2017•2018
Faculteit Industriële ingenieurswetenschappen
master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Implementation of active Markov localisation on a mobile robot for didactic purposes

PROMOTOR :
Prof. dr. ir. Eric DEMEESTER

COPROMOTOR :
Prof. dr. ir. Ronald THOELEN

Sander Grommen

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

Gezamenlijke opleiding UHasselt en KU Leuven

2017 • 2018

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterthesis

Implementation of active Markov localisation on a mobile robot for didactic purposes

PROMOTOR :

Prof. dr. ir. Eric DEMEESTER

COPROMOTOR :

Prof. dr. ir. Ronald THOELEN

Sander Grommen

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

UHASSELT

KU LEUVEN

Acknowledgements

I would like to thank everyone who helped me achieve my thesis during the year. First of all, I would like to thank Prof. Dr. Ir. Demeester, who guided me through my thesis, by offering practical guidance when it was necessary, and for correcting my thesis.

Next, I'd like to thank Prof. Dr. Ir. Thoelen, for the multiple meetings he attended, and for providing useful feedback.

Furthermore, I'd like to thank the whole ACRO team including, but not limited to, Drs. Ing. Aerts, for helping me whenever I was confused by certain concepts.

I would also like to thank Ing. Kelher for the good advice and various discussions during the year.

Finally, I'd like to thank my family for making this all possible through financial and moral support.

Table of contents

List of Figures

List of Tables

Abstract

This thesis has been performed within KU Leuven research group ACRO at Diepenbeek, which is mainly active in the field of robotics and vision. In robotics, autonomous mobile robots such as autonomous cars or AGVs, have received increasing interest over the last decades. Localisation is crucial for these robots, since a robot has to know its location before being able to execute a task in a targeted manner. However, in symmetric environments, localisation may be ambiguous. The goal of this thesis is to implement an active localisation algorithm for a mobile robot, such that the robot takes actions to remove ambiguities in location as fast as possible. Another goal is to visualize how the algorithm “thinks” during active localisation, so that we end up with a less steep learning curve for people who are new to this field.

To achieve this the code has first been tested in Matlab, with simulated sensor and robot motion data. These simulated data will afterwards be replaced with real data measured by the mobile robot. This robot has been realised with the idea of making an easy-to-understand and reproducible product in mind.

The passive localisation algorithm has been completed and visualised, while the active localisation algorithm is still under debugging. There is a working prototype for the mobile robot so the localisation algorithms can be tested in a real environment.
.

Abstract in het Nederlands

Deze thesis is uitgevoerd binnen de KU Leuven onderzoeksgroep ACRO te Diepenbeek, die voornamelijk actief is in het onderzoeksgebied van robotica en visie. Binnen de robotica hebben autonome mobiele robots zoals autonome auto's of AGV's de afgelopen decennia steeds meer aandacht gekregen. Lokalisatie is een belangrijk onderdeel van dit onderzoeksgebied, aangezien een robot eerst zijn locatie moet weten voor deze een actie gericht kan uitvoeren. Echter kan lokalisatie in een symmetrische omgeving ambigu zijn. Het doel van deze thesis is het implementeren van een actief lokalisatie-algoritme voor een mobiele robot, zodanig dat de robot acties onderneemt om de ambiguïteit op een locatie te verwijderen. Een bijkomend doel is het visualiseren van hoe het algoritme "denkt", zodat de leercurve voor lokalisatie-algoritmes verkleind wordt.

Om dit te realiseren wordt de code eerst getest in Matlab met gesimuleerde robot-data. Hierna zullen de gesimuleerde data vervangen worden door data die gemeten worden door middel van een mobiele robot. Deze robot is gerealiseerd met als doel, een eenvoudig reproduceerbaar product te bekomen.

Het passieve lokalisatie-algoritme werkt en werd gevisualiseerd, terwijl in het actieve algoritme de laatste gekende fouten worden gecorrigeerd. Verder is er een werkend prototype van de robot, zodat de lokalisatie-algoritmen in de echte wereld getest kunnen worden.

1 Introduction

1.1 Background

This thesis has been performed within ACRO. ACRO, short for *Automation, Computer vision and Robotics,* is a research facility of the *KU Leuven,* and is located at the science park in Diepenbeek. ACRO is mainly active in the field of Robotics and Vision. [1]

This thesis focuses mainly on these fields of research. At the moment of writing this thesis, ACRO is realising the Technology Transfer (TETRA) project “Ad usum navigantium” [2]. A part of this project is the realisation and comparison of different mapping algorithms. These algorithms require distance information from their environment. At this moment this information is gathered by people who scan the room with a distance sensor. To simplify this information gathering in the future, the people will be replaced by autonomous robots. To achieve this, the robot must execute two tasks simultaneously. One task is localising itself within the environment while constructing a map by using the distance information, received from distance sensors. This simultaneous process is called SLAM.

1.2 Problem Statement

For a mobile robot to perform a task safely and correctly, it must first determine its pose. This is done based on localisation algorithms. The most common practice is passive localisation. These algorithms are, despite their popularity, not always efficient in localising the robot. The reason for this inefficiency, is that many human-made environments are symmetric and that robotic sensors typically cannot capture the rich sensory information that humans can capture. Passive localisation means that the algorithm has no control over the robot’s actuators. In other words, the actions the robot takes, are not aimed at determining its pose. A better and more efficient method for localisation is to let the localisation algorithm decide which motion or sensing action the robot takes next, insofar this is in line with the task the robot must execute. This method is called active localisation.

This active localisation is very computationally expensive and can therefore not be executed on a microcontroller, located in the robot. This means that an external processor will need to take care of the calculations, which implies that they somehow need to communicate with one another. This communicating will be done over a network that exchanges the measurements of the robot to the computer, and sends back the instructions calculated on the computer, to the robot.

These algorithms and concepts are hard to comprehend for someone who’s new to the field. Therefore, we wrap all the algorithms and concepts in an educational setting that is easy to understand and implement. This way, everyone who is interested in this field can learn in a painless manner. This is achieved by writing classes and creating printable 3D-models for the mobile robot and visualizing certain parts of the localisation algorithm.

1.3 Objectives

The main objective of this thesis is the development of an autonomous robot that will be able to localise itself within a static environment, with the aid of active localisation. The first step of realising such an algorithm, is theoretic research.

After this theoretic research, the implementation will follow. First as a passive algorithm to later build towards an active algorithm. This algorithm will be analysed and displayed for educational reasons, so everyone who uses the robot knows what is happening internally.

Afterwards a communication network will be set up to exchange the measurements with the external processor, so that it can compute heavily computational calculations and afterwards send instructions back over the same network, so the robot knows what its next move will be.

Finally the robot will be made reproducible, by writing robust classes and 3D-printable schematics and PCBs.

1.4 Research methods

The active algorithm will be written, tested and visualized in Matlab. The robot and network code will be written in the Arduino IDE [3]. The 3D-models that form the robot's shell are designed in Inventor [4] to later export to a STL-file. This file will be fed into the slicer that will print the object layer by layer. The design and drafting of the PCB takes place in EAGLE [5], afterwards the design will be transformed into a prototype PCB with the help of JLC PCB [6]. The PCBs will be manually assembled.

1.5 Outline

Chapter 2
Chapter 2 will describe localisation and the difference between passive- and active localisation, together with their Matlab implementations.

Chapter 3
Chapter 3 explains the communication network that has been used; together with the server- and client flowchart.

Chapter 4
Chapter 4 will describe in detail how the robot is build, and why certain choices, regarding components and 3D-printing, are made. It will furthermore describe the practical implementation and the design of the PCBs from top to bottom.

Chapter 5
The 5th chapter will summarize everything that has been realised, starting from breadboard circuits, to building a fully functional robot that can function over a network. The localisation algorithms will also be tested here is a simulated environment.

Chapter 6
Chapter 6 will define the conclusion of this thesis, the main bottlenecks and future work that has to be done.

2 Passive and active localisation

2.1 What is localisation?

Robot localisation is the process of determining the robots pose, which consists of its position and orientation, within a certain environment at time t. Localisation is a key part of a mobile robot, because without any knowledge of its pose the robot ca. The robot initially has no knowledge of where it is, and considers the probability of it being in a certain place equal over all possible poses **x** = [x,y,θ]. These probabilities will be updated later, based on the measured sensor data, to end up with a high probability for a certain pose. When the probability of a certain pose is higher than the localisation threshold, the robot may be considered to be localised. Figure 1 shows the workflow of localisation algorithms.

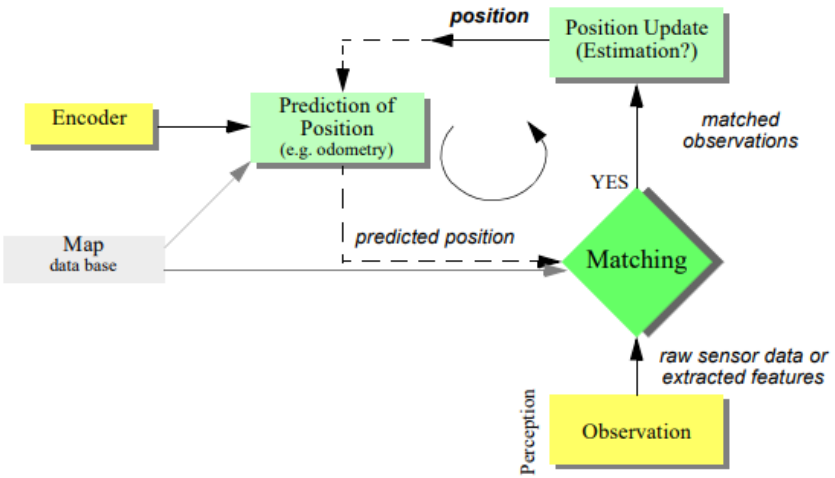

Figure 1: Workflow of localisation algorithms [7]

The process flow on Figure 1 starts at the observation of the robot's environment based on sensor data. Then the robot starts to ask itself the possibility of it moving to a certain position, knowing its movement and the map, with the help of its sensor data. This is basically the robot predicting its position based on a measurement of motion for example through the measurement of wheel rotation. This prediction is not immediately perfect, and that is why the process will repeat itself until the robot no longer doubts about its pose. Since the environment sensor measurements and robot motion sensors are subject to noise, we will adopt a probabilistic localisation approach. [7]

Before moving on to the passive localisation algorithms explained in Section 2.2, some basics need to be explained.

2.1.1 The robot state

The state of a robot is the pose of the robot. This is defined as **x** = [x,y,θ] where x and y define the position of the robot, and θ the angle in which it is positioned. If the robot moves within an environment, the state can be notated as $\mathbf{u_t} = [\{\boldsymbol{x}, \boldsymbol{y}, \theta\}\{\dot{x}, \dot{y}, \dot{\theta}\}]$, where x, y and θ are the current coordinates and orientation and $\dot{x}$, $\dot{y}$ and $\dot{\theta}$ are the new coordinates and orientation. [8, pp. 20–22]

2.1.2 The environment

Many things in a real-life environment can be considered as random, a person kicking the robot, the wind giving it a hard blow, a slippery surface which makes it harder to move. All these uncertainties can be countered by modifying the localisation algorithm. In this thesis the environment is static, so there are no moving objects and no external influences.

2.1.3 The believe (Bel)

Initially, the robot state $\boldsymbol{x_t}$ may not be known; in that case, the robot initially considers the probability of it being in a certain state equal among all possible states, the belief state bel is uniformly distributed. Bel($\boldsymbol{x_t}$) denotes the probability that the robot is located at position $\boldsymbol{x_t}$ at time t. Over time, when considering motion measurements and measurements of the robot's surroundings, this believe will no longer be equally distributed but will change into a high certainty for a certain state, the state in which the robot is localised. [8, p. 25]

2.1.4 Motion and sensor models

Moving the robot requires engines, and measuring data requires sensors. In a perfect, and unrealistic, world both are perfectly accurate. In reality noise plays an important factor and decreases the accuracy of every movement and measurement. To cope with these inaccuracies, the algorithm uses models which describes these inaccuracies. There are two main models the sensor model and the motion model. The sensor model gives the probability of measuring the distance $\boldsymbol{z_t}$ at the state $\boldsymbol{x_t}$ and is noted as $p(\boldsymbol{z_t}|\boldsymbol{x_t})$. The motion model gives the probability that the robot can move a distance $\boldsymbol{u_t}$ if the robot starts moving at state $\boldsymbol{x_t}$ and is noted as $p(\boldsymbol{u_t}|\boldsymbol{x_t})$. [8, pp. 22–24]

2.1.5 The Bayes filter algorithm

This algorithm is the main building block of localisation. In localisation we want to calculate the belief of a robot being in state $\boldsymbol{x_t}$, given state $\boldsymbol{x_{t\text{-}1}}$ and the measurement data regarding the robot's surroundings at time *t*, $\boldsymbol{z_t}$. This is done in two steps, one is called *prediction*, which calculates $\boldsymbol{x_t}$ based on $\boldsymbol{x_{t\text{-}1}}$ and measurements regarding the robot's motion, and the other one is called *correction*, which corrects the calculated $\boldsymbol{x_{t\text{-}1}}$ given the measurement(s) $\boldsymbol{z_t}$. The formula for calculating the prediction is as follows:

$$\overline{bel}(\boldsymbol{x_t}) = \int \mathrm{p}(\boldsymbol{x_t}|\boldsymbol{u_t}, \boldsymbol{x_{t\text{-}1}}) * \mathrm{bel}(\boldsymbol{x_{t\text{-}1}})\, \mathrm{d}\boldsymbol{x}_{\mathrm{t\text{-}1}} \qquad \text{Eq(2.1)}$$

This formula states that the new belief is equal to the old belief times the certainty of $\boldsymbol{x_t}$ given the movement measurement $\boldsymbol{u_t}$ of the robot and the previous state, integrated over all possible previous robot states $\boldsymbol{x_{t\text{-}1}}$. The movement of the robot is important because of certain inaccuracies within the engine itself, as mentioned before.

