Tampere University

Tuomas Lauttia

# ADAPTIVE MONTE CARLO LOCALIZATION IN ROS

Bachelor’s Thesis
Faculty of Information Technology and Communication Sciences
Examiner: Prof. Joni Kämäräinen
October 2021

# ABSTRACT

Tuomas Lauttia: Adaptive Monte Carlo Localization in ROS
Bachelor's Thesis
Tampere University
Computing and Electrical Engineering
October 2021

---

The purpose of this work was to gain insight into the world of robot localization and to understand the characteristics of the algorithms widely used for this task. The algorithm chosen for inspection was the Adaptive Monte Carlo Localization (AMCL) algorithm. AMCL is one of the most popular algorithms used for robot localization.

AMCL is a probabilistic algorithm that uses a particle filter to estimate the current location and orientation of the robot. The algorithm starts with an initial belief of the robot's pose's probability distribution, which is represented by particles that are distributed according to such belief. As the robot moves, the particles are propagated according to the robot's motion model. Data from the robot's odometry sensors and range finders are used to evaluate the particles based on how likely it is to obtain such sensor readings at the current pose estimate. The particles are then resampled, with the higher-ranked particles having a higher likelihood of being sampled. Eventually, the lower likelihood particles will disappear and the algorithm will converge towards a single cluster of higher likelihood particles. If localization is successful, this would be near the true pose of the robot.

The AMCL algorithm was tested using a randomly selected subset of the HouseExpo dataset. The HouseExpo dataset contains 2D binary maps of indoor areas. The algorithm was tested in the specific case of global localization, where the initial particle filter estimate is a uniform distribution over the entire map. The tests were run using Gazebo for simulating the robot, ROS for communication and controlling the robot, and Matlab for running the AMCL algorithm and visualizing the particle filter. The localization process was done 10 times for each map, for a maximum of 100 particle filter updates. The runs were evaluated based on whether the AMCL algorithm converged near the true position of the robot and how many steps it took until convergence.

Overall the AMCL algorithm performed quite well, converging near the true position of the robot 878 times out of 1000 test runs. For 53 of the 100 maps, the algorithm converged to the correct position 10 times out of the 10 test runs. The number of rooms and number of vertices per map doesn't seem to significantly affect whether the algorithm converges to the correct position or not. The test results indicate a slight increases in the average amount of steps required until convergence as the number of rooms increase. Symmetric areas and a high amount of similar-looking rooms seem to cause the most problems for the algorithm.

Keywords: ROS, AMCL, particle filter, robot localization

# TIIVISTELMÄ

Tuomas Lauttia: Adaptive Monte Carlo Localization ROS:ssä
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikan TkK-tutkinto-ohjelma
Lokakuu 2021

---

Tämän työn tarkoituksena oli tutkia robotin paikannuksessa käytettyjä algoritmeja ja saada parempaa ymmärrystä niiden ominaisuuksista. Tutkinnan alle valittu algoritmi oli Adaptive Monte Carlo Localization (AMCL) -algoritmi, joka on yksi eniten käytetyistä algoritmeista robotin paikannuksessa.

AMCL on probabilistinen algoritmi, joka käyttää partikkelisuodatinta robotin paikan ja orientaation ennustamiseen. Algoritmi alustetaan todennäköisyysjakaumalla, joka edustaa arviota robotin asennosta. Partikkelit valitaan tästä todennäköisyysjakaumasta. Kun robotti liikkuu, partikkeleiden sijaintia ja orientaatiota päivitetään robotin liikemallin mukaisesti. Robotin tunnistimista saatavia mittauksia käytetään partikkelien arvottamiseen siten, että sijainnit, joista on korkeampi todennäköisyys saada kyseinen mittaus, saavat korkeamman arvon. Partikkelit uudelleennäytteistetään niille laskettujen painoarvojen perusteella, jolloin korkeamman todennäköisyyden saaneet partikkelit pysyvät mukana ja pienemmän todennäköisyyden saaneet partikkelit kuolevat pois. Lopulta kaikki partikkelit keskittyvät yhteen rykelmään, joka sijoittuu lähelle robotin todellista sijaintia, jos paikannus onnistui.

AMCL -algoritmia testattiin käyttämällä HouseExpo -tietoaineistosta satunnaisesti valittuja karttoja. HouseExpo -tietoaineisto sisältää kaksiulotteisia binäärikarttoja sisätiloista. Algoritmia testattiin globaalin paikannuksen tapauksessa, jolloin partikkelisuodatin alustetaan koko kartan laajuisella tasajakaumalla. Testit suoritettiin käyttämällä Gazeboa robotin simulointiin, ROS:ää kommunikointiin ja robotin ohjaamiseen ja Matlabia AMCL -algoritmin ajamiseen ja partikkelisuodattimen visualisointiin. Paikannusprosessi suoritettiin 10 kertaa per kartta niin, että partikkelisuodatinta päivitettiin maksimissaan 100 kertaa. Testiajot arvioitiin sen perusteella, kuinka monta kertaa algoritmi konvergoitui kohti robotin oikeaa sijaintia ja kuinka monta partikkelisuodatinpäivitystä konvergoitumiseen keskimäärin vaadittiin.

Kaikenkaikkiaan AMCL -algoritmi suoriutui testeistä melko hyvin. Yhteensä 1000:sta testiajosta algoritmi konvergoitui robotin oikeaan paikkaan 878 kertaa. Yhteensä 100:sta testeissä käytetyistä kartoista algoritmi konvergoitui oikeaan paikkaan 10 kertaa 10:stä 53:lla eri kartalla. Kartalla olevien huoneiden tai kulmien määrä ei näytä vaikuttavan merkittävästi algoritmin konvergoitumiseen. Konvergoitumiseen vaadittavien partikkelisuodatinpäivitysten määrä näyttää nousevan hieman huoneiden määrän kasvaessa. Symmetriset alueet ja kartat, joissa on suuri määrä samankaltaisia huoneita aiheuttavat algoritmin konvergoitumiselle suurimpia haasteita.

Avainsanat: ROS, AMCL, partikkelifiltteri, robotin paikannus

# PREFACE

This document was written as a Bachelor’s Thesis for the department of signal processing at Tampere University.

Tampere, 23rd October 2021

Tuomas Lauttia

# CONTENTS

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| AMCL | Adaptive Monte Carlo Localization |
| ROS | Robot Operating System |

# 1. INTRODUCTION

Historically, autonomous robots have used laser or sonar-based range finders to gain information about their surroundings. Developments in the field of machine learning over the past years have sparked interest in utilizing vision-based information in the tasks of path planning and localization in autonomous robots. However, due to the probabilistic nature of neural networks, machine learning based methods can never be 100% accurate and there is always some error and noise present in the output. We would like to find out, how this affects the algorithms used in robot localization.

The aim of the work in this thesis is to explore the existing algorithms used in robot localization and analyze how they behave in different situations and what causes difficulties for the algorithms. The goal is to gain some understanding of how vision-based measurement models could be used in the task of robot localization. In chapter 2 we will take a look at the theory and mathematics behind robot localization, specifically the Monte Carlo Localization algorithm, which is the algorithm that is used for all of the testing in this work. In chapter 3 we detail the testing environment and testing methods. Chapter 4 reviews the results of the tests. Chapter 5 is the final concluding chapter.

All of the testing in this work was done by using ROS with robot simulator models by Gazebo [1] and maps provided by the HouseExpo dataset [2]. Matlab was used for running the localization algorithm and visualizing the progress.

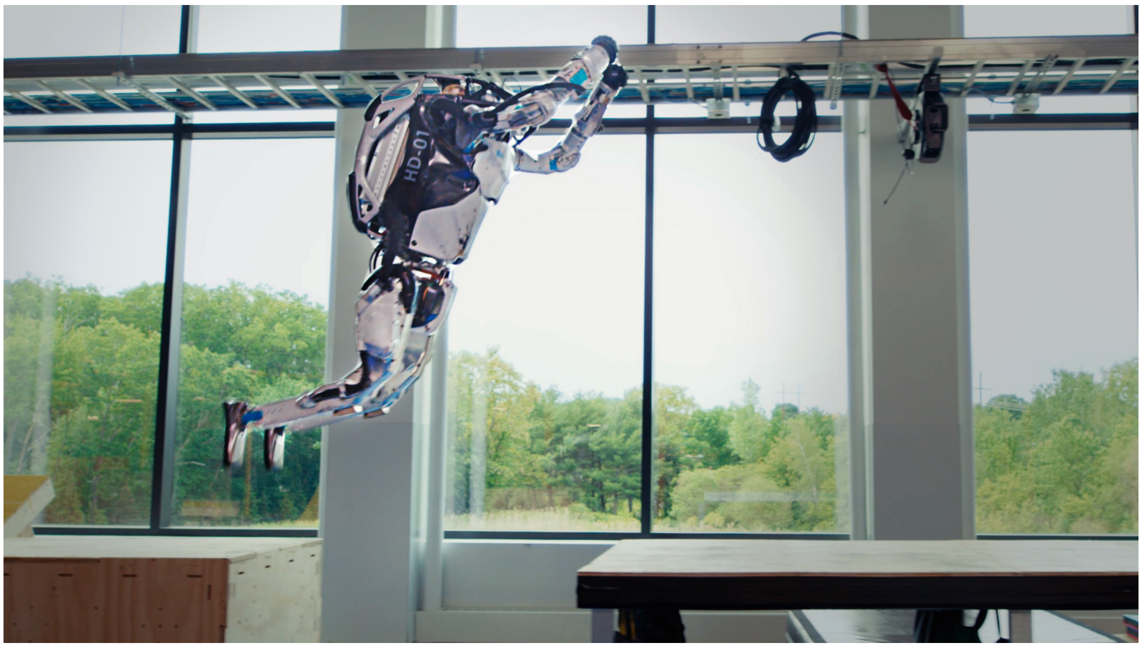

***Figure 1.1.*** *Atlas robot by Boston Dynamics jumping on an obstacle course. The Atlas robot uses depth sensors to generate point clouds of the environment and detect its surroundings, much like the TurtleBot used in the experiments in this thesis. Image taken from Boston Dynamics blog post by Calvin Hennick [3].*

# 2. ROBOT LOCALIZATION

In this chapter, we will briefly go over some of the fundamental concepts in robot localization. The aim is to provide readers with no previous knowledge of robotics a brief introduction to the topic. The information presented in this chapter should be sufficient to understand the fundamentals of the AMCL algorithm and the premise and results of the tests presented in chapters 3 and 4. The concepts and algorithms presented in this chapter are taken directly from the book Probabilistic Robotics, by Thrun, Burgard, and Fox [4]. Probabilistic Robotics is a great source of knowledge on the subject and is highly recommended for anyone seeking to gain further understanding. Some of the algorithms used in ROS, including the AMCL algorithm, are also taken directly from this book.

## 2.1 Uncertainty in robotics

Today's robots are increasingly moving away from highly controlled environments, such as factory lines, into more uncontrolled and unpredictable environments, such as our homes and public roads. This means that the ability to deal with uncertainty is a requirement for the algorithms used in today's robots. In the book Probabilistic Robotics, by Thrun, Burgard, and Fox, uncertainty is said to arise from five different factors: environments, sensors, robots, models, and computation. The ability to cope with uncertainty arising from all of these factors is critical. This has led to the development of probabilistic algorithms in robotics, one of which is the algorithm that is used in this work, Monte Carlo localization. Probabilistic Robotics has detailed descriptions of many of the algorithms used in robotics, including Monte Carlo localization. [5]

In probabilistic robotics, uncertainty is expressed explicitly, using the mathematics of probability theory. Instead of relying on a single "best guess" to determine the state of the robot, probabilistic algorithms represent this information as probability distributions over the space of all possible hypotheses. This enables them to accommodate all sources of uncertainty in a mathematically sound way.

