Implementation of particle filtering in a turtlebot

By
Frederik Penné

Department: Industrial Engineering in Energy

Defended on
June 15, 2015

Internal Promoters:
Johan Baeten (KULeuven, Belgium)
Leo Rutten (KULeuven, Belgium)

External Promoters:
Pere Ridao Rodriguez (UDG, Spain)
Josep Bosh Alay (UDG, Spain)

Abstract

The interest in autonomous mobile robot applications is growing. In order to be autonomous the robots need to have a certain awareness of their environment. Therefore maps are build and sensor data is obtained. One of the problems is to localize a robot with the use of sensor data in a known environment. This can be done with the use of a localization method called particle filter. This is a robust localization method that allows the robot to estimate its position in a given map. We were able to implement this method into the turtlebot platform and the performances are tested on different environments

Acknowledgements

This master thesis could not have been performed if not for the assistance and support of many individuals.

Foremost, I would like to express my sincere gratitude to my advisor Josep Bosh Alay for the continuous support of my master thesis. In addition, a thank you to Prof. Pere Ridao Rodriguez, who introduced me to Robotics and the CIRS research center of Girona. Their guidance helped me in all the time of this master thesis.

Besides my external promoters, I would like to thank the people in the CIRS research center of Girona for their assistance.

I am also grateful to my internal promoters, Prof. Johan Baeten and Prof. Leo Rutten, for their guidance in my master thesis.

Last but not least, I am extremely grateful for the opportunity to be able to go in Erasmus in Girona, Spain. I would like to thank therefore all the people who made this possible for me.

Table of Contents

1 Introduction

Over the past few years a lot of research and development associated with robotics have been done. The use of robots gets more important. The main goal of robots is to assist humans by taking over tasks that are difficult or repetitive. But robots are capable of much more. For example, the exploration of Mars or surveys the sea bottom. These are tasks that can be done much easier with the use of autonomous robots. All these applications require a robot able to make decisions on his own without human intervention. Therefore, a lot of sensor data processing is needed to make a robot autonomous. One of the main concerns is the interpretation of this data to act autonomously. For this a lot of methods have been developed. The main research topics are: localization, obstacle avoidance and map building. These problems are not easy to solve because of the uncertainties in sensor values. [1]

To simplify the development of these methods and simplify these methods and simplify their use for other people, a standard framework will be used. One of the most promising frameworks to do so is ROS. It is an open source robotic operating system, it is modular build and code can be programmed in different languages. Because of his flexibility ROS is becoming a standard in robotics research. There are more than 60 types of robots that support ROS, and one of them is the turtlebot. [2]

The turtlebot is a platform to learn and experiment with ROS. The turtlebot is used to solve localization problems, obstacle avoidance and map building. It is the perfect platform to experiment with these algorithms. The turtlebot itself consist of a mobile base, a laptop and a Microsoft Kinect sensor. This is sufficient to solve a wide variety of indoor problems.

One of the most promising algorithms for localization and also for map building is the particle filter, also referred to as Sequential Monte Carlo. It is a robust localization method that uses a set of particles spread over a map. These particles are compared with the sensor data gathered by the robot. In this way it is possible to localize a robot in a known map. One of the big advantages of this localization method is that the initial position of the robot can be unknown.

The particle filter can also be used for SLAM (simultaneous localization and mapping). An example of this is the Rao-Blackwellized Particle Filter that is used to build maps. In this method each particle carries an individual map of the environment. In this way the particle filter is used to build a map. [3]

1.1 Purpose

The purpose of this paper is to get familiar with robotics and localization applications. This is done by first learning ROS which is done by overviewing the tutorials available in the web. The tutorials are easy to follow and go step by step. No prior knowledge of any programming languages is required. Next step in learning ROS is experimenting with a simple simulator called turtlesim. This simulator is provided by ROS and is useful to learn the basics about sending, receiving and recording data. [4]

Once the ROS basics are known a program languages is required to develop any robotic application. We have decided to learn and use python. We have chosen for python because it is easier to learn and debug. Python is recommended for new programmers. Both python and C++ can be combined in ROS. The only concern about python is its computation speed. However in this application this is not a problem.

Python is learned through the ROS tutorials and furthermore using basic tutorials that are available at the web.
After knowing the basics of program language and ROS the turtlebot platform is explored. For the turtlebot there are many experiments and demo’s available to get familiar with it.

Once all these elements are learned it is possible to start experimenting with a localization method. One of the most interesting localization methods is the particle filter.

In section 2 a literature review is done about ROS and the turtlebot. In section 3 the particle filter will be explained and the different steps followed to obtain a correct implementation. In Section 4 we show the performance of the particle filter and present a discussion about the obtained results. At section 5 some conclusions are made about the implementation.

2 Literature Review

2.1 Mobile robot navigation

Mobile robot navigation can be done. If we focus on indoor navigation, methods can be divided into three categories which are Map-Based Navigation, Map-Building-Based Navigation and Mapless navigation. In the first method the map is already known and it is easy to plan a path in a the environment. Here the problem is that the map has to be well defined and changes in the environment are not taken into account. In the second category the focus is to build a map from scratch while localizing the vehicle with respect to his map. This is also known as SLAM (simultaneous localization and mapping). The third one, Map less navigation, is often based on a behavior of the robot with the use of vision. The required robot motions are determined by observing and extracting relevant information from the landmarks in the environment. A big advantage of map Less navigation is that it can be used in a dynamic environment. [5]

Another problem was to display the world in an easy map that is understandable for robots.
One of the most common ways to represent a map is an occupancy-grid map. This contains a map of small cells where each cell can be occupied or free. A disadvantage with the use of these maps is that the resolution of the map affect the accuracy of the navigation and as higher the resolution is, the more computation power is needed. An advantage of occupancy-grid map is that the maps can be dynamic. If something in the environment changes it is not hard to change it in a occupancy-grid map. [6]

In contraposition with the occupancy grid maps there are the object maps that represent obstacle's with points, lines or walls. The edges of an obstacle are defined with coordinates. These maps are more compact than occupancy grid maps and are more accurate. Disadvantages are that not all real environments are convertible to object maps, because of the complexity of the real world. [6]

2.2 ROS

2.2.1 introduction

ROS is a robot operation system that can be used for hardware abstraction, low-level device control, implementation of commonly-used functionalities, message-passing between processes and package management. It also provides tools and libraries. It is distributed under the term of the BSD license, and is also open source software which is free for development of both non-commercial and commercial project. [2], [7]

ROS was developed in 2007 and was mainly developed by Willow Garage and now for the open source robotics foundation (OSRF). So ROS is a newer growing platform and is becoming a standard for robotic applications. There are already a big list of robots that support ROS and the list is still growing. One of the most famous one for research purposes is the turtlebot. The turtlebot is commonly used to solve path planning problems , obstacle avoidance and navigation. [8]

A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model. This is a big advantage of using ROS because the nodes don't have to run on the same system (i.e. they can be distributed in a network) and are very flexible.

ROS does not depend on one program language. Different languages like c++ and python can be used together.

2.2.2 General Concepts

ROS is build out of packages. Packages are the main unit for managing software in ROS and can contain libraries, tools, executables, configure files, etc.

Nodes are the processes which are combined together into a graph and communicate with other processes using steaming topics, RPC services, and the Parameter Servers. Every node is declared to the master.

The master (also known as roscore) is a kind of a server where nodes are declared and registered. It allows nodes to find each other and exchange data. The roscore starts the rosout topic. It is the console log reporting mechanism used in ROS. The main idea is that the rosout node subscribes to /rosout, logs the messages to file and re-broadcasts the messages to /rosout_agg. [4]

Topics are the connections between nodes. They are named buses over which nodes exchange messages with a publish/subscribe paradigm. A node sends out a message by publishing it to a given topic. A node which is interested on a certain kind of data will subscribe to the suitable topic.

There are a lot of different messages to send on a topic . The use of messages depends on the application. Messages are data structures made of a combination of primitive types like Boolean, integers, floating point,.... This is a compact and fast way to send data to different systems. [9]

2.3 Turtlebot

Turtlebot is a mobile robot platform to get familiar with ROS and robots in general. It is an open source platform so a lot of applications are already available. The turtlebot is composed by a kobuki robot as base, a laptop with ROS and a xbox kinect as sensor. With this simple setup the turtlebot is able to handle vision, localization, communication and mobility.

The turtlebot can also be modified for other purposes. For example there is the turtlebot arm, this is a robotic arm that can be place on the turtlebot for intervention operations. The turtlebot is not only used for research purposes but also for recreation. A well programmed turtlebot can help you in your house and execute simple tasks. Note that the turtlebot is mainly used indoor and not resistant to outdoor activities.

The turtlebot was developed in 2010 by willow garage which are also the founders of ROS. There are two versions of turtlebot available the turtlebot 1 and the turtlebot 2. The main difference is the robot base. The turtlebot 1 uses an iRobot Create base while the turtlebot 2 uses the newer iClebo Kobuki base. [9], [10]

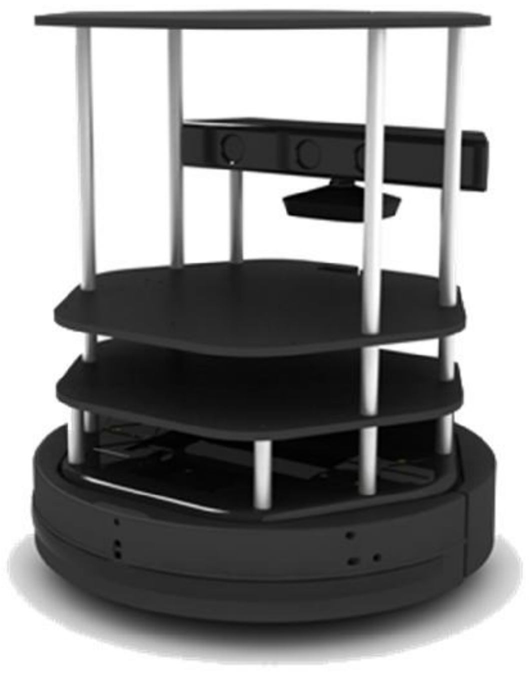


Figure 1 Turtlebot

2.3.1 Kobuki robot

iClebo Kobuki is the perfect base for the turtlebot. It is a mobile base. It has sensors, motors and power sources, however by itself, it cannot do anything. This base has replaced the previous iRobot Create base that was used in the first turtlebot. The iRobot Create was based on the Roomba vacuum cleaner that was also developed by iRobot.