The correction step will update this belief, considering the inaccuracy of the sensors that measure the environment. This leads to the following formula:

$$bel(\boldsymbol{x_t}) = n^* p(\boldsymbol{z_t}|\, \boldsymbol{x_t}, m)^* \overline{bel}(\boldsymbol{x_t}) \qquad \text{Eq(2.2)}$$

This formula states that the new belief in pose $\boldsymbol{x_t}$ is equal to the belief obtained in the prediction step times the probability of sensor data $\boldsymbol{z_t}$ occuring at state $\boldsymbol{x_t}$ given the map, or environment, m; this takes the uncertainty of the measurement data into account. Factor n can be considered as a scaling factor used for normalisation. [8, pp. 26–28]

Knowing all the basics, let us move on to the differences between passive and active localisation, and their sub-algorithms.

2.2 Passive localisation as base for active localisation

Passive localisation is a very common type of localisation but may be, as explained earlier, inefficient in symmetric environments. Nowadays, popular passive localisation algorithms are the Markov and Monte Carlo localisation algorithms.

2.2.1 Markov localisation

Markov localisation uses a histogram filter that divides the robot's environment into a set of bins, i.e. non-overlapping and typically rectangular cells. Each bin represents a possible state of the robot. The Markov localisation algorithm calculates the probability or belief of all bins with the discrete Bayes Filter principle. This assumes a finite environment and the robot is assumed to be localised in one of the bins [8, pp. 86–89]; the continuous Bayes formula Eq(2.1) becomes for a discrete state space:

$$bel(\boldsymbol{x_t}) = \sum_{\boldsymbol{x_{t-1}}} p(\boldsymbol{x_t}|\boldsymbol{u_t}, \boldsymbol{x_{t\text{-}1}})^* bel(\boldsymbol{x_t}) \qquad \text{Eq(2.3)}$$

This calculates the new belief for every state given the old belief for the previous states. The correction of this algorithm remains the same and is determined by:

$$bel(\boldsymbol{x_t}) = n * p(\boldsymbol{z_t}|\boldsymbol{x_t}, m)^* bel(\boldsymbol{x_t}) \qquad \text{Eq(2.4)}$$

Figure 2 shows such a bin division:

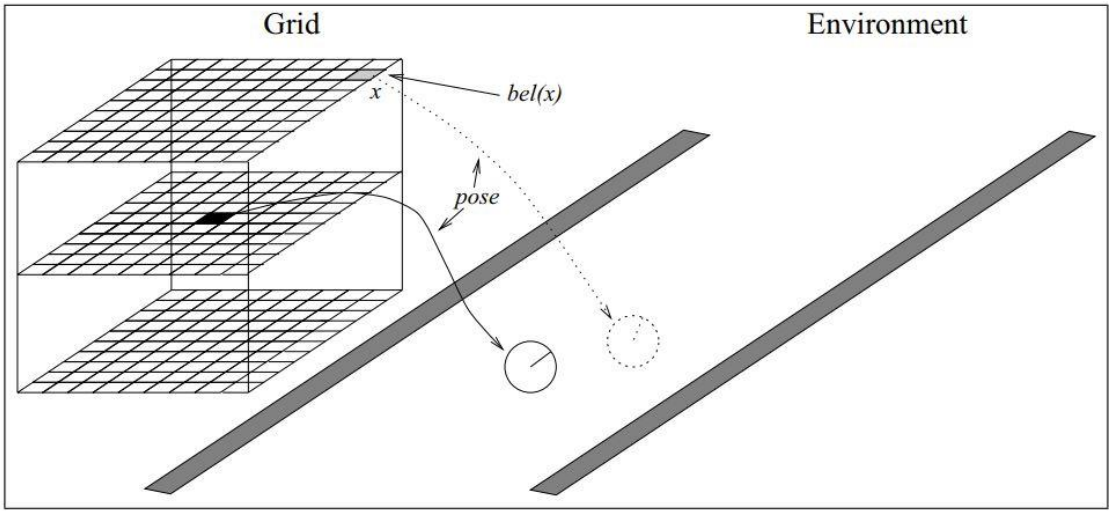

Figure 2: Representation of grid cells [8, p. 241]

In Figure 2 each grid cell represents a location in which the robot could be located, and each layer defines the orientation, or angle, of the robot. This leads to x*y*$\boldsymbol{\theta}$ bins, each with its own probability.

The next set of images will explain the process of Markov localisation better. In the initial step the robot has not moved yet and considers the probability of it being in a bin equal over all bins. This is given below.

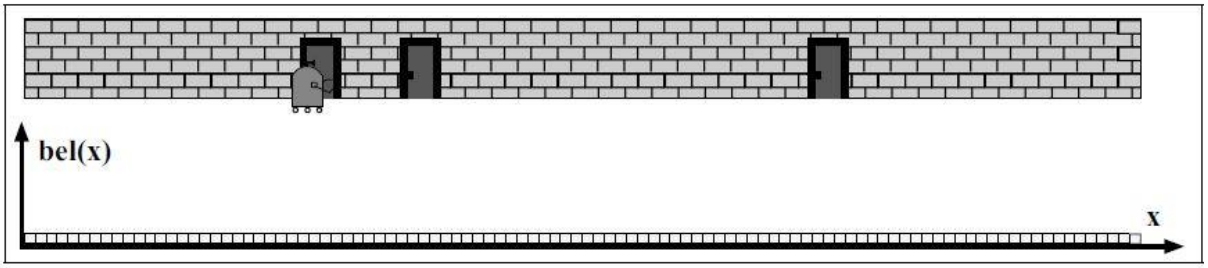

Figure 3: Markov localisation, initial probability distribution [8, p. 240]

Next the measurement- or sensor data will update its belief since the robot has knowledge of the room, and it can measure that it is located at a door. This is displayed by the next illustration. However, given that there are three doors in the environment, the robot is still unsure about the door at which it is located.

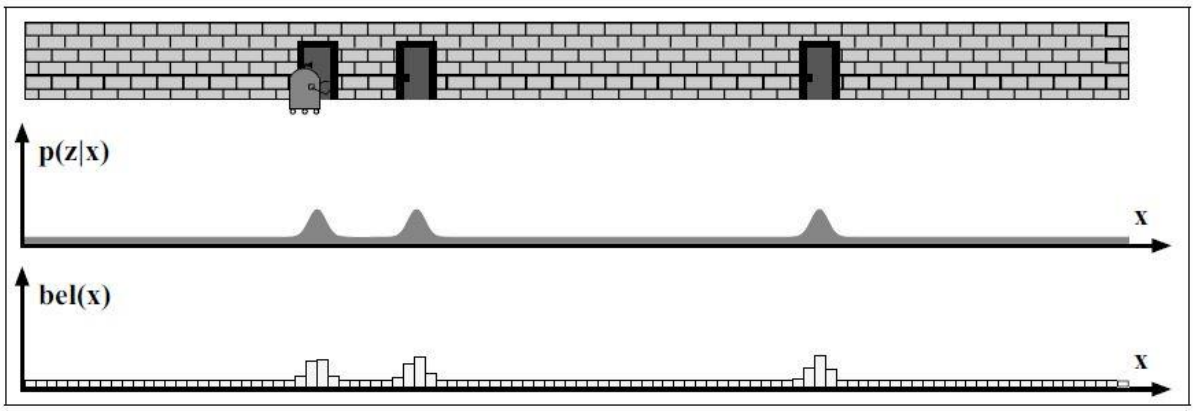

Figure 4: Markov localisation, measurement update at t=0 [8, p. 240]

Now comes a movement update (Eq2.3), the robot will move into a direction, and the beliefs automatically move along. The uncertainty in the distribution increases, as the robot's movement is not perfectly known.

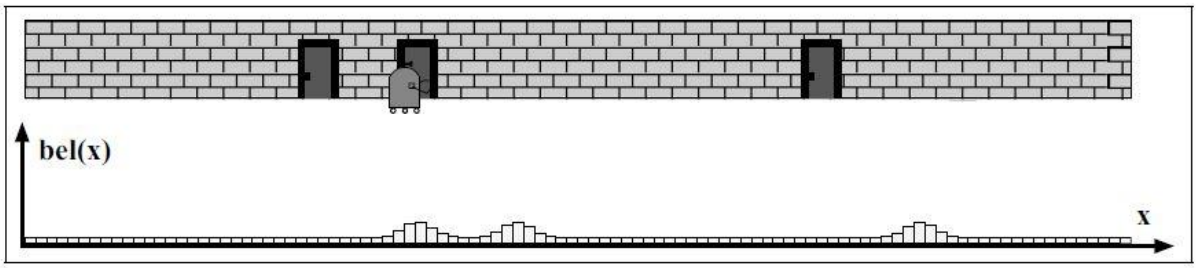

Figure 5: Markov localisation, motion update at t=1 [8, p. 240]

Then, the robot measures its environment again and detects that there is a door located next to him, the robot has probabilistic knowledge of its previous position, in front of a door, the room and the update in movement it made, and therefore knows with a high certainty where it is located. This will update the belief of the robot being at the second door.

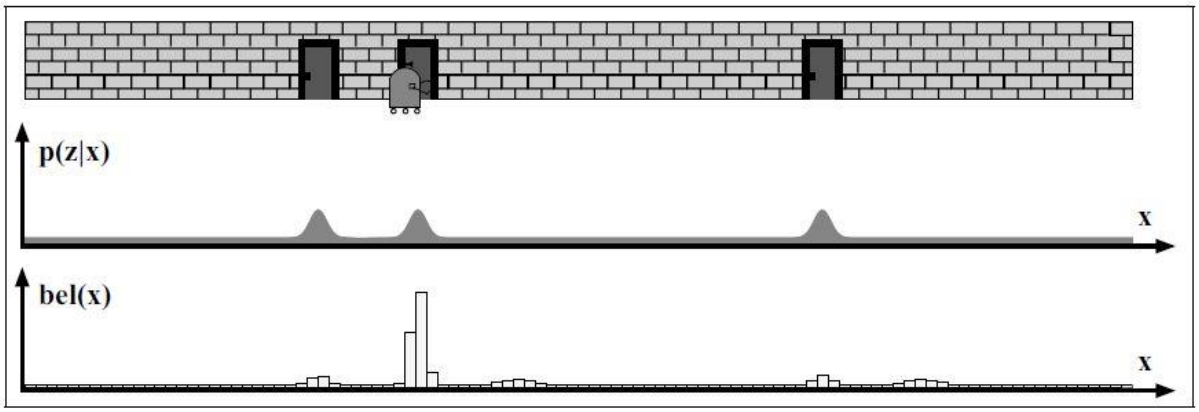

Figure 6: Markov localisation, measurement update at t=1 [8, p. 240]

The robot has now been localised and will now know at any given moment where its location is on the known static map.

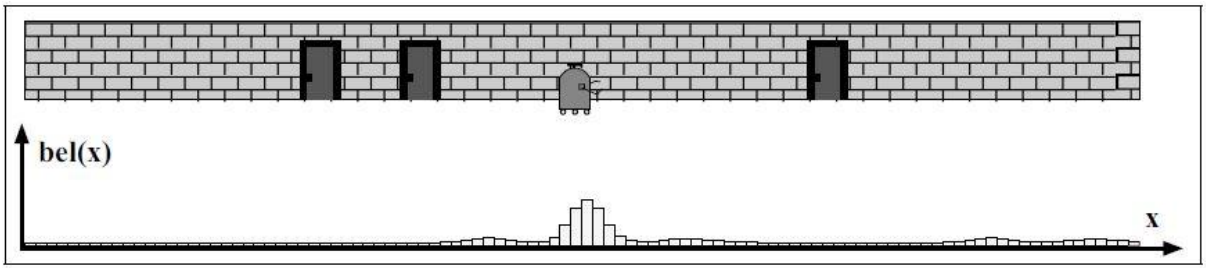

Figure 7: Markov localisation, robot localized [8, p. 240]

2.2.2 Monte carlo localisation

This algorithm differentiates itself from Markov Localisation by being based on particles, therefore it is often referenced to as a particle filter. Particles are basically possible locations of the robot, which are initially uniformly spread over the whole room. The robot moves, and the particles move along. The robot now measures the room with a distance sensor and weighs the particles on how likely they are given the measurements.