Consider the situation, where a robot is initialized in a random location inside a known environment. It has an internal map of the environment, but no knowledge about its location with respect to the environment. This particular localization problem is known as *global localization*. The probabilistic paradigm represents the robot's momentary *belief* by a probability density function over the space of all locations. The robot's initial *belief* is a uniform distribution over all locations. Now suppose the robot takes a first sensor measurement and observes that it is next to a door. Probabilistic techniques exploit this information to update the belief. The 'posterior' belief places an increased probability at places near doors, and lower probability anywhere else. Assuming the environment has multiple doors, the robot still does not *know* where it is. However, it has hypotheses that are plausible given the sensor data. It is also possible that the robot has erroneously sensed a door. Therefore a small, non-zero probability is placed in areas that have no doors. The ability to maintain low-probability hypotheses is essential for attaining robustness. Now suppose the robot moves. As the robot performs control actions via its actuators to move, it keeps a record of the actions it has performed and shifts the belief in the direction of motion accordingly. After going through enough iterations of taking measurements and performing control actions and updating the belief accordingly, most of the probability mass should converge close to the actual position of the robot. Figure 2.1 illustrates this process.

## 2.2 Recursive State Estimation

At the core of probabilistic robotics is the idea of estimating state from sensor data. State estimation addresses the problem of estimating quantities from sensor data that are not directly observable, but that can be inferred. Sensors carry only partial information about those quantities, and their measurements are corrupted by noise. State estimation seeks to recover state variables from the data. Probabilistic state estimation algorithms compute belief distributions over possible world states.

### 2.2.1 State

Environments are characterized by *state*. It is convenient to think of the state as the collection of all aspects of the robot and its environment that can impact the future. Certain state variables tend to change over time, such as the whereabouts of people in the vicinity of a robot. Others tend to remain static, such as the location of walls in (most) buildings. The state also includes variables regarding the robot itself, such as its pose, velocity, whether or not its sensors are functioning correctly, and so on.

The main state variable considered in this work is the robot *pose*. Pose is comprised of the robot's location and orientation relative to a global coordinate frame. Rigid mobile robots possess six such state variables, three for their Cartesian coordinates, and three for their angular orientation (pitch, roll, and yaw). For rigid mobile robots confined to planar environments, the pose is usually given by three variables, its two location coordinates in the plane and its heading direction (yaw).

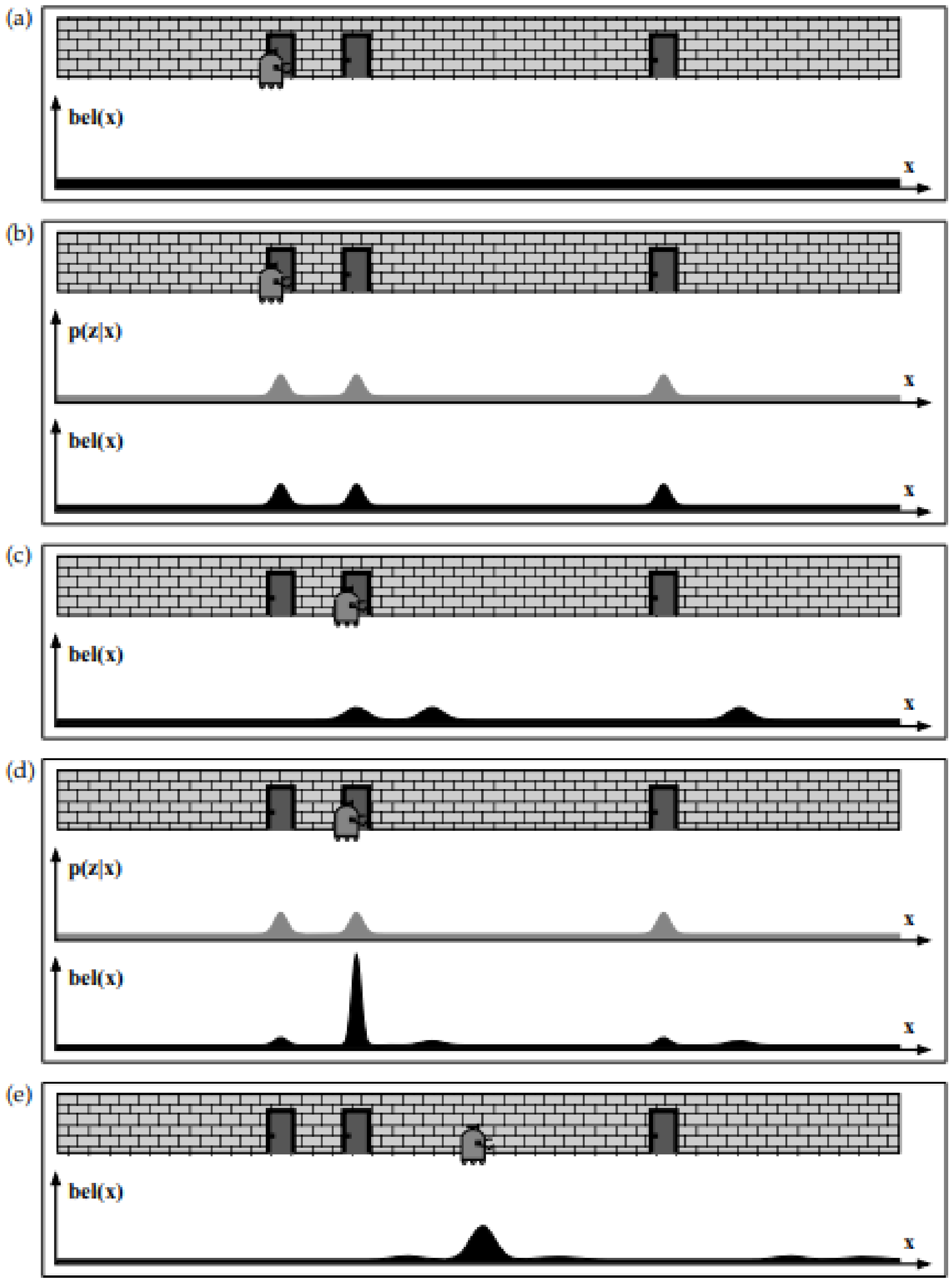

***Figure 2.1.*** *A mobile robot during global localization. Image taken from Probabilistic Robotics by Thrun, Burgard, and Fox [4].*

A state $x_t$ will be called *complete* if it is the best predictor of the future. Completeness entails that knowledge of past states, measurements, or controls carry no additional information that would help us predict the future more accurately.

### 2.2.2 Environment interaction

There are two types of fundamental ways a robot interacts with its environment: The robot can influence the state of the environment via its actuators, and it can gather information about the environment via its sensors. These two types of interactions are often respectively referred to as *control actions* and *measurements*. In accordance with the two types of environment interactions, the robot has access to two different data streams: *control data* and *measurement data*.

Control data carry information about the *change of state* in the environment. In mobile robotics, a typical example of control data is the velocity of a robot. Setting the velocity to 10 cm per second for the duration of five seconds suggests that the robot's pose, after executing this motion command, is approximately 50 cm ahead of its pose before command execution. Thus, control conveys information regarding the change of state. Control data will be denoted $u_t$. The variable $u_t$ will always correspond to the change of state in the time interval $(t-1;t]$.

Measurement data provides information about a momentary state of the environment. Examples of measurement data include camera images, range scans, and so on. The measurement data at time $t$ will be denoted $z_t$.

### 2.2.3 Probabilistic Generative Laws

The evolution of state and measurements is governed by probabilistic laws. In general, the state $x_t$ is generated stochastically from the state $x_{t-1}$. At first glance, the emergence of state $x_t$ might be conditioned on all past states, measurements, and controls. However, if the state $x$ is *complete* then it is a sufficient summary of all that happened in previous time steps. In particular, $x_{t-1}$ is a sufficient statistic of all previous controls and measurements up to this point in time, that is, $u_{1:t-1}$ and $z_{1:t-1}$. Hence, the probabilistic law characterizing the evolution of state is given by a probability distribution of the following form: $p(x_t \,|\, x_{t-1}, u_t)$.

For modeling the process by which measurements are being generated, if $x_t$ is complete, the probability distribution becomes: $p(z_t \,|\, x_t)$. In other words, the state $x_t$ is sufficient to predict the (potentially noisy) measurement $z_t$. Knowledge of any other variable, such as past measurements, controls, or even past states, is irrelevant if $x_t$ is complete.

The probability $p(x_t \,|\, x_{t-1}, u_t)$ is called the *state transition probability*. It specifies how environmental state evolves over time as a function of robot controls $u_t$. Robot environments are stochastic, which is reflected by the fact that $p(x_t \,|\, x_{t-1}, u_t)$ is a probability distribution, not a deterministic function.

The probability $p(z_t \,|\, x_t)$ is called the *measurement probability*. The measurement probability specifies the probabilistic law according to which measurements $z$ are generated from the environment state $x$. It is appropriate to think of measurements as noisy projections of the state.

The state transition probability and the measurement probability together describe the dynamical stochastic system of the robot and its environment.

### 2.2.4 Belief distributions

In probabilistic robotics, the state of the robot is represented by belief distributions. A *belief* reflects the robot's internal knowledge about the state of the environment. Since state cannot be measured directly, and must instead be inferred from data, we make a distinction between the true state and the robot's internal *belief* regarding that state.

Probabilistic robotics represents beliefs through conditional probability distributions. A belief distribution assigns a probability (or density value) to each possible hypothesis with regards to the true state. Belief distributions are posterior probabilities over state variables conditioned on the available data. We will denote belief over a state variable $x_t$ by $bel(x_t)$, which is an abbreviation for the posterior $bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t})$. This posterior is the probability distribution over the state $x_t$ at time $t$, conditioned on all past measurements $z_{1:t}$ and all past controls $u_{1:t}$.

Occasionally, it is useful to calculate a posterior before incorporating $z_t$, just after executing the control $u_t$. Such a posterior is denoted as follows: $\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t})$. This probability distribution is often referred to as prediction in the context of probabilistic filtering. This terminology reflects the fact that $\overline{bel}(x_t)$ predicts the state at time $t$ based on the previous state posterior, before incorporating the measurement at time $t$. Calculating $bel(x_t)$ from $\overline{bel}(x_t)$ is called *correction* or the *measurement update*.

### 2.2.5 The Bayes filter algorithm

The most general algorithm for calculating beliefs is given by the *Bayes filter* algorithm. This algorithm calculates the belief distribution *bel* from measurement and control data. Algorithm 1 depicts the basic Bayes filter in pseudo-algorithmic form.

**Algorithm 1:** The Bayes filter algorithm

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

The Bayes filter is recursive, that is, the belief $bel(x_t)$ at time $t$ is calculated from the belief $bel(x_{t-1})$ at time $t-1$. Its input is the belief $bel$ at time $t-1$, along with the most recent control $u_t$ and the most recent measurement $z_t$. Its output is the belief $bel(x_t)$ at time $t$.

