.3 Inertial Navigation System

An Inertial Navigation System (INS) estimates location using inertial measurements components, such as gyroscopes, accelerometers and magnetometers. By taking advantage of the kinetic laws of movement, one can calculate the location of an object by simply double integrating the acceleration acting on it over time.

The following formula uses the first integral of acceleration to compute the linear velocity component:

$$v_i = \int_{t_{i-1}}^{t_i} a \, dt + v_0$$

or

$$v_i = v_0 + a_{i-1} \times (t_i - t_{i-1})$$

where t_i is the time where current acceleration sample was measured, and t_{i-1} is the time when the previous acceleration measure was taken, and v_0 is the initial condition of the velocity for the time interval, i.e. the previous node velocity v_{i-1} , thus:

$$v_i = v_{i-1} + a_{i-1} \times (t_i - t_{i-1})$$

with the velocity function over time, one may integrate once again to have the position:

$$p_i = \int_{t_{i-1}}^{t_i} (v_{i-1} + a_{i-1} \times (t_i - t_{i-1})) \, dt + p_0$$

$$p_i = p_0 + v_{i-1} \times (t_i - t_{i-1}) + \frac{a_{i-1} \times (t_i - t_{i-1})^2}{2}$$

where p_0 is the initial condition of p for the time interval, i.e. the previous node position p_{i-1} , thus:

$$p_i = p_{i-1} + v_{i-1} \times (t_i - t_{i-1}) + \frac{a \times (t_i - t_{i-1})^2}{2}$$

Figure 3 describes a graphical representation of the numerical integration approach, taking the initial value as the reference value to calculate the area below the curve for each sample interval.

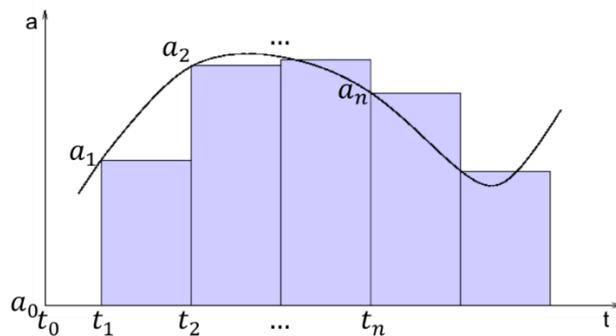


Figure 3 - Graphical representation of the numerical integration.

The position coordinates extracted from accelerometer data are in respect to its own axis orientation, since this axis can change due to a rotation, this approach is not enough to define the direction where the system axis itself is oriented. That is the purpose of using gyroscope. By knowing the initial orientation of the system, the gyroscope will keep track of its current orientation.

The gyroscope data is usually provided in angular velocity units (e.g. degrees per second). Therefore, a single integration on the samples provides the angular position.

$$\theta_i = \int_{t_{i-1}}^{t_i} \omega dt + \theta_0$$

$$\theta_i = \theta_{i-1} + \omega_{i-1} \times (t_i - t_{i-1})$$

In order to be able to estimate the nodes position in a 3D space, the IMU must contain at least a 3-axis accelerometer to sense acceleration to any direction, as well as a 3-axis gyroscope, to sense rotation in the any orientation (6). Along with these two components, in more sophisticated systems, others sensors like barometers, Pitot tubes (air speed sensors) and magnetometers are added to increase the location precision.

In order to enable the use of accelerometers and gyroscopes in small and low cost platforms such as WSN nodes, MEMS (Microelectromechanical systems) based solutions replace the traditional accelerometers and gyroscopes. The use of MEMS is crucial, since it allows the system to shrink dramatically in size, weight and power requirements.

2.3.1 Accelerometer

MEMS based accelerometers works in a very similar manner to the traditional ones. The core element is a moving beam structure composed of two sets of fingers: one set is fixed to a solid base; the other set is attached to a known mass spring like structure that can bounce according to the acceleration applied to it. When acceleration is applied to the system, the capacitance between the fixed fingers and the moving fingers varies as the distance between them changes, the sensor then evaluates these capacitance changes and outputs the equivalent value of acceleration (usually in G, i.e. gravity acceleration) to the system. Figure 4 shows a simple schematic of a MEMS based accelerometer.

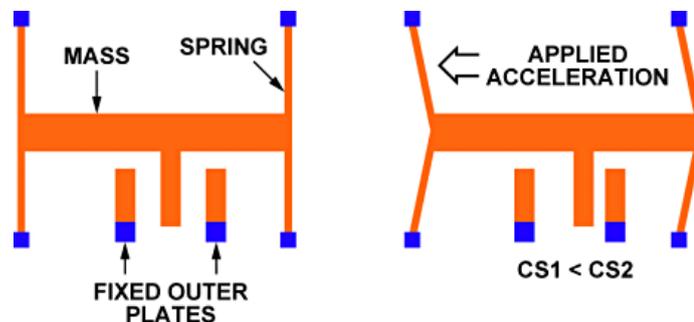


Figure 4 - MEMS based accelerometer (7).

2.3.2 Gyroscope

Unlike traditional spinning wheel gyroscopes, MEMS gyroscope does not have a spinning wheel inside them, they work analyzing the Coriolis Effect generated over a pair of oscillating structures when the MEMS rotate. Both structures oscillates synchronously but in opposite directions (Figure 5) and they are placed symmetrically on each side of the axis they are measuring rotation. When a rotation is applied, each of the vibrating structures perceive forces generated due the Coriolis Effect on opposite directions and like in the MEMS accelerometer, this forces dislocate finger like structures modifying the capacitance between them. This capacitance differential is then translated into angular velocity and outputted usually as degrees per second.

These two oscillating structures guarantee that only rotation movements are perceived. If lateral movement is executed, both structures will get dislocated to the same direction, therefore the differential value between them will be null, guaranteeing that this arrangement is not affected by linear acceleration such as tilt, shock or vibration.

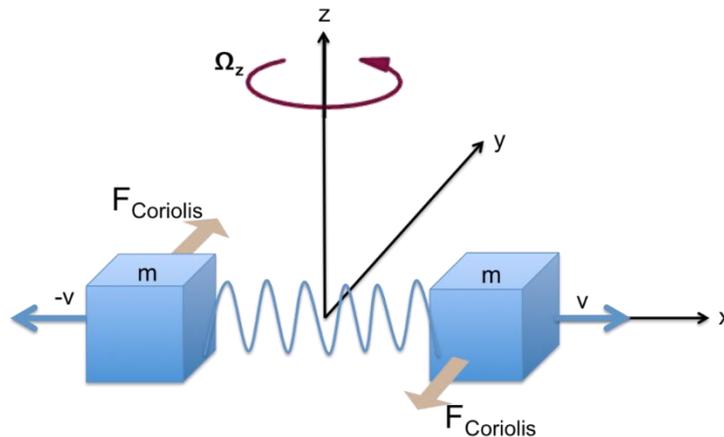


Figure 5 - MEMS based gyroscope diagram.

2.3.3 Combining inertial navigation and other localization techniques

The idea of combining other localization techniques with inertial localization is not a novelty. In (8) (9) the combination of low costs inertial sensors with GPS in outdoor land and pedestrian navigation has achieved reliable and accurate positioning. Therefore, inertial sensors in mobile devices can improve WSN localization, since both technologies have complementary characteristics.

The integration of RF navigation systems and inertial sensors has been used for decades in airborne, vehicle and military applications (10). They rely on military grade classified sensors (11), which are highly accurate, but neither affordable nor installable for low-cost WSN applications due to size, weight, cost, and power restrictions. However, the MEMS technology has been helping to proliferate the availability of reasonably accurate low-cost inertial sensors (9). Based on that premise, the interest in combining these sensors with more traditional WSN localization techniques have been increasing. The following related works describes some of these studies.

Sczyslo et al. (12) proposed a hybrid localization system combining inertial sensors and Ultra-Wideband (UWB) radios. Their sensor fusion algorithm can increase the accuracy and robustness of a UWB localization system that is subject to Non-Line-Of-Sight (NLOS). The performance is not substantially superior to the traditional UWB radios,

but since the performance of MEMS IMUs is in progress, utilization of such sensors will consequently improve the hybrid localization approach.

Zmuda et al. (13) proposed a technique of merging heterogeneous signals from inertial and RF sensors. They used joint Probability Distribution Function (PDF) to estimate position, velocity and acceleration of robots. At each step, this PDF is updated based on the RF readings and then updated again with the inertial sensors readings. They compared three solutions, RF only, IMU only and IMU+RF, all regarding error measurements to the real position of the robot. They showed that, in short period, IMU alone has a small advantage over the other solutions, since RF performs poorly and some of the RF errors affect the IMU+RF solution. However, for a solution only for IMU, the accumulation of drift errors is directly proportional to the time, while for RF solutions with these errors, it remains constant. Finally, they conclude that the IMU+RF solution is the one with the lowest error.

The work of Fink et al. (14) on sensor fusion of the RSS-based localization with an INS leads to a more precise tracking. The long-term stability of the RSS-based localization and the good short-term accuracy of the INS are combined using a Kalman filter (15). Experimental results on a motion test track show that the human tracking in multipath environments is possible with low infrastructural costs. Schmid et al. (16) performed a similar study and reached similar conclusions. However, they identified that the system requires a considerably high node density to operate properly.

Lee et al. (17) proposed an indoor pedestrian localization scheme, which can correct the accumulative error of IMU even if there is no a priori information on the location of the anchor nodes. They applied particle filtering to estimate the most probable position of the anchor nodes and pedestrians at run time. Their approach solves the drift error problem of inertial sensors in localization and alleviates the costs of system installation, since the number and position of the anchor nodes does not need to be known a priori.

2.4 Kalman filters

The Kalman filter (18) is an efficient recursive filter that estimates the state of a dynamic system from a series of measurements of signals with noise. It is a set of mathematical equations used to estimate the states of a linear dynamic system when disturbed by noise. The filter aims to minimize the mean square error, in other words, reducing the difference between the predicted state and the current state, thus, given an

initial value, the filter predicts the next state, and based on reading the current state, updates the prediction and thus minimizes the error at each update. Because of its recursive characteristic, this type of filter is computationally acceptable, since the calculations occur as soon as data are processed. The known techniques are based on the fact that the noise has a Gaussian distribution, and even if this hypothesis is not true, they remain a great estimator.

2.4.1 Kalman filter in continuous time

The following sections with definitions and descriptions are based on the work of Jwo, Hu and Tse (19).

