UNIVERSITY OF SOUTHERN DENMARK

COURSE PROJECT

ROBOTICS 5TH SEMESTER - FALL 2018

Marble finding robot

Alexander Tubæk Rasmussen
alras16@student.sdu.dk

Kenni Nielsen
kenil16@student.sdu.dk

Marcus Enghoff Hesselberg Grove
grov16@student.sdu.dk

1 Abstract

Contents

2 Introduction

Mobile robots are an increasingly larger part of our world. This increases the demand for algorithms that enables the robot to navigate around in its environment, and effectively search the environment. This sets focus on creating planning algorithms and using reinforcement learning to optimise different problems.

In this report a mobile robot platform will be given the task of navigating an environment and collect marbles. While optimising its strategy for each run. The robot will be fitted with a camera, and a lidar scanner.
To accomplish this algorithms must be written to enable the robot to navigate the environment. To do this marble detecting and obstacle detecting algorithms must be written, so the robot would be able to localise the marbles and circumnavigate the obstacles.
To ensure that the robot will learn from its previous trials, and optimisation algorithm based on reinforcement learning must be written.

2.1 Problem statement

The overall problem was to design and implement algorithms that would enable the robot platform to navigate an environment, collect marbles, circumnavigate obstacles as well as optimise its strategy for each trial.
This should be done utilising the knowledge of reinforcement learning, computer vision and robot motion theory.

2.1.1 Problems

The following problems are stated to better describe the focus points throughout the project. XXXX

1. How can an algorithm, which can find marbles in a closed map, be designed?
2. How can this algorithm be optimized, so the robot can find the ball faster, based on previous trails?
3. How can marbles and obstacles be detected from lidar data?
4. How can marbles and obstacles be detected from camera data?
5. How can a Fuzzy-control algorithm be constructed?
6. How can computer vision be used to detect marbles and obstacles, and to construct the map?

XXXX

2.1.2 Limitations

The robot must be able to find marbles in a closed map and optimize itself based on previous trails. Thus the following limitations to the algorithms is listed:

1. To detect marbles and obstacles.
2. To construct a map.
3. To optimize for the shortest distance and time.

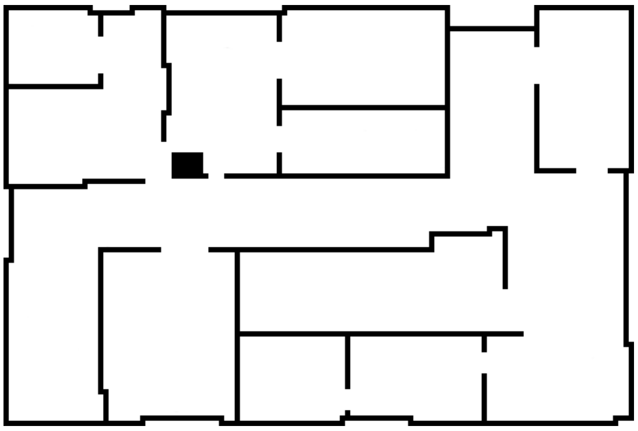
QT Creator, Fuzzy Lite, Gazebo and Matlab will be used throughout the project.

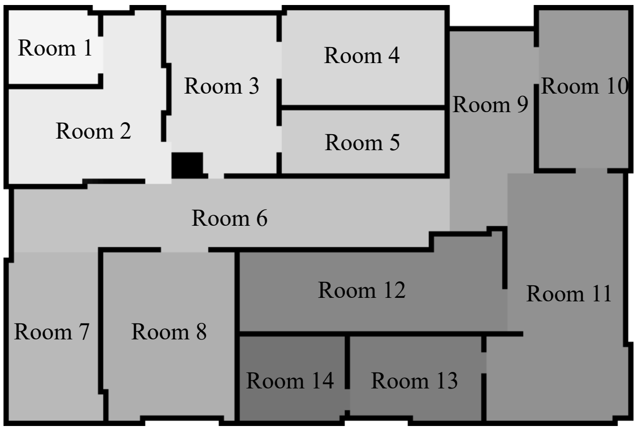
2.2 Readers guide

3 Design

3.1 Environment

The two-wheeled robot should navigate around the environment called “bigworld” shown on figure **??**. To do so, it have been chosen to divide the environment into rooms. The natural definition of *’rooms’* have been taken into account, resulting in a 14 room layout. This can be seen on figure **??**, where each room can be distinguished by different grey scale colours.


(a) Illustration of the world ”bigworld”



(b) ”bigworld” divided into 14 rooms

Figure 1: Illustration of the world ”bigworld” before and after it has been divided into 14 rooms

The division into rooms are useful in terms of knowing which rooms that have been searched and which that not yet have been searched for marbles.
It is also useful as an abstract state space for reinforcement learning such that it can be found which order the rooms should be visited.

3.2 Lidar Scanner

The two-wheeled robot given for this project is among other equipped with a 2d lidar (Light Detecting and Ranging) scanner. A lidar scanner detects the distance to targets by emitting a laser pulse and analyzing the time it takes for the beam to reflect and return to its source. The lidar scanner maps the environment and has a detecting range of 10 pixels in Gazebo and a 260 degrees field of view.

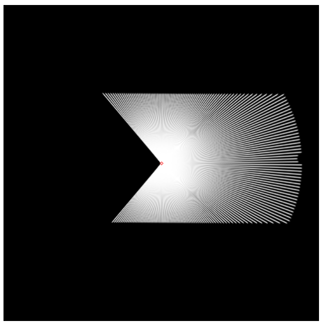
The script converts this range into mm by first scaling the pixel map to double size, such that the detecting range is 20 pixels. Afterwards, the script trace the pixel map to a eps-figure using 72 dpi as standard for the postscript. Then, the script converts this range from inches to mm by using a conversion factor of 25.4 mm per inch. This means that this range can be converted into mm by using the following formula:

$$\frac{20\, pixels}{72\, dpi} \cdot 25.4\, \frac{mm}{inch} = 7.06\, mm$$

Now the script scales the world from mm to m, which means that the range of the lidar scanner therefore is 7.06 m.
The lidar scanner maps the surrounding environment by collecting 200 datapoint, which must be processed in order to recognize objects such as walls and marbles. This means that circle and line detecting algorithms must be writing.

The datapoints from the lidar scanner is first visualized by drawing white lines from the robot’s location to each of the 200 datapoints using the `cv::line()` function as shown on figure **??**. Afterwards the datapoints are sorted by checking if the range is equal to max detecting range in order to get the points, that reflects from different object.

The filtered datapoints are drawn as lines upon the unsorted data. These lines can be distinguished by different blue colours depending on the range between the two points ranging from blue to cyan. This blue coloured and white lines are shown on figure **??**.


(a) Illustration of the unfiltered data

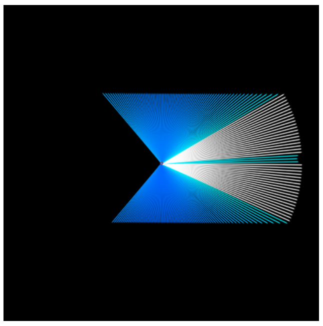

(b) Illustration of the filtered data

Figure 2: Illustration of the unfiltered and filtered data

The two sections below, describes the design of a line and a marble detection algorithm.

3.2.1 Line detection

It is usually important for a mobile robot to know its environment. There are several reasons for that, one is that robot must know the location of the obstacles (walls) relative to it to avoid driving into them. In the "bigworld" environment, the obstacles is walls which can be represented as lines from the filtered datapoints. These lines can be represented using a normal parametrization in polar coordinates, given by the following formula:

$$l : \quad r = x \cdot \cos(\alpha) + y \cdot \sin(\alpha) \tag{3.1}$$

where r represents the distance from the origin to the closest point on the line and α is the angle between the x-axis and the plane normal.

The Total Least Square method assumes that a line can be represented as in equation **??**. This method also involves a determination of the orthogonal distance (the shortest distance) from a point p_i to a line l as shown on figure **??**. The normal parametrization of the line l_i is given by the following formula:

$$l_i : \quad r_i = x_i \cdot \cos(\alpha) + y_i \cdot \sin(\alpha) \tag{3.2}$$

The separation between those two lines (l and l_i) is given by the difference $d_i = r_i - r$, since both lines have the same α. This means that the orthogonal distance can be described using the following formula:

$$d_i = x_i \cdot \cos(\alpha) + y_i \cdot \sin(\alpha) - r \tag{3.3}$$

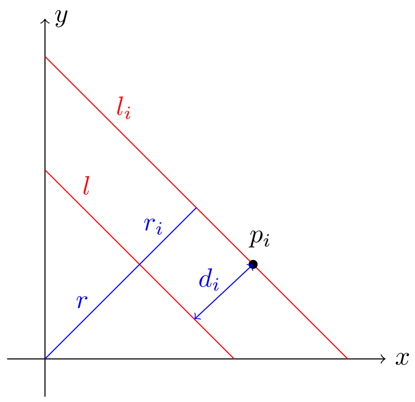


Figure 3: Orthogonal distance from point p_1 to line l

This only applies if we assume that there is no noise on the measurements.

This method gives an solution to the problem of fitting a straight line to a dataset of points p with n measurements having errors. The problem of fitting a line can be determined using the following sum:

$$\chi^2 (l, z_1, ..., z_n) = \sum_{i=1}^{n} \left[\frac{(x_k - X_k)^2}{u_{x,k}^2} + \frac{(y_k - Y_k)^2}{u_{y,k}^2} \right] \tag{3.4}$$

where (x_k, y_k) are the points coordinates with corresponding uncertainties $(u_{x,k}, u_{y,k})$ and (X_k, Y_k) denote its corresponding point of the straight line l. In the case of fitting the best line to the dataset, minimizes the expression for χ^2 by setting $u_{x,k} = u_{y,k} = \sigma$ and $k = 1, ..., n$. This reduces the problem to the Total Least Square method and minimizing is equal to minimizing the orthogonal distance of the measurements to the fitting line. Therefore, in the case of fitting the best line minimizes the expression above to the following:

$$\chi^2(l; Z) = \sum_{i=1}^{n} \frac{d_i^2}{\sigma^2} \tag{3.5}$$

$$= \sum_{i=1}^{n} \frac{(x_i \cos(\alpha) + y_i \sin(\alpha) - r)^2}{\sigma^2} \tag{3.6}$$

$$= \frac{1}{\sigma^2} \cdot \sum_{i=1}^{n} (x_i \cos(\alpha) + y_i \sin(\alpha) - r)^2 \tag{3.7}$$

A condition for minimizing χ^2 is done by solving the nonlinear equation system with respect to each of the two line parameters (r and α)

$$\frac{\partial \chi^2}{\partial r} = 0 \qquad \frac{\partial \chi^2}{\partial \alpha} = 0 \tag{3.8}$$

The solution of this nonlinear equation system is determined to the following:

$$r = \overline{X} \cos(\alpha) + \overline{Y} \sin(\alpha) \tag{3.9}$$

$$\alpha = \frac{1}{2} \arctan \left(\frac{-2 \sum_{i=1}^{n} \left[(x_i - \overline{X}) - (y_i - \overline{Y}) \right]}{\sum_{i=1}^{n} \left[(x_i - \overline{X})^2 - (y_i - \overline{Y})^2 \right]} \right) \tag{3.10}$$

where $\overline{x}$ and $\overline{y}$ are the means of x and y.

As explained earlier, the two-wheeled robot should avoid obstacles (walls). To do so, the robot needs to know the location of the walls. It is done by processing the datapoints from the lidar scanner and determining the points which fits to a straight line using a line extraction algorithm. There are several different line extraction algorithms to choose from. The incremental line extraction algorithm is chosen, because it is simple to implement. The pseudo code for the incremental algorithm is shown below.

Algorithm: Incremental

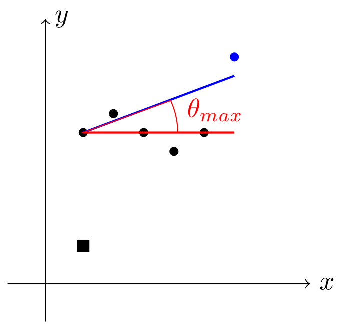
```
1. Start by the first 2 points, construct a line
2. Add the next point to the current line model
3. Recompute the line parameters
4. If it satisfies line condition (go to 2)
5. Otherwise, put back the last point, recompute the line parameters, return the line
6. Continue to the next 2 points, go to 2
```

This algorithm implements the Total Least Square method then computing the line parameters. Furthermore, the line model consists of points, which all must comply some line conditions. In that case all points must comply these conditions. The first condition is a threshold for the angle between the previous and current line model as shown on figure **??**. The threshold is defined to be the following:

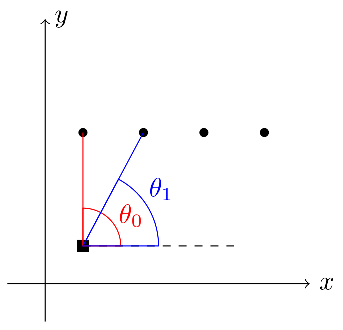
$$\theta_{max} = 0,0025$$

The second condition is the angle between two points relative to the robot location as shown on figure **??**. This angle should be greater than this difference, but not twice as great, since this condition should separate the points into two lines, if one point is missing on the list as shown on figure **??**. This angle is therefore defined to be the following:

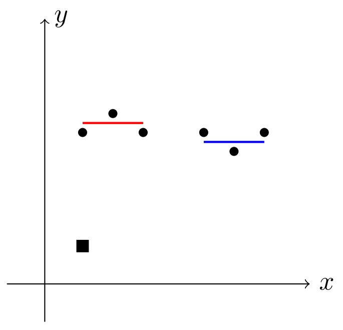
$$\Delta\theta = (\theta_0 - \theta_1) \cdot 1,25$$



(a) Illustration of the angle between current line model (blue line) and previous line model (red line)



(b) Illustration of the angle from horisontal to datapoint relative to the robot



(c) Illustration of the separation of to lines due to a missing datapoint

Figure 4: Illustration of the angle between two datapoints and two line models

3.2.2 Marble detection

The objective for the two-wheeled robot is to collect marbles effectively, while avoiding obstacles such as walls. The Hough transform is a algorithm, that maps from image space to the probability of the existence of features such as lines, circles or other general shapes. The algorithm is also capable of detecting partial object as in the case of the lidar scanner. However the algorithm are not very robust to noise, and can be very hard to calibrate in order to make the result from the algorithm useful. Furthermore the algorithm have a high computational cost.

