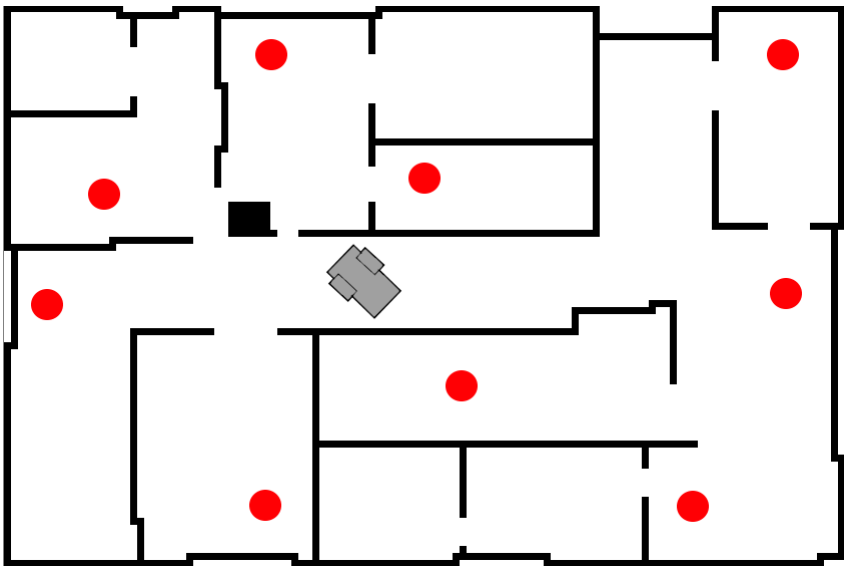
Syddansk universitet, Civil Robotics Engineer, 5. Semester

RCA PROJECT 5 - MARBLE FINDING ROBOT

6 December 2020

Authors:	**Study-ID**
Frida Bech Schrøder	frsch18
Nicoline Thomsen	nthom18
Mikkel Skov Maarssø	mimaa18



Abstract

In this project multiple aspects with regards to control, navigation and search of a two-wheeled robot in a virtual environment is explored. Solutions to part of the issue of having the robot navigate through the environment searching for marbles which are scattered randomly in the environment in three different disciplines are looked into. The areas are robotics, Computer vision and Artificial intelligence. In robotics a method for controlling the robot is implemented. Localization of the robot is attempted using a particle filter. A map of the environment in which the robot interacts is divided into rooms using trapazoidal decomposition and a strategy for visiting all rooms in an efficient ways is determined. In the computer vision part of the project marbles are detected in the environment by applying the Hough transform on the input from the robots camera. Based on the detection of marbles using Hough, marbles are localized in the environment, by determining the distance between the robot and the marble using camera geometry. In the artificial intelligence part an approach for obstacle avoidance using fuzzy control is explored. Moreover Q-learning is applied to an abstraction of the workspace to determine an efficient way of navigating the environment with the objective of collecting as many marbles, which are scattered with varying probability throughout the environment, as possible.

Contents

1 Project Description

1.1 Introduction

The objective of this project is to explore solutions to various components of the overall problem of control and navigation of a two wheeled robot in a simulated environment, with the goal of collecting marbles which are randomly distributed in the surroundings. The environment and the robot are simulated in the simulation program Gazebo B5. The simulation can be found in A9 A map of the environment is available, this can be found here A8. Three different disciplines are explored with regards to different aspects of the project, these are robotics, computer vision and artificial intelligence. The requirements for each discipline can be seen under subsection 1.3. The different aspects of the project are solved in different ways and are ultimately thought to be combined in multiple ways to solve the problem of allowing the robot to find marbles in the environment but a combination of the various solutions for the goals described in subsection 1.3 is not within the scope of this project.

1.2 Project definition

To make solutions for different parts of the issue of allowing a two wheeled robot to efficiently search for, detect and collect all marbles in a virtual environment.

1.3 Specifications

Robotics:

1. Control for the two-wheeled robot.
2. Localization of the robot in the environment assuming a known initial position.
3. Efficient strategy to visit all rooms of the environment.

Computer vision:

1. Marble detection based on Hough transform
2. 3D localization of found marbles using camera geometry and performance evaluation of the method

Artificial intelligence

1. Fuzzy control for local obstable avoidance.
2. Apply Q-learning to find effective navigation strategies.

2 Roadmap and strategy for visiting all rooms

Implementations of an efficient strategy to navigate, and thereby allow the robot to collect marbles, in the environment in which the robot acts is a requirement. A map of the environment is supplied as described in subsection 1.1. The map, which can be seen in figure 1 , consist of a connected white area which is the the free space in which the robot can move, and black areas which are the obstacles. The aim is to to visit a combination of points in the map, which ensures that all the white area will be visible if the robot rotates about itself in each of the points. All these points will henceforth be refereed to as nodes. Each node is placed in the center of what is defined as a room. All rooms are rectangle and thereby figures where any point inside the figure can be reached from another point inside the figure without having to go outside the figure.

The chosen approach is to locate all rooms, and then use a strategy to determine the order in which the robot should visit the rooms to efficiently visit all rooms on the map. The location of the rooms is described in section 2.1. The strategy to determine the room visitation order is described in subsection 2.2.

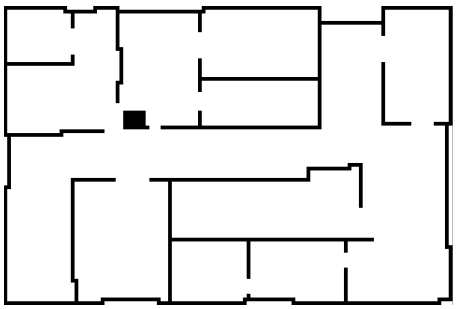


Figure 1: Map of robot environment

2.1 Localisation of rooms

Trapezoidal decomposition is used to locate the rooms on the map of the environment, and the implementation is based on B2 page 162. This consists of three parts, corner detection, obstacle detection and generation of rooms. The corner detection is done using a vertical sweep lines which moves from left to right through the map and detects all corners. This is further described in subsection 2.1.1. Obstacle detection utilizes the corners detected in the previous step and generates lines upwards or downwards from each corner until they encounter an obstacles. The point where the line encounters an obstacle is detected as another corner. The obstacle detection is described in detail in subsection 2.1.2. Generation of rooms consists of making rooms from the detected corners, this is explained in subsection 2.1.3.

2.1.1 Corner detection

As explained in section 2, the free space on the map is white and the obstacles are black. The detection algorithm detects the corners of the black obstacles. A black corner fulfills one of two conditions, either it consists of 4 black pixels, including the center pixel and 5 white pixels, or it consists of 1 white pixel and 8 black pixels once again including the center pixel. The corner is registered as the center pixel. These conditions can be seen in figure 2.

