DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2018

A Comparison of Path Planning Algorithms for Robotic Vacuum Cleaners

TOBIAS EDWARDS

JACOB SÖRME

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

A Comparison of Path Planning Algorithms for Robotic Vacuum Cleaners

TOBIAS EDWARDS
JACOB SÖRME

Bachelor in Computer Science
Date: June 6, 2018
Supervisor: Jana Tumova
Examiner: Örjan Ekeberg
Swedish title: En Jämförelse av Ruttplaneringsalgoritmer för Robotdammsugare
School of Electrical Engineering and Computer Science

Abstract

Household robotics is on the rise with robotic vacuum cleaners taking the lead. One of the challenges in creating great robotic vacuum cleaners is path planning for coverage of the room. For instance, the robot might not know how the environment looks, or where in the room the robot is.

Three path planning algorithms are tested: the random path algorithm, the snaking algorithm and the spiral algorithm. In our simulations the random algorithm was initially faster than the snaking algorithm. However, the random algorithm's performance worsened and it performed similar to the snaking algorithm in the end. The spiral algorithm performed worse than the other tested algorithms. The random algorithm showed a more stable performance than both the snaking and spiral algorithms. However, we believe that the spiral algorithm could be complemented with other algorithms. Errors in the results could possibly come from the method used in the software to detect wall collisions. Future research could improve the software and also cover collision with objects in rooms and rooms of other shapes.

We concluded that the random algorithm performed best in our simulations and questioned the practical use of more advanced algorithms.

Sammanfattning

Antalet robotar i hemmet ökar i världen och robotdammsugaren är ett populärt val. Bra ruttplaneringsalgoritmer är en av utmaningarna för att tillverka bra robotdammsugare. Exempelvis behöver inte roboten känna till sin position i ett rum eller ens känna till rummets utseende, hur ska utvecklare gå till väga?

Tre ruttplaneringsalgoritmer har testats: en slumpartad algoritm, en sicksack-algoritm och en spiral-algoritm. I våra simuleringar fann vi att den slumpartade algoritmen var initialt snabbare än sicksack-algoritmen. Dock försämrades slumpalgoritmen över tid och till slut gav result likt de vi såg för sicksack-algoritmen. Spiral-algoritmen gav sämre resultat än de två övriga testade algoritmerna. Slumpalgoritmen hade mindre avvikelse än både spiral-algoritmen och sicksack-algoritmen. Vi tror att spiral-algoritmen skulle kunna kombineras med andra algoritmer för att ge bättre resultat. Fel i våra resultat kan bero på hur vi upptäcker väggkollisioner i simuleringarna. Framtida forskning kan förbättra och utveckla mjukvaran samt testa rum med varierande utseende.

Sammanfattningsvis var slumpalgoritmen bäst i våra simuleringar och vi ifrågasätter den praktiska användbarheten hos andra mer avancerade algoritmer.

Contents

1 Introduction **1**
1.1 Aim & Goal 3

2 Path Planning **4**
2.1 Random Walk Based Algorithm 4
2.2 Spiral Algorithm 6
2.3 Snaking and Wall Follow Algorithms 6
2.4 Path Transforms 8
2.5 Genetic Algorithm 8

3 Methodology **10**
3.1 Simulation, Testing & Software 10
3.1.1 Used techniques 10
3.1.2 Assumptions 11
3.1.3 Robot 11
3.1.4 Movement of robot & Updates 11
3.1.5 Trail behind robot 12
3.1.6 Rooms 12
3.1.7 Algorithms Implemented 12
3.1.8 Runs 12
3.1.9 Displaying & Calculating Data 13

4 Hypotheses **15**

5 Results **16**

6 Discussion **22**
6.1 Limitations 25
6.2 Future Research & Improvements 26

7 Conclusion **27**

Bibliography **28**

Chapter 1

Introduction

In Greek and Roman mythology the gods of metalwork could build mechanical servants made from gold. In 1495 Leonardo da Vinci made a sketch of a mechanical knight that could move its arms and legs [3]. However, it is only in the past few decades that robots have become a reality. The resemblance is still somewhat the same to the servants and knights of old. We can compare the appliance of robotics within military purposes as the knights and the robotics for household applications as servants.

The use of robotics within everyday life has grown as new research and innovative ideas lead to new application possibilities. These applications often simplify and automate previously tedious work that needed to be performed by humans, e.g. heavy lifting, lawn mowing and warehouse logistics [2].

The robotics industry suffers from some of the problems the personal computer industry suffered from 30 years ago. In 2007, the author of [3] pointed out that lack of standards often resulted in designers having to start from scratch. However, the robotics industry can take advantage of the development of computers. Consider that sensors and computing power are becoming more convenient and cheaper. Furthermore, the increase in connectivity can push workloads from smaller units to larger computers or servers [3].