The iClebo Kobuki is nothing on his own it is a base for customization. Therefor it is a low-cost mobile research base that provides power supplies for an external computer as well as additional sensors and actuators. The expected operation time is around 3 hours with a small battery and 7 hours with a big battery. The maximum speed of the base is 70 cm/s. The base also has a cliff detector and will not ride of a cliff that is higher than 5 cm. [11], [12]

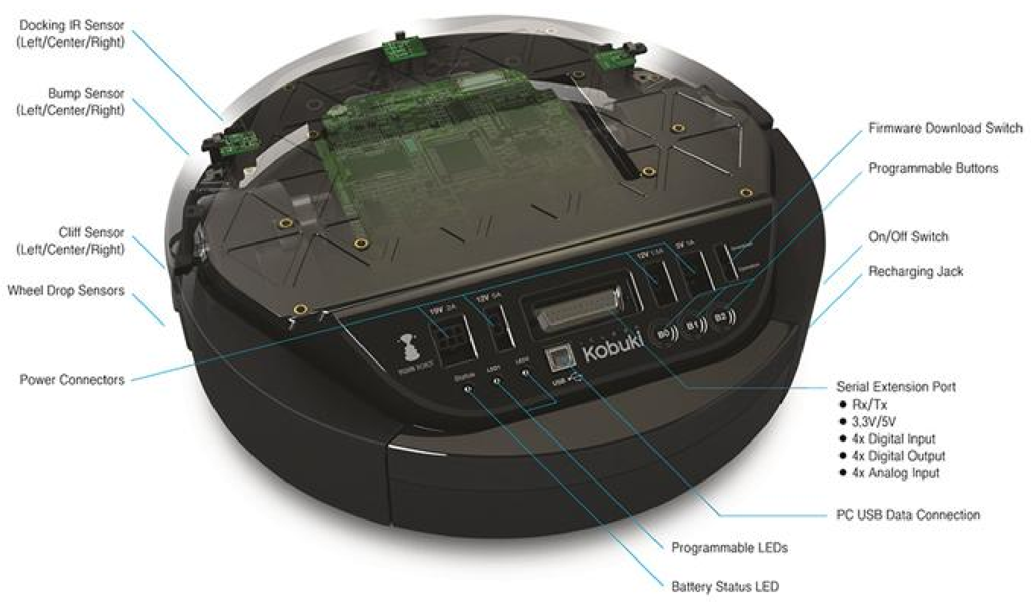


Figure 2 Kobuki base

As you can see in Figure 2 the base has a lot of possible customization. The serial extension port ensure that the base can use a large selections of sensors. For connecting the Kinect to the base the USB port is used. The base has programmable buttons and programmable LEDs for your own custom applications.

2.3.2 Kinect

The Microsoft kinect was developed for the Xbox for gaming applications. But this high accurate depth sensor can be used for much more. One of those applications is in the turtlebot. The turtlebot uses the kinect sensor as the robot main sensor. This leads to a versatile usage of the turtlebot.

The overview of Kinect sensor is shown in Figure 3 Kinect sensor. The middle hole is the "RGB camera" which is a camera that gathers color images at 30 frames per second. The left hole and the right one are "3D depth sensors", the sensor on the left side is an infrared projector. On the right side there is an infrared camera which is used to compute how far the objects are from the camera. It has four microphones which are built in to the Kinect and are used to record audio or even for speech recognition. The last one is a motorized tilt which allows the Kinect to move in vertical way which means up and down. The sensor can tilt between -27° and 27°. [9], [13]

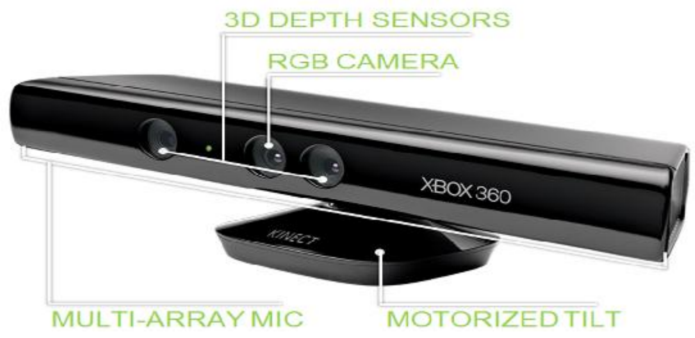


Figure 3 Kinect sensor

The most important feature of the kinect is the 3D depth sensor. The Kinect sensor doesn’t use the common used time of flight principle but another technique that uses a triangulation between the source (projector) and the receiver (infrared sensor). The infrared projector projects a pattern to the scene and the CMOS image sensor is able to register this pattern into a point cloud. Now the Kinect is able to extract 3D data with the use of various algorithms. This technique is also referred as light coding. Obtaining the depth information is our main objective for the application that is explained later on.

The field of view of the Kinect is 57,8°.The range of the sensor is between 0,6 meters and 4 meters. The Kinect has a blind spot if the obstacle are closer than 0,6 meters. Like all other laser scanners the Kinect cannot register glass objects. [14], [15], [16]

Thanks to this sensor the turtlebot has a lot of applications. A few examples of experimental possibilities are testing planning algorithms, obstacle avoidance and SLAM Simultaneous localization and mapping . The kinect sensor itself can also be used for recognizing gestures, body movements and speech recognition.

2.3.3 Turtlebot getting started

First of all the necessary software have to be installed. The most important software packages are the main turtlebot packages, gazebo and rviz to simulate the turtlebot or visualize measurements.

After installing all the software it is a matter of connecting the turtlebot laptop with your own computer using ROS. It is possible to only use the turtlebot laptop but it is recommended to connect your own computer to limit the processing effort of the turtlebot laptop. The most common way to connect those two computers is throughWi-Fi. Standard setup is shown In Figure 4.

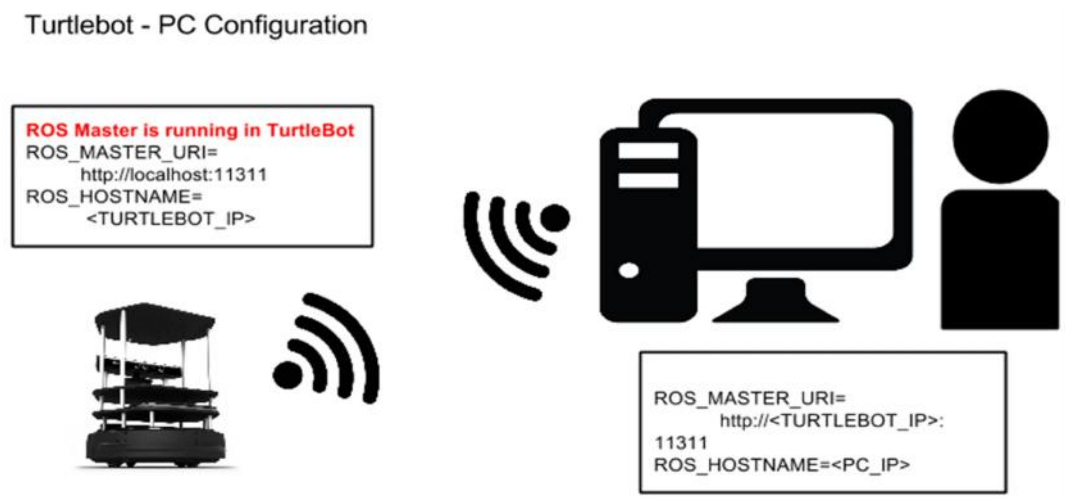


Figure 4 connecting environment

SSH is used to acces the turtlebot from the remote pc. Ssh is a command to log in to a remote service. Now the ROS master can be started with the command roscore. If everything is configured properly the topics, nodes, services,… can be found by the main pc even when the topics, nodes, services are in the turtlebot pc.

2.3.4 Turtlebot bring-up

Once the workspace is configured and a roscore is running it is possible to bring up the turtlebot. This bring-up starts the basic nodes for the turtlebot. There are different bring-ups available but in this case the most simple one is used (minimal). This is done by opening a terminal and executing the command roslaunch turtlebot_bringup minimal.launch. This creates the following topics and nodes shown in Figure 5. [4]

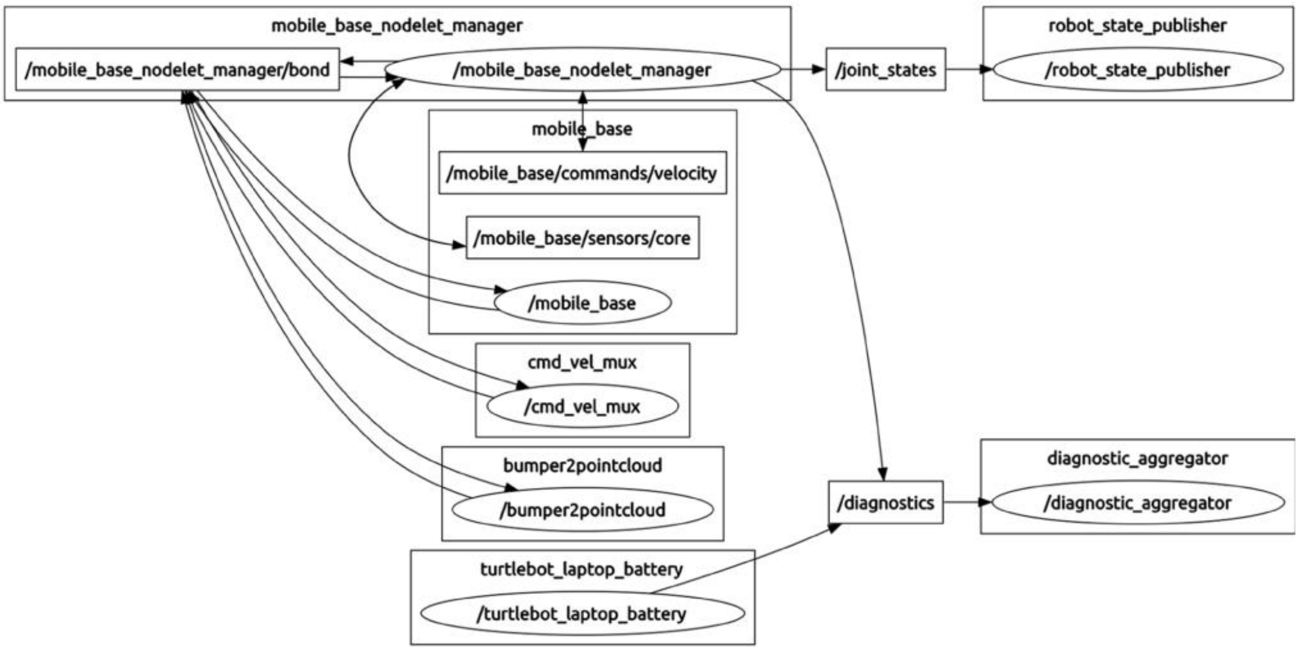
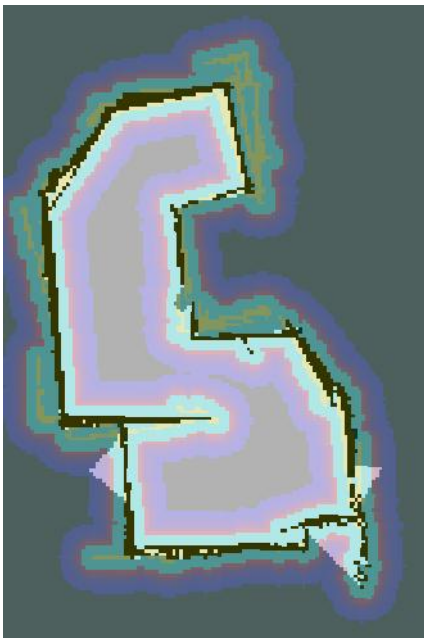


Figure 5 published nodes

After bringing-up the turtlebot. It is ready to use. The first thing to do to test if everything is working correctly is bringing up the turtlebot dashboard. If everything is working correctly a dashboard application will show the battery states of the laptop and the turtlebot and other useful information about warnings and errors . If there is something wrong with the turtlebot the problem will be pointed in the dashboard application.