The next step for the algorithm is to generate new particles, these particles will be distributed according to their weight. This means that an area with a higher weight will now have a higher particle distribution. The robot moves again, and the process repeats until all particles are within the same area, or when the weight of a particular particle is distinguishably higher than that of the other particles, the robot has localised itself. [8, pp. 250–252]

The following images will illustrate this localisation technique and provide a less abstract explanation. In the initial state the particles are uniformly distributed over every possible state. The robot has no idea of his location.

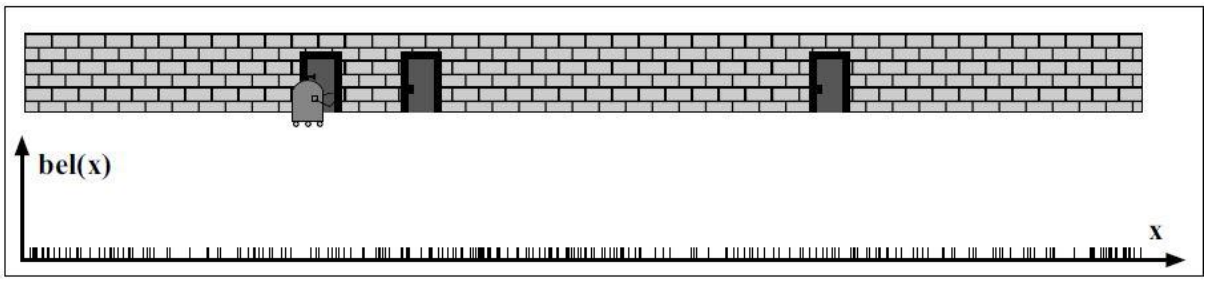

Figure 8: Monte Carlo localisation, distribute the particles over the enviroment [8, p. 251]

Next, the measuring starts, and the robot will notice that it is located near a door, this will add weight to the believe of the states which can observe a door.

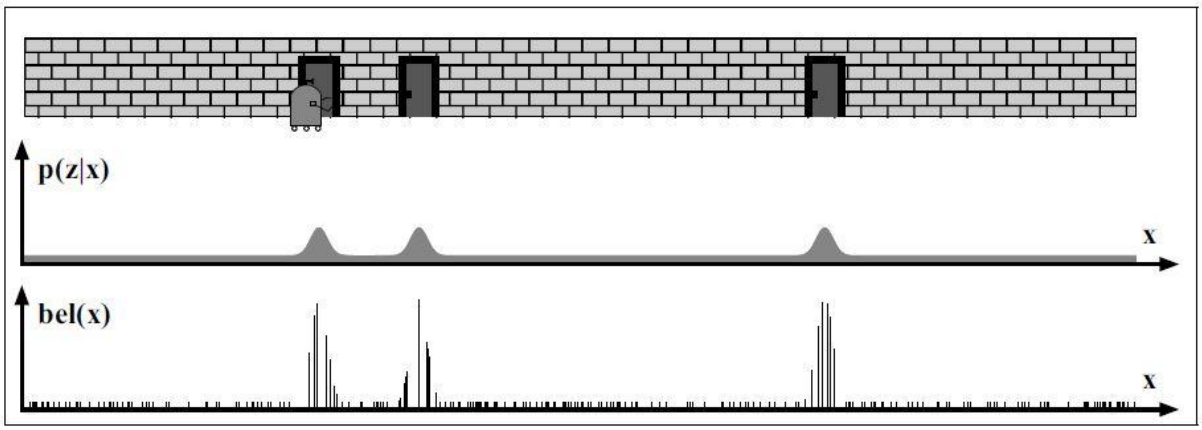

Figure 9: Monte Carlo localization, measurement update at t=0 [8, p. 251]

The particles get replaced and now the places with a high weight will have a higher particle distribution. The robot moves, and the particles move along in the same direction.

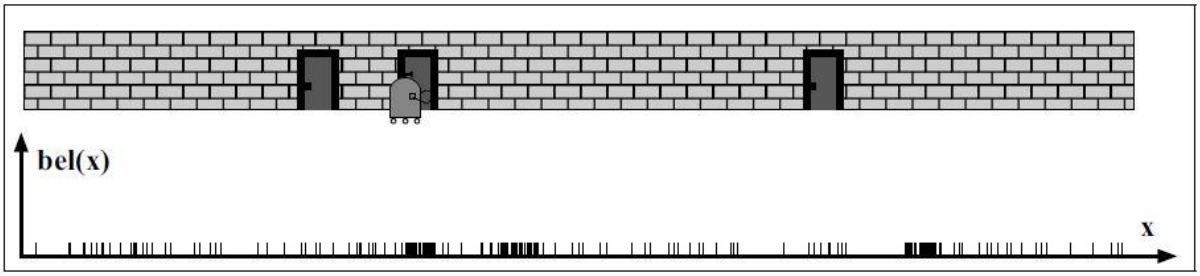

Figure 10: Monte Carlo localisation, resample the particles at t=1 [8, p. 251]

New measurements will be applied, and it measures a new door; given its previous position and the room, it knows where it is located. This will make sure that the robot's belief at the right (but unknown) location, will have the highest weight.

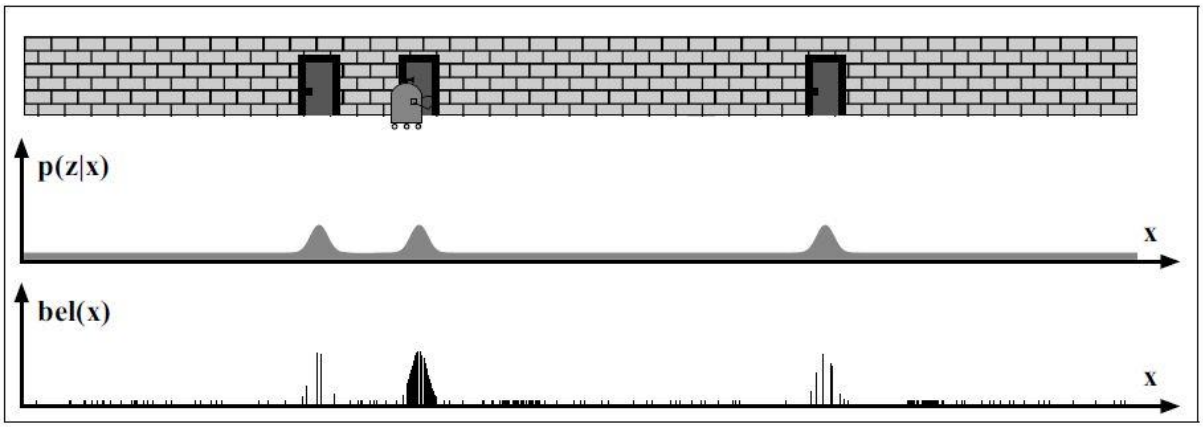

Figure 11: Monte Carlo localisation, measurement update at t=1 [8, p. 251]

The robot has localised itself, now most particles will be distributed around the true robot location and follow along with its movements. The robot knows the room and its pose in relation to the room.

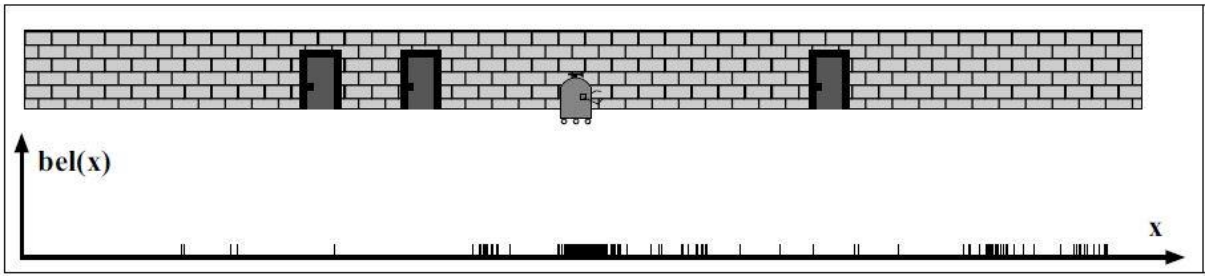

Figure 12: Monte Carlo localisation, the robot localized [8, p. 251]

2.3. Active localization by entropy reduction

Active localization is different from passive localization because now the robot gets its action commands from the algorithm to determine his location faster. This action instructions are calculated to maximise the localisation efficiency. There are two main steps that a robot needs to go through to localise itself. The first one is called active navigation, here the robot asks himself what the best movement action would be to minimize the entropy. The second one is active sensing, here's the question not where to move, but where to look. These steps will be explained in more detail in 2.3.2 Active navigation and 2.3.3 Active sensing., but first the basics.

2.3.1 Basic equations.

To determine the optimal action for the robot, the algorithm needs to make a tradeoff between the utility U_t and the cost C_t of a certain action **a**. The goal of the robot is to minimize this entropy, which means that the belief of a robot is maximized on a certain state. Initially we assume a belief that is uniformly distributed over all states, so the entropy here is at its peak. When the robot is localised the entropy has reached its zero-point, meaning a complete certainty of the robot's location. [9, p. 5]
The following formula shows this entropy of belief:

$$H(L) = -\sum_l Bel(L = l) \log(Bel(L = l)) \qquad \text{Eq(2.5)}$$

$H(L)$ measures the robot's uncertainty of its state.
The expected entropy, and therefore simulated, is defined by $E_{\boldsymbol{a}}[H(L_{t+1})]$. This is the entropy in a new location after performing an action, and measurement. This means that we can calculate the decrease in entropy for all possible actions, by subtracting the entropy $H(L_t)$ and the expected entropy $E_{\boldsymbol{a}}$. [9, p. 5] This reduction is called the utility factor U_t and is given by the following formula.

$$U_t(\boldsymbol{a}) = H(L_t) - E_{\boldsymbol{a}}[H(L_{t+1})] \qquad \text{Eq(2.6)}$$

This states that the utility factor is equal to the current entropy minus the expected entropy after performing action a and measuring the environment. This utility factor is the decrease in uncertainty and needs to be maximized so that we achieve a robot that's certain of its location. To achieve this the algorithm needs to calculate $E_{\boldsymbol{a}}[H(L_{t+1})]$, for every given state that the robot can move to, so that the robot can choose to move to the state that delivers the highest utility factor, or the highest decrease in uncertainty [9, p. 5]. This is done by the following formulas.

$$E_{\boldsymbol{a}}[H(L_{t+1})] = \sum_s H(L_{t+1}|s, \boldsymbol{a}) * p(s|\boldsymbol{a}) \qquad \text{Eq(2.7)}$$

This formula states that the predicted entropy equals the entropy, in the future location after executing the action and measuring the environment, times the certainty of those measurements happening after action a. This can be further written by filling in the entropy as function of belief Eq(2.5) in Eq(2.7):

$$E_{\boldsymbol{a}}[H(L_{t+1})] = -\sum_{s,l} Bel(L_{t+1} = l|s, \boldsymbol{a}) * \log(Bel(L_{t+1} = l|s, \boldsymbol{a})) * p(s|\boldsymbol{a}) \qquad \text{Eq(2.8)}$$

In the last two formulas you can see that the action is already performed and that the sensing is already done, this sensing is impossible to predict, and therefore can't be simulated. Therefore, the formula can be written as:

$$E_{\boldsymbol{a}}[H(L_{t+1})] = -\sum_{s,l} p(s|l)*Bel(L_{t+1} = l|\boldsymbol{a})* \log\left(\frac{p(s|l)*Bel(L_{t+1}=l|\boldsymbol{a})}{p(s|\boldsymbol{a})}\right) \qquad \text{Eq(2.9)}$$

Here you can see that the sensing isn't modeled yet, the only thing that is modeled is the certainty of the sensing being accurate after performing action a, or at location l. This can be modeled and therefore the utility factor can be predicted before performing an action, making the algorithm much more efficient. To further calculate the optimal action, one has also take into account the cost for executing given action, there are many ways to implement such a cost function, eg. the time the robot needs to perform an action[9, pp. 5–6].The final formula for calculating the optimal action is:

$$\boldsymbol{a}^* = \operatorname{argmax}(U_t(\boldsymbol{a}) - \beta * C_t(\boldsymbol{a})) \qquad \text{Eq(2.10)}$$

The chosen action is the action that maximizes $\boldsymbol{a}^*$, here β is a scaling or weight factor, $U_t(\boldsymbol{a})$ is the utility of given action and $C_t(\boldsymbol{a})$ the cost.