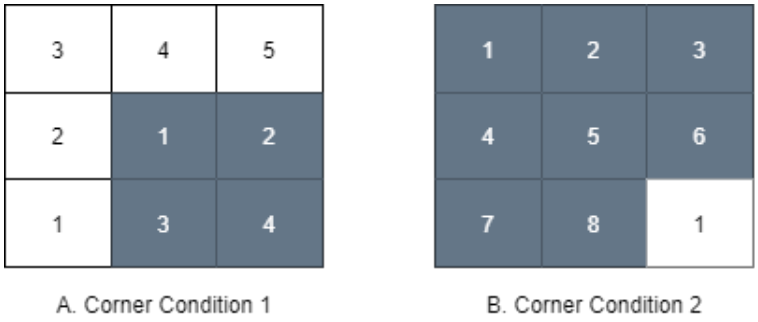


Figure 2: Condition for corners

A vertical sweep line checks all black pixel as it moves across the map. At each black pixel it and its surrounding 8 pixels colors are counted. Whenever one of the two corner conditions are met it is registered as a corner. The corners detected on the map of the environment can be seen on figure 3A.

2.1.2 Obstacle detection

For the corners detected in subsection 2.1.1, a line is generated upwards, downwards or not at all, depending on the corner type, which is described in 2.1.1. If the corner satisfy corner condition one a line is generated from the center pixel either upwards or downwards depending on the orientation of the corner. For the corner shown figure 2 a line starting at the white pixel 4 and continuing upwards until another obstacle is met would be created. If the corner satisfy corner condition two, no line is created. At the point where the line intersects with an obstacle another corner is created. The result using the corner detection can be seen in figure 3b.

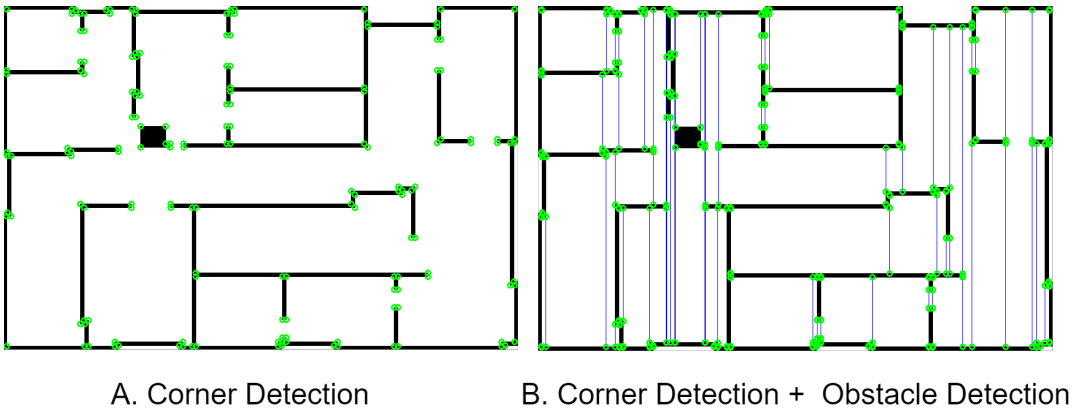


Figure 3: all corners detected with the two different methods

2.1.3 Room detection

After the corner detection in subsection 2.1.1 and detection of additional corners in subsection 2.1.2 the corners is to be matched up to constitute the correct rooms. A room is made up of four corners which makes up two sets of parallel lines both vertically and horizontally, this means that the two of the four corners the same x-value, and that the other two have another x-value which is also the same between the two corners. Likewise with y-values. A room contains as previously stated no obstacles, and the entire area within the room is thereby white. To determine which corners match up all corners are sorted in a matrix by y-value and then by x-value. This means that corners with the same y-value are placed on the same row in the matrix sorted by their x-value. This principle is illustrated in figure 4.

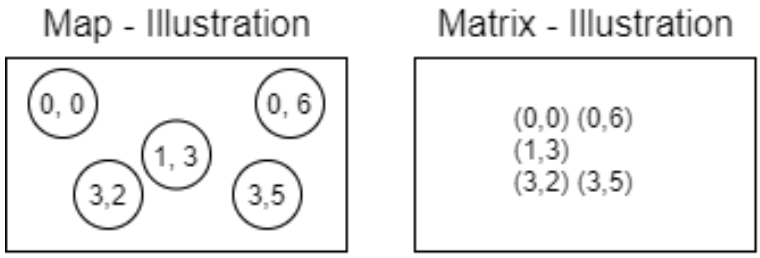


Figure 4: Visualisation of matrix calculation

Using the matrix sorted by x-value and then y-value it is determined for each corner if a rectangle exists where the corner in question is the upper left corner. This is done by looking through the rows, starting from the one below where the corner is placed, looking for a corner with the same x-value. Each time a corner which meets this requirement is found the corner to the right of the original corner, meaning the next corner in the same row, and the corner to the right of the new corner are compared to see if they have the

same x-value, if so it is a possible room. This is done until a possible room is found or all corners with matching x-values are tested.

If a possible room is found the diagonal between the upper right corner and the lower left corner are tested for collisions using a Linear Bezier curve represented as a line, the math for this can be found described in B6. If there is no collision the rectangle is not. The formula for a Bezier curve used can be seen in equation (1).

$$P(t) = (1-t)P_0 + tP_1, 0 \leq t \leq 1 \tag{1}$$

The Bezier line generates 100 evenly distributed points between the upper left corner and the lower right corner, which are tested for collisions. Whenever the Linear Bezier Curve encounter a black pixel it will be referred to as a collision in the rest of this paper. If any of the points along the Linear Bezier Curve is a collision the rectangle is not room. If no collisions were found the room is accepted. The center of each room is determined and is refered to as a node. If two nodes are within 25 pixels of each other, one of them will be removed. The result of the locations can be seen in figure 5, the yellow lines show the diagonal lines in the rooms and the blue circles show the location of their nodes, which is equal to the center of gravity.

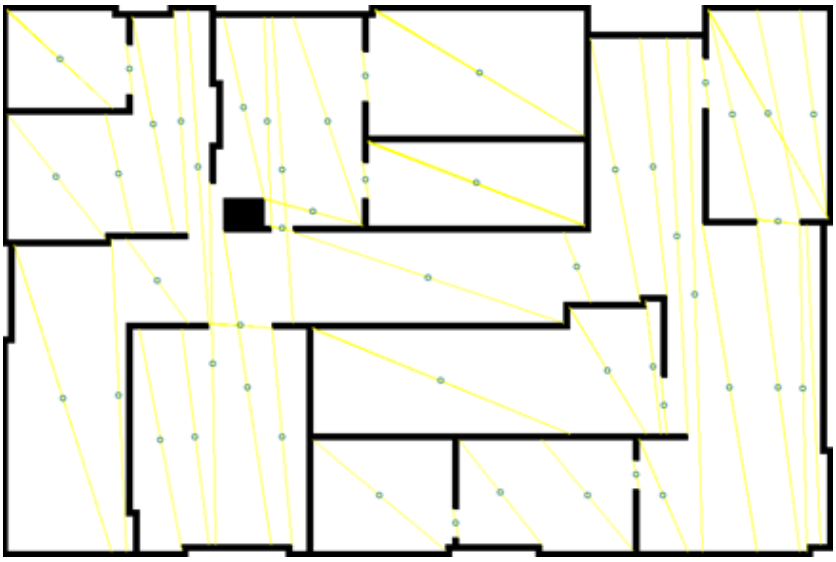


Figure 5: Rectangles found in map

2.2 Strategy for making an efficient route through the map

The objective of the strategy described in the following is to determine an efficient route to visit all rooms on the map. The strategy for the ordering of the rooms is based on the assumption that all rooms are connected to at least one other room. All rooms must be

visited to ensure the entire free space is covered. Each room has a node in the center and it is the locations the path is planned between, the process of finding the room and their nodes are described in subsection 2.1.3

From a start position, which can be anywhere in the free space, the route is calculated. The start position is node 0 on the route, this point is connected to the nearest existing node on the map.