2.3.5 SLAM

One of the packages that are provided in the turtlebot is the Gmapping package. This is a package that performs an SLAM out of the box. The only thing to do is starting the demo and drive the robot around in the environment. After running this software on a bag file previously recorded, was able to recreate the environment map as shown in Figure 6 displayed in Rviz. The Gmapping uses a Rao-Blackwellized particle filter. Where each particle holds his own map. The map is built with the data extracted from the laser scan. [4], [7], [3]



Figuur 6 SLAM with Gmapping

2.3.6 Odometry

Odometry is the use of data from moving sensors to estimate change in position over time. Odometry is used by some robots, whether legged or wheeled, to estimate their position relative to a starting location. The odometry is always an estimate and not an exact location. This is because there is inaccuracy on the sensor data.

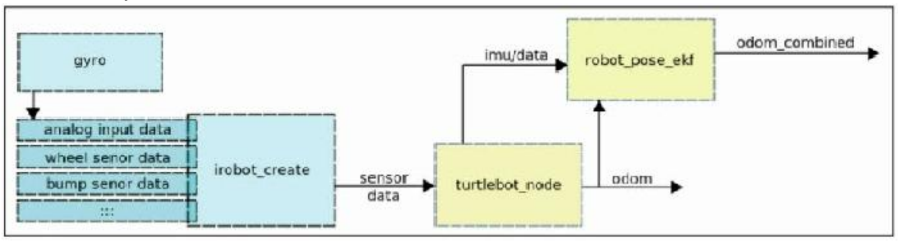


Figure 7 Odometry Turtlebot

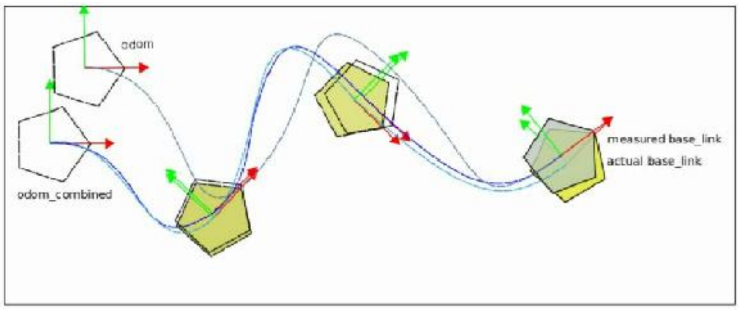


Figure 8 Odometry Comparison

To improve the turtlebot odometry a gyro is added to the turtlebot base. The robot_pose_ekf node use the gyro and odom data to compute and publish the more accurate odom_combined. The robot_pose_ekf is an extended kalman filter (ekf) that uses measurements observed over time containing noise and other inaccuracies. The kalman filter produce values that tend to be closer to the true values of the measurements than their associated calculated values. As a result of this the odometry of the turtlebot is more accurate. The comparison of these results is shown in Figure 8 Odometry Comparison. [9], [17], [18]

2.3.7 Simulators

The power of a simulator may never be underestimated. The use of a simulator has proved his worth in many applications. It is very useful to test new algorithms or experiment with different environments. If it was necessary to build a new environment or a new robot for each application there wouldn't be a lot of improvement of the development of robots.

Before working on the real turtlebot it is useful to first test the code on a simulator. The most used simulators for the turtlebot are Rviz and Gazebo. These simulators provide a good alternative for the real robot. The same topics can be used in the simulator as in the robot.

2.3.7.1 Gazebo

Gazebo is developed in cooperation with player and stage simulators. Player is a networked device server, and Stage is a simulator for large populations of mobile robots in complex 2D domains. A natural complement for these two projects is a high fidelity outdoor dynamics simulator. This has taken form in the Gazebo project. [8]

The development of Gazebo has been driven by the increasing use of robotic vehicles for outdoor applications. While Stage is quite capable of simulating the interactions between robots in indoor environments, the need for a simulator capable of modeling outdoor environments and providing realistic sensor feedback have become apparent. Gazebo, therefore, is designed to accurately reproduce the dynamic environments a robot may encounter. All simulated objects have mass, velocity, friction, and numerous other attributes that allow them to behave realistically when pushed, pulled, knocked over, or carried. The robots themselves are dynamic structures composed of rigid bodies connected via joints. The world itself is described by landscapes, extruded building and other user created objects. Almost every aspect of the simulator is changeable, from lighting conditions to friction coefficients. Gazebo is completely open source this is one of the reasons why gazebo is so widely used. [19]

One of the major drawbacks of the Gazebo simulator is that it requires a lot of computational power because of the high quality graphics and realistic physical engine. The simulator is designed for outdoor applications but the fidelity of this simulation is limited. For example, physics models of soil, sand, grass, and other pliable surfaces normally found in nature are well beyond the scope of the simulator. Future work of the gazebo simulator is implementing programmable features like doors, elevators and lights.

2.3.7.2 Rviz

In computational science and computer graphics, there is a strong requirement to represent and visualize information in the real domain, and many visualization data structures and algorithms have been proposed to achieve this aim. Unfortunately, the dataflow model that is often selected to address this issue in visualization systems is not flexible enough to visualize newly invented data structures and algorithms because this scheme can accept only specific data structures. A solution for this problem is the use of Rviz. [4]

Rviz is a 3D visualization tool that helps to visualize what the robot is seeing and doing. Rviz can visualize robot sensors of all kinds like camera’s, laser scanners, point clouds as well as coordinate frames. Visualizing through Rviz is much easier than looking on raw sensors values. [20]

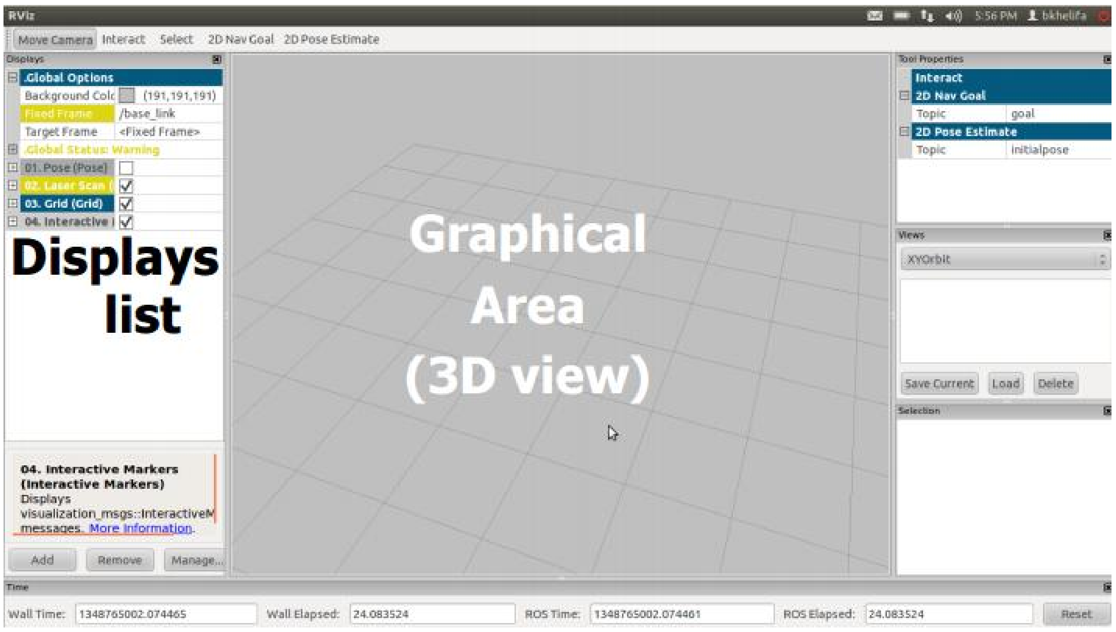


Figure 9 Rviz Overlook

Rviz consist out of simple modular components that are easily added and removed. These components are shown in the right side of the screen as can be seen in Figure 9. If you want to visualize the messages that are published on a topic you can simply add them to your screen. For example odometry messages can be displayed with a simple arrow in Rviz instead of looking at x and y coordinates.

Rviz is often combined with Gazebo, Gazebo simulated the real robot and Rviz visualizes all the measurements done by the robot.