Both active navigation and - sensing have their own Interpretation of these formulas, which will be explained in the next chapters.

2.3.2 Active navigation

Active navigation attempts to select the optimal movement by applying a cost calculation based on the distance. This defines how much it costs for a robot to execute the action. The chosen formula is displayed below:

$$C_t(a) = \sum_l Bel(L_t = f_{\boldsymbol{a}}(l)^{-1}) * D_{path} + p_{occ}(l) * \delta \qquad \text{Eq(2.11)}$$

The cost is the sum of every state with a probability $Bel(L_t = f_{\boldsymbol{a}}(l)^{-1})$ times the path distances D_{path} with an additional offset $p_{occ}(l) * \delta$, why this offset is needed will be explained in the A star algorithm. Here $f_{\boldsymbol{a}}(l)^{-1}$ is the inverse of the movement formulas defined by:

$$\begin{aligned} [f_{\boldsymbol{a}}(l)]_x &= l_x + \cos(l_\alpha)*a_f + \sin(l_\alpha)*a_s \\ [f_{\boldsymbol{a}}(l)]_y &= l_y + \sin(l_\alpha)*a_f - \cos(l_\alpha)*a_s \\ [f_{\boldsymbol{a}}(l)]_\alpha &= l_\alpha + a_\alpha \end{aligned} \qquad \text{Eq(2.12)}$$

These formulas can define a translation and rotation, they map the action of the robot to euclidean grid coordinates.

The path distance Dpath, is determined with the A*-algorithm. This algorithm is based on the next formula:

$$f = g + h \qquad \text{Eq(2.13)}$$

Here g represents the movement cost to go from the starting point to a certain grid cell on the map. While h represents the movement cost to get from that certain grid cell to the end destination. h is the flight distance between the two points, for the sake of simplicity.
The grid cell that the robot is going to move to, will be the one with the lowest f. This is the grid cell that will make sure that action **a** has the lowest possible movement cost. [10]

Figure 13 displays the distance g in blue, and h in red, while the starting point is located in the lower left corner, and the target in the upper right.

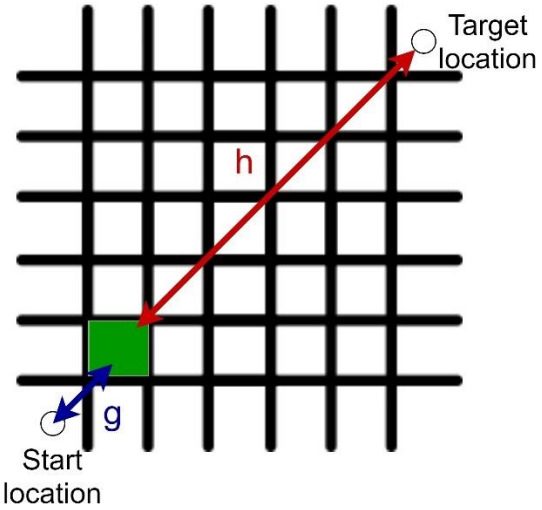

Figure 13: Representation of h and g in the A algorithm*

If we apply the f, or optimal path for each movement we get a map that looks like the one given in Figure 14

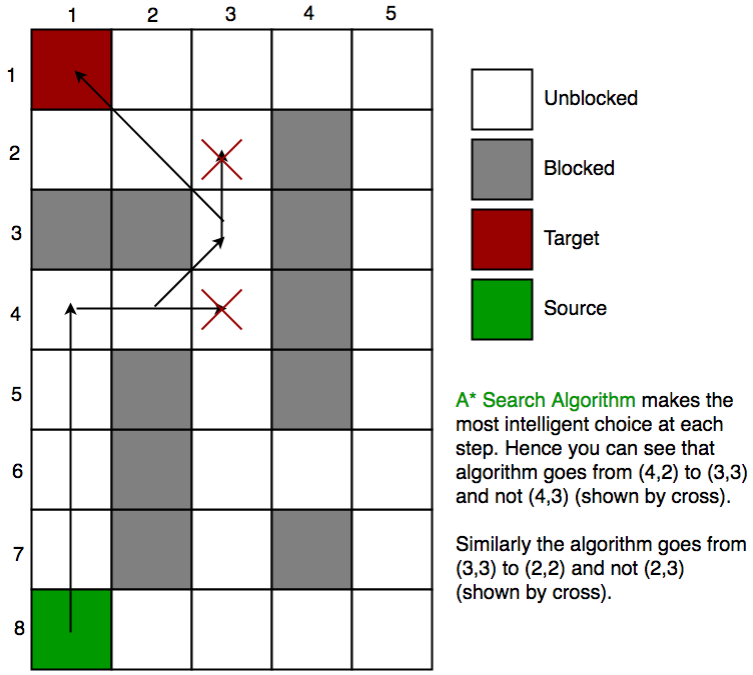

*Figure 14: Pathplanning with A** [11]

The algorithm returns the shortest path, $\mathrm{D}_{\mathrm{path}}$, between the starting point of the robot and the target location of the robot. There is however one exception, this is when the destination is located on an object. In this case the robot will not be able to calculate its path and assign zero to $\mathrm{D}_{\mathrm{path}}$, leading to a zero cost. This is unwanted behavior and is countered by an offset in equation Eq(2.11), this offset penalizes destination points by a weight δ, because the probability of the target location being blocked, $\mathrm{p}_{\mathrm{occ}}(\mathrm{l})$, is one.

The expected entropy slightly variates from the basic entropy explained in formula Eq(2.9) and is given by:

$$E_{\boldsymbol{a}}[H(L_{t+1})] = -\sum_{s,l} p(s|l)*\mathrm{Bel}(L_t = f_{\boldsymbol{a}}(l)^{-1})* \log\left(\frac{p(s|l)*\mathrm{Bel}(L_t = f_{\boldsymbol{a}}(l)^{-1})}{p(s|\boldsymbol{a})}\right) \quad \text{Eq(2.14)}$$

This defines the entropy expected after executing navigation action **a**. To compute $Bel(L_{t+1} = l|\boldsymbol{a})$ the inverse of the coordinate transformation given in Eq(2.12) is applied

The final step in active navigation is determining the optimal action, which will help the robot locate itself faster. This can be done by using the utility and cost in equation Eq(2.10). [9, p. 7]

2.3.3 Active sensing

Active sensing aims to look in a direction most optimal for the robot. If the robot's action would be to point the sensor in a certain direction, then the expected entropy of such action would be given by following formula.

$$E_{\boldsymbol{a}}[H(L_{t+1})] = -\sum_{s_\alpha,l} p(s_\alpha|l)*\mathrm{Bel}(L_t = l)* \log\left(\frac{p(s_\alpha|l)*\mathrm{Bel}(L_t = l)}{p(s_\alpha|\boldsymbol{a})}\right) \quad \text{Eq(2.15}$$

Here s is the sensing that can be perceived while pointing in direction **α**. Please note that $\mathrm{Bel}(L_{t+1} = l|\boldsymbol{a})$ from previous equation Eq(2.9) has been replaced by Bel(Lt = l) because pointing a sensor in a certain direction does not require any movement from the robot, but rather from the sensor head.
This thesis assumes that the cost for pointing a sensor is not dependent on the pose of the robot, thus we can neglect any cost calculations and only focus on the utility, as defined previously in Eq(2.10). [9, p. 8]

2.4 Active localization in Matlab

2.4.1 Passive localization

This code and explanation is heavily inspired by Parker Lusk's grid-localisation code. [12] Several parts of the code have been debugged, and drastically changed, in order for it to work in this application. The algorithm that will be explained is the Markov algorithm, this because it's the easiest algorithm to visualise, the hardest algorithm to implement who therefore explains the base of localisation very well. This algorithm can also easily be expanded towards a dynamic environment and can rectify the robot movements introduced by external forces. The cost of all these advantages is that the algorithm is very computationally expensive once the room expands.

The following code chart depicts in simple code what the matlab code does.

1: **Algorithm Grid_localization**$(\{p_{k,t-1}\}, u_t, z_t, m)$**:**
2: *for all* k *do*
3: $\bar{p}_{k,t} = \sum_i p_{i,t-1}$ **motion_model**$(\text{mean}(\mathbf{x}_k), u_t, \text{mean}(\mathbf{x}_i))$
4: $p_{k,t} = \eta\, \bar{p}_{k,t}$ **measurement_model**$(z_t, \text{mean}(\mathbf{x}_k), m)$
5: *endfor*
6: *return* $\{p_{k,t}\}$

Figure 15: Simple code Markov localization [8, p. 238]

Line 3, in Figure 15 is the Markov update step, Eq(2.3), in this equation the motion model is written as $p(\boldsymbol{x_t}|\boldsymbol{u_t}, \boldsymbol{x_{t\text{-}1}})$, which indicates how certain it is that a instructed movement $\boldsymbol{u_t}$ has been executed precisely, this possible uncertainty is because of noise within the engines. The motion model is calculated by the following code.

1: **Algorithm motion_model_odometry**(x_t, u_t, x_{t-1})**:**

2: $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
3: $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
4: $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5: $\hat{\delta}_{\text{rot1}} = \text{atan2}(y' - y, x' - x) - \theta$
6: $\hat{\delta}_{\text{trans}} = \sqrt{(x - x')^2 + (y - y')^2}$
7: $\hat{\delta}_{\text{rot2}} = \theta' - \theta - \hat{\delta}_{\text{rot1}}$

8: $p_1 = \textbf{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1\hat{\delta}^2_{\text{rot1}} + \alpha_2\hat{\delta}^2_{\text{trans}})$
9: $p_2 = \textbf{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3\hat{\delta}^2_{\text{trans}} + \alpha_4\hat{\delta}^2_{\text{rot1}} + \alpha_4\hat{\delta}^2_{\text{rot2}})$
10: $p_3 = \textbf{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1\hat{\delta}^2_{\text{rot2}} + \alpha_2\hat{\delta}^2_{\text{trans}})$

11: *return* $p_1 \cdot p_2 \cdot p_3$

Figure 16: Simple code motion model [8, p. 134]

This code compares the hypothetical movement with the actual movement. A movement exists out of three separate movements, a first rotation, a translation and a final rotation. Figure 17 shows these movements.

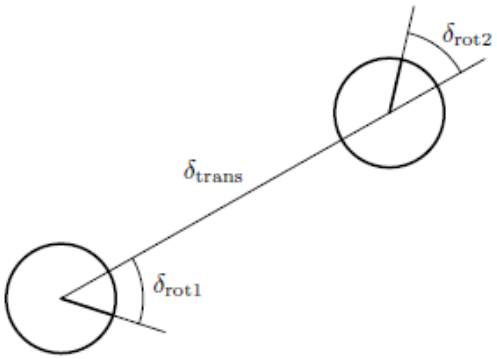

Figure 17: Odometry model of movement [8, p. 133]

The first rotation equals the rotation you have to get to, minus the begin rotation. So, if the robot's goal is to reach 90° and it is located at 10°, then the rotation equals 80°. The begin rotation is initially 0, this is done because the robot is not allowed to know its actual orientation in the map. So even if the robot is orientated at 10°, this will be considered as 0°

The next movement is a translation which is given by a variant of the Pythagoras formula, and will not be explained in detail.

The final rotation is equal to the angle it needs to get to minus the first rotation minus the begin orientation. So, if the robot wants to move to 100°, we get 100° - 80° - 10° = 10° which would be the final rotation of the model.

These three movements are compared to what movement has actually been done by the robot and the function finally returns a certainty of the executed movement.

Line 4 In Figure 15 is the Markov update step, Eq(2.4). In this equation the measurement model is defined by $p(\boldsymbol{z_t}|\boldsymbol{x_t}, m)$, which indicates how accurate the read out of the location is by giving it a certainty.

The measurement model is implemented by following code chart.

1: **Algorithm beam_range_finder_model**(z_t, x_t, m)**:**

2: $q = 1$
3: *for* $k = 1$ *to* K *do*
4: *compute* z_t^{k*} *for the measurement* z_t^k *using ray casting*
5: $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$
6: $+z_{\text{max}} \cdot p_{\text{max}}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$
7: $q = q \cdot p$
8: *return* q

Figure 18: Simple code measurement model [8, p. 158]

This chart calculates the measurement probability for each sensor separately. This probability can be split into four separate probabilities. The first defines the measurement error of the sensor, the second the measuring of an unexpected object (which is not implemented in our application, since we assume a static environment), the third are measurement errors (measuring 8m while the room is only 5m is impossible) and finally there is the probability of random measurements that are left unexplained. These separate probabilities sum up to a total certainty indicating the accuracy of the sensor measurement.

