LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

MOBILE ROBOT LOCALIZATION USING SONAR RANGING AND WLAN INTENSITY MAPS

Bachelor's thesis
Supervisor: D.Sc. Ville Kyrki

Lappeenranta, February 21, 2007

Janne Laaksonen
Janne.Laaksonen@lut.fi

ABSTRACT

Lappeenranta University of Technology

Department of Information Technology

Janne Laaksonen

Mobile robot localization using sonar ranging

and WLAN intensity maps

Bachelor's thesis

2007

50 pages, 19 figures, 2 tables and 1 appendix.

Supervisor: D.Sc. Ville Kyrki

Keywords: Mobile, Robot, Localization, MCL, WLAN

Main goal of this thesis was to implement a localization system which uses sonars and WLAN intensity maps to localize an indoor mobile robot. A probabilistic localization method, Monte Carlo Localization is used in localization. Also the theory behind probabilistic localization is explained. Two main problems in mobile robotics, path tracking and global localization, are solved in this thesis.

Implemented system can achieve acceptable performance in path tracking. Global localization using WLAN received signal strength information is shown to provide good results, which can be used to localize the robot accurately, but also some bad results, which are no use when trying to localize the robot to the correct place. Main goal of solving ambiguity in office like environment is achieved in many test cases.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto

Tietotekniikan osasto

Janne Laaksonen

Mobile Robot Localization using sonar ranging and WLAN intensity maps

Kandidaatintyön loppuraportti

2007

50 sivua, 19 kuvaa, 2 taulukkoa ja 1 liite.

Ohjaaja: TkT Ville Kyrki

Hakusanat: Mobiili, Robotti, Paikannus, MCL, WLAN

Tämän projektin päätarkoitus oli toteuttaa paikannusjärjestelmä, joka käyttää hyväksi kaikuluotaimia sekä WLAN-kuuluvuuskarttoja. Paikannusjärjestelmää käyttää sisätiloissa toimiva mobiilirobotti. Todennäköisyyteen perustuva menetelmä, Monte Carlo paikannus tulee toimimaan paikannusmenetelmänä. Teoria tämän menetelmän takana tullaan myös selvittämään. Kaksi pääongelmaa mobiilirobottien paikannuksessa, lokaali ja globaali paikannus, tullaan ratkaisemaan tässä projektissa.

Toteutettu järjestelmä pystyy ratkaisemaan lokaalin pakannuksen ongelman hyväksyttävästi. Globaali paikannus, jossa käytetään hyväksi WLAN-signaalin tasoinformaatiota, antaa hyviä mutta myös joitain huonojakin tuloksia. Hyviä tuloksia voidaan käyttää tarkasti paikantamaan robotti, mutta huonoilla tuloksilla näin ei voida tehdä. Toimistoympäristössä globaali paikannus pystyy kuitenkin erottamaan eri alueet toisistaan useissa tapauksissa.

Contents

ABBREVIATIONS AND SYMBOLS

IDE Integrated Development Environment
MCL Monte Carlo Localization
SIP Server Information Packet
RSS Received Signal Strength
WLAN Wireless Local Area Network

η Normalizing factor

1 INTRODUCTION

This section introduces background and objectives for this thesis'. Also the structure of the entire thesis is explained.

1.1 Background

This thesis would not be possible without a mobile robot. Robot used in this case is Pioneer 3-DX. This robot offers opportunities to study different fields found within mobile robotics. This thesis concentrates on localization of a robot.

Pioneer 3-DX comes with its own navigation and localization system, SONARNL. However it is closed source and does not allow any other sensors than the sonars on the robot to be used with the localization. Use of common WLAN(Wireless Local Area Network) to provide information for the robot is also one of the focus points of this thesis. So to be able to study WLAN as a sensor for the robot, implementing a new localization system is necessary.

1.2 Objectives and Restrictions

First objective is to implement an equivalent open system to the closed SONARNL localization system, which uses the sonars on the Pioneer 3-DX robot. This system uses a 2D line map of the environment where the Pioneer 3-DX robot can move. With this, the system is limited to work on a 2D plane. On a 2D plane a robot can have 3 degrees of freedom, (x, y, θ), where x, y are the coordinates in the plane and θ is the orientation of the robot.

Second objective is to implement global localization in the localization system using WLAN RSS(Received Signal Strength) maps. An earlier project handled the collection of the data needed for the global localization[2].

Furthermore, one of the key objectives is to study how can a system, like localization, be built for Pioneer 3-DX robot. Software that comes with Pioneer 3-DX robot is quite complex and requires extensive study of the existing libararies and their functions.

The localization system is based on Monte Carlo Localization. This method was selected because of its ease of implementation and its ability to update multiple pose hypothesis. This is crucial in global localization as multiple locations may look the same from the perpective of the robot sonars.

The system is restricted to work in a 2D environment, which is suitable for office environments. A map of the Lappeenranta University of Technology, phase 6, floor 5 is used as a test map, this area is combined office and laboratory environment.

C++-language will be used to implement the system. This is because ARIA software library, which is used as an interface to Pioneer 3-DX robot, is also implemented in C++.

1.3 Structure of the Thesis

Section 2 presents an introduction to the field of mobile robots. It also give motivation why different localization methods are needed in conjunction with mobile robots.

Section 3 explains the theory behind localization techniques that rely on probability models. Focus is on Monte Carlo localization as it forms the basis for the method used in practical implementation.

Section 4 presents the system on which the localization system is built. Both hardware and software aspects are considered. Section 4 also describes the implemented localization system.

In Section 5, experiments and the results from the experiments are explained in detail. The results are also interpreted to explain their meaning for the localization system.

This thesis ends in conclusion, which can be found in section 6. Further development of the localization system and related work is discussed.

2 INTRODUCTION TO MOBILE ROBOTS

In this section, different kinds of mobile robots are discussed as well as mobile robot locomotion and the kinds of sensors used to sense the environment. An introduction to localization of mobile robots will also be given. Different techniques and problems will be described.

2.1 Overview of Mobile Robots

Robots have been used in industry since 1961. First industrial robot was Unimate[3], this robot was simply an arm which was used in welding cars and moving hot die-castings. Today industrial robots are used widely in repetitive and accuracy demanding tasks, such as welding cars or placing components on printed circuit boards. However these robots are immobile.

Mobile robots can be used for exploration and transportation. This expands the use of robots considerably. However mobility itself causes problems for mobile robots. Immobile robots do not need to localize as the robot is always at the same place. A mobile robot, on the other hand, must know its correct position in the real world to be able to make rational decisions about its actions, for example reaching a certain location. Fulfilling this demand is one of the key problems in the field of mobile robots.

2.1.1 Mobile Robot Locomotion

There are three basic forms of robot locomotion on the ground: wheeled, legged or tracked. This thesis will concentrate on the wheeled locomotion as the Pioneer 3-DX is a wheeled robot. There are many different configurations for a wheeled robot[4, pp. 34-36]. Wheeled robots work best on a flat surface where a wheel is efficient. It also easy to calculate how much a wheel has traveled on this kind of surface. On the other hand, wheeled locomotion is not as effective on rough surfaces, where elevation and texture of the ground changes constantly, which can cause wheel slip. This is where a legged robot is more suitable. Configurations for legged robots vary from one leg upwards, although six-legged robots are popular as they provide static stability, which means they are stable even when moving[4, pp. 21-30]. The downside is that legged robots are not as efficient

on flat surfaces as wheeled robots. Tracked robots offer better mobility on rough terrain than wheeled robots and treads are more efficient on a flat surface than legs. Problems with tracked robots arise from the fact that the treads slip when the robot turns, making it difficult to predict the orientation and position of the robot after movement[4, p. 42].

2.1.2 Mobile Robot Sensors

To be able to sense the environment, a mobile robot needs some kind of sensors. Sensors can be divided to active and passive sensors as well as proprioceptive and exteroceptive sensors[4, p. 89]. An active sensor sends a signal and waits for it to deflect back from obstacles. After this it can determine the distance of the obstacle by measuring the time difference between sending the signal and receiving it.

Mobile robots often use sonars or lasers as active sensors, which emit sound or light, respectively. Both of these sensors have advantages and disadvantages compared to each other, but the main advantage for laser is its accuracy and sonar's advantage is its price compared to laser.

Passive sensors only receive signals from the environment. There are many kinds of passive sensors which are used with mobile robots. A camera is a good example of a passive sensor, but there are also many other types of passive sensors, for example passive sonars.

Proprioceptive sensors measure data from the robot. This data can come from variety of sources such as the battery or motors. Exteroceptive sensors collect data from the environment and extract features from the data received, for example distance from an object if the sensor is a sonar.[4, p. 89]

2.2 Localization of Mobile Robots

This section presents an introduction to the field of mobile robot localization. Different localization methods are explained and fundamental problems of localization are discussed.

2.2.1 Terminology

Next, we define some terminology for robot localization. A pose describes the position and the orientation of the mobile robot. In this thesis we will be dealing with poses in the form of (x, y, θ), where x, y describe the position of the robot in 2D plane and θ describes the orientation of the robot. This means that there are 3 degrees of freedom or DOF. Degrees of freedom is another term used in robotics and it tells in how many ways the robot can move. For example, the robot presented in this thesis is capable of travelling in a 2D plane to any point, which is 2 DOF and is capable also to orient itself to any orientation in any position. This together forms a 3 DOF system. Belief is also one important concept used in localization. Belief represents the pose or poses where the robot thinks it is, which is not the true pose in the real world. Belief leads us to the concept of hypothesis, which is used also in statistics. In this context hypothesis means possible pose of the robot that we are localizing. There can be multiple or just a single hypothesis depending on the method used. [4, pp. 194-200]

2.2.2 Odometry

The most basic way for a robot to know its position is to use some kind of measurement of how the robot moves. An analogue for this is how a human can calculate distances by measuring his or her step distance and then calculating how many steps the trip took. With this we can see a problem, the step distance is not always the same and over a long distance small error in step distance is cumulative. This same problem occurs with robots, there is always noise in the sensors which measure the distance from the motors that run the robot, and thus we cannot be sure where the robot is after traveling a long distance. This is one of the key reasons why localization systems are needed.

