Husbie 2.0

The study of different patterns' areal coverage efficiency

ERIK SVENSSON

DARIA GRANSTRÖM
Abstract

The aim of this thesis is to examine the areal coverage efficiency of different patterns, such as Minimizing Square, Parallel Movement and Random Bounce. By implementing these patterns on the demonstrator consisting of hobby-electronics, the movement patterns were performed in an experiment during four discrete time intervals with covered area as the dependent variable.

By comparing theoretical and practical performance, following conclusions have been made that in the shortest tested period of time, the Parallel Movement pattern is the most efficient one in terms of areal coverage, while on the longest tested time interval there is no statistical evidence, which implies that there is a difference between the patterns. In order to further explore the patterns efficiency at longer time intervals, more tests needs to be conducted with potentially an increased size of experimental setting.

When it comes to the practical implementation of the required tests, the later has been a challenge to implement by using a three-wheel construction base for the demonstrator. Thus, the decision to switch to a four-wheel construction has been made, which enabled conditions required for answering the thesis statements.
Referat

Syftet med denna avhandling är att undersöka hur effektivt olika rörelsemönster kan täcka en area. Mönstrena som undersöktes var Minimising Square, Parallel Movement och Random Bounce.

En prototyp av hobby elektronik konstruerades i en ständig förbättringsprocess genom hela projektet. Med hjälp av den konstruerade prototypen testades sedan de olika rörelsemönstrena under fyra distinkta tidsintervall för att mäta hur mycket area som täckts under respektive körmning.

Från resultaten av experimentet kunde följande slutsatser erhållas: Parallel Movement hade det bästa resultatet på det kortaste tidsintervallet, medan på det längsta tidsintervallet så förelåg ingen statistisk säkerställd skillnad mellan mönstrena. Detta tyder på att frågan måste utforskas vidare med t.ex större area och längre tidsintervall.
Acknowledgements

We would like to devote an extra gesture of gratitude towards:

Nihad Subasic for directing us in the right direction along the project as well as giving us motivation and inspiration.

Didem Gündür for being supportive supervisor and helpful, when most needed. We appreciate your constructive feedback about the thesis and technical details of the demonstrator.

All the FiM 2017 assistants for providing assistance with 3D-printers, technical advise and lots of patience as well as all the FiM students for sharing our moments of joy and despair.

All of the PhD students for critical review during seminars.

To Staffan Qvarnström for technical advise regarding encoders, H-bridges and soldering.

To Sofia Svensson, Kim Fogelström and Viktor Lahti for useful comments and feedback on the report.
Acknowledgements

1 Introduction
1.1 Background .............................................. 1
1.2 Purpose .............................................. 1
1.3 Scope .............................................. 2
1.4 Method .............................................. 2

2 Theory
2.1 Gauging the distance with ultrasonic sound ................ 3
2.2 Areal Coverage Patterns .................................. 4
  2.2.1 Theoretical performance ....................... 5
  2.2.2 Minimizing Square .............................. 6
  2.2.3 Parallel Movement ............................. 7
  2.2.4 Random Bounce .............................. 8

3 Demonstrator
3.1 Hardware .......................................... 9
  3.1.1 Raspberry Pi 3 ................................ 9
  3.1.2 Ultrasonic sensor .............................. 9
  3.1.3 Voltage regulator ............................. 9
  3.1.4 H-bridge ................................... 10
  3.1.5 Robot Base .................................. 10
3.2 Software .......................................... 10
  3.2.1 Operating system .............................. 10
  3.2.2 Python 3 ................................... 11

4 Experiment
4.1 Setup of the experimental setting ...................... 13
4.2 The Evaluation Method ................................ 14
4.3 Findings ........................................... 14

5 Discussion
5.1 Conclusion .......................................... 18
List of Figures

2.1 Sensor with the path of a sound-wave and distances, drawn in Microsoft Visio .................................................. 4
2.2 A simple graph, drawn in Microsoft Visio ................................. 5
2.3 Grid Map, drawn in Microsoft Visio ........................................ 6
2.4 Minimizing Square, drawn in Paint ........................................... 7
2.5 Parallel Movement, drawn in Paint .......................................... 8
2.6 Random Bounce, drawn in Paint ............................................. 8

4.1 Experimental setting, drawn in Microsoft Visio ......................... 13
4.2 Plot of the experimental data and the resulting regression, generated by Matlab .................................................. 15

A.1 Experimental setting with obstacles, drawn in Microsoft Visio ....... 24

C.1 Raspberry Pi ......................................................................... 29
C.2 HC-SR04 ultrasonic sensor ................................................. 29
C.3 Robot base called Magician chassis with three wheels .............. 30
C.4 Robot base called 4WD chassis from Dagu with four wheels ....... 30
<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Statistical results for a respective pattern</td>
<td>14</td>
</tr>
<tr>
<td>4.2</td>
<td>Confidence interval of the regression constants</td>
<td>15</td>
</tr>
<tr>
<td>4.3</td>
<td>Statistical results from the paired t-test for a respective pair of patterns on the 15 seconds time interval</td>
<td>16</td>
</tr>
<tr>
<td>4.4</td>
<td>Statistical results from the paired t-test for a respective pair of patterns on the 30 seconds time interval</td>
<td>16</td>
</tr>
<tr>
<td>4.5</td>
<td>Statistical results from the paired t-test for a respective pair of patterns on the 45 seconds time interval</td>
<td>16</td>
</tr>
</tbody>
</table>
List of Abbreviations

CPU  Central Processing Unit
GPIO  General Purpose Input and Output
PID  Proportional-Integral-Derivative
RAM  Random Access Memory
Chapter 1

Introduction

This chapter is an introduction to the project Husbie 2.0. It includes the background studies, purpose, scope and methods used to acquire the results.