The Bayes filter algorithm possesses two essential steps. In line 3, it processes the control $u_t$. It does so by calculating a belief over the state $x_t$ based on the prior belief over state $x_{t-1}$ and the control $u_t$. In particular, the belief $\overline{bel}(x_t)$ that the robot assigns to state $x_t$ is obtained by the integral (sum) of the product of two distributions: the prior assigned to $x_{t-1}$, and the probability that control $u_t$ induces a transition from $x_{t-1}$ to $x_t$.

The second step of the Bayes filter is called the *measurement update*. In line 4, the Bayes filter algorithm multiplies the belief $\overline{bel}(x_t)$ by the probability that the measurement $z_t$ may have been observed. It does so for each hypothetical posterior state $x_t$. As will become apparent further below when actually deriving the basic filter equations, the resulting product is generally not a probability. It may not integrate to 1. Hence, the result is normalized, by virtue of the normalization constant $\eta$. This leads to the final belief $bel(x_t)$, which is returned in line 6 of the algorithm.

To compute the posterior belief recursively, the algorithm requires an initial belief $bel(x_0)$ at time $t = 0$ as boundary condition. If one knows the value of $x_0$ with certainty, $bel(x_0)$ should be initialized with a point mass distribution that centers all probability mass on the correct value of x0, and assigns zero probability anywhere else. If one is entirely ignorant about the initial value $x_0$, $bel(x_0)$ may be initialized using a uniform distribution over the domain of $x_0$ (or a related distribution from the Dirichlet family of distributions). Partial knowledge of the initial value $x_0$ can be expressed by non-uniform distributions; however, the two cases of full knowledge and full ignorance are the most common ones in practice.

The algorithm Bayes filter can only be implemented in the form stated here for very simple estimation problems. In particular, we either need to be able to carry out the integration in line 3 and the multiplication in line 4 in closed form, or we need to restrict ourselves to finite state spaces, so that the integral in line 3 becomes a (finite) sum.

## 2.3 The Particle Filter

The *particle filter* is a nonparametric implementation of the Bayes filter. Particle filters approximate the posterior by a finite number of parameters. The key idea of the particle filter is to represent the posterior $bel(x_t)$ by a set of random state samples drawn from this posterior. Such a representation is approximate, but it is nonparametric, and therefore can represent a much broader space of distributions than, for example, Gaussians.

In particle filters, the samples of a posterior distribution are called particles and are denoted

$X_t := x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}$

Each particle $x_t^{[m]}$ (with $1 \leq m \leq M$) is a concrete instantiation of the state at time $t$. Put differently, a particle is a hypothesis as to what the true world state may be at time $t$. Here $M$ denotes the number of particles in the particle set $X_t$. In practice, the number of particles $M$ is often a large number, e.g., $M = 1,000$.

The intuition behind particle filters is to approximate the belief $bel(x_t)$ by the set of particles $X_t$. Ideally, the likelihood for a state hypothesis $x_t$ to be included in the particle set $X_t$ shall be proportional to its Bayes filter posterior $bel(x_t)$:

$x_t^{[m]} \sim p(x_t \mid z_{1:t}, u_{1:t})$

Just like the basic Bayes filter algorithm, the particle filter algorithm constructs the belief $bel(x_t)$ recursively from the belief $bel(x_{t-1})$ one time step earlier. Since beliefs are represented by sets of particles, this means that particle filters construct the particle set $X_t$ recursively from the set $X_{t-1}$.

The most basic variant of the particle filter algorithm is shown in Algorithm 2. The input of this algorithm is the particle set $X_{t-1}$, along with the most recent control $u_t$ and the most recent measurement $z_t$. The algorithm then first constructs a temporary particle set $\bar{X}$ that represented the belief $\overline{bel}(x_t)$. It does this by systematically processing each particle $x_{t-1}^{[m]}$ in the input particle set $X_{t-1}$. Subsequently, it transforms these particles into the set $X_t$, which approximates the posterior distribution $bel(x_t)$. In detail:

**Algorithm 2:** Basic particle filter algorithm

```
1  Algorithm Particle_filter($X_{t-1}, u_t, z_t$):
2  $\bar{X}_t = X_t = \emptyset$
3  for m = 1 to M do
4      sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5      $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6      $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
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 $X_t$
```

1. Line 4 generates a hypothetical state $x_t^{[m]}$ for time $t$ based on the particle $x_{t-1}^{[m]}$ and the control $u_t$. The resulting sample is indexed by $m$, indicating that it is generated from the $m$-th particle in $X_{t-1}$. This step involves sampling from the state transition distribution $p(x_t \mid u_t, x_{t-1})$. To implement this step, one needs to be able to sample from this distribution. The set of particles obtained after $M$ iterations is the filter's

representation of $\overline{bel}(x_t)$.

2. Line 5 calculates for each particle $x_t^{[m]}$ the so-called *importance factor*, denoted $w_t^{[m]}$. Importance factors are used to incorporate the measurement $z_t$ into the particle set. The importance, thus, is the probability of the measurement $z_t$ under the particle $x_t^{[m]}$ , given by $w_t^{[m]} = p(z_t \mid x_t^{[m]})$. If we interpret $w_t^{[m]}$ as the weight of a particle, the set of weighted particles represents (in approximation) the Bayes filter posterior $bel(x_t)$.

3. The real “trick” of the particle filter algorithm occurs in lines 8 through 11. These lines implemented what is known as *resampling* or *importance sampling*. The algorithm draws with replacement $M$ particles from the temporary set $\bar{X}_t$. The probability of drawing each particle is given by its importance weight. Resampling transforms a particle set of $M$ particles into another particle set of the same size. By incorporating the importance weights into the resampling process, the distribution of the particles change: Whereas before the resampling step, they were distributed according to $\overline{bel}(x_t)$, after the resampling they are distributed (approximately) according to the posterior $bel(x_t) = \eta\, p(z_t \mid x_t^{[m]})\overline{bel}(x_t)$. In fact, the resulting sample set usually possesses many duplicates, since particles are drawn with replacement. More important are the particles no longer contained in $X_t$: Those tend to be the particles with lower importance weights.

## 2.4 Monte Carlo Localization

One of the most popular localization algorithms that utilise the particle filter is *Monte Carlo Localization*, or *MCL*. MCL is applicable to both local and global localization problems. It is easy to implement and tends to work well for a broad range of localization problems. The amcl package in ROS [4] implements the adaptive (or KLD-sampling) Monte Carlo localization approach.

### 2.4.1 The MCL Algorithm

Algorithm 3 shows the basic MCL algorithm, which is obtained by substituting the appropriate probabilistic motion and perceptual models into the basic particle filter algorithm (Algorithm 2). The basic MCL algorithm represents the belief $bel(x_t)$ by a set of $M$ particles $X_t = \{x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]}\}$. Line 4 in the algorithm samples from the motion model, using particles from present belief as starting points. The measurement model is then applied in line 5 to determine the importance weight of that particle. The initial belief $bel(x_0)$ is obtained by randomly generating $M$ such particles from the prior distribution $p(x_0)$, and assigning the uniform importance factor $M^{-1}$ to each particle. The functions **sample_motion_model** and **measurement_model** implement a motion model and a

**Algorithm 3:** basic MCL algorithm

```
Algorithm MCL($X_{t-1}, u_t, z_t, m$):
$\bar{X}_t = X_t = \emptyset$
for m = 1 to M do
    $x_t^{[m]} =$ sample_motion_model$(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} =$ measurement_model$(z_t, x_t^{[m]}, m)$
    $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
end for
for m = 1 to M do
    draw $i$ with probability $\propto w_t^{[i]}$
    add $x_t^{[i]}$ to $X_t$
end for
return $X_t$
```

measurement model respectively. The parameter $m$ in **measurement_model** is the map of the environment. In theory, the MCL algorithm supports any kind of sensor, as long as the appropriate models are implemented. At the time of writing the amcl package in ROS [4] only supports laser scans and laser maps.

### 2.4.2 Properties of MCL

MCL can approximate almost any distribution of practical importance. Increasing the total number of particles increases the accuracy of the approximation. The number of particles $M$ is a parameter that enables the user to trade-off the accuracy of the computation and the computational resources necessary to run MCL. A common strategy for setting $M$ is to keep sampling until the next pair $u_t$ and $z_t$ has arrived. In this way, the implementation is adaptive with regards to the computational resources: the faster the underlying processor, the better the localization algorithm. However, care has to be taken that the number of particles remains high enough to avoid filter divergence.

### 2.4.3 KLD-Sampling

The size of the sample sets used to represent beliefs is an important parameter for the efficiency of particle filters. Unfortunately, to avoid divergence due to sample-depletion in MCL, one has to choose large sample sets so as to allow a mobile robot to address both the global localization and the position tracking problem. This can be a waste of computational resources. While a high number of particles (e.g. 100,000) might be necessary during the early stages of localization, only a small fraction of this number is sufficient to track the position of the robot once it knows where it is.

*KLD-sampling* is a variant of MCL that adapts the number of particles over time. The name *KLD-sampling* is derived from the *Kullback-Leibler divergence*, which is a measure of the difference between two probability distributions. The idea behind KLD-sampling is to determine the number of particles based on a statistical bound on the sample-based approximation quality.

# 3. EXPERIMENTS

Testing was done by running the AMCL algorithm in a simulated environment using 100 different randomly selected binary occupancy grids obtained from the HouseExpo dataset [2]. Gazebo [1] was used for simulating the environment and the robot. Matlab was used to send commands to the robot via ROS, running the AMCL algorithm and plotting the results. The 3D environments used in the simulation were generated from 2D binary occupancy maps by using map2gazebo [6].

## 3.1 Test environment

The test environment was a virtual machine running Ubuntu Bionic 18.04 with ROS Melodic installed. The virtual machine was obtained from the MathWorks website [7]. The virtual machine contains a working installation of ROS Melodic and ROS 2 Dashing with a variety of sample scripts and launch files usable with ROS. The Gazebo simulator is also preinstalled on the virtual machine. Matlab with ROS Toolbox was used on the host machine to communicate with the robot over LAN and run the test scripts.

## 3.2 Data preparation

HouseExpo is a large-scale dataset of indoor layouts. The dataset contains 35,126 2D floor plans with 252,550 rooms total [2]. 100 of these maps were randomly selected and used for testing the AMCL algorithm. Figure 3.1 shows an example .png image from the dataset. Entirely black walls were removed, as initial testing showed that these caused problems for the AMCL algorithm. This is because for particles enclosed completely inside a wall, a large range of measurements are equally likely and therefore the fully enclosed particles gain a high likelihood and the algorithm can start converging inside the walls of the map. Figure 3.2 shows the same map but with the black area removed.

The 2D images were converted into 3D meshes usable in Gazebo by using map2gazebo. Map2gazebo generates 2 meter high walls in the occupied areas of the 2D binary occupancy grid [6]. Figure 3.3 shows the map inside Gazebo.