Wheeled robots use rotary encoders to measure the distance traveled by each wheel. This is called odometry and with odometry the position and the orientation of the robot can be calculated. Odometry is fine for short distances and is often used as a base for further calculations in localizing the robot.[5, p. 23]

As discussed above, odometry does not provide reliable information on location over long distances. Localization has been researched quite intensively in the last decade[4, p. 181], and there are many different methods of localization, which work under different assumptions and in different environments.

2.2.3 Maps

Localization methods require a map. Without a map, no meaningfull localization can be done. Even though odometry can be used to calculate pose of the robot in relation to its starting position, but without a map, the real world position cannot be deducted. The same is true for GPS-like(Global Positioning System) systems. They provide the real world position, but without map this position has no use, as the position does not describe the environment. There are multiple types of maps, which are suitable for use with different methods of localization.

Two of the common types are feature-based maps and occupancy grid maps. Feature-based maps contain the features of the map, usually these maps contain lines and points to describe walls and other features found in the environment that the map depicts. Occupancy grid maps have a different approach, instead of just having the features, i.e. walls, the map is divided into a grid over the whole map, and the cells of the grid are marked either occupied or free.[4, pp. 200-210]

Advantage of the feature map compared to the occupancy grid are that it only contains the needed features. Occupancy grid however has the advantage that it shows immediately if a certain location on the map is accessible and free. For example a position inside the wall is occupied, so the robot cannot be in such a location. With feature maps, some extra processing is needed to get same kind of information out of the map and into use.

2.2.4 Localization techniques

One of the most successful localization methods has been probabilistic localization[5]. Main reason why it has been successful is, that it does not require any changes to the environment where the robot moves and it is able to handle the uncertainty which is always present in real world such as unpredictable environments and noisy sensors[5, p. 4]. Details of probabilistic localization will be discussed later on as this thesis uses one of the methods found within probabilistic localization, Monte Carlo localization.

One class of localization methods is based on beacons or landmarks[4, p. 245,248]. Common factor here is that they both require modifications to the environment. Beacons are usually active, which means that they emit signals into the environment from a well known location. When a robot receives these signals from multiple beacons, it can determine its

location in the environment. GPS(Global Positioning System) can be considered a beacon system as the satellites send signals and the signal is received on Earth. GPS can be used in outdoor mobile robot systems where the signal can be used. GPS signal is too inaccurate to be used in indoor localization. When using a single GPS receiver, accuracy of the received position is within 1 to 10 meters from the real position.[6]. This is not acceptable in indoor environments. With landmarks, the modifications to the environment can be quite extensive as the robot can only localize where there are landmarks visible to the robot's sensors[4, p. 245]. Of course it is possible to combine this method to the previously mentioned probabilistic localization[5, p. 260] but triangulation is also used [7],[8].

Route-based localization is an inflexible way of localizing a robot, but it is still used in industry[9]. It is basically equivalent to a railway system, where the rails are marked with something that the robot can detect. Rails can be physical rails or they can be painted pathways or hidden wires which the robot can detect with its sensors. The advantage for this kind of localization is that the position of the robot is well-known, because it follows the route. The downside for this method is that the robot can only move by following the fixed route. [4, p. 249]

2.2.5 Fundamental problems of localization

All abovementioned methods are different solutions to the problems which robot localization presents. There are three main problems in the field of robot localization: position tracking, global localization and kidnapped robot problem [5, pp. 193-194]. Position tracking is the easiest of these problems and kidnapped robot problem is the most difficult. In position tracking the initial pose of the robot is known and the goal is to track the position of the robot while the robot moves. In global localization the initial position is not known and it must be determined by the robot before it can move back to the path tracking problem which is much simpler. Kidnapped robot problem is similar to the global localization but in addition it includes the problem of detecting the kidnapping and, after the kidnapping has been detected, using global localization to localize the robot to correct pose. Kidnapping in this context usually means that the robot is moved so that it cannot detect the movement, for example with odometry.

3 PROBABILISTIC ROBOT LOCALIZATION

In this section motivation for using probabilistic localization will be given and different methods based on probabilistic localization are explained, especially Monte Carlo localization. Previous robot localization systems which are based on probabilistic localization will also be described.

3.1 Probabilistic Framework for Localization of Mobile Robots

As mentioned in the section 2.2, probabilistic methods for localization have been quite successful in many applications, [10], [11], [12], [13]. One of the underlying reasons for the success of probabilistic localization is that it can tolerate the uncertainties of the real world, which is a direct result of using statistical techniques which operate on probabilities.

3.1.1 Bayes filter

One of the most important concepts found in probabilistic robot localization is the Bayes filter. Almost all the probabilistic methods used in localization are derivates of the Bayes filter[5, pp. 26-27] and are approximates of it.

1: **Algorithm Bayes_filter(**$bel(x_{t-1}), u_t, z_t$**):**
2: **for** all x_t **do**
3: $\quad \overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$
4: $\quad bel(x_t) = \eta \quad p(z_t|x_t)\overline{bel}(x_t)$
5: **end for**
6: return $bel(x_t)$

Figure 1: Bayes filter[5, p. 27]

In figure 1 we can see the algorithm for Bayes filter update rule. Bayes filter update rule algorithm has three parameters. $bel(x_{t-1})$ is the belief from previous time step, u_t is the control, which updates the belief state and finally z_t, which is the measurement from the environment. This algorithm has two steps in it: control update and measurement update. Control update step can be seen in line 3. It updates the belief by using the previous belief and the control parameter. When we have the new belief state from the control update step, we can use the measurement update step in line 4 to calculate weights for the belief

state. η is simply a normalizing constant to force the weight between $[0, 1]$. After this, the new belief state has been calculated and it can be returned.

The update rule is only a single iteration of the Bayes filter. Bayes filter is run recursively as the result of the previous calculation is used in the next and this continues as time goes on. This same mechanism can also be seen in the derivates of the Bayes filter.

It is also important to mention that most of the localization methods based on the Bayes filter assume that each time step is independent of each other. This assumption is called the Markov assumption. These assumptions are not always true in the real world. For example, people around the robot may cause different measurements than were expected for multiple sensors. This means that the measurements are not independent, neither spatially nor temporally. However Bayes filter can tolerate these violations quite well, which means that all the possible disruptions need not to be modelled.[5, p. 33]

There are two kinds of filters which are based on the Bayes filter, gaussian filters and nonparametric filters. Gaussian filters represent beliefs as normal distributions. This is good for position tracking, where prior knowledge of the robot position is available but for global localization gaussian filters cannot be used as effectively as they cannot represent arbitrary distributions. However, Gaussian filters are popular in probabilistic robotics. Methods such as Kalman filtering are based on this type of filter.[5, pp. 39-40,85]

Nonparametric filters are more suitable for situations like global localization where the distribution of hypotheses cannot be represented by normal distributions. This is done by approximating the distribution of the hypotheses by a finite amount of elements, which allows representing of any kind of distribution. Nonparametric filters allow to choose between computation time and accuracy of the approximation by changing the amount of elements used.[5, pp. 85-86]

3.1.2 Gaussian filters

As mentioned above, one popular Gaussian filter method is the Kalman filter[5, p. 61]. Basic Kalman filter has been modified in many ways. Extended Kalman filter and unscented Kalman filter are examples of this. Kalman filter was invented by Swerling[14] and Kalman[15] in the end of 1950's. When using Kalman filter methods for localization, they use mean μ_t and covariance Σ_t of a Gaussian to represent the belief at each time step. Thus Kalman filter cannot represent more than one hypothesis at a time. However there

1: **Algorithm Particle_filter**(X_{t-1}, u_t, z_t)**:**
2: $\overline{X}_t = X_t = \emptyset$
3: **for** $m = 1$ to M **do**
4: sample $x_t^{[m]}$ $p(x_t|u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t|x_t^{[m]})$
6: $\overline{X}_t = \overline{X}_t + \left\langle x_t^{[m]}, w_t^{[m]} \right\rangle$
7: **end for**
8: **for** $m = 1$ to M **do**
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to X_t
11: **end for**
12: return X_t

Figure 2: Particle filter[5, p. 98]

are modifications to the extended Kalman filter, which allow the use of multiple Gaussians to represent more than one hypothesis at a time in localization[5, pp. 218-219].

Extended Kalman filter is used in localization, because it has important changes to it compared to the Kalman filter. Kalman filter requires linear transforms which is not the case when using a robot, except in trivial cases. Extended Kalman filter is changed so that it does not require linear transforms and it can be used in localization.[5, p. 54] Unscented Kalman filter can also be used in localization of mobile robots[5, p. 220].

3.1.3 Nonparametric filters

Particle filter is one of the nonparametric filters. It is used in the practical work of this thesis as it forms the basis for the Monte Carlo Localization. Particle filter was first introduced by Metropolis and Ulam[16] and it is used in many fields including artificial intelligence and computer vision[5, p. 115]. The algorithm for particle filter can be seen in figure 2. In practice the particle filter algorithm is easy to implement, which can be seen from the algorithm of Monte Carlo localization, which will be introduced in section 3.2.

As can be seen from the figure 2, the algorithm follows the Bayes filter structure. First in line 4 the controls u_t, given as a parameter, are processed. After that the weights from the measurements z_t are calculated in line 5. In particle filter algorithm the belief $bel(x_{t-1})$ from Bayes filter is represented by particles which are denoted by X_t. Control and measurement updates are done for each particle individually.

However starting from line 8, the particle filter algorithm does something which cannot be seen in the Bayes filter. This is called resampling. Resampling is a very integral part of this algorithm as it favors the particles with high weights, which in the context of robot localization would mean that the particles near the correct pose of the robot are weighted more, which of course is the desired outcome. Resampling takes the particles from the set $\overline{X}_t$ and places the particles with high weights into the set X_t, which can then be returned.

Histogram filter is also one of the nonparametric filters. It functions as a histogram, but instead of 1D histogram commonly found with image analysis, it represents the state space by regions, each of which has a single probability value. Dimensions of the regions correspond to the DOF of the system. This discretizes the state space and this is called discretized Bayes filter[5, p. 86].