2.4.2 Active localization

To refresh the theory of active localisation, following formula is presented:

$$\boldsymbol{a}^* = \text{argmax}(U_t(\boldsymbol{a}) - \beta^* C_t(\boldsymbol{a})) \qquad \text{Eq(2.10)}$$

This formula exists out of two factors, the utility and the weighted cost. Active navigation has been implemented first, since it redefines both the cost and the utility, giving it the most educational value. To calculate the cost we used to formula in equation Eq(2.11), this has been fully explained in 2.3. Active localization by entropy reduction, and not everything will be explained again. The implementation of the A*-algorithm to calculate the D_{path} of a given action has been limited to movements of 90°, this is a limitation of the passive localization algorithm. Figure 19 shows that the Matlab implementation has found the optimal path to a certain target point on the map.

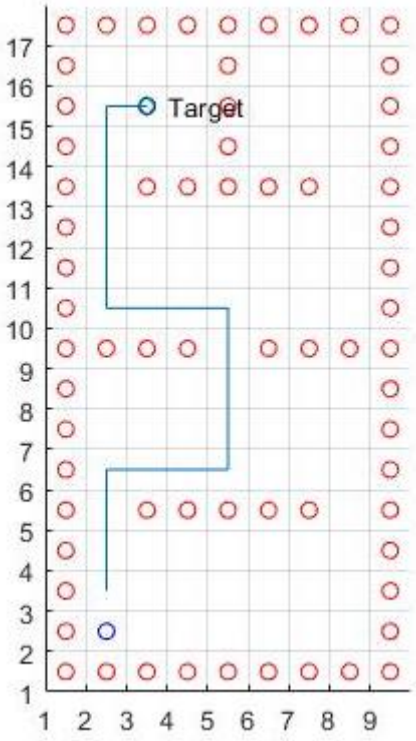

Figure 19: Path calculation in matlab

The sigma as seen in equation Eq(2.11) has been given the value 3. This to penalise objects as the target location. The utility is defined by the current entropy and the expected entropy, as shown in equation Eq(2.6). The expected entropy has been determined by the equation in Eq(2.14), this is the expected entropy for active navigation and will be repeated below.

$$E_{\boldsymbol{a}}[H(L_{t+1})] = -\sum_{s,l} p(s|l)*\text{Bel}(L_t = f_{\boldsymbol{a}}(l)^{-1})* \log\left(\frac{p(s|l)*\text{Bel}(L_t = f_{\boldsymbol{a}}(l)^{-1})}{p(s|\boldsymbol{a})}\right) \qquad \text{Eq(2.14)}$$

This formula is not successfully implemented in Matlab. The red section of the formula is the main bottleneck of this implementation, see following explanation that will try to explain why this is so.

Before explaining any further, let's assume a perfect world, where there's no noise, making the robot 100% certain of its movements and measurements. This assumption is made to narrow the scope of our problem. The image below shows that the algorithm considers two different locations for the robot. The black square stands for a high certainty, the gray for a low one, both at an angle of 0°. The current entropy can be calculated by Eq(2.5)which leads us to a value of 0.5004.

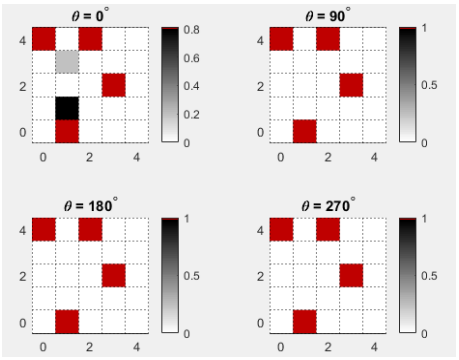

Figure 20: Representation of belief in Matlab

The next step is to let the robot perform an action, as explained in 2.2.1 Markov localisation, the beliefs move along with the robot. The action that has been chosen, is an action that can finished by both belief states (the two squares), while staying in the provided map.
If we calculate the expected entropy, by using the provided formula in Eq(2.14), one can notice that it solely depends on the beliefs, given the fact that the measurements and the movements are completely accurate. This means that the expected entropy remains the same, given that the beliefs do as well, which is the case. All the beliefs (the two squares, being 0.2 and 0.8) sum up to 1 therefore there's no need for a normaliser $p(s|\boldsymbol{a})$.

However, if one of the belief states, would undergo an action that forces itself to move outside the map, then the belief of that state would be thrown away, and $\mathrm{Bel}(\mathrm{L_t} = \mathrm{f}_{\boldsymbol{a}}(\mathrm{l})^{-1}$ would be equal to 0. This is unwanted behavior, since the sum of all beliefs would be less than 1, meaning that the robot has permanently lost certainty. As example, it the belief state 0.2 would be forced out of the map, only 0.8 would remain. The algorithm now has 0.8 as its maximum certainty, and therefore needs to be rescaled. This needs to be tackled by the normaliser p(s|a), which implementation is not yet clear, leaving the algorithm unfinished.

Figure 21 shows the expected entropy, proving that there's need for a normalizer. At action (0,0) there is the expected entropy of 0.5004. Now when the action 'move to the right' is introduced (see the green line), the expected entropy stays the same. This is because both states can go to the right, without meeting obstacles, until the end of the map, here both beliefs, and therefore the expected entropy disappears. When the action 'move down' is introduced (see the orange line), you'll notice that at a given point the expected entropy lowers, meaning that the sum of all beliefs for that action has decreased. This is caused by the black square that moved onto an object (red square), this is the unwanted behaviour that a normaliser, p(s|a), needs to counter.

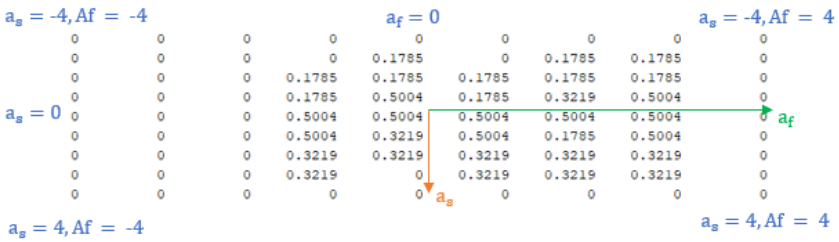

Figure 21: Expected entropy Ea

3 Communication network

3.1 Network structure

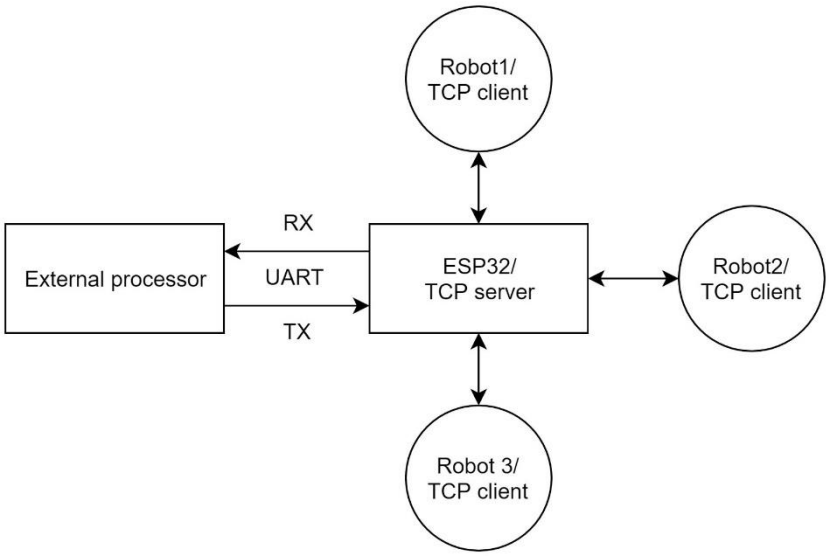

Figure 22:Structure of the communication network

The chosen network is as one can see in Figure 22 a star-network. The PC is the external processing power that does all the calculations based on the measurement data from the robot. These measurement data are sent by the central server, which receives the data from the client ESP32s, located on the robot. The transmitting between client and server is done over WiFi, while the server stands as an access point, allowing the PC to communicate over a UART connection, in case no network card is present. Once all calculations are made, movement instructions will flow from the PC to the server, who on its turn, transmits everything to the connected client. This entire communication flow, or structure, is realised with the TCP/IP protocol.

3.2 Code

3.2.1 TCP Server

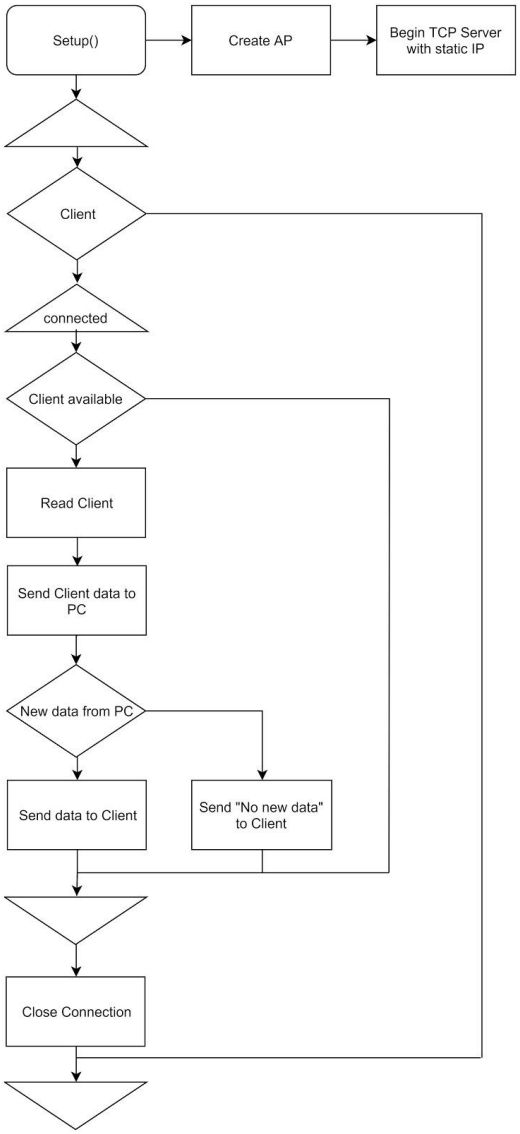

Figure 23: Flowchart TCP-server code

The server is setup and waits till it receives a HTTP GET message from the client. Once the message is received, the data transmitted from the client will be read, and send further to the PC, which will calculate and determine the next instruction based on the received data. The client does not wait till the PC has finished its calculations, this would mean that the client needs to keep its connection open which is bad for the battery life. Instead, the client closes the connection and reopens it within 1 second to ask if the server has already received new instructions from the PC. Once the PC is done with calculating, instructions will be sent back to the server, and back to the client. The client will make sure that the robot finishes the instructions. Once finished, the client will repeat the process.

