Obstacle detection and mapping in low-cost, low-power multi-robot systems using an Inverted Particle Filter

Adam Leon Kleppe, Amund Skavhaug

Obstacle detection and mapping in low-cost, low-power multi-robot systems using an Inverted Particle Filter

Adam Leon Kleppe and Amund Skavhaug

Institude of Cybernetics, NTNU

Abstract. There are a vast number of methods for detecting obstacles, though not taking resource consumption into consideration. The proposed method is an obstacle detection and mapping method focused on systems with constrained memory capacity and processing power, and is called the Inverted Particle Filter. This method has small resource requirements, and can be fine tuned according to available resources. The conducted experiments show promising results.

Keywords: computational power, low-cost, low-energy, power consumption, obstacle detection, mapping

1 Introduction

The real world changes, and therefore obstacle detection is an important capability for robots manoeuvring through it. It is also a key component in creating maps of the environment, which again benefits the robots manoeuvrability. Obstacle detection has gained interest over the years and is widely used in cars, ships, planes and many other automated vehicles.

The problems in obstacle detection has been researched in many years, and there are a number of implementations and methods, each with different advantages and disadvantages. The DARPA Grand Challenge [1] invites participants to solve the general problem, and the outcome of these challenges has been used elsewhere in general. However, these solutions often focus on sensor fusion and redundancy, something which require expensive solutions with numerous sensors and computers. To the authors knowledge, a solution for low-cost, low-power applications in the challenge has not been proposed yet.

Low-power applications has gained more and more attention over the years. Applications with less power consumption, produce less heat and cost less, which are desired properties in both user interface, design and production of a device. Moore's law[2] predicts that there is exponential growth in processing power and memory capacity on a given area of integrated circuit, which again leads to more available processing power and memory capacity.

There are many benefits that come with this increased processing power and memory capacity on smaller circuits. The most common benefit is that implementations and methods can be less concerned about the resource consumption

as the circuits are getting more powerful over time. Thus, newer methods can be more complex with more focus on processing data into complete instances containing more information rather than focusing on the speed and performance of the processing, and the memory capacity required to process the data. Another way to exploit this exponential growth is that the methods that have a low resource consumption can be implemented on smaller circuits. An important benefit of this is reduced power consumption opening for new battery-powered or energy harvesting applications. To take further advantage of this it is more crucial to develop methods that require less resources such that smaller circuits in the new devices and robots, can be developed.

In the fall of 2012 a project to study the possibility of creating a low-cost, low-power multi-robot system, was commenced at the department of cybernetics at NTNU. One of the key features of this system was the obstacle detection and mapping system, which had the constraints of both low cost and low power.

In this paper, a novel particle filter method, the "Inverted Particle Filter", used for detecting obstacles and creating the map in the multi-robot system, is presented.

2 Problem definition

The main objective of the robots used in the experiment, was that they would have low-cost, low power consumption and be small in size. Another target was to make the robots independent of external reference systems.

Low cost and power consumption often requires a computer with low processing power and constrained resources. Therefore it is beneficial with an obstacle detection and mapping approach that has constant and predictable memory consumption and processing time, and that both of these are linearly dependent with the desired resolution of the map.

It is worth noting that low power consumption does not necessarily mean low processing power nor constrained resources. Nonetheless, if a method is less demanding it will in general perform better with restricted resources. Thus it is a goal to develop methods that are more resource efficient.

Since the robots were not using any external reference systems, each robot had to equipped with a sensor to detect obstacles.

Computer vision often use either one or two cameras and a computer to process each frame captured by the camera(s). There are different algorithms to use on a frame or a series of frames to detect objects within the field of view. These algorithms often require intense computation, which is not favourable in this application.

Ultrasonic and laser range sensors are often more affordable, and require only an analogue reading to measure the distance to an object. Laser sensors are more accurate than ultrasonic sensors, since ultrasonic waves spread out and create additional echoes. The drawback with these sensors is that they only measure distances, and do not have a way of determining if there is an obstacle or not.

The two algorithms that are needed in this application is an obstacle detection algorithm and a mapping algorithm. These algorithms can easily be fused together, as once an obstacle is detected, only the position of the obstacle is required to add it to a map.