3.2 Monte Carlo Localization

Monte Carlo localization(MCL) was first developed by Dellaert et al.[11] and Fox et al.[10]. They took the particle filter method used in other areas such as computer vision and applied the method to localization of mobile robots[11]. Monte Carlo localization has become one of the most popular methods in localization, because it offers solutions to wide variety of problems in localization[5, p. 250]. Global localization has especially been one of the key points with MCL, which is something that Kalman filter methods cannot do as easily, because they are Gaussian filter methods.

1: **Algorithm MCL**(X_{t-1}, u_t, z_t, m)**:**
2: $\overline{X}_t = X_t = \emptyset$
3: **for** $m = 1$ to M **do**
4: $\quad x_t^{[m]} =$**sample_motion_model**$(u_t, x_{t-1}^{[m]})$
5: $\quad w_t^{[m]} =$**sample_measurement_model**$(z_t, x_t^{[m]}, m)$
6: $\quad \overline{X}_t = \overline{X}_t + \left\langle x_t^{[m]}, w_t^{[m]} \right\rangle$
7: **end for**
8: **for** $m = 1$ to M **do**
9: $\quad$ draw i with probability $\alpha\ w_t^{[i]}$
10: $\quad$ add $x_t^{[i]}$ to X_t
11: **end for**
12: return X_t

Figure 3: Monte Carlo Localization[5, p. 252]

The algorithm for Monte Carlo localization can be seen in figure 3. It almost identical

with the particle filter algorithm from figure 2. The only things that have changed are the lines 4 and 5 and the use of map m in the algorithm which ties the particle filter with localization. The use of motion and measurement models also make the particle filter more concrete, because they can be calculated with the data from the robot movement and sensors.

MCL has also been in use in many robot systems[17], [18], [19] after it was published in 1999. The basic MCL has been modified in many ways. One of these modifications to MCL is adding random samples, which gives better results[5, pp. 256-261]. This method has also been implemented in the practical system of this thesis. Method of adapting sample set size within MCL is called KLD-Sampling[5, pp. 263-267], this also improves the localization result and lessens the computation by statistically calculating error in localization and keeping the error within defined limits. Other modifications include Uniform MCL[17] and Extended MCL[19].

3.2.1 Movement models

The movement model used in the lpractical localization system is based on the odometry of the robot and the model is called odometry motion model. Another motion model, based on the translational and rotational velocities of the robot, is called the velocity motion model. Odometry model is usually more accurate[5, p. 121], which is why it was chosen in for the localization system. Motion model corresponds to the control update found in the Bayes filter in figure 1. Motion models process the control u_t and output the posterior probability $p(x_t|u_t, x_{t-1})$.

In figure 4 we can see the odometry model. It uses two rotations, δ_{rot1} and δ_{rot2}, and one translation, δ_{trans}, to describe the motion between any two points[5, p. 134]. Algorithm for implementing the odometry motion model can be seen in figure 5. This algorithm is based on sampling from the distribution instead of calculating the probability for a sample. So instead of giving the whole posterior probability distribution, this algorithm gives a sample from the posterior probability distribution.

Sample_motion_model_odometry algorithm is divided into three logical parts. The first part is in lines 2-4. This part calculates the variables $\delta_{rot1}, \delta_{rot2}$ and δ_{trans} from the control u_t. u_t in this case contains a vector $(\overline{x}_{t-1}\ \overline{x}_t)^T$, where $\overline{x}_{t-1} = (\overline{x}\ \overline{y}\ \overline{\theta})$ and $\overline{x}_t = (\overline{x}'\ \overline{y}'\ \overline{\theta}')$. The vector contains the poses estimated by the internal odometry of the robot from previ-

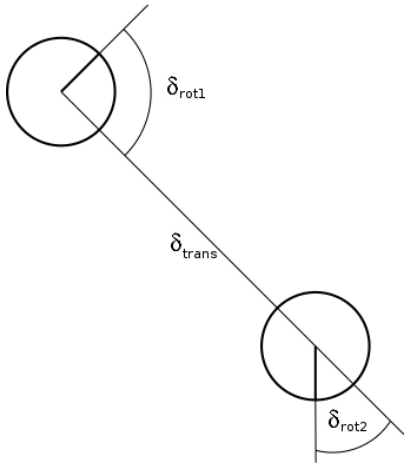

Figure 4: Odometry model

1: **Algorithm sample_motion_model_odometry**(u_t, x_{t-1})**:**
2: $\delta_{rot1} = \overline{y}^{'} - \overline{y}, \overline{x}^{'} - \overline{x}) - \overline{\theta}$
3: $\delta_{trans} = \sqrt{(\overline{x} - \overline{x}^{'})^2 + (\overline{y} - \overline{y}^{'})^2}$
4: $\delta_{rot2} = \overline{\theta}^{'} - \overline{\theta} - \delta_{rot1}$
5: $\hat{\delta}_{rot1} = \delta_{rot1}$−**sample**$(\alpha_1|\delta_{rot1}| + \alpha_2\delta_{trans})$
6: $\hat{\delta}_{trans} = \delta_{trans}$−**sample**$(\alpha_3|\delta_{trans}| + \alpha_4(|\delta_{rot1}| + |\delta_{rot2}|))$
7: $\hat{\delta}_{rot2} = \delta_{rot2}$−**sample**$(\alpha_1|\delta_{rot2}| + \alpha_2\delta_{trans})$
8: $x^{'} = x + \hat{\delta}_{trans}\cos(\theta + \hat{\delta}_{rot1})$
9: $y^{'} = y + \hat{\delta}_{trans}\sin(\theta + \hat{\delta}_{rot1})$
10: $\theta^{'} = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$
11: return $x_t = (x^{'}, y^{'}, \theta^{'})^T$

Figure 5: Odometry motion model algorithm[5, p. 136]

ous time step and current timestep.[5, p. 136]

Second part is in lines 5-7. This part adds noise to the variables $\delta_{rot1}, \delta_{rot2}$ and δ_{trans}. This is done because we do not assume that measurements done by the robot are noise free, which is true in practical applications. The noise is calculated using a distribution with zero mean. Variance of the noise is dependent on the parameters α_1, α_2, α_3 and α_4. α_1 and α_2 affect the noise in rotation and α_3 and α_4 affect the noise in translation. These parameters are robot specific and they have to be estimated.

The final part, found in lines 8-10, apply the noisy translation and rotations from $\hat{\delta}_{rot1}$, $\hat{\delta}_{trans}$ and $\hat{\delta}_{rot2}$. The end result is that the sample position is distributed around the original pose, which was taken from the robot's internal odometry. When this model is used on multiple samples, we get a cluster of samples representing the probability distribution.

3.2.2 Measurement models

The measurement model for sonar used in the localization system is called the beam range finder model. This model uses the difference of measured distance and the distance calculated from the map m to assign a probability for each measurement from the robot sensors, either from laser or sonar. Figure 6 shows the algorithm. Base for the algorithm has been proposed by Thrun et al.[5, p. 158], but it has been modified slightly, the modification can be seen in lines 4-7.

1: **Algorithm beam_range_finder_model**(z_t, x_t, m)**:**
2: $q = 1$
3: **for** $k = 1$ to K **do**
4: compute z_t^{k*} for the measurement z_t^k using ray casting
5: **if** $z_t^{k*} \neq$ maximum beam range **then**
6: $p = z_{hit} \cdot p_{hit}(z_t^k|x_t, m) + z_{short} \cdot p_{short}(z_t^k|x_t, m) + z_{max} \cdot p_{max}(z_t^k|x_t, m) + z_{rand} \cdot p_{rand}(z_t^k|x_t, m)$
7: **else**
8: $p = z_{short} \cdot p_{short}(z_t^k|x_t, m) + (1 - z_{short} - z_{rand}) \cdot p_{max}(z_t^k|x_t, m) + z_{rand} \cdot p_{rand}(z_t^k|x_t, m)$
9: **end if**
10: q = q * p
11: **end for**
12: return q

Figure 6: Beam range finder model algorithm

In the beam range finder model algorithm, the parameters contain the sensor readings, z_t, a pose, x_t and the map m, the map is in this case a feature based map. K in line 3 denotes the number of sensor readings in z_t, so the body of the loop is applied for each individual sensor reading. In line 4, the artificial distance z_t^{k*} from the pose, x_t, is calculated using ray casting. This is done by taking the pose and casting a ray from it to same direction where the real sensor reading z_t^k was obtained. Then using the map m, we can calculate where the ray intersects with the map. This way we get the distance from the ray origin to the map. If z_t^{k*} is at maximum beam range, it is considered as a special case. Effect of this is that we expect a maximum range reading from z_t^k. This lowers the probability of readings that were not at maximum range. This is called negative information[5, p. 231] as the model can use the readings that were not correct, instead of just readings that were correct.

The probability of the measurement is calculated by using 4 different probability distributions, p_{hit}, p_{short}, p_{max} and p_{rand}. Distribution p_{hit} is defined by:

$$p_{hit}(z_t^k|z_t, m) = \begin{cases} \eta\ N(z_t^k; z_t^{k*}, \sigma_{hit}^2) & \text{if} \quad 0 \leq z_t^k \leq z_{max} \\ 0 \quad \text{otherwise} \end{cases}$$

While z_t^k remains between 0 and maximum beam range z_{max}, probability $p_{hit}(z_t^k|z_t, m)$ is defined by Gaussian distribution:

$$N(z_t^k; z_t^{k*}, \sigma_{hit}^2) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}}\ e^{-\frac{1}{2}\frac{(z_t^k - z_t^{k*})^2}{\sigma_{hit}^2}}$$

The normalizing factor η is defined by the integral of the Gaussian distribution:

$$\eta = \left(\int_0^{z_{max}} N(z_t^k; z_t^{k*}, \sigma_{hit}^2)\ dz_t^k \right)^2$$

This distribution models the inherent sensor noise of the measurements. It is used to give high probability to correct ranges measured from the map.