The maps in the HouseExpo dataset are defined by a set of 2D vertices, based on which the binary occupancy maps are generated. The maps have varying numbers of vertices

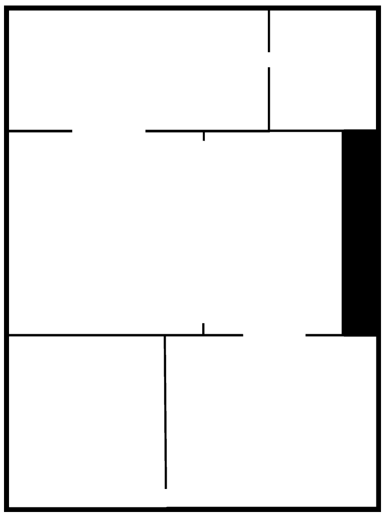

***Figure 3.1.*** *Example map from the HouseExpo dataset*

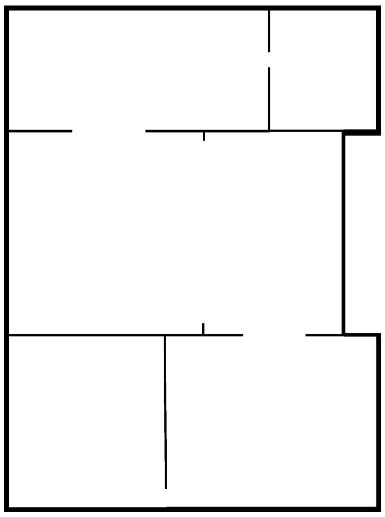

***Figure 3.2.*** *The same map as Figure 3.1, but with the black area removed*

and a varying number of rooms. Figures 3.4 and 3.5 show histograms for the number of vertices and the number of rooms per map, respectively.

## 3.3 Test setup

A sample MATLAB® script was used and adapted for usage with different maps. The sample script uses AMCL to localize a simulated TurtleBot robot inside an office environment. The current MATLAB® AMCL implementation can be applied to any differential drive robot equipped with a range finder [8].

The Gazebo TurtleBot uses a simulated range finder, with noise variance set to 0.01 and maximum range increased from the default value of 3 meters to 35 meters for testing. Most of the AMCL parameters used for testing were the same as the sample script, except for SensorLimits and MaxLikelihoodDistance, which were increased to match the

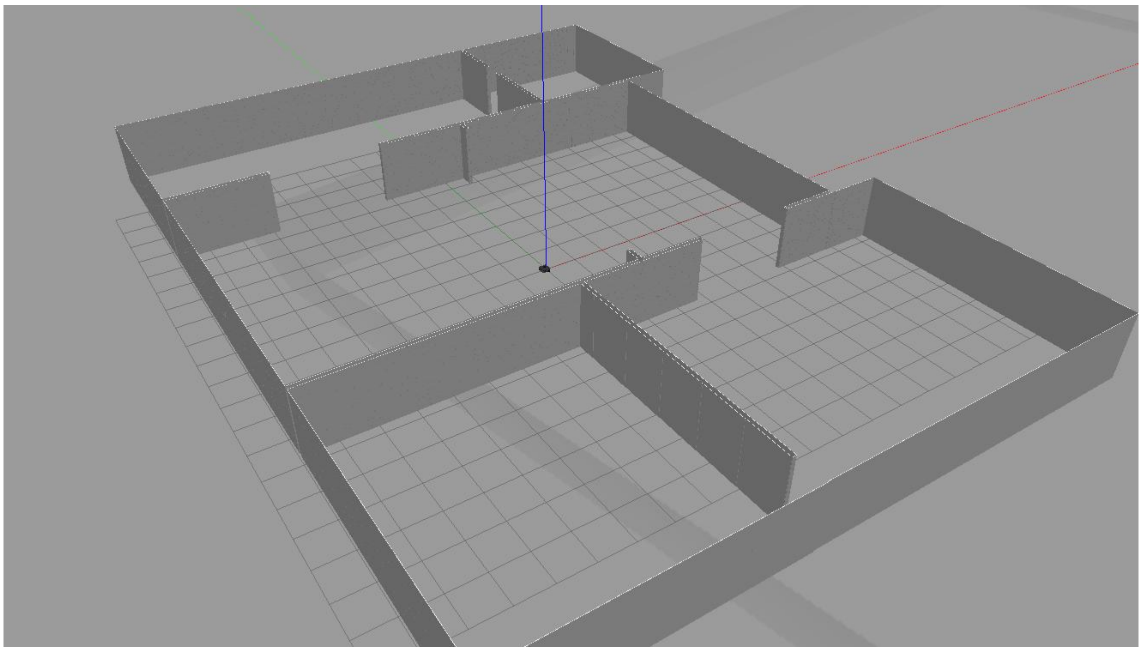

***Figure 3.3.*** *Map from figure 3.1 inside Gazebo*

values of the simulated range finder. SensorLimits determines the range of measurements that are used for the likelihood calculation. MaxLikelihoodDistance limits the area for searching for obstacles in the binary occupancy grid.

The test script takes as parameter a binary occupancy grid and drives the robot around randomly while avoiding obstacles. The controllerVFH class is used for driving the robot while avoiding obstacles. This class uses vector field histograms to compute an obstacle-free steering direction given laser scan readings and a target direction to drive toward [9].

The particle filter is only updated when the displacement of the robot exceeds a set threshold, defined by amcl.UpdateThresholds. The particle distribution is resampled after every update and the AMCL algorithm is run for a maximum of 100 updates. The algorithm is considered converged when 90% of all of the particles are within 0.3 meters of the true pose of the robot. This threshold was chosen arbitrarily based on values observed when running the script. The test is run 10 times for each map. The tests are evaluated based on how many times the particle filter converged to the correct position and the number of steps it took for the algorithm to converge.

The particles were initialized using a uniform distribution over the binary occupancy grid, with the lower and upper bounds for the particle count set to 500 and 50,000 respectively.

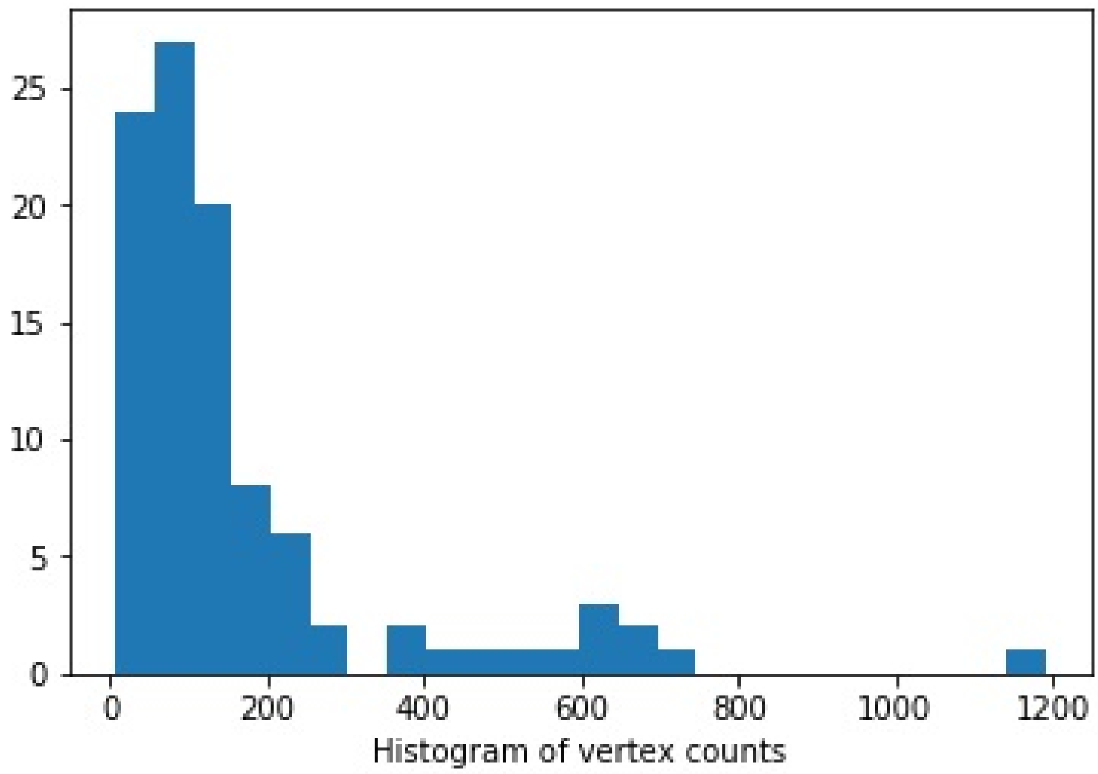

***Figure 3.4.*** *Vertex count histogram. Vertex count means the number of vertices used to define the map in the HouseExpo dataset. The HouseExpo dataset defines a map as a set of 2D vertices [2].*

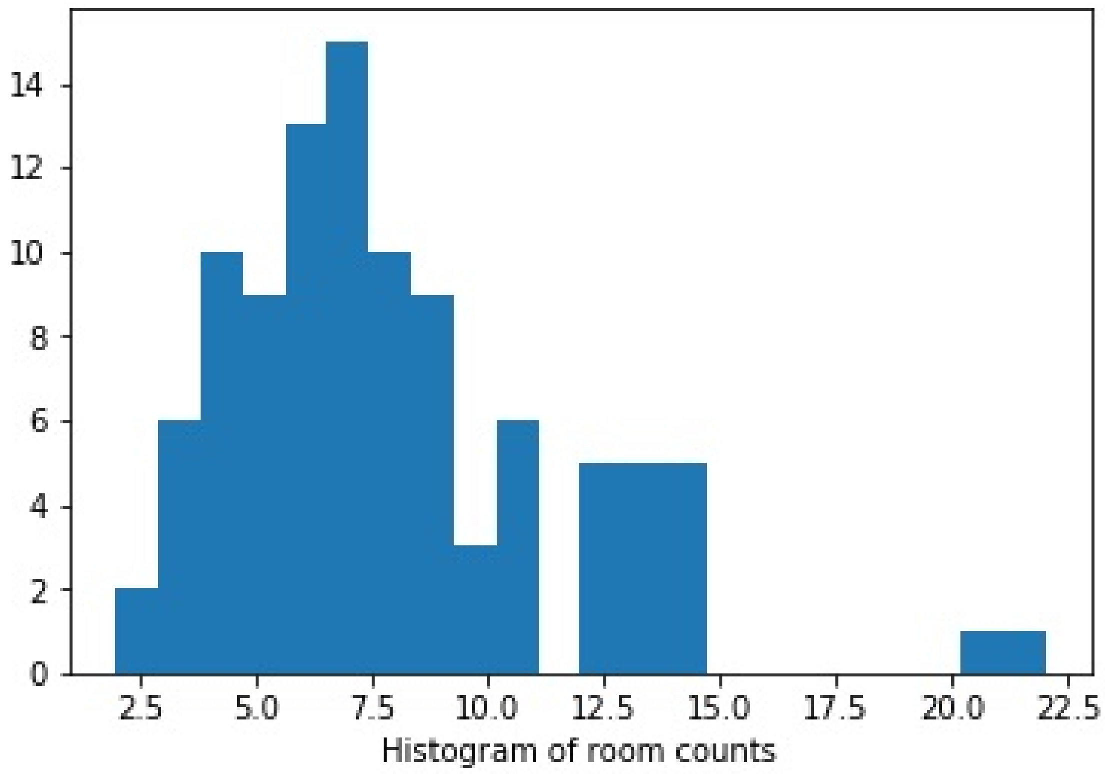

***Figure 3.5.*** *Room count histogram*

# 4. RESULTS

The metrics recorded from the test runs were the number of runs where the algorithm converged, mean and variance for the number of steps it took until convergence if the algorithm converged to the correct position, and the mean estimated distance from the ground truth. The minimum and maximum step counts until convergence were also recorded. In this chapter, the word "converge" is used to indicate a successful localization. Note that the AMCL algorithm always converges, but not necessarily to the correct position.