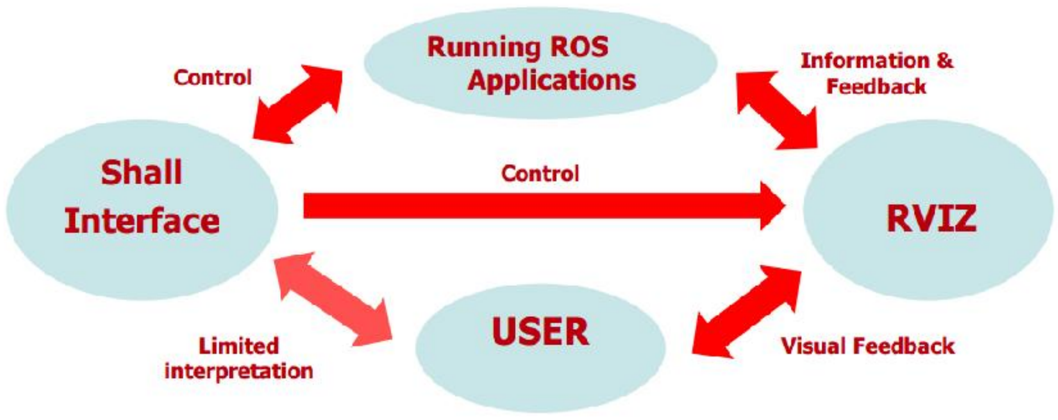


Figure 10 Rviz Structure

3 Particle filter

3.1 introduction

The main goal of a localization method is to estimate the exact location of a robot. A robots location can be defined by its odometry. But the odometry could have a lot of noise and therefore the robot location is inaccurate. Hereby the need of a localization method occurs. One of these probabilistic methods is the particle filter also referred to as the sequential Monte Carlo (SMC) method. This is a localization method which represent the density function using multiple samples called particles. [21]

The particle filter is able to localize itself by combining odometry measurments with observation data from exteroceptive sensor. In our setup this sensor is a Kinect 3D camera. The particle filter is a nonlinear method this means that it can handle nonlinear motion and measurement models moreover it can represent arbitrary and possibly a Gaussian pdf . the output is not directly proportional to the input. This has a lot of advantages , one of these advantages is that the particle filter can solve the global localization problem. This means that a robot is not told its initial pose, but instead has to determine it from scratch. It is also interesting that the particle filter can solve the kidnapped robot problem. In which a well-localized robot is teleported to some other position without being told. This problem differs from the global localization problem in that the robot might firmly believe to be somewhere else at the time of the kidnapping. The kidnapping robot problem is often used to test a robot's ability to recover autonomously from catastrophic localization failures. [22]

When the localization method starts its execution the robots location is unknown all the particles are randomly spread over a known environment. Comparing the actual robot measurement with the predicted sensor measurement at each particle position the particles are weighted. The more weight they have the better that the measurements corresponds. The particle with the most weight has the biggest chance to be the robot position. After weighting, the particles resampling is needed, otherwise there is a risk that after a few iterations there would be only one particle with a lot of weight and the weight of others particles will be almost zero. This is called the Degeneracy Problem. There are different re sampling methods to solve this. The most common one is SIR (Sampling Importance Resampling) . [23]

A particle filter works in three Steps.

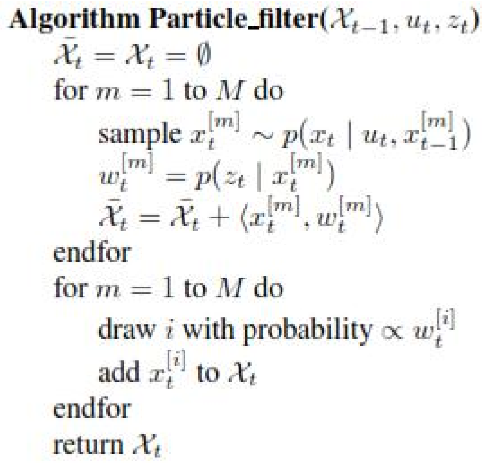
1. **Prediction step**: Each particle is moved by sampling from the motion model of the robot.
2. **Weighting step**: Weights are computed for each of the particles based on how well sensor measurements agree with the particle map and expected location.
3. **Resampling**: Particles are randomly resampled according to their weights. Particles with higher weights are more likely to be replicated while particles with low weights are more likely to be removed.

3.2 Code

All the code is written in python. This gives us probably a slower implementation than a C++ implementation. The class particle filter has the following input arguments:

- **map** : an array of lines in the form [x1 y1 x2 y2]
- **num**: number of particles to use
- **odom_lin_sigma**: odometry linear noise
- **odom_ang_sigma:** odometry angular noise
- **meas_rng_noise:** measurement linear noise
- **meas_ang_noise:** measurement angular noise

The class particle filter is used in a node file where all necessary topics are subscribed and published. A pseudo code can be found in Figure 11.



```
Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):
    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
    for $m = 1$ to $M$ do
        sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
        $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
        $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
    endfor
    for $m = 1$ to $M$ do
        draw $i$ with probability $\propto w_t^{[i]}$
        add $x_t^{[i]}$ to $\mathcal{X}_t$
    endfor
    return $\mathcal{X}_t$
```

Figure 11 pseudo code particle filter

The Class can be divided in three main functions. The prediction step , the weighting step and the resampling step. Furthermore there is the function to get the mean particle position and the initialization. In the initialization step all the particles are random spread over the map with a random orientation. [24]

3.3 Prediction step

The first step is to use a prediction function that increments the particles with the robot movement. First, noise is added to the particles around the current robot position. Next all the particles are incremented with the movement and angle of the robot odometry.

If all the particles are set on the robot starting point and only the prediction step is applied the particles will follow the robots movements. Due the noise the particles will spread further and further. This is shown in Figure 12.

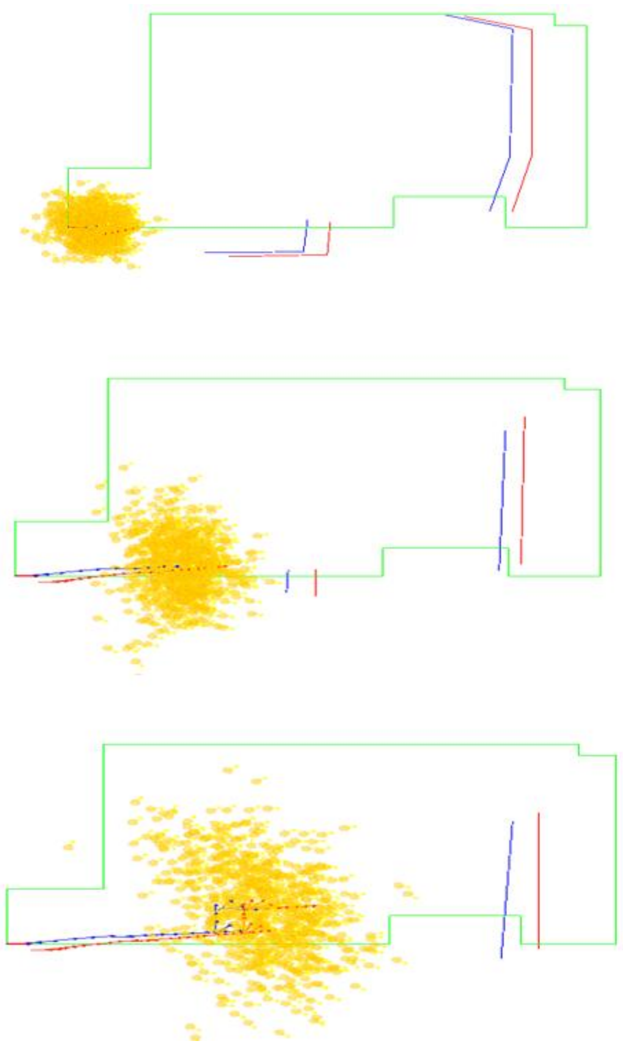


Figure 12 Dead Reckoning

The red lines show the robot scan plotted as the robot estimated by the odometry and the blue lines represents the same measurements plotted of the mean particle.
In the figures you can see how the particles spread when noise is added.

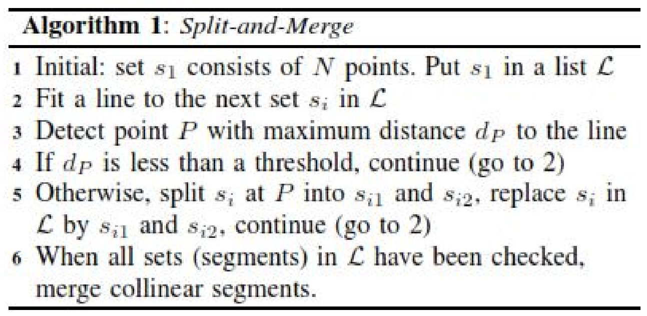
3.4 Weighting step

After the prediction step is done the weighting step is executed. In this step the particles get more or less weight depending on how much the observation predicted from the particle pose does agree with the actual measurement**.** It is worth noting that in our case the robot measurement is a laser scan from one line of the Kinect 3D sensor.

3.4.1 Extract data laser scan

For the measurements, the laser scanner (Kinect) data of the turtle bot is used. It is important to find a good algorithm to convert the raw laser values to a usable set of lines. The Kinect sensor do not get a 2D laser scan but a 3D point cloud. In order to convert this 3D point cloud to a 2D laser scan a rospackage called *pointcloud_to_laserscan* is provided.
Once that a set of 2 D points are received from the scan it is possible to apply a algorithm on the raw values. There are a lot of line extraction algorithms to do so. The most common ones are: split and merge, RANSAC, Line Regression, Incremental, Expectation –maximization and hough transform algorithm. Most of these algorithms are also used for image processing or 3D reconstructions. [23]

we have chosen for the split and merge algorithm, because it is a fast algorithm and is has good results.



```
Algorithm 1: Split-and-Merge
1 Initial: set $s_1$ consists of $N$ points. Put $s_1$ in a list $\mathcal{L}$
2 Fit a line to the next set $s_i$ in $\mathcal{L}$
3 Detect point $P$ with maximum distance $d_P$ to the line
4 If $d_P$ is less than a threshold, continue (go to 2)
5 Otherwise, split $s_i$ at $P$ into $s_{i1}$ and $s_{i2}$, replace $s_i$ in
  $\mathcal{L}$ by $s_{i1}$ and $s_{i2}$, continue (go to 2)
6 When all sets (segments) in $\mathcal{L}$ have been checked,
  merge collinear segments.
```

Figure 13 Pseudo code split and merge

In Figure 13 is a quick overview of the algorithm. What the algorithm does is connecting two points, and searches for the furthest point from this line segment. Initially the first and the last point are selected as shown in figure 14-1. Then the furthest point away from this line segment is selected. The next step is to connect the first point with the new selected point as shown in Figure 14-2. After simplifying the line, the new point and the last point are selected (Figure 14-3). This procedure is repeated recursively until a simplified solution is obtained (Figure 14-4). [25], [26]

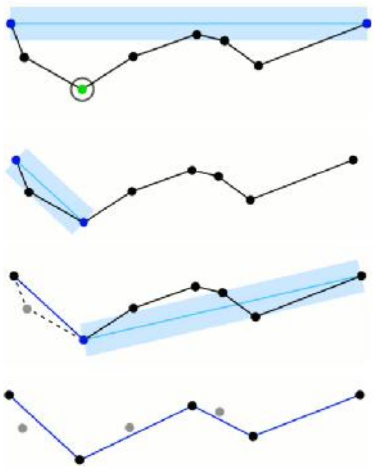


Figure 14 Split and Merge

After the lines are obtained they are converted to polar coordinates for an easier comparison with the map lines.

3.4.2 Weighting the particles

For each observed line and each mapline , the likelihood to the line that the robot actually had observed such particular map line is computed and the one with maximum likelihood is selected as the corresponding line. If only one line has been observed this weight (like hood probability) is used as the particle weight. Otherwise all the likelihoods are multiplied among themselves to obtain the particle weight.