2.4.1.1 Defining the problem

Consider the application of a generic system with multiple inputs and multiple outputs (MIMO), given a dynamic system subject to linear stationary process noise $v_x(t)$ and measurement noise $v_y(t)$, the characteristic equation can be given by:

$$\dot{x}(t) = Ax(t) + Bu(t) + v_x(t)$$

$$y(t) = Cx(t) + Du(t) + v_y(t), \text{ and}$$

$$v(t) = \begin{bmatrix} v_x(t) \\ v_y(t) \end{bmatrix}, \text{ where the noise } v_x(t) \text{ and } v_y(t) \text{ are uncorrelated in time.}$$

Without much loss of generality we can consider $D = 0$, thus we can write a covariance matrix as $V = \begin{bmatrix} \tilde{Q} & Z \\ Z & \tilde{R} \end{bmatrix}$, where

$$Z = Cov[v_x(t), v_y(t)]$$

$$\tilde{Q} = Var[v_x(t)]$$

$$\tilde{R} = Var[v_y(t)]$$

As additional assumptions are taken: $\tilde{R} > 0$, in other words, the noise component at non-zero covariance on each output; $Z = 0$, in other words, the noise on the state and output are uncorrelated; the state is modeled as a Gaussian random variable such that:

$$x_0 = x(0)$$

$$E[x(0)] = \bar{x}_0$$

$$E[(x_0 - \bar{x}_0)(x_0 - \bar{x}_0)'] = \tilde{P} \geq 0$$

Moreover, the noise and the state are not correlated, i.e. $E[x_0 v'] = 0$.

2.4.1.2 Observer state

In this point is considered the observer state:

$$\frac{d\hat{x}(t)}{dt} = \dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L(t)[y(t) - C\hat{x}(t)]$$

Now, with simple algebraic steps is possible to write the error dynamics as follows:

$$\dot{e}(t) = \dot{x}(t) - \dot{\hat{x}}(t)$$

$$\dot{e}(t) = A_c(t)e(t) + B_c(t)v(t)$$

Where,

$$A_c(t) = A - L(t)C$$

$$B_c(t) = I - L(t)$$

Now write,

$$\bar{e}(t) = E[e(t)]$$

$$\dot{\bar{e}}(t) = A_c(t)\bar{e}(t) + B_c E[v(t)] = A_c(t)\bar{e}(t)$$

It is noticed at this point that the expected value of the error is an autonomous system. In this case, we define the covariance matrix of the error:

$$\tilde{P}(t) = E[e(t)]$$

$$\tilde{P}(0) = \tilde{P}_0$$

The goal of optimization is therefore find $L(t)$ that minimizes the figure of merit $\min_{L(t)} \gamma' \tilde{P}(t) \gamma = \min_{L(t)} \|\gamma\|_{\tilde{P}(t)}^2$, where γ is a generic vector of appropriate dimensions.

The goal of the optimization is therefore to find a value for $L(t)$ in order to minimize the estimated error. It is observed that the gain $L(t)$ that solves the optimization problem can be defined with the following equation:

$L(t) = \tilde{P}(t)C'\tilde{R}^{-1}$, where $\tilde{P}(t)$ is the solution of Riccati equation defined by:

$$\dot{\tilde{P}}(t) = A\tilde{P}(t) + \tilde{P}(t)A' + \tilde{Q} - \tilde{P}(t)C'\tilde{R}^{-1}C\tilde{P}(t), \text{ with initial conditions } \tilde{P}(0) = \tilde{P}_0.$$

2.4.1.3 Stability of the Kalman filter

The stability of the Kalman filter can be defined by matrix B_q as a partition of the matrix \tilde{Q} , ie such that $B_q B'_q = \tilde{Q}$. The Kalman filter is stable if the pair (A, B_q) is reachable and the pair (A, C) is observed.

Based on these assumptions, a large estimator can be given by the following equation $\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L^*[y(t) - C\hat{x}(t)]$, where $L^* = \tilde{P}^* C' \tilde{R}^{-1}$, and that \tilde{P}^* is the only definite solution of the Riccati equation stationary, $0 = A\tilde{P}^* + \tilde{P}^* A' + \tilde{Q} - \tilde{P}^* C' \tilde{R}^{-1} C \tilde{P}^*$.

2.4.2 Kalman filter in discrete time

Consider the application of a generic MIMI system, given a dynamic system subject to linear stationary process noise $v_x(t)$ and measurement noise $v_y(t)$, the characteristic equation can be given by:

$$x(k+1) = Ax(k) + Bu(k) + v_x(k)$$

$$y(k) = Cx(k) + v_y(k)$$

$$v(k) = \begin{matrix} v_x(k) \\ v_y(k) \end{matrix}$$

Where the noise $v_x(k)$ and $v_y(k)$ are uncorrelated.

$$\hat{x}(k|k) = \hat{x}(k|k-1) + L(k)e(k)$$

$$P(k|k) = P(k|k-1) - L(k)N(k)L'(k)$$

$$\hat{x}(k+1|k) = F\hat{x}(k|k) + ZR^{-1}_y(k)$$

$$P(k+1|k) = FP(k|k)F' + M$$

Where,

$$L(k) = P(k|k-1)C'[CP(k|k-1)C' + R]^{-1}$$

$$e(k) = y(k) - C\hat{x}(k|k-1)$$

$$N(k) = CP(k|k-1)C' + R$$

$$F = A - ZR^{-1}C$$

$$M = Q - ZR^{-1}Z'$$

2.4.3 Kalman filter for IMU signals in MatLab

The use of the Kalman filter as a solution to the noise problem as the signals from the IMU is justified by the nature of the signal, i.e. the 3-axis IMU (x, y, z) for all three sensors (accelerometer, gyroscope and magnetometer) has as output a signal with white Gaussian noise. Figure 6 represents a signal with a lot of noise, however this set of signals can be represented as a set of signals with Gaussian distribution of zero mean. The red line in the figure indicates the average of the points, i.e. our aim is to represent the red line which is a noise-free signal.

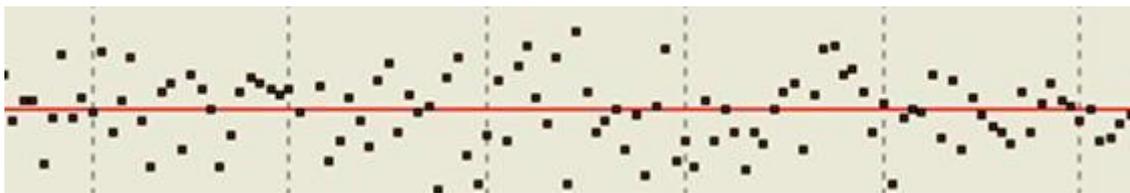


Figure 6. Example of a signal with noise.

The result of Figure 6 can be corrected with the Kalman filter easily and efficiently. Below the filter was described in MatLab code. It is high-performance software designed to perform calculations with arrays, and it can function as a calculator or as a scientific programming language. Furthermore, Matlab commands are closer of the way we write algebraic expressions, making it easier to use.

```
function [k,s] = kfilter(A,C,V1,V2,V12)
%KFILTER can have arguments: (A,C,V1,V2) if there are no cross
% products, V12=0.
%   KFILTER calculates the kalman gain, k, and the stationary
%   covariance matrix, s, using the Kalman filter for:
%
%           x[t+1] = Ax[t] + Bu[t] + w1[t+1]
%           y[t]   = Cx[t] + Du[t] + w2[t]
%           E [w1(t+1)] [w1(t+1)]' = [V1   V12;
%           [ w2(t) ] [ w2(t) ]      V12' V2 ]
%
% where x is the mx1 vector of states, u is the nx1 vector of controls, y is
% the px1 vector of observables, A is mxm, B is mxn, C is pxm, V1 is mxm,
% V2 is pxp, V12 is mxp.
m=max(size(A));
[rc,cc]=size(C);
if nargin==4; V12=zeros(m,rc); end;
if (rank(V2)==rc);
    A=A-(V12/V2)*C;
    V1=V1-V12*(V2\V12');
    [k,s]=doubleo(A,C,V1,V2);
    k=k+(V12/V2);
else;
    s0=.01*eye(m);
    dd=1;
    it=1;
    maxit=1000;
    while (dd>1e-8 & it<=maxit);
        k0= (A*s0*C'+V12)/(V2+C*s0*C');
        s1= A*s0*A' + V1 - (A*s0*C'+V12)*k0';
        k1= (A*s1*C'+V12)/(V2+C*s1*C');
```

```
    dd=max(max(abs(kl-k0)));
    it=it+1;
    s0=s1;
end;
k=kl;s=s0;
if it>=maxit;
    disp('WARNING: Iteration limit of 1000 reached in KFILTER.M');
end;
end;
```

Figure 7. MatLab code for the Kalman filter.

3 SIMULATION AND OTHER TOOLS

3.1 Discrete Event Simulation

Discrete-event simulation (20) is widely used and simply works with a list of pending events, which can be simulated by specific routines. The global variables describe the system state and simulation time, which allow the scheduler to predict next events and increment this time. In-house simulators could profit from discrete event principles, since it only needs to set up an event scheduler, a global simulation clock, a pseudo-random numbers generator, and some mechanism of statistical processing of collected samples.

For WSN simulation, events such as communication between nodes, sensing and sleeping can be scheduled in a list of events based on discrete times of occurrence defined by pseudo-random trials. The guarantee of some confidence interval for the statistical results is dependable of the number of samples collected in a long run simulation. This framework is very straightforward and simple to implement, and in addition, very useful for validating complex behavior without the actual system implemented or even being monitored only after its deployment in the real environment.

3.2 WiNeS

WiNeS framework (21) was developed to be a simulation framework to test protocols and architectures for wireless networks, from scenarios predetermined by designers within a simple API. WiNeS presents advantages: (i) it was developed at PUCRS in the context of Flexgrid project (22); (ii) the framework was written in Java, then it is multiplatform; (iii) simulation of various scenarios, including those with heterogeneous elements; (iv) the API is simple and easy to understand; (v) allows the inclusion of separately modules in Java; (vi) also allows the simulation of wired networks, if needed only ignoring environment settings such as the distance verification between communicating nodes. Comparing WiNeS with major simulators, it can be concluded that all intend the same goal of facilitating the creation of communication network topologies.

3.3 OMNeT++

OMNeT++ (23) (24) is a modular simulation framework written in the C++ programming language based on discrete events. Each component of this simulator is

based on modules that can be hierarchically nested for creating composite modules. Simple modules are used for defining algorithms and they are located last in the hierarchy, while composite modules are composed of simple modules that interact with one another through message exchanging. Any of these modules can also be used in different simulation projects. A good example is a network sensor where each of its modules implements one layer of the protocol stack.