3.2.2 TCP Client/Robot

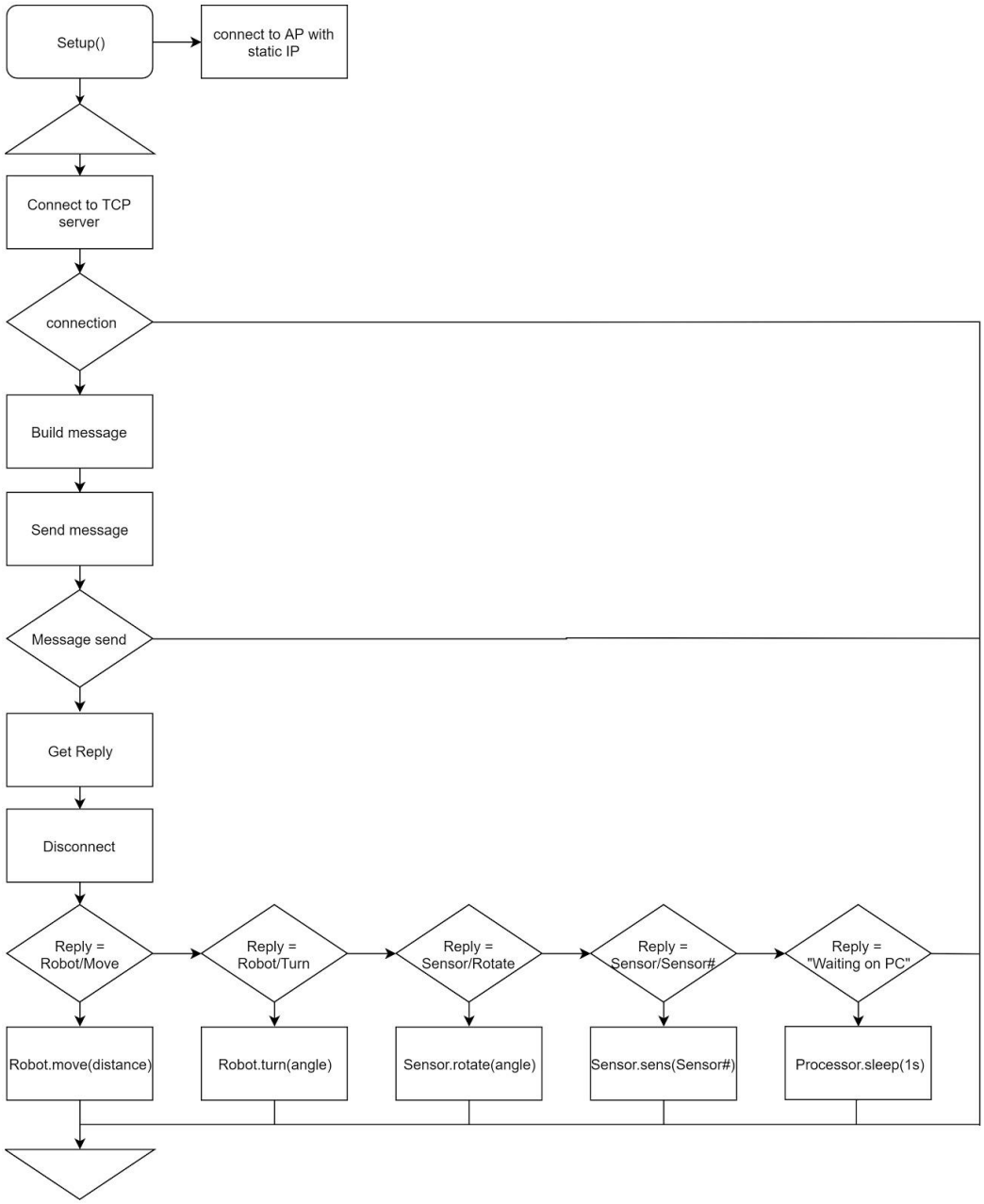

Figure 24: Flowchart robot code

The client, which is also the robot, will initialise and start a connection with the server. Once established, the client prepares a HTTP GET message containing all the measurement data. This message will be send to the server, who on his turn, will forward it to the PC. The PC starts calculating and the server sends a message back to the client to indicate that the PC is busy. The client will disconnect and retry after 1 second. Once the PC is done calculating, the server will send the data to the client, and the client will process the instructions and communicate them with the robot. The robot executes these instructions, and the process can repeat itself.

4 Mobile robot

The mobile robot is used to navigate himself through the environment while gathering sensor data for the localisation algorithm. This robot is build while keeping its educational values in mind, therefore most of the chosen components are easily obtainable on the market, for a fair price. Furthermore, the robot's PCB-models are build to make it easily reproducible, by minimizing the required know-how of electronics. The 3D-models make the robot complete, these models can be printed by an amateur, and the sensors can easily be replaced by other variants.

4.1 Hardware selection

This chapter will describe the chosen components that form the heart of the robot, and the reasoning behind them. The most deterministic factor of choosing most of the components, was their availability, market price, power usage and operating voltage. These characteristics are import because of the educational and easily reproducible background of the robot, and to make the robot last longer on a single battery cycle.

4.1.1 Microcontroller

The microcontroller is the most crucial part of the robot, since it makes sure that the robot receives instructions from the external processor, and that the external processor receives the measurement data from the sensors to apply calculations on. The main problem of finding the right microcontroller, is the huge arsenal of microcontrollers, where you have to pick one from. The one that has been chosen for this project is the ESP32. The reason for this, is the fact that it can establish a Wifi- and bluetooth connection, but also because of the wide selection of GPIO pins, the high clock frequency of the CPU and its availability on the market. Table 1 shows all the relevant characteristics of some of the most popular microcontrollers.

Table 1: Hardware selection, Microcontroller

	ESP8266 [13]	ESP32 [14]	Raspberry Pi Zero W [15]	nRF52-DK [16]	nRF51-DK [17]
Wireless connections	802.11a/b/g/n	802.11a/b/g/n BLE	802.11a/b/g/n BLE	BLE bluetooth5.0	BLE bluetooth5.0
#GPIO	8	22	22	32	31
Current consumption					
WiFi TX	120mA	130mA	170mA		
WiFi RX	56mA	95mA	170mA		
BLE TX		190mA	160mA	9,2mA	12,8mA
BLE RX		95mA	160mA	9,2mA	14,5mA
Running processor	15mA	25mA	120mA	3,7mA	4,8mA
Idle Processor	8mA	20mA	100mA	2mA	2,6mA
Sleep	0,5mA	0,8mA	No Sleep	1,5µA	1,6µA
CPU					
RAM	64kB	64kB	512MB	64kB	64kB
Flash	512kB	2MB	MAX 64GB	512kB	512kB
Clock	80MHz	80MHz	1GHZ	64MHz	16MHz
Work voltage	2,5-3,6V	2,3V-3,3V	5V	1,8-3,6V	1,8-3,6V
Price	€6,23	€19,90	€11	€33,09	€33,15

4.1.2 Sensor

The most important criteria when picking a sensor are the sensitivity, the range so it can scan a room, the time it takes to do a measurement and its power usage. The most perfect sensor would be the optimal choice for all these criteria, but one has to take the price into consideration. The LIDAR-sensors are not used in this project due to their high data rate and price, which makes the choice more narrow. The chosen sensor is the VL53L0X, this because it's the most efficient sensor power wise while maintaining a high enough sensitivity and range for this project. Table 2 shows a brief summary of the most popular sensors for distance measuring

Table 2: Hardware selection, Sensor

	HC-SR04[18]	VL53L0X[19]	LIDAR Lite[20]	RPLidar [21]
Type	Ultrasonic	Laser	Laser	Laser
Protocol	Single wire	I²C	I²C	USB/UART
Accuracy	5%	3% - 6%	2,5%	1%
Range	0,05m – 4m	0,05m – 1,2m	0,05m- 8m	0,2m-6m
Measurement angle	Single point	Single point	Single point	360°
Measuring time	60ms	33ms	2ms	0,5ms
Current consumption	15ma	19ma	130mA	350mA
Work voltage	5V	2,6V-3,5V	4,75-6V	4,9-5,5V
Price	€3,30	€12,50	€115	€200

4.1.3 Actuators

The robot's movements are realised by an actuator, also called "a mover". The most important aspect of this actuator is its precision. It needs to make sure that the instructions received from the server are executed correctly and precise, otherwise there would be a confusion between the actual pose and the calculated pose, which would lead to incorrect calculations, since the previous pose is always used to calculate the next.

DC motor

DC motors are not precise enough for this application, simply because you can't deliver a current pulse accurate enough to perform a perfect movement. This can be solved by placing external sensors, but these are too expensive for this purpose. Another patch to this problem could be to measure the time the motor is spinning to estimate the robots movement distance, but this is very inaccurate.

Figure 25: DC motor [22]

Servo motor

A servo motor relies on a PWM signal to indicate to which orientation it needs to position itself. The motor will draw current from the power to drive the shaft to that location. It uses a position-sensing device to determine how long it has to drive the shaft for it to go to the right direction and place. The most important limitation of a servo engine is that it can't turn around infinitely, instead it has a limited rotation range, dependent on the type of servo motor. This makes it unusable for our application.[22]

Figure 26: Servo Motor [22]

Stepper motor
A stepper engine is a DC engine that allows you to control how many steps it moves. Every step is a predefined accurate angle, this allows you to adjust the robots position in a precise manner. Another advantage is the relatively low price and the availability. The disadvantage, however, is that this engine does not provide a lot of torque and can only deliver low speeds, but these limitations are of no importance to our application. This is also the type of actuator used for this robot.

Figure 27: Stepper motor

4.2 Practical implementation

4.2.1 The robot's practical implementation

This chapter will go through the design of the robot, and which components have been used to realise this. The image below show the robot in its full glory. The robot is equipped with sensors, to measure the environment, and engines, to move around. The sensors are located in the sensor head, as shown on the right image. This sensor head can rotate itself to get a better feel with the environment. There are in total three VL53L0X-sensors who can communicate separately through a I²C-bus with the microcontroller. When the robot boots up, all the sensors have the same address, to change the address of a sensor, one has to make its shutdown-pin low and specify a unique address. This has to be done for every sensor, so the microcontroller can ask every sensor separately for measurement data.

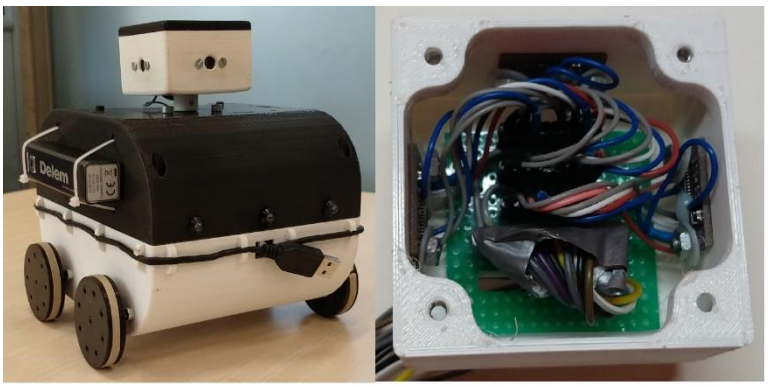

Figure 28: Mobile robot platform and the inside of the sensorhead

Before building the PCB, it first needs to be tested on a breadboard. The engine to control the robot is a stepper motor connected to an ESP32. But the GPIO's of the ESP32 cannot drive the stepper motor directly. Therefore a darlington array has been added. This is a device that allows a microcontroller to control the stepper motor, while the stepper motor receives its supply voltage directly from the power supply. Beside the low power, the ESP32 has another problem, too few connections. It has only 22 connections, while the robot requires a total of 25 connections, this problem has been countered by using a port expander. This adds another 8 outputs, used to control the engines. The components of choice are the ULN2003 as Darlington bridge and the 74HC595 as port expander. Figure 29 shows the ESP32 with engine and Darlington bridge, with the port expander located between the ESP32 and the Darlington bridge. Afterwards sensors have been added to the circuit.

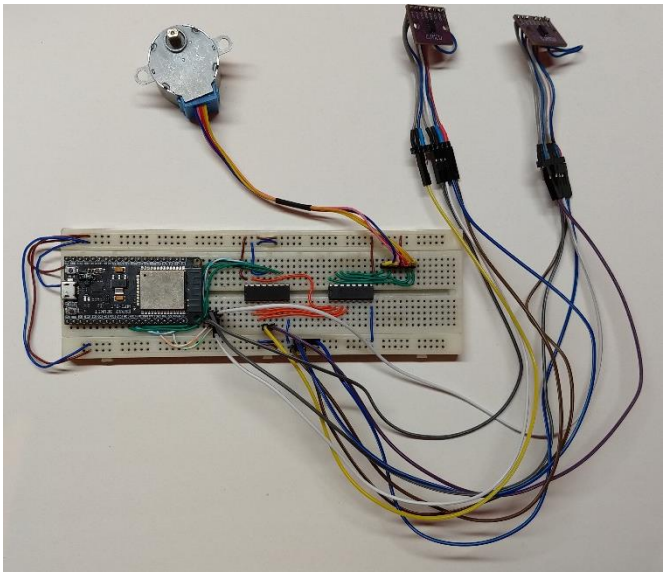

Figure 29: Proof of concept on a breadboard

4.2.2 PCB-design

This subchapter will show all the designed PCB's, starting at the heart of the robot, the microcontroller. The first PCB contains the ESP32, six connectors to port the GPIO pins to the components that need to be controlled and a connector to apply the power supply to. This last connector is equipped with a polarity protection, this protects the microcontroller when the supply lines have been accidently switched.

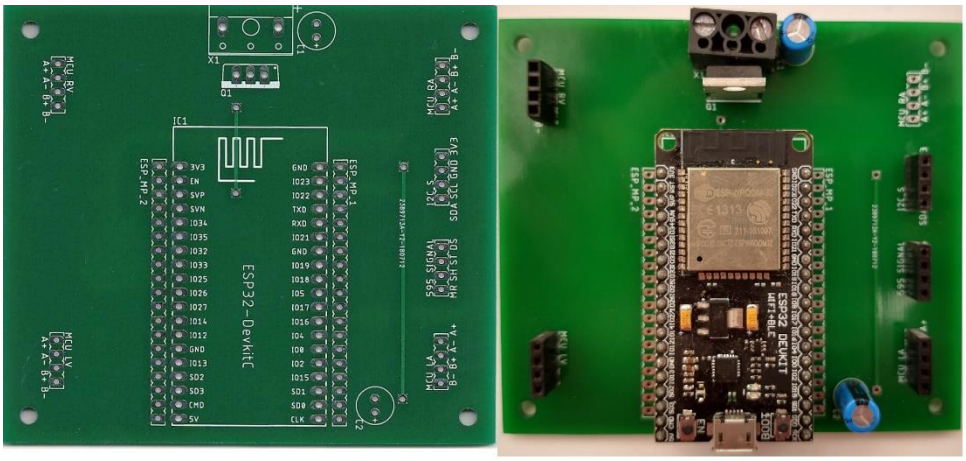

Figure 30: PCB microcontroller

Figure 31 shows the circuit for the engines. This PCB is equipped with four darlington arrays, to control and supply the engines, an external supply connector and connectors for the microcontroller and engines.