Out of the total of 1000 (10 per each of the 100 maps) test runs, the AMCL algorithm converged to the correct location 878 times. The median number of converged runs per map was 10. Figure 4.1 shows a collage of all of the 100 maps and the amount of converged test runs per map. Note that the maps are not to scale. Figure 4.2 shows a histogram of the number of converged runs per map. Overall the algorithm seems to perform quite well, with 53 of the tested maps converging to the correct position 10 out of 10 times. Figure 4.3 shows the mean number of converged runs plotted against the number of rooms. Figure 4.4 shows the mean number of converged runs plotted against the number of vertices. Based on these plots it is fairly safe to say that there is no correlation between the number of converged test runs and the room count or the number of vertices of the map.

Figure 4.5 shows the mean number of steps until convergence against the number of rooms for the converged runs. Figure 4.6 shows the mean number of steps until convergence against the number of vertices. These plots show a slight upward trend as the number of rooms and number of vertices increase.

The three maps with the worst performance are plotted in Figure 4.7. The worst performing map was *bb2cee7cd62bcbbc10e4b79d7c632267*. For this map, the AMCL algorithm converged to the correct location only 2 times out of 10. It is quite easy to see why this would be the case, as all 4 rooms on the map look nearly identical. Figure 4.8 shows the particle filter during one of the test runs after the 10th particle filter update. Figure 4.9 shows the final particle filter update and the true position of the robot after 100 updates. Even though none of the rooms are entirely symmetric, they are similar enough so that due to the probabilistic nature of the algorithm, the algorithm can start converging towards the wrong position. Figure 4.10 shows an "easy" map where the AMCL algorithm con-

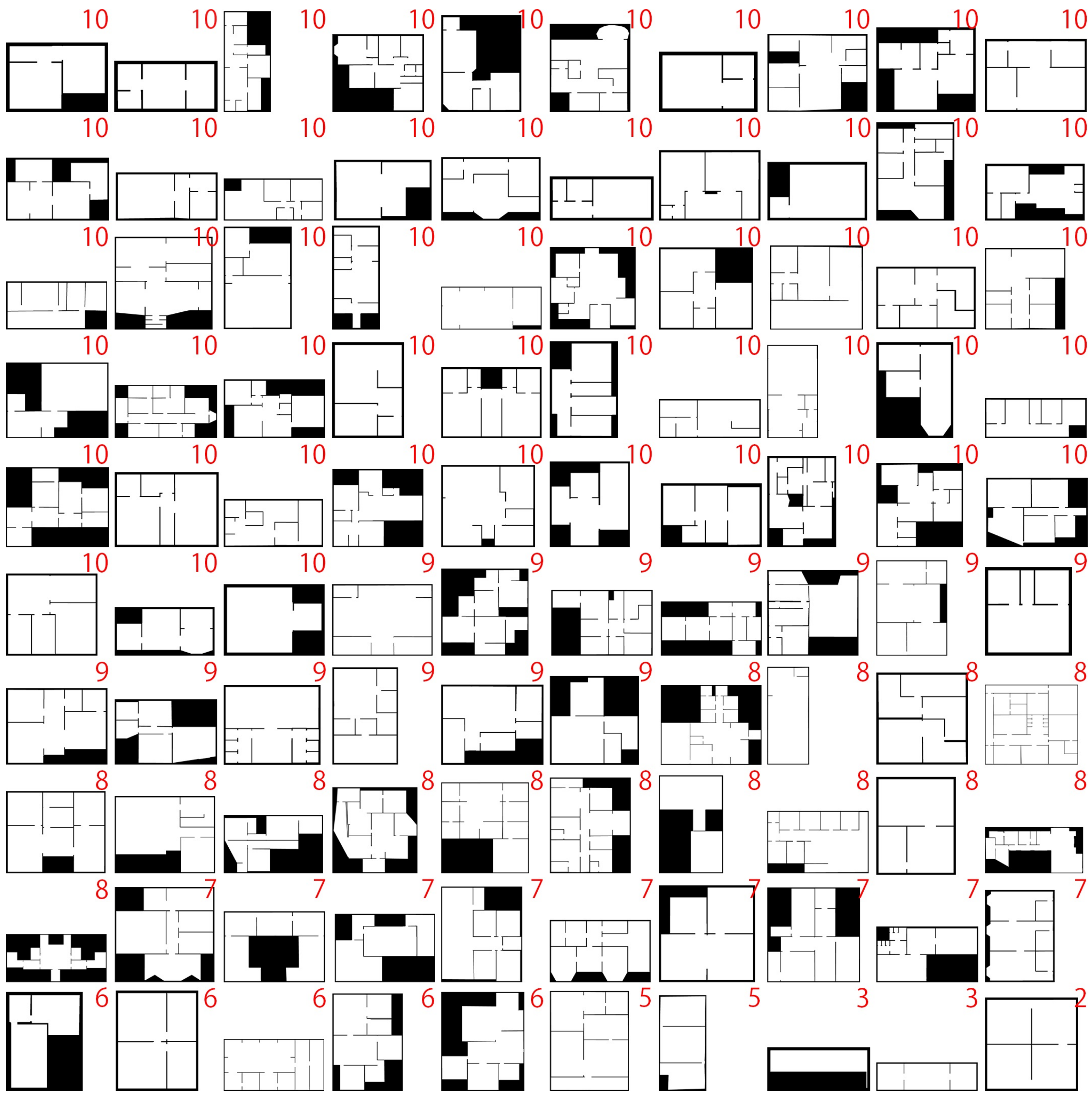

***Figure 4.1.** All of the tested maps with the number of test runs where the robot was localized successfully for each map.*

verged to the correct position 10 times out of 10 and a "medium" map where the algorithm converged to the correct position 6 times out of 10.

Figure 4.11 shows one of the maps in Gazebo and the robot's view of the world visualized in rviz. Rviz is a 3D visualization tool for ROS [10]. Note that some of the sensor readings are on the robot itself. Sensor readings that are shorter than the robot's diameter are ignored in the AMCL probability calculations.

## 4.1 Detailed analysis

Figures 4.12 and 4.13 show two example test runs for one of the maps for which the AMCL algorithm converged to the correct position only 5 times out of 10. The actual trajectory of the robot's movement is plotted in magenta. Figure 4.12 shows a test run

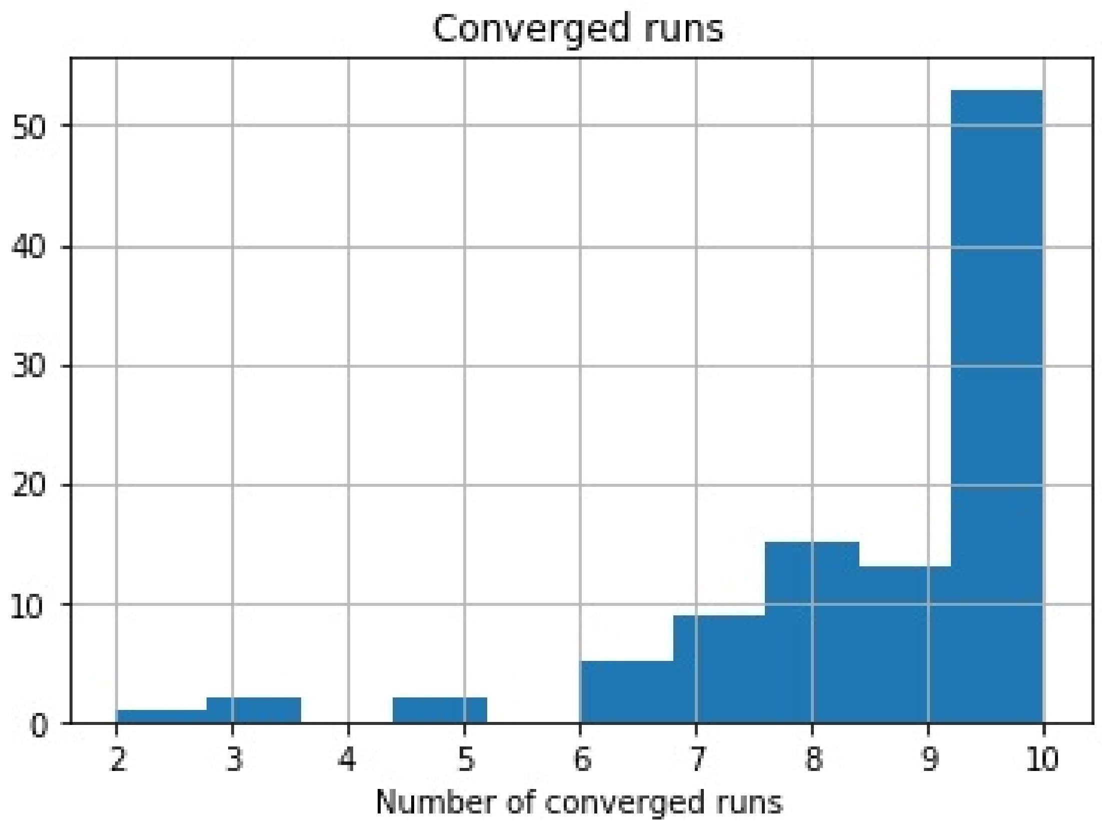

***Figure 4.2.*** *Histogram of converged test runs per map*

where the algorithm converged to the wrong position and failed to recover. Figure 4.13 shows a successfully converged test run on the same map.

Finally, the map was tested by changing the number of particles and running the tests with an upper particle limit of 50,000 and 100,000, for 300 test runs each. With the upper particle limit set to 50,000, the AMCL algorithm converged to the correct position 197 times out of 300. With an upper limit of 100,000 particles, the algorithm converged to the correct position 291 times out of 300. Based on these results, at least for this particular map, the number of particles seems to have a noticeable effect on the performance of the AMCL algorithm. The results are plotted in figure 4.14.