This modular approach gives OMNeT++ a greater flexibility for implementing different network scenarios. However, just like with NS-2 (25) and NS-3 (26), a greater level of understanding on how these modules interact with each other is required by the user for creating new modules. OMNeT++ offers a robust graphical interface and the possibility of plotting the network scenario and simulation results. It also comes with a kernel library for creating new modules for different network algorithms. Other simulators are also built on top of OMNeT++'s framework (27).

3.4 Random-based mobility models for nodes

Mobility models represent the movement of mobile nodes, and how their location, velocity and acceleration change over time. Such models are frequently used for simulation purposes when e.g. new communication techniques are investigated. In random-based models, mobile nodes move randomly and freely without restrictions.

3.4.1 Random Waypoint model

It was first proposed by Johnson and Maltz (28), then it became a 'benchmark' mobility model to evaluate the MANET routing protocols.

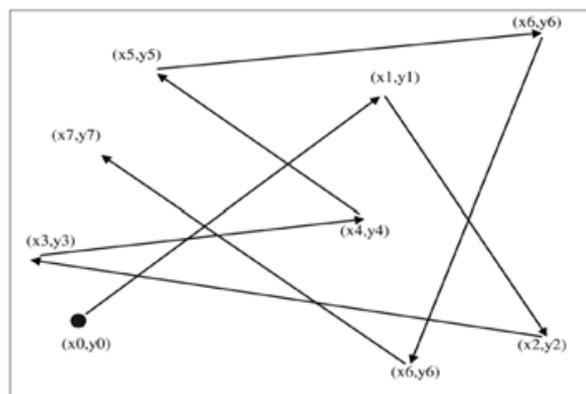


Figure 8. Example of node movement using Random Waypoint model.

The movement of nodes is governed in the following manner: each node begins by pausing for a fixed number of seconds. The node then selects a random destination in the simulation area and a random speed between 0 and some maximum speed. The node moves to this destination and again pauses for a fixed period before selecting another random location and speed. This behaviour is repeated for the length of the simulation.

The random placement (i.e Random Waypoint model) is one of the simplest models that was first proposed by Johnson and Maltz (Bai & Helmy). The model determines that each node choose a random point within the sensing field, and then get around to this point (Figure 8). The node moves around associated with a velocity vector \vec{V} entre $[0:V_{max}]$, where V_{max} is the node maximum speed. After arriving at the destination, the node waits for a t_{pause} time, and then chooses a new point to get around. This cycle may repeat indefinitely covering a larger sensing area. Figure 9 shows a node at position A that initially chooses to get around to position C. Position B represents an intermediate position between position A and C. The node in position B has a velocity vector \vec{V} . After reaching the destination C, the node waits a while, and randomly decide that the new destination is to position D.

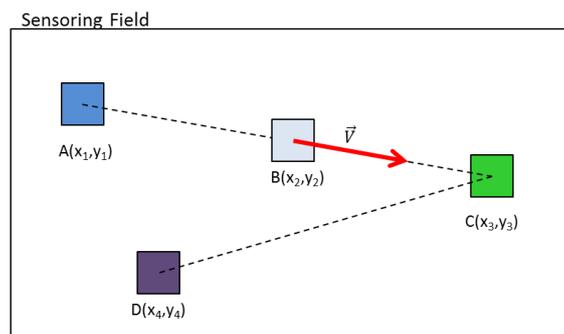


Figure 9. Example of the mobility model using random point.

3.4.2 Random Direction model

The Random Direction model randomly sets a direction to be traveled, and it stops only when reaches the limit of the sensing field. Upon arriving at the threshold, the node is stopped for a time t_{pause} , and chooses another path to move. As the Random Waypoint model, has a velocity vector \vec{V} between $[0:V_{max}]$, V_{max} being the maximum speed the node can reach. This cycle is repeated indefinitely.

In Figure 10, the node starts at position A and then randomly chooses the east direction (0° with respect to the abscissa axis) with velocity vector \vec{V} . Upon arriving at the boundary of the sensing area, the node stops for a while (t_{pause}), and then chooses a new

3.5 Considerations about simulators and WSN evaluation

About simulators for evaluating WSN, there are several software packages, modules and implementations available in the literature (29). But the great difficulty, among all these options, it is a learning curve to implement case studies in more robust tools like OMNeT ++ (23), and also the lack of documentation, but also for more simplified tools and easy to understand how the case of Wines (21), developed in the context of the research project Flexgrid (22), which presents the basic module for node simulation but still has limited development of diverse libraries for the integration of realistic models of WSN.

3.5.1 Sensor nodes characteristics

A sensor node consists of four main modules: processing unit, power unit, sensing unit and communication unit (30). Figure 3 depicts the basic organization of a node.

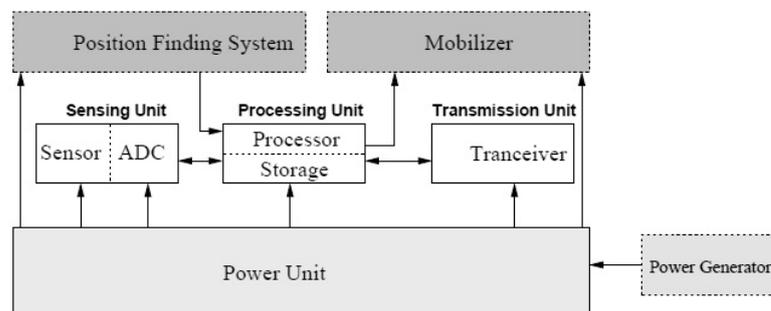


Figure 12. Components of a sensor node (31).

The processing unit contains the node application and does the basic processing of data collected by the sensing unit. It also manages the sending of this information to the target, the energy supply, the mobility and localization. The processing unit also contains a memory unit. The power unit provides power to the node and may also be associated with a collector power unit, for example, solar cells. The sensing unit includes a sensor component and an ADC (Analog to Digital Converter) that converts sensor's analog information to a digital system to be processed by the processing unit. The communication unit is responsible for the connection of the node to the network (32). Some other units may also be useful in the architecture of a sensor node, such as localization systems for mobile nodes. Most sensor network routing techniques and sensing tasks require the node location with precision, thus it is required localization techniques (5).

One of the challenges related to the construction of sensor nodes is getting them to work with an extremely low power consumption to increase the node lifetime. For this,

there is an alternative to extract energy from the environment, or other alternatives for energy saving in communication methods, such as: information reduction, improvement the network organization and improvement on the methods for data synchronizing during communication.

3.5.2 Nodes energy consumption

A wireless sensor node has limited energy source, and in some scenarios, it is very difficult (or impossible) to replenish the power supply. "The depletion of some nodes in the network can significantly alter the network topology being necessary to reconfigure packets routing and network reorganization" (1). The communication system is responsible for most part of the energy spent on a wireless node. Remark that sending messages is much more energetically costly than data processing (9). Then, a preliminary analysis of data collected for a reduction of network traffic is required to increase the WSN lifetime.

3.5.3 Energy models for simulation

For WSN simulation we usually assume specific amounts of energy spent for each activity on the network, or on individual sensor nodes, in order to allow estimates about network lifetime and/or duration of batteries. In energy models proposal becomes impractical to represent all the components of a sensor node that consume energy in each activity. Thus, it is necessary to abstract hardware characteristics (many components has negligible energy consumption and therefore may be abstracted). Many models also represent the most performed operations, e.g. packets transmission and reception, assuming that these energy costs may already represent a large portion of the total energy consumption - even so simulations can provide results with some inaccuracy.

Several studies in the literature have explored different models for energy measurement in simulations (33), (34), (35), (36), (37), (24), (38), (39). The majority of these models focus on energy consumption to perform data transmission and reception. Evaluations performed for LEACH protocol by Heinzelman et al. (33), for example, abstracted several factors that influence the energy consumption of nodes, using only the abstraction of packets transmission/reception.

Tabela 1. Example of energy model for simulation purposes.

WSN operation	Estimated energy consumption
Packet reception - maximum size of 500 bytes	$e_{Elec} = 25 \text{ uJ}$
Packet transmission - maximum size of 500 bytes	$e_{Elec} + e_{Amp}$, where $e_{Amp} = 50 \text{ nJ} \times \text{distance}^2$

Tabela 1 illustrates that for the radio module to receive a packet, it just dissipates e_{Elec} energy, which is energy that the radio spends to process incoming packages (unpacking, processing). This energy is a function of packet size where each bit of data received consumes 50 nJ energy. Thus, for a packet of 500 bytes, e_{Elec} takes 25 μ J.

The energy spent in the transmission is given by the sum of two energies: (i) energy spent in radio module for packaging and compression, which is the same energy expended in activity receiving; (ii) the power amplifier on the e_{Amp} . This energy model uses "free space channel" for computer energy dissipated, where each bit consumes 100 pJ multiplied by distance squared (between transmitter and receiver). Thus, for a packet of 500 bytes, is given by $e_{Amp} 50nJ \times distance^2$.

4 PROPOSED LOCALIZATION TECHNIQUE

In many applications, the area where nodes are able to navigate is far superior from the area covered by the WSN anchor nodes; this means that for a considerable amount of time, the nodes will not be able to use the RF techniques to correct the drift errors of the inertial systems. The related works analyses, presented in the previous session, enable to notice that none of these approaches cover this important and very common aspect. This work aims to solve this issue, providing a localization schema that allows the nodes to fix part of the drift error accumulated in the inertial system and correct its estimated position, even with the absence of anchor nodes with known positions in range.

4.1 Algorithm description

The basic idea behind the proposed technique is that each node will be able to try to define its position by not only using anchor nodes position as reference, but also other non-anchor nodes that get in connectivity range with it. To achieve that, each mobile node will have two internal parameters: its position relative to a known coordinate system and its degree of confidence; that is a real number between 0 and 1 that represents how much the node trust in the position it estimated. In this initial work, the position may be defined using one of two techniques, Connectivity or Inertial Navigation.

Connectivity: when a node is in range of communication with another node, both will exchange estimated positions and degrees of confidence, and the position of the node with the highest degree of confidence will be selected as the position of both nodes. This technique will have the highest priority.

Inertial Navigation: if a node is out of connection range of others, the node will use its previous known position and, based on inertial sensors, estimates its new position. The degree of confidence will also be updated (i.e. it will be decreased).

In this model, the anchor nodes are simple nodes with known position and degree of confidence equal to 1. Therefore, anytime a mobile node connects with an anchor node, the position of the mobile node will be set as equal to the anchor node position (since anchor nodes have a constant and maximum degree of confidence) and the degree of confidence in the mobile node will be updated to 1. As the mobile node moves away from an anchor node, it will lose connection and will rely only on the inertial measuring in order to estimate its current position. During this period, the degree of confidence will be

decreasing from 1 down to 0, until it encounters another anchor node or non-anchor node with a degree of confidence higher than itself.