Distribution p_{short} is an exponential distribution:

$$p_{short}(z_t^k|z_t, m) = \begin{cases} \eta\ \lambda_{short}\ e^{-\lambda_{short} z_t^k} & \text{if} \quad 0 \leq z_t^k \leq z_t^{k*} \\ 0 \quad \text{otherwise} \end{cases}$$

Normalizing factor η is once again defined by the integral of the distribution. When derived, it is in following form:

$$\eta = \frac{1}{1 - e^{\lambda_{short} z_t^{k*}}}$$

Probability distribution p_{short} models unexpected objects which are not visible in the map m. These could be people or other moveable objects. Unexpected objects can only cause measurements which are shorter than expected. This can be seen from the definition of the distribution as it only gives probabilities when the measured distance z_t^k is smaller than the distance z_t^{k*} obtained by ray casting.

Distribution p_{max} is a point distribution defined by:

$$p_{max}(z_t^k|z_t, m) = \begin{cases} 1 & \text{if} \quad z = z_{max} \\ 0 & \text{otherwise} \end{cases}$$

This point distribution is for failed sensor measurements. This means that the beam emitted from the sensor has deflected from objects, like walls, in a way that the beam is not received by the sensor, giving maximum range of the sensor as a result. This means that the measurement could be right, even though the range received from the sensor is not. This phenomen is modeled by the point distribution.

Finally the distribution for p_{rand} is given by:

$$p_{rand}(z_t^k|z_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{if} \quad 0 \leq z_t^k < z_{max} \\ 0 & \text{otherwise} \end{cases}$$

This uniformly distributed probability distribution is for random readings. This simply means that this distribution models all the other phenomena which are not covered by the other 3 distributions.

Variables σ and λ for Gaussian distribution p_{hit} and exponential distribution p_{near} can be selected freely. However setting the variables require knowledge of the environment and robot, so that the variables are chosen properly. In this case, σ controls how much noise the robot and environment causes to the sensor readings and λ tells how static or dynamic the map is, for example if the environment is heavily populated. The final probability distribution is shown in figure 7.

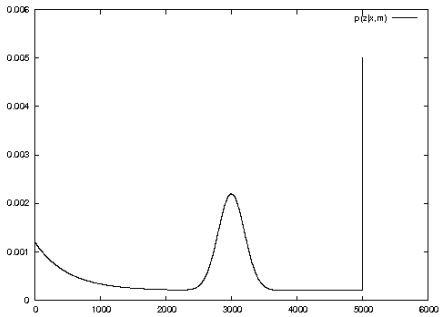

Figure 7: Beam range finder model probability distribution

Algorithm in figure 6 is used with beam type of sensors, sonars or lasers for example. However the other part of this thesis was to study how the WLAN intensity, or RSS, maps could be used in conjunction with the sonar localization. For that a new algorithm was designed. The algorithm can be seen in figure 8. The algorithm was designed after the beam range finder model algorithm, but changed so that the characteristics of WLAN can be used inside the algorithm.

1: **Algorithm WLAN_model**(z_t, x_t, m)**:**
2: $q = 1$
3: **for** $k = 1$ to K **do**
4: calculate z_t^{k*} for the measurement z_t^k using m
5: $p = z_{hit} \cdot p_{hit}(z_t^k|x_t, m) + z_{near} \cdot p_{near}(z_t^k|x_t, m) + z_{rand} \cdot p_{rand}(z_t^k|x_t, m)$
6: q = q * p
7: **end for**
8: return q

Figure 8: WLAN model algorithm

Much of the algorithm in figure 8 is still similar to the algorithm in figure 6. Most significant difference is that z_t^{k*} is not calculated by ray casting, instead the difference between the measured value z_t^k and z_t^{k*} is used as the map m contains the values directly. Other difference is that the algorithm uses only 3 distributions, p_{hit}, p_{near} and p_{rand}.

Only p_{near} distribution is different from the beam_range_finder_model algorithm. This distribution is Rayleigh distribution which has been found approriate with the use of WLAN RSS measurements, when a human is blocking the signal from WLAN access point[20]. This kind of event is quite common in an office environment. p_{near} distribution is defined by:

$$p_{rand}(z_t^k|z_t, m) = \begin{cases} \eta \frac{(z_t^k - z_t^{k*})\, e^{\frac{-(z_t^k - z_t^{k*})^2}{2\sigma^2}}}{\sigma^2} & \text{if} \quad 0 \leq z_t^k < z_t^{k*} \\ 0 \quad \text{otherwise} \end{cases}$$

Normalizing factor η in this case is defined as:

$$\eta = 1 - e^{\frac{-(z_{max} - z_t^{k*})^2}{2}}$$

the same problem of choosing the proper parameter values for different distributions occurs with the WLAN model. In this case σ in p_{hit} has the same effect as with beam range

finder model, it controls how much noise the system can tolerate, either from the environment, like multi-path effect, or from the sensor itself. σ in p_{near} handles how much degration in RSS can still be considered as a correct measurements. Usually this kind of degration comes from people, who move around and absorb the signal emitted from the WLAN access points. So by adjusting this parameter, we can take into account the density of people disrupting the WLAN signal in the robot environment. Again the final WlanModel probability distribution is shown in figure 9.

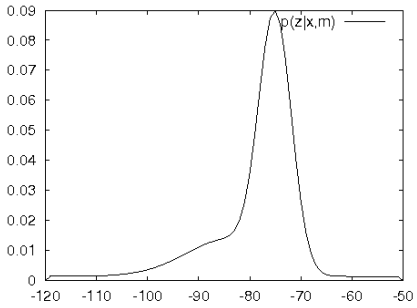

Figure 9: Wlan model probability distribution

4 PRACTICAL WORK

In this section, description of the actual localization system that was implemented will be given. Also different components that the system was built on will be explained in detail.

4.1 System

The overall system which was used in this thesis consists of hardware and software working together. Both hardware and software had multiple components which had to be fitted together.

4.1.1 Hardware

Hardware of the system is comprised of 3 distinctive elements, these elements can be seen in figure 10. First there is the robot, Pioneer 3-DX(P3-DX) model which is manufactured by ActivMedia(MobileRobots Inc)[21].The P3-DX is a differential drive robot, with two motored wheels with 16.5 cm diameter and one castor wheel. Robot measures the traveled distance using encoders on both wheels. Encoders have 500 ticks, meaning that the angular accuracy of the encoders is 0.72 degrees. This means that the smallest distance that can be measured is approximately 1.037 mm. The robot has 16 sonars at different angles, 4 of the sonars point perpendicular to the robot's current heading. Sonars are divided into 2 banks, each with 8 sonars. Sonars in each of the banks fire with 40 ms intervals. Time that it takes to get readings from all 16 sonars is 320 ms. Configuration of the sonars can be seen in figure 11. Inside the robot, there is an operating system called ARCOS, which handles low level functionality of the robot. This includes, for example, operating the motors and sonars. P3-DX robot can be seen in figure 12 with a mounted camera and a WLAN omnidirectional antenna. The camera was not used as a part of localization system.

Another critical component of the hardware system was the laptop. The laptop used in conjunction of the robot was a IBM ThinkPad with 1.6 GHz Pentium M processor and 512 MB of memory and Debian Linux as the operating system. The laptop is connected to the robot using USB converter which converts the data from the laptop USB port to RS-232 port found in the robot. Default communication interval between the robot and

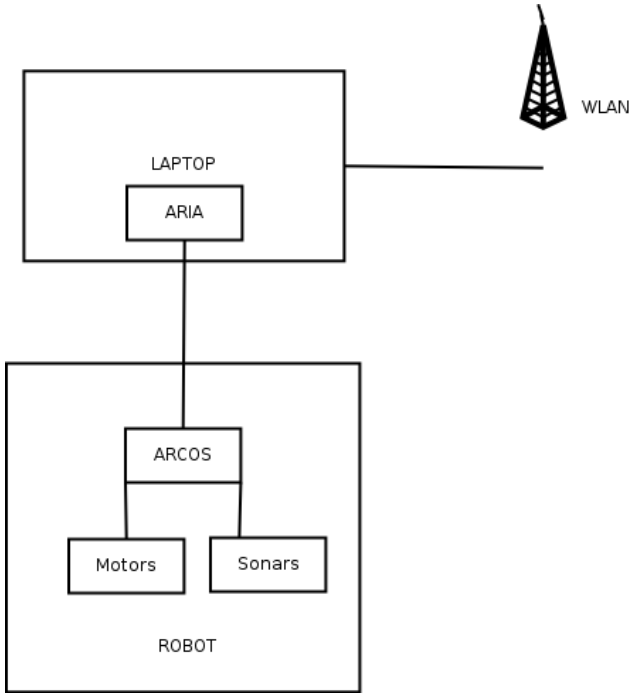

Figure 10: Hardware configuration

the laptop running the software is 100 ms. During this time the robot collects data from sonars and drives the motors according to the instructions from the software. The laptop is usually hidden in the metal cage seen in figure 12.

Final component used is the WLAN antenna and a PCMCIA WLAN interface card, which is used with the laptop and the antenna is plugged into the card. This enabled the use of WLAN measurements in localization while the robot is moving in the environment. The WLAN interface card is the same that was used in collecting the WLAN RSS maps, what was done independently from this thesis. This is done to make sure that the RSS maps match the measurements. Measurements from six different WLAN access points were used in localization.

4.1.2 Software

On the software side we have software library, which helped in developing the localization system and the interface to the robot, which is called ARIA. ARIA is a C++-library, which helps in developing software for the robot and it also includes all the functionality that the robot needs to be autonomous. Details of the ARIA library will be exlained more thoroughly in section 4.2.