As stated above, the weight of a particle states the likehood of observing the actual robots measurement at the current particle pose. This weight has to be evaluated for each particle. To do so, the map lines have to be represented in polar lines. the likehood function used to compute the weights is shown in Equation 1.

$$w = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

Equation 1

Where x is the measured value (range or angle) ,µ the expected value and σ the uncertainty of the measurement.
With this function the lengths of the lines are not taken into account. If a line of the laser scan is longer than a line of the map the matching is incorrect, therefore the weight of this particular particle is set to zero.

3.5 Re-sampling

This is the most important step to solve degeneracy problem. This means avoiding that one particle get all the weight. Resampling is done by the Stochastic universal sampling algorithm. This method uses a systematic resampling in comparison with roulette wheel resampling which uses a random method. The systematic method is acquired by incrementing a random value with m/M . Where m is the current number of particle and M the total number of particles. As long as this incremental value is greater than the weight the weight is increased (sum of more weights) until the weights are bigger than the incremental value. After that the particles are added to a new particle list. The code can be implemented as.

```
**Algorithm Low_variance_sampler**($\mathcal{X}_t, \mathcal{W}_t$):
    $\bar{\mathcal{X}}_t = \emptyset$
    $r = \text{rand}(0; M^{-1})$
    $c = w_t^{[1]}$
    $i = 1$
    for $m = 1$ to $M$ do
        $u = r + (m-1) \cdot M^{-1}$
        while $u > c$
            $i = i + 1$
            $c = c + w_t^{[i]}$
        endwhile
        add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$
    endfor
    return $\bar{\mathcal{X}}_t$
```

The resampling is executed only when the robot is moving. Otherwise the resampling causes particle impoverishment. This means that good particles are removed if the resampling is applied to many times.

3.6 Environments

Once the particle filtr has been implemented it has to be tested. In our case we have tested it in two different environments. This is done by recording a bag file of the robot exploring an environment while using Rviz for visualization. Using these bag files it is possible to execute the particle filter in a real time.

3.6.1 Bag file

A bag file is a logging format for storing ROS messages in files. Bags are typically created by a tool like rosbag, which subscribe to one or more ROS rostopics, and stores the serialized message data in a file as it is received. These bag files can also be played back in ROS to the same topics they were recorded from, or even remapped to new topics.
Using bag files within a ROS computation graph is generally no different from having ROS nodes send the same data, though you can run into issues with time stamped data stored inside of message data. For this reason the rosbag tool includes an option to publish a simulated clock that corresponds to the time the data was recorded in the file. [4]

There are numerous tools and commands to modify and replaying recorded bag files. So bags are perfect for recording a robot in real time and use the data afterwards in a simulator. The main topics that are recorded for the particle filter application are the laser scan data and the odometry messages.

3.6.2 First environment

The first environment is defined as followed.

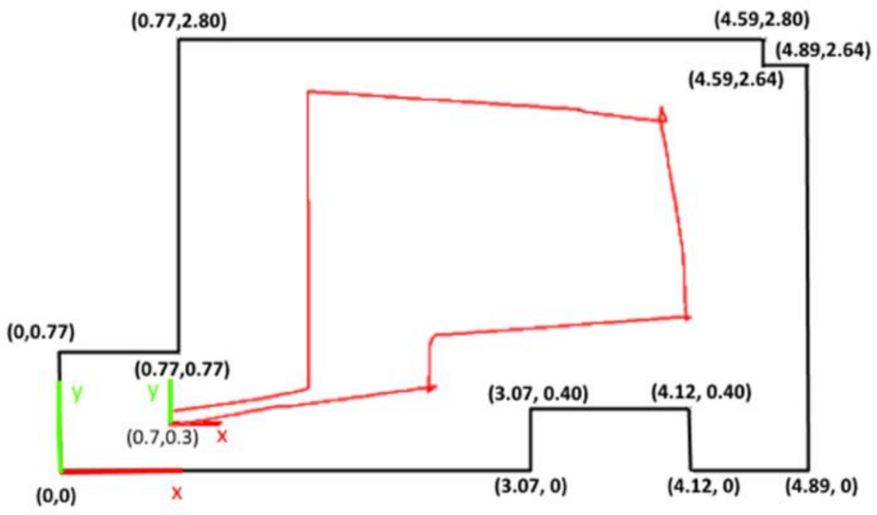


Figure 15 Environment 1

The definition of the first map is shown in figure 15. This displays all the coordinates of the map and the trajectory that is recorded with the bag file. The begin position of the robot is at (0.7,0.3). To transform the robot coordinates to world coordinates a simple translation is sufficient.

3.6.3 Second environment

The second environment is made more complex as the first one. This environment is shown in figure 16.

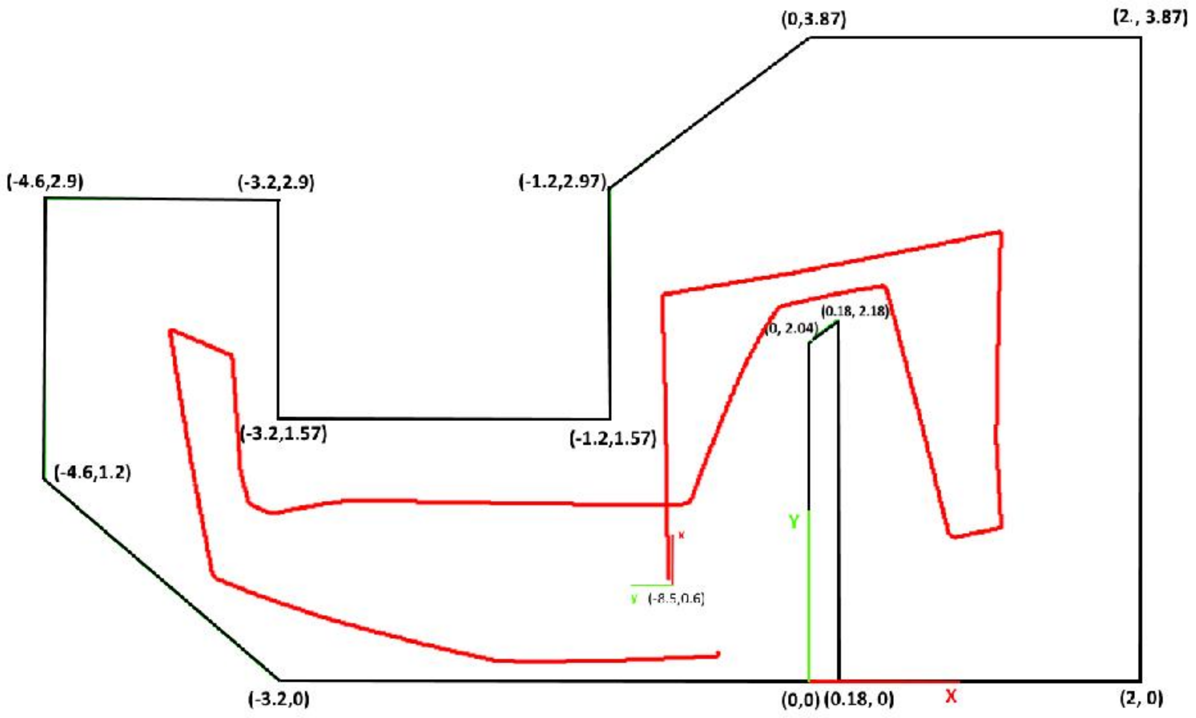


Figure 16 Environment 2

The zero point of the map is defined in the corner. The robot is placed on a offset of (-8.5,0.6) in world coordinates. This is seen as the initial position of the robot. This position is inaccurate because the measurements are done by hand. The trajectory covers more or less the whole environment so that all laser scans are obtained.

In this environment there are landmarks added. These landmarks are spread against the walls of the environment. The use of these landmarks is out of the scope of this thesis but they can be used in the future for visual navigation.

4 Results

4.1 First environment

By executing the particle filter algorithm on a provided bag file and a known map the following results are obtained.

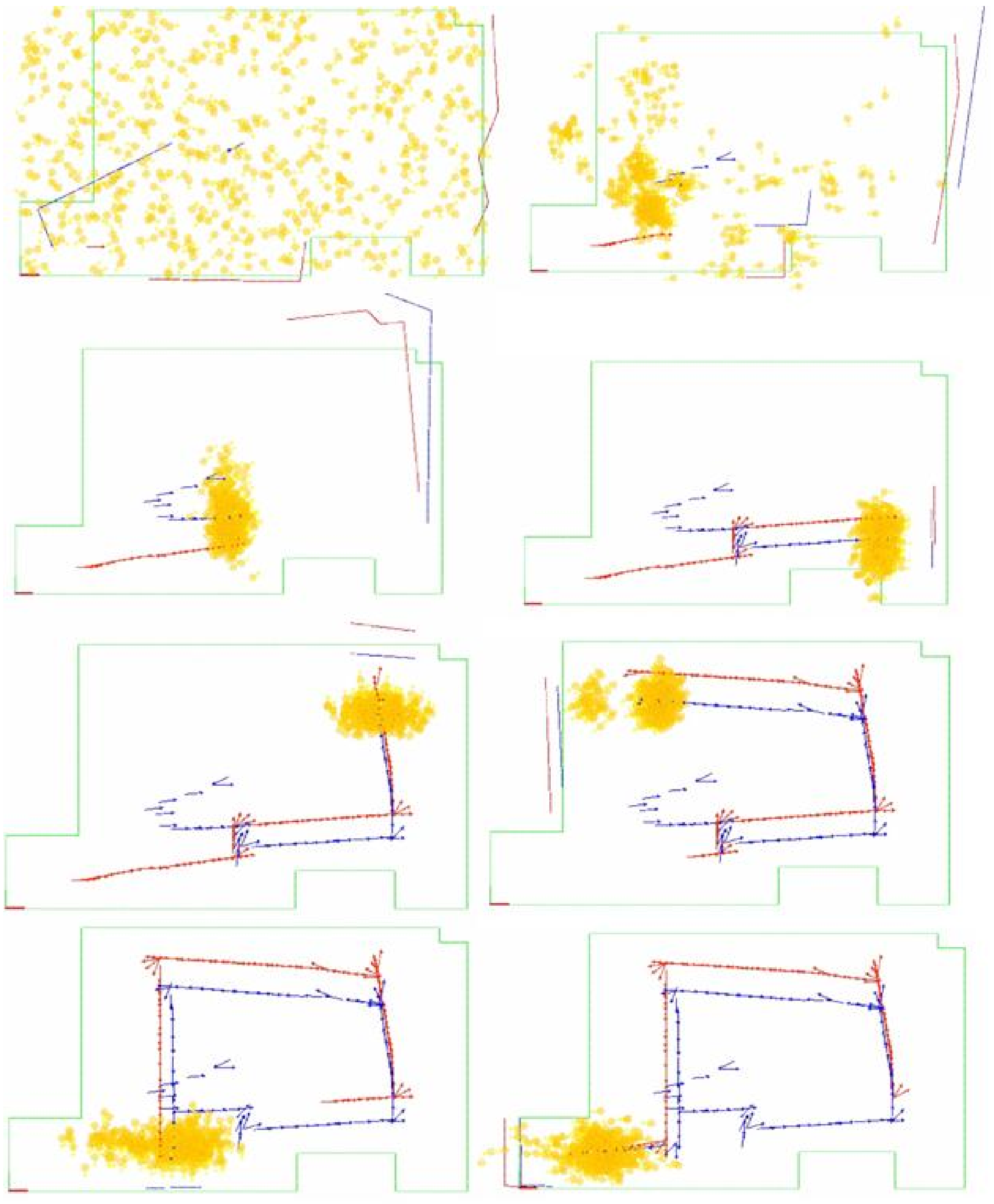


Figure 17 Environment 1 particle filter

The trajectory can be seen in Figure 17 where the blue lines represent the mean particle the green lines the map and the red lines the robots odometry. At first all the particles are spread in the environment. The number of particles used in this simulation is equal to 500. After the robot starts moving, the particles are able to find the robot position after a few iterations. So 500 particles for this simple environment is enough to estimate the robot position.