In 2008 toy robots largely out sold utilitarian robots [8]. In 2016 vacuum and floor cleaning robots made up the largest share of robotic units at work in households. In 2016 it was predicted that worldwide sales of household robots would rise from the 2015 value of 3.6 million units to reach around 30 million units within 2019. In 2016 vacuum and floor cleaning robots accounted for 96 percent of domestic robot sales. Robotic mowers and pool cleaning robots ranked second and third respectively [9].

A robotic vacuum cleaner is an electrical device that cleans floors in houses. The authors of [6] describes the *iRobot Roomba* 2005 as the at the time most popular robotic vacuum cleaner in the United States. The model *Roomba Red* is circular with a diameter of 33 cm and a height of 9 cm. It carries a removable dirt bin and is powered by two wheels, one on either side. Steering is performed by alternating the power to each wheel. The parts made for suction and cleaning looks much like a hand held vacuum cleaner. The Roomba has a brush to transport dirt to its suction area. It can clean for about two hours on a single charge. The self navigation of the robot is what makes it robotic. The biggest difference between the cheapest and the most expensive models is the extent and precision of the navigation system. *Electrolux* has a 2018 model called *PURE i9*, which uses a *3D Vision System*$^{\text{TM}}$. It uses cameras and lasers to navigate and has a charging station that it can return to by itself. The vacuum cleaner can be connected to the internet and be controlled via a mobile app.

Robotic vacuum cleaners need to be equipped with sensors and use path planning algorithms to be able to complete the task with good results. The path planning algorithms should be sophisticated in order to make educated assumptions about the operating environment and be able to react to a changing indoor environment.

The subject area of path planning is particularly interesting due to the number of variables that affect how well a robotic vacuum cleaner can tidy. For instance, consider the situation where the user requires a room as tidy as possible but within a shortened amount of time. Does the robot aim to cover the largest area as possible or try determining where the most dust is?

When developing path planning algorithms, developers must decide whether the robot will have some a priori knowledge of the operating space. Developers must also decide to what extent the robot should be able to localise itself within a room using sensors. The problem of covering every accessible area in the entire room is known as path planning of coverage region, PPCR. An efficient PPCR can be formulated as one that should be short in distance and contain a minimum number of turning angles and revisited areas [11].

When a robot issues a movement, the robot cannot always be certain that the movement was performed correctly. For example, a robotic vacuum cleaner might not turn exactly 45 degrees even though it itself thinks it has. This could be due to the lack of understanding the environment. A solution could be to use more advanced sensors. Adding more sensors generally results in more management of hardware and more algorithm design. These additions to the robot lead to greater development costs. Developers could assume that the robot must operate without knowledge of its surroundings or position.

Varying degrees of randomness can be used in path planning. *Husqvarna*, a manufacturer of robotic lawn mowers, say on their website regarding a lawn mower that moves in a random based pattern:

> *"After 20 years of testing, mowing a lawn in a seemingly random pattern has proven to be the most dependable way to cover the whole yard, without leaving spots. Other methods do not have the same precision and would not pose an improvement over the current random based system used today."* [1]

1.1 Aim & Goal

The goal of this paper is to attempt to highlight advantages and disadvantages of some path planning algorithms. Thus the questions we wish to answer are:

- Which of the proposed algorithms performs best?
- What factors affect the performance of the different algorithms?

To achieve this goal, the aim of this paper is a short literature study and a simulation where some of the discussed algorithms in this paper are tested.

Chapter 2

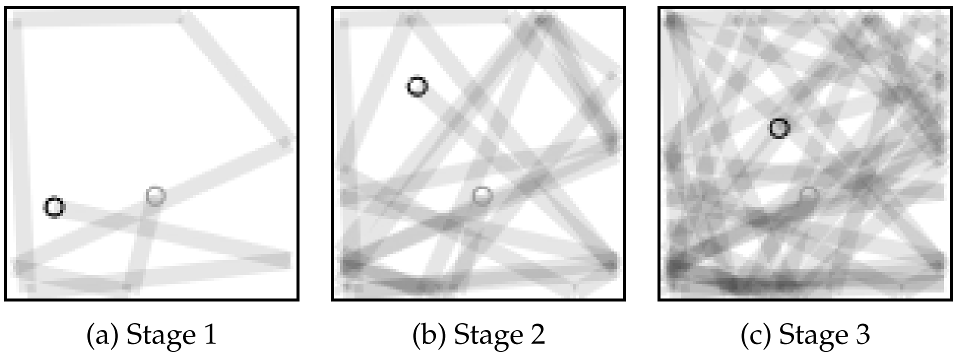
Path Planning

Path planning for autonomous cleaning robots is the act of using algorithms to decide how and where a robot should move in an environment [7]. The problem is often to decide how to completely cover a room. Covering a room can for example be done by following a precomputed path from start to finish, or by following certain rules along the way. All path planning algorithms rely on some type of knowledge of the operating space. In the case of an algorithm that uses a precomputed map the knowledge needs to be extensive. However, a simpler algorithm could only rely on the knowledge of hitting an object. In the latter case, the robot must be equipped with at minimum a simple bumper sensor that signals when the robot hits an object. With more sophisticated positioning equipment like IR sensors and GPS, algorithms that allow for more efficient path planning in terms of time and redundant (repeated) surface coverage can be developed [7].