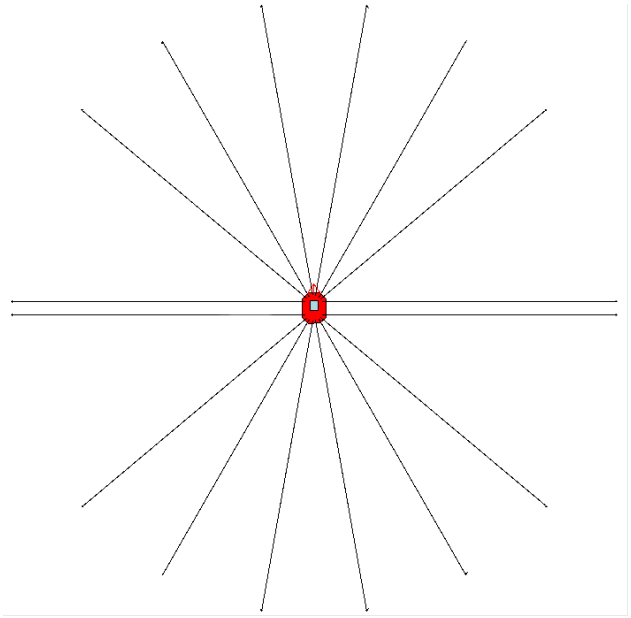

Figure 11: Configuration of sonars on P3-DX

Probably the most important tool that was used, was the MobileSim robot simulator, which allowed development of the localization software without using the robot. This accelerated the development considerably. Simulator was supplied by the robot manufacturer, ActivMedia, LLC. The simulator does not fully represent the real robot, mainly there are differences with sonar behaviour. The simulator returns always valid readings from sonars, even though with real sonar the signal could be deflected in a way that it would not return back to the sensor. Also all of the sonars have new readings after each communication cycle, which is not the case with real robot as explained before in section 4.1.1. This meant that additional steps had to be taken to ensure that localization works with the real robot as well as the simulator.

Another important tool is Kismet[22], which is a WLAN sniffer. It can detect and measure RSS from every WLAN client and access point that it can hear through the WLAN interface. In this thesis, Kismet was used to measure RSS data from access points, which were previously used in collecting the data for the RSS maps. Kismet itself is only a server which provides information through it's TCP server to a client. This was sufficient functionality. More information about the Kismet server is provided in section 4.2 along with client implementation.

On the programming side the tool that was used was thr Eclipse IDE[23] as the programming environment. It was chosen, because it provided integrated CVS(Concurrent Ver-

Figure 12: P3-DX robot

sions System)[24] functionality, and the use of CVS was mandatory. CVS is a SCM(Source Configuration Management) tool. Although Eclipse was originally developed for Java development, it has a plugin, CDT, which enables C++ development.

4.2 Implementation

The implemented software will be explained in the following sections. They cover the internal structure of the implemented software as well as instructions on how to use the localization system that was implemented. One of the key points of this section is to provide information on how the system works so that it can be understood and possibly modified and expanded in the future. Most important sections considering this are sections 4.2.2 and 4.2.3. The implemented software is currently compatible only with Linux operating system.

4.2.1 General overview

At first it is useful to get to know the internal structure of the software. A coarse overview of the entire software can be seen in figure 13. This figure depicts the different software components that had to be implemented. Arrows show the direction of communication. The dotted lines represent boundaries between local and remote components. Components, which start with the letters "Ar" were already implemented in the ARIA library. Also it was not necessary to implement the user interface as a suitable user interface "MobileEyes" was already available from the manufacturer of the robot. This leaves 3 separate components to be implemented.

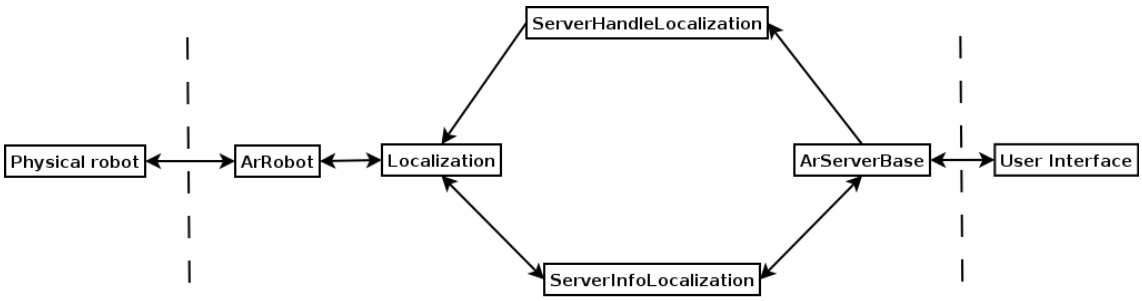

Figure 13: Overview of the software implementation

ArRobot component is the most critical component of this system and the Localization component would not work without it. Basic operation of ArRobot is to send control commands and receive sensor data(odometry, sonar) from the robot. Tasks which the ArRobot component runs are shown in figure 14. As can be seen from the figure, tasks form a tree like structure. Server Information Packet(SIP) handler is run first and it invokes the whole tree as SIPs come from the robot itself. This causes that the entire task tree with

it's tasks is run. Default time interval for SIP is 100 ms, meaning that all tasks which are on the tree should not run for more than 100 ms when running times are combined. This has had some impact on the design of the Localization component. Order of execution in the tree is determined by first task type, i.e. the branch. After that the tasks have priority level ranging from 1 to 100 in individual branches. Figure shows only three branches but in reality there are two more branches in the tree. These branches are action handling and state reflection. Action handling handles the internal actions which can be added to this branch and state reflection updates the robot's internal information. These two branches are between the sensor interpretation task and user task branches shown in figure 14.[25]

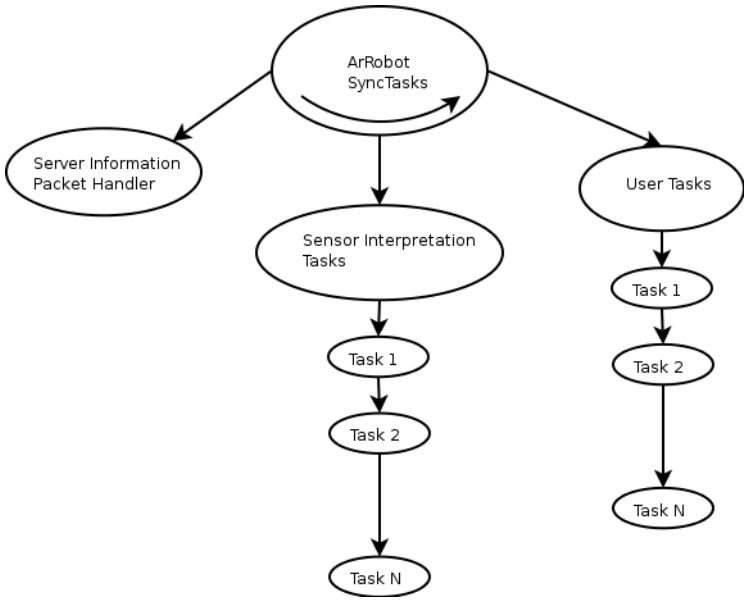

Figure 14: ArRobot task tree

Different parameters required by the measurement models were estimated by using data from test runs. Beam range finder model, found in section 3.2.2, parameters were evaluated with a straight run along an office corridor. Ranging data from the sensors was then evaluated, based on knowledge of approximate width of the corridor. After this, approximate parameters were inputted to the system. WLAN sensor data was collected from a standstill position using the WLAN antenna on the robot. Data was then evaluated and parameters, mainly the variance of the Gaussian distribution, were chosen.

4.2.2 Class structure

As the Localization component from figure 13 is by far the most important component, a simplified class diagram of this component is depicted in figure 15. This diagram shows the relations between different classes inside the Localization component. Localization component was implemented as library so that it could be easily put into use independent of other functions. Each of the classes that were implemented in this component will be described next, so that their meaning and function is clear.

Most important class of all is the Localization class. Its main functions are to gather data from various sources, in this case sonar and WLAN data, and to initialize all other classes. Also it runs each iteration of the MCL at defined time intervals and makes decisions about the condition of the localization, e.g. if the robot is lost and needs to use global localization. Localization-class itself has been derived from ArAsyncTask class, which is provided with the ARIA library. Classes inherited from the ArAsyncTask class can be run asynchronously, which is the required functionality that the localization of the robot can run independent of any other component. When created, Localization class adds a callback function task to the ArRobot task tree. This enables Localization class to read sensor data(odometry, sonar) from the actual robot each time ArRobot class receives a SIP from the robot.

ServerInfoLocalization and ServerHandleLocalization classes handle requests from the user interface. ServerInfoLocalization provides information about the localization, for example, state and location of the particles used in MCL. ServerHandleLocalization handles change of pose given by user through user interface. This means that that the user can re-localize the robot at any time and the localization system can handle this by resetting the robot to the desired position.

ISensorModel and IMotionModel classes in figure 15 are interfaces. Actual models for MCL can be inherited from these interfaces, motion models from IMotionModel class and measurement models from ISensorModel. Interfaces use tightly-coupled strategy design pattern to accommodate future models based on different sensors which are not used in this thesis. OdometryModel class was inherited from the MotionModel, which implements the odometry model from section 3.2. Two classes were inherited from the SensorModel class, BeamRangeFinderModel, for sonar measurement model, and WlanModel, for WLAN RSS model. Explanations for these models can be found in section 3.2. Methods inside the interface can be called from Localization class when needed for the