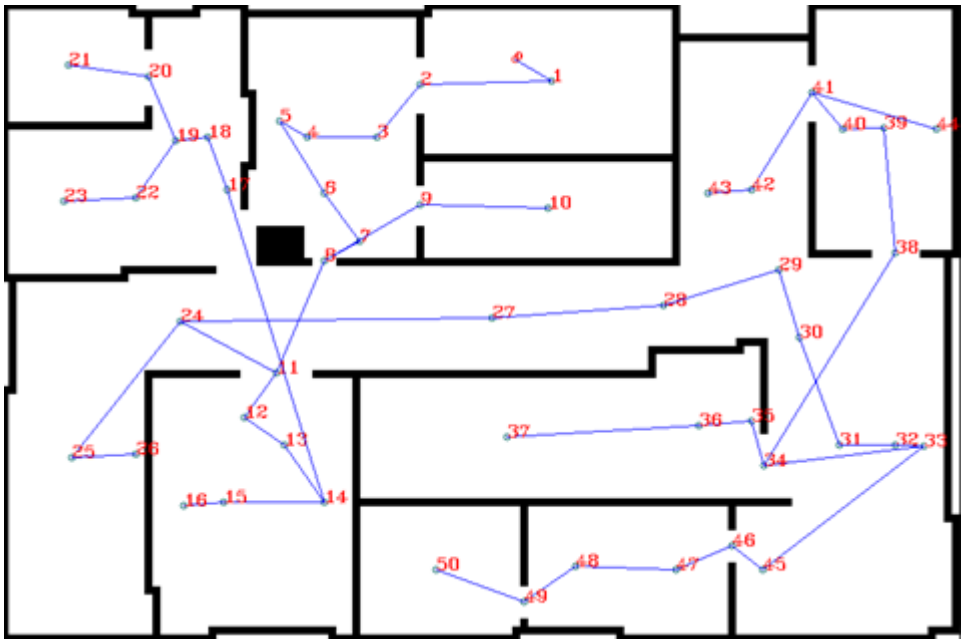


Figure 6: Robot room visiting strategy

When the first node is reached from the start point, this becomes the current node, all remaining nodes are added to the route one by one. When a node is added it receives a number one higher than the previous node which determines the order with which they will be visited. The nodes are added by the following method: The algorithm searches for all not yet visited node which are reachable from the current node without collision. If multiple nodes without collision is available, the closest is chosen as the next node. If no new node is available from the current node the algorithm goes back through the nodes in the route in descending order and determines if any nodes which has not been visited before are available, this process is referred to as backtracking in the rest of the paper. Once a node is chosen as the next node in the route, it becomes the current node and the process is repeated until all nodes are assigned a number. The route determined by the strategy used and applied on the map used in the project, can be seen on figure 6, the number represents the order in which the nodes should be visited, starting from 0 and ending with 50. The code for room detection and strategy can be found in A12.

3 Control of the robot

Control of the two-wheeled robot in the environment is requirement for the project. The robot has two inputs; direction and speed. The speed, meaning it's forward/backwards momentum, is given as $[m/s]$ and can be positive for forward motion and negative for backwards motion. The direction is the angular velocity, yaw, and given by $[rad/s]$, the direction can be positive or negative, positive values rotates the robot to the right and negative values rotates the robot left.

A controller class is implemented which changes direction and speed in order to achieve a desired outcome. This controller is described in subsection 3.1

3.1 Implementation of control

There are two functions, one which changes the orientation of the robot and one which moves the robot. In the current implementation the robot is controlled by execution of the functions before the main loop. Both functions take three inputs "Change of angle" [degree]/"Distance to move" [m], "Time to perform change" [s] and "Time until start" [s]. The function ChangeSpeed gives the robot an input in $[rad/s]$ depending on the distance and allotted time which is specified in the function call. The change is performed at a certain time after the main loop is entered which too is specified as an input to the function. Likewise the function ChangeDirection changes the direction according to the inputs. An example of two function calls to control the robot can be seen in figure 7. Line 1. makes the robot move 80 degrees to the left over four seconds, resulting in a angular velocity of 20 [degree/s]. The code will execute 5 seconds after the program starts, this is determined by the last 5 in the function call. Likewise, for the second line will move the robot 6 meters forwards over five seconds, resulting in a velocity of 1.2 [m/s], the instruction will start 10 seconds after program start. The code for control can be found in A11.

```
1.  control.ChangeDirection(-80, 4, 5);
2.  control.ChangeSpeed(6, 5, 10);
```

Figure 7: Control Code Example

4 Localization

It is a requirement of the project to performe localization of the robot to determine where on the map the robot is located. In this project the robots initial position is of the robot is known. The robot has a LIDAR scanner which relays information about the surroundings. Meaning the robot is able to interpret the environment based on the input from LIDAR sensors. In this project a localization scheme based on a particle filter.

To performe the localization a factor to scale between the gazebo environment and the map is nessesary, this is desribed in subsections 4.1. The particle filter is described in subsection 4.2.

4.1 Scaling factor between gazebo environment and map

A scale between the gazebo environment and the picture map is necessary. An estimate of the scale is achieved by first obtaining the gazebo map size, by driving the robot to each corner and read the coordinates of gazebo in the GUI, and using this as the size of the gazebo environment. The scale factor is calculated as the difference between the size of the gazebo environment and the dimensions of the map. The equation for determining scale factor can be seen in equation (2).

$$ScaleFactor = \sqrt{\frac{map.rows * map.size}{((\sqrt{gz\vec{2}.x^2} - \sqrt{gz\vec{1}.x^2}) \cdot (\sqrt{gz\vec{2}.y^2} - \sqrt{gz\vec{1}.y^2}))}} \tag{2}$$

The scaleFactor between the gazebo environment and the used map is found to be 11.31.

4.2 Particle Filter

A particle filter is used to estimate the position of the robot each time step. This is done by creating a number of particles every timestep. Each timestep half the particles, the part which resembles the robot best is kept and as many is new particles is created which resembles the kept particles with some noise added. By doing this continually the average of the particles converges to the position of the robot. This implementation is based on the principles from B2 page 316 and B3. In the project each created particle consists of a position and a vector. The particle position corresponds to the position of the robot and the particle vectors orientation correlates to the orientation of the robot, likewise with length of the particle vector and input from the center LIDAR of the robot. The center LIDAR is the input from the LIDAR scanner directly forwards seen from the robots perspective. A flow chart of the particle filter can be seen on figure 8. When the particle filter is first applied it goes from box 1, through "yes" to the 2 box. Here particles are created and