Mapping an obstacle has the benefits of reusability and that it can be distributed to other applications and even humans. However, using this approach neglects the possible movement of an obstacle. If an obstacle moves and this is not detected the map gets outdated. Another problem with the map is that the map needs to be stored, and the more obstacles detected, the more data needs to be stored in the map. This problem is an issue of decreasing concern, as the memory and storing capacity in microcontrollers grows, as mentioned earlier.

SLAM (Simultaneous location and mapping)[3] is a definition for methods that constructs a map and estimates the position of the robot on the map, based on sensor data. SLAM uses either stereo vision, LIDAR[4] or other sensors that can be used to detect obstacles. When an obstacle gets detected it is placed in the map. Current SLAM methods normally requires large amounts of memory and processing power, which is not well suited for restrictions in processing power and memory capacity. Therefore other methods had to be used in this application.

As time elapses the environment changes, and obstacles may have moved or changed shape, making the map irrelevant. Therefore it is not meaningful to create a map which is larger than a robot can maintain. In this aspect, maintainability depends on how fast the environment changes and how fast a robot can detect the changes.

It would therefore be beneficial to modify the mapping approach such that the robot only keeps a local map around the robot, where obstacles outside a defined boundary are removed from the map. Only the obstacles nearby needs to be avoided, and therefore information about the other obstacles are irrelevant. This also gains the benefit of having a more accurate map, since the obstacles in the map can be frequently updated. The other benefit of a map with boundaries is that the map has a constant size, meaning there is a finite and known number of obstacle. This makes the memory consumption predictable.

There are normally two main approaches of describing an obstacle in a map: The vector approach and the grid approach. Examples of methods for both of these approaches can be viewed in [3] and [5] respectively.

The vector approach tries to describe the obstacle with parameters and attributes. E.g.the obstacle is a circle with a radius of 1 meter in coordinates (1.2, 0.8). The obstacle use little resources when stored, and it is easy to update the obstacle since this only requires to change a few of the parameters. It is though harder to determine how to describe the obstacle, and what the parameters are. It requires much information and sensor data to make the parameters correct, which is a problem when having a limited number of resources. The resolution of the map is potentially infinite, but it is dependent on the error of the parameters.

The grid approach sets up a grid where each cell contains information about what is in the specific position of the cell. Each cell requires little information, usually only a boolean determining if there is something solid in that position or

not. The resolution and error of the map is dependent on the grid size. Therefore complex shapes will not be described well enough if the resolution is to low. There are two main problems regarding limited resources. Since it is beneficial to use a local map around the robot, then as the robot moves, the map changes. In the case of a grid map, each cell needs to use a shift operation to move to the new position. This requires much processing power, especially if the map has a high resolution. The second problem is that all the cells contain the same amount of information. This means that cells which does not contain an obstacle contains information about there not being an obstacle. This is redundant since only cells that contains an obstacle are needed.

A third approach of describing an obstacle in a map is by using particles. This is based on the vector approach, but with some properties similar to the grid approach. Each particle is part of a detected obstacle. As with the grid method, the map resolution is dependent on the number of particles, and does not have to cover the whole size and shape. Complex shapes can be described with a much higher degree than with the grid map since the position of a particle can be floating point. The particles does not have to be shifted, only change their position, making movement much easier. The particles are only contained within an obstacle, meaning that information about empty space does not need to be contained. Therefore, using particles has many benefits when having limited resources.

3 Particle filters

Particle filters[6] are recursive implementations of Monte Carlo-based statistical signal processing. In robotics they are commonly used to find the position of a robot based on measurements of the environment, and is an alternative to the model-based Kalman filter. There are advantages and disadvantages with both filters. One advantage for particle filters is that it is multimodal. If there is ambiguity in the measurements the filter can converge to multiple solutions. This is a required property when detecting multiple obstacles.