Figure 6 illustrates a schematic of such algorithm where BS1, BS2 and BS3 are anchor nodes with known position; notice that anchor node can also be mobile, as long as they are equipped with a GPS or any other similar location system that can provide accurate location. S-1 and S-2 are mobile nodes of interest, equipped with INS and able to connect and exchange information between themselves and the anchor nodes. The light grey circles represent the connectivity area of the nodes, when two or more are inside a circle, they can connect and exchange data between them.

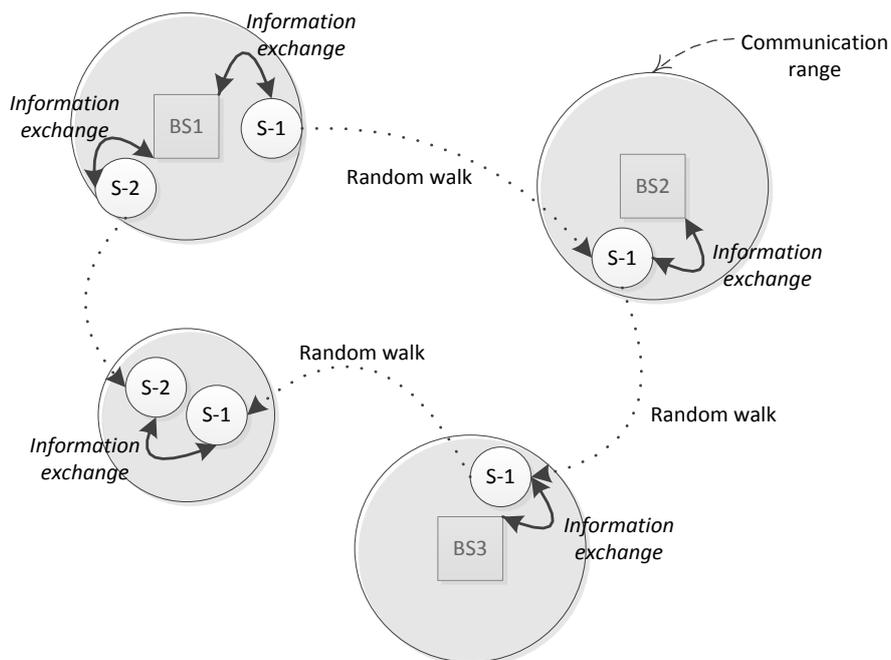


Figure 13 - Proposed localization algorithm schematics.

The calculation of the confidence degree will always be in an interval $(0, 1]$, where 1 represents full confidence in the node position and as closer to 0 it gets, less reliable is the estimated position. Note that if two or more nodes are in connection range and have a degree of confidence of 1, they must have the same position; otherwise the system will be inconsistent.

In order to achieve an accurate operation of this algorithm, it is crucial that the mobile nodes cross each other path during the moving period, the frequency necessary to ensure the feasibility of the algorithm is not yet known.

Another important point is that the random meetings between non-anchor nodes should reduce the overall system accuracy. Thus, it is crucial that mobile nodes cross with

anchor nodes periodically so that the system can remove the drift error and maintain a high confidence degree of the system in average.

This work proposes a localization algorithm based on both, RF and inertial systems to estimate the position of moving nodes in a spread scenario; i.e. a scenario where the majority of the areas where the nodes can move along are not covered by the communication range of anchor nodes. Trying to determine the minimum frequency of random encounters, as well as the number and range of the anchor nodes that will allow the system to work efficiently is a secondary objective.

4.2 Preliminar hardware tests

4.2.1 The proposed test bed

In order to validate the localization algorithm, a test bed will be deployed; it will be composed of multiple nodes running the localization algorithm firmware. Each node will have a programmable microcontroller, a RF transceiver, an IMU and the power supply.

The objective of the test bed is to track and store the nodes position during a period and export these values so we can calculate the error between the actual position and the estimated position of the nodes. The test bed firmware will be running in the nodes and it is responsible for the communication protocol, sensors reading and position estimation. The same firmware will be employed in all anchor or non-anchor nodes.

4.2.1.1 Hardware components

A microcontroller, a RF transceiver, an IMU and the power supply will compose each node; they will be assemble over a protoboard and interfaced using a window on PC.

4.2.1.1.1 Microcontroller

The microcontroller board chosen is an Arduino Uno R3 [3] shown in Figure 7. The Arduino Uno is a microcontroller board based on the ATmega328 [16]. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. Another advantage of this board is that it is programmable through USB. Table 1 describes the characteristics of the board.

Table 1 - Characteristics of Arduino Uno R3 board.

Description	Type / Value
Microcontroller	ATmega328
Operating voltage	5V
Input voltage (recommended)	7-12V
Input voltage (limits)	6-20V
Digital I/O pins	14 (of which 6 provide PWM output)
Analog input pins	6
DC current per I/O pin	40 mA
DC current for 3.3V pin	50 mA
Flash memory	32 KB (inside ATmega328)
SRAM	2 KB (inside ATmega328)
EEPROM	1 KB (inside ATmega328)
Clock speed	16 MHz

4.2.1.1.2 RF Transceiver

The transceiver chosen is an XBee-PRO 802.15.4 Wire Antenna Series 1 (Figure 8). This is a very popular 2.4GHz RF module from Digi (formally Maxstream). The XBee-PRO series have an output power of 60mW with a theoretical communication range of 1500 m. These modules come with an IEEE802.15.4 stack and wrap it into a simple to use serial command set. It support point-to-point and multipoint networks.

The XBee-PRO can operate either in transparent data mode or in packet-based Application Programming Interface (API) mode [17]. In the transparent mode, data coming into the Data IN (DIN) pin is directly transmitted wirelessly to the intended receiving radios without any modification. Incoming packets can either be directly addressed to one target (point-to-point) or broadcast to multiple targets. In API mode, the data is wrapped in a packet structure that allows for addressing, parameter setting, packet delivery feedback, including remote sensing and control of digital I/O and analog input pins. Another important feature is the ability to extract the RSS information of each packet received.



Figure 14 - Arduino Uno R3.



Figure 15 - XBee-PRO Series 1 Wired Antenna.

4.2.1.1.3 Inertial Measurement Unit (IMU)

The chosen IMU is the MPU-6050 from Sparkfun Electronics (Figure 9). The MPU-6050 combines a MEMS 3-axis gyroscope and a MEMS 3-axis accelerometer on the same silicon die together with an onboard Digital Motion Processor (DMP), capable of processing complex 9-axis motion fusion algorithms [18].

The breakout board for the MPU-6050 facilitates the chip connection to a protoboard or microcontroller. Every pin needed are broken out to 0.1" header pins, including the auxiliary master I2C bus, which allows the MPU-6050 to access external magnetometers and other sensors.

The interface with the MPU is done using I2C bus; it is possible to request the inertial data in a variety of formats, rotation matrix [21], quaternion [22], Euler Angle [23], or raw data format. In addition, the MPU-6050 can be configured with one of multiple sensitivities, making them suitable for a higher number of applications. Table 2 describes the general features of the MPU-6050.

Table 2 - IMU Features

Description	Type / Value
Digital output	I ² C
Operating voltage	2.4 V to 3.5 V
Gyroscope scale range (°/sec)	[-250,+250]; [500,+500]; [1000,+1000]; [-2000,+2000]
Gyroscope sensitivity (LSB/°/sec)	131; 65.6; 32.8; 16.4
Accelerometer scale range (g)	[-2,+2]; [-4,+4]; [-8,+8]; [-16,+16]
Accelerometer sensitivity (LSB/g)	16384; 8192; 4096; 2048

Legend: ° - Degree sec - second LSB –Least Significant Bit g - Gravity



Figure 16 - MPU-6050 breakout board.

4.2.1.1.4 Power supply

In order to power the microcontroller board, it is required a minimum of 6 V. Therefore, it was decided to use two 3.7 V lithium rechargeable batteries (Figure 10) in series, for 7.4 V. The MPU-6050 and the Xbee-PRO will be powered by the 3.3 V-regulated output from the microcontroller board.



Figure 17 - 3.7 V lithium rechargeable batteries.

4.2.2 Procedures for the Arduino platform tests

The preliminary tests were performed with the Arduino connected directly to a microcomputer, using a USB cable, eliminating the need of a power supply and improving the speed of the changes in the code used (please refer to Appendix I). In a protoboard, it was attached the IMU, connected to the microcontroller using the configuration provided by the datasheet of Sparkfun electronics (40). The only parameter measured in these tests was the acceleration, due to its more important role in the definition of spatial positioning.

The test was executed according to the following steps: first, the code was uploaded to the Arduino, including all the sensor configurations. So, with the IMU properly connected to the Arduino, the protoboard, where the IMU was attached, was translated 60 centimeters in one axis only, over a plane table. After these 60 centimeters, the protoboard was translated, returning to its initial position. All these movements lasted for approximately 5 seconds. After that, the protoboard was left stopped for more 5 seconds.

The program of the Arduino registered the change of the acceleration suffered by the IMU during the movement, taking new measures every 3 miliseconds (approximately). All this measures were recorded in a text file. Using the data accumulated during the 10 seconds, it is possible to plot a graphic that shows how the fluctuation of the acceleration performs during the movement execution.

4.2.2.1 First results achieved

Analysing the graphics generated by the Microsoft Excel, it is possible to conclude that the acceleration measured is compatible with the movement executed. When the protoboard left the resting state, the acceleration showed an increase, result of the great change of speed (if we realize that it started stopped, or with speed zero). After the 60 centimeters, the acceleration measured showed a decrease, due to the returning movement executed. In fact, the plot shows a mirrored variation of the acceleration.

However, we could note that the data has a bit of noise, result of the constructive characteristics of the accelerometer inside the IMU. This fact can be noted if we pay attention to the instability of the data acquired, that changes a little in a certain limit, even in the stopped state.

Another situation found is about the orientation of the IMU. Any disturb in the protoboard, which is not related to the specific translations, results in a significant error in the acceleration measured. The question is that if we move the IMU while it's not perfectly plane, the acceleration will be divided in two (or three) coordinates. So, the values measured can be different from the real ones.

These facts turn out to interfere in the double integral, used to obtain the distance from the acceleration. Integrating numerically all the data, we determined the velocity, and after that the distance. The value obtained for the distance is far different of the real one, resulting in an error of approximately 30 times.