Algorithms that generate a map of the surrounding environment and attempt to localise the robot within the map during runtime are known as SLAM-algorithms. However, for this thesis SLAM will not be discussed further.

2.1 Random Walk Based Algorithm

A subset of the existing path planning algorithms are random walk based algorithms [5]. Random algorithms require little from the robot in terms of hardware and computational power as the main aspect of the algorithm is random number generating. However, a result of today's robots being unable run endlessly (it has to run out of power at some point) is that random based approaches have no guarantee of cleaning performance within a given amount of time [7].



(a) Stage 1 (b) Stage 2 (c) Stage 3

Figure 2.1: Pictures of a random walk based path in different stages in a square room. Black circle is the robot, grey circle is start point. Turns are ongoing. A darker shade indicates that the robot has passed the area more times. Pictures taken from our simulation software.

Robots that apply a sense of randomness to their path are considered to use randomised path planning algorithms. These algorithms can be entirely random, e.g., after each collision with an obstacle the robot randomises a new heading without regards to where it has come from or been [4], see Figure 2.1. To this simplistic approach developers can then add special case handling such as getting stuck in small areas or corners [4].

A random walk based algorithm can be configured via sensors to be able to decide whether to turn left or right when it hits an obstacle (i.e. if an obstacle is sensed on the left, it will turn right). This way the robot can make ongoing turns from an object. Finally a random number is generated, and the random turn is performed thereafter. The manner in which the robot moves with this method depends strongly on its surroundings [10]. See Figure 2.1 for picture of our simulated random path.

2.2 Spiral Algorithm

A spiral path algorithm combines the use of making spirals and random turns [10]. This algorithm is much like the before mentioned random walk based algorithm, but when the robot senses that there is enough space it makes a spiral path. To begin with, the robot checks if there is enough space to start making a spiral. This is decided with the use of on board sensors. If there is enough space for a spiral, then continuous turns with increasing radius from the current position are performed. On a robot with two main wheels, like the Roomba mentioned in section 1, this could be performed by gradually levelling the amount of spin on each wheel. Starting with more spinning on the wheel on one side (sharper turn) and eventually giving wheels on both sides almost the same amount of spin. When an object is sensed, or there is not enough space for a spiral, the robot will go straight until a spiral can be made. The robot might make random ongoing turns on the way. Refer to Figure 2.2.

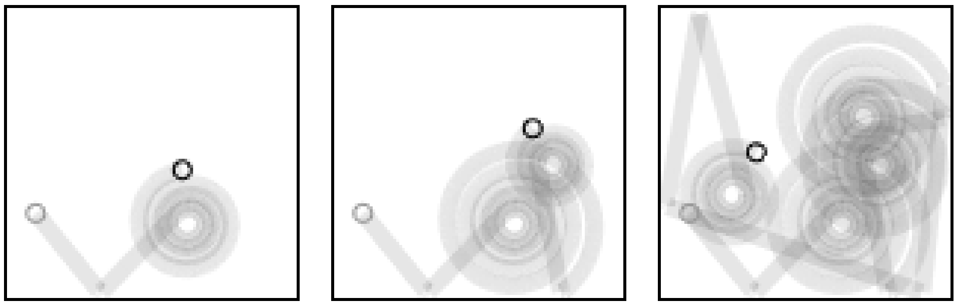

(a) Stage 1 (b) Stage 2 (c) Stage 3

Figure 2.2: Pictures of the spiral algorithm being simulated in a square room. As soon as the robot detects it is far enough away from the objects a spiral is initiated. While being too close to walls it performs a random walk. Robot in black circle, starting point in grey circle. Pictures from our simulation software.

2.3 Snaking and Wall Follow Algorithms

Path planning based on a snake-like movement is also common for domestic robots [7]. See Figure 2.3 for example of movement in a room. This pattern realistically generates many errors as it potentially must perform lots of stopping and rotating. Consider that wheels are not aligned or that sensors are cheap and imprecise. Over time errors will accumulate [7]. For this reason the authors of [7] suggest using the snaking pattern only for local, small areas that are cleared quickly in order to prevent positional errors accumulating. [7] combine the snaking path with a random walk path so that the random walk explores the room and the snaking pattern is used for the actual area coverage. They suggest this is more efficient than running either the random or snaking path separately.

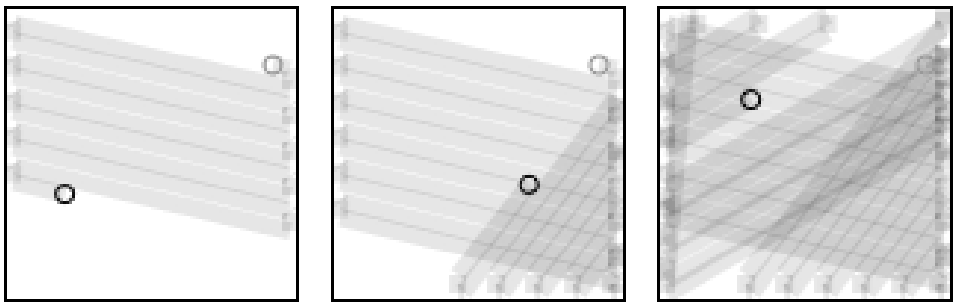

(a) Stage 1 (b) Stage 2 (c) Stage 3

Figure 2.3: Pictures of a snaking path in different stages in a square room. Black circle is the robot, grey circle is start point. Turns are ongoing. A darker shade indicates that the robot has passed the area more times. Pictures taken from our simulation software.