A short description of the particle filter can be that to create a finite set of particles, and give each particle in the filter the same movement characteristics as the robot using the filter. As the robot moves, each particle moves in the same manner, and as the robot takes measurements, the particle takes the same simulated measurements. The weight of a particle reflects its consistency of its measurement compared to the measurement of the robot. After a given period, the filter enters a resampling phase where particles with low weight are moved to the position of the particles with high weight with a random offset. This will eventually cause all particles to move to the same position, and thereby giving the most likely position of the robot. Figure 1 shows a particle filter using an IMU to measure the movement of the target. As the target moves, the particles that moves through the walls gets lower weights due to the target being highly unlikely to move through walls. The low-weighed particles are relocated into a more likely position, which converges to a final solution. Notice that there

are many solutions for the position of the target, but as time elapses there is less ambiguity in the measurements and the filter eventually converges to one solution. Here uses a full map of the environment to estimate the position of the target.

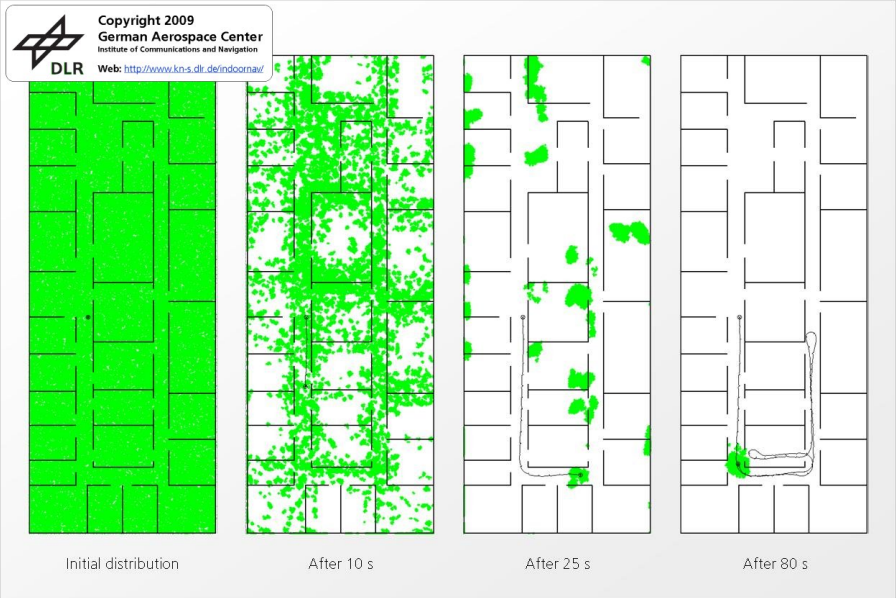


Fig. 1. A particle filter in action. Initially the particles are evenly distributed over a map, but as measurements are taken they move to a more plausible location of the target. Image provided by German Aerospace Center, Institute for Communication and Navigation[7]

The resampling phase can be designed to suit any situation. The most common approach is called the resampling wheel. This approach distributes the particles closer to the particles with high weights, increasing the number of high weighted particles. The more measurements are taken, the less ambiguous the solution gets, leading to less particles with high weights. This cycle eventually makes the particles converge to a solution.

The resampling phase can make or break the particle filter. Particle depletion is one of the major issues when resampling. Particle depletion happens when particles converge to the wrong solution. What happens is that as particles drift around according to the robots motion there may be only one or a few particles within the area with high probability of having the correct solution, and be a lot of particles in very unlikely solutions. The particles with the wrong positions is

overrepresented, and the particles with the correct answer will eventually give in and move to a different location making it very hard to converge to the correct solution unless the filter restarts.

Particle filters have some beneficial properties for low-cost, low-power systems. First of all, there are always a constant number of particles in the filter. This leads to a predictable usage of memory. Secondly, the update sequence of a particle filter is $O(n)$ which leads to a constant processing time. This means that both memory and processing time are linearly dependant on the number of particles. And thirdly, each particle can be designed to have whatever state is needed, making the particle filter a versatile approach suited for many needs.

3.1 Inverted Particle Filter

Particle filters are often used to find the position of a robot based on its measurements of the environment. The *Inverted Particle Filter* described below inverts the problem: It finds the position of the environment based on its measurements of the position.