1.1 Background

Shoveling, raking leaves and cutting grass are examples of both physically demanding and time consuming tasks. In a broader sense, they consist of removing material from a specified area and vacating it on another. This is a repeated task that could be automated, which could potentially save people money and repetitive strain injuries. This task can be generally divided in three sub-problems, which are collecting material, covering an area and vacating material.

When it comes to area coverage, it has numerous applications for search and rescue missions, robotic vacuum-cleaners or mapping other planets[24][15][12]. For the scope of this thesis, different patterns have been chosen for comparing them with each other. The Random Bounce pattern has been especially influenced by motion pattern used by a robotic vacuum cleaner, which combines a random wall bounce and spiraling at random time intervals to vacuum the room[15]. The patterns Minimizing Square and Parallel Movement have been inspired by the search patterns used in the search and rescue manual issued by the US government[24].

1.2 Purpose

This thesis will examine the different moving patterns for covering an area and investigate the best suited ones for a respective task by measuring the proportion of the area covered during fixed time intervals. Other aspects, which are examined, are whether the patterns theoretically successfully cover the entire area and to test time efficiency per covered area unit in order to optimize the operating time and expenses for a potential robot application.
Thus, the research questions are to be investigated in this thesis are stated as following:

- What kind of sweeping patterns exists in literature for area coverage?
- What kind of pattern is the most efficient in terms of areal coverage of a predefined area?
- How does the implementation of the chosen hardware affect the practical performance of the respective pattern?

1.3 Scope

The scope of this thesis are:

- Room restrictions are square or rectangular. Experimental setting is situated within a specified indoor area. There is no height deviation within the area.
- By constructing a robot, such parameters as adaptability to the market as well as design thinking haven’t been taken into the consideration. No mass production of the demonstrator is intended.
- By using hardware as Raspberry Pi, Encoders and Ultrasonic Sensors in the demonstrator construction, the research questions are intended to be examined and answered.
- Materials used in robot construction are not suitable for industry usage and in order to adapt the demonstrator to the desired industry usage additional researches are needed to be done in the fields of Material Science as well as Solid Mechanics.

1.4 Method

The method, which is used in order to investigate time efficiency of a respective pattern in terms of area coverage, is to create an experimental setting with areal squares represented by fixed dots. By setting time as a fixed variable, notations of how many dots have been covered during the chosen period of time are made. The number of dots passed over by the demonstrator is notated by observers. In the Experiment section there are more details about the specifications for the experimental setting as well as the description of the experimental process. In the same section information about what method has been used in order to analyze the results received from the experiment is to be found. The choice of Husbie’s hardware and software are explained in the Demonstrator section.
Chapter 2

Theory

In the Theory section relevant background research and literature studies for the experiment are described. Topics such as the choice of the patterns, their algorithms and the theoretical proof of their areal efficiency are covered as well as the theoretical background explaining the principle behind measuring the distance with ultrasonic sensors.

2.1 Gauging the distance with ultrasonic sound

There are different techniques available today in order to measure distance. In this thesis, ultrasonic technology was used, because it is an economical solution with a sufficient accuracy for this type of research [20].

Ultrasonic distance sensors are designed to measure distance between the source and the target using ultrasonic waves. The variables, which are needed to be taken into consideration, when measuring distance with using ultrasonic sound waves, are the speed of the sound and the time it takes for a sound wave to return when colliding with a obstacle. The medium, robot operates in, have a direct impact on the speed of sound [9]. By making the assumption that air is an ideal gas, the speed of sound can be calculated by the following equation

\[ V_{\text{sound}} = \sqrt{R \cdot T \cdot \kappa}, \]  

(2.1)

where T is the temperature in Kelvin, k is the heat ratio and R is the individual gas constant. The individual gas constant and temperature are not constant and depends on the given conditions in the medium, where robot operates.

The distance to an object is calculated by gauging the time difference between transmission and reception using the speed of sound as stated in the following equation [17]

\[ v = \frac{d}{t}. \]  

(2.2)
By combining equation 2.1 and 2.2 the distance between the robot and a respective object can be calculated as followed

\[ d = \frac{t}{2} \cdot V_{\text{sound}}. \]  

(2.3)

Thus, by using equation 2.3, the distance C is calculated (see figure 2.1).

Figure 2.1: Sensor with the path of a sound-wave and distances, drawn in Microsoft Visio

In order to calculate the distance B seen in figure 2.1, the Pythagoras Theorem has been used. Pythagoras Theorem uses the relationship between the three sides of a right triangle in order to show the correlation between the length of these sides. The distance A is the half of the distance between the receiver and transmitter. The sides A,B,C form an orthogonal triangle with the relationship accordingly to equation 2.4

\[ B = \sqrt{C^2 - A^2}. \]  

(2.4)

2.2 Areal Coverage Patterns

Areal Coverage Patterns are used in various applications. One of them is search and rescue, where boats and helicopters scout large areas in predefined patterns to optimize covered ground. The most common search patterns are Minimizing Square and Parallel Movement [24]. The other application, where areal coverage patterns are used, is automated vacuum cleaners, where Random Bounce is the most usual
2.2. AREAL COVERAGE PATTERNS

pattern [15]. Thus, the aim of this thesis is to test these three patterns in terms of their areal coverage efficiency.

2.2.1 Theoretical performance

In terms of theoretical performance, the assumption has been made, that the chosen patterns cover the whole predefined area effectively or by using the best path. In order to test this assumption, the theoretical performance has been estimated by graph theory according to minimum cost flow problem[25]. One of the implementations of minimum cost flow problem is "Chinese postman problem" [2]. The adaption of the Chinese postman problem has been made in the way that the postman doesn’t have to return home (in the Husbie’s case the robot doesn’t need to return to its start position).