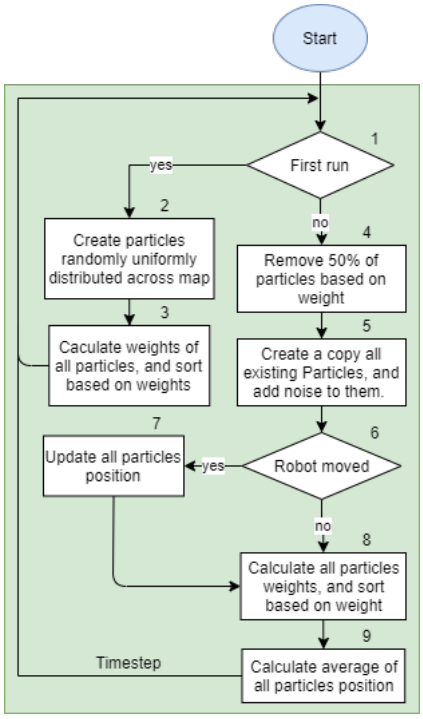


Figure 8: Flow Chart - Particle Filter

uniform randomly distributed across the map. After this, as seen in box 3, the weight of all particles are calculated. The weight of a particle depends on the length of the particle vector compared to the length of the center LIDAR of the robot, the smaller the difference between the particle and the center LIDAR the close a weight to one, one being an exact match to the input from the center LIDAR. Once the particles are assigned a weight they are sorted according to distance from one. After the sorting the program waits until the rest of the current timestep has passed after which it enters the box 1 again. This time it passes through the "no" and goes to box 4 where half the particles are deleted. The half which are furthest from one are deleted, as the other half more closely resembles the actual robot. In the next step, box 5, a copy of all kept particles is created, to every new particle, which is a exact copy of one other particle noise is added. The noise consists of a random change in rotation and position. The rotation is changed by a random degree between 90° and −90°. This is done by applying a rotation matrix seen in equation (3)

$$\begin{bmatrix} x^{\circ\prime} \\ y^{\circ\prime} \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x^{\circ} \\ y^{\circ} \end{bmatrix} \tag{3}$$

The matrix implementation is based on B4. The particles position is changed by a random distance between 40 and -40 pixels in the direction of both the x-axis and the y-axis. What happens next is dependent on whether or not the robot has moved, box 6, if the robot has

moved an extra step, box 7, happens before box 8. When the robot moves, the particles have to move as well. The move of the particles are based on the input the robot in the timestep and the estimated length of the timestep, this way of moving the particles will henceforth be called transformation of particles. If the robot is rotated a rotation is applied to the particles by using the rotation matrix in equation (3), using the estimated angle the robot has turned. If the robot has moved forwards or backwards particles are updated translated in the direction of the particle vector. If both a rotation and a translation has happened the particles are first rotated then translated along the new direction of the particle vector. When the particles have been moved or this step is skipped altogether because the robot did not move, the weight of the particles are calculated in the same way as in box 3. After this, in box 9, the average position of all particles are calculated, this is the estimate of the robots position. The code for the particle filter can be found in A10.

In this project 500 particles is used. The particles distribution when they are first created can be seen in figure 9A and the particles after 423 timesteps can be seen in figure 9B.

A. Particles First Created B. Particles after 423 timesteps

Figure 9: Particle Filter resutls

5 Fuzzy control for obstacle avoidance

The purpose of the fuzzy control is obstacle avoidance. By utilizing the sensor input from the LIDAR scanner the objective is to avoid collision with the robot and the walls or other obstacles in the map. A fuzzy control system is a system which takes a crisp input, fuzzifies this value by mapping it to a fuzzy value, where-after the fuzzy logic is applied and the result of this is again defuzzified to a crisp value. The input from the LIDAR scanner is converted into inputs which are appropriate for the control system, this is described in section 5.1. The inputs and outputs of the fuzzy control system is described in subsection 5.2 and - 5.3. The theory for creating a fuzzy controller is from the book B8.

5.1 Input from the robot

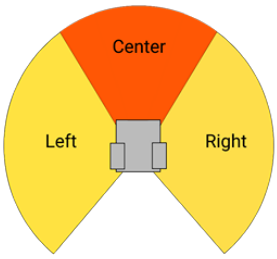


Figure 10: The LIDAR input split into three sections: left, right and center.

The simulated robot in the environment has a LIDAR sensor which is used to determine how the surroundings are at any time. The robots LIDAR sensor covers about 3/4th of the circle surrounding the robot, the part which is not covered is behind the robot as can be seen in figure 10. The LIDAR scanner returns an array of distances for different angles of the robot. This spectrum of distances is split up into three parts; center, left and right. The center part of the LIDAR input are illustrated with orange in the figure, this is about 1/5 of the entire range. The input is split up in 2 parts, a left partition and a right partition, which are illustrated in the figure. These sections are used to determine if there is anything if front of, to the left of or the the right of the robot. For each part of the spectrum all the distances are inspected to determine the shortest one, this is the closest obstacle to the robot in the direction in question. These three values are the crisp input values for the fuzzy controller.

5.2 Fuzzy inputs

The crisp values mentioned in the previous subsection are fed to the fuzzy controller. This controller has three corresponding fuzzy sets; ObstacleCenter, ObstacleLeft and ObstacleRight. For each pair of input and fuzzy set, the input is mapped to a membership

function in that universe of discourse. The membership functions are defined as 'veryClose', 'close' and 'far'. An illustration of the fuzzy sets can be seen in figure 11 The input distance is a value from 0 - 10, which is normalized. This processes is fuzzification of the inputs.

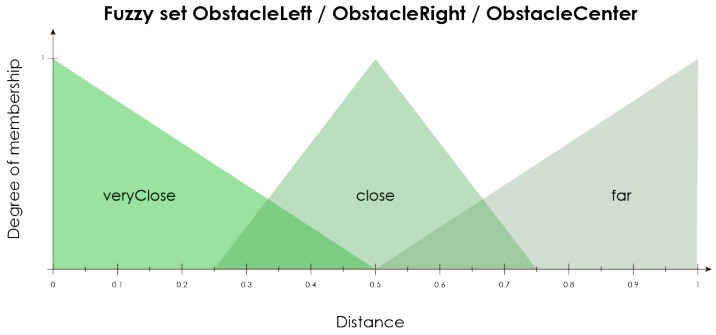


Figure 11: The universe of discourse for the three input fuzzy sets: ObstacleLeft, ObstacleRight and ObstacleCenter.

5.3 Fuzzy outputs

There are two outputs in the fuzzy control system, these control the speed and direction of the robot. The speed is dependent on the closeness of the objects around the robot. The speed output has three output levels which can be seen in figure 12. The ObstacleCenter input dictates the speed. As seen from the figure, brake will stop the robot before reaching an obstacle. The direction of the robot is determined from the left and right input to the robot, if there is an obstacle left of the robot it will turn right to avoid a collision. The robot will try to balance the left and right distance to the object to be the same, resulting in the robot driving in the middle of a straight hallway. The directional output has six levels which can be seen in figure 13. An output of 0.5 means that the robot drives forward, meaning it has a rotation of 0. Values lower than 0.5 rotates the robot towards the left, likewise do values above 0.5 rotated the robot to the right.

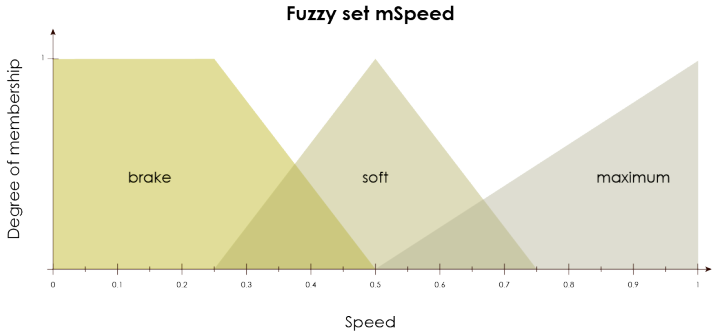


Figure 12: The universe of discourse for the output fuzzy set that controls speed.