The output from the Inverted Particle Filter is a map of particles, with the robot always in the center. The map is therefore relative to the robot. In this application the map is circular. The weight of each particle represents the likelihood of the particle being a part of an obstacle. The weights are values between 0 and 1. The weight of 0 represents the particle not being part of an obstacle, 1 represents the particle being part of an obstacle, and when the weight is 0.5 it cannot be determined. We define low-weighted, mid-weighted and high-weighted particles to have weights close to 0, 0.5 and 1 respectively.

As the robot moves, each particle on the map will move in the opposite direction. This reflects the position of the particle relative to the new position of the robot, as shown in Figure 2.

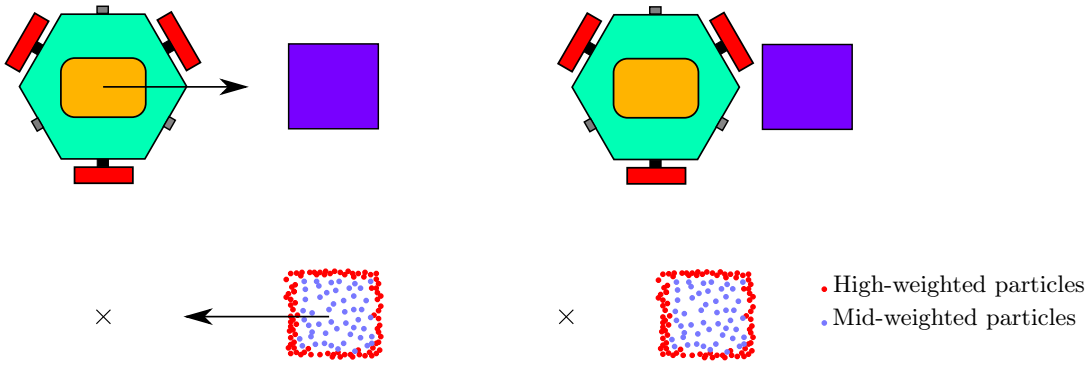


Fig. 2. A movement of the robot is represented as all the particles moving in the opposite direction in the particle filter