When testing our assumption, the predefined area can be seen as a graph. A graph in graph-theory consists of two sets, Vertices(V) and Edges (E) [3]. Vertices describes the nodes of the graph while the edges describes the arcs that connects the egdes (as seen in the figure 2.2).

![Figure 2.2: A simple graph, drawn in Microsoft Visio](image)

Vertices are the set of nodes \( V = \{A, B, C, D, E\} \).
Their edges are \( E = \{(a, b), (b, c), (c, d), (d, e), (e, a)\} \).

By dividing the predefined area into a grid of small squares, where each square is represented by a node, the grid map is created (depicted in figure 2.3). Regarding the efficiency of each pattern, the pattern with minimal sum of the edge-costs would be deemed the most efficient in terms of traveled distance according to the minimum cost flow problem.

Instead of setting and solving the equation for the minimum cost flow problem, one can apply following reasoning: the network consists of a set of nodes \( N = \{1, ..., n\} \) with undirected edges starting from node \( j \). In order to reach all the nodes, without
visiting them twice, traveling must be done along \( n - 1 \) edges. With the assumption that all edges are of the same length, the set of optimal patterns must meet the condition of having \( n - 1 \) edges along its path.

![Figure 2.3: Grid Map, drawn in Microsoft Visio](image)

This condition has been met by the Minimising Square Pattern and the Parallel Movement Pattern. When it comes to the Random Bounce, as its motion pattern is a stochastic variable, then there is a sample space of stochastic variables or motion patterns for Random Bounce, which meets this condition mathematically-wise.

### 2.2.2 Minimizing Square

Minimizing square is a motion pattern where the demonstrator moves in a spiral-shaped patterns as seen in figure 2.4. The motion starts in a straight line along the wall keeping a distance of \( s \) millimeters from the wall. Based on measurements from the right and left sensor, the decision is being made in terms of it is going to be a clock- or counterclockwise motion. Once the front sensor measures a distance less or equal the distance \( d \) the demonstrator stops and revolves 90° to

1. the left if the motion is counterclockwise and both the right and front sensor is triggered by the distance to the wall

2. the right if the motion is clockwise and both left and front sensor is triggered by the distance to the wall.

After three consecutive turns the distance \( d \) is enumerated to decrease the square, this continues until a breaking condition is met.
2.2. AREAL COVERAGE PATTERNS

2.2.3 Parallel Movement

Parallel Movement pattern is a pattern, where a robot goes in parallel lines between the walls as shown in a figure 2.5. The algorithm, which has been used in order to empower this movement, is described below:

- Phase 1: The demonstrator moves along the wall and than when the distance received from the front sensor has reached the minimum distance, which is predefined, the demonstrator’s movement stops. The next phase starts.

- Phase 2: The demonstrator rotates $90^\circ$ in the direction from the corner, moves the distance equal to the width of the demonstrator’s body and then rotates again $90^\circ$ in order to start the new parallel line. Then it goes to the phase one again.

In the beginning of the execution, the robot uses information from the right and the left sensor in order to analyze what corner it starts from. With the regards on the later information, the decision about what direction it should rotate after the first parallel line movement, can be made. Later after every rotation, the direction of turning changes into the opposite one. Program executes until the demonstrator detects a corner, when the last straight movement executes.
CHAPTER 2. THEORY

2.2.4 Random Bounce

The demonstrator moves in a direction until it "bounces" against a wall, where the new trajectory is determined by a stochastic variable, which chooses a new rotational angle and a turning direction. The bouncing phase is determined by the front sensor by using the least allowed predefined distance the robot can take in order to stop before hitting the wall. The range of the rotational angle is from 0 to 360 ° and the turning direction is either the left or right one. The embedded functions in Python from module random are used in order to get the random number for angle and the decision, whether to turn right or left. In the figure below 2.6 described one of possible motion trajectory the robot can take, but as mentioned above the trajectory for the Random Bounce is a stochastic variable, which leads to a different chosen trajectory for each execution.
Chapter 3

Demonstrator

3.1 Hardware

3.1.1 Raspberry Pi 3

Raspberry Pi seen in figure C.1 (found in the Appendix B) is a unadorned computer system in the size of a credit card that has been developed by Raspberry Pi foundation, primary to promote coding in an affordable way for students. Raspberry Pi 3 uses a Broadcom BCM2837 SoC with a clock frequency of 1.2GHz, 64-bit quad-core ARM Cortex-A53 processor. It has 1GB of ram and comes with 40 GPIO ports. GPIO is an abbreviation for general input and output, these ports are programmable for control of hardware such as diodes, transistors or relays. They can also be programmed to receive signals from other electronic equipment such as buttons, sensors or other computers[22].

3.1.2 Ultrasonic sensor

The demonstrator utilizes three ultrasonic sensors of the model HC-SR04, which can be seen in figure C.2 (found in the Appendix B). The sensor measures distances between 2-400 cm with an accuracy up to three millimeters. The circuit consists of three modules: Transmitter, receiver and control-circuit. The control circuit operates with a feed of 5V. To trigger the sensor a voltage pulse with the minimum length of 10\( \mu \)s is required. It causes the control-circuit to send eight ultrasonic burst from the transmitter. Once the ultrasonic sound bounces of an object and is perceived by the receiver, the echo output is set high at 5V from the control circuit. The time difference between the trig and the echo signal is then used in equation 2.3 and 2.4 to calculate the distance to the object.

3.1.3 Voltage regulator

A switched based voltage regulator regulates the desired output voltage by keeping the circuit closed in specific time intervals \( T_{on} \) and keeping it open otherwise. \( T_{on} \)
is calculated according to

\[ T_{on} = \frac{\text{output voltage}}{\text{input voltage}}, \]  

(3.1)

which means that the fraction of time the circuit should be closed is equal to the fraction between input and output voltage. The voltage switch regulator used in the demonstrator is based on the LM2596 circuit.