Due to the errors found during the tests using only the accelerometer, alternatives were searched to improve the determination of positioning, mostly to reduce the noise during the data acquisition. The options already studied are presented in the next sections.

4.2.3 Magnetometer tests

One of the options is to use another sensor to help the accelerometer in the measurements. As one of the problems was related to orientation, the use of a magnetometer was studied.

The magnetometer is a sensor that measures the magnetic field around it, determining its strength and its direction in space. Using this information, would be (theoreticaly) possible to compensate the noise generated by the changes (that are not related to the main movement) in the orientation of the IMU, because the magnetic field is a magnitude that doesn't modify in small distances.

Nevertheless, the tests showed some difficulties in the use of the magnetometer. This sensor is very susceptible to any kind of interference, like monitors and cellphones, making their use restrict to enviroments of controled radiation.

Another difficulty discovered was the manner of how the data is measured by the magnetometer. The values acquisited are complex to work with because they are related to different coordinates of the spatial ones. In another words, the coordinate system of the magnetometer is directly related with the planet Earth magnetic field, rather different from the coordinate system of the accelerometer. Because of that, it's complicated to unite both data measured.

4.2.3.1 Numerical filters

A second option to minimize the errors was the use of numerical filters.

4.2.3.2 Signal power

A third option, which would help with the data acquisition of the accelerometer, is the use of signal power.

4.3 Matlab and Arduino experiments

4.3.1 Filters and algorithms

The purpose of a filter is to extract the key information of a signal. Figure 18 shows an experimental schematic of obtaining signals related to the IMU (accelerometer, magnetometer, and gyroscope). The IMU sensors detect movement and transmit signal to the Arduino, which contains embedded software responsible for managing the results of the IMU and transmit to a USB port. The letter "A" of this figure is represented plotting in MatLab, the result obtained directly from the Arduino port, it is observed that this signal has a lot of variation, as shown in detail in the bottom rectangle of this image. This variation is known as noise.

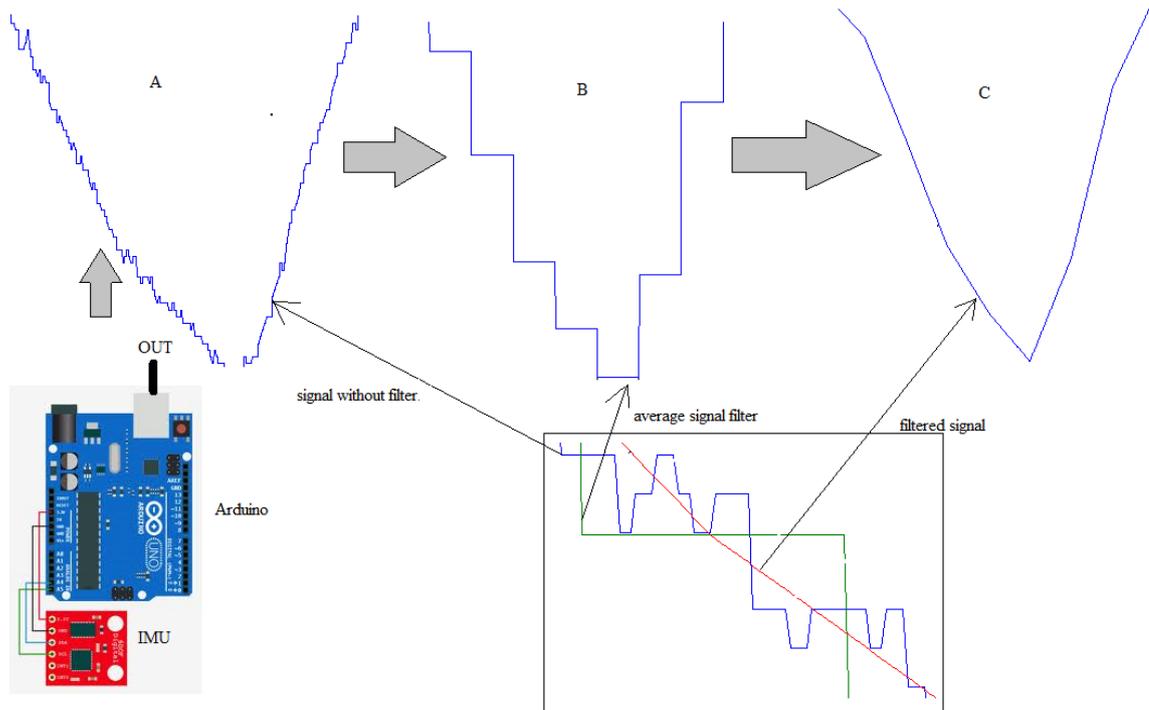


Figure 18. Schematic of obtaining signals related to the IMU.

The filter was implemented as follows. First, an algorithm is embedded in the Arduino platform with the algorithm that manages the IMU. This algorithm stores the IMU output into two buffers, i.e. it stores each of the 100 points of IMU output: the first buffer stores the first 50 points and the second buffer the other 50 points. In this example, the buffers have size 50. Then, it performs two arithmetic averages calculations, one for each buffer. This result was plotted in MatLab and it is represented in the letter "B" in Figure 18. Finally, the algorithm creates a third filter buffer of size 50 which stores increasing or decreasing values. The first value of the third buffer is the average of the first, and the fiftieth buffer value is the average of the second buffer. Refer to letter "C" in this figure, which represents the final result of the filter.

4.3.2 Results obtained with the IMU data and filter

Figure 19 represents in the upper line, the rotation of a variable (XYZ) in one of the three axes (XYZ); in the middle line represents the result of this rotation, referring to the magnetometer; and in the bottom line represents the result after filtering the signal. In the first column, in the top row of this figure, is represented the rotation X in the Y axis of the IMU; in the middle line has the plot representing the MatLab output X of the magnetometer; in the last line, of the first column, is represented the same output after applying the filter. The three IMU variables, despite having individual data, were initially

implemented for a single variable, however, it is necessary to implement all the three coordinates of the IMU sensors, since it is known that all signals are sent on the same frequency, therefore, beyond the noise that is natural this system, there is the possibility of interference between the signals. Considering these problems, it is planned to implement the Kalman filter in this work, because it is a more robust and efficient filter.

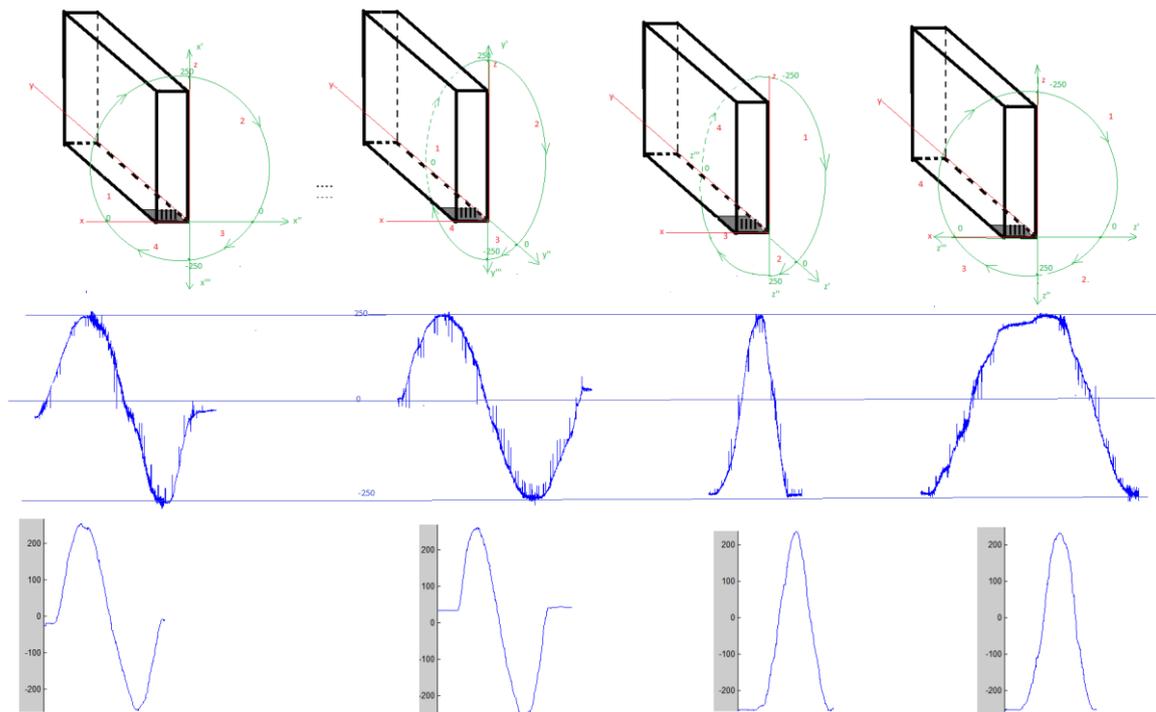


Figure 19. Results obtained with the IMU data and filter.

4.3.3 Comments on the Arduino experiments

Currently, we have an algorithm (embedded in Arduino) responsible for collecting IMU variations. The connection between Arduino and IMU is performed by wire, using I2C technology. The aim of this work is to develop a wireless sensor, thus it will be necessary the communication between I2C IMU with a wireless device (XBee). The wireless device sends data to a router XBee Pro, communicating physically with the Arduino. The Arduino is used to analyze the IMU data. In addition, a communication protocol among these sensors must be defined.

The experiments showed a significant amount of noise in the signal. We know that the wireless communication generates more noise when compared to wired communication because the medium interference is more significant. Thus, a Kalman filter is embedded in Arduino and it is expected and probably embedded in the router's wireless network. It is the greatest challenge of implementation, since the Kalman filter makes use of a lot of mathematics, easy to be implemented in MatLab, but extremely difficult to implement in C programming language.

4.4 Localization simulation scenario using WiNeS

4.4.1 Module implementation on the WiNeS environment

Primarily the network structure and nodes must contain the attributes and methods required for the simulation environment configuration. This simulation environment should provide the requirements for implementation and evaluation of the proposed heuristic algorithm (refer to Section 4.1). Figure 20 describes a class diagram containing these proposed characteristics integrated into the WiNeS framework (on node layer) (21). For example, attributes and methods for node positioning, transmission power, sensing frequency, speed mobility, mobility patterns. Following we present some examples of specific descriptions on WiNeS framework, algorithms and simulation output format.

4.4.1.1 NodeLayer

transmissionPower – It was necessary to create this basic attribute in the WiNeS framework because this attribute allows calculating the distance based on the power of the incoming signal.