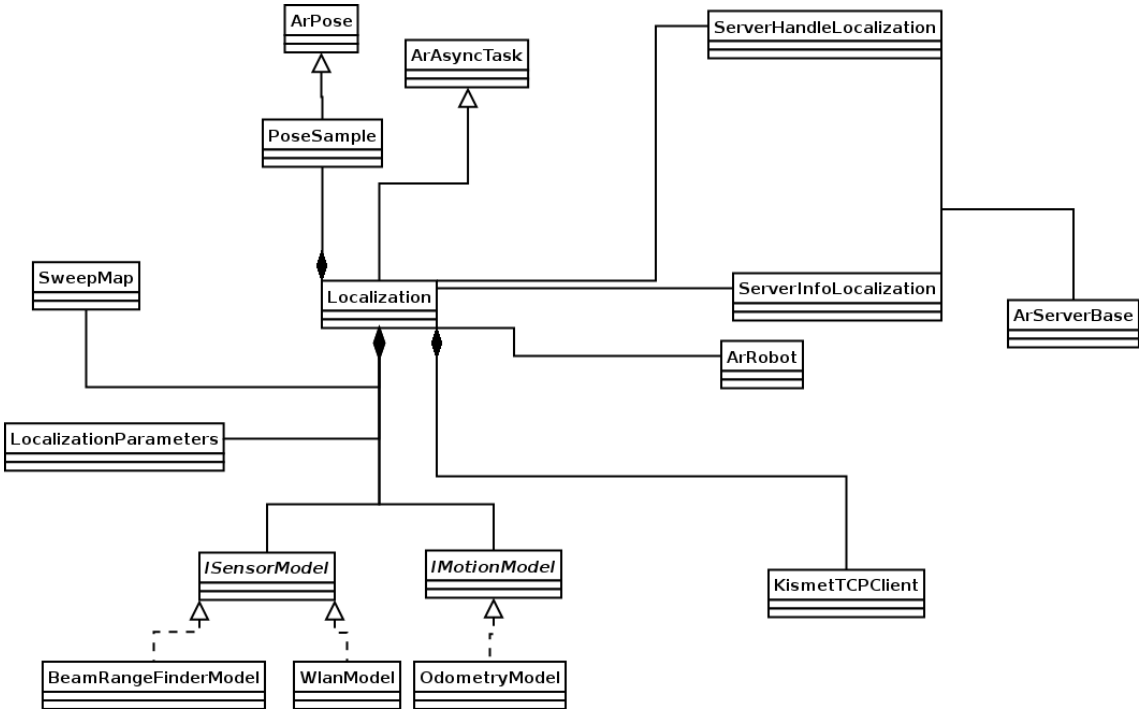

Figure 15: Class diagram of Localization component

MCL algorithm.

PoseSample class is inherited from ArPose class provided within the ARIA library. It is used to be able to save sample probability along with the pose of the sample. This helps considerably in communicating between the models and Localization class. Motion models require the pose of the sample, i.e. hypotesis of the robot pose. Measurement models require both the pose and probability of the pose so that probability for each sample can be assigned and accessed in the Localization class.

KismetTCPClient class handles the communication to the Kismet[22] server. The server offers simple text data to clients. Client can send a text string to the server to request different types of data. In this system all that was needed was the access points' MAC address, its channel and the signal strength from the basestation. KismetTCPClient collects this data and the data can be requested from Localization class to be forwarded to WlanModel, which is the end user of the data collected from Kismet.

SweepMap class takes the map used in localization and creates a map which covers the inside of the map. Using this class, Localization class can generate new samples inside the map, this does not "waste" samples as the robot can only move inside the map. Generating new samples happens when robot loses localization and needs to use global localization

or when localization adds new random samples.

LocalizationParameters class handles parameters. Parameters critical to the localization can be changed from a file and then the parameters are read to the LocalizationParameters class and then the Localization class can use these values to set parameters for different models and inside the Localization class itself.

4.2.3 Functionality of Localization library

Flow digrams of the localization component can be seen in figures 16 and 17. Figure 16 shows how the localization component is initialized and figure 17 shows how iterations of the localization system are carried out. Using these diagrams along with the source code should help to understand how the localization operates.

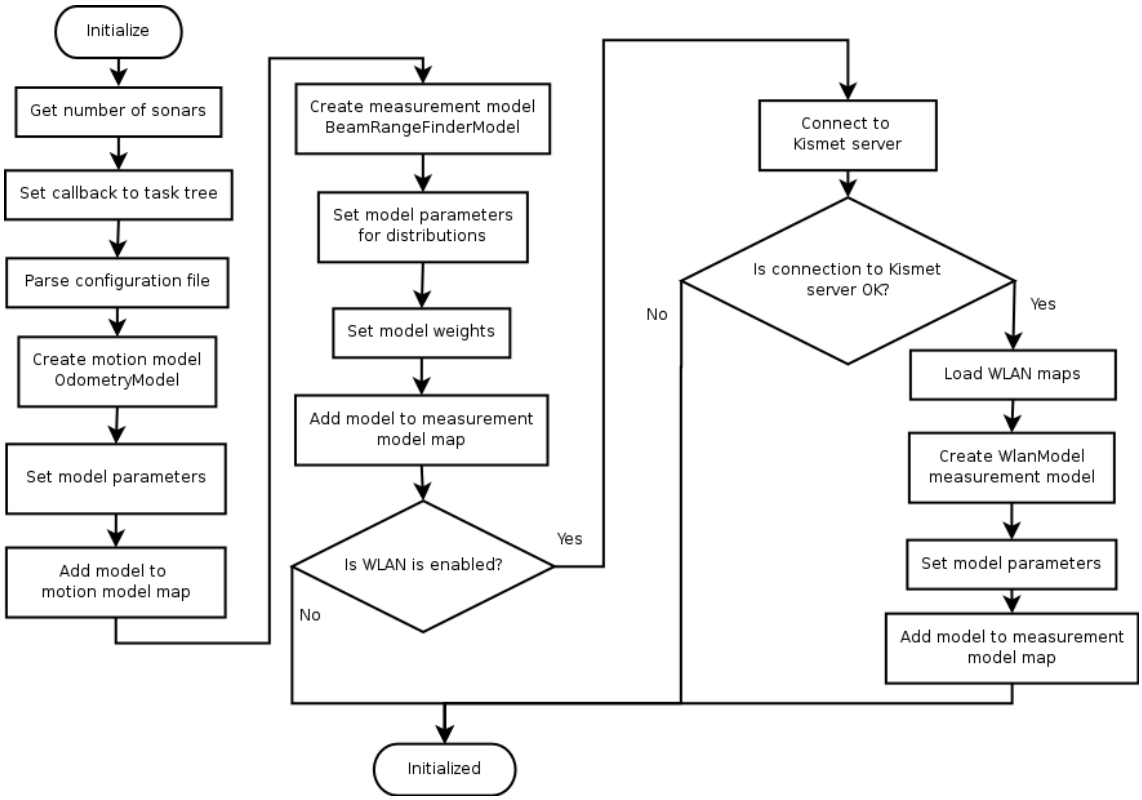

Figure 16: Flow chart of initializing localization

The order of operations can be seen from the figure 17. It shows what kind of operations are used in different conditions. The most important condition is localization success or failure. At the moment, this condition is triggered, if the amount of samples in the speci-

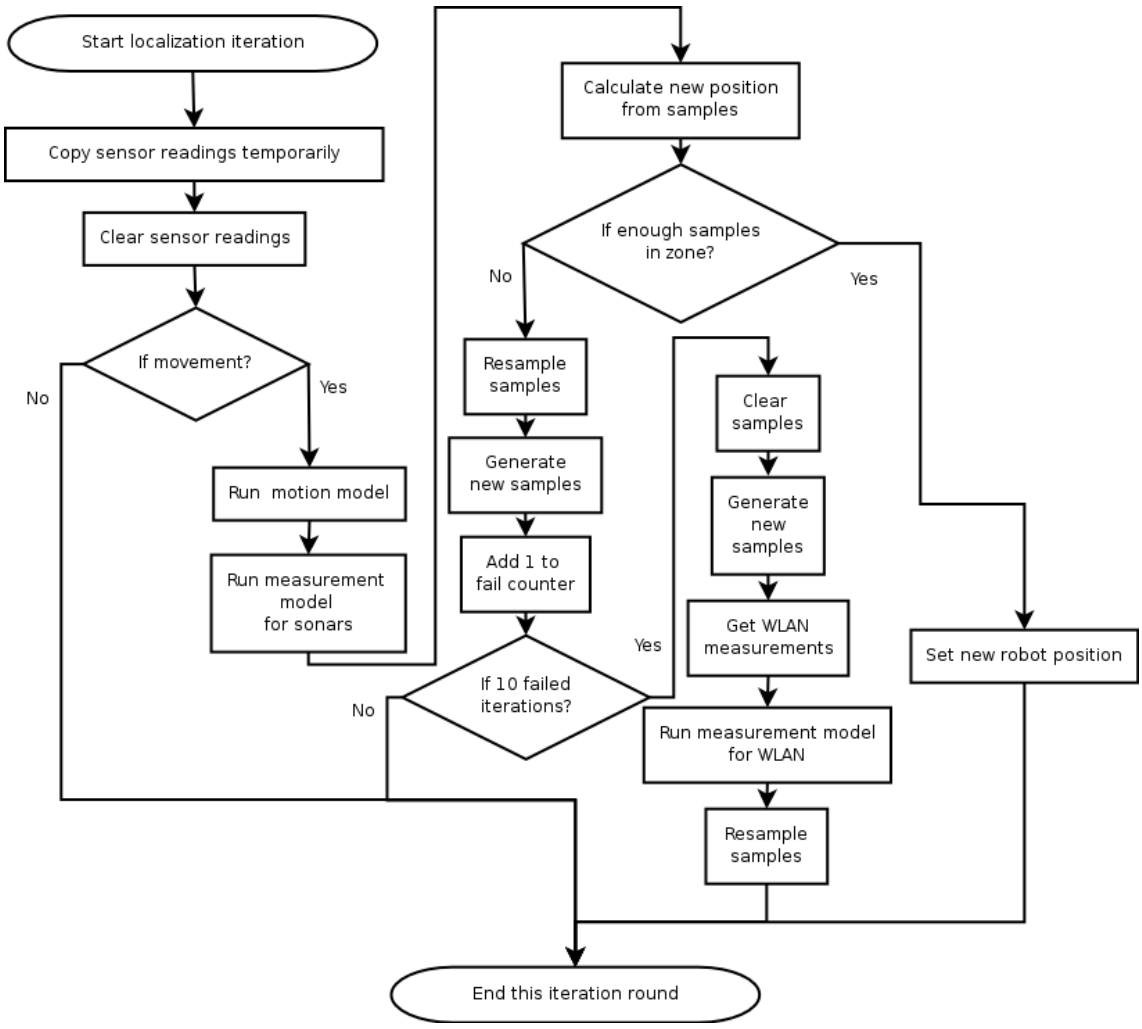

Figure 17: Flow chart of localization iteration round

fied zone, more about this zone in section 4.2.5, is less than 10% of the total samples. This allows multiple, even completely different, robot poses to be kept "alive". The diagram shows that after 10 failed localization, i.e. 10 failed iterations, WLAN localization is used on completely new samples, which are distributed randomly inside the map. The number of new samples is 3000 samples, which gives enough coverage on the map. WLAN localization assigns probabilities to each of these samples according to the WlanModel algorithm in figure 8. After this, the samples are resampled to 500 samples, so that only viable samples are left for the next iteration.