Because of this, it have been chosen to design a simpler method for finding marbles in the lidar data.
Due to the fact that only a part of the circles would be visible from the data, it was chosen to calculate the center and radius of the circles from the chord and arch height of the circles. This can be used to determine the location of the marbles relative to the robot. Given this information the planner would be able to drive towards the marbles and "collect" them.
It is assumed that the two detected outer points on the circle periphery defines the circle chord. It is also assumed that the orthogonal distance from the detected middle point on the circle periphery to the circle chord defines the circle arch height. This only applies for an uneven number of detected point.
The circle chord can be determined using the formula:

$$C = 2r \cdot \sin\left(\frac{\theta}{2}\right)$$

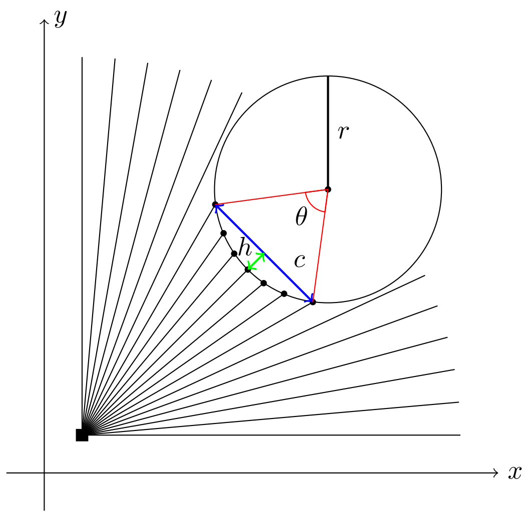
The circle camber can be determined using the formula:

$$h = r\left(1 - \cos\left(\frac{\theta}{2}\right)\right)$$

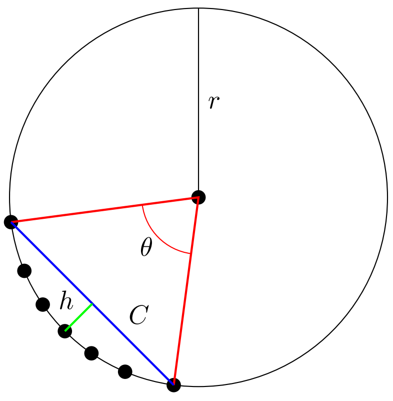
Solving this equation system, the circle radius was found to be:

$$r = \frac{K^2 + 4h^2}{8h}$$

The polar coordinates of the circle center can be found by adding the radius to the range of the detected middle point.



(a) Illustration of detected points (marked as ending point of the black lines), circle chord and arch heigh can be found from the lidar data



(b) Illustration of circle chord and arch heigh can be found from the lidar data og thereby can determine the circle's radius and the chord angle

Figure 5: Illustration of ...

3.3 Tangent bug

The tangent bug algorithm is a sensor based planer which relies on inputs from a sensor to determine whether it should continue towards a specified location q_{goal} or to follow an obstacle O_i until a free path is available. The advantage of using this algorithm is the computational minimization and the fact that it only needs a start and end point. The reason for using this algorithm is the benefits just explained and that it will always try to achieve a shortest path solution to the goal location. Pseudo code for the implementation for the tangent bug algorithm is shown below.

Algorithm: Tangent Bug Algorithm

```
Input: A robot with a range sensor
Output: A path to the $q_{goal}$ or a conclusion no such path exist
    while True do
        repeat
            Continuously move toward the point $n \epsilon (T, O_i)$ which minimizes $d(x, n) + d(n, q_{goal})$
            until
            • The goal is encountered or
            • The direction that minimizes $d(x,n)$ +d$(n, q_{goal})$ begins to increase d$(x, q_{goal})$ so the robot
              detects a local minimum d$(x, O_i)$ + d$(O_i, q_{goal})$ on the boundary $i$.

            Now choose the boundary following direction which continuous in the same direction as the most
            recent motion-to-goal direction.
            repeat
            Continuously update $d_{reach}$, $d_{followed}$ and $O_i$.
            Continuously moves toward n $\epsilon$ $\{O_i\}$ that is the chosen boundary direction.
            until
            • The goal is reached
            • The robot completes a cycle around the obstacle in which case the goal cannot be reached.
            • $d_{reach} < d_{followed}$
    end while
```

It has been decided to make small changes to the algorithm and make it greedy. Instead of following the boundary $d_{reach} < d_{followed}$, the robot will follow that obstacle that minimizes $d(x, n)$ + d$(n,\ q_{goal})$. The reason for doing this is to reduce the path from a giving point to a target location.

3.4 Model based planner

The brushfire algorithm uses a grid to approximate distance to obstacles. The idea is to give obstacles a starting value of 1 and free-space pixels a value of 0. Then continue until the $'fire'$ has consumed all free pixels thereby giving pixels furthest away from obstacles the highest value. It was decided to use a eight-point connectivity grid. Pseudo code for the implementation of the brushfire algorithm is giving below.

Algorithm: Brushfire algorithm

```
while True do
label++
    for all i hight of image
        for all j width of image
        if adjacent pixel values to image($i, j$) is label & image($i, j$) is 0 set image($i, j$) to label + 1
    until
      • All pixels have an assigned value
end while
```

The reason for choosing this algorithm is that it gives a set of values to the pixels furthest away from the obstacles, which can be used to generate a path for the robot by picking out the pixels that leads to a complete road map for the robot to navigate through the entire map.

By using the brushfire algorithm as described above it is possible to get information about certain points of the map with the highest value. Now by looking at the corners and center lines generated by the brushfire algorithm one can get the desired values. The idea here is the use a 4-point connectivity grid where each $pixel(i, j)$ value is compared to its neighbours. For instance if $pixel(i - 1, j)$ equals $pixel(i + 1, j)$ and the center point between them, $pixel(i, j)$, has a different value, it is considered to be a point on a line. The same thing goes for the horizontal case. To find corner points, the $pixel(i-1, j-1)$ must equal $pixel(i - 1, j + 1)$ and the point $pixel(i, j)$ must then be different from those two points.

Algorithm: FindCenterPoints

```
for i hight of image
    for all j width of image
    if $pixel(i - 1, j)$ & pixel$pixel(i + 1, j)$ & $pixel(i + 1, j) \neq pixel(i, j)$
    push vector on list of center points
```

Now one can assign a *weight* to the list of corner and center points so one can differentiate the wanted points from others. After doing this the newly created map will contain the edges and center lines which is interesting for the creation of the path. It was decided to use the center of these found lines to use for the creation of the road map.

Algorithm: FindLinePoints (Horizontal case)

```
for i hight of image
    for all j width of image
    while True do
    push $pixel(i,j)$ on vector
    until
      • $pixel(i,j) \neq weight$
    end while
    if size of vector > 2
    push the middle $pixel(i,j)$ of vector on list of wanted points
```

Now in the creation of the road map every point is considered to be a vertex and a connection between two vertexes is an edge. Now the interesting part is to find out which vertexes that can be connected to one another without hitting an obstacle. Because all obstacles have been assigned the value of 1 it is possible to iterate through the map and see if $pixel(i,j) == 1$ is located somewhere on the edge between these two vertexes. If that is not the case, the edge is considered valid and saved as an edge and thereby a possible connection between two vertexes.

Algorithm: Connected vertexes

```
for i hight of image
    for all j width of image
    if line between V1 and V2 is not on an obstacle
    push edge on vector
```

Algorithm: Remove duplicates

```
for all edges
if $edge(V1,V2) == edge(V2,V1)$
Remove one of them
```

Now the edges have to be sorted by distance with shortest distance as the first element on the list. This is done so that Kruskal's algorithm can be used to connect all the edges on the list and make a complete connection between all vertexes. Another benefit of using Kruskal's algorithm is to avoid cycles in the graph and thereby making the implementation of finding a path from an initial position to a target location easier. Kruskal's algorithm uses the *disjoint set union/find* algorithm which is an algorithm used to find relations between vertexes. It starts by initializing a vector by the size of the number of edges and sets them to -1. The *unionSets()* connects two vertexes if there connection will not result in a cycle. The *find()* method uses recursion to see if the vertexes are joint or disjoint, meaning that they form a cycle if they are connected. Pseudo code for both Kruskal's and union/find algorithm can be seen below.

Algorithm: Kruskal's algorithm

```
for all edges
    integer V1 = find(edge.V1)
    integer V2 = find(edge.V2)
    if V1 $\neq$ V2
    push edge on vector
    unionSets(V1,V2)
```

Algorithm: Find

```
Input: A vertex
Output: The set containing the vertex

if vector(vertex) < 0
return vertex
else
return find(vector(vertex))
```

Algorithm: UnionSet

```
Input: V1 and V2
Output: The set containing the vertex

vector(V1) = V2
```

Now the idea of generating connections between all vertexes on the map is complete. The last thing which needs to be solved is to be able to generate a path between vertexes from an initial position to a target location. It must be known which vertexes can be connected to one another. This will give a list of adjacent vertexes to a specific vertex.

Algorithm: Adjacent vertexes

```
for all vertexes
    for all edges
    if vertex == edge.V1
    push edge.V2 to vector of vectors
    else if vertex == edge.V2
    push edge.V1 to vector of vectors
```

The final step in getting a path from any of the vertexes to any other vertex is using an extended version of the *Depth-First Search* algorithm. This algorithm uses recursion to find a path between vertexes. It uses the newly found vector of vector (vertexes) to recursely generating a full path from a start vertex to target vertex.

Algorithm: DFS extended

```
Input: Start and target location
Output: Vector of vertexes in order
push start on vector
mark start as visited

if start == target
return vector

for all vertexes adjacent to start
if adjacent == target
push adjacent on vector
return vector

else if adjacent ≠ visited
DFS(adjacent,target)
pop last element from vector
```

Thus the user should be able to give a start and target location from the number of vertexes from the list yielding a complete route for the robot to travel.

3.5 Search strategy

For the robot to be able to avoid obstacles as well as navigating through the map Fuzzy Control will be used. For short the a fuzzy controller consist of a fuzzification interface that converts input into information that can be used by the inference mechanism. The inference mechanism evaluates the giving input in addition to the rule base based on the expert's linguistic description. The output of the fuzzy controller will then be defuzzified to crisp values which will be used as input to control the plant. The following linguistic terms will be used:

- The linguistic input variable called $ObstacleDirection = \{right, center, left\}$ with the named linguistic values. The universe of discourse is set to $U = [-1.6, 1.6]$. It defines the direction towards and obstacle and the choose of U is based on the angle range from the sensor.
- The linguistic input variable called $obstacleFree = \{right, center, left\}$ with the named linguistic values. The universe of discourse is set to $U = [-1.6, 1.6]$. It defines the angle to which and obstacle is furthest away from the robot. This is used when the robot is driving towards a corner and has to avoid collision. The choose of U is based on the angle range from the sensor.
- The linguistic input variable called $ObstacleDistance = \{veryclose, close, far\}$ with the named linguistic values. The universe of discourse is set to $U = [0, 10]$. It defines the distance to an obstacle and the chose of U is based on the sensors maximum detection range.
- The linguistic input variable called $MarbleDirection = \{right, center, left\}$ with the named linguistic values. The universe of discourse is set to $U = [-30, 30]$. It defines the direction from the robot's point of view as a changes in pixel values in the picture from the camera placed on the robot. The universe of discourse was found as a suitable deviation from the center point.
- The linguistic variable called $MarbleFound = \{no, yes\}$ with the named linguistic values. The universe of discourse is set to $U = [0, 50]$. It defines if an marble is detected where the input is a radius of the marble on the picture from the camera on the robot. The chose of U was found suitable.
- The linguistic variable called $GoalDirection = \{right, straight, left\}$ with the named linguistic variables. The

universe of discourse is set to $U = [-3.14, 3.14]$. It defines the direction in which a target location is located. The chose of U is based on a complete rotation from the robot's point of view.

- The linguistic input variable $BoundaryDirection = \{right, straight, left\}$ with the named linguistic values. The universe of discourse is set to $U = [-3.14, 3.14]$. It defines boundary direction on an obstacle in which the robot has to follow if it is in an obstacle following behaviour. The chose of U is based on a complete rotation from the robot's point of view.
- The linguistic output variable called $SteerDirection = \{sharpright, right, softright, straight, softleft, left, sharpleft\}$ with the named linguistic values. The universe of discourse is set to $U = [-1.57, 1.57]$. It defines the direction in which the robot has to navigate. The chose of U was found suitable.
- The linguistic output variable called $Speed = \{backward, softbackward, softforward, forward\}$ with the named linguistic values. The universe of discourse is set to $U = [-1, 1]$. It defines the speed giving to the robot and the chose of U was found suitable for the implementation of the controller.