The rule base for mSpeed is:

```
if obstacleCenter is veryClose then mSpeed is brake
if obstacleRight is close or obstacleLeft is close or obstacleCenter is close then
mSpeed is slow
if obstacleRight is far or obstacleLeft is far or obstacleCenter is far then mSpeed
is maximum
```

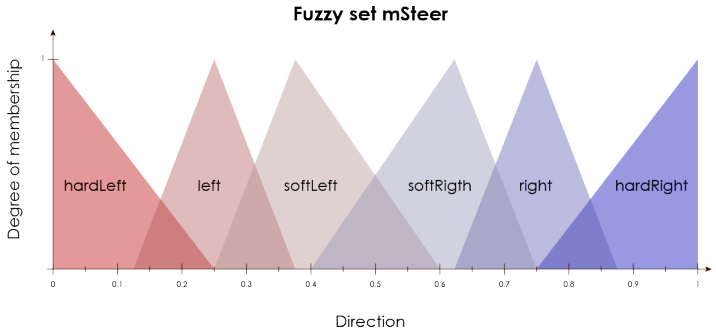


Figure 13: The universe of discourse for the output fuzzy set that controls direction.

The rule base for mSteer is:

```
if obstacleLeft is veryClose then mSteer is hardRight
if obstacleLeft is close then mSteer is right
if obstacleLeft is far then mSteer is softRight
if obstacleRight is veryClose then mSteer is hardLeft
if obstacleRight is close then mSteer is left
if obstacleRight is far then mSteer is softLeft
```

5.4 Performance of the fuzzy control system

A test of the performance of the implemented fuzzy control system is described in journal A7. The test found that the robot performed differently than expected as it did in fact not actively turn away from walls as anticipated. However the robot did not collide with any obstacles in the tests.

6 Navigation using q-learning

6.1 Introduction

The objective is for the robot to collect as many marbles as possible limiting the movements needed. Two approaches to solve this is explored in this project, including this one which utilizes reinforcement learning. The other navigation strategy is documented in section 2 and uses a drastically different approach. Q-learning is a type of reinforcement learning. Q-learning uses a Q-table to store an estimate of each possible action for all states the agent can be in. These estimates are based upon the rewards the agent has received taking the action earlier, and to some extend the estimates of the best actions the agent could choose as the next action, when the action was originally performed. The Q-table is updated each time an agent takes an action. The strategy used in this implementation to choose the next action is the epsilon-greedy strategy, how this is done is described in subsection 6.5.2. Furthermore there are two other factors which affect how the agent learns, these are the discount factor, γ, and the learning rate, α. The discount factor determines how much emphasis is put on the best action the agent can perform from the next state. Learning rate is how much the Q-value is updated with new knowledge. Both learning rate and discount factor are values between 0 and 1.

6.2 Workspace

The Q-learning is performed on an abstraction over the "bigworld.world" work space. This abstraction can be seen in figure 14. Each of the rooms in the road map has at least one node, symbolised with a red dot. The number beside the nodes are the index of the node. The red lines between the nodes are paths. The amount of paths leading away from a certain node is the amount of actions the agent can take from that node. As can be seen on the image of the road map the amount of paths leading from a node, and by that the possible actions, varies from one to at the least five at the most. The actions are an abstraction of the movements performed by the actual robot i the gazebo environment, meaning that when the agent i located on node 2 and chooses the action correlated with node 4, it is a parallel to the real robot moving to the room containing node 4 to look for marbles.

6.3 Marbles

The road map, which is descried in section 6.2 and seen in figure 14, is a number of interconnected nodes. These nodes are equivalent to the rooms that the actual robot would visit, and marbles are randomly scattered throughout these rooms. Marbles reward the robot for looking in the rooms, but only once. If the robot returns to a room for a

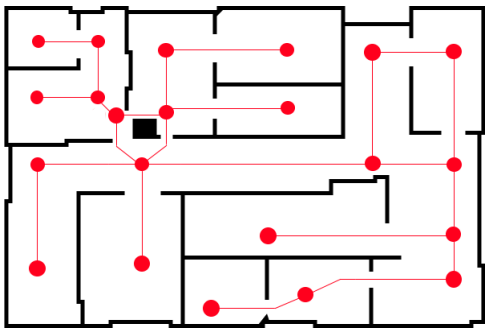


Figure 14: Road map used in Q-learning

second time in the same iteration it will receive a reward of 0, as "the marbles" are collected the first time a room is visited. Rewards larger than 0 will henceforth be referred to as marbles or rewards of 0 as no marbles. The location of the marbles are generated randomly each iteration of the code. To correspond to some rooms having a higher or lower chance of containing marbles some rooms will generally be assigned larger or lower rewards than others. There are three possible rewards other than 0, these are 5 for rooms with lower chance of marbles, 10 for rooms with normal chance of marbles and 15 for rooms with a high chance of marbles. Notice that the chance of a reward being assigned a particular room with the high reward of 15 is no larger than that of a particular room with a the normal reward of 10, only the reward is larger when it does. The amount of marbles put on the map varies but the total reward is a constant of 100. When distributing rewards a room is randomly chosen and given the reward the room is worth. If the amount of rewards left to distribute is lower that the score of the chosen room to receive that reward, the room is given what is left, resulting in that room being worth a lower score, however still be worth some. Once a room has received a reward it cannot receive another, meaning that a room's reward won't surpass the score it is worth.

6.4 Q-table

For deciding which action is the best the agent has a Q-table. This Q-table contains a value for each possible action in every possible state. The state includes the node in which the robot is located and whether or nor every other node has been visited, this is relevant to the decision the agent is making. This results in a rather large amount of states. For each of the nodes, of which there are 17, there is a state for every combination of visited/not-visited of the other nodes. This results in 2^{16} different states for each node. For the 17 nodes there is a total of $17 \cdot 2^{16} \approx 1$M states. For each of these states there is an estimated reward for each possible action which can be chosen in the state, the actions are equal to the adjacent nodes, this is described in subsection 6.2. The estimates are all stored in

the Q-table which is updated when relevant, how the values are updates is described in subsection 6.5.3. All the estimates are zero when the Q-table is created.

6.5 The process of Q learning

With a roadmap with rewards, described in subsection 6.2 and an associated Q-table with the appropriate states, described in subsection 6.4 the agent can act in the environment to learn about it and update the Q-table with this knowledge. The process consist of three pars, firstly choose next action to take, is described in subsection 6.5.1, then the action is taken and the reward is observed, this is described in subsection 6.5.2, lastly the Q-table is updated based on received reward and the next state is set as state, this is described in subsection 6.5.3.