4.4.1.2 DefaultNode

Range – it is an attribute used to determine the radius of action of the node. It should be based on transmission power of the signal and the type of transmission device.

Energy – it is the total energy of the node to simulate a power source like a battery.

Alive – it indicates whether the node is alive, and this variable is useful only for simulation purposes and statistics.

getRange – this method returns the maximum distance that a node can transmit a packet.

setRange(range: double) – this method assigns a value to the range attribute.

logPosition() – this method creates a string containing node information: nodeID, current simulation time; X, Y and Z positions; X, Y and Z approximated positions. In order to create the string, it uses `positionLogFile(structuredString: string)` method to write data in the simulation file.

createLogFile(fileName: string, timeSimulation: int, spaceLimits: double) – this method creates the output file containing in its header: simulation name, simulation time, and boundaries of the simulation field.

positionLogFile(structedString: string) – this method inserts in the simulation file the information contained in the structuredString.

run() – this method overrides the Node class containing the node's loop execution.

4.4.1.3 MobileNode

Frequency – it indicates the frequency for choosing another target position in the simulation using RandomWaypoint or RandomDirectionPoint models.

Accuracy – indicates the positioning precision achieved.

waitTime – it indicates the current waiting time based on Frequency, i.e. pause time for choosing a new node positioning.

waitTimeTotal – it indicates the ratio $1/\text{Frequency}$.

moving – this attribute indicates that the node is moving in the environment.

destinyX – it indicates the target value in X, and the node will move until it arrives in this X position.

destinyY – it indicates the target value in Y, and the node will move until it arrives in this Y position.

steps – it indicates the number of iterations that will be performed to arrive at the target position. It is a natural number and it is based on the maximum established speed.

maxVelocity – it indicates node maximum velocity.

velocityX – it indicates a velocity in X.

velocityY – it indicates a velocity in Y.

SetDestinity(...) – this method determines the new destination (target position).

AreInFinalPosition() – this method verifies the node positioning, if it has arrived on the target destination.

UpdatePositon() – this method is used to perform node mobility.

SetDirection(angleRad : double) – it defines a new direction angle in radians.

RandomAngle() – this method is used for choosing a random angle.

RandomWaypoint() – it choses a new aleatory destination to move following the attributes of Frequency and maximum velocity.

RandomDirection() – it chooses a new angle of destination to move until finding the simulation field boundaries.

4.4.1.4 StaticNode

A static node may easily exchange information with mobile nodes adding confidence to the localization heuristic.

4.4.2 Class diagram

Below we present the class diagram (Figure 20) implemented to compose simulation scenarios on WiNeS. Attributes and methods are illustrated in order to exemplify how nodes in the system will be capable to behave and respond to simulation environment inputs.

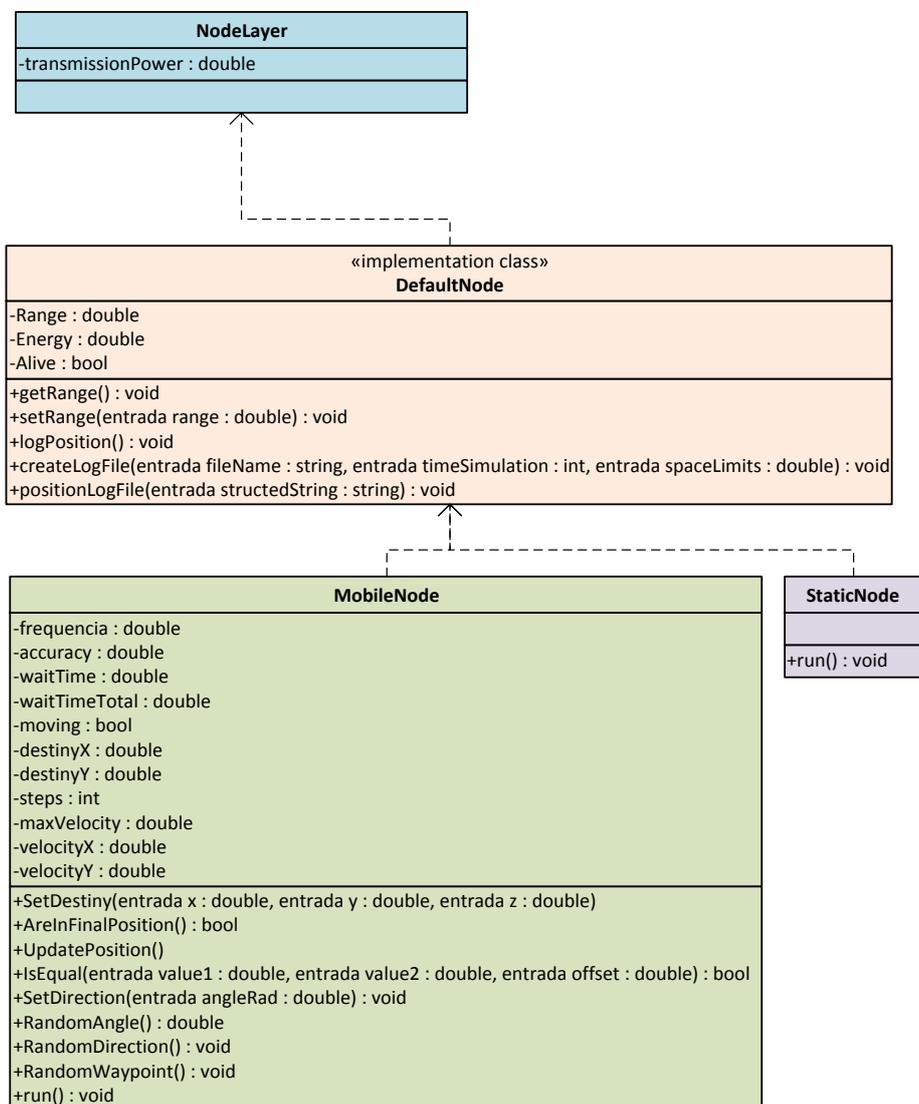


Figure 20. Proposed class diagram with node attributes and methods.

4.4.3 Algorithms description

Following the methods and algorithms used for node mobility and positioning.

Algorithm 1(setNewDestiny): it defines a new target position based on an angle.

<pre> lim_x ← x ← currentPosition.X lim_y ← y ← currentPosition.Y </pre>	Variables initialization
<pre> if angle > 0 and angle < $\frac{\pi}{2}$ then lim_x ← simulationLimit lim_y ← simulationLimit else if angle > $\frac{\pi}{2}$ and angle < π then lim_x ← 0 lim_y ← simulationLimit else if angle > π and angle < $\frac{3\pi}{2}$ then lim_x ← 0 lim_y ← 0 else if angle > $\frac{3\pi}{2}$ and angle < 2π then lim_x ← simulationLimit lim_y ← 0 end if </pre>	Definition of target boundaries. Refer to Figure 21.
<pre> if angle = 0 or angle = 2π then x ← simulationLimit else if angle = $\frac{\pi}{2}$ then y ← simulationLimit else if angle = π then x ← 0 else if angle = $\frac{3\pi}{2}$ then y ← 0 else x ← (lim_y - currentPosition.Y)/tan(angle) + currentPosition.X y ← (lim_x - currentPosition.X)/tan(angle) + currentPosition.Y if x < 0 then x ← 0 else if x > simulationLimit then x ← simulationLimit end if y < 0 then y ← 0 else if y > simulationLimit then y ← simulationLimit end if end if </pre>	Positioning is calculated in the intersection of two lines. The limit line and the line based on input angle and current node position. Refer to Figure 22.
<pre> destinyX ← x destinyY ← y </pre>	New target assignement

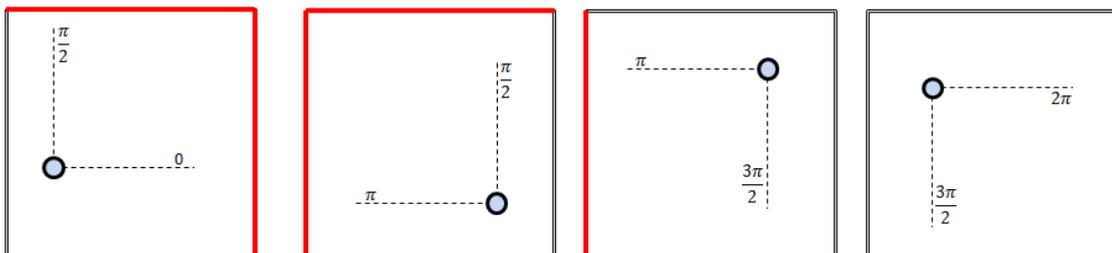


Figure 21. Lines boundaries (four cases).

In the first case (leftmost) boundaries are the limits of the simulation area, both for X and Y. In the second case, boundaries are the limits of the simulation area for Y and 0 for

X. For the third case, the limits are set to 0 for X and Y. For the fourth case, the limit is 0 for Y and the limit of the simulation area for X.

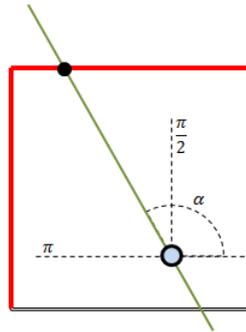


Figure 22. Example of lines intersection based on angle α .

