University of Southern Denmark

Course project

Robotics $5^{TH}$ 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

## 2.1 Problem statement

### 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.

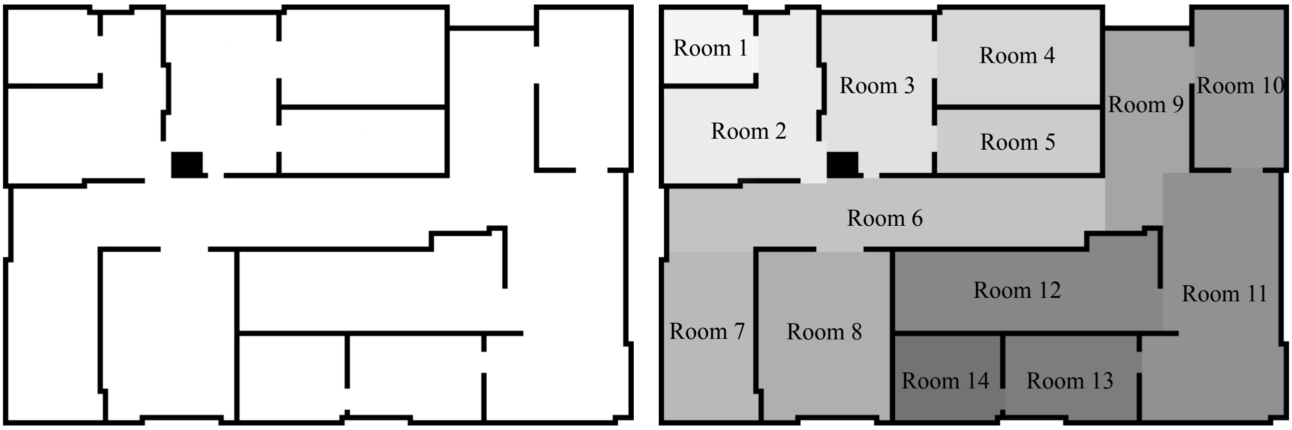
QT Creator, Fuzzy Lite, Gazebo and Matlab will be used throughout the project.

## 2.2 Readers guide

# 3 Design

## 3.1 Environment

The two-wheeled robot should navigate around the environment called “bigworld” shown on figure 1a. 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 1b, 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 2a. 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 rest of the 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 2b.

*(a) Illustration of the unsorted data* *(b) Illustration of the soted data*

*Figure 2: Illustration of the sorted and unsorted data*

### *3.2.1 Line detection*

**Least Square Method**The Total Least Square method uses the normal parametrization of a line in polar coordinates, which is 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. This method involves a determination of the orthogonal distance (the shortest distance) from a point $(x, y)$ to a line $l$ (see figure xx). The normal parametrization of the line $l_i$ is given by the following formula:

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

Missing figure

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:

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

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

The Total Least Square Method solves 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 $\chi^2$ by setting $u_{x,k} = u_{y,k} = \sigma$ and $k = 1, \ldots, n$. 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 $\chi^2$ is done by solving the nonlinear equation system with respect to each of the two line parameters ($r$ and $\alpha$)

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

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 $y$.

**Line Extraction Algorithms**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 relative to it to avoid driving into them. The environment is predefined in this project, and only has walls as obstacles. These walls can be detected using the robots Lidar sensor and a line extraction algorithm, since all obstacles in the Lidar data can be presented as a straight line. Here, there are several line extraction techniques to choose from. We have chosen to implement the incremental line extraction algorithm, because of its simplicity. The incremental algorithm is shown in table xx.

| **Table 3: Incremental Algorithm** | |
|---|---|
| 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 |

*Figure 3: Description of the Incremental algorithm*

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 line conditions. The first condition is a threshold for the angle between the previous and current line model (see figure xx). 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 (see figure yx). 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 (see figure yy). This angle is therefore defined to be the following:

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

Missing 3 figures

## 3.3 Marble detection

## 3.4 Tangent bug

## 3.5 Search strategy

## 3.6 Q-learning

Here is something on Q-learning bla bla bla

***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

***Algorithm: $\epsilon$-greedy policy***

if random number < $\epsilon$
    return random action
else
    return policy action

***Algorithm: getNextAction()***

loop all actions for a given state
    if Q-value is higher than maxValue

# 4 Implementation

## 4.1 Line detection

## 4.2 Marble detection

## 4.3 Tangent bug

## 4.4 Search strategy

## 4.5 Q-learning

# 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 3.1.

#### *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 1. 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 2.

#### *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 1: 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 2: 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 Basic test

This test is based on value iteration

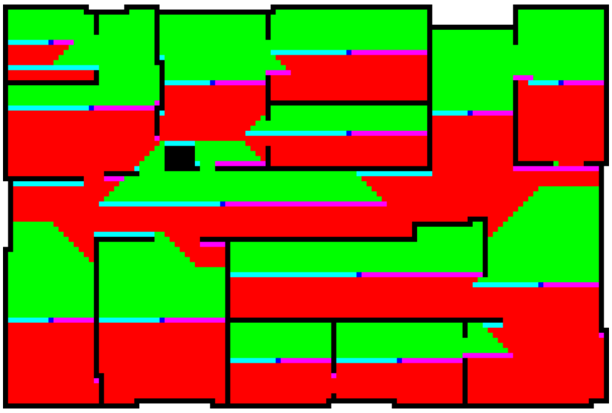


*Figure 4:* $dt = 0.1$