3.1.4 H-bridge

An H-bridge is an electronic component that allows currents to move in different directions. Its main application is to enable control of rotation and speed in different directions of an DC-motor. The H-bridge, used in the demonstrator, is a L298N based board, which houses dual H-bridges for control of two DC-motors or one step-motor.

3.1.5 Robot Base

First version

The initial version of the demonstrator was based on a construction kit called Magician Chassi by DAGU Hi-Tech Electronic Robotics. It can be seen in figure C.3 (found in the Appendix B). It consists of two acryl-plastic plates mounted with threaded metal rods to create the different floors, where components can be attached. The robot base has three wheels, two of which are mounted on gearboxes and the third one is a balancing wheel mounted in the front, which enables different steering properties.

Second version

The final version of the demonstrator is based on a robot kit called 4WD chassis from the same manufacturer as version one, seen in figure C.4 (found in the Appendix B). Like the first version it consists of two acryl-plastic plates, which create two levels. The difference is that it consists of four wheels instead of three, each mounted with a gear-boxed engine.

3.2 Software

3.2.1 Operating system

The operating system recommended by the Raspberry Pi foundation for the Raspberry Pi 3 is Raspbian, which is a operating system based on Debian and optimized for Raspberry Pi hardware. The system includes a set of basic programs and utilities that enable the Raspberry Pi’s functionality. There are also third party operation systems available, for example Windows 10 IOT core, Ubuntu mate and PINET.
3.2. SOFTWARE

3.2.2 Python 3

Python is a high-level program language. High-level refers to the abstraction from the machine-close commands such as memory handling, which is automatically handled by Python. A high-level level language can express concepts with fewer lines of code than a low-level language such as C[21]. In the Raspbian environment all of the motions, algorithms and sensor-readings have been implemented in Python 3. Python was chosen as the programming language of the project because of the following reasons[6]:

- Easy to read and to execute complex functionalities.
- Supports Raspbian and enables complex features for this software.
- Object-oriented driven programming.

For more information about the specific Python modules and libraries that have been used can be found in the Appendix B.
Chapter 4

Experiment

4.1 Setup of the experimental setting

The experiment is built up to match the theoretical performance in section 2.2.1. The experimental setting is made up of a square measuring 4 m$^2$ with walls as outer bounds as seen in figure 4.1. The square is divided into a grid map with smaller squares with the same dimension as the size of the demonstrator. When the demonstrator passes over a node, which is represented by a dot, the area in it will be accounted as covered.

Instead of using the traveled path as a measure of efficiency, time is used as a fixed variable. Each implementation of the patterns is to operate within the setting during the time limit, whilst two observers keep track of the number of dots passed. The pattern with the most counted dots will be deemed as the most efficient during the fixed period of time.

Figure 4.1: Experimental setting, drawn in Microsoft Visio
4.2 The Evaluation Method

A statistical test has been used in order to analyze the results of the efficiency of a respective pattern in terms of areal coverage. As mentioned above, the notation of how many dots passed by the demonstrator has been made during testing under the fixed period of time. For each pattern testings have been made during four fixed periods of time, which are 15, 30, 45 and 60 seconds. In order to make better precision of the results, the same testing for a respective pattern has been made 10 times. Then results, has been analyzed with assumption that the efficiency of a respective pattern is distributed according to the normal distribution. In this way, the expected value and the deviation value of the dots passed have been estimated.

A linear regression has been used in order to present a linear behavior of a respective pattern during continuous time in terms of areal coverage with the based on their performance during the discrete time intervals.

A paired sample t-test has been used in order to test the null hypothesis, whether there is a difference between the patterns on the 5% significance level [14]. If the test rejects the null hypothesis, then there is a difference between the patterns.

4.3 Findings

Following results have been accomplished and presented in the table 4.1:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Time (sec)</th>
<th>Mean Area coverage</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel pattern</td>
<td>60</td>
<td>74.4%</td>
<td>8.5%</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>51.1%</td>
<td>5.6%</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>48.6%</td>
<td>3.3%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>26.4%</td>
<td>1.5%</td>
</tr>
<tr>
<td>Minimizing square</td>
<td>60</td>
<td>73.9%</td>
<td>7%</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>62.2%</td>
<td>6%</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>47.3%</td>
<td>3.3%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>24.8%</td>
<td>0.9%</td>
</tr>
<tr>
<td>Random bounce</td>
<td>60</td>
<td>70.1%</td>
<td>8.6%</td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>54.5%</td>
<td>7.7%</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>38.1%</td>
<td>5.4%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>21.9%</td>
<td>5%</td>
</tr>
</tbody>
</table>

Table 4.1: Statistical results for a respective pattern
4.3. FINDINGS

The Linear Regression

To estimate the patterns continuous behavior a linear regression was produced to fit the experimental data on the form $y = ax + b$ for the respective pattern[14], the results can be seen in the figure 4.2.