After that the particles have found the robot they keep track of the robot. In Figure 17 is also noticeable how the particles spread due the noise if the robot is riding forward for a longer time. Once the robot starts turning and measure a wall parallel with his previous movement direction the particle cloud starts narrowing.

These trajectories can also be plotted in matlab with the use of csv files. These files contain the data messages obtained and save them in a comma separated values. These files can be opened with matlab and the data can be extracted to get the following plot.

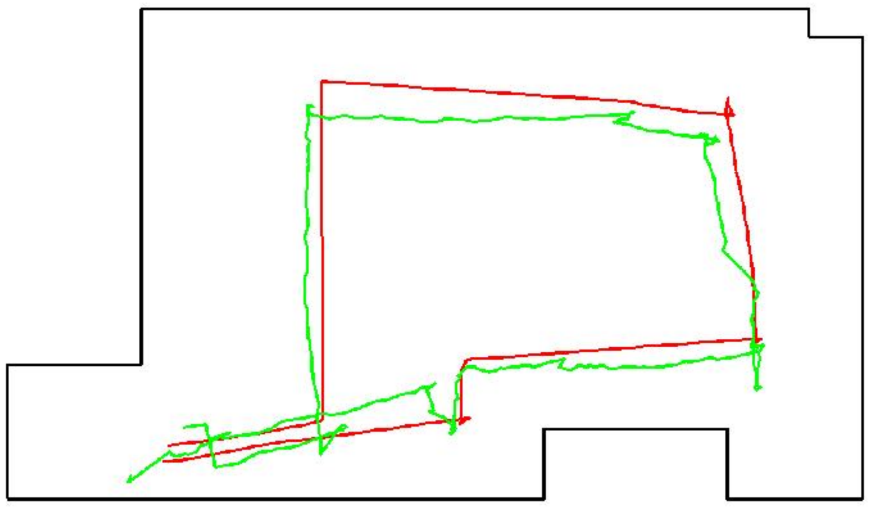


Figure 18 particle filter trajectory

In Figure 18 the whole trajectory of the normal odometry (red) and the mean particle (green) are displayed. The particle filter is able to follow the robot with a certain offset. This offset is due the inaccuracy of the defined map or through comparing wrong lines.

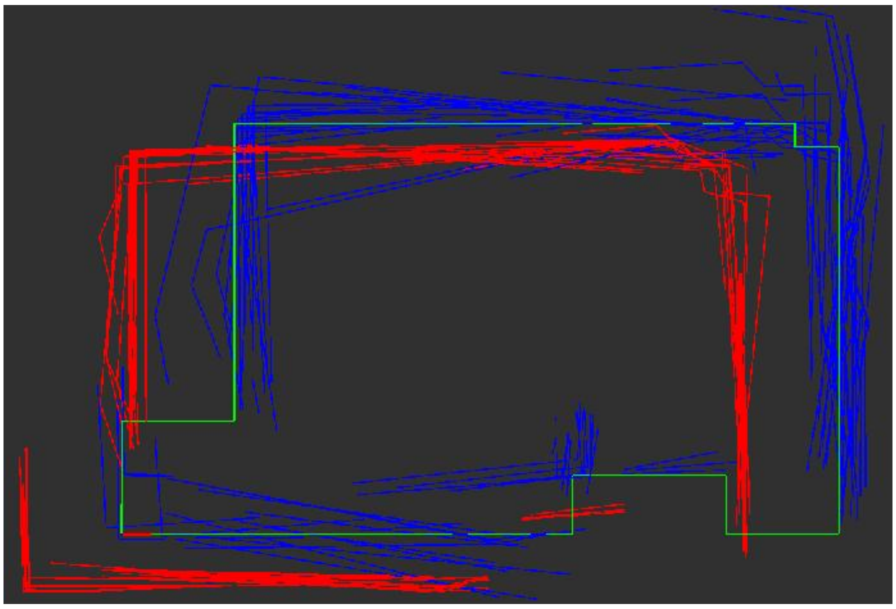


Figure 19

As can be seen on Figure 19 the blue lines (measured by mean particle) are less focused than the red lines obtained from the robots measurement. It is also noticeable that the map is shifted.

4.2 Second environment

After using the code on the first map, the code can be tested on the second map.

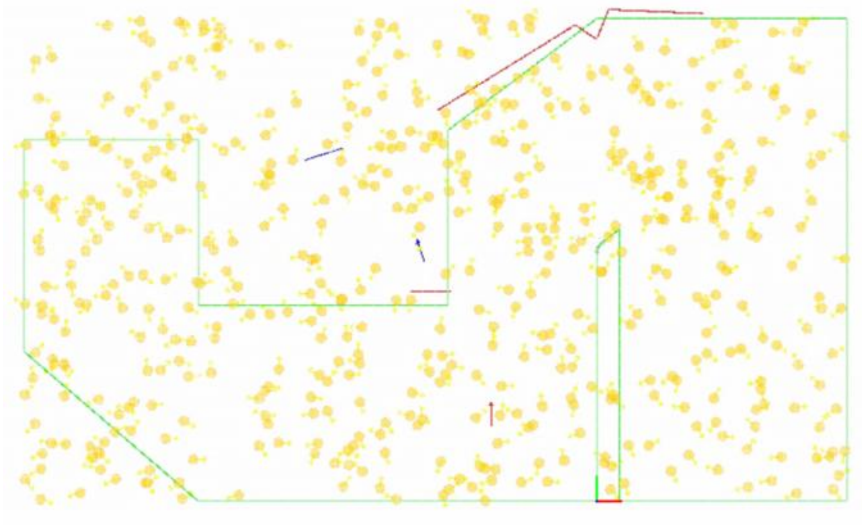


Figure 20 Distribution particles environment 2

In Figure 20 is shown how the particles are spread at the beginning. Once the robot starts moving the particles are weighted and resampled. After a few iteration the following distribution is obtained.

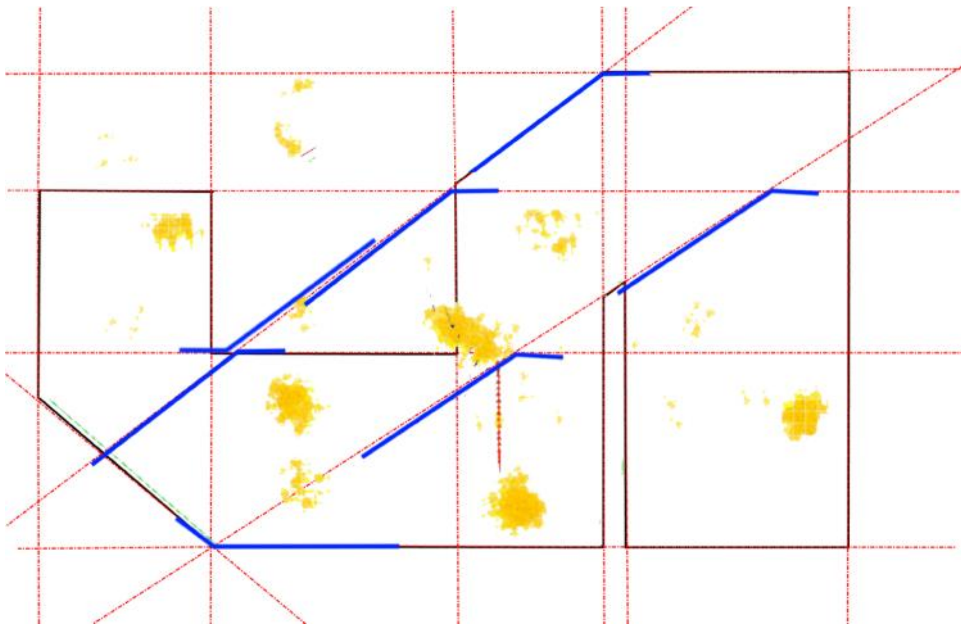


Figure 21 Laser scans environment 2

The blue line corresponds to the laser scan that is observed by the robot.
At the weighting step as it has been explained that all the lines that the particle sees are compared with the map lines. Since all these map lines and the laser scans are in polar coordinates. The distance and the angle of the scan and map are compared to weight the particles. It is possible to have ambiguity in the measurements. An example of this is given in figure 18. The red lines are the map lines in polar coordinates. So the laser scan (blue lines) can be registered at several places. The possible places are shown with the particle distribution. As the robot starts to move the particles get more data about the environment and the distribution of particles becomes less spread.
After a few iteration the following distribution is obtained. Shown in Figure 22