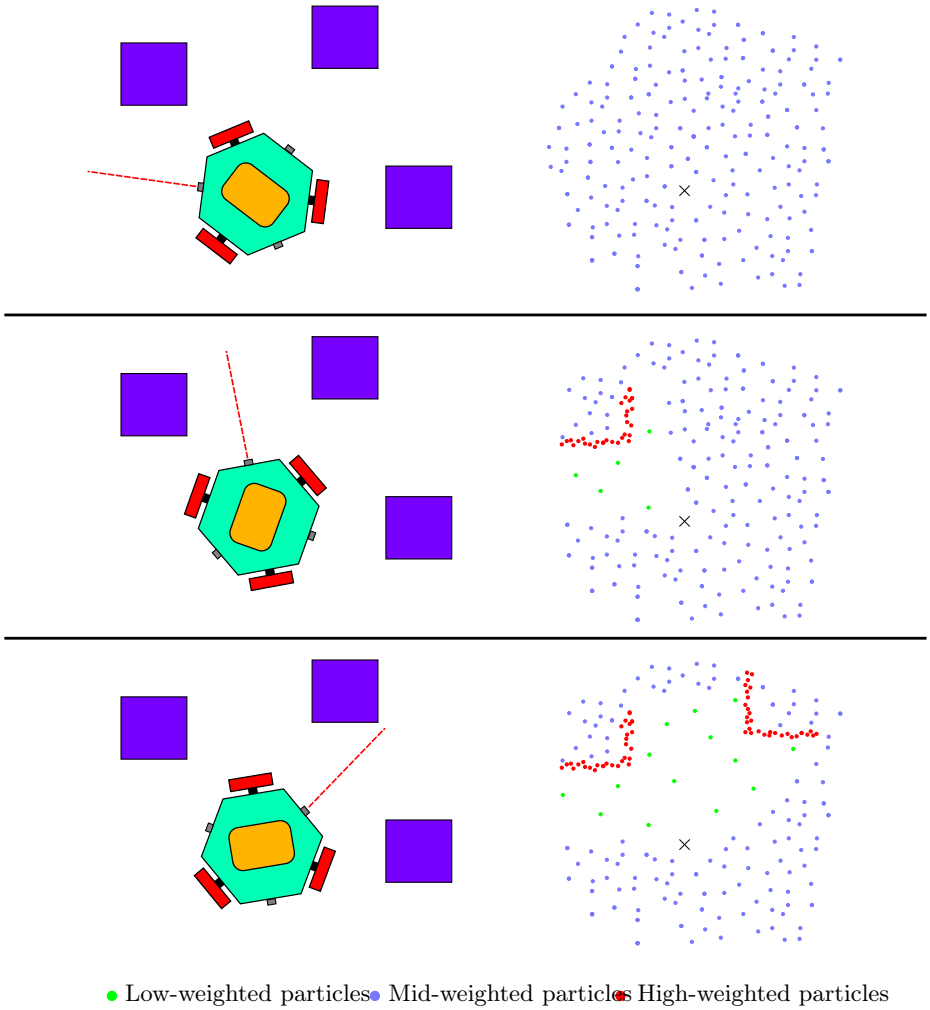


Fig. 3. On the left: A robot rotating while taking laser measurements in an environment with three obstacles. On the right: The particles in the Inverse Particle Filter as the filter is updated.

In our application a laser measurement is used to detect the distance to an obstacle, but any sensor that measures the distance to an obstacle can be used. Whenever a measurement is taken, the filter is updated. The measurement together with the orientation of the robot creates a polar coordinate. A line is then constructed from the center of the map to the measurement coordinate. The line represents, in this application, the actual laser beam. All of the particles close to the line are given a low weight, i.e. unlikely being part of an obstacle. The particles close to the measurement coordinate are given a high weight, i.e. likely being part of an obstacle. If the line runs through a particle there is nothing there, but particles where the line stops are most likely part of an obstacle. Figure 3 shows the particle filter in action. It can be seen here that at first the particles are evenly distributed, but as more measurements are taken, more of the particles move towards the position of the obstacles.

There are many benefits of using the Inverted Particle Filter. It has a predictable and constant memory usage and processing time, and depending on the number of particles in the filter. The detail of the map is also dependent on the number of particles; the more particles the more detail and obstacles can be detected. The filter is also versatile. As long as the weights are set according to the measurements, the filter will construct the map. If there are many robots using Inverted Particle Filters then the particles can be distributed among the robots. As mentioned, our application uses circular maps. This means that the robots can easily detect when the maps overlap. The particles within the overlap can be shared making obstacles detected by one robot available to another. The particles can also be sent and stored in an external computer creating a larger more connected map.

However, there are numerous of potential problems with the resampling phase in the Inverted Particle Filter. Because of this there are some guidelines that should be followed:

1. Mid-weighted particles should be untouched when resampled. This is because the particles in the mid range tells the filter that it is not known what is in the particles location. Moving such a particle will cause the filter to assume that the location is empty, potentially hiding an obstacle.
2. If the obstacles can move, there should always be particles in the empty space around the obstacle. This is to avoid particle depletion. If the obstacles move to a location where there are no particles, the obstacle will not be detected.
3. If particles move outside the boundaries of the map, then they must be relocated to another place on the map. This is because particles outside the map are unused.
4. If the robot moves, then particles with weights of 0.5 should enter from the outer boundaries, representing unknown terrain. This combined with point 3 means that particles outside the boundaries should reenter on the other side of the map. An unfortunate part of this is that there will be an uneven stream of particles reentering the filter due to the particles clustering together when they detect an obstacle. To get an even stream, it is therefore wise not only

to use the particles outside the map, but also the particles inside the map. Figure 4 shows an example of this.