![Parallel pattern](image1)

![Minimizing square](image2)

![Random bounce](image3)

(a) Regression points  (b) Linear Regression Curve

Figure 4.2: Plot of the experimental data and the resulting regression, generated by Matlab

Closer inspection of the graphs in the relation to the data points shown in figure 4.2 reveals that the confidence interval of the the constants $a$ and $b$ are relatively large compared to the actual values. The table 4.2 below show the numerical values of the constants and their respective confidence interval, from the observation that all confidence intervals intersect, the conclusion is that there is no statistical evidence to prove that the difference between the regression models of the motion patterns exists.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>a</th>
<th>Confidence interval</th>
<th>b</th>
<th>Confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel pattern</td>
<td>0.9760</td>
<td>0.8421 - 1.1100</td>
<td>15.5156</td>
<td>8.0120 - 19.0192</td>
</tr>
<tr>
<td>Minimizing Square</td>
<td>1.0802</td>
<td>0.9736 - 1.1869</td>
<td>11.5625</td>
<td>7.1817 - 15.9433</td>
</tr>
<tr>
<td>Random bounce</td>
<td>1.0844</td>
<td>0.9570 - 1.2117</td>
<td>5.625</td>
<td>0.3935 - 10.8565</td>
</tr>
</tbody>
</table>

Table 4.2: Confidence interval of the regression constants
CHAPTER 4. EXPERIMENT

The Paired T-tests

From the paired tests’ results, following conclusion has been made that at the fixed interval of 15 seconds the Parallel Movement has shown the best performance of the all examined patterns. The confidence interval for the difference between respective patterns are described in the table 4.3.

<table>
<thead>
<tr>
<th>Pattern 1</th>
<th>Pattern 2</th>
<th>Confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel pattern</td>
<td>Minimizing Square</td>
<td>0.0038 – 0.0274</td>
</tr>
<tr>
<td>Parallel pattern</td>
<td>Random Bounce</td>
<td>0.0090 – 0.0816</td>
</tr>
</tbody>
</table>

Table 4.3: Statistical results from the paired t-test for a respective pair of patterns on the 15 seconds time interval

On the 30 seconds period of time, the conclusion can be made that both Minimizing Square and Parallel Movement were better than random bounce as shown in the table 4.4:

<table>
<thead>
<tr>
<th>Pattern 1</th>
<th>Pattern 2</th>
<th>Confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimizing Square</td>
<td>Random Bounce</td>
<td>0.0492 – 0.1351</td>
</tr>
<tr>
<td>Parallel pattern</td>
<td>Random Bounce</td>
<td>0.0682 – 0.1412</td>
</tr>
</tbody>
</table>

Table 4.4: Statistical results from the paired t-test for a respective pair of patterns on the 30 seconds time interval

On the 45 seconds period of time, the results showed that Parallel Movement was better than Minimizing Square as described in the table 4.5:

<table>
<thead>
<tr>
<th>Pattern 1</th>
<th>Pattern 2</th>
<th>Confidence interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel pattern</td>
<td>Minimizing Square</td>
<td>0.0485 – 0.1734</td>
</tr>
</tbody>
</table>

Table 4.5: Statistical results from the paired t-test for a respective pair of patterns on the 45 seconds time interval

In all other tests, the null-hypothesis couldn’t be rejected, which means that there is no statistical significance proving that there is a difference in the performance between the tested patterns.
Chapter 5

Discussion

During the implementation of the algorithm for gauging the distance from sensors, changes of this algorithm have been made. According to the figure 2.1 mentioned in the section 2.1, the initial algorithm measured only the distance C, which resulted in the erroneous measurements. Thus, the adjustment of the algorithm has been made by using the properties of Pythagoras theorem for minimizing this error.

Regarding the implementation of the chosen patterns’ algorithms, during the coding phase an issue about the right turning angle has been faced. Thus, the decision of implementing encoders on the robot base, in order to have better precision for turning angle, has been made. On the other hand, this implementation has resulted in other challenges, such as calculating the right gearing ratio between robot turning angle and encoders turning angle. The accuracy of this ratio could potentially have a major impact on the precision of the turning angle, as this ratio is used in the execution of the movement. In terms of quality of the chosen encoders, the precision of encoders is insufficient for the PID control. Later in the project, in terms of the poor resolution of encoders, the decision has been made to abandon their implementation on a construction base after several tests.

The other challenges with the Husbie has been the different velocities of the motors, which has led to the deviation of the robot from the desired straight path. One of the possible solutions for this problem is the implementation of a PID controller for steering the motors, which is described in more depth in the Future Work Section (found in the Appendix A). As mentioned in the Demonstrator Section the initial Husbie’s construction had two rear wheels and one wheel in the front, which purpose was to balance the construction. With the regards to the poor quality of the balancing wheel and quite considerable load of the front part of the construction, a deviation in the movement occurred, which could be omitted if a four-wheel construction had been considered. That is why the decision to change the robot base from the one with three wheels to four wheels has been made. That enhanced the performance of the robot regarding the straight movement, as its deviation with
four-wheel’s construction is marginal. There are eventually other possible reasons for the deviation from the straight movement. Other challenge has been the oxidation of the contacts to the H-bridge, which has had a considerable impact on the increasing deviation from the straight movement, which has resulted in potentially more erroneous results during the testing for the 45-seconds time interval.

When it comes to all other issues regarding eventual development of the project or future work, the information about this matter can be found in the Future Work Section, which is in the Appendix A.

5.1 Conclusion

The sweeping patterns that have been chosen to compare in this thesis are Random Bounce, Minimizing Square and Parallel Movement. The former pattern is commonly used in the vacuum cleaner applications, while the later two are common in search and rescue missions. These three patterns have been tested on the specified indoor area during the four discrete time intervals in order to analyze their performance in the relation to each other. The efficiency measurement in this case is the coverage of the specified area percentage-wise. By using statistical tests, following conclusions have been made. On the shortest period of time, the Parallel Movement pattern has shown the best result regarding areal coverage from the all tested pattern, while on the longest period of time there is no statistical significant evidence, which implies that there is a difference between the tested patterns. In order to conclude how patterns relate to each other with an increased time intervals, more tests should be done in as well as potentially larger size of the experimental setting.