The normal iteration round is simple, the sonar measurement model is used to calculate sample probabilities and the samples are resampled. After this, a new robot position is calculated and the robot's internal state is updated to this acquired pose. However, if the robot has not moved during the last iteration, the models are not run. Time between the

iterations has been set to 1 second. This was found suitable to provide good localization results. More detailed view of the process can be found from the source code. Here the most important class method is calculating new pose for the robot. This method also determines if localization has failed or not. It also returns the new pose of the robot. The pose of the robot is calculated by using the highest probability pose and an average pose calculated from all the samples inside the zone. Final position is the average between these poses and the orientation is taken from the average pose. This reduces large sudden movements of the robot pose.

4.2.4 Documentation

The implemented software was documented with the help of Doxygen[26]. Main features of Doxygen is that it does not require any external files, only the source files. Documentation is added to the source files by special comment markings which the Doxygen reads. Doxygen then adds the comments automatically to the classes. This kind of operation enables fast documenting of the source code, for example classes and methods, as the comments can be done at the same time as something new is implemented in the source files.

Doxygen can create the documentation in many different fileformats, such as Adobe's PDF(Portable Document File) through LaTeX files, RTF(Rich Text Format) or HTML(HyperText Markup Language). Only HTML was used as it can be read using only a web browser.

Documentation can be created using `doxygen Doxyfile` command from the main directory, `naviloc/localization`. The documentation appears under `doc/html` directory. Documentation can be then found on each individual class and their individual methods and variables.

4.2.5 Instructions

Examples of functioning servers for the robot can be found under `examples` directory. This directory contains two examples, `testServer.cc` and `testServer2.cc`. The latter example is more complete and offers users the option to relocate the robot. Most of the example has been taken from ARIA example, `guiServer.cpp`. Changes have been made so that it takes into use the implemented software, i.e. Localization library,

also the navigation functionality has been removed, because SONARNL navigation does not work without SONARNL localization. Localization functionality is done on lines 120, 265 and 266. On line 120 the Localization object is initialized. On lines 265 and 266 ServerInfoLocalization and ServerHandleLocalization objects are initialized which connects the server to the localization. Only Localization object is mandatory, but no information can be passed to UI without ServerInfoLocalization and without ServerHandleLocalization, user cannot control the localization. After the Localization component has been initialized in the program, it runs without any input from the user.

Compiling the library `Localization` and the examples can be done by executing `make` on the top level directory of localization component, `naviloc/localization`. `Localization` library must be available when example servers are compiled or executed.

WLAN localization requires the WLAN RSS maps under `naviloc/localization/maps`. The maps have to be grayscale (0-255) PNG(Portable Network Graphics) images. The names of the images have to be in the following format: "xx_xx_xx_xx_xx_xx.png". The x's denote the MAC address of the WLAN access point. If these conditions are fulfilled, the rest of the process is automated.

Under directory `params` there is a file called `localization.p`. This file controls some of the important localization parameters. There are currently 17 parameters. Next, all the parameters and their effect on localization will be explained. Parameter names are written with fixed-width font.

`NumSamples` controls the amount of samples used in localization. More samples mean better localization result, but it also increases computing time. `ZoneSize` controls the size of the zone which is placed on top of the most probable sample after each iteration. If enough samples are within the zone the localization is considered successful. `ZoneSize` parameter defines half of the width of a square, side of the square is 2 times `ZoneSize`. Parameter `wFast` defines coefficient for calculating short-term average result of the localization. Parameter `wSlow` in the otherhand defines coefficient for calculating long-term average result of the localization. These two last parameters have relation $wFast >> wSlow \geq 0$. All these parameters control general behaviour of the localization system.

Next parameters affect the Odometry model of the localization system. `Alpha1`, `Alpha2`, `Alpha3` and `Alpha4` correspond to the α_1, α_2, α_3 and α_4 parameters in the algorithm 5.

Some additional noise can be introduced to the system through parameters `PerX`, `PerY`, `PerTh`, they simply add gaussian noise to the final sample pose.

Beam range finder model used in localization can be effected by the next parameters. `ZNorm`, `ZShort`, `ZMax` and `ZRand` are the weights for the distributions p_{hit}, p_{short}, p_{max} and p_{rand}. Weights should add up to 1. Two other parameters control the parameters of p_{hit} and p_{short}. `NormDistDev` defines parameter σ in distribution p_{hit} and `ExpDistLambda` defines λ in distribution p_{short}.

Rest of the parameters affect the WlanModel. `EnableWlan` parameter controls the use of WLAN in localization. If the parameter is 1, WLAN is enabled. `MapScale` defines how many millimeters is one pixel on the WLAN RSS map. `MapOffsetX` and `MapOffsetX` define the difference between the image top left corner or the image origin in relation to the ARIA feature map origin $(0, 0)$.

5 EXPERIMENTS

Experiments were divided into two test types: algorithmic tests with the simulator MobileSim, and tests with real robot. Furthermore the tests with the real robot were divided into testing path tracking ability of the robot and global localization ability using WLAN information. Main idea behind all of these tests was to investigate how accurate the localization was. In total 4 different tests were conducted. The results from these tests can be found in appendix 1.

5.1 Algorithmic testing

Only path tracking accuracy was tested using algorithmic testing, and only 2 parameters affecting the localization accuracy were tested in 2 separate tests. The parameters were the amount of samples used in localization and perturbation of the samples. Perturbation of the samples is composed of 3 elements: noise in x axis, noise in y axis and noise in angle of the pose. These elements map naturally to the 2D pose of the robot, (x, y, θ). Table 1 shows the results of tests 1 and 2. Each change to the parameters was tested 10 times using the wander operation, which drives the robot automatically and evades the walls. After each 4 minute run, the distance to the real pose, obtained from simulator, was compared to the robot pose and distance calculated between these. Map, which was used in the simulator(LUT phase 6, floor 5) can be seen in figure 18. Starting point of each run can be found at the top corridor of the map.

Test 1 shows that increasing samples yield better results. Some of the large distances in test with 10 samples was caused by low sample count, which resulted in robot taking the wrong direction at the beginning of the run. But other results using 10 samples are adequate, which means that the robot could localize itself even with 10 samples. However, using 100 and 500 samples shows clearly that, the higher the amount of samples is, the

Test nr.	Batch 1	Batch 2	Batch 3	Remarks
1	15.8	0.15	0.08	Average distance(m)
1	24.03	0.12	0.05	Deviation(m)
2	0.08	0.05	0.05	Average distance(m)
2	0.05	0.02	0.05	Deviation(m)

Table 1: Test results for tests 1 and 2

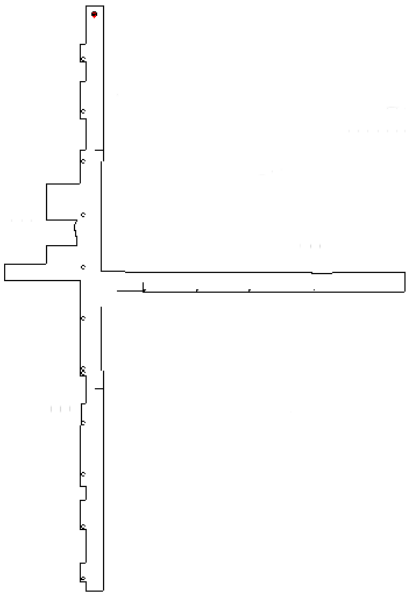

Figure 18: Map of the Lappeenranta University of Technology phase 6, floor 5

better and more consistent the final localization accuracy is.

Changing of the perturbation parameters in test 2 show that these parameters have an effect on the accuracy of the localization. Perturbation parameters tests were conducted to see the effect of adding noise to the samples. The units used in parameters are millimeters. Idea behind 3 tests here was to investigate if adding noise can improve the localization accuracy compared to using no noise at all. MCL and it's motion model already add some noise to samples, but it is dependent on how the robot moves, this perturbation is independent of the robot movement. The results show that it is beneficial to the localization accuracy to add some noise, but not too much. However the difference is quite small and all of the results gained from using different perturbation paramters are acceptable. The noise adds some robustness to the system by spreading the samples and this way localization system can localize the robot a bit better.

5.2 Tests with real robot

Tests with the real robot, P3-DX, were divided into path tracking testing and global localization testing. Path tracking test is test number 3 in appendix 1 and main results from this test can be found also in table 2. Goal of this test was to confirm that the localization system works in real environment and not just in simulator. The path that robot was

Test nr.	Batch 1	Batch 2	Batch 3	Remarks
3	0.25			Average distance(m)
3	0.08			Deviation(m)
4	4.6	6		Average distance(m)
4	3.51	3.86		Deviation(m)

Table 2: Test results for tests 3 and 4

programmed to follow is shown in figure 19. The results are only approximate results as no other method than human eye were used to evaluate the distance between the true pose and the robot pose. We can see immediately that the results are worse than in the simulator, but the results are acceptable for use in applications.