6.5.1 Choose action A

The next action A is chosen after the epsilon-greedy principal. There are two different options for how the agent chooses which action to take, either it takes a exploitative choice or an exploitational choice. The exploration is used to learn more about the environment, this is done by taking a random action. The expoloitational action uses the knowledge already gathered by the agent to take the best possible action from the current state, meaning it chooses the action with the highest estimated reward. How these values are estimated is explained in subsection 6.5.3. The balance between exploration and exploitation is determined by a variable epsilon, which is between 0 and 1, where 0 would be only exploration and 1 would be only exploitation. An epsilon of 0.1 is used in the project.

6.5.2 Taking action A

When an action is chosen, described in the previous subsection, the agent takes this action and gets the reward associated with the action. This reward may be 0, 5, 10 or 15, how the rewards are distributed throughout the map is explained in subsection 6.3.

6.5.3 Update Q-value

When an action is taken and the reward for taking this action is observed and the Q-table can be updated with this new knowledge. The Q-table contains estimates for the expected return for taking an action. This estimate is based on which rewards the agent has received from taking the action before and it is also based on the estimated value of the best action the agent can take from the next state.

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a Q(S', A) - Q(S, A] \tag{4}$$

$$S \leftarrow S' \tag{5}$$

The equation for calculation a new estimated value for an action, A, given a state, S, can be seen in equation (4), equation (4) and equation 5 are from slide Q-learning: Off-Policy Control in B7. There are two parts to this equation, the first part $Q(S, A)$ is the old estimated value and the second part is the updated part. In the update a constant value α determines how much the estimate is changed based on the new information, if α were to be 1 all previous knowledge would be cleared and the entire estimate would be made from the latest experience. Inside the brackets there is the reward received from taking the action and then there is $\gamma max_a Q(S', A)$ which is the discount factor γ multiplied with the expected reward from the estimated best action from the next state. The discount factor determines how much the next action is taken into account when calculating the expected reward for an action. The discount factor is a value between 0 and 1. A test has been executed which is described in journal A6, which found that the agent made the optimal choices with $\alpha = 0.25$ and $\gamma = 0.85$, these values are used in the implementation of the Q-learning which can be found in A13.

7 Computer Vision

The robot has a camera which relays information about the environment. There are marbles randomly scattered throughout the environment which the robot must be able to detect. Computer vision is used to detect the marbles. It is a requirement that Hough Transform is used as the detection algorithm, and the marbles are localized in the 3D space by utilizing the camera geometry. There are some distortion on the camera which is rectified before the image is used to detect the marbles, the rectification of the distortion is described in subsection 7.1. After rectification of camera distortion Hough transform is used to determine if and if so where marbles are located on the image, this is explained in subsection 7.2. Lastly localization of marbles from the camera geometry i described in subsection 7.3.

7.1 Rectification of distortion in the camera

In a real-world application a camera is not perfectly made, and the image will have some distortion. It is therefore desirable to calibrate the camera and remove this distortion. The rectification of this distortion is performed by using the OpenCV library, which uses the intrinsic parameters of the camera and a known distortion. The camera distortion is described by five distortion coefficients, which can be estimated by calibrating the camera, usually by scanning a known object.

In this project the camera is a simulated model, and as such the distortion too is simulated. The exact distortion parameters of the camera can be found in the `model.sdf` file defining the camera in the Gazebo folder. These are used in the project as the cameras distortion parameters.

A pinhole camera model is used to model the camera. Two types of distortion is present in the pinhole camera model, these are radial and tangential distortion. These types of distortion are described in B1 in chapter 5.6.

$$i' = i(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \qquad j' = j(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{6}$$

The radial distortion is described by equation (6). Where $f(i, j)$ is the origin of the image at the optical axis and the corrected image $f'(i', j')$. $r = \sqrt{i^2 + j^2}$ and k_1, k_2 and k_3 are distortions coefficients.

$$i' = i + (2p_1 ij + p_2(r^2 + 2i^2)), \qquad j' = j + (2p_1 ij + p_2(r^2 + 2i^2)) \tag{7}$$

The tangential distortion in described by equation (7). Where p_1 and p_2 are distortions

coefficients.

The distortions coefficients can be seen in equation (8).

$$[k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3] = [0.25 \quad -0.12 \quad -0.00028 \quad -0.00005 \quad 0.0] \tag{8}$$

$$\begin{bmatrix} i \\ j \\ 0 \end{bmatrix} = \begin{bmatrix} f_i & 0 & c_i \\ 0 & f_j & c_j \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{9}$$

To remove distortion in the camera the intrinsic parameters of the camera are also needed, they are listed in the camera matrix on the form seen in equation (9), where x, y and z are a point in the 3D world, i and j are a point in the image plane, f_i and f_j is a combination of the focal length, c_i and c_j are the optical centre. This theory is from B1 chapter 2.1.

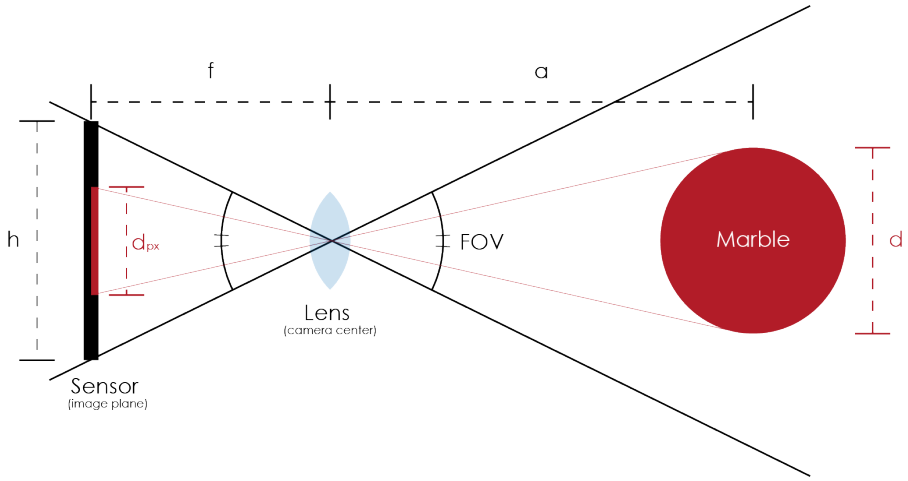


Figure 15: Illustration of a marble projected on the image plane of the camera. The camera is modelled after the pinhole model.

The optical centre can be found in the file `model.sdf` where the distortion parameters were found earlier, but the focal length is not. Instead the horizontal field of view and the image size is given. Horizontal field of view will henceforth be referred to as FOV. An equation to calculate the FOV can be written using the trigonometric identity $\tan(\theta) = \frac{Opposite}{Adjacent}$ and the relationship between FOV and the image plane which is illustrated in figure 15. Rewriting this equation results in an expression for the focal length f, which can be seen

in equation 10.

$$\begin{aligned} FOV &= 2 \cdot \tan^{-1}\left(\frac{h}{2f}\right) \\ \Leftrightarrow f &= \frac{h}{2 \cdot \left|\tan\left(\frac{FOV}{2}\right)\right|} \end{aligned} \tag{10}$$

h is the sensor width in pixels, and is 320. The FOV is 1.047 rad. Inserting the values in equation 10 results in equation (11).

$$f = \frac{320[px]}{2 \cdot \left|\tan\left(\frac{1.047[rad]}{2}\right)\right|} \approx 277.2[px] \tag{11}$$

The size of the sensor is in in pixels instead of the usual mm. This results in the focal length also being represented in pixels.