In order for the inference mechanism to work, one has to define a rule base in which the linguistic variables and values are used. To be able to move the robot, find marbles and avoid obstacles the following rule base has been made:

- **Rule 1:** ***if*** *ObstacleDistance* is veryclose and *ObstacleDirection* is left and *MarbleFound* is no ***then*** *SteerDirection* is softright
- **Rule 2:** ***if*** *ObstacleDistance* is veryclose and *ObstacleDirection* is right and *MarbleFound* is no ***then*** *SteerDirection* is softleft
- **Rule 3:** ***if*** *ObstacleDistance* is veryclose and *ObstacleDirection* is center and *ObstacleFree* is left and *MarbleFound* is no ***then*** *SteerDirection* is softleft
- **Rule 4:** ***if*** *ObstacleDistance* is veryclose and *ObstacleDirection* is center and *ObstacleFree* is right and *MarbleFound* is no ***then*** *SteerDirection* is softright
- **Rule 5:** ***if*** *ObstacleDistance* is veryclose and *MarbleFound* is no ***then*** *Speed* is forward
- **Rule 6:** ***if*** *ObstacleDistance* is close and *MarbleFound* is no and *BoundaryDirection* is left and *GoalDirection* is left ***then*** *SteerDirection* is softleft
- **Rule 7:** ***if*** *ObstacleDistance* is close and *MarbleFound* is no and *BoundaryDirection* is right and *GoalDirection* is right ***then*** *SteerDirection* is softright
- **Rule 8:** ***if*** *ObstacleDistance* is close and *MarbleFound* is no and *BoundaryDirection* is left and *GoalDirection* is right ***then*** *SteerDirection* is softleft
- **Rule 9:** ***if*** *ObstacleDistance* is close and *MarbleFound* is no and *BoundaryDirection* is right and *GoalDirection* is left ***then*** *SteerDirection* is softright
- **Rule 10:** ***if*** *ObstacleDistance* is close and *MarbleFound* is no and *BoundaryDirection* is straight then *SteerDirection* is straight
- **Rule 11:** ***if*** *ObstacleDistance* is close and *MarbleFound* is no ***then*** *Speed* is forward
- **Rule 12:** ***if*** *ObstacleDistance* is far and *MarbleFound* is no and *GoalDirection* is right ***then*** *SteerDirection* is right
- **Rule 13:** ***if*** *ObstacleDistance* is far and *MarbleFound* is no and *GoalDirection* is left ***then*** *SteerDirection* is left
- **Rule 14:** ***if*** *ObstacleDistance* is far and *MarbleFound* is no and *GoalDirection* is straight ***then*** *SteerDirection* is straight
- **Rule 15:** ***if*** *ObstacleDistance* is far and *MarbleFound* is no and *GoalDirection* is straight ***then*** *Speed* is forward

- **Rule 16:** ***if*** *MarbleFound* is yes and *MarbleDirection* is left ***then*** *SteerDirection* is softleft
- **Rule 17:** ***if*** *MarbleFound* is yes and *MarbleDirection* is right ***then*** *SteerDirection* is softright
- **Rule 18:** ***if*** *MarbleFound* is yes and *MarbleDirection* is center ***then*** *Speed* is forward

Center of gravity will be used as the defuzzification method to convert the fuzzy output to crisp output which is defined as

$$x = \frac{\int_{\mu_A}(x)xdx}{\int_{\mu_A}(x)dx} \tag{3.11}$$

where x is the defuzzified output and A is any fuzzy set. The following membership functions can be seen in the figures below.

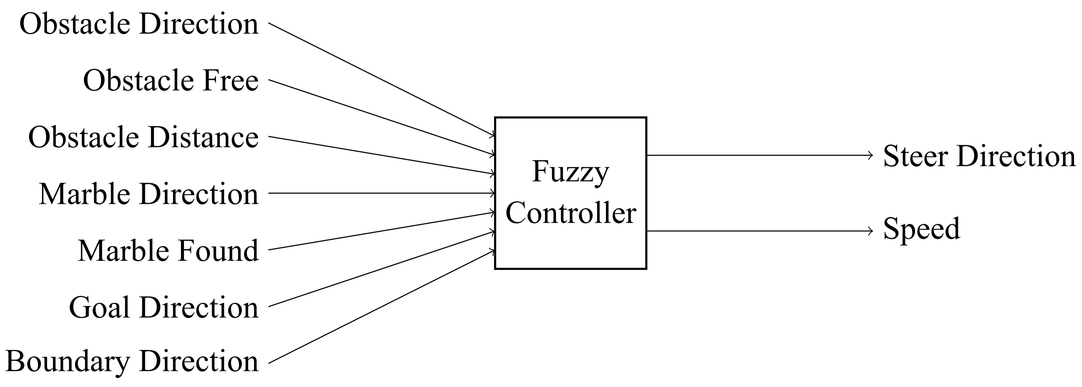
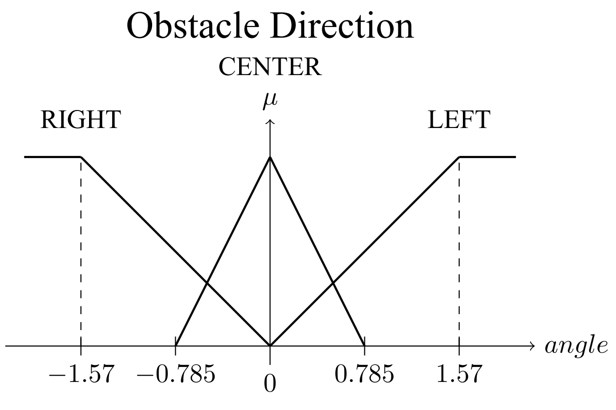
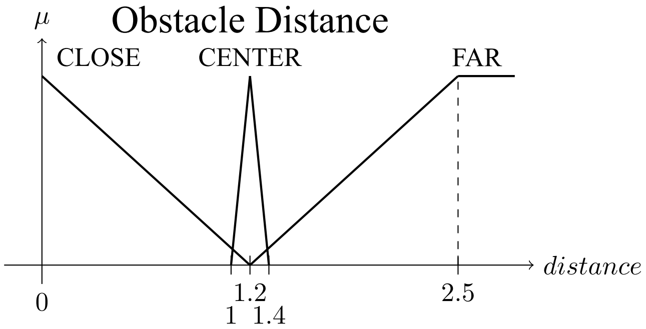


Figure 6: caption

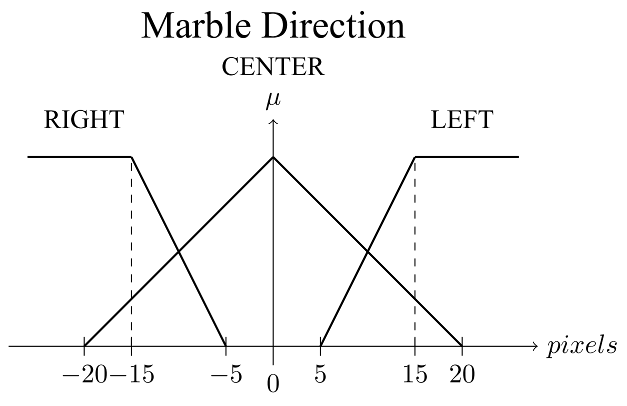


(a) caption

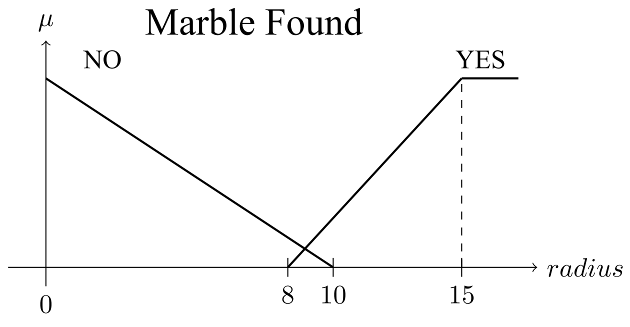


(b) caption

Figure 7: Overall caption

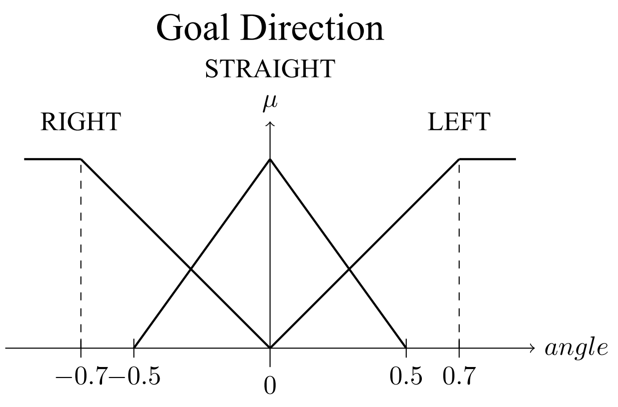


(a) caption

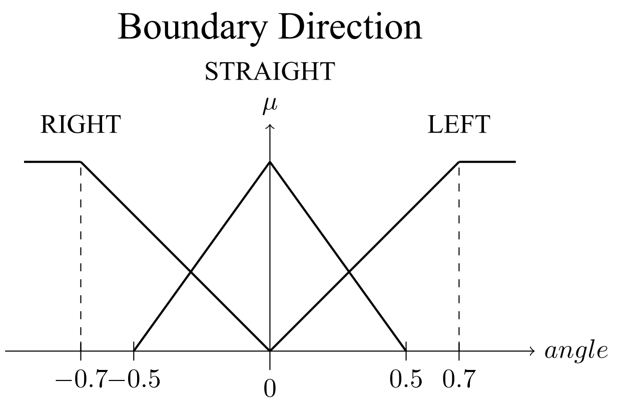


(b) caption

Figure 8: Overall caption



(a) caption



(b) caption

Figure 9: Overall caption

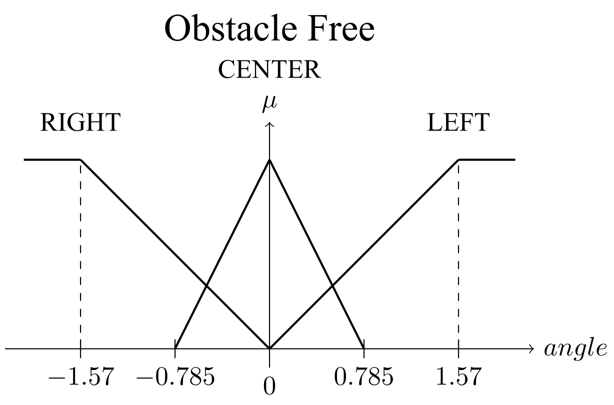


Figure 10: caption

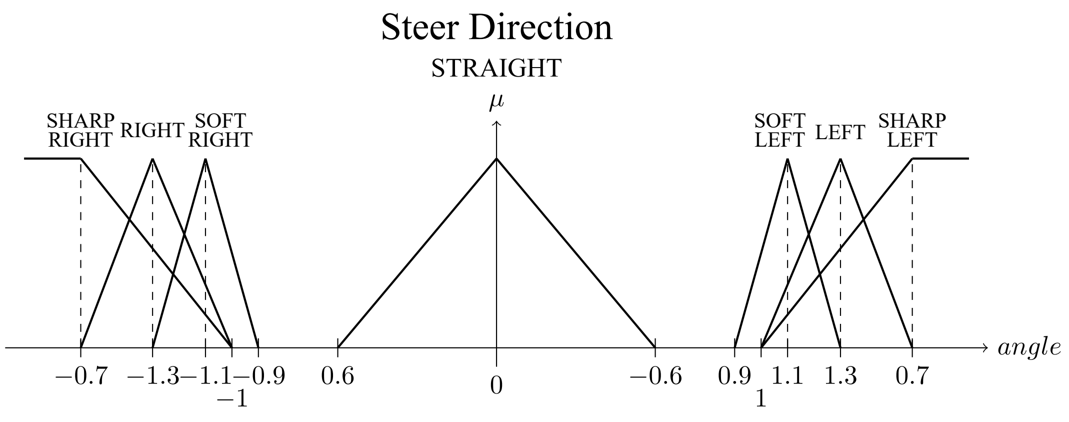


Figure 11: caption

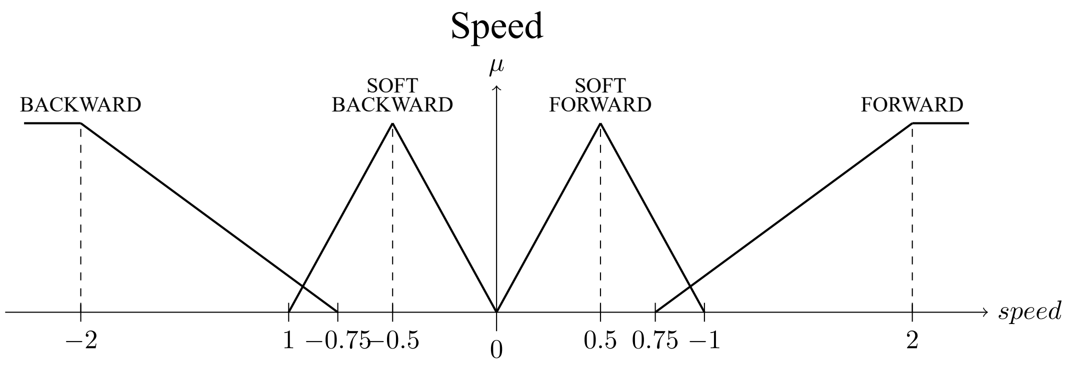


Figure 12: caption

3.6 Q-learning

In order to effectively search the environment and collect marbles, a good search strategy must be found. This can be done by utilising reinforcement learning. By using reinforcement learning, the robot can learn from its experience and obtain a good strategy for navigating the environment.