When it comes to the practical performance and implementation of patterns and their testing, the demonstrator’s deviation from the straight movement has been the major obstacle for the tests’ conduction, which has led to the complete reconstruction of the robot base from the two rear wheels and one balancing wheel construction to the one with four wheels. That enhanced the performance of the robot and enabled required conditions for conducting of the relevant testings.
Bibliography


[5] Ben Croston. RPi.GPIO 0.6.3. URL: https://pypi.python.org/pypi/RPi.GPIO (visited on 03/31/2017).


Appendix A

Future work

A.1 Implementation of obstacles

In order to explore and cover an unknown territory, a navigation system is required, which gives the information about current positional coordinates of the robot as well as how these coordinates refer to the space, robot operates in. The navigation system also needs to include path planning.

Humans are believed to solve this problem by storing an allocentric representation of the space and by using specific landmarks to orient in the general layout the information about the position can be found[19]. If this process would be automated in a robot it essentially must have the ability to build a representation of a territory and use this representation to navigate it[29].

By implementing obstacles in the room new challenges arise. One of them is to investigate how collision can be avoided. By placing obstacles on the experimental setting, the robot must be automated and perform tasks as following a set path, detecting obstacles, then avoiding them by deviating from its path while passing them and then returning to the previously defined path, while still covering all possible ground around the obstacles.

In order to define this path, firstly a discretized map should be created, which includes all the obstacles placed on the area. The Occupancy Grid Map can be created. If the most optimal path is of the interest, the algorithm based on minimal flow cost optimization can be implemented, t.ex. the Chinese postman problem mentioned in the section 2.2.1.

A.1.1 Robot Mapping

The Robotic Mapping is the method of acquiring spatial models of dynamic and static environments through different robotic applications. [28] In order to create the map, the robot uses information from ultrasonic sensors in order to decide if
the grid is occupied or not. The problem with the Robotic Mapping is that the map and the robot localization are uncertain and by estimation of both variables at the same time makes the measurement an independent variable regarding to the state. [28] In order to make the problem easier to solve, the robot’s pose is decided to be an known variable. In the other words, the robot is going to be placed in the predefined position for all running tests. The acquired map from the Robot Mapping is going to be used afterwards for robot navigation.

In order to create a map, the sensors measurements creates a map by using probabilistic inference. Various probabilistic algorithms address the problem by taking into consideration different sources of noise and their effects on the measurements. That is why, the Bayes filter is applied.

A.1.2 Occupancy-Grid Map

Occupancy-grid map uses probability value on each cell on the 2D-map in order to estimate if it is occupied or not. There are two assumptions, which have been made in order to use this model. The first one is that all the grids on the map and their respective occupancy are independent. The second one is that the robot position is known before-hand [28].
A.2. PROPORTIONAL-INTEGRAL-DERIVATIVE (PID) CONTROLLER

For updating the map, Bayes filter is used, which is a probabilistic method for unknown probability estimation. By using a recursive approach of measurement between latest measurement and former estimation, it estimates the probability if the respective grid is occupied or not. It uses a Hidden Markov Model with the prediction step and measurement step, where the next assumption of Markov property is that future and past states are independent [27].

A.1.3 Motion planning

A simple but effective method of finding an optimal path is the wavefront algorithm which is a breadth-first search in a graph representation of the surrounding area. The breadth first algorithm searches all the child-nodes to find the shortest path to the goal[4].

A.2 Proportional-Integral-Derivative (PID) controller

One of the issues faced during the experiment with the different speed of motors could have been omitted if quality of the motors used on the robot base has been better. One of the possible solutions for the given problem is to implement the PID regulator for motor steering.

Proportional-Integral-Derivative (PID) controller is the control algorithm, which enables steering for the closed loop industrial systems. It consists of three basic coefficients, which are proportional, integral and derivative, in order to get the most optimal solution.

The basic idea behind a PID controller is that it calculates an error value between the current value of output and the desired value and then adjust the system in order to reach the aimed output. In the Husbie case, same velocity of the motors is desired to reach and a PID controller enables that by examining the values from two motors and adjusting them.

A controller attempts to minimise the error of a control variable $u(t)$ [8]

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \quad (A.1)$$

where $K_p, K_i$ and $K_d$ all non-negative coefficients for the proportional, integral and derivative terms. In this model, P depicts the present error, I gives the information about the past error and D accounts for possible trends for the error that might occur.
For the Husbie’s PID application, only two terms could have been used, which are PI, since derivative terms is sensitive to the measurement noise, which might lead to a considerable error in the Husbie’s case, where no low pass filter has been deployed.

A.3 Code Optimization

There is a potential for optimizing the algorithms for the chosen patterns, especially for the Random Bounce. The implemented algorithm for Random Bounce enables the straight movement as well as it uses the rotational angular as the stochastic variable. Potentially, more complex algorithm with more enabled motion patterns including spiral and zigzag movement can be included, which together with the straight line motion are the stochastic variables. Potentially, that could increase the areal coverage efficiency for the Random Bounce pattern.
Appendix B

Python libraries

B.1 GPIO library

The General Purpose Input Output library is an efficient library for control of motors, servos and sensors in the Python programming language. It has been chosen because of C-code that lies under the Python functions, which connects directly to the GPIO pins and allows Python to use functions as "wait", drastically decreases CPU usage.[5][11]

B.2 Threading library

The Threading library was chosen to enable multiple sensor-readings at a time. A thread, or more accurately a thread of execution, is a way of pseudo-splitting up operations in a computer and seemingly making it look like parallel operations. What actually happens is that the scheduler, which is part of the operating system, splits the current tasks in smaller computations. It then spends a small time on each computation for each task, seemingly making it look like the codes are executed simultaneously[16]. In Python threading is implemented by the Threading module.[7].
Appendix C

Hardware

The figure for Raspberry Pi 3:

![Raspberry Pi](image1.png)

Figure C.1: Raspberry Pi

The figure for the Ultrasonic Sensors:

![Ultrasonic Sensor](image2.png)

Figure C.2: HC-SR04 ultrasonic sensor
The figure for the initial robot base construction:

![Robot base called Magician chassis with three wheels](image)

Figure C.3: Robot base called Magician chassis with three wheels

The figure for the final robot base construction:

![Robot base called 4WD chassis from Dagu with four wheels](image)

Figure C.4: Robot base called 4WD chassis from Dagu with four wheels
Appendix D

Code

Code for the the Parallel Movement and the Random Bounce patterns:

Huvudprogram.py

'\n
KTH Royal Institute of Technology
Bachelor Thesis at ITM School: Husbie 2.0
Erik Svensson and Daria Granstrom
TRITA MMK 2017:11 MDAB 629

The Code Function : Steers the robot to perform the Parallel Movement and the Random Bounce Pattern.