The camera matrix with the numeric values from `model.sdf` and the focal length calculated in equation (11) is seen in equation (12).

$$camera \quad matrix = \begin{bmatrix} 277.2 & 0 & 160 \\ 0 & 277.2 & 120 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

The distortion coefficients and the camera matrix is passed as input for OpenCV's `cv::undistort()` function to remove the distortion from the image.

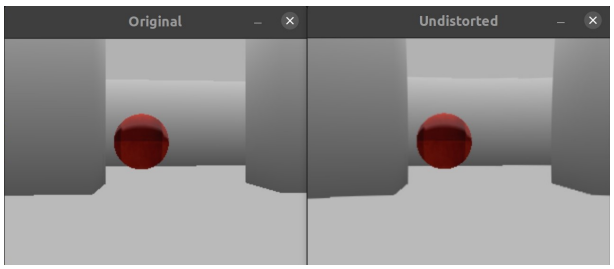


Figure 16: Comparison of the original image before any undistortion (left) and the image with the distortion removed (right)

In figure 16 the original of an image is shown to the left and the same image after undistortion is shown to the right. The undistorted image might appear more blurry than the original, this is due to resampling performed by `cv::undistort()`, which by default is done using bilinear interpolation. This has not been changed for this project.

7.2 Hough Transform

Hough transform is to be used to detect the marbles in the image. The marbles are spheres which means they are projected to circles in the 2D image plane obtained through the camera of the robot. The Hough Transform is a line detection algorithm, which can be modified to detect circles instead of lines. It takes a binary image as input with the edges from in the original image.

The idea behind the Hough transform is shortly explained for lines whereafter the application using circles is described. For every edge point all lines, which goes through the point, is mapped in a discrete manner. The equation used to describe these lines is $p = x\cos\theta + y\sin\theta$, rather that the more simple $y = ax + b$, to allow for vertical lines. Each edge point is mapped from a point in the (x, y) plane, to a sinusoidal in the Hough space. The Hough space is the (ρ, θ) plane. ρ is the closest point on the line from origin, and θ is the slope of that line in degrees. An illustration of four edge points in the (x, y) plane and in the Hough space can be seen in figure 17, the upper plot is the (x, y) plane and the Hough space is the lower plot.

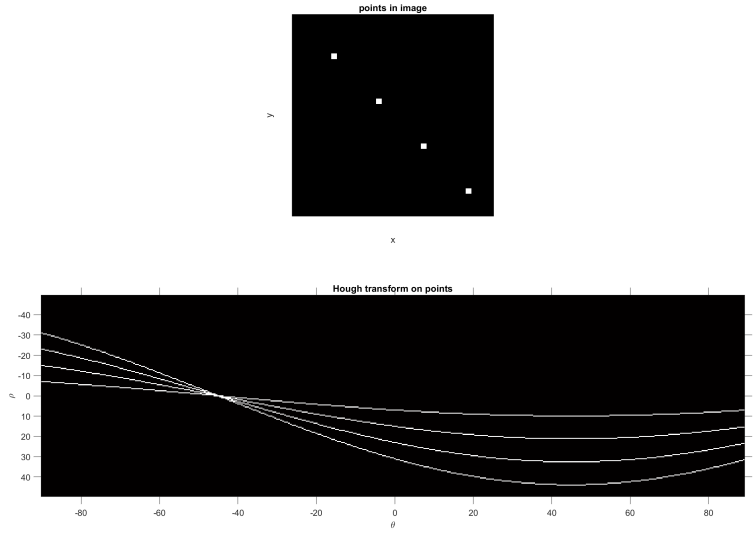


Figure 17: Visualisation of points in the (x, y) plane mapped into the (ρ, θ) plane.

The point in which the sinusoidals cross each other is a detected as a line, because this is the line that all the points have in common. To detect circles rather than lines, the equation for circles are used instead:

$$(i - a)^2 + (j - b)^2 = r^2$$

Where r is the radius of a circle, (a, b) is the circle center coordinates, and (i, j) is the edge point in the image space. With a constant radius the edge point in the image space (i, j) can be mapped to Hough space (a, b) and used to find locations with a high likelihood of being a circle center B1.

When using openCV's `cv::HoughCircles()` circles with different radii are being searched for. Before calling `cv::HoughCircles()` the input image is blurred to reduce noise, and this reduces the amount of false detected circles.

7.3 3D localization of marbles

This subsection the distance from the robot to a marble is determined based on the camera geometry. To determine the distance to a marble in 3D space from the camera center, the know diameter of the marbles is utilized.

From figure 15 on page 21, it is seen that the two red triangles have congruent angles and the sides are in proportion. The relation between the two triangles are described by equation (13).

$$\frac{d_{px}}{f} = \frac{d}{a} \tag{13}$$

Where d_{px} is the diameter of the marble projected into the image plane in pixels, f is the focal length in pixels, d is the diameter of the actual marble in the 3D environment in meters, and a is the distance from the center of the camera center to the marble in meters.

The radius of the circle, in pixels, in the image plane is found using the Hough Transform, the focal length was calculated in equation (11) and the actual diameter of the marble is a known value of 1. Thus the only unknown variable is the distance. Isolating a equation (13) results in equation (14) which determined the distance to a detected from the camera geometry.

$$a = \frac{f}{d_{px} \cdot d} \tag{14}$$

7.3.1 Performance of 3D localization

The experiment determining the reliability of the implemented 3D localization method is described in appendix A5. The estimated distance has on average an error of 6.51%. This error is closely related to the error of the Hough transform because this method uses the output from Hough. The error varies largely and is smallest in the range where the Hough performes best.

8 Discussion

8.1 Strategy for room visitation

The strategy for room visitation explored in section 2.2, has been tested with an succesrate of 90%, with 10 tests being executed, these tests are described in A1.

The test showed that most of the start configurations resulted in a fairly efficient path through the nodes in the environment however, one did not. It is considered a bad choice of path when few nodes are left in an area instead of being visited, so that a lot of backtracking is necessary to reach these nodes at a later point. In the current implementation the closest node which has not yet been visited is chosen, and this wrongful path choice is likely due to the fact that the order in which the nodes were reached led to a situation with very few reachable nodes, and the closest happened to lead away from a few un-visited nodes.

This issue can possibly be reduced by looking back through all nodes in the created route looking for reachable nodes, and then choosing the node closest to an already visited node, whenever backtracking is necessary. However this is a more expensive way of computing the route and as seen from the test results, in 90% of the tested cases a situation such as this did not happen.

8.2 Accuracy of the control method

A test of the control of the gazebo robot is documented in journal A2. Tests were performed with two different approaches to time counting, one based on real time, and one based on loop time, utilizing the program sleeptime of 10[ms] between loops.

In the real time test showed a non-constant undershoot to of the intended result. On the contrary the loop time test showed a non-constant overshoot of the expected result. A likely reason to this inaccuracy is the time calculation. The robots movement likely isn't based on real time or loop time since it is in a simulated environment. It is likely dependent on the frame count, which is neither consistent or accounted for in these tests. Therefore if the simulated environment is slower than real time, the undershooting is explained. Likewise, if it is faster than the simulated loop time, it explains the undershoot associated with this choice of time step.

A solution for this problem could be accounting for the deltatime value, which describes the time between frames in a simulation environment. A way of obtaining this value wasn't discovered during this project.