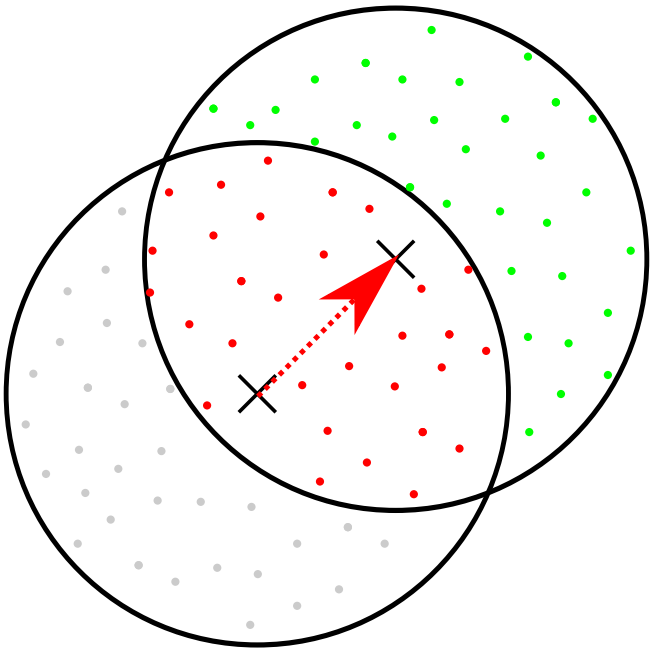


Fig. 4. The particle filter when the robot moves. The particles with grey colour are outside the map and reenteres as the particles shown in green

Using the variables in Table 1 the initialization and update algorithms for the Inverted Particle Filter can be viewed in Algorithm 1 and Algorithm 2 respectively. In Algorithm 1 it is seen that the particles are spread out evenly in a circle around origo (which is the robot). It can also be seen that the particles are rotated in the opposite direction of the robots rotation. Algorithm 2 can be split into three sections. First the weight of each particle moves toward to 0.5. This is due to the increase in uncertainty of the weight over time. After this the algorithm uses the most current measurement and constructs a line from the center of the map to the coordinate of the measurement as described earlier, and sets the weights of the particles according to their distance to the line. The last section moves the particles according to the robots movement, and if the particles move outside the boundaries of the map, the particle is removed and a new particle is created on the other side. The algorithm for the resampling phase is not presented. The resampling method used in the experiment is a resampling wheel method with a slight modification. The method is only efficient if both the robot and the obstacles are stationary, which is sufficient for this experiment.

Algorithm 1 Pseudo code for initializing the Inverted Particle Filter algorithm

Require: p, θ, ω
Ensure: $p.length <= r$
  for all Particles i **do**
    // set the position and angle of the particle
    $p^i = rand(x, y);\ where\ x^2 + y^2 <= r$
    $\theta^i = \phi + \pi;$
    $\omega^i = 0.5$
  end for

Algorithm 2 Pseudo code for updating the particle filter the Inverted Particle Filter algorithm

Require: $p, \theta, \omega, y, m, e_y, e_m$
  $Line\ l = Line((0,0), y)$
  for all Particles i **do**
    // reduce the information in the filter
    if $\omega^i < 0.5$ **then**
      $\omega_t^i = \omega_{t-1}^i \cdot (1 + e_\omega)$
    else
      $\omega_t^i = \omega_{t-1}^i \cdot (1 - e_\omega)$
    end if

    // update regarding measurement
    $d_y =$ closest distance from p^i to y
    $d_l =$ closest distance from p^i to l

    if $d_y \leq l_y$ **then**
      $\omega^i = 1 - \frac{d_y}{l_y} \cdot e_y$
    else if $d_l \leq l_y$ **then**
      $\omega^i = \frac{d_l}{l_y} \cdot e_y$
    end if

    // update regarding movement
    $p_t^i = p_{t-1}^i - m$
    if $length(p^i) > r$ **then**
      remove particle
      create new particle on the other side
    end if
    $\theta^i = \phi + \pi$
  end for

Table 1. Variables used in the Inverted Particle Filter

p	The position of the particle
θ	The angle of the particle
ω	The weight of the particle
y	The measurement of the robot
m	The movement of the robot
ϕ	The rotation of the robot
e_y	The probability of error in the measurement
e_m	The probability of error in the movement
e_ω	The probability of error in the weight
r	The radius of the Inverted Particle Filter map
l_y	The length of the beam of the measurement