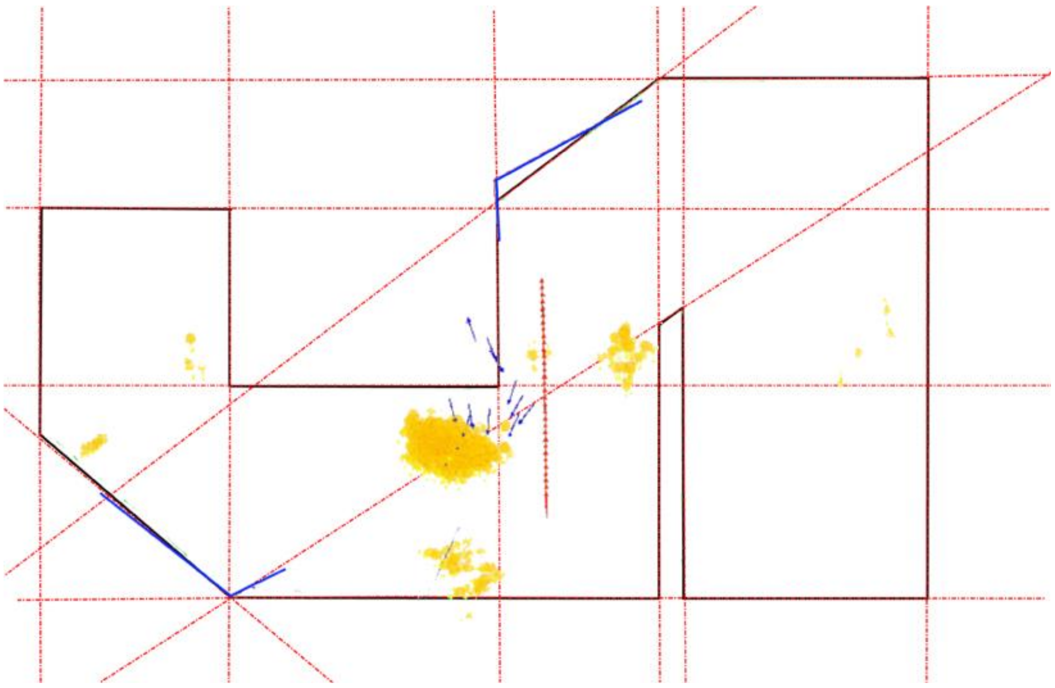


Figure 22 Environment 2 wrong distribution

In this figure it is noticeable that the particles are concentrating in comparison with the previous figure. But the particle distribution is leading to wrong side of the map. It seems that the particles are getting wrong matching's in the map. But still after a few iteration the particles are able to estimate the robots position. This is shown in figure 23.

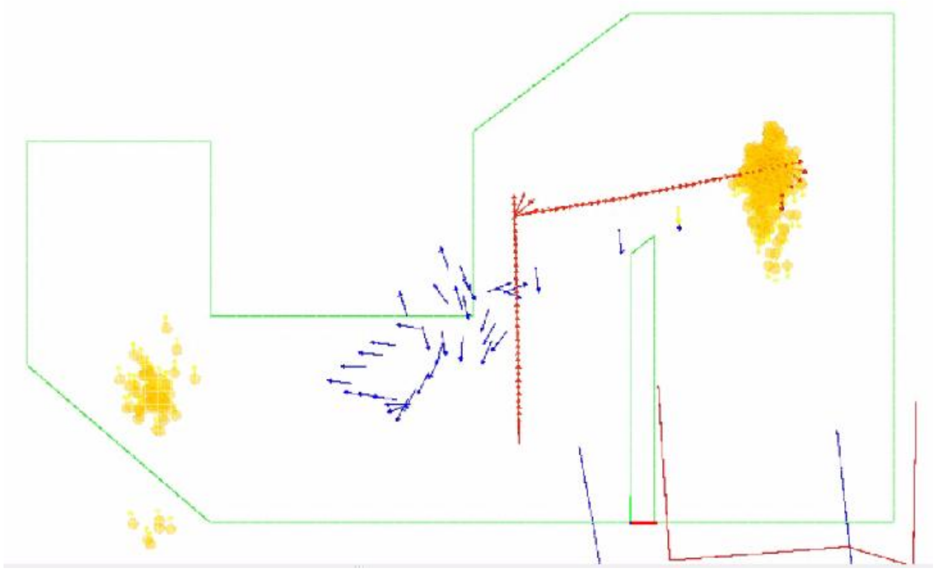
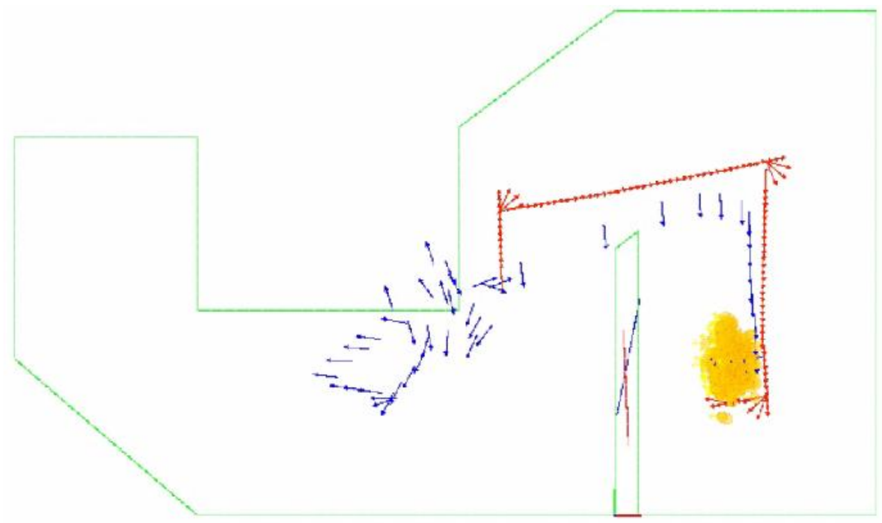


Figure 23 Environment 2 right distribution

Here more and more particles are getting the right position of the robot. After a few iterations more the robots position is better localized.



Figuur 24 Environment right distribution

Once the robot is found the particles will keep track of it and estimate its position. The obtained path with the dead reckoning is shown in figure 24.

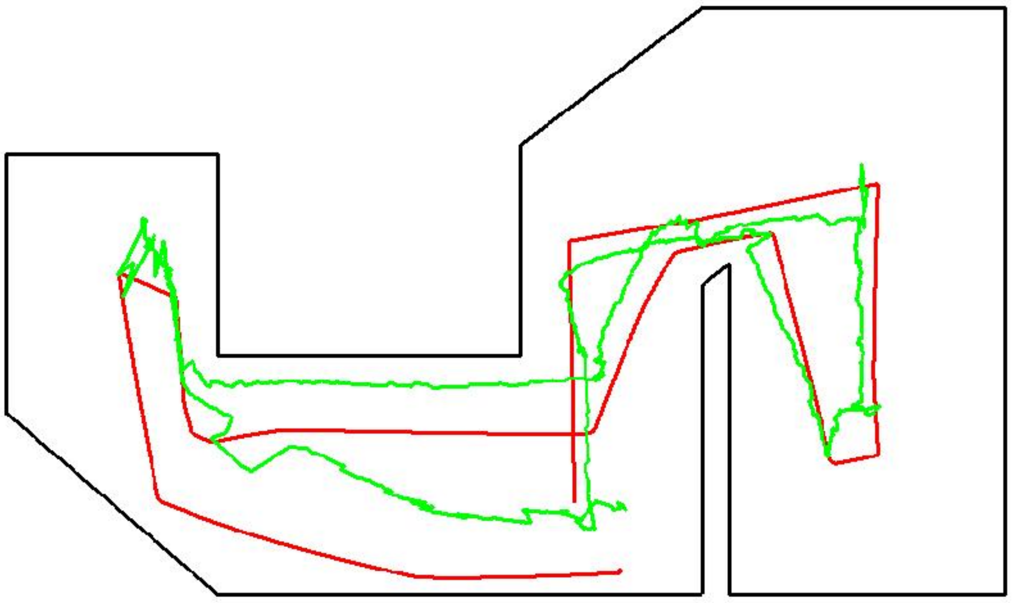


Figure 25 Environment 2 path

This is other execution where the particles where able to find the robot much faster than in the other example. Through the long trajectory the odometry of the robot is inaccurate. Also the starting position is inaccurate determined because it was measures by hand. The assumption was made that the robot was facing exactly 90 degrees with the world coordinates. A small change in this orientation can result in a big change of translation.

To set an idea of the consistency of the results the observed scans have been plotted on the odometry trajectory (red) as well as over the trajectory of the mean particle (blue)(see Figure 26).

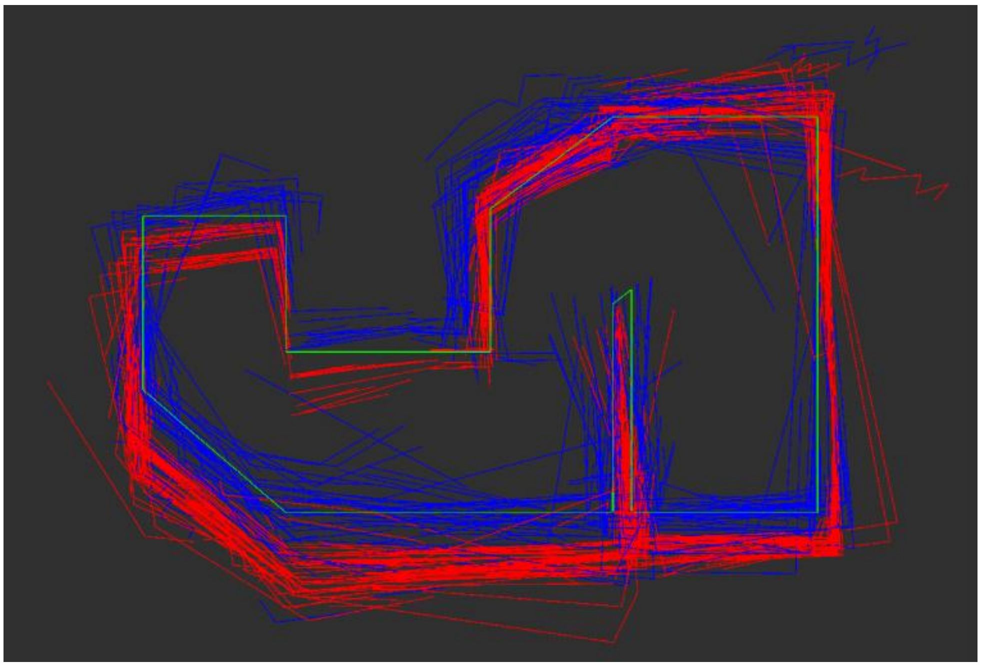


Figure 26 Laser scans

It can be appreciated that the particle filter derived map is more contrived and the correct solution from the odometry derived map, which suffers an offset due the possible error in the selected initial position. In this test the robot executed complete turns to the room it can also be appreciated that there is a drift (typical odometry estimation) between the first and the second turn. this drift is connected by the particle filter and it is not appreciated in the blue map.

Nevertheless, the red trajectory appears more focused (narrower thickness of the walls) than the in the in the odometry map. This is probably due to an excessive noise added to the particles during the prediction.

The blue lines are the laser scans of the mean particle and the red lines the laser scans of the robot. These are the plots of two trajectory's through the map. As you can see the laser scan of the first trajectory are shifted in comparison with the second trajectory. This due the inaccuracy of the starting point and odometry.