8.3 Localization of the robot

A test of how accurately the implemented particle filter estimates the robot position is described in the journal A3. Four main observations was made in the performed tests. First, the program has a bug that leads to termination after between the 20-40 minute mark. Second, program starts out very accurate. Third, the program becomes in-accurate after more time-steps are processed. Fourth, its in-accuracy reaches a max, where after its offset seems to have a downwards going tendency.

The second, third and fourth observations can partially be explained. Since all particles created are uniformly distributed across the map, the estimated position should ideally be very close to the center, which the robot starts in, this explains the fact that the estimated position resembles that of the robot closely when the program starts. The particle filter only uses one LIDAR value to compare environments, so when the program begins it is likely to find a lot of areas that match the environment without being the correct one, and therefore unsurprisingly will diverge from the actual position in the beginning. This divergence is counteracted with the large amount of particles, and should be reduced over time, as the robot moves and the wrong particles are removed, but it will likely take many time steps for this to happen. The tested program had 4000 available time steps, but the program terminate after a maximum of 600 time steps, and therefore we can't definitively conclude that the program converges over time, even though the fourth observation shows a downwards going tendency after reaching its maximum. Another possible solution to this divergence instead of convergence could be using a more complex input from the robot to compare with the particles, this could be a range of the LIDAR scanner or simply three LIDAR rays pointing in different direction and create the particles according to this. A comparison between more complex inputs would give a better ground to determine the similarity between robot and particles on. Due to the same problem in control, as described in subsection 8.1 with a missing delta time value, the accuracy of translation of the particles is questionable, this will result in the particles moving differently than the robot when moves are performed.

The noise generated to add to the new particles is based on a constant value, for the position noise it is always -40 and +40 in x and y, and for the rotation noise it is always between -90 and 90 degrees. For a more ideal filter the amount of noise should scale with the weight of the particle it is copying. This could lead to less total error caused by transformation of particles since the new particle could compensate for misinterpreted data.

8.4 Test of Hough

An experiment determining the reliability of the implemented Hough Transform algorithm is described in appendix A4. The test showed that at a distance in the range between 2- and 7.5 meters the method were able to correctly detect a marble in the Gazebo environment. The same test showed that the error with regards to localizing the marble on the image is on average 0.62% for the x coordinate, 1.93% for the y coordinate and 3.28% for the radius. The inaccuracy of the marble detection could be caused by noise in the picture. Likewise the fact that the circle is in fact not round because it consists of pixels could contribute to this inaccuracy. It was observed that the error in localization of the marbles increases substantially beyond 7.5 meters, and the implemented Hough Transform is no longer an appropriate method.

8.5 Distance to marbles

The method for determining the distance to a marble using the camera geometry is most accurate if the subject is located in the middle of the image plane. If a marble is off-center, it will due to the pinhole camera model be skewed and represented as an ellipse. However, this is a miniature amount, and is not significant to be accounted for in the scenario in this project. If more accuracy is desired it could be worth accounting for.

The distance calculation is also dependent on the accuracy of the Hough Transform. Should this not be precise, the distance calculated would also be inaccurate.

9 Conclusion

Control of the robot has been implemented with two functions that allow the user to specify what direction to turn the robot, and its forward and backwards momentum. In the current implementation these methods either undershoots or overshoots the desired movement with a non-constant unknown amount.

A strategy for visiting all rooms in the environment has been implemented. Rooms are detected using trapeziodal decomposition and an order in which the rooms are to be visited are determine by an algorithm which chooses the next room to visit based on distance. It attempts to isolate sections of the map, to try and avoid backtracking. This strategy had a 90% success rate.

A method which were intended for localization of the robot is implemented using a particle filter. In the current iteration of the code this method is not implemented in a way which enables it to determine the position of the robot as the robot interacts with the environment. In test it crashed due to unknown reasons and did not clearly converge towards the robots location.

Marble detection using Hough transform is implemented. The marbles detection is successful with one marble at a distance between 2 and 7.5 meters from the camera center.

3D localization of found marbles are implemented utilizing the geometry of the camera. This method can determine the distance to a marble with an average error of about 6%. The precision of this method is limited by the precision of the marble detection using Hough transform, because the inaccuracy is reflected in the accuracy of the 3D localization.

A fuzzy controller has been implemented. This controller ensures that the robot avoids collisions with obstacles when interacting with the environment.

Q-learning has been implemented on an abstraction of the map used in the project. On the road map the agent interacts with the environment by take actions with the objective of maximizing the collection of reward in terms of marbles which are distributed in the environment. The implemented Q-learning uses an $\alpha = 0.85$ and $\gamma = 0.25$ for the optimal navigation resulting in the largest average amount of collected marbles.

10 Bibliography

B1 Kenneth Dawson-Howe, *A Practical Introduction to Computer Vision with OpenCV*, Wiley, 2014

B2 Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki and Sebastian Thrum, *Principles of Robot Motion - Theory, algorithms and implementation*, The MIT Press, 2005.

B3 Brian Douglass, *Autonomous Navigation, Part 2: Understanding the Particle Filter*, MATLAB, 2020, link: https://www.youtube.com/watch?v=NrzmH_yerBU

B4 Weisstein, Eric W, MathWorld-A Wolfram Web Resource- Rotation Matrix, https://mathworld.wolfram.com/RotationMatrix.html

B5 Gazebo website : http://gazebosim.org/

B6 Bezier curve - Math : https://en.wikipedia.org/wiki/B%C3%A9zier_curve

B7 Anders Lyhne, Christensen, Reinforcement Learning 4

B8 Kevin M. Passino and Stephen Yurkovich, Fuzzy Control, 1998, Addison-Wesley Longman, Inc.

11 Appendix

A1 Project group : *Journal - Strategy*, Journal describing the test of strategy planing, can be found in "Journals" as "Journal - Strategy"

A2 Project Group : *Journal - Control*, Journal describing the test of control of robot, can be found in "Journals" as "Journal - control"

A3 Project Group : *Journal - Localisation*, Journal describing the test of particle filter and localisation with it, can be found in "Journals" as "Journal - Localisation"

A4 Project Group : *Journal - Hough*, Journal describing the test of Marble detection with Hugh transform. Can be found in "Journals" as "Journal - Hough".

A5 Project Group : *Journal - Distance*, Journal describing how reliable the distance detection is. Can be found in "Journals" as "Journal - Distance".

A6 Project Group : *Journal - QLearning*, Journal describing test of Qlearning, can be found in "Journals" as "Journal-QLearning",

A7 Project Group : *Journal - Fuzzy*, Journal describing test of fuzzy, can be found in "Journals" as "Journal - Fuzzy"

A8 Map of gazebo environment : Can be found in folder "images" as "upscaledMap" (Is up scaled from original for ease of viewing).

A9 Code for running en gazebo simulation : Can be found in "Code" names "Gazebo".

A10 Code for running localisation, can be found in "Code", named "ParticleFilter".

A11 Code for running robot control, can be found in "Code", named "Control".

A12 Code for running strategy and room detection, can be found in "Code", named "strategy_planer".

A13 Code for running QLearning, can be found in "Code", named "qLearning".

A14 Code for fuzzy and Hugh Transform, can be found in "Code", named "Fuzzy_And_CV".