By using a Temporal-Difference learning strategy the optimal action-value function can be estimated by every move taken unlike a Monte Carlo strategy where an episode terminates before any learning is obtained. In some cases with long episodes the Monte Carlo strategy is considered too slow.
Generally there are two categories of Temporal-Difference learning; on-policy and off-policy methods. One of the advantages of an off-policy over an on-policy method are that the action-value function can be estimated independent from the policy being used. The policy only influences which state-action pairs that are visited and updated.

Based on this Q-learning are chosen, the Q-learning method builds on the following update function for updating the action-value function (Q-values).

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left[R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a\right) - Q\left(S_t, A_t\right)\right] \tag{3.12}$$

The update function for Q-learning consists of the old value for a given state-action combination plus a scaled difference between the old value, the immediate reward and the maximal value for the next state. The learning rate are denoted α and ranging from 0-1 preferable closer to 0, in order to not to base the policy on this action only. γ denotes the discount factor and are also ranging from 0-1, preferable closer to 1, to ensure that future actions matter.

In the box below, the algorithm for Q-learning can be seen.

Algorithm: Q-learning

```
Algorithm parameters: step size: $\alpha \in [0,1]$, small $\epsilon > 0$
Initialise $Q(s,a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(terminal, \dot{}) = 0$
Loop for each episode:
    Initialise $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q(e.g., \epsilon - greedy)$
        Take action $A$, observe $R$, $S^{'}$
        $Q(S,A) \leftarrow Q(S,A) + \alpha \left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
        $S \leftarrow S^{'}$
    until $S$ is terminal
```

3.6.1 Definition of states

In order to perform Q-learning a definition of states must be made. These states must have the Markov property meaning that the probability of a marble being in a room must not depend on whether other rooms have been visited or not.

In order to achieve this, it have been chosen to implement a vector of boolean values, one element for each room. This will be used to store which rooms have been visited.
By doing this all possible combinations will be possible and the reward of entering a room will not depend on the other rooms, only the room itself.

It have also been chosen to implement the state with an integer storing the room number, and a boolean for storing whether the state is terminal or not. The definition of the state can be seen below.

```
qState
    int roomNumber
    std::vector<bool> roomsVisited
    bool isTerminal
```

4 Implementation

4.1 Line detection

4.1.1 Marble detection

4.2 Tangent bug

4.3 Search strategy

4.4 Q-learning

Q-learning was implemented by designing a class containing the functionality of holding the state space and update the Q-matrix. This class was named `q_learning`.

Due to the definition of the state, each room will have $2^{\text{number of rooms}}$ number of states, corresponding to the number of possible combinations of how the rooms could be visited. This will result in a very large Q-matrix. This could be overcome by splitting the Q-matrix into several smaller Q-matrices and then keep track of the mapping between them.
The Q-matrix would consist of number of base states by the number of actions which would be a 6 x 6 matrix with 5 rooms (including an initial state)

	Start	**Room 1**	**Room 2**	**Room 3**	**Room 4**	**Room 5**
Start	0	-100	-100	13.33971	-100	-100
Room 1	-100	0	8.681309	-100	-100	-100
Room 2	-100	0.405394	0	10.74771	-100	-100
Room 3	-100	-100	7.757309	0	7.837891	17.456
Room 4	-100	-100	-100	11.01171	0	-100
Room 5	-100	-100	-100	10.79571	-100	0

Table 1: Rewards for all state-action combinations given that no rooms have been visited and initial state is room 3

5 Discussion

6 Conclusion

Appendices

A Tests

A.1 Room based probability of marbles spawning

The purpose of this test is to determine the probability of a marble spawning in each of the 14 rooms described in section **??**.

A.1.1 Description of test

This test was done by conduction a total of 50 tests, where the position of the 20 marbles in the Gazebo environment are saved resulting in a total of 1000 samples.

These marbles was then mapped to one of the 14 rooms using the class `map_class`. The total amount of marbles found in each room can be seen in table **??**. This data was then divided by the total number of marbles, to find the probability.

Due to the fact that the rooms are not the same size, this probability was divided by the size of the room (number of pixels in the map) and the normalised. The result can be ssen in table **??**.

A.1.2 Test parameters

- World used	bigworld
- Number of spawned marbles	20
- Number of tests	50

A.1.3 Data

Distribution of marbles	
Total number of marbles found in room 1	5
Total number of marbles found in room 2	65
Total number of marbles found in room 3	75
Total number of marbles found in room 4	52
Total number of marbles found in room 5	72
Total number of marbles found in room 6	158
Total number of marbles found in room 7	17
Total number of marbles found in room 8	118
Total number of marbles found in room 9	100
Total number of marbles found in room 10	22
Total number of marbles found in room 11	75
Total number of marbles found in room 12	160
Total number of marbles found in room 13	47
Total number of marbles found in room 14	34

Table 2: Table of how the marbles are distributed in the 14 rooms based on all 50 tests

Probability of marbles	
Probability of marbles found in room 1	0.012232
Probability of marbles found in room 2	0.061057
Probability of marbles found in room 3	0.078610
Probability of marbles found in room 4	0.059975
Probability of marbles found in room 5	0.117993
Probability of marbles found in room 6	0.098801
Probability of marbles found in room 7	0.019629
Probability of marbles found in room 8	0.099063
Probability of marbles found in room 9	0.114518
Probability of marbles found in room 10	0.027633
Probability of marbles found in room 11	0.044562
Probability of marbles found in room 12	0.130379
Probability of marbles found in room 13	0.072915
Probability of marbles found in room 14	0.062541

Table 3: Table of how the probabilities are distributed in the 14 rooms with room size taken into account

A.1.4 Conclusion

It can be concluded that the highest number of marbles was found in room 12 closely followed by 6 and 8. But due to the size of the rooms, the highest probability is found in room 12, followed by 5 and 9.

A.2 Estimate for path lengths

The purpose of this test was to find an estimate for the distances between the rooms in order to have a distance punishment for the Q-learning.

A.2.1 Description of test

This test was conducted using Geogebra a cas tool for geometry and algebra. An image of the environment "bigworld" was loaded into the program. The width of the image was set to 8.4 cm.

Based on this lines was drawn by hand between the centroids of the rooms navigating any obstacles. This can be seen on figure **??**.

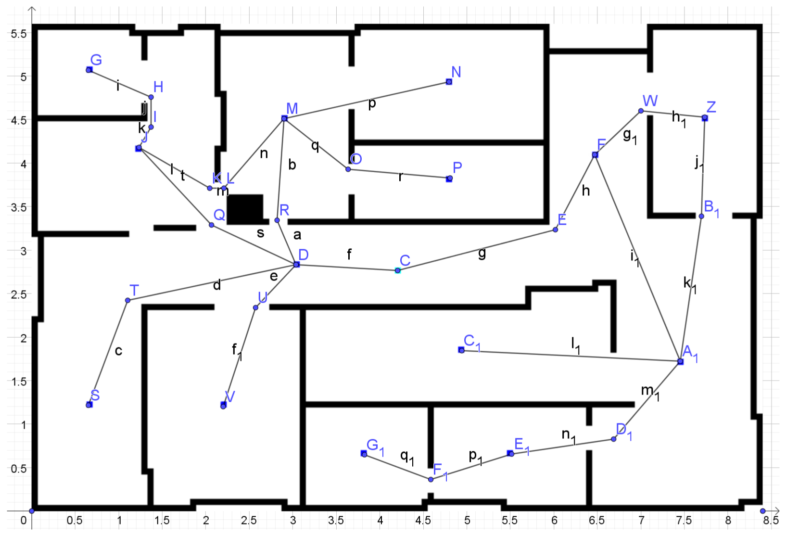


Figure 13: Illustration of paths in geogebra

Geogebra calculates the length of each line. Based on this the total path length from one room to another was found. This can be seen in table gegegepgkek

A.2.2 Test parameters

- World used	bigworld
- Length of world	8.4 cm

A.2.3 Data

Distribution of marbles			
Distance from room 1 to 2	-2.78	Distance from room 2 to 3	-4.32
Distance from room 2 to 6	-4.58	Distance from room 3 to 4	-3.88
Distance from room 3 to 5	-4.24	Distance from room 3 to 6	-3.46
Distance from room 6 to 7	-6.54	Distance from room 6 to 8	-3.76
Distance from room 7 to 9	-8.02	Distance from room 9 to 10	-2.94
Distance from room 9 to 11	-5.14	Distance from room 10 to 11	-5.64
Distance from room 11 to 12	-5.02	Distance from room 11 to 13	-4.72
Distance from room 13 to 14	-3.56		

Table 4: Distances between rooms

A.3 The impact of ϵ on Q-learning performance

The purpose of this test is to show how different values of ϵ influences the performance of Q-learning.

A.3.1 Description of test

This test was done by performing 100 trials of each selected value of ϵ for a range of episodes. The test was conducted using the world "5-room world" seen on figure **??**.
The distance punishments are bases on those found in the test in appendix **??**. The distance punishments was scaled with a factor of 1.2 to make the distance a bigger factor in final path.
The probabilities used for each room was found in the test in appendix **??**. These probabilities was divided by the maximal value and scaled by a factor of 20. This was done to ensure that the total reward for entering a room the first time would be positive.

The initial state for all tests was set to room 3 in order to ensure greatest number of possible paths.

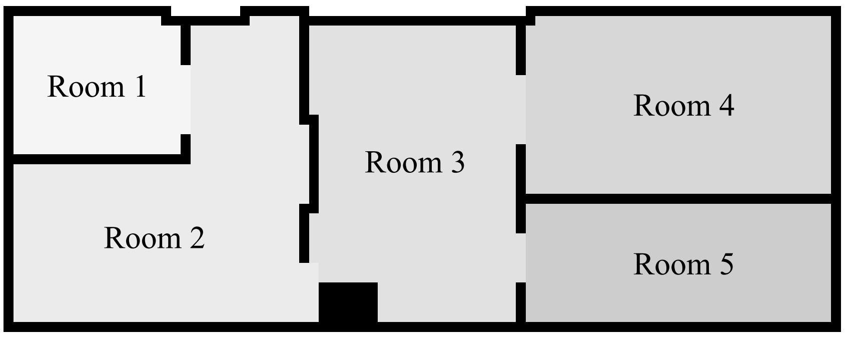


Figure 14: Illustration of "5-room world"

A.3.2 Test parameters

In the following tables, the parameters for the test can be seen. In order not to make the agent act completely randomly, it have been chosen to make the test on values of ϵ between 0.01 and 0.5.

- World used	5-room world
- Initial room	room 3
- Probabilities based on	50 tests
- Number of tests	100
- Scaling factor distance	1.2
- Scaling factor reward	20
- Learning rate α	0.1
- Discount factor γ	0.9

Tested values of ϵ	
0.01	0.15
0.025	0.2
0.05	0.3
0.075	0.4
0.1	0.5

Table 5: Table of tested the values of ϵ. Ranging from 1 % to 50 %

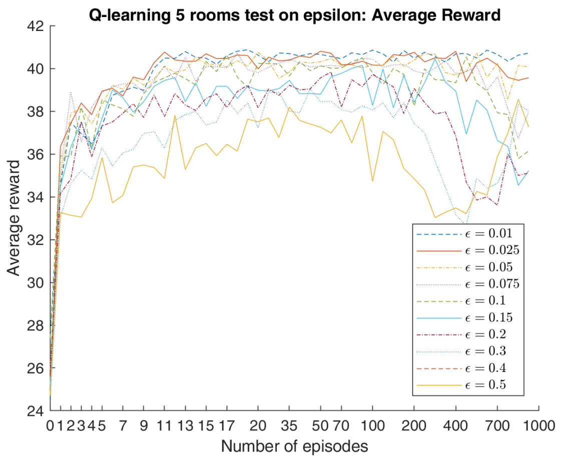
Distance punishments	
Start to room 3	0
Room 1 to room 2	-1.668
Room 2 to room 3	-2.592
Room 3 to room 4	-2.328
Room 3 to room 5	-2.544

Table 6: Table of the distance punishments. The distances are found in the test in appendix **??**

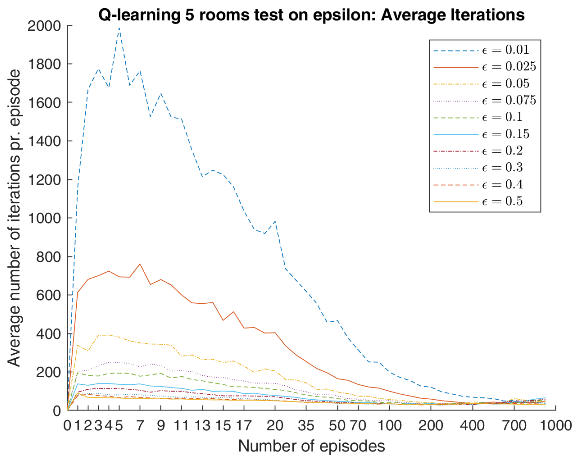
A.3.3 Data

On figure **??** the average reward for all tests can be seen. It can be seen that the lower the value of ϵ the higher average reward, meaning that the policy will converge to the optimal policy.

On figure **??** the average number of iterations per episoden can be seen. It can be seen that the higher the value of ϵ the lower the average number of iterations will be. Given the fact that the more random the agent acts, the faster the agent will search alternative paths to the policy, and find its way through the environment.



(a) Plot of how the average reward develops as a function of the number of episodes for each value of ϵ



(b) Plot of how the average number of episodes develops as a function of the number of episodes for each value of ϵ

Figure 15: Plots of both the average reward and average number of iterations pr. episode for each value of ϵ

A value of 0.05 have been chosen as the best compromise between a high average reward and a low average number of iterations per episode.