Algorithm 2 : RandomDirection method

<pre> if moving = false then if waitTime = waitTimetotal then angle <- randomNewAngle setNewDestiny(angle) waitTime <- 0 timeToDestiny <- calcTimeToDestiny; velocityX <- distanceInX / timeToDestiny velocityY <- distanceInY / timeToDestiny moving <- true else waitTime <- waitTime + 1 end if end if </pre>	
---	--

Algorithm 3: UpdatePosition

<pre> If not AreInFinalPosition then x <- x + velocityX y <- y + velocity moving = not AreInFinalPosition else moving = false end if </pre>	
---	--

Algorithm 4: AreInFinalPosition

<pre> if x = destinyX and y = destinyY then return true else return false </pre>	
--	--

Node Main: <run>

<pre> while true then randomDirection updatePosition end while </pre>	For alive nodes, for mobility in the simulation field.
---	--

4.4.4 Output Logs

Following an example of simulation log file, for a static node and a mobile node, using RandomDirection mobility pattern. The simulation was performed for 30 seconds.

```

run:
Adicionado o nodo 100 a simulacao.
Adicionado o nodo 101 a simulacao.
ID[100] <Static> X: 0.0 Y: 0.0 Z:0.0
ID[101] <Mobile> X: 45.0 Y: 39.0 Z:0.0
RandomDirection: POS[45.00,39.00]Angle:0.00° Destiny [100.00,39.00]
RandomDirection: steps:3.0 dX: 55.00 dY: 0.00 velX:18.33 velY:0.00 |Vmax|:20.00
1;101;63.333;39.000;0.000;30.000
2;101;81.667;39.000;0.000;30.000
3;101;100.000;39.000;0.000;30.000
4;101;100.000;39.000;0.000;30.000
5;101;100.000;39.000;0.000;30.000
RandomDirection: POS[100.00,39.00]Angle:210.75° Destiny [34.44,0.00]
RandomDirection: steps:4.0 dX: -65.56 dY: -39.00 velX:-16.39 velY:-9.75 |Vmax|:20.00
6;101;83.611;29.250;0.000;30.000
7;101;67.222;19.500;0.000;30.000
8;101;50.833;9.750;0.000;30.000
9;101;34.444;0.000;0.000;30.000
10;101;34.444;0.000;0.000;30.000
11;101;34.444;0.000;0.000;30.000
RandomDirection: POS[34.44,0.00]Angle:123.87° Destiny [0.00,51.31]
RandomDirection: steps:6.0 dX: -34.44 dY: 51.31 velX:-8.61 velY:12.83 |Vmax|:20.00
12;101;25.833;12.827;0.000;30.000
13;101;17.222;25.654;0.000;30.000
14;101;8.611;38.481;0.000;30.000
15;101;0.000;51.308;0.000;30.000
16;101;0.000;51.308;0.000;30.000
17;101;0.000;51.308;0.000;30.000
RandomDirection: POS[0.00,51.31]Angle:359.39° Destiny [100.00,50.25]
RandomDirection: steps:6.0 dX: 100.00 dY: -1.06 velX:16.67 velY:-0.18 |Vmax|:20.00
18;101;16.667;51.131;0.000;30.000
19;101;33.333;50.954;0.000;30.000
20;101;50.000;50.777;0.000;30.000
21;101;66.667;50.600;0.000;30.000
22;101;83.333;50.423;0.000;30.000
23;101;100.000;50.246;0.000;30.000
24;101;100.000;50.246;0.000;30.000
25;101;100.000;50.246;0.000;30.000
RandomDirection: POS[100.00,50.25]Angle:121.17° Destiny [69.91,100.00]
RandomDirection: steps:3.0 dX: -30.09 dY: 49.75 velX:-10.03 velY:16.58 |Vmax|:20.00
26;101;89.969;66.831;0.000;30.000
27;101;79.937;83.415;0.000;30.000
28;101;69.906;100.000;0.000;30.000
29;101;69.906;100.000;0.000;30.000
30;101;69.906;100.000;0.000;30.000
Simulacao encerrada.
Aguardando a finalizacao da fila de processamento.socket closed
socket closed
Core finalizado.
simulador.Core Threads terminadas.
CONSTRUÍDO COM SUCESSO (tempo total: 36 segundos)

```

Figure 23. Example of a textual log file of WiNeS.

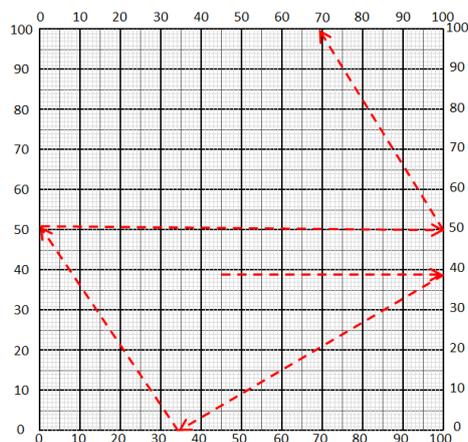


Figure 24. Example of node movement from the Log (Figure 23).

5 FINAL CONSIDERATIONS

The tasks involved in this work could be divided into three categories: preliminary study, software simulation and hardware validation. The preliminary study consisted in defining all the required background for the adjacent steps, from the mathematical models and algorithms up to what hardware platform will be employed. The software simulation includes setting up the simulation software in order to test the proposed algorithm as well as to estimate the gains of using the proposed localization technique. Finally, the hardware validation consists in two phases: (i) the platform assembling to the development of the firmware to run on it, and (ii) the performance analysis of the system in a real world scenario. This is a work in progress. This technical report gathers all the steps executed during eight months of research and related experimental analysis and studies.

6 REFERENCES

1. *Wireless sensor networks: a survey*. **Akyildiz, I., et al.** 4, 2002, Computer Networks, Vol. 38, pp. 393–422.
2. **Bai, Fan and Helmy, Ahmed.** *A Survey Of Mobility Models in Wireless Adhoc Networks*.
3. *Localization Schemes in Wireless Sensor Networks*. **Anita Panwar, Sh. Ashok Kumar.** Hamirpur, India : s.n., 2012. pp. 443-449.
4. **Guoqiang Mao, Barış Fidan.** *Localization Algorithms and Strategies for Wireless Sensor Networks*. New York : Information Science Reference, 2009. 978-1-60566-396-8.
5. **Mao, G. and Fidan, B.** *Localization Algorithms and Strategies for Wireless Sensor Networks*. New York, USA : Information Science Reference, 2009. p. 372 p.
6. **Parsa, K., Lasky, T. and Ravani, B.** Design and Implementation of a Mechatronic, All-Accelerometer Inertial Measurement Unit. *IEEE/ASME Transaction Mechatronics*. 2007, Vol. 12, 6, pp. 640-650.
7. **Mag, Sensor.** Accelerometer internal structure. [Online] <http://www.sensormag.com/sensors/acceleration-vibration/sonic-nirvana-mems-accelerometers-acoustic-pickups-musical-i-5852>.
8. **Gabaglio, V.** *GPS/INS Integration for Pedestrian Navigation*. Institut für Geodäsie und Photogrammetrie Zürich: . Zürich : ETH Zürich, 2003. Doctorate thesis.
9. *Market opportunities for advanced MEMS accelerometers and overview of actual capabilities vs. required specifications*. **Jean-Michel, S.** Monterey, CA, USA : s.n., 2004. Position Location and Navigation Symposium (PLANS2004). pp. 78-82.
10. **Hofmann-Wellenhof, B., Legat, K. and Wieser, M.** *Navigation: Principles of positioning and guidance*. s.l. : Springer, 2003. p. 427p.
11. *Web-based resources on GPS/INS integration*. **Lee, H., Hewitson, S. and Wang, J.** 3, 2004, GPS Solutions, Vol. 8, pp. 189-191.
12. *Hybrid localization using uwb and inertial sensors*. **Sczyslo, S., et al.** 2008. IEEE International Conference on Ultra-Wideband. pp. 89-92.

13. *Robot localization using RF and inertial sensors.* **Zmuda, M., Elesev, A. and Morton, Y.** 2008. IEEE National Aerospace and Electronics Conference (NAECON). pp. 343-348.
14. *Hybrid indoor tracking with bayesian sensor fusion of RF localization and inertial navigation.* **Fink, A. and Beikirch, H.** 2011. IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS) . pp. 823-827.
15. **Simon, D. and Chia, T.** Kalman Filtering with State Constrains: A Survey of Linear and Nonlinear Algorithms. *IET Control Theory and Applications.* 2010, Vol. 4, 8, pp. 1303-1318.
16. *On the fusion of inertial data for signal strength localization.* **Schmid, J., et al.** 2011. Workshop on Positioning Navigation and Communication (WPNC). pp. 7-12.
17. **Lee, S., et al.** Inertial Sensor-Based Indoor Pedestrian Localization with Minimum 802.15.4a Configuration. *IEEE Transactions on Industrial Informatics.* August 2011, Vol. 7, 3, pp. 455-466.
18. **Xue-bo, J., Jing-jing, D. and Jia, B.** Target Tracking of a Linear Time Invariant System Under Irregular Sampling. *International Journal Adv Robot Syst.* 2012, Vol. 9, p. 219.
19. **Jwo, D. J., Hu, C.-W. and Tseng, C.-H.** Nonlinear Filtering with IMM Algorithm for Ultra-Tight GPS/INS Integration. *International Journal Adv Robot Syst.* 2013, Vol. 10, p. 222.
20. **Jain, R.** *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* s.l. : Wiley, 1991. p. 685p.
21. *A flexible framework for modeling and simulation of multipurpose wireless networks.* **Bohrer, V., et al.** Montreal, Canada : s.n., 2013. International Symposium on Rapid System Prototyping (RSP). pp. 94-100.
22. **Flexgrid.** Flexgrid - Um Framework para o Projeto de Arquiteturas e Protocolos de Redes de Sensores Sem Fio. *Projeto FAPERGS/CAPES Edital Docfix nº 09/2012.* Porto Alegre, RS, Brasil : s.n., Outubro 2012.
23. *An overview of the OMNeT++ simulation environment.* **Varga, A. and Hornig, R.** Marseille, France : s.n., 2008. Proceedings of the 1st international conference on

Simulation tools and techniques for communications, networks and systems & workshops (Simutools'08). Vol. 60, pp. 1-10.

24. *An Energy Model for Simulation Studies of Wireless Sensor Networks using OMNeT++*. **Chen, F., et al.** 2, 2009, Praxis der Informationsverarbeitung und Kommunikation (PIK), Vol. 32, pp. 133–138.

25. **ns-2**. The Network Simulator ns-2: Documentation.

26. **ns-3**. ns-3 Network Simulator.

27. *Performance Evaluation of Castalia Wireless Sensor Network Simulator*. **Rastegarnia, A. and Solouk, V.** Budapest, Hungary : IEEE, 2011. 34th International Conference on Telecommunications and Signal Processing (TSP). pp. 111-115.

28. *A performance comparison of multi-hop wireless ad hoc network routing protocols*. **Broch, J., et al.** Dallas, Texas, USA : ACM, 1998. Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom98). pp. 1-13.

29. *A Survey of Simulation in Sensor Networks*. **Singh, C., Vyas, O. and Tiwari, M.** Vienna, Austria : IEEE, 2008. International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA). pp. 867-872.

30. **Misra, S. C., Woungang, I. and Misra, S.** *Guide to Wireless Sensor Networks*. s.l. : Springer, 2009.

31. **Matin, M. A. and Islam, M. M.** *Overview of Wireless Sensor Network*. [ed.] M. A. Matin. s.l. : InTech, 2012. InTech - Wireless Sensor Networks - Technology and Protocols.

32. **Misra, Sudip, Woungang, Isaac and Misra, Subhas Chandra.** *Guide to Wireless Sensor Networks*. s.l. : Springer, 2009.