Wall follow paths are mainly used to allow the robot to gain knowledge of a room's perimeter. As presented by the authors of [4], the robot moves forwards until it collides with an obstacle. Thereafter it performs turns until the robot is no longer colliding and can continue along the edge of the obstacle. Wall following is particularly useful for clearing tricky spots such as corners. The authors of [4] suggest combining it with other techniques to achieve good coverage. Refer to Figure 2.4.

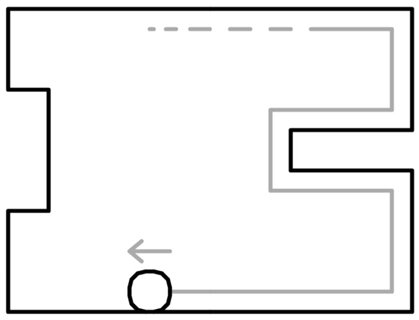


Figure 2.4: Picture of wall following robot with its trail in a room [4]. Each time the robot hits the wall it rotates clockwise until it can go forwards again.

2.4 Path Transforms

A path transform methodology is proposed by the authors of [12]. This solution handles the task of complete coverage from a starting point to a goal point in a known environment. The environment is imagined as a grid of square cells. The cells are given values depending on distance to goal, distance to obstacles and level of discomfort.

This approach is supposed to not only make a robot follow the circular contour patterns which radiate from the goal point, but also follow the shape profile of obstacles in the environment. This way creating an end result with fewer turns. A robot using this method needs great accuracy, and can not solely rely on dead reckoning. A sensor is needed. That sensor sights landmarks, which are compared with a precomputed map.

2.5 Genetic Algorithm

To minimise the amount of turns, trajectory length and multiple visits to the same place the authors of [11] have investigated an evolutionary approach based on genetic algorithms. The robot is assumed to be able to construct a complete map of the environment via sensory equipment. The complete map will consist of cells of the same size as the robot. The cells can have three different states: *unvisited*, *visited* or *obstacle*. The robot is then assumed to be able to recognise all these cells, in particular neighbouring cells, with the exact number depending on the length of a certain on board sensor, during its trajectory.

The approach uses a fitness function to optimise the robot's path. The fitness function (similar to natural selection) combines different solutions to create a new path. Every cell in the map gets a number and becomes a *gene*, and a partial paths consisting of the neighbouring cells within reach of a sensor becomes a chromosome.

The fitness function evaluates the partial paths possible to take from every position of the robot. It takes into consideration: the total length of the partial path, the number of unvisited cells it visits, and the total X-axis distance every visited cell has to the current position of the robot. The latter is supposed to make the evolution proceed with the partial paths that travel as little as possible in X-axis - meaning the complete path will most likely travel along the Y-axis, making turns and then travel back along the Y-axis but a bit farther away on the X-axis.

Chapter 3

Methodology

3.1 Simulation, Testing & Software

We have created software for simulating and testing the random walk based algorithm, the spiral algorithm and the snaking algorithm under different circumstances. We chose to create not only a graphical simulation but also graphic based testing to be able to overview the algorithms in a clear way, in order to make sure they perform as they should.

The software is written in JavaScript. The graphics are displayed as an *HTML canvas* within a web page. The robot in the simulation leaves a trail of covered area behind it as it moves. Pixels that have not been covered are left clear. The software periodically counts coloured pixels, where coloured pixels are considered as covered. The software also measures the total amount of turning the robot does.

Our software for the simulations can be seen on GitHub, see `https://github.com/jacobsorme/kexjobb`.

3.1.1 Used techniques

This section describes key points of what is used in this paper.

- *HTML* - HyperText Markup Language. Used to create web pages.
- *JavaScript* - A programming language widely used for web related programming.

- *Canvas* - Essentially a digital painting on the screen. In this paper an HTML Canvas is used.
- *HTML canvas getImageData() Method* - A method derived from the *HTML Canvas* used to receive all pixels and their values as a long list, an array. Used here to calculate amount of coverage of the canvas.

3.1.2 Assumptions

In order to be able to create a simulation environment within the given time frame, certain limitations and assumptions were made. The impact that these limitations have on our test results will be discussed in section 6. The following assumptions were made to the simulation and testing:

1. There are no obstacles within the rooms.
2. The walls of the rooms are orthogonal.
3. The robot has perfect and precise movement.
4. The robot seemingly uses only bumper- and distance sensors.
5. Turning the robot happens instantly.