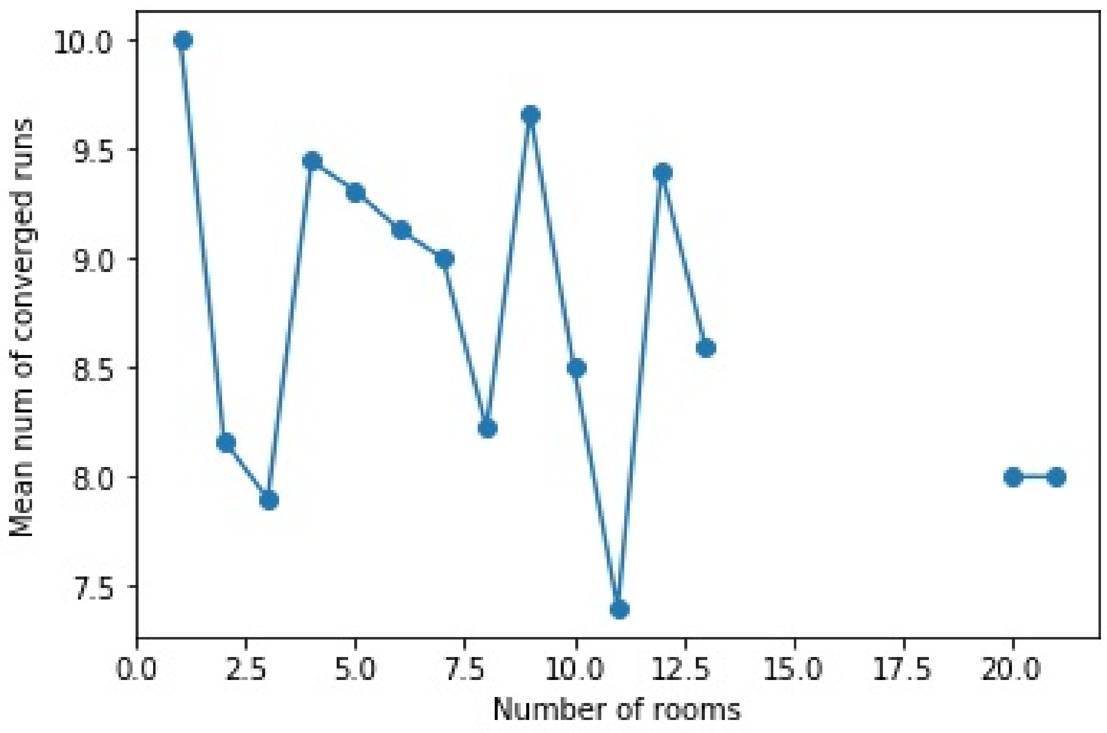

***Figure 4.3.*** *Mean number of converged runs per number of rooms*

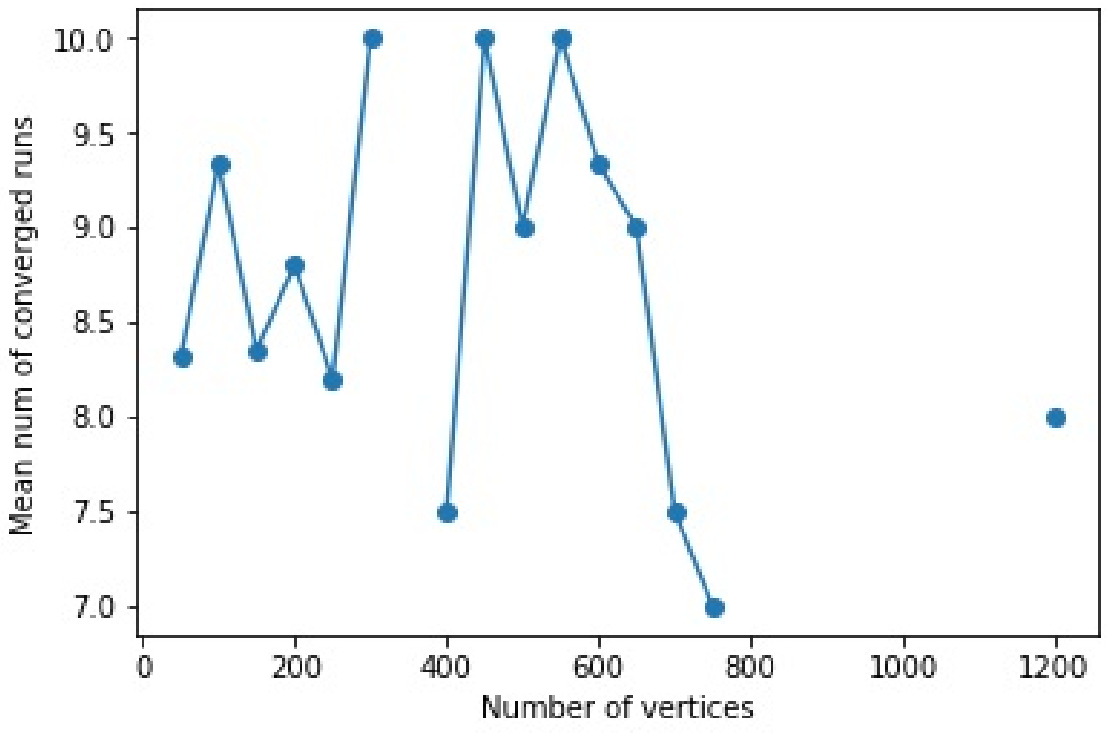

***Figure 4.4.*** *Mean number of converged runs per number of vertices*

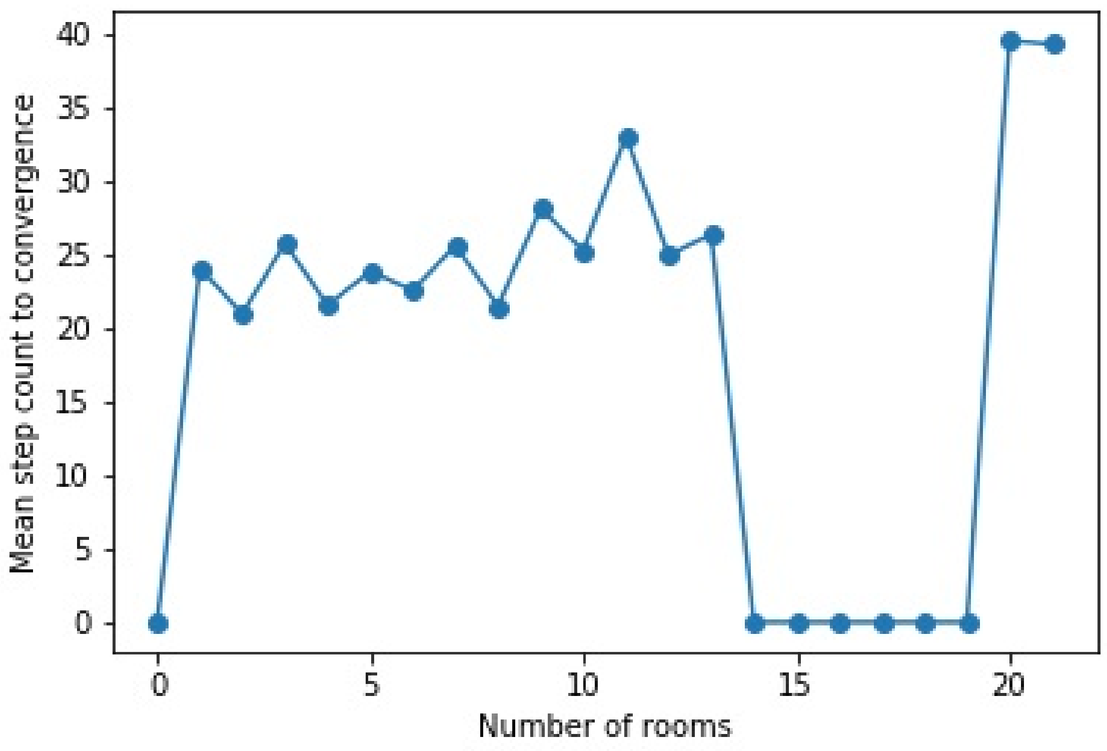

***Figure 4.5.*** *Mean step count to convergence per number of rooms*

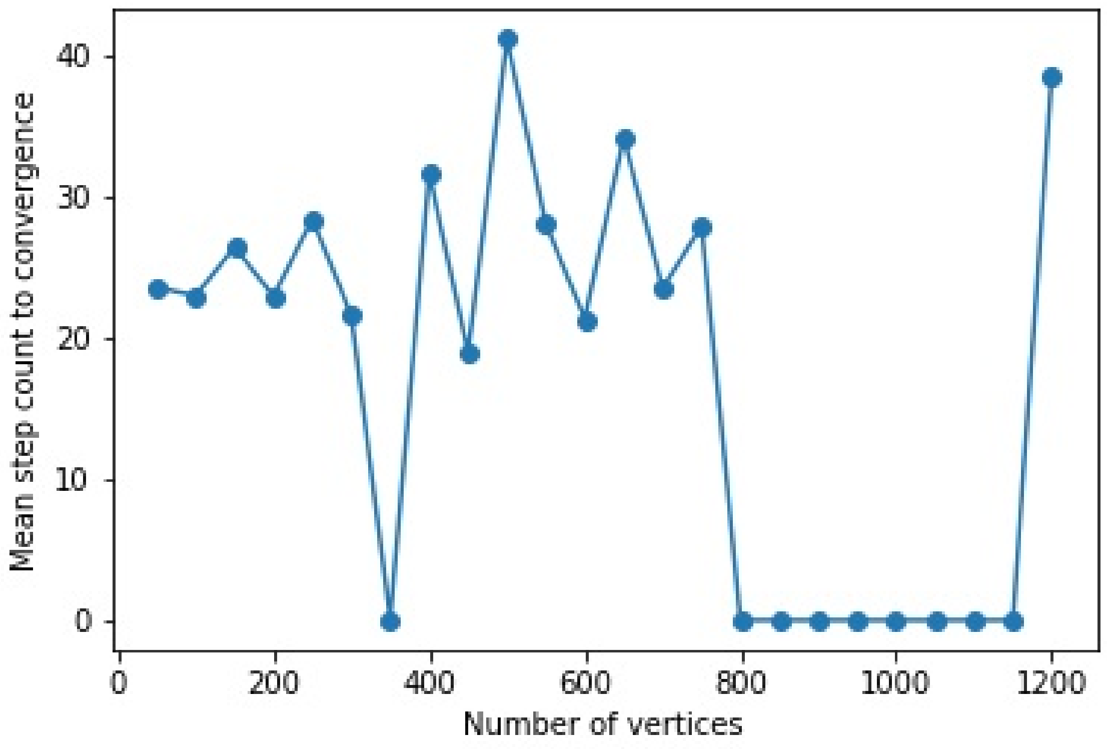

***Figure 4.6.*** *Mean step count to convergence per number of vertices*

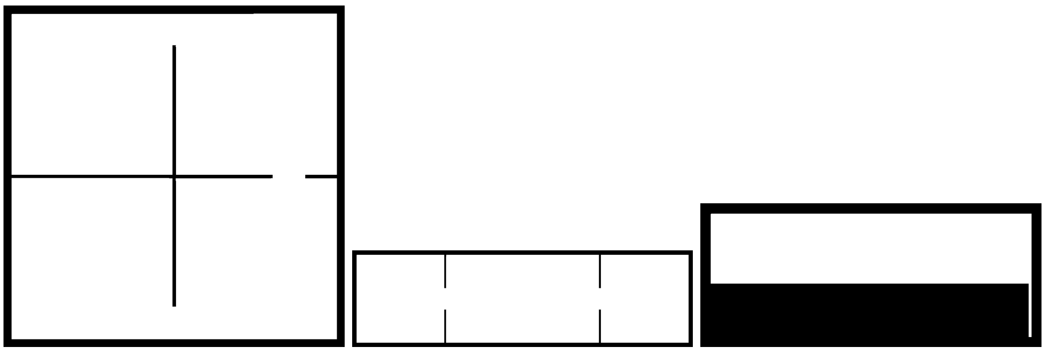

***Figure 4.7.*** *The three maps with the worst performance. For the leftmost map, the AMCL algorithm converged to the correct position only 2 times out of 10. For the other two, the algorithm converged to the correct position 3 times out of 10.*