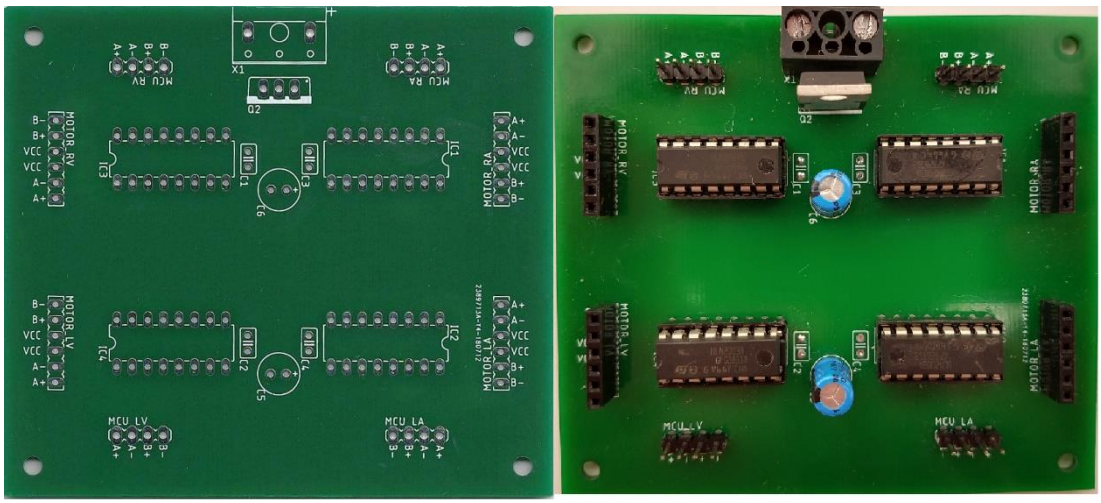

Figure 31: PCB steppermotor controller

Figure 32 shows the PCB for the port expanders. This PCB contains three 74HC595 IC's which serve as a port expander, every port expander a can serve as 8 additional outputs for the microcontroller. So this PCB adds 24 outputs to the microcontroller.

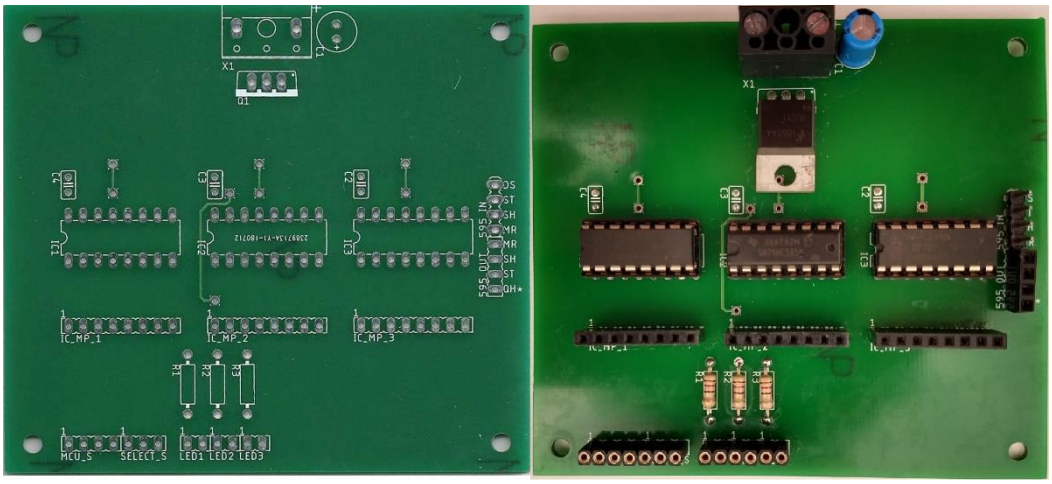

Figure 32: PCB port expander

Figure 33 shows the engine to control the head. On the PCB you can see the darlington array and a few connectors to serve as GPIO-pins and power supply.

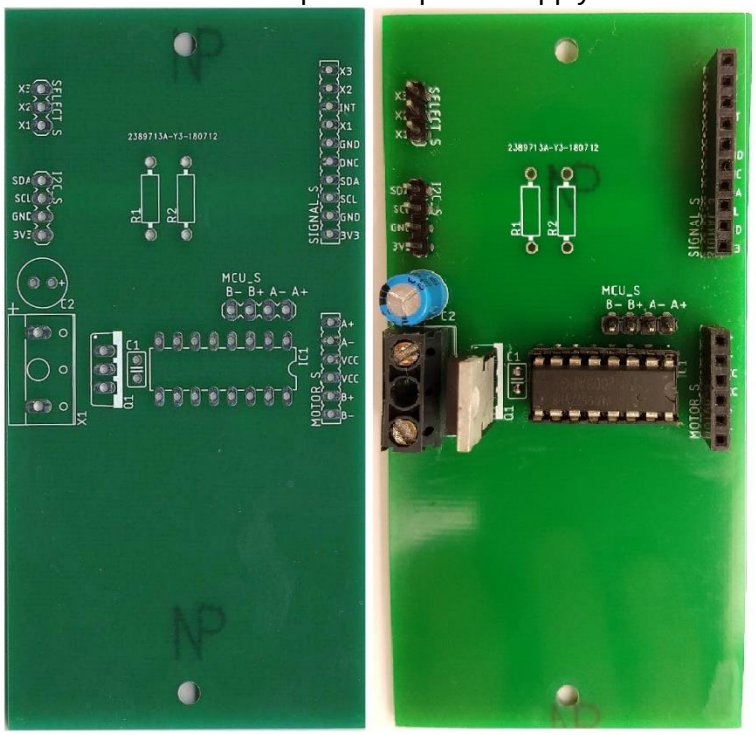

Figure 33: PCB sensor head

4.3 The robot's 3D-model

The hull of the robot exists solely out of 3D-printed parts. These parts have been designed in Autodesk Inventor. The designed models and how they fit together can be seen in Figure 34

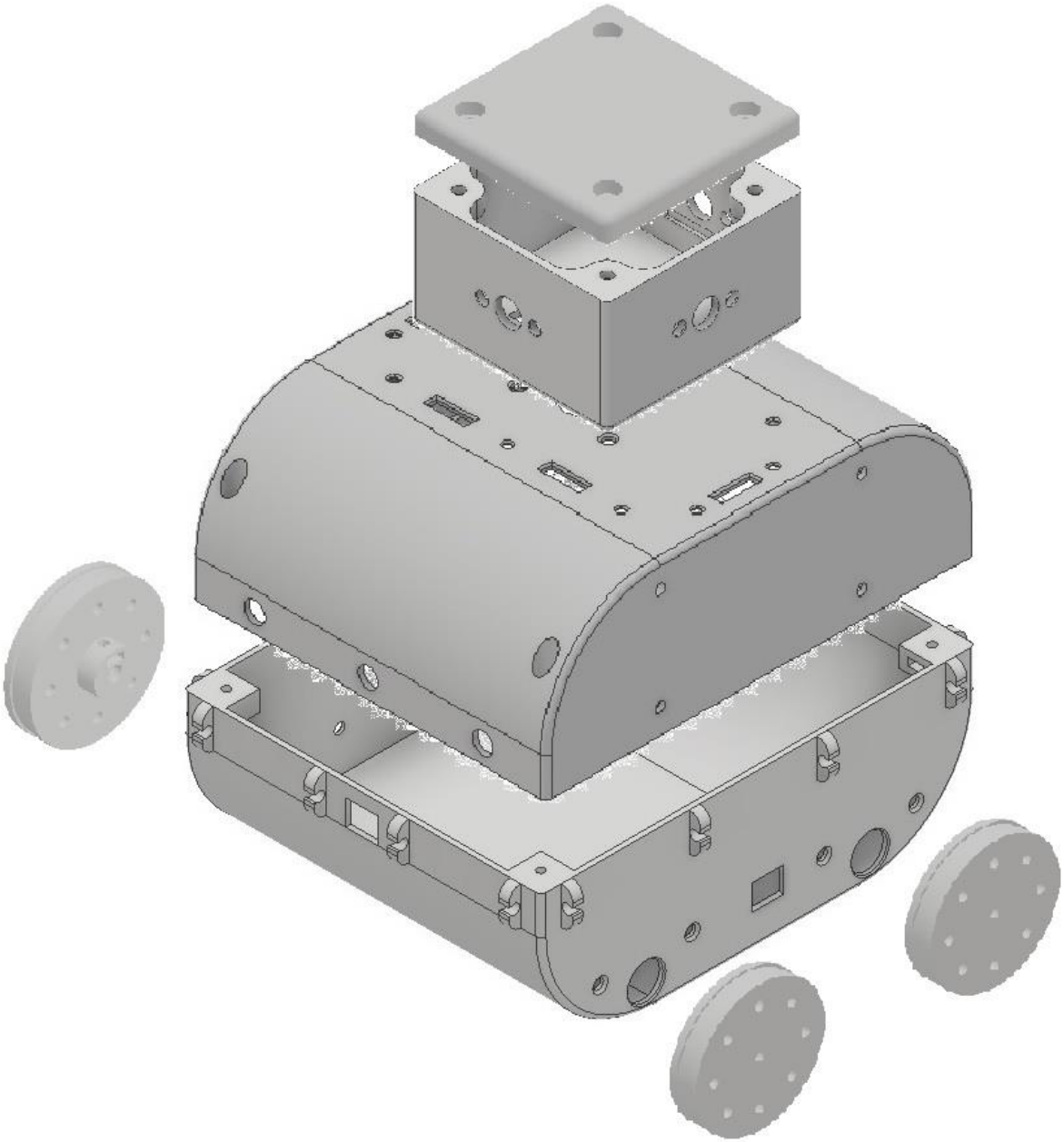

Figure 34: 3D design of the robot

Designing these models alone is not enough, they need to be 3D printed. To do this, one has to define each layer that has to be printed, this is done with a slicer. The software package that took care of this task was Simplify3D. A sliced model can be seen in Figure 35.

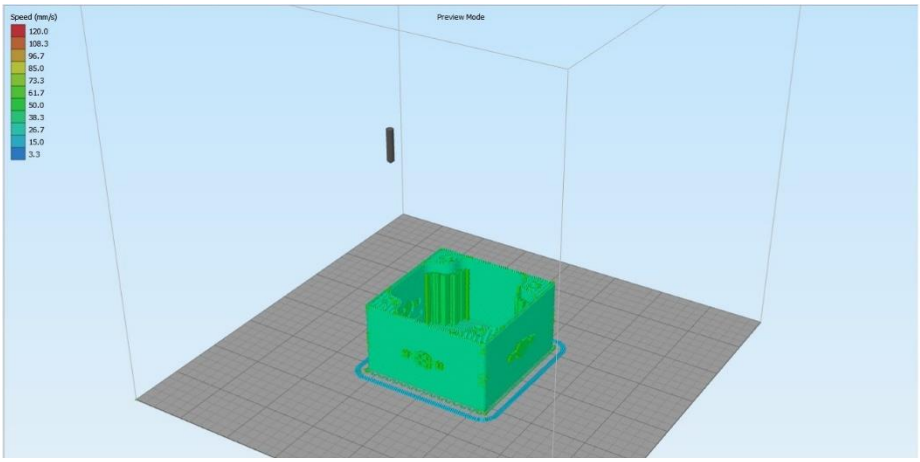

Figure 35: Sliced 3D model

Once all models have been sliced, the 3D printing can begin. The slicer's STL-files can be loaded in the 3D printer, which takes care of the final step, the printing. Figure 36 shows the 3D-printed robot in its full glory.