3.1.3 Robot

The robot is represented with a circle. It has a diameter of 12 pixels. It has the ability to move anywhere in the room. A wall collision is detected if the centre point of the robot is less than 6 pixels (one radius from the center) away from a virtual wall.

In simulations the robot is shown. However, while performing the actual tests the robot is not shown on the canvas - only the trail behind it is shown.

3.1.4 Movement of robot & Updates

The robot can move in any direction. Positional precision is limited by the limitations of representing decimal numbers in JavaScript. The robot is moved in an *update*. Each update does calculations and moves the robot by a distance of 3 pixels in the robot's current heading. The updates will be counted in order to measure performance.

3.1.5 Trail behind robot

The robot leaves a 12 pixels wide (the diameter of robot) trail behind in a black colour with an alpha value of 0.1. The alpha value means that the trail is slightly transparent.

Since the robot moves 3 pixels in every update it will cover 36 pixels (12×3, a rectangle with sides 12 and 3) in every update. The trail does not include the circular shape of the robot since the actual vacuum part on the robot is a line on the robot's underside [6], and this is what leaves a trail.

3.1.6 Rooms

The square room has a side length of 200 pixels. The rectangular room has a width of 333 pixels and height of 120 pixels. Both of the rooms have a total area of (roughly) 40000 pixels. For example: if a real robotic vacuum cleaner has a 30 cm diameter, with our proportions the square room would have sides of 5 meters.

3.1.7 Algorithms Implemented

We have implemented a random walk based algorithm, a spiral algorithm and a snaking path. Refer to subsections 2.1, 2.2 and 2.3. The spiral algorithm had a margin (spiral-making threshold) of 70 in the 200x200 square room and a margin of 50 in the 333x120 room.

3.1.8 Runs

Measuring the *time* during a run is done by counting the number of updates of the canvas. An update of the canvas is equivalent to calculating the new position of the robot and drawing its trail. For each algorithm 2000 runs are performed in each room. The number of updates and turns will be stored at each percentage of coverage. The percentages of coverage examined are 1%, 2% ... 95%. One of the reasons we do not examine percentages greater than 95% is that we think that a percentage of *100%* is unrealistic in practice. Also, examining percentages above 95% will most likely be time consuming for random based algorithms. A run is defined as follows:

- The robot's starting position and starting angle is randomised in the room.
- The robot moves according to the applied algorithm.
- Calculations of current coverage are performed after every update of the robot by analysing the accumulated trail left behind by the robot on the canvas. Each pixel is examined via the function *getImageData* originating from the HTML canvas. If a trail was left on a pixel, meaning it is coloured, it is considered as covered. The amount of covered pixels are compared to the total amount of pixels, 40000, and the current percentage of coverage can be calculated.
- After each turn the turned radians are calculated. The robot will rotate in the direction that results in the shortest turn to get to the new direction.
- Once the robot reaches an integer percentage between 1 and 95, the current number of updates and total turns in radians are stored. In total, data will be stored 95 times during a run, for 1% coverage, 2%, 3% ... 95%.
- When the robot reaches 95% coverage the data is stored, then a new run is initiated.
- Before the canvas is cleared and a new run begins, the whole trail of the run is analysed. Data for a heat map is accumulated. The opacity of each pixel is added to a average of that pixel from previous runs.

3.1.9 Displaying & Calculating Data

The number of updates required for every one of the 95 percentage break points is averaged for the 2000 runs. For every percentage break point the amount of turning is averaged as well. A standard deviation for the updates and turns for each percentage is calculated. The data is presented as multiple numbers, that is collected to form graphs.

The heat map is displayed as a picture, a canvas. A blue colour indicates that a pixel was visited few times, while a red pixel indicates that the pixel was frequently visited.

Chapter 4

Hypotheses

We expect to be able to compare the tested algorithms as well as be able to look at the simulations and understand how they work and why they perform as they do.

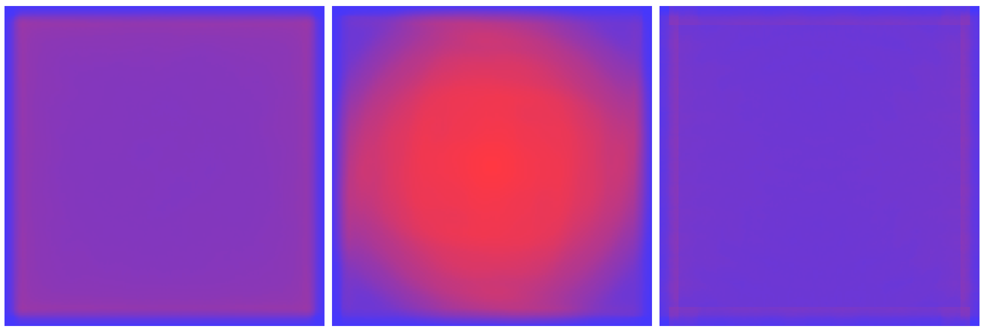
Our first hypothesis is that the random algorithm and the spiral algorithm will have similar performance, as stated in [10]. We believe that the spiral algorithm might have difficulties reaching the corners of the room.