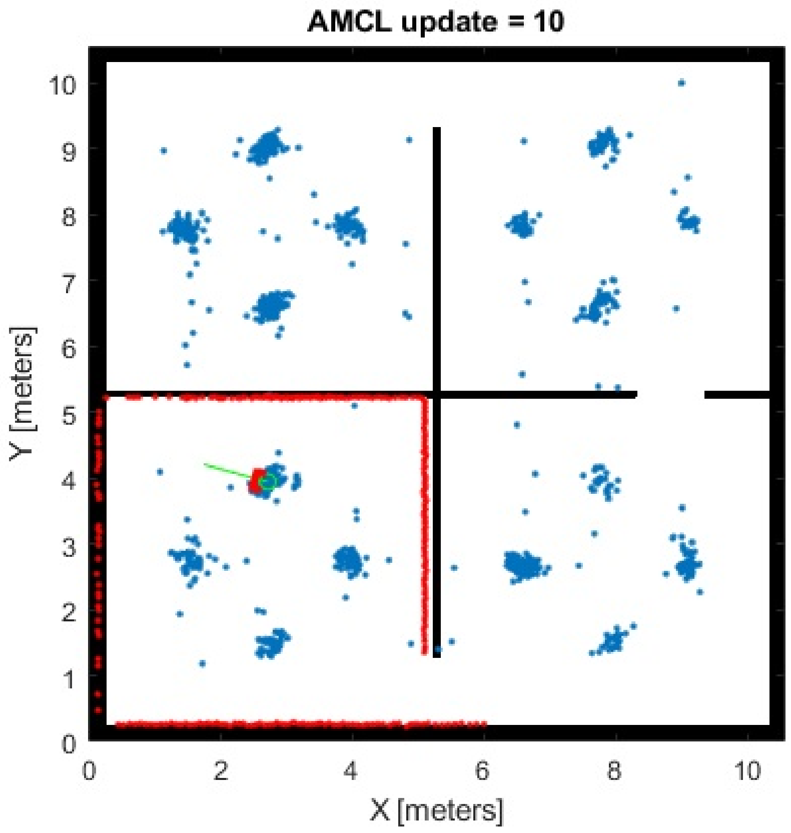

***Figure 4.8.*** *Particle filter after 10th update*

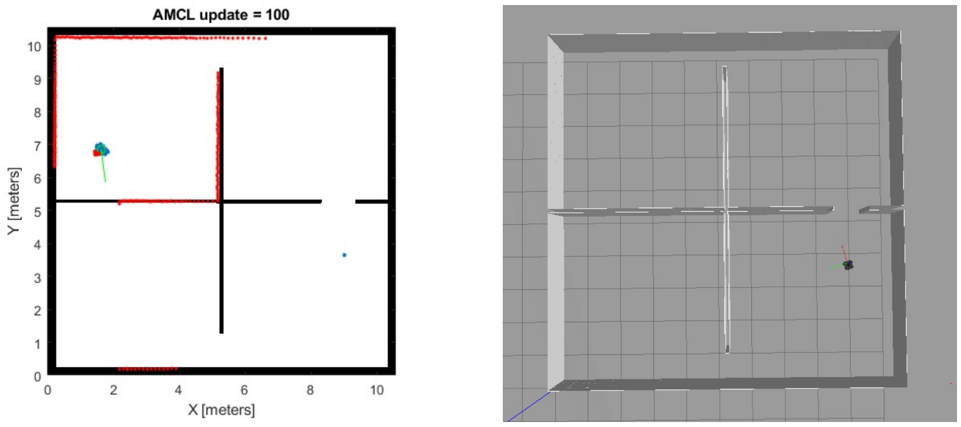

***Figure 4.9.*** *The particle filter and the true position of the robot after a test run where the AMCL algorithm converged to the wrong position.*

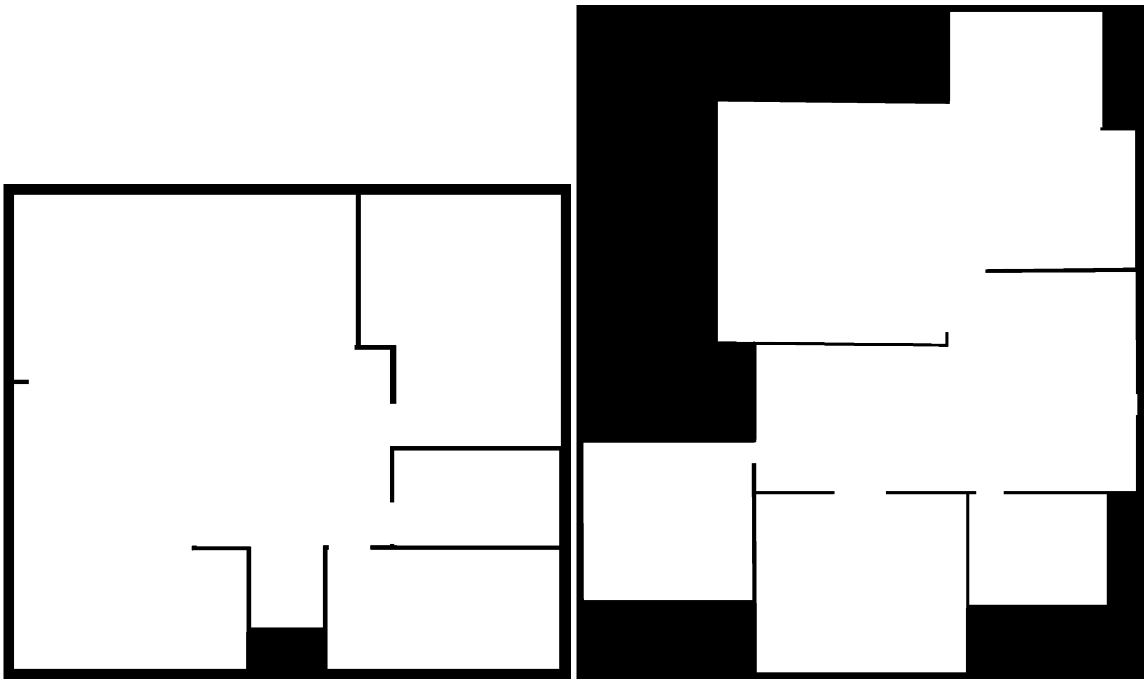

***Figure 4.10.*** *For the map on the left, the AMCL algorithm converged to the correct position 10 times out of 10. For the map on the right, the algorithm converged to the correct position 6 times out of 10.*

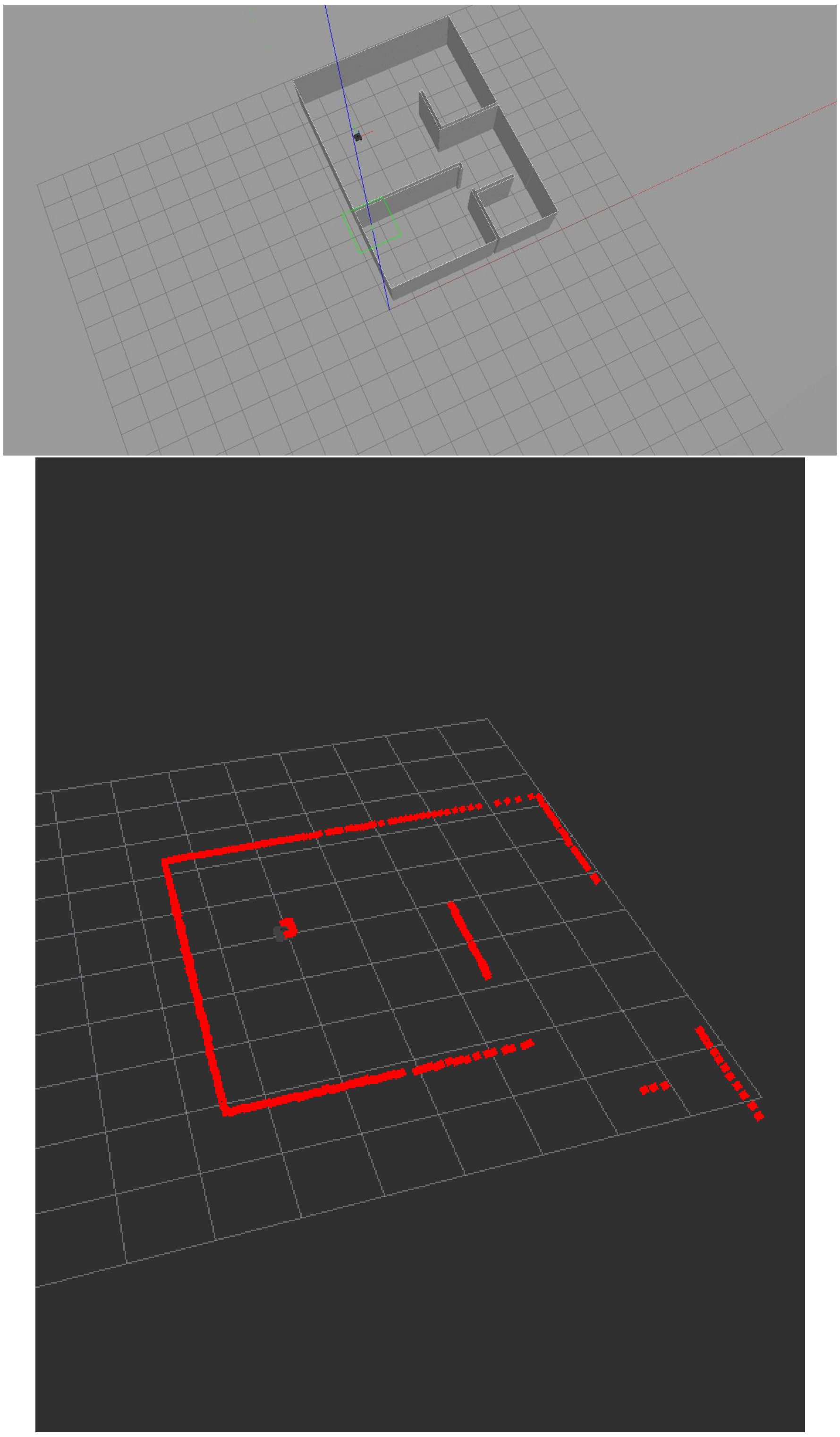

***Figure 4.11.*** *A map in Gazebo and the robot's view of the world visualized with rviz. The red dots are laser scan measurements.*