'\n
'the file biblotek.py is found to be in the Apendix D''

from biblotek import *
import RPi.GPIO as GPIO
import time
import sys
from math import sqrt
import random

def parallell(motors, sensor):
    """Function, which empowers the parallel movement pattern."""

    angleright = 0.75
    angleleft = 0.73

    start_time = time.time()
    current_time = time.time()
    end_time = 60

    sensor_right = sensor.right()
APPENDIX D. CODE

```python
sensor_front = sensor.front()
sensor_left = sensor.left()

""the condition, which decides if the robot is in the right
or in the left corner"

if sensor_right > sensor_left:
    direction='right'
else:
    direction='left'

while current_time-start_time < end_time: ""time constraint"

    sensor_right = sensor.right()
sensor_front = sensor.front()
sensor_left = sensor.left()
motors.forward()

    while sensor_front > 5: ""the least distance the robot can
come closest to the wall"
        sensor_front = sensor.front()
        time.sleep(0.1)

    motors.stop()

if direction=='right':
    motors.backward()
    time.sleep(0.0005)
motors.stop()
motors.rotateright(angleright)
startd=sensor.front()
print(direction)
motors.forward()
    time.sleep(0.05)
motors.stop()
motors.rotateright(angleright)
direction='left'
print(direction)
else:
    motors.backward()
```

```
time.sleep(0.0005)
motors.stop()
motors.rotateleft(angleleft)
startd=sensor.front()
print(direction)
motors.forward()
time.sleep(0.05)
motors.stop()
motors.rotateleft(angleleft)
direction='right'
print(direction)
current_time=time.time()

def Randombounce(motors, sensor):
    '''function, which enables Random Bounce pattern'''
    start_time=time.time()
current_time=time.time()
end_time=15

    while current_time - start_time < end_time:  '''time constraint''
        sensor_right = sensor.right()
sensor_front = sensor.front()
sensor_left = sensor.left()
motors.forward()

        while sensor.front() > 5:
            time.sleep(0.01)

        motors.stop()

        if sensor.left < 6:
            motors.backward()
time.sleep(0.15)
motors.stop()
        n=random.uniform(0,2)  '''function, which decides
        a rotational angle randomly'''
motors.rotateright(n)
    elif sensor.right < 6:
        motors.backward()
```
time.sleep(0.15)
motors.stop()
n=random.uniform(0,1)
motors.rotateleft(n)
else:
'*for the front sensor, it can rotate in both directions'*
motors.backward()
n=random.uniform(0,2)
time.sleep(0.15)
motors.stop()
'*function, which decides direction randomly'*
x=random.randint(1,2)
if x==1:
    n=random.uniform(0,2)
motors.rotateright(n)
else:
    n=random.uniform(0,2)
motors.rotateleft(n)

current_time=time.time()

def main():
    try:
        GPIO.setmode(GPIO.BOARD)
sensor = Sensor()
motors = Move()

        parallell(motors,sensor)
Randombounce(motors,sensor)

    except KeyboardInterrupt:
        '*The end of the program's execution'*
motors.stop()
GPIO.cleanup()

    if __name__=='__main__':
        main()
Code for the Minimizing Square pattern:

```
from biblotek import *
import RPi.GPIO as GPIO
from math import sqrt

def measure(sensor):
    a = sensor.right()
    b = sensor.front()
    c = sensor.left()

def alfa():
    GPIO.setmode(GPIO.BOARD)
    motors = Move()
    time.sleep(0.1)
    motors.stop()
    sensor = Sensor()
    turns = 0
    d = 15
    try:
        while True:
            R = sensor.right()
            F = sensor.front()
            motors.forward()
            if F < d:
                turns += 1
                motors.stop()
                time.sleep(0.1)
                motors.rotateleft(0.7)
            if turns == 3:
                turns = 0
                d = d + 15
            motors.stop()
            time.sleep(1)
    except KeyboardInterrupt:
```
def MinimizingSquare():
    '''Initiates the motors and the sensors'''
    GPIO.setmode(GPIO.BOARD)
    sensor = Sensor()
    motors = Move()

    '''Sensors' values'''
    R = sensor.right()
    F = sensor.front()
    L = sensor.left()
    R_list = []
    F_list = []
    L_list = []

    R_list.append(R)
    F_list.append(F)
    L_list.append(L)

    '''Robot's dimensions'''
    bredd = 80
    langd = 90

    '''Room's dimensions'''
    A = R + bredd + L
    B = langd + sensor.front()

    '''Boundary for the biggest deviation'''
    bound = 3

    '''The increase in the distance from the wall'''
    C = 30

    '''Rotation'''
    if R > L:
        direction = "right"
    else:
        direction = "left"
''Direction of the rotation''
if direction == "left":
    dist = R
else:
    dist = L