Our second hypothesis is that the snaking algorithm will perform better than both the random- and snaking algorithms. Finally, we also believe that all algorithms will perform better in the beginning of a run than in the end. We believe that this is due to the fact that as the covered area grows, the probability that the robot finds uncovered area decreases.

Chapter 5

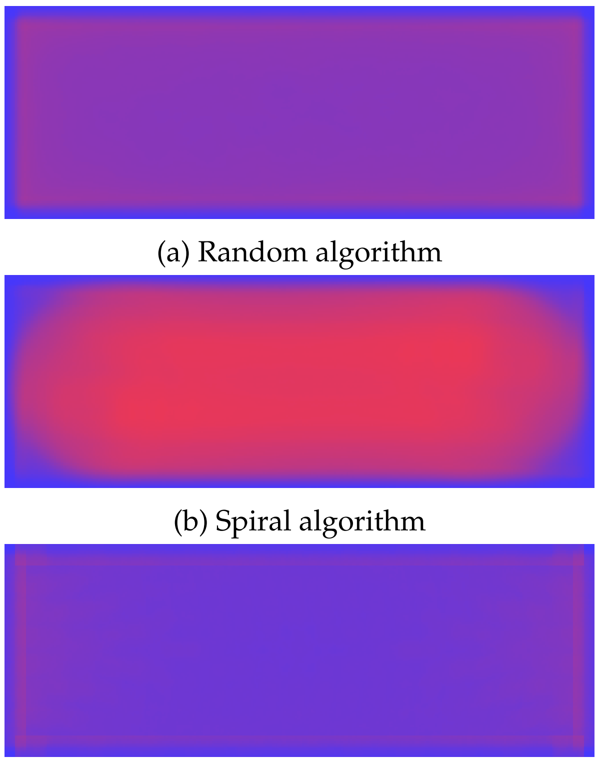
Results

Figures 5.1 and 5.2 show heat maps of the robot after 2000 runs in square and elongated rooms, respectively. A red colour indicates that the area was frequently visited by the robot whereas a blue colour indicates the opposite.



(a) Random algorithm (b) Spiral algorithm (c) Snaking algorithm

Figure 5.1: Figures showing the heat maps for the three tested algorithms after 2000 runs in the square room. A strong red colour indicates frequent visits, while a strong blue colour indicates few visits.



(a) Random algorithm

(b) Spiral algorithm

(c) Snaking algorithm

Figure 5.2: Figures showing the heat maps for the three tested algorithms after 2000 runs in the elongated room. A red colour indicates frequent visits, while a blue colour indicates few visits.

To achieve the maximum coverage of 95% the snaking algorithm needed the fewest amount of updates for both the elongated room and the square room. See Figure 5.3. The random walk based algorithm used fewer updates than the snaking algorithm up until around 50% of coverage in the square room. The random algorithm required fewer updates than the snaking algorithm up until 80% of coverage in the elongated room. Refer to Figure 5.3.

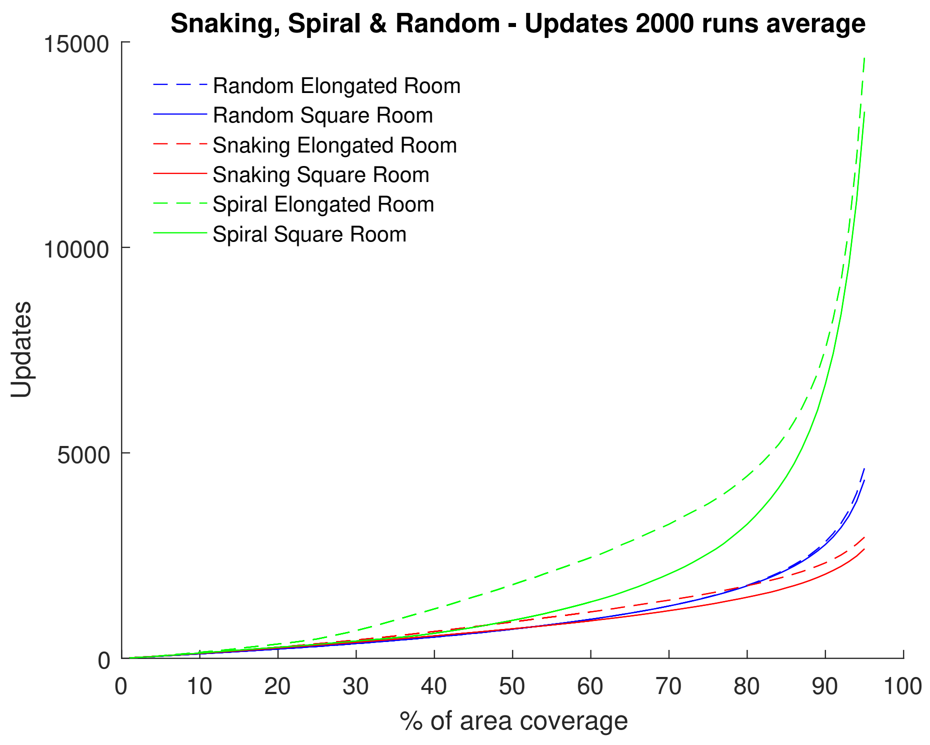


Figure 5.3: Graph showing average number of updates required to achieve each percentage of coverage. 2000 runs of the three algorithms in both rooms.