A.3.4 Conclusion

It can be concluded that the smaller the value of ϵ the higher the average reward will be for a given number of episodes.
It can as well be concluded that the higher the value of ϵ the lower the number of iterations per episode will be.
The value of 0.05 was chosen as the best compromise.

A.4 The impact of α on Q-learning performance

The purpose of this test is to show how different values of α influences the performance of Q-learning.

A.4.1 Description of test

This test was done by performing 100 trials of each selected value of α for a range of episodes. The test was conducted using the world ”5-room world” seen on figure **??**.
The distance punishments are bases on those found in the test in appendix **??**. The distance punishments was scaled with a factor of 1.2 to make the distance a bigger factor in final path.
The probabilities used for each room was found in the test in appendix **??**. These probabilities was divided by the maximal value and scaled by a factor of 20. This was done to ensure that the total reward for entering a room the first time would be positive.

The initial state for all tests was set to room 3 in order to ensure greatest number of possible paths.

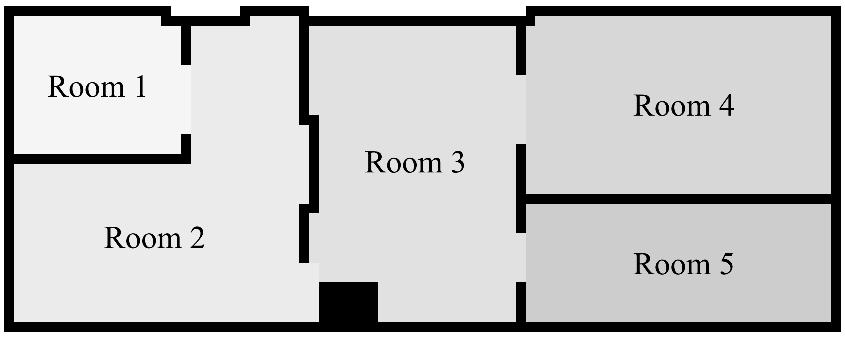


Figure 16: Illustration of ”5-room world”

A.4.2 Test parameters

In the following tables, the parameters for the test can be seen. It have been chosen to make the test on values of α between 0.01 and 0.5.

- World used	5-room world
- Initial room	room 3
- Probabilities based on	50 tests
- Number of tests	100
- Scaling factor distance	1.2
- Scaling factor reward	20
- Randomness factor ϵ	0.05
- Discount factor γ	0.9

Tested values of α	
0.01	0.15
0.025	0.2
0.05	0.3
0.075	0.4
0.1	0.5

Table 7: Table of tested the values of α. Ranging from 0.01 to 0.5

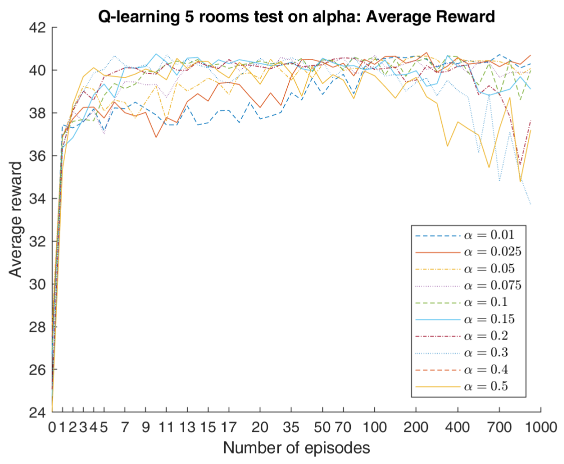
Distance punishments	
Start to room 3	0
Room 1 to room 2	-1.668
Room 2 to room 3	-2.592
Room 3 to room 4	-2.328
Room 3 to room 5	-2.544

Table 8: Table of the distance punishments. The distances are found in the test in appendix **??**

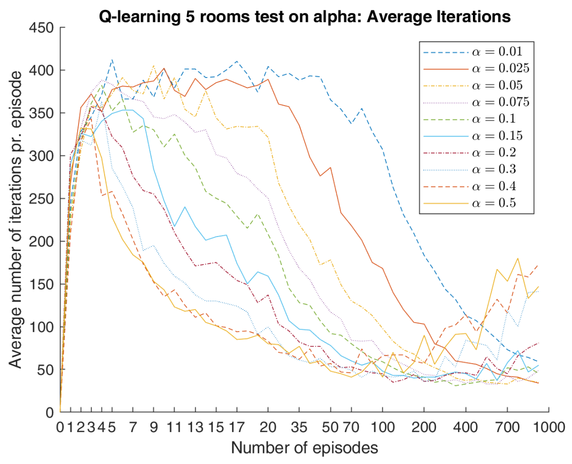
A.4.3 Data

On figure **??** the average reward for all tests can be seen. It can be seen that the lower the value of α the higher average reward, meaning that the policy will converge to the optimal policy. Whereas a higher value of α will result in a lower average reward. It would also seem like the average reward are oscillating more.

On figure **??** the average number of iterations per episoden can be seen. It can be seen that the higher the value of α the lower the average number of iterations will be. This is due to the fact, that the lower the learning rate are, the slower the algorithm learns, and will therefore need more steps to get to the goal.



(a) Plot of how the average reward develops as a function of the number of episodes for each value of ϵ



(b) Plot of how the average number of episodes develops as a function of the number of episodes for each value of α

Figure 17: Plots of both the average reward and average number of iterations pr. episode for each value of α

A value of 0.025 have been chosen as the best compromise between a high average reward and a low average number of iterations per episode.

A.4.4 Conclusion

It can be concluded that the smaller the value of α the higher the average reward will be for a given number of episodes. It can as well be concluded that the higher the value of α the lower the number of iterations per episode will be.
The value of 0.025 was chosen as the best compromise.

A.5 The impact of γ on Q-learning performance

The purpose of this test is to show how different values of γ influences the performance of Q-learning.

A.5.1 Description of test

This test was done by performing 100 trials of each selected value of γ for a range of episodes. The test was conducted using the world ”5-room world” seen on figure **??**.
The distance punishments are bases on those found in the test in appendix **??**. The distance punishments was scaled with a factor of 1.2 to make the distance a bigger factor in final path.
The probabilities used for each room was found in the test in appendix **??**. These probabilities was divided by the maximal value and scaled by a factor of 20. This was done to ensure that the total reward for entering a room the first time would be positive.

The initial state for all tests was set to room 3 in order to ensure greatest number of possible paths.

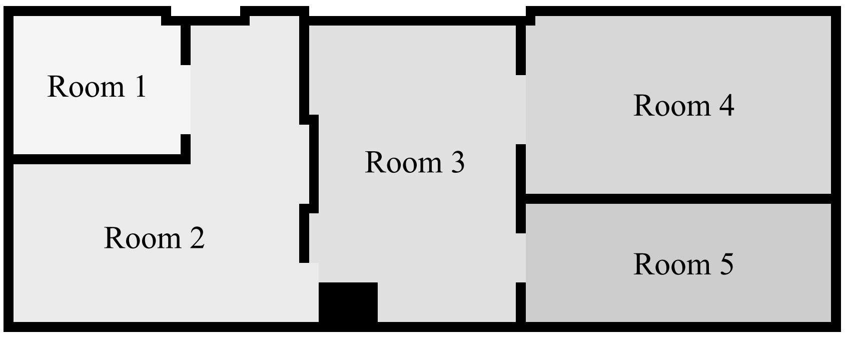


Figure 18: Illustration of ”5-room world”

A.5.2 Test parameters

In the following tables, the parameters for the test can be seen. It have been chosen to make the test on values of γ between 0.99 and 0.5.

- World used	5-room world
- Initial room	room 3
- Probabilities based on	50 tests
- Number of tests	100
- Scaling factor distance	1.2
- Scaling factor reward	20
- Randomness factor ϵ	0.05
- Learning rate α	0.025

Tested values of γ	
0.99	0.85
0.975	0.8
0.95	0.7
0.925	0.6
0.9	0.5

Table 9: Table of tested the values of γ. Ranging from 0.99 to 0.5

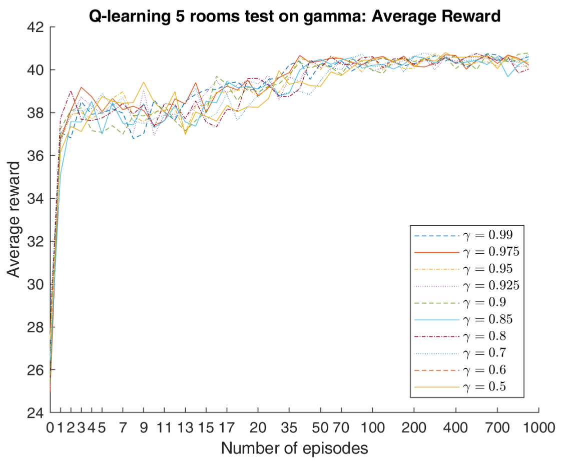
Distance punishments	
Start to room 3	0
Room 1 to room 2	-1.668
Room 2 to room 3	-2.592
Room 3 to room 4	-2.328
Room 3 to room 5	-2.544

Table 10: Table of the distance punishments. The distances are found in the test in appendix **??**

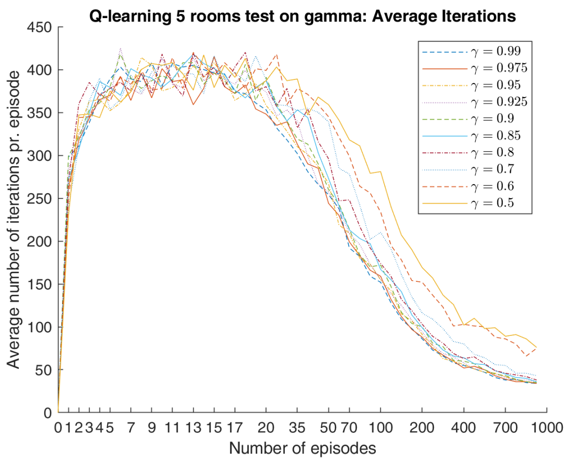
A.5.3 Data

On figure **??** the average reward for all tests can be seen. It can be seen that the higher the value of γ the higher average reward. Whereas a lower value of γ will result in a lower average reward.

On figure **??** the average number of iterations per episoden can be seen. It can be seen that the higher the value of γ the lower the average number of iterations will be.



(a) Plot of how the average reward develops as a function of the number of episodes for each value of ϵ



(b) Plot of how the average number of episodes develops as a function of the number of episodes for each value of α

Figure 19: Plots of both the average reward and average number of iterations pr. episode for each value of γ

A value of 0.99 have been chosen as the best compromise between a high average reward and a low average number of iterations per episode.

A.5.4 Conclusion

It can be concluded that the higher the value of γ the higher the average reward will be for a given number of episodes.
It can as well be concluded that the higher the value of γ the lower the number of iterations per episode will be.
The value of 0.99 was chosen as the best compromise.

A.6 Best path based on Q-learning

The purpose of this test was to find the optimal path for covering 5-room world in the best possible way

A.6.1 Description of test

This test consisted of 5 subtest, where each test was started in different rooms. This was to find out, which room was the most optimal to start in. Each subtest consisted of 10 trials, where all parameters where the same.
The test was conducted using the world "5-room world" seen on figure **??**.
The distance punishments are bases on those found in the test in appendix **??**. The distance punishments was scaled with a factor of 1.2 to make the distance a bigger factor in final path.
The probabilities used for each room was found in the test in appendix **??**. These probabilities was divided by the maximal value and scaled by a factor of 20. This was done to ensure that the total reward for entering a room the first time would be positive.
All tests ran for 2000 episodes, with ϵ varying from 0.5 to 0.05. ϵ was started on 0.5 and reduced by 0.05 for each 200 episodes, enabling the algorithm to make larger learning steps in the beginning.

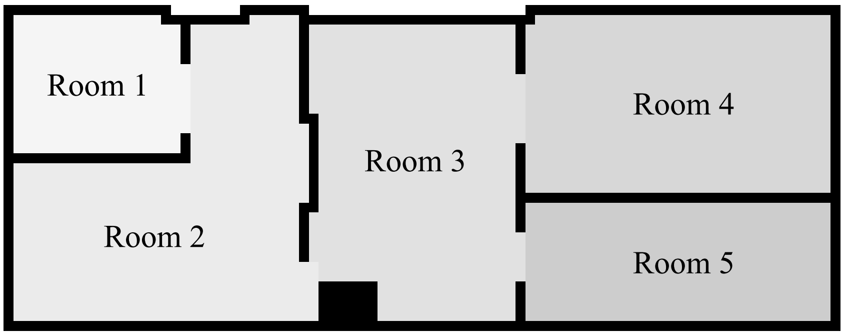


Figure 20: Illustration of "5-room world"

A.6.2 Test parameters

In the following tables, the parameters for the test can be seen.

- World used	5-room world
- Probabilities based on	50 tests
- Number of tests	10
- Scaling factor distance	1.2
- Scaling factor reward	20
- Randomness factor ϵ	0.5, 0.45, 0.4, 0.35, 0.3, 0.25, 0.2, 0.15, 0.1, 0.05
- Learning rate α	0.025
- Discount factor γ	0.99

Distance punishments	
Start to room 3	0
Room 1 to room 2	-1.668
Room 2 to room 3	-2.592
Room 3 to room 4	-2.328
Room 3 to room 5	-2.544

Table 11: Table of the distance punishments. The distances are found in the test in appendix **??**

A.6.3 Data

In the following tables, the result from those tests can be seen. Looking at the average reward from those 5 tests, it can be seen that the best average was achieved in the test starting in room 1. The best case was found to be was found to be both start in room 1 and 5.