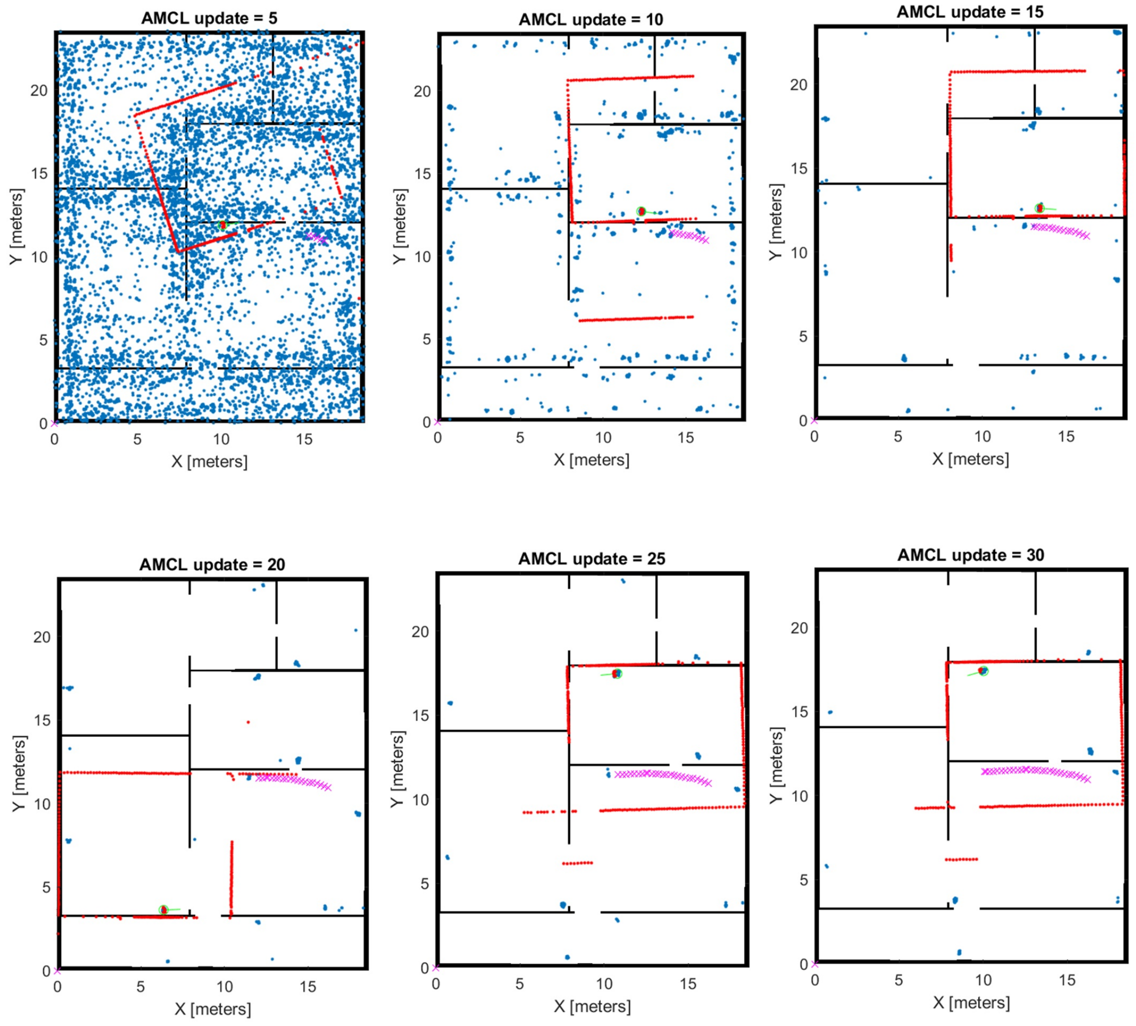

***Figure 4.12.*** *A test run where the AMCL algorithm failed to converge to the correct position and failed to recover. The estimated pose of the robot is plotted in green. The actual trajectory of the robot is plotted in magenta.*

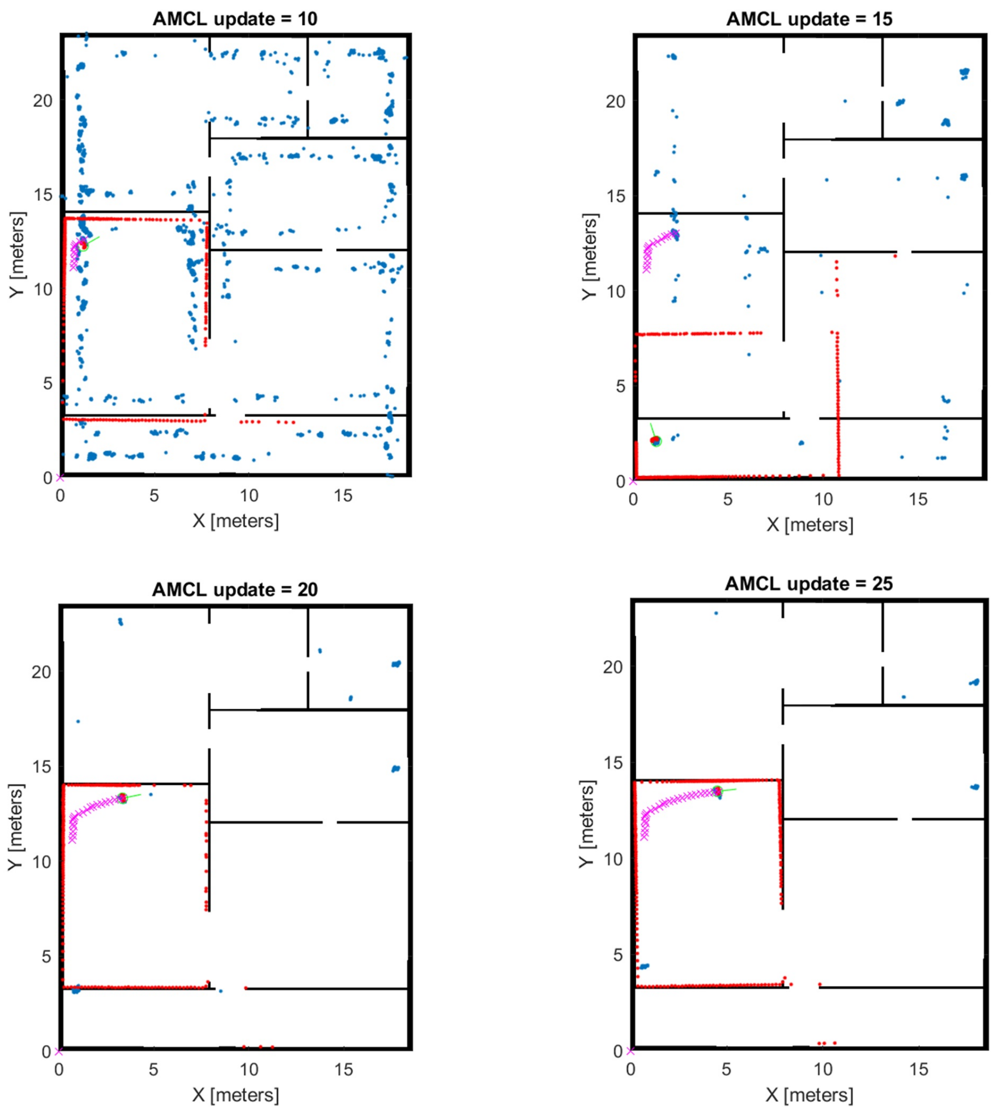

***Figure 4.13.*** *A test run where the AMCL algorithm successfully converged to the correct position. The estimated pose of the robot is plotted in green. The actual trajectory of the robot is plotted in magenta.*

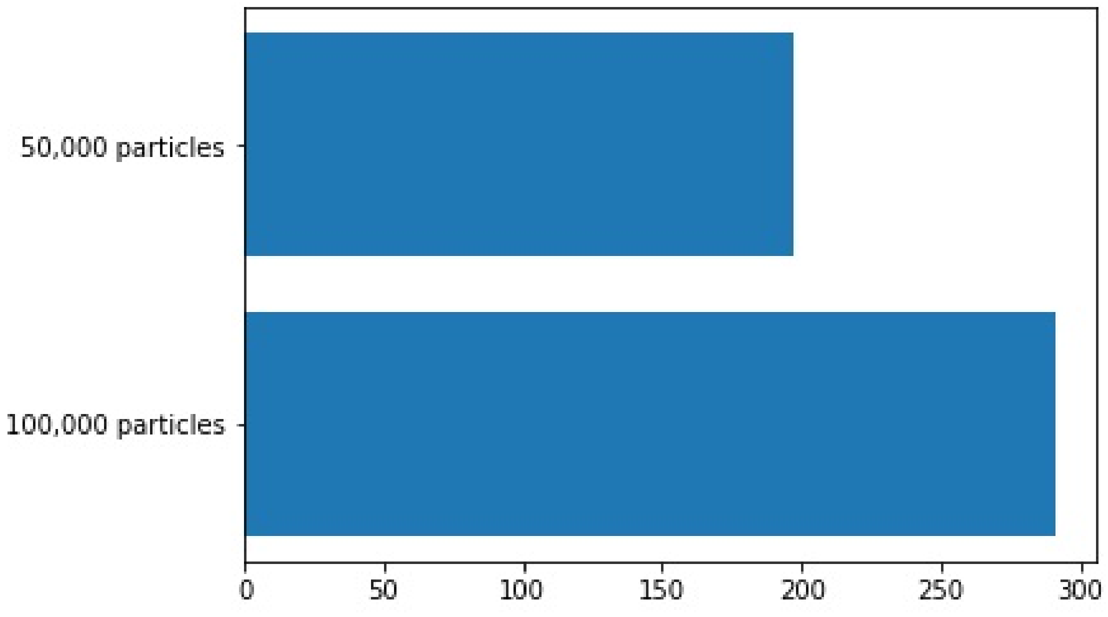

***Figure 4.14.*** *The number of converged test runs with the upper particle limit set to 50,000 and 100,000 respectively. The tests were run a total of 300 times for each set of parameters, using the same map for all of the tests. Increasing the number of particles seems to have a noticeable positive impact on the accuracy of the AMCL algorithm.*

# 5. CONCLUSION

In this work, we have examined the inner workings of the AMCL algorithm as it is used for robot localization and tested the performance of the algorithm on random maps picked from the HouseExpo dataset using MATLAB® and ROS. The tests show that overall the algorithm performs quite well for the maps in this dataset.

The number of rooms or the number of vertices in the map did not seem to affect the performance of the algorithm significantly. The worst performance happens in maps that have a high degree of symmetry or similarity between multiple areas of the map. In these cases, the AMCL algorithm can begin to converge towards the wrong position and be unable to recover with further sensor updates.

Overall the AMCL algorithm is very robust to noise and can recover from bad initial estimates in many cases. This suggests that for example computer vision-based measurements could integrate quite well with the algorithm.

# REFERENCES

[1] Koenig, N. and Howard, A. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, Sept. 2004, 2149–2154.

[2] Tingguang, L., Danny, H., Chenming, L., Delong, Z., Chaoqun, W. and Meng, M. Q.-H. HouseExpo: A Large-scale 2D Indoor Layout Dataset for Learning-based Algorithms on Mobile Robots. *arXiv preprint arXiv:1903.09845* (2019).

[3] Hennick, C. *Leaps, bounds and backflips*. 2021. URL: `https://blog.bostondynamics.com/atlas-leaps-bounds-and-backflips` (visited on 10/18/2021).

[4] Open Source Robotics Foundation. *ROS AMCL documentation*. 2020. URL: `http://wiki.ros.org/amcl` (visited on 11/10/2020).

[5] Thrun, S., Burgard, W. and Fox, D. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005. ISBN: 0262201623 9780262201629. URL: `http://www.amazon.de/gp/product/0262201623/102-8479661-9831324?v=glance&n=283155&n=507846&s=books&v=glance`.

[6] Curtis, S. *map2gazebo*. `https://github.com/shilohc/map2gazebo`. 2020.

[7] The MathWorks, I. *ROS 2 Dashing and Gazebo virtual machine installation*. 2021. URL: `https://se.mathworks.com/support/product/robotics/ros2-vm-installation-instructions-v4.html` (visited on 09/09/2021).

[8] The MathWorks, I. *Matlab AMCL sample*. 2021. URL: `https://se.mathworks.com/help/nav/ug/localize-turtlebot-using-monte-carlo-localization.html` (visited on 09/09/2021).

[9] The MathWorks, I. *controllerVFH*. 2021. URL: `https://se.mathworks.com/help/nav/ref/controllervfh-system-object.html` (visited on 09/09/2021).

[10] Open Source Robotics Foundation. *Rviz package website*. 2021. URL: `http://wiki.ros.org/rviz` (visited on 10/13/2021).