The spiral algorithm needed the most updates and turns to complete any percentage of coverage. Refer to Figure 5.3 and Figure 5.4. The spiral showed an increase in number of updates required for coverage after around 25% of the area was already covered in the elongated room, see Figure 5.3. A similar increase for the spiral algorithm in the square room happened at around 40% of coverage. The random algorithm showed to need fewer turns than the snaking algorithm. The largest difference of turning was in the elongated room, where the random algorithm needed around ten fewer complete turns to complete a 70% coverage. See Figure 5.4.

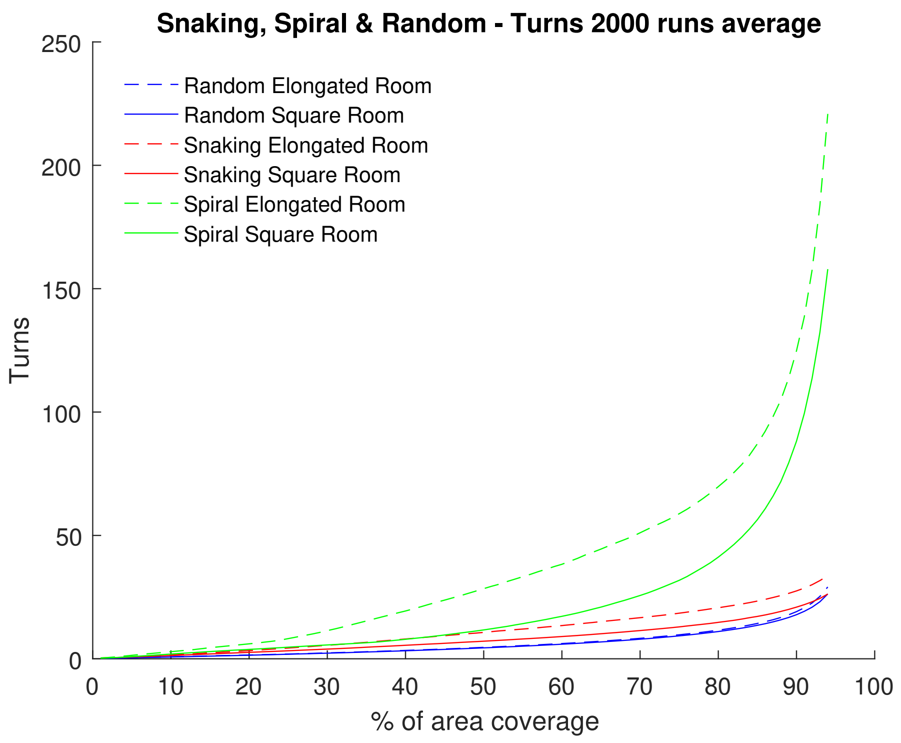


Figure 5.4: Graph showing average number of turns required to achieve each percentage of coverage.