4 Experiment

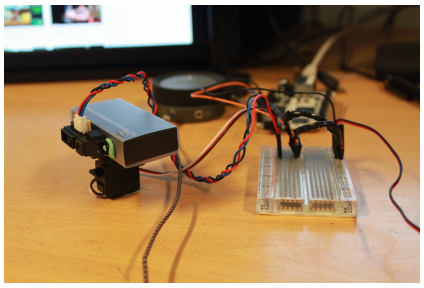
The experiment was set up with a laser[1] placed upon a servo[2]. The laser and servo were controlled by an application which ran on a Beaglebone, which is a single creditcard-sized board computer provided by Beagleboard.org/bone with a Linux operating system. The application was implemented in C++. The servo was set up to move from 0° to 90° with the step of 1° after each time the laser took a measurement. This happened at an interval of 500 ms. Each measurement outputted an analogue signal ranging from 370 to 3560 with a resolution of 16-bit. The signal was transformed in a lookup table to output the measurements in centimetres. The application treated every measurement over 80 cm as 80 cm, since these measurements were more unpredictable. Each measurement was complimented with a the (known) angle of the servo, resulting in a polar coordinate relative to the laser. A various number of obstacles were placed in the field of view of the laser as shown in Figure 5(c).

After the program started, the servo moved from 0° to 90° then back to 0°, while the laser measurements were taken. The filter was updated after each measurement. When the servo reached 0°, each particle was saved in a CSV-file and transmitted to an external computer. Images of the setup of the experiment can be viewed in Figure 5.

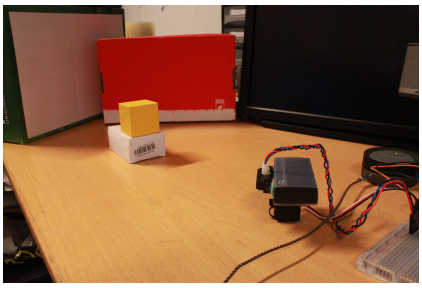
5 Results and discussion

In Figure 6(a) the contours in the distribution of the particles represents the obstacles in the environment. There is only one quadrant in the plot that has any valuable information, Figure 6(b) is a plot of this, the upper left quadrant. There were no measurements taken in the direction of the other three quadrants. and therefore particles within those do not contain any information

[1] The laser was a Sharp GP2Y0A21YK IR Range Sensor acquired from Robonor.no.
[2] The servo was a 9g Small Servo acquired from Sparkfun.com.



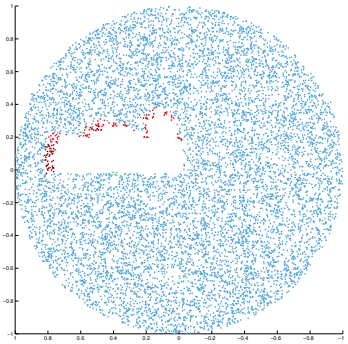
(a) View of the laser on top of the servo(left) and the Beaglebone(back)



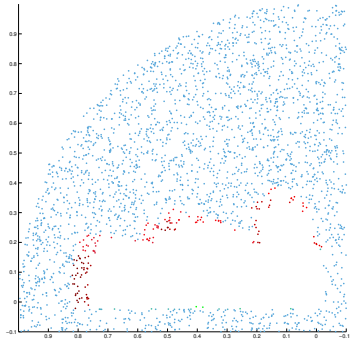
(b) Overview of the testing area

(c) Top view of the testing area. The laser can be viewed in the centre bottom of the image, while the obstacle terrain can be viewed in the top half of the image

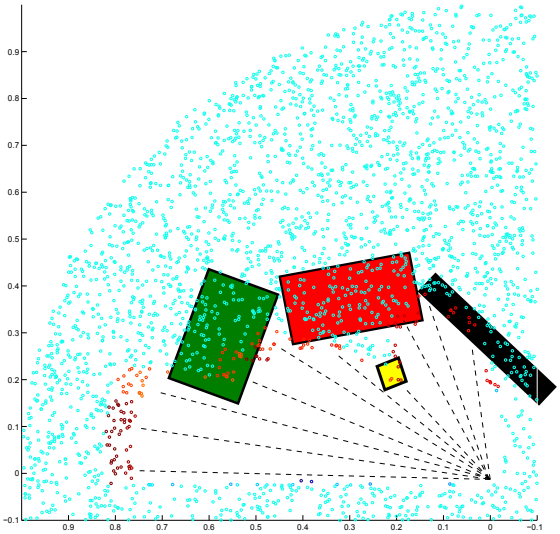
Fig. 5. Images of the set-up and testing area