Start in room 1		
Trial	**Path**	**Reward**
1	0, 1, 2, 3, 5, 3, 4	44.2521
2	0, 1, 2, 3, 5, 3, 4	44.4681
3	0, 1, 2, 3, 4, 3, 5	44.2521
4	0, 1, 2, 3, 5, 3, 4	44.2521
5	0, 1, 2, 3, 5, 3, 4	44.2521
6	0, 1, 2, 3, 5, 3, 4	44.2521
7	0, 1, 2, 3, 5, 3, 4	44.4681
8	0, 1, 2, 3, 4, 3, 5	44.2521
9	0, 1, 2, 3, 5, 3, 4	44.2521
10	0, 1, 2, 3, 5, 3, 4	44.2521
Average Reward		44.3001
Best	0, 1, 2, 3, 5, 3, 4	44.4681

Table 12: All 10 paths from test started in room 1

Start in room 2		
Trial	**Path**	**Reward**
1	0, 2, 3, 5, 3, 4, 3, 2, 1	39.3322
2	0, 2, 3, 5, 3, 4, 3, 2, 1	39.3322
3	0, 2, 3, 5, 3, 4, 3, 4, 3, 2, 1	34.6762
4	0, 2, 3, 5, 3, 4, 3, 5, 3, 5, 3, 2, 1	29.1562
5	0, 2, 3, 4, 4, 3, 5, 3, 4, 3, 2, 1	34.6762
6	0, 2, 3, 4, 3, 5, 3, 4, 3, 4, 3, 4, 3, 2, 1	25.3642
7	0, 2, 3, 5, 3, 4, 3, 5, 3, 2, 1	34.2442
8	0, 2, 3, 5, 3, 4, 3, 5, 3, 3, 2, 1	34.2442
9	0, 2, 3, 4, 3, 5, 3, 5, 3, 2, 1	34.2442
10	0, 2, 3, 5, 5, 3, 4, 3, 5, 3, 4, 3, 2, 2, 1	29.5882
Average Reward		33.4858
Best	0, 2, 3, 5, 3, 4, 3, 2, 1	39.3322

Table 13: All 10 paths from test started in room 2

Start in room 3		
Trial	**Path**	**Reward**
1	0, 3, 5, 3, 2, 1, 2, 3, 4	39.9921
2	0, 3, 4, 3, 2, 1, 2, 3, 5	40.2081
3	0, 3, 2, 1, 2, 3, 5, 3, 4	39.9921
4	0, 3, 5, 3, 4, 3, 2, 1	41.9241
5	0, 3, 2, 1, 2, 3, 4, 3, 5	40.2081
6	0, 3, 2, 1, 2, 3, 5, 3, 4	39.9921
7	0, 3, 2, 1, 2, 3, 5, 3, 4	39.9921
8	0, 3, 2, 1, 2, 3, 5, 3, 4	39.9921
9	0, 3, 4, 3, 5, 3, 2, 1	41.9241
10	0, 3, 4, 3, 5, 3, 4, 3, 0, 3, 2, 1	37.2681
Average Reward		40.1493
Best	0, 3, 5, 3, 4, 3, 2, 1	41.9241

Table 14: All 10 paths from test started in room 3

Start in room 4		
Trial	**Path**	**Reward**
1	0, 4, 3, 5, 3, 2, 1	44.2521
2	0, 4, 3, 2, 1, 2, 3, 5	42.5361
3	0, 4, 3, 2, 3, 5, 3, 4, 3, 4, 3, 2, 1	29.7562
4	0, 4, 3, 2, 1, 2, 3, 5	42.5361
5	0, 4, 3, 5, 3, 2, 1	44.2521
6	0, 4, 3, 2, 3, 5, 3, 2, 1	39.0681
7	0, 4, 3, 5, 3, 2, 1	44.2521
8	0, 4, 0, 4, 3, 2, 1, 2, 3, 5	42.5361
9	0, 4, 3, 5, 3, 2, 1	44.2521
10	0, 4, 3, 5, 3, 2, 1	44.2521
Average Reward		41.7693
Best	0, 4, 3, 5, 3, 2, 1	44.2521

Table 15: All 10 paths from test started in room 4

Start in room 5		
Trial	**Path**	**Reward**
1	0, 5, 3, 2, 1, 2, 3, 4	42.5361
2	0, 5, 3, 2, 1, 2, 3, 4	42.5361
3	0, 5, 3, 5, 5, 3, 4, 3, 2, 1	39.3801
4	0, 5, 3, 2, 1, 2, 3, 4	42.5361
5	0, 5, 3, 2, 1, 2, 3, 4	42.5361
6	0, 5, 3, 2, 1, 1, 2, 3, 2, 3, 4	37.3521
7	0, 5, 3, 2, 1, 2, 3, 4	42.5361
8	0, 5, 3, 4, 3, 2, 1	44.4681
9	0, 5, 3, 4, 3, 2, 1	44.4681
10	0, 5, 3, 4, 3, 2, 1	44.4681
Average Reward		42.2817
Best	0, 5, 3, 4, 3, 2, 1	44.4681

Table 16: All 10 paths from test started in room 5

A.6.4 Conclusion

It can be concluded that the best average reward was found in the test starting in room 1. It can also be concluded that the best path overall was found in either room 1 or 5, where the best case in both tests had the same reward.
The best path was found to be either of the following two.

Initial room	**Path**	**Reward**
1	0, 1, 2, 3, 5, 3, 4	44.4681
5	0, 5, 3, 4, 3, 2, 1	44.4681

A.7 Motion planers in action

The purpose of this test is to find out how well the motion planners work in an actual environment with obstacles. The goal is to lead the robot from an initial position to a target location.

A.7.1 Description of test

The initial position of the robot will be in origo of the environment of the gazebo simulator. Here 14 test of both the tangent bug algorithm and model based planner will be conducted for the 14 rooms of the *bigworld* map. The idea is to test the success rate of finding rooms, the robots distance to the closest obstacle on the path to the goal and the distance travelled along the way. The model based planer will be tested with a higher speed than the tangent bug algorithm because it will be further away from obstacle most of the time and thereby less likely of hitting obstacles.

A.7.2 Test parameters

- World used	bigworld
- Speed of tangent bug	-1 to 1
- Speed of model planer	-2 to 2
- Number of tests	14

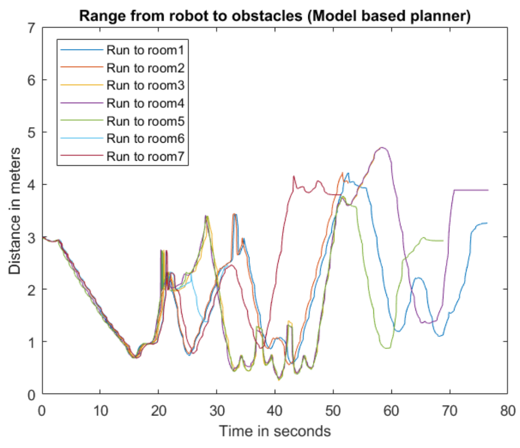
A.7.3 Data

Modelbased					
	Successrate (%)	Distance traveled (m)	Time (s)	Approx velocity (m/s)	Distance to obstacle (m)
Room 1	100	63.087	76.474	0.825	1.990
Room 2	100	43.331	52.589	0.824	1.876
Room 3	100	44.450	58.178	0.764	1.886
Room 4	100	57.150	76.660	0.745	2.157
Room 5	100	50.800	68.931	0.737	1.809
Room 6	100	23.576	31.978	0.737	1.822
Room 7	100	41.920	51.427	0.815	2.127
Room 8	100	25.400	33.170	0.766	2.401
Room 9	100	31.044	36.554	0.849	2.752
Room 10	100	44.450	52.626	0.845	2.435
Room 11	100	37.355	35.697	1.046	2.628
Room 12	100	59.851	69.724	0.858	2.966
Room 13	100	57.029	67.993	0.839	3.094
Room 14	100	76.906	87.035	0.884	2.893
Average	**100**	**46.882**	**57.074**	**0.824**	**2.345**

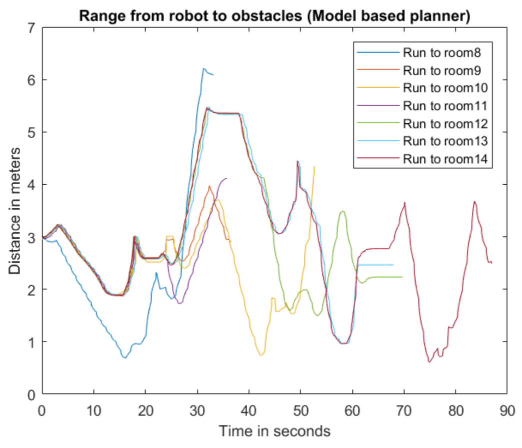
Table 17: Table over test data for the model based planner

Sensorbased					
	Successrate (%)	Distance traveled (m)	Time (s)	Approx velocity (m/s)	Distance to obstacle (m)
Room 1	0	-	-	-	-
Room 2	100	31.044	87.956	0.353	1.613
Room 3	100	23.283	79.409	0.293	1.683
Room 4	100	38.806	135.232	0.287	1.316
Room 5	100	28.222	415.227	0.068	1.033
Room 6	100	8.467	28.323	0.299	3.050
Room 7	0	-	-	-	-
Room 8	100	23.283	79.528	0.293	1.724
Room 9	100	23.989	83.720	0.287	1.193
Room 10	100	38.806	123.614	0.314	1.432
Room 11	100	35.278	102.771	0.343	1.320
Room 12	0	-	-	-	-
Room 13	0	-	-	-	-
Room 14	0	-	-	-	-
Average	**64.286**	**27.909**	**126.198**	**0.282**	**1.596**

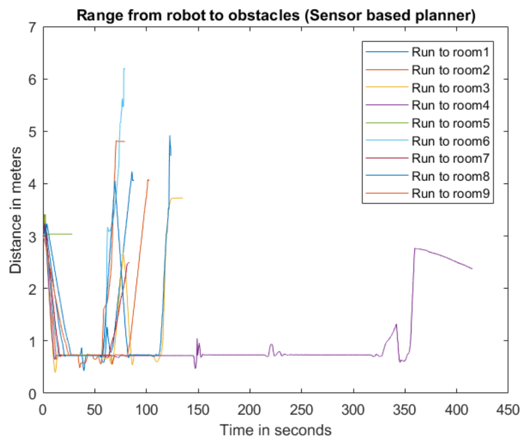
Table 18: Table over test data for the sensorbased planner



(a) Illustration of the distance to the closest obstacle for room 1-7 for the model based planner

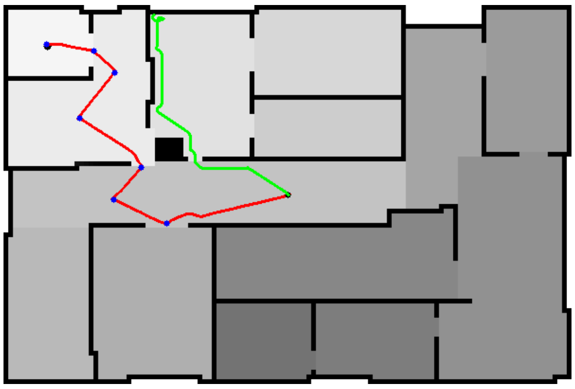


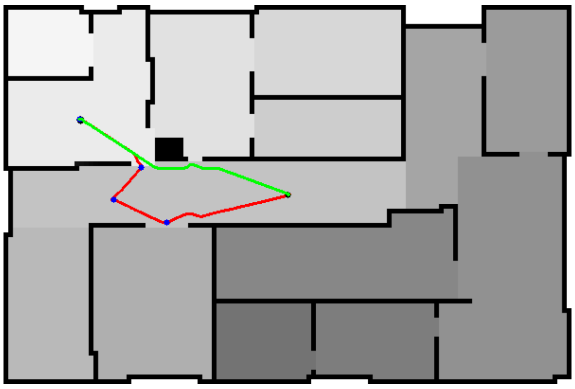
(b) Illustration of the distance to the closest obstacle for room 8-14 for the model based planner

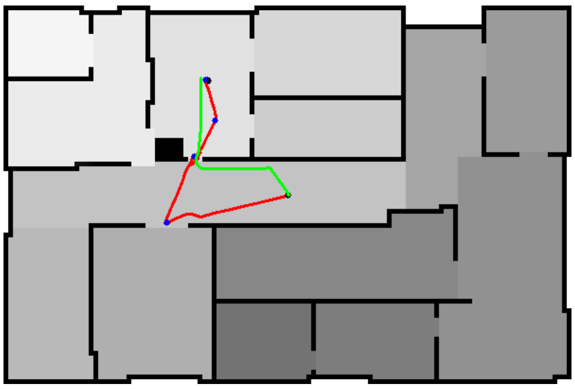


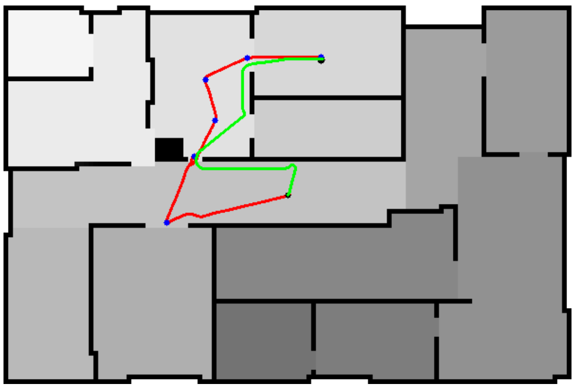
(c) Illustration of the distance to the closest obstacle for room 1-14 for the sensor based planner