''Time for the rotation''
rot_left = 0.73
rot_right = 0.78
rot = 0.1

''Time for the waiting''
drive_time = 0.1
loop_time = 0.2
stop_time = 0.1

''Counter''
i = 1
turns = 0

''Time constraints''
start_time=time.time()
current_time=time.time()
end_time = 60

try:
    while current_time - start_time < end_time:
        R = sensor.right()
        F = sensor.front()
        L = sensor.left()
        R_list.append(R)
        F_list.append(F)
        L_list.append(L)
        motors.forward()
        if F < dist:
            turns += 1
            motors.stop()
            time.sleep(stop_time)
            if direction == "left":
                motors.backward()
                time.sleep(0.005)
                motors.rotateleft(rot_left)
elif direction == "right":
    motors.backward()
    time.sleep(0.005)
    motors.rotateright(rot_right)
else:
    raise ValueError

'''Controll weather the robot is in the right correlation with the wall'''

if turns == 3:
    dist += C
    print("dist: %d " %(dist))
    turns = 0

    i += 1
    time.sleep(loop_time)
    current_time = time.time()

except KeyboardInterrupt :
    motors.stop ()
finally :
    printlog(R_list,F_list,L_list)
    GPIO.cleanup()

if __name__ == "__main__":
    MinimizingSquare()
The code for the motors' and the sensors' initiations:

```python
import RPi.GPIO as GPIO
import time
import sys
from math import sqrt

class Move(object):
    def __init__(self):
        self.IN1 = 11
        self.IN2 = 15
        self.IN3 = 16
        self.IN4 = 18
        self.ENA = 40
        self.ENB = 38
        self.channelList = [self.IN1, self.IN2, self.ENA, self.IN3, self.IN4, self.ENB]
        GPIO.setup(self.channelList, GPIO.OUT)
        self.freq = 300  #Hz
        self.PWM_left = GPIO.PWM(self.ENB, self.freq)
        self.PWM_right = GPIO.PWM(self.ENA, self.freq)
        self.duty_left = 90
        self.duty_right = 90

def Start(self):
    self.PWM_left.start(self.duty_left)
    self.PWM_right.start(self.duty_right)

def stop(self):
    self.PWM_left.stop()
    self.PWM_right.stop()
```

challenge 

KTH Royal Institute of Technology
Bachelor Thesis at ITM School: Husbie 2.0
Erik Svensson and Daria Granstrom
TRITA MMK 2017:11 MDAB 629
The Code Function : Enables all inputs and outputs, which are required for the patterns’ implementation.
def forward(self):
    GPIO.output(self.IN1, GPIO.HIGH)
    GPIO.output(self.IN2, GPIO.LOW)
    GPIO.output(self.IN3, GPIO.HIGH)
    GPIO.output(self.IN4, GPIO.LOW)
    self.Start()

def backward(self):
    GPIO.output(self.IN1, GPIO.LOW)
    GPIO.output(self.IN2, GPIO.HIGH)
    GPIO.output(self.IN3, GPIO.LOW)
    GPIO.output(self.IN4, GPIO.HIGH)
    self.Start()

def rotateleft(self, Time):
    GPIO.output(self.IN1, GPIO.LOW)
    GPIO.output(self.IN2, GPIO.HIGH)
    GPIO.output(self.IN3, GPIO.HIGH)
    GPIO.output(self.IN4, GPIO.LOW)
    self.Start()
    start_time = time.time()
    current_time = time.time()
    while abs(start_time - current_time) < Time:
        current_time = time.time()
        self.stop()

def rotateright(self, Time):
    GPIO.output(self.IN1, GPIO.HIGH)
    GPIO.output(self.IN2, GPIO.LOW)
    GPIO.output(self.IN3, GPIO.LOW)
    GPIO.output(self.IN4, GPIO.HIGH)
    self.Start()
    start_time = time.time()
    current_time = time.time()
    while abs(start_time - current_time) < Time:
        current_time = time.time()
        self.stop()
class Sensor(object):
    """Based on the code from https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi""
    def __init__(self):
        GPIO.setwarnings(False)
        self.TRIG = [8,36,32] """"The left, the middle and the right one"
        self.ECHO = [10,37,35]
        GPIO.setup(self.TRIG,GPIO.OUT) """"Actionvation of sensors"
        GPIO.setup(self.ECHO,GPIO.IN)

def left(self):
    GPIO.output(self.TRIG[0],True)
    time.sleep(0.00001)
    GPIO.output(self.TRIG[0],False)
    while GPIO.input(self.ECHO[0]) == 0:
        pulse_start = time.time()
    while GPIO.input(self.ECHO[0]) == 1:
        pulse_end = time.time()
    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17150
    width = 2.7 """"width between receiver and broadcaster"
    distance =sqrt(distance*distance - 1.8225 )
    distance = round(distance,2)
    time.sleep(0.1)
    return distance

def front(self):
    GPIO.output(self.TRIG[1],True)
    time.sleep(0.00001)
    GPIO.output(self.TRIG[1],False)
    while GPIO.input(self.ECHO[1]) == 0:
        pulse_start = time.time()
while GPIO.input(self.ECHO[1]) == 1:
    pulse_end = time.time()

pulse_duration = pulse_end - pulse_start

distance = pulse_duration * 17150
width = 2.7
distance = sqrt(distance*distance - 1.8225)

distance = round(distance,2)
time.sleep(0.1)
return distance

def right(self):

    GPIO.output(self.TRIG[2], True)
    time.sleep(0.00001)
    GPIO.output(self.TRIG[2], False)

    while GPIO.input(self.ECHO[2]) == 0:
        pulse_start = time.time()

    while GPIO.input(self.ECHO[2]) == 1:
        pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start

    distance = pulse_duration * 17150
    width = 2.7
    distance = sqrt(distance*distance - 1.8225)

    distance = round(distance,2)
time.sleep(0.1)
return distance