33. *Energy-efficient communication protocol for wireless microsensor networks*. **Heinzelman, W., Chandrakasan, A. and Balakrishnan, H.** Maui, Hawaii, USA : IEEE, 2000. Proceedings of the 33rd Annual Hawaii International Conference on System Sciences. pp. 1-10.

34. *Simulating the Power Consumption of Large Scale Sensor Network Applications*. **Shnayder, V., et al.** Baltimore, Maryland, USA : ACM, 2004. Proceedings of

the 2nd international conference on Embedded networked sensor systems (SenSys'04). pp. 188-200.

35. *Accurate Prediction of Power Consumption in Sensor Networks*. **Landsiedel, O., Wehrle, K. and Götz, S.** Sydney, Australia : IEEE, 2005. Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II). pp. 37-44.

36. *Energy Consumption Model for Power Management in Wireless Sensor Networks*. **Wang, Q. and Yang, W.** San Diego, CA, USA : IEEE, 2007. 4th Annual IEEE Communication Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON '07). pp. 142-151.

37. *Towards a Realistic Energy Model for Wireless Sensor Networks*. **Kellner, S., et al.** Garmisch-Partenkirchen, Germany : IEEE, 2008. Fifth Annual Conference on Wireless on Demand Network Systems and Services (WONS 2008). pp. 97-100.

38. *An estimation of sensor energy consumption*. **Halgamuge, M. N., Zukerman, M. and Ramamohanarao, K.** 2009, Progress In Electromagnetics Research B, Vol. 12, pp. 259–295.

39. *Modeling of Node Energy Consumption for Wireless Sensor Networks*. **Zhou, H.-Y., et al.** 2011, Wireless Sensor Network, Vol. 3, pp. 18-23.

40. **Invensense**. *MPU-6000 and MPU-6050 Product Specification Revision 3.1*. Invensense Inc. Sunnyvale, CA, USA : s.n., 2011. Document number: PS-MPU-6000A-00.

APPENDIX I - ARDUINO PLATFORM TESTS

I2C Library - IC2.ino file

```

/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.

This software may be distributed and modified under the terms of the GNU
General Public License version 2 (GPL2) as published by the Free Software
Foundation and appearing in the file GPL2.TXT included in the packaging of
this file. Please note that GPL2 Section 2[b] requires that all works based
on this software must also be made publicly available under the terms of
the GPL2 ("Copyleft").

Contact information
-----

Kristian Lauszus, TKJ Electronics
Web      : http://www.tkjelectronics.com
e-mail   : kristianl@tkjelectronics.com
*/

const uint8_t IMUAddress = 0x68; // AD0 is logic low on the PCB
const uint16_t I2C_TIMEOUT = 1000; // Used to check for errors in I2C communication
uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop) {
    return i2cWrite(registerAddress, &data, 1, sendStop); // Returns 0 on success
}

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop) {
    Wire.beginTransmission(IMUAddress);
    Wire.write(registerAddress);
    Wire.write(data, length);
    uint8_t rcode = Wire.endTransmission(sendStop); // Returns 0 on success
    if (rcode) {
        Serial.print(F("i2cWrite failed: "));
        Serial.println(rcode);
    }
    return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
}

uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {
    uint32_t timeOutTimer;
    Wire.beginTransmission(IMUAddress);
    Wire.write(registerAddress);
    uint8_t rcode = Wire.endTransmission(false); // Don't release the bus
    if (rcode) {
        Serial.print(F("i2cRead failed: "));
        Serial.println(rcode);
        return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission
    }
    Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true); // Send a repeated start and then
    release the bus after reading

    for (uint8_t i = 0; i < nbytes; i++) {
        if (Wire.available())

```

```

        data[i] = Wire.read();
    else {
        timeOutTimer = micros();
        while (((micros() - timeOutTimer) < I2C_TIMEOUT) && !Wire.available());
        if (Wire.available())
            data[i] = Wire.read();
        else {
            Serial.println(F("i2cRead timeout"));
            return 5; // This error value is not already taken by endTransmission
        }
    }
}
return 0; // Success
}

```

I2C Library - MPU6050.ino file

```

/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.
This software may be distributed and modified under the terms of the GNU
General Public License version 2 (GPL2) as published by the Free Software
Foundation and appearing in the file GPL2.TXT included in the packaging of
this file. Please note that GPL2 Section 2[b] requires that all works based
on this software must also be made publicly available under the terms of
the GPL2 ("Copyleft").
Contact information
-----
Kristian Lauszus, TKJ Electronics
Web      : http://www.tkjelectronics.com
e-mail   : kristianl@tkjelectronics.com
*/
#include <Wire.h>
#include "Kalman.h" // Source: https://github.com/TKJElectronics/KalmanFilter
#define RESTRICT_PITCH // Comment out to restrict roll to ±90deg instead - please read:
http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf
Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;
/* IMU Data */
double accX, accY, accZ;
double gyroX, gyroY, gyroZ;
int16_t tempRaw;
double gyroXangle, gyroYangle; // Angle calculate using the gyro only
double compAngleX, compAngleY; // Calculated angle using a complementary filter
double kalAngleX, kalAngleY; // Calculated angle using a Kalman filter
uint32_t timer;
uint8_t i2cData[14]; // Buffer for I2C data
void setup() {
    Serial.begin(115200);

```

```

Wire.begin();
TWBR = ((F_CPU / 400000L) - 16) / 2; // Set I2C frequency to 400kHz
i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz Gyro filtering, 8 KHz
sampling
i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s
i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ±2g
while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers at once
while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope reference and disable sleep mode
while (i2cRead(0x75, i2cData, 1));
if (i2cData[0] != 0x68) { // Read "WHO_AM_I" register
    Serial.print(F("Error reading sensor"));
    while (1);
}
delay(100); // Wait for sensor to stabilize
/* Set kalman and gyro starting angle */
while (i2cRead(0x3B, i2cData, 6));
accX = (i2cData[0] << 8) | i2cData[1];
accY = (i2cData[2] << 8) | i2cData[3];
accZ = (i2cData[4] << 8) | i2cData[5];
// Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf eq. 25 and eq. 26
// atan2 outputs the value of -π to π (radians) - see http://en.wikipedia.org/wiki/Atan2
// It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
    double roll = atan2(accY, accZ) * RAD_TO_DEG;
    double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
    double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
    double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif
    kalmanX.setAngle(roll); // Set starting angle
    kalmanY.setAngle(pitch);
    gyroXangle = roll;
    gyroYangle = pitch;
    compAngleX = roll;
    compAngleY = pitch;
    timer = micros();
}
void loop() {
    /* Update all the values */
    while (i2cRead(0x3B, i2cData, 14));
    accX = ((i2cData[0] << 8) | i2cData[1]) + 2000.0;
    accY = ((i2cData[2] << 8) | i2cData[3]) + 3319.84;
    accZ = ((i2cData[4] << 8) | i2cData[5]) + 664.48;
    tempRaw = (i2cData[6] << 8) | i2cData[7];
    gyroX = (i2cData[8] << 8) | i2cData[9];
    gyroY = (i2cData[10] << 8) | i2cData[11];
}

```

```

gyroZ = (i2cData[12] << 8) | i2cData[13];
double dt = (double)(micros() - timer) / 1000000; // Calculate delta time
timer = micros();
// Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf eq. 25 and eq. 26
// atan2 outputs the value of -π to π (radians) - see http://en.wikipedia.org/wiki/Atan2
// It is then converted from radians to degrees
#ifdef RESTRICT_PITCH // Eq. 25 and 26
    double roll = atan2(accY, accZ) * RAD_TO_DEG;
    double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;
#else // Eq. 28 and 29
    double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
    double pitch = atan2(-accX, accZ) * RAD_TO_DEG;
#endif

double gyroXrate = gyroX / 131.0; // Convert to deg/s
double gyroYrate = gyroY / 131.0; // Convert to deg/s
#ifdef RESTRICT_PITCH
    // This fixes the transition problem when the accelerometer angle jumps between -180 and 180
    degrees
    if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {
        kalmanX.setAngle(roll);
        compAngleX = roll;
        kalAngleX = roll;
        gyroXangle = roll;
    } else
        kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter
    if (abs(kalAngleX) > 90)
        gyroYrate = -gyroYrate; // Invert rate, so it fits the restricted accelerometer reading
    kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);
#else
    // This fixes the transition problem when the accelerometer angle jumps between -180 and 180
    degrees
    if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {
        kalmanY.setAngle(pitch);
        compAngleY = pitch;
        kalAngleY = pitch;
        gyroYangle = pitch;
    } else
        kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate the angle using a Kalman
filter

    if (abs(kalAngleY) > 90)
        gyroXrate = -gyroXrate; // Invert rate, so it fits the restricted accelerometer reading
    kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter
#endif

gyroXangle += gyroXrate * dt; // Calculate gyro angle without any filter
gyroYangle += gyroYrate * dt;
//gyroXangle += kalmanX.getRate() * dt; // Calculate gyro angle using the unbiased rate
//gyroYangle += kalmanY.getRate() * dt;

```

```
    compAngleX = 0.93 * (compAngleX + gyroXrate * dt) + 0.07 * roll; // Calculate the angle using a
Complimentary filter
    compAngleY = 0.93 * (compAngleY + gyroYrate * dt) + 0.07 * pitch;
    // Reset the gyro angle when it has drifted too much
    if (gyroXangle < -180 || gyroXangle > 180)
        gyroXangle = kalAngleX;
    if (gyroYangle < -180 || gyroYangle > 180)
        gyroYangle = kalAngleY;
    /* Print Data */
#if 0 // Set to 1 to activate
    Serial.print(accX); Serial.print("\t");
    Serial.print(accY); Serial.print("\t");
    Serial.print(accZ); Serial.print("\t");
    Serial.print(gyroX); Serial.print("\t");
    Serial.print(gyroY); Serial.print("\t");
    Serial.print(gyroZ); Serial.print("\t");
    Serial.print("\t");
#endif
    Serial.print(roll); Serial.print("\t");
    Serial.print(gyroXangle); Serial.print("\t");
    Serial.print(compAngleX); Serial.print("\t");
    Serial.print(kalAngleX); Serial.print("\t");
    Serial.print("\t");
    Serial.print(pitch); Serial.print("\t");
    Serial.print(gyroYangle); Serial.print("\t");
    Serial.print(compAngleY); Serial.print("\t");
    Serial.print(kalAngleY); Serial.print("\t");
#if 0 // Set to 1 to print the temperature
    Serial.print("\t");
    double temperature = (double)tempRaw / 340.0 + 36.53;
    Serial.print(temperature); Serial.print("\t");
#endif
    Serial.print("\r\n");
    delay(2);
}
```