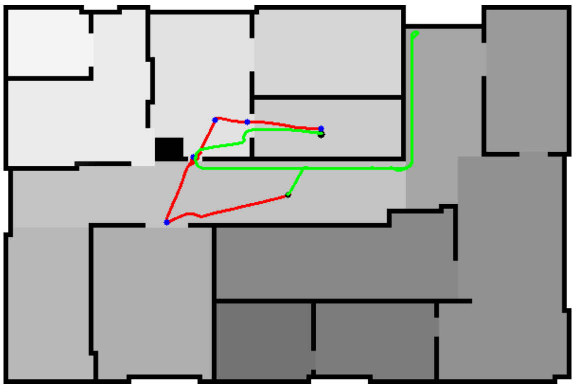
Figure 21: Illustration of the distance to the closest obstacle for room 1-14 for both planners

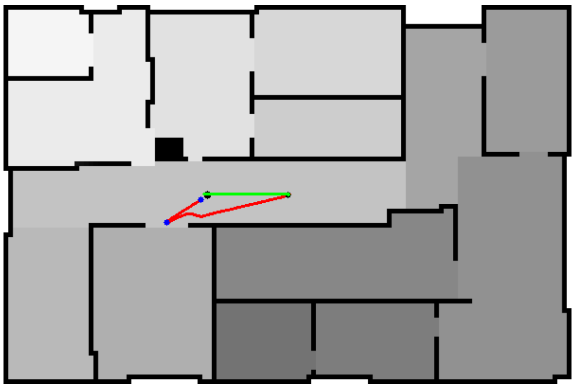

(a) Illustration of the run from the origin to room 1

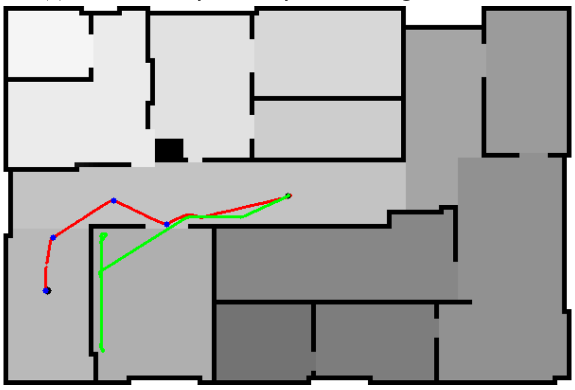

(b) Illustration of the run from the origin to room 2


(c) Illustration of the run from the origin to room 3


(d) Illustration of the run from the origin to room 4


(e) Illustration of the run from the origin to room 5


(f) Illustration of the run from the origin to room 6


(g) Illustration of the run from the origin to room 7

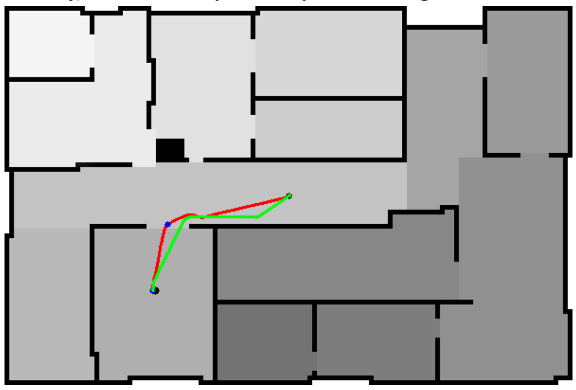
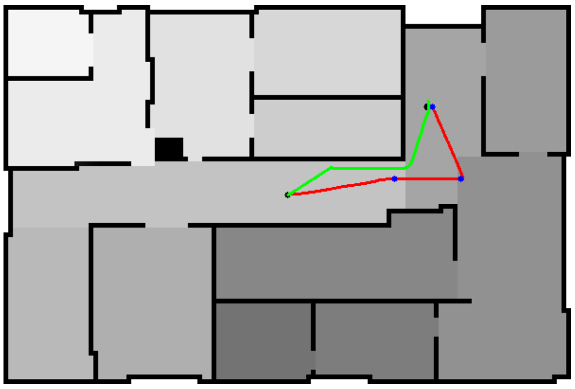
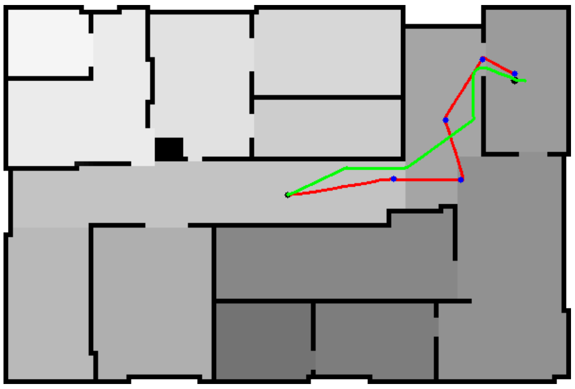
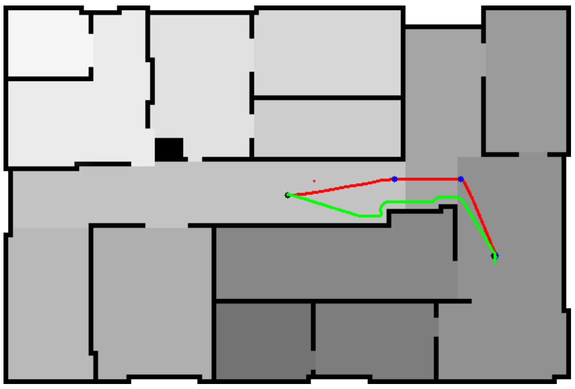
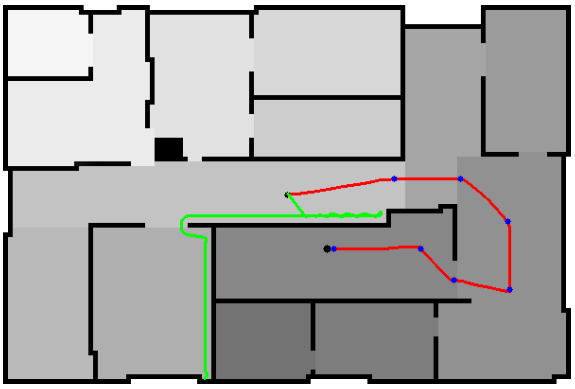

(h) Illustration of the run from the origin to room 8

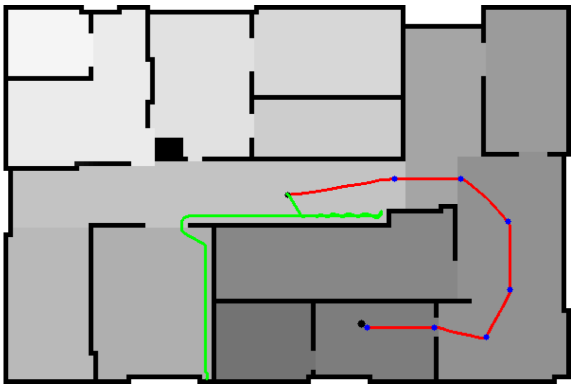
Figure 22: Illustration of the run from the origin to room 1-8 for both the model based (red line) and the sensor based (green line)


(a) Illustration of the run from the origin to room 9


(b) Illustration of the run from the origin to room 10


(c) Illustration of the run from the origin to room 11


(d) Illustration of the run from the origin to room 12


(e) Illustration of the run from the origin to room 13

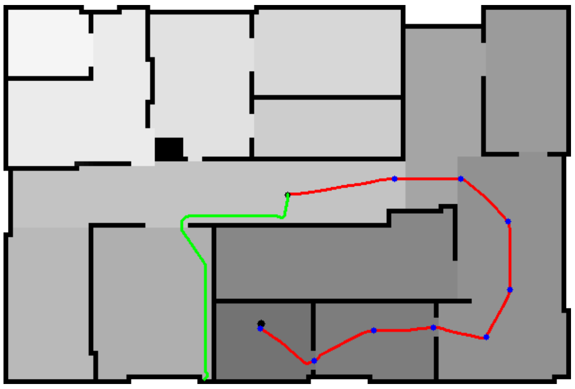

(f) Illustration of the run from the origin to room 14

Figure 23: Illustration of the run from the origin to room 9-14 for both the model based (red line) and the sensor based (green line)

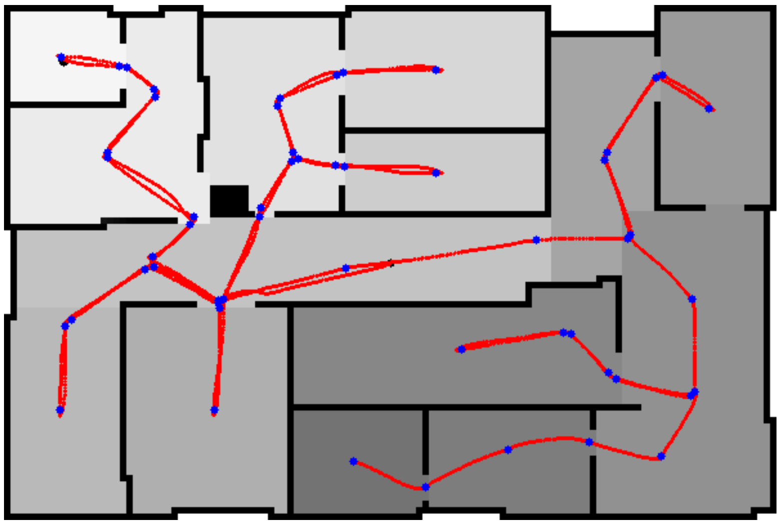


Figure 24: Illustration of the run visiting all rooms starting from the origin for the model based planner

A.7.4 Conclusion

A.8 Model based planners effectiveness to collect marbles

The purpose of this test is to figure out the effectiveness of the model based planner to collect marbles.

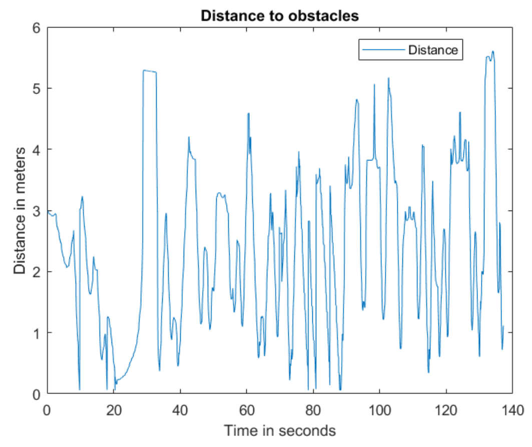
A.8.1 Description of test

A.8.2 Test parameters

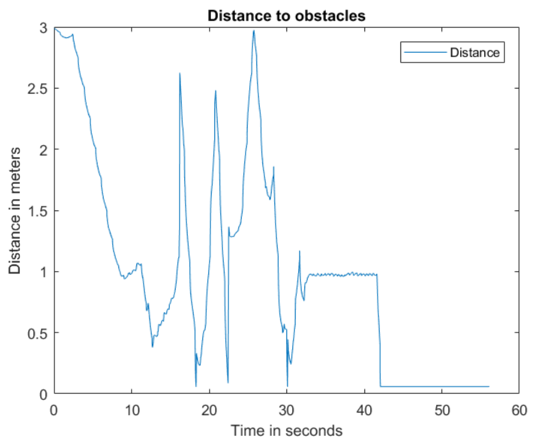
- World used	bigworld
- Speed of tangent bug	-1 to 1
- Speed of model planer	-2 to 2
- Number of tests	10
- Number of marbles	20

A.8.3 Data

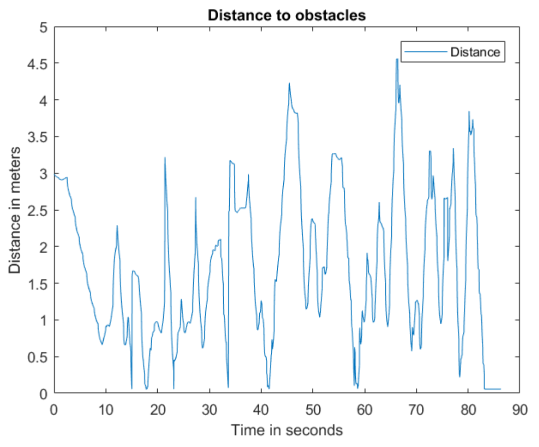
Model based										
	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Marble 1	9.801	18.274	14.988	8.323	3.741	6.455	15.610	8.146	13.430	6.290
Marble 2	18.047	22.438	17.533	11.381	8.203	18.743	36.665	22.374	20.013	22.489
Marble 3	20.430	30.108	23.129	29.490	61.622	89.419	57.234	26.909	31.216	29.828
Marble 4	78.501	42.058	33.680	71.099	70.176	-	58.387	-	43.563	48.259
Marble 5	80.805	-	41.228	95.845	-	-	86.742	-	47.022	78.271
Marble 6	87.962	-	58.009	98.683	-	-	96.940	-	-	79.921
Marble 7	88.036	-	83.116	116.301	-	-	-	-	-	87.543
Marble 8	-	-	-	120.020	-	-	-	-	-	93.739
Marble 9	-	-	-	131.455	-	-	-	-	-	97.356
Marble 10	-	-	-	169.128	-	-	-	-	-	101.262



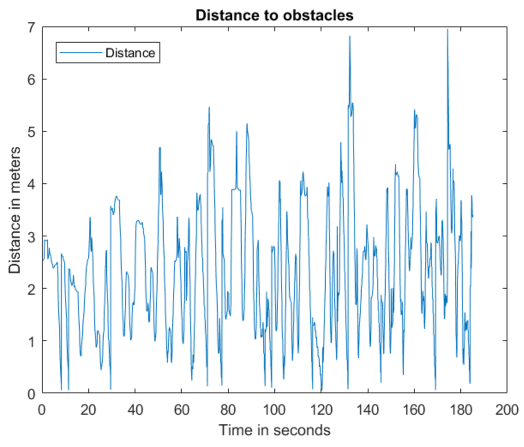
(a) *Illustration of distance to obstacles for test 1*