(a) Plot showing all the particles after the experiment was terminated. Red colour is a high-weighted particle, blue is a low-weighted particle and turquoise is a mid-weighted particle



(b) Zoomed in plot of Figure 6(a).



(c) Overlap of Figure 6(b) over the setup environment

Fig. 6. Plots of the particles in the Inverted Particle Filter after the experiment terminated. The colour of a particle is based on the weight ranging from green(low) to blue(mid) to red(high).

The contours of the obstacles that can be seen in Figure 5(c) are detected and plotted in Figure 6(b). An overlap of the plot in Figure 6(b) and the setup of the obstacles is found in Figure 6(c). The particles printed with red color have a high weight and represents the contours of an obstacle. Note that the particles in red with a distance of more than 0.8 m from the center represents as the maximum range of the sensor used is 80 cm.

It can be seen in Figure 6(b) that there are no particles in the area where there are no obstacles. This is, as mentioned earlier, problematic if either the robot or obstacles move, but in this experiment it makes the figure more clear.

It is worth noting that there has been conducted experiments with the robot moving while taking measurements. The experiments were successful, but the reaction time of the laser is too slow. The laser's measurements does not give the exact distance of the laser beam instantly. If the distance of the laser beam changes, the measurements gradually converges to the distance. Therefore the laser measurements cannot keep up with the robot's movements if it moves to quickly.

Each particle in the implementation only used 5 single-float values, meaning a filter of 10 000 particles only require 50 kB of memory on a 32-bit processor. An approximate measurement of the memory usage of the filter was measured to 1.5 MB on the Beaglebone. The measurement is a rough estimate since there were no simle way of measuring memory usage on the Beaglebone with our selected tools.

This experiment was also done when the Beaglebone was replaced with an Arduino Mega, a single board microcontroller using an 8-bit ATmega1280. The experiment was successful, implying that the filter in general can be implemented even on smaller microcontrollers. Note that the number of particles was reduced to 1000 particles.

An objection that might be raised here in these experiments is that the particles do not represents avoidable obstacles. These could be found by creating lines between neighbouring high-weighted particles which then again could be combined to form obstacles, or other feature extraction methods used in visual processing. In contrast to the motivation many of these methods are computationally intensive. However, if the system is too constrained to perform such a processing, the filter could easily just return the nearest particle to be avoided.

6 Conclusion

The Inverted Particle Filter is a tool for detecting and mapping obstacles, with a focus on limited resources. The main advantages of the filter is that it requires small amounts of resources both for executable code and memory while running. These resource requirements for the filter are predictable and, the filter can be tuned according to the limitations of the system.

References

1. Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.e., Koelen, C., Markey, C., Rummel, C., Niekerk, J.V., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P.: Stanley : The Robot that Won the DARPA Grand Challenge. Journal of Field Robotics **23**(April) (2006) 661–692
2. Moore, G.E.: Cramming more components onto integrated circuits. **38**(8) (1975)
3. Thrun, S.: The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. The International Journal of Robotics Research **25**(5-6) (May 2006) 403–429
4. Campbell, J., Wynne, R.: Introduction to Remote Sensing. (2011)
5. Chae, H., Yu, W.: Artificial landmark map building method based on grid SLAM in large scale indoor environment. 2010 IEEE International Conference on Systems, Man and Cybernetics (October 2010) 4251–4256
6. Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R., Nordlund, P.j.: Particle Filters for Positioning , Navigation and Tracking. Signal Processing, IEETransactions **50**(2) (2002) 1–13
7. Robertson, P.: http://www.kn-s.dlr.de/indoornav/