The red lines do not match the environment perfectly but it is a good approximation. Because of the shift and the map the simulator is not perfectly accurate.

4.3 Number of particles

Sometimes the use of 500 particles is not enough for a more complex map like this one. Than the robot is unable to locate himself and choses the wrong particles to estimate his position. If the particles are increased the computation time increases a lot because the function is looping much more. Hereby the simulator loses data about the robot odometry, because the new messages are already published before the data of the old one is processed.

To solve this a other initialization method was tested. Because once the robot location is estimated the particles give good results. The normal initialization is to spread the number of particles over the map. This gives us the problem that the robot sometimes uses the wrong polar lines to locate himself. To solve this problem two other approaches of initialization are tested. The first one is to spread the particles around a known begin position to account for the uncertainty of the initial pose of the robot. This is shown in Figure 27.

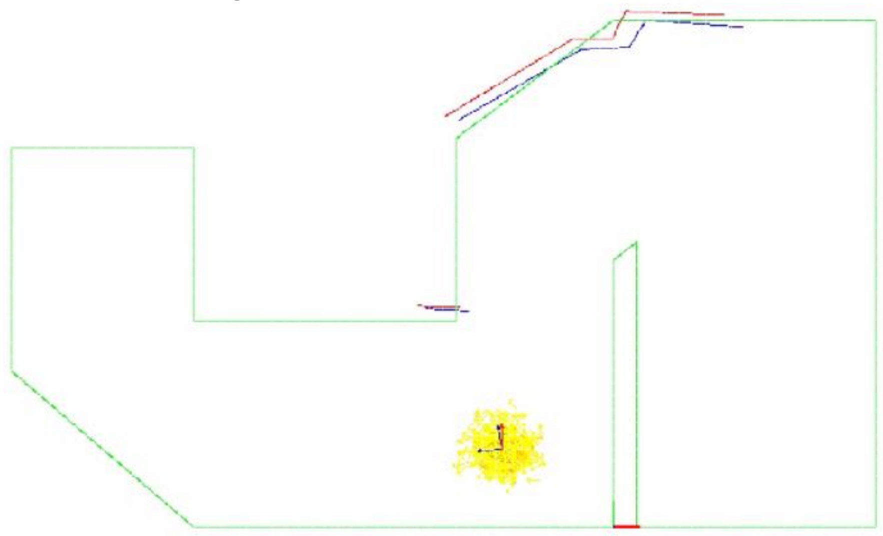


Figure 27 Distribution around starting pose

This is only possible when the robots position is although uncertain. Although we lose the advantage of the capability to locate a robot with a unknown initial pose (global localization), with this method less particles can be used and this benefits the computation time.

The other method that is tested is when the orientation of the robot is known. All the particles are than spread with the same orientation as the robot. In this way the robot can be anywhere in the map only the orientation has to be fixed. This is shown In figure 27.

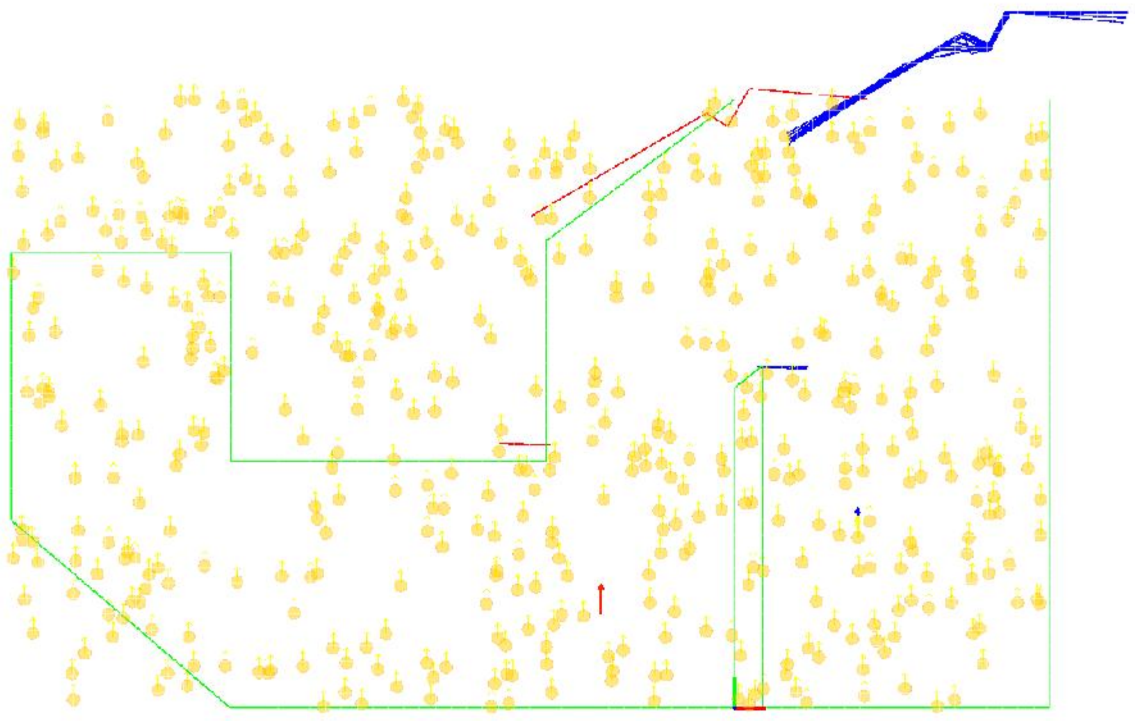


Figure 28 Fixed orientation

This distribution helps the particle filter to estimate the robots position much quicker.

These methods that are discussed are only implemented to estimate the robots position with less particles to save on computation time and to not skip messages. If it was possible to use more than 5000 particles the filter will not have a problem to find the robots location but due the computation effort this is not possible.

5 Conclusion

The main goal of this thesis was to get familiar with robotic applications by the use of ROS and the turtlebot platform. After learning the basics about ROS, python and the turtlebot platform, a particle filter considering odometry displacements with joint scans has been successfully implemented and tested in two different environments.

.

6 Resources

[1] "eHow," [Online]. Available: http://www.ehow.com/about_4596141_importance-robots.html.

[2] "ROS," ROS, [Online]. Available: http://www.ros.org/. [Accessed May 2015].

[3] G. Giorgio, S. Cyrill and B. Wolfram, "improved Techniques for grid mapping with Rao-Blackwelliz Particle Filters," *IEEE TRANSACTIONS ON ROBOTICS*, no. VOL. 23, NO. 1, p. 13, 2007.

[4] "ROS wiki," ROS wiki, [Online]. Available: http://wiki.ros.org/. [Accessed May 2015].

[5] S. G. Mehmet and B. Robert, "A Behaviour-Based Architecture for Mapless Navigation Using Vision," *International Journal of Advanced Robotic Systems*, p. 13, 2012.

[6] S. Thrun, "Robotic Mapping: A Survey," *School of Computer Science*, 2002.

[7] C. Rik, M. Yannick and S. Benjamin, "Graph-based Simultaneous Localization and Mapping on the TurtleBot platform," p. 16, 2013.

[8] O. S. R. Foundation, Open Source Robotics Foundation, [Online]. Available: http://www.osrfoundation.org/. [Accessed 27 5 2015].

[9] M. Ruensuk, "KINECT-ENABLED AUTONOMOUS OFFICE ASSISTANT ROBOT," p. 38, 2012.

[10] B. B. G. a. K. Conley, "Robot Developer Kits," *IEEE ROBOTICS & AUTOMATION MAGAZINE* , 2011.

[11] i. Kobuki, "iClebo Kobuki," [Online]. Available: http://www.rita2012.org/data/Kobuki_introduction_Yujin_Robot.pdf. [Accessed 27 5 2015].

[12] C. –. Robotics, "Notes on Turtlebot robots," p. 13, 2014.

[13] "Wikipedia," [Online]. Available: http://nl.wikipedia.org/wiki/Kinect. [Accessed 25 5 2015].

[14] "iheartrobotics," 17 December 2010. [Online]. Available: http://www.iheartrobotics.com/2010/12/limitations-of-kinect.html. [Accessed 25 5 2015].

[15] "quepublishing," Pearson, [Online]. Available: http://www.quepublishing.com/articles/article.aspx?p=1768327&seqNum=2. [Accessed 25 5 2015].

[16] "Installing and Using the Kinect Sensor," [Online]. Available: http://channel9.msdn.com/Series/KinectSDKQuickstarts/Understanding-Kinect-Hardware. [Accessed 28 5 2015].

[17] "TurtleBot Odometry Primer," [Online]. Available: https://www.youtube.com/watch?v=3S8MXsnNe3U. [Accessed 25 5 2015].

[18] Wikipedia, "Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Odometry.

[19] K. Nathan and H. Andrew, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *IEEE/RSJ international Conference on Intelligent Robots and Systems*, sendai, japan, 2004.

[20] R. K. Hyeong, L. Sung-Ho, P. Taejung and K. Chang-Hun, "RViz: a toolkit for real domain data visualization," *Springer Science+Business* , no. Media New York , p. 9, 2015.

[21] F. Dieter, T. Sebastian, B. Wolfram and F. Dellaert, "Particle Filters for Mobile Robot Localization".

[22] D. Frank, F. Dieter, B. Wolfram and S. Thrunt, "Monte Carlo Localization for Mobile Robots," *IEEE International Conference on Robotics & Automation*, p. 7.

[23] R. Gabriel and R. R. , "Simultaneous Localization and Mapping," p. 59, 2013.

[24] T. Sebastian, B. Wolfram and F. Diter, "Probalistic Robotics," in *Probalistic Robotics* , 2000, pp. 76-82.

[25] wikipedia, "wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm. [Accessed 25 5 2015].

[26] N. Viet, M. Agostino, T. Nicola and S. Roland, "A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics," p. 6.

[27] M. S. Guzel and B. Robert, "A Behaviour-Based Architecture for Mapless Navigation Using Vision," *International Journal of Advanced Robotic Systems,* p. 13, 2012.

[28] A. M. Sanjeev, M. Simon, G. Neil and C. Tim, "A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking," *IEEE TRANSACTIONS ON SIGNAL PROCESSING,* no. VOL. 50, NO. 2,, p. 15, 2002.

[29] F. Dieter, T. Sebastian, B. Wolfram and D. Frank, "Particle Filters for Mobile Robot," in *Statistics for Engineering and Information Science,* New York, Springer, 2001, pp. 401-428.

[30] O. S. R. Foundation, "Turtlebot," Open Source Robotics Foundation, [Online]. Available: http://www.turtlebot.com/. [Accessed 25 5 2015].