(b) *Illustration of distance to obstacles for test 2*

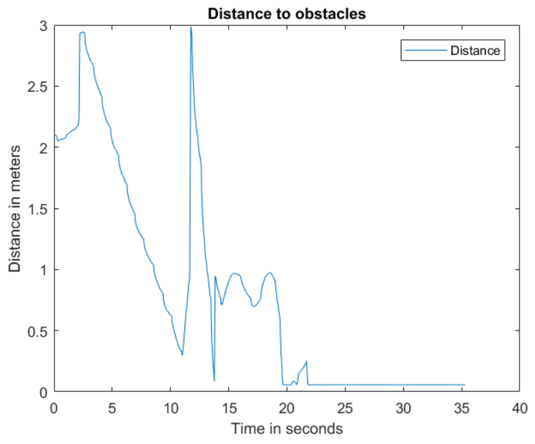


(c) *Illustration of distance to obstacles for test 3*

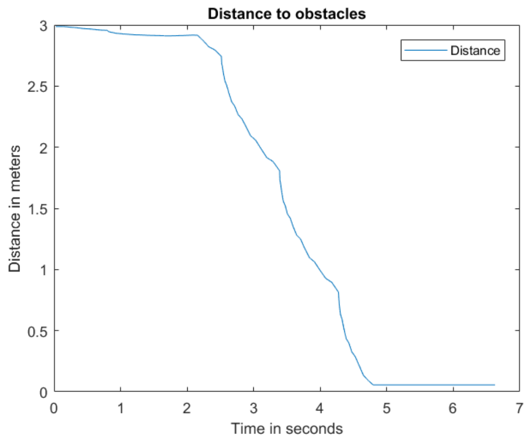


(d) *Illustration of distance to obstacles for test 4*

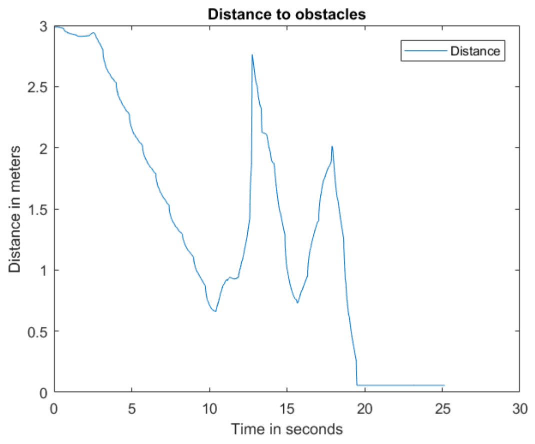
Figure 25: Illustration of distance travelled for test 1-4, where the robots visits all 14 rooms while collecting marbles



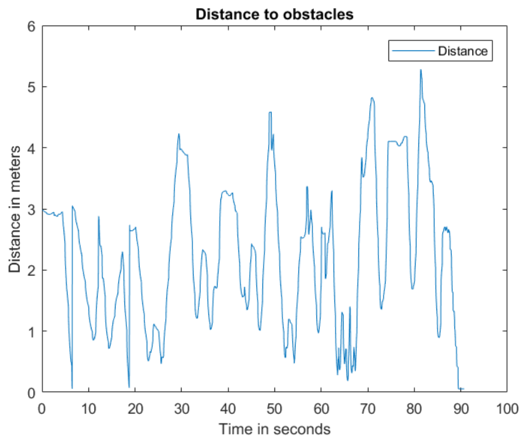
(a) *Illustration of distance to obstacles for test 5*



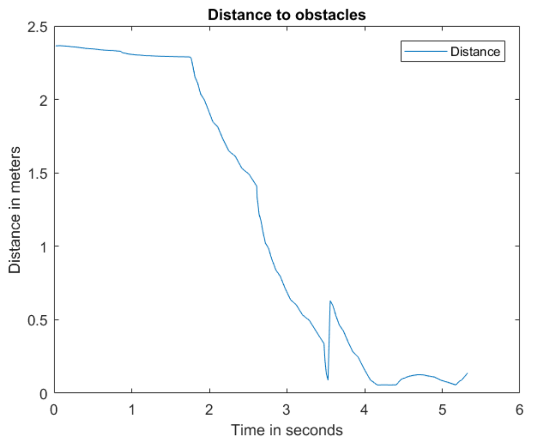
(b) *Illustration of distance to obstacles for test 6*



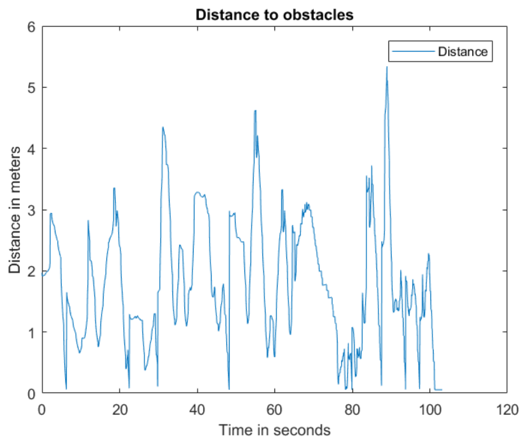
(c) *Illustration of distance to obstacles for test 7*



(d) *Illustration of distance to obstacles for test 8*

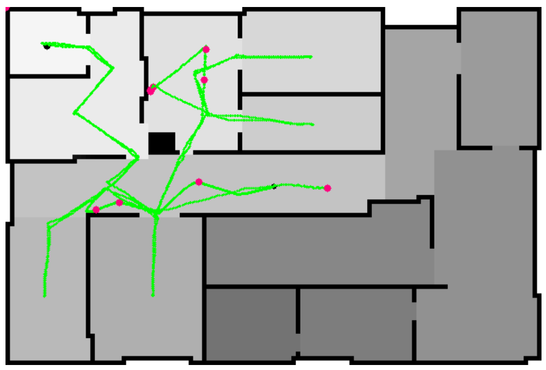


(e) *Illustration of distance to obstacles for test 9*



(f) *Illustration of distance to obstacles for test 10*

Figure 26: Illustration of distance to obstacles for test 5-10


(a) Illustration of distance travelled for test 1, where the robots visits all 14 rooms while collecting marbles

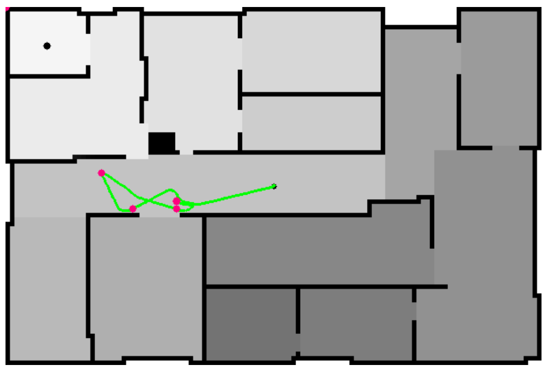
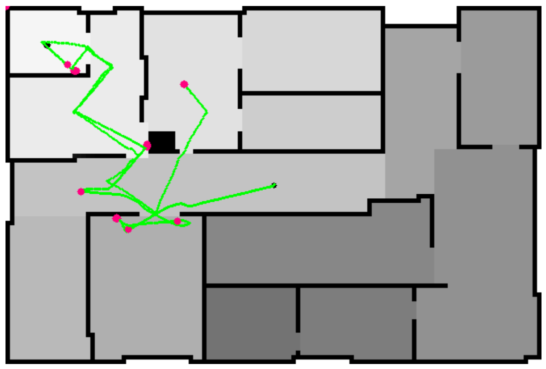
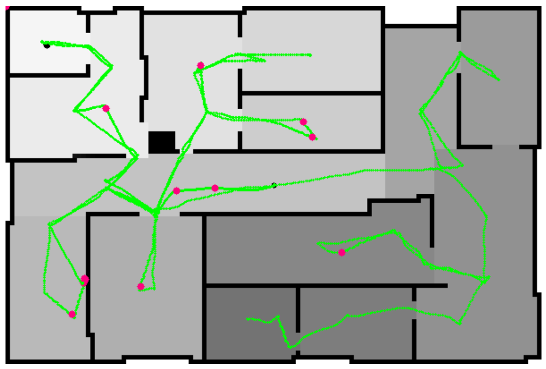
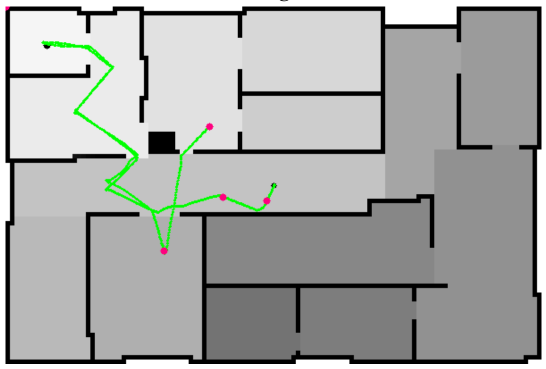
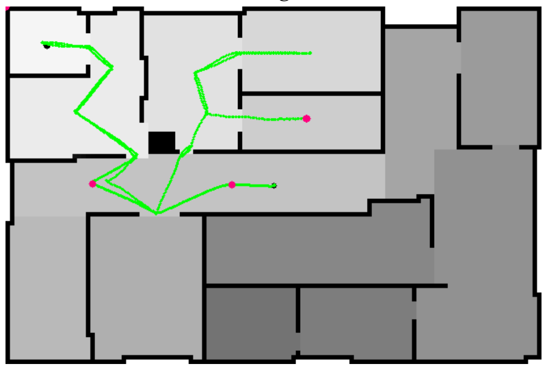

(b) Illustration of distance travelled for test 2, where the robots visits all 14 rooms while collecting marbles

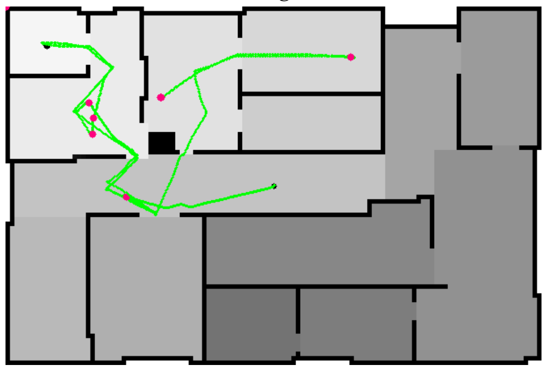
Figure 27: Illustration of distance travelled for test 1-2, where the robots visits all 14 rooms while collecting marbles

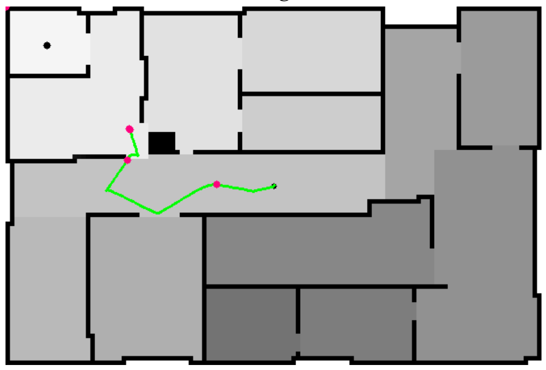

(a) Illustration of distance travelled for test 3, where the robots visits all 14 rooms while collecting marbles

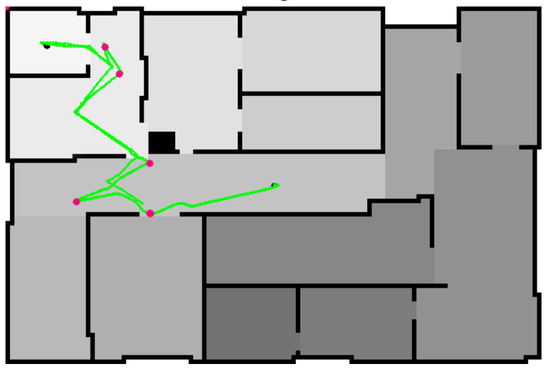

(b) Illustration of distance travelled for test 4, where the robots visits all 14 rooms while collecting marbles


(c) Illustration of distance travelled for test 5, where the robots visits all 14 rooms while collecting marbles


(d) Illustration of distance travelled for test 6, where the robots visits all 14 rooms while collecting marbles


(e) Illustration of distance travelled for test 7, where the robots visits all 14 rooms while collecting marbles


(f) Illustration of distance travelled for test 8, where the robots visits all 14 rooms while collecting marbles


(g) Illustration of distance travelled for test 9, where the robots visits all 14 rooms while collecting marbles

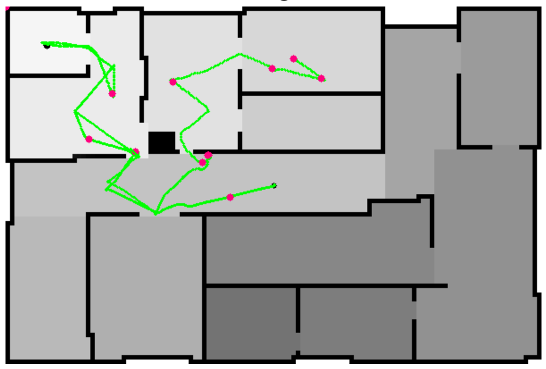

(h) Illustration of distance travelled for test 10, where the robots visits all 14 rooms while collecting marbles

Figure 28: Illustration of distance travelled for test 3-10, where the robots visits all 14 rooms while collecting marbles

A.8.4 Conclusion