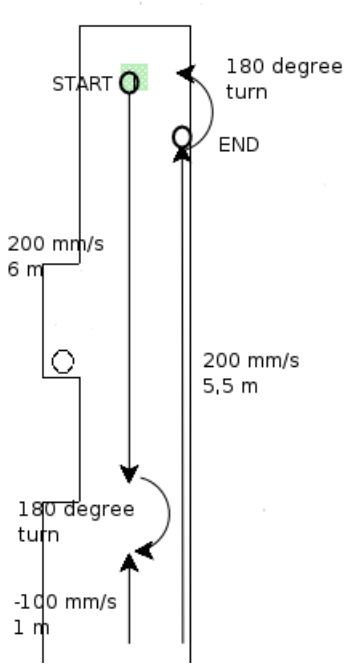

Figure 19: Robot path in test 3

Global localization(WLAN localization) was tested by human operator controlling the robot. Each test run was started at the same position for each test set. In the first test, the robot started from the same pose as in test 3. In the second test, robot starting pose was in front of the elevator in phase 6, floor 5. In both test sets, robot was driven southwards until localization was stable(no jumping between poses). Test number 4 in appendix 1 shows the results of these tests and table 2 has the main results. Tests show that global localization could be better. However it can be seen from both tests, that in several cases, localization was able to localize the robot at least in the right part of the floor. In tests where result was over 25 meters from the right pose, the wlan model gave high probabil-

ities to samples that were in correct location but also to samples that were far from the correct location and in the end the robot was jumping between these poses and finally deciding on the worse pose. These kind of cases are quite bad and should be eliminated.

6 CONCLUSIONS

In the end, the system, which is described in this thesis, was implemented successfully and enables further development. The main goal was to create the sonar localization, which could be extended. However, due to fast development, including WLAN global localization to this thesis was possible. The WLAN localization does not perform yet so well that it could be used reliably in all circumstances as can be seen from the tests. This could be one area of further development.

Previous WLAN models used in localization have been focusing on RSS fingerprinting. The fingerprints are collected beforehand from the desired environment. Each fingerprint has information about its location and RSS. A location can then be determined from the fingerprints by comparing them to the actual RSS. This method has been used, for example, by Ladd et al.[7] and Kaemarungsi et al.[20]. Method used in this thesis is different from fingerprinting. The model used here uses a probability distribution to model the environment, instead of discrete set of fingerprints. However, this method requires samples from the real environment the same way as in fingerprinting. By using the previously created system[2], the collection of samples can be done automatically.

It is also possible to create more accurate maps using the same method which resulted in current maps. Maps used in this thesis were a result from an earlier project, a bachelor's thesis by Arno Hietanen[2]. The current maps are not perhaps accurate enough for reliable WLAN localization, but this could be alleviated by using more measuring points which will improve the map quality.

Localization using sonars could also be improved. One of the problems in the current system is computation time. This could be optimized further and as a result of this optimization, more accurate path tracking could be achieved by using more samples in localization. Also there are myriad of techniques which improve the MCL. However, current path tracking of the implemented system works within acceptable range of accuracy.

As the localization system was implemented successfully, it should also be possible to use other sensors than sonar and WLAN to aid localization. One example of this could be the camera, which is installed on the robot. Visual localization systems have been developed before[18], [13], [27], which shows that there is potential. Humans also use mostly vision to localize themselves, so there is certainly an interesting connection how a robot could use vision in the same way as humans.

REFERENCES

[1] MobileRobots Inc. P3-DX. [Retrieved 28.9.2006]
Available: http://www.activrobots.com/ROBOTS/p2dx.html.

[2] A. Hietanen. WLAN-kuuluvuuskartoitus liikkuvalla robotilla, 2006. Lappeenranta University of Techonlogy, Department of Information Technology.

[3] Robot hall of fame. [Retrieved 28.9.2006]
Available: http://www.robothalloffame.org/unimate.html.

[4] R. Siegwart and I.R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2005.

[5] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[6] D. Doyle. Accuracy test of consumer grade GPS receivers. [Retrieved 28.9.2006]
Available: http://www.doylesdartden.com/gis/gpstest.htm.

[7] A.M. Ladd, K.E. Bekris, A. Rudys, L.E. Kavraki, D.S. Wallach, and G. Marceau. Robotics-based location sensing using wireless ethernet. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, Atlanta, Georgia, USA, 2002.

[8] P. Hii and A. Zaslavsky. Improving location accuracy by combining WLAN positioning and sensor technology. In *Proceedings of the Workshop on Real-World Wireless Sensor Networks*, Uppsala, Sweden, 2005.

[9] Advanced Warehouse Automation OY. AWA solutions for paper and converting industry. [Retrieved 2.10.2006]
Available: http://www.awaoy.fi/paper.html.

[10] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo localization: Efficient position estimations for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence*, Orlando, Florida, USA, 1999.

[11] F. Dellaert, S. Thrun, W. Burgard, and D. Fox. Monte Carlo Localization for mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, Antwerp, Belgium, 1999.

[12] J.-S. Gutmann. Markov-Kalman localization for mobile robots. In *Proceedings of the 6th International Conference on Pattern Recognition*, Quebec, Canada, 2002.

[13] N. Tomatis K.O. Arras. Improving robustness and precision in mobile robot localization by using laser range finding and monocular vision. In *Third European Workshop on Advanced Mobile Robots*, Switzerland, 1999.

[14] P. Swerling. A proposed stagewise differential correction procedure for satellite tracking and prediction, 1958. Technical report P-1292, RAND Corporation.

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

[16] N. Metropolis and S. Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, 44:335–341, 1949.

[17] R. Ueda, T. Fukase, Y. Kobayashi, T. Arai, H. Yuasa, and J. Ota. Uniform Monte Carlo localization - fast and robust self-localization method for mobile robots. In *IEEE International Conference on Robotics and Automation*, Arlington, VA, USA, 2002.

[18] J. Wolf, W. Burgard, and H. Burkhardt. Robust vision-based localization by combining an image-retrieval system with monte carlo localization. *IEEE Transactions on Robotics*, 21(2):208 – 216, 2005.

[19] D. Xianzhong M. Xudong and S. Wen. Vision-based extended monte carlo localization for mobile robot. In *IEEE International Conference Mechatronics and Automation*, Niagara Falls, Canada, 2005.

[20] K. Kaemarungsi and P. Krishnamurthy. Properties of indoor received signal strength for WLAN location fingerprinting. In *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Boston, Massachusetts, USA, 2004.

[21] MobileRobots Inc. MobileRobots Inc. [Retrieved 24.1.2007]
Available: `http://www.mobilerobots.com/corporate.html`.

[22] dragorn@kismetwireless.net. Kismet. [Retrieved 16.11.2006]
Available: `http://www.kismetwireless.net/`.

[23] The Eclipse Foundation. Eclipse - an open development platform. [Retrieved 08.12.2006]
Available: `http://www.eclipse.org/`.

[24] Ximbiot D. R. Price. Cvs - concurrent versions system. [Retrieved 08.12.2006]
Available: `http://www.nongnu.org/cvs/`.

[25] ActivMedia Robotics LLC. ARNL Reference Manual. Version 2.4.1.

[26] D. van Heesch. Doxygen. [Retrieved 27.11.2006]
Available: `http://www.stack.nl/ dimitri/doxygen/`.

[27] C. Schroeter, A. Koenig, H-J-Boehme, and H-M Gross. Multi-sensor Monte-Carlo-localization combining omni-vision and sonar range sensors. In *Proceedings of the European Conference on Mobile Robots*, Ancona, Italy, 2005.

APPENDIX 1. Test cases

SONAR LOCALIZATION ACCURACY TESTS – distance measure after test run

Parameter tests with simulator

TEST 1 – Effect of amount of samples on localization accuracy

4 minute run with “Wander” - other settings at default settings

Date: 11.12.2006 – 12.12.2006

Sample count: 10

Test run #	Apprx. distance
1	0.4
2	0.2
3	0.5
4	0.1
5	67
6	0.08
7	0.32
8	36
9	0.38
10	53
Avg:	15.8
Deviation:	26.03

Sample count: 500

Test run #	Apprx. distance
1	0.05
2	0.13
3	0.04
4	0.21
5	0.03
6	0.09
7	0.07
8	0.07
9	0.05
10	0.04
Avg:	0.08
Deviation:	0.05

Sample count: 100

Test run #	Apprx. distance
1	0.07
2	0.19
3	0.12
4	0.11
5	0.08
6	0.14
7	0.06
8	0.47
9	0.1
10	0.14
Avg:	0.15
Deviation:	0.12

TEST 2 – Effect of odometry perturbation parameters, PerX, PerY, PerTh on localization accuracy

4 minute run with “Wander” - other settings at default settings

Date: 12.12.2006

PerX = 0, PerY = 0, PerTh = 0

Test run #	Apprx. distance
1	0.05
2	0.12
3	0.09
4	0.03
5	0.05
6	0.17
7	0.04
8	0.11
9	0.13
10	0.03
Avg:	0.08
Deviation:	0.05

PerX = 75, PerY = 75, PerTh = 5

Test run #	Apprx. distance
1	0.16
2	0.01
3	0.04
4	0.14
5	0.03
6	0.03
7	0.03
8	0.02
9	0.03
10	0.04
Avg:	0.05
Deviation:	0.05

PerX = 25, PerY = 25, PerTh = 2

Test run #	Apprx. distance
1	0.09
2	0.05
3	0.01
4	0.04
5	0.06
6	0.03
7	0.03
8	0.07
9	0.04
10	0.03
Avg:	0.05
Deviation:	0.02

Tests with real robot

TEST 3 – Accuracy of localization with real robot

A predetermined path, 92 seconds - all settings at default settings, start at home position

Date: 13.12.2006

Location: LUT phase 6, floor 5, North corridor

Test run #	Apprx. distance
1	0.1
2	0.2
3	0.3
4	0.2
5	0.3
6	0.3
7	0.2
8	0.4
9	0.3
10	0.2
Avg:	0.25
Deviation:	0.08

WLAN GLOBAL LOCALIZATION ACCURACY – distance measure after test run
Tests with real robot

TEST 4 – Accuracy of WLAN localization with real robot

WLAN global localization accuracy, measured after robot has lost localization

and used WLAN+sonars to globally localize itself
Date: 18.12.2006

Location: LUT phase 6, floor 5, North corridor

Test run #	Apprx. distance
1	> 25
2	> 25
3	> 25
4	5
5	5
6	2
7	> 25
8	1
9	10
10	> 25

Avg(only < 25): 4.6

Deviation: 3.51

Location: LUT phase 6, floor 5 center

Test run #	Apprx. distance
1	5
2	15
3	5
4	4
5	10
6	2
7	3
8	5
9	7
10	4

Avg: 6

Deviation: 3.86