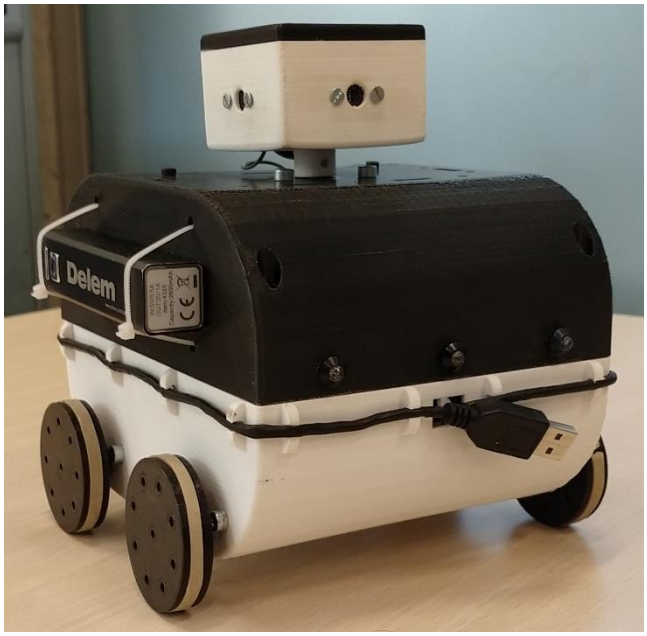

Figure 36: 3D printed robot

5 Realisation

The first thing that has been realised is the robot, this includes the design of PCBs as well as 3D - models. Before the PCBs could be designed, a proof of concept had to be made. This proof of concept was building the circuit and testing it. Once tested the PCB were designed and finally manufactured by JLC-PCB. Once the PCBs have been manufactured, and delivered, they were tested for quality and completeness, to confirm that all connections were intact. The next step was to manually solder all the components and test if they could communicate with one another.
Once the PCBs have been realised, a hull had to be made to protect the PCBs and to create a presentable package. How they were printed, is explained in 4.3 The robot's 3D-model. The material that has been used is PETG, this material is less brittle than the more popular materials PLA and ABS.

The robot requires an external processor to do its calculations, this means that there needs to be a form of communication between the robot and said processor. This communication is done over a wireless network. Initially this communication was realised by setting up a TCP/IP-server in Python. This has later on been exchanged by a ESP32 module, given the fact that not every network allows TCP/IP-communication, and not every PC-network card can set-up its own WiFi-network. The ESP32-module can set-up its own WiFi-network to which the robot can connect. The ESP32-module serves now as a server and communicates wirelessly over TCP/IP with the robot. The communication between the ESP32-module and the PC, or external processor, has been done over UART. The image below shows on the left side the connection to the computer, and on the right the UART connection of the ESP-32.

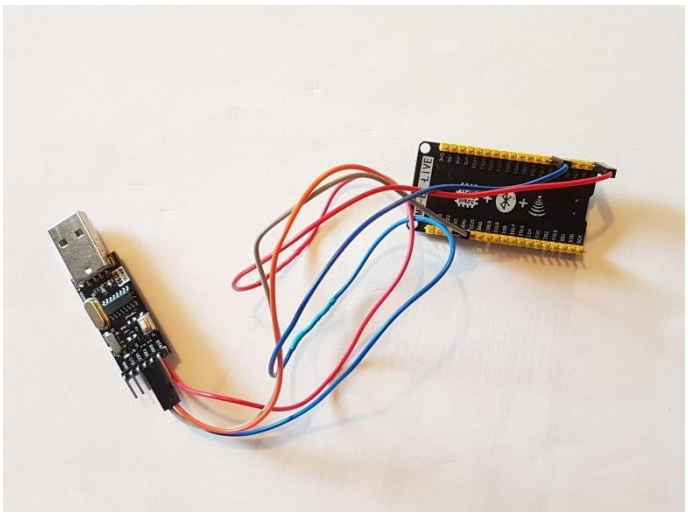

Figure 37: USB to UART dongle, to connect the ESP server to the PC

The robot has four basic movements, that can be communicating through TCP/IP in the following manner host/member/action/value:

- Robot movement → host/robot/move/$DISTANCE
- Robot rotation → host/robot/turn/$ANGLE
- Sensor rotation → host/sensor/rotate/$ANGLE
- Sensor measurement → host/sensor/measure/#sensor

Here $ stands for the value of a variable, and # for an identifying integer number. This message protocol has been tested with Terminal, a software package to communicate over COM-ports.

The robot is now complete and can be used to localise itself. The first step in doing this, was to create a map of the environment. This thesis has chosen for a symmetric map, since it's harder for the robot to localise itself. The following image on the left shows the designed map in Matlab. One this image you can see that the robot has to move to a specific location in order to be able to localise itself, this location is behind the two horizontal walls. Later on the environment has been build out of wood, to test the algorithm in real life, this can be seen on the right side.

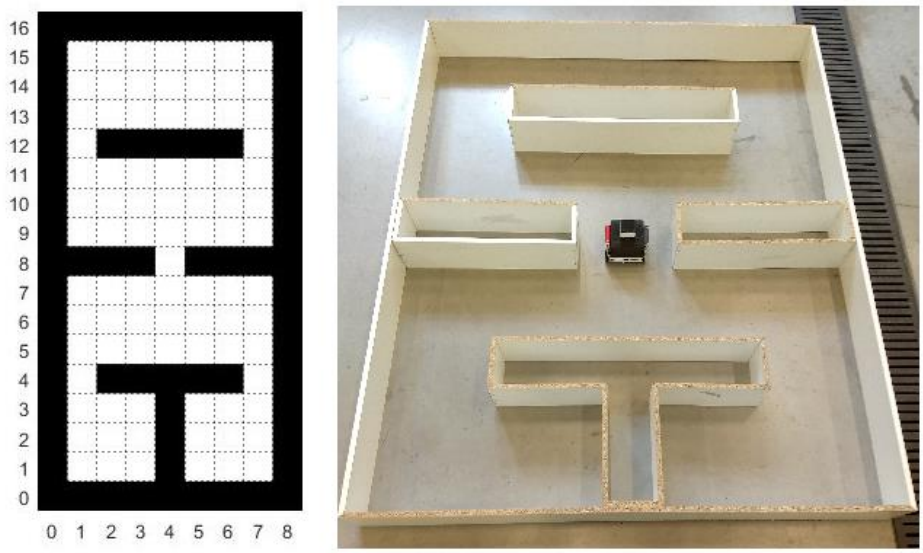

Figure 38: Creating a testing environment for the algorithm

The passive localisation algorithm applied to the robot is based on code found online. [12] This code has been debugged line by line, and multiple fundamentally wrong parts have been adjusted to reach a working passive localisation algorithm. This code has been tested by injecting self-generated measurement- and movement data. The logic and operations within the code has been visualised for educational purposes, as seen in the following figures. The image on the left shows the simulated map, the robot's location within the map. The right image shows the probability of the robot being in a certain location, the blue square is where the algorithm thinks the robot is located, which is correct. The other squares represent other possible locations, with a low certainty because an algorithm is never 100% sure of its predictions. The passive localisation algorithm works but is not tested in a real environment due to time constraints.

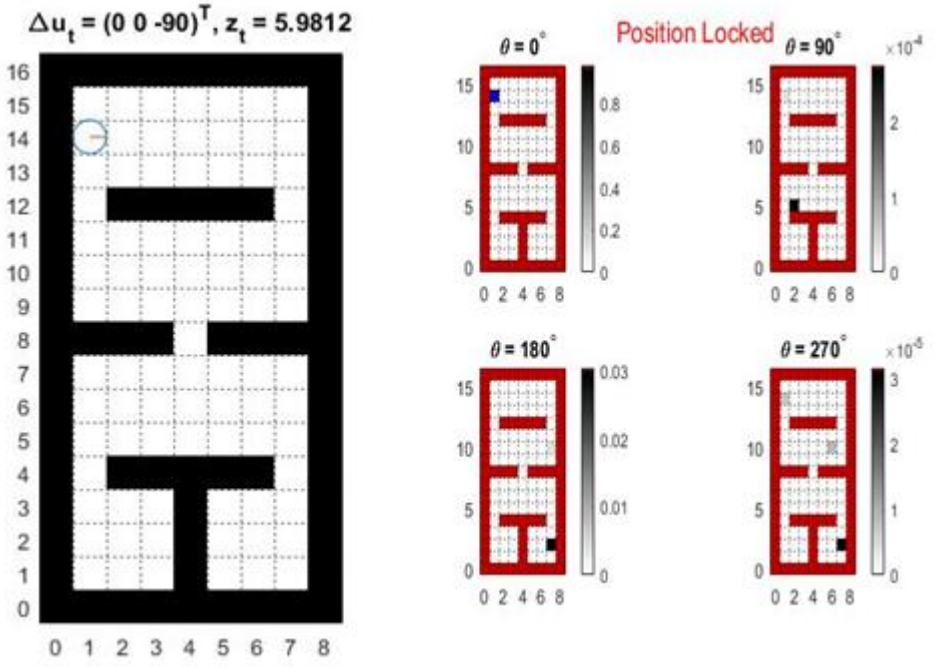

Figure 39: Passive Markov localisation with generated measurement and movement data

The active localisation is not yet implemented due to time constraints. There is something fundamentally wrong with the calculation of the expected entropy, which is fully explained in 2.4.2 Active localization .

6 Conclusion

Building a fully working mobile robot which can localise itself within a room, is no easy feat. First the robot has to be built and tested before it can attempt to localise itself.

This thesis realised to build a robot that can localise itself through passive localisation and communicate over a network. The main bottleneck, and simultaneously the most educational one, of this thesis was the lack of experience in the given subjects. The network protocol, together with the building and testing of the robot also defined a big part of this thesis, leaving less time for the localisation algorithms.

All by all, there has been a huge learning curve and interest in the subject, leaving room open for future work that will be tackled by myself. This future work includes the building of a 3D-robot, but based on a building kit, to minimise the 3D-printing hours to a limited number of 3D-objects that need to be refurbished. Another task would be to debug the active localisation algorithm and to test it in real life, together with multiple passive localisation algorithms, namely the Markov- and Monte Carlo algorithm. Finally, some additional efforts must be taken to make this project more educational friendly, including but not limited to the writing of high level classes.

References

[1] ACRO, “About – Acro,” 2018. [Online]. Available: https://iiw.kuleuven.be/onderzoek/acro/about.

[2] KU Leuven, “Ad Usum Navigantium,” 2017. [Online]. Available: http://www.adusumnavigantium.com/pageen.html.

[3] Arduino, “Arduino - Software,” 2018. [Online]. Available: https://www.arduino.cc/en/Main/Software.

[4] Autodesk, “Inventor | Werktuigbouwkundig ontwerp- en 3D CAD-software | Autodesk,” 2018. [Online]. Available: https://www.autodesk.be/nl/products/inventor/overview.

[5] Autodesk, “EAGLE | PCB Design Software | Autodesk,” 2018. [Online]. Available: https://www.autodesk.com/products/eagle/overview.

[6] JLCPCB, “PCB Prototype & PCB Fabrication Manufacturer - JLCPCB,” 2018. [Online]. Available: https://jlcpcb.com/.

[7] I. Nourbakhsh and R. Siegwart, “Mobile Robot Localization,” in *Autonomous mobile robots*, pp. 159–227.

[8] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. 1999.

[9] D. Fox, W. Burgard, and S. Thrun, “Active Markov Localization for Mobile Robots,” 1998. [Online]. Available: ttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.3850&rep=rep1&type=pdf.

[10] A. Patel, “Introduction to A*,” *1997*. [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#the-a-star-algorithm.

[11] GeeksforGeeks, “A* Search Algorithm.” [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/.

[12] P. Lusk, “grid-localization,” 2016. [Online]. Available: https://github.com/plusk01/grid-localization.

[13] Espressif, “ESP8266EX datasheet,” 2018. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf.

[14] Espressif, “ESP32 Datasheet,” 2018. [Online]. Available: www.espressif.com/en/subscribe.

[15] Broadcom, “Raspberry Pi Zero W SoC datasheet,” 2012. [Online]. Available: https://www.raspberrypi.org/app/uploads/2012/02/BCM2835-ARM-Peripherals.pdf.

[16] Nordic Semiconductors, “nRF52832 Development Kit v1.1.x User Guide v1.2,” 2017. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF52_DK_User_Guide_v1.2.pdf.

[17] Nordic Semiconductors, “nRF51832 Development Kit v1.1.x User Guide v1.2,” 2014. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF51_DK_UG_v1.1.pdf.

[18] ELEC Freaks, “Ultrasonic Ranging Module HC-SR04 datasheet.” [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.

[19] ST Microelectronics, “VL53L0X datasheet,” 2018. [Online]. Available: https://www.st.com/resource/en/datasheet/vl53l0x.pdf.

[20] PulsedLight Inc., “LIDAR lite v3 datasheet.” [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/LIDAR-Lite-Data-Sheet.pdf.

[21] SLAMTEC, “RpLIDAR a1m8 datasheet,” 2009. [Online]. Available: https://www.robotshop.com/media/files/pdf/rplidar-a1m8-360-degree-laser-scanner-development-kit-datasheet-1.pdf.

[22] HandyBoard, “What is the difference between a DC motor and servo motor?” [Online]. Available: http://handyboard.com/hb/faq/hardware-faqs/dc-vs-servo/.

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Implementation of active Markov localisation on a mobile robot for didactic purposes

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2018**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Grommen, Sander

Datum: **20/08/2018**