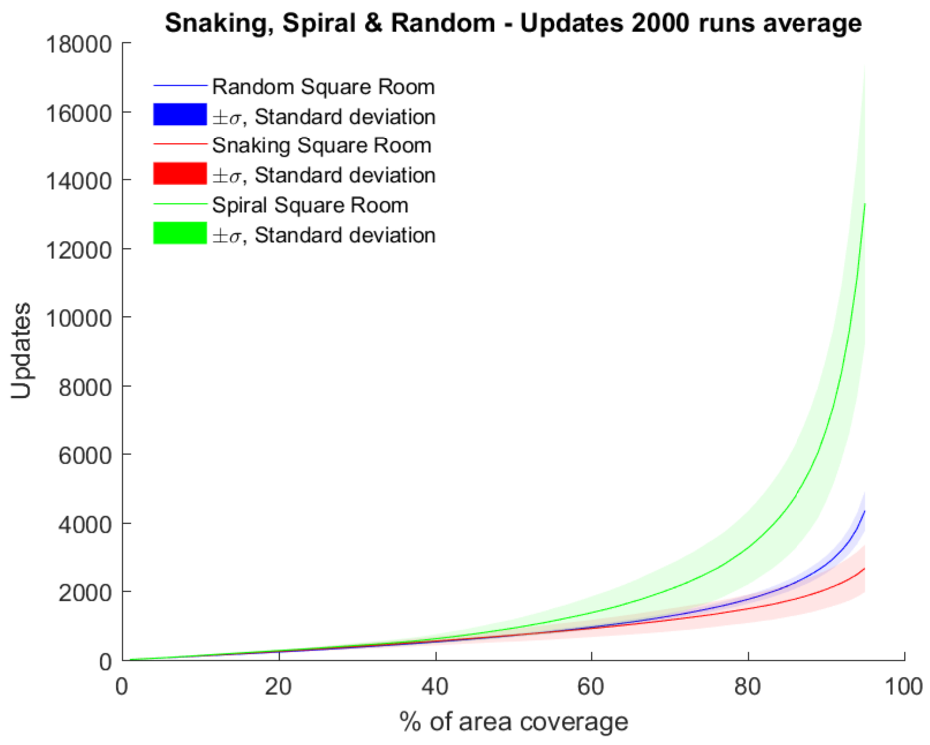


Figure 5.5: Graph showing average number of updates required to achieve each percentage of coverage, and the standard deviation. 2000 runs of the three algorithms in the square room.

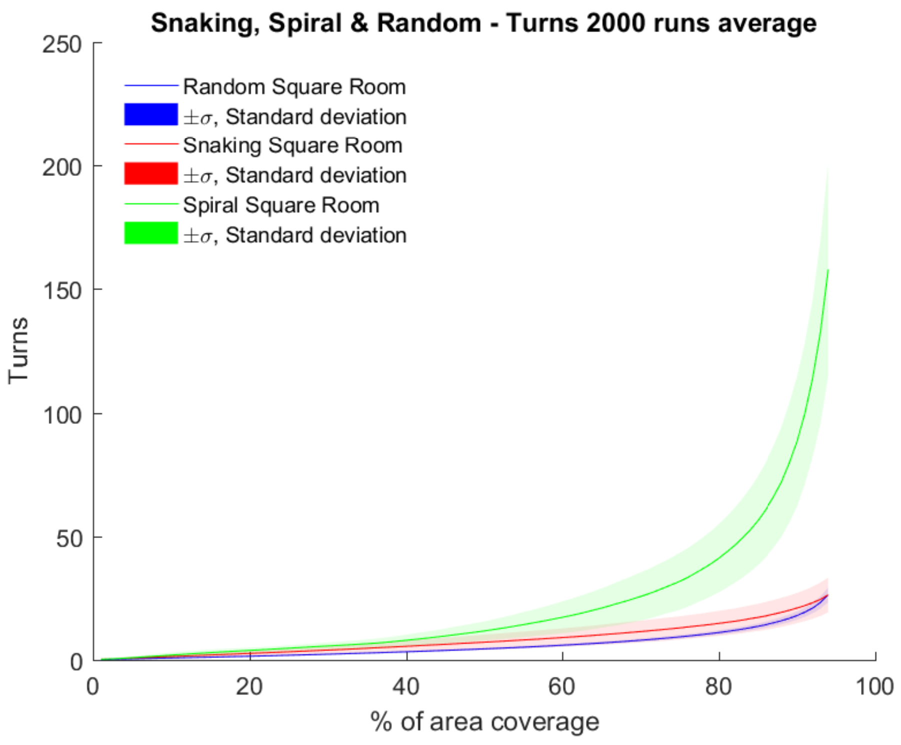


Figure 5.6: Graph showing average number of updates required to achieve each percentage of coverage, and the standard deviation. 2000 runs of the three algorithms in the square room.

Chapter 6

Discussion

We will continue to discuss the results. But what kind of results are desirable? It might actually be desirable to have repeated coverage to some extent, in order to achieve a clean surface. In our case this means that fewer required updates might not always mean a better result. Does the same goes for turns? Probably not, since turning does not contribute to covering or cleaning a room and is only time consuming [7]. Hence a great run would conclude of as few turns as possible and few updates, but not too few.

As initially believed in our first hypothesis (see Section 4) the spiral algorithm did not reach the corners of the rooms efficiently. We can conclude from the heat maps that the spiral algorithm showed a predominance of coverage in the central regions of the room. It made many spirals until it by chance bounced into a corner and covered that area. The behaviour of the spiral algorithm resulted in many visits to the central parts of the rooms. Refer to the heat maps for spiral algorithm in Figure 5.1 and Figure 5.2. We can conclude from our results that the spiral algorithm is not the best algorithm of choice. However, it would be interesting to examine the spiral algorithm in realistic rooms filled with objects. It might also perform better if combined with a wall following algorithm, in order to cover the corners efficiently. It might suffer more from its lacking ability to cover corners in a completely square room than in a realistic room filled with objects. As the authors of [6] mentioned, similar algorithms to the spiral algorithm are actually used in industry.

Furthermore, our first hypothesis stated that we believed the spiral- and random algorithms would have similar performance. From our results we can conclude that this did not turn out to be true. The spiral- and random algorithms initially performed similarly. However, at around 20 percent of covered area the performance of the algorithms diverged from each other. At the end of a run, at 95% of coverage, the algorithms differed by an average of around 10000 updates. The amount of turning for the algorithms differed with around 200 revolutions. See Figures 5.4 and 5.3.

We believed that the snaking algorithm would perform better than both the random- and snaking algorithms. See our second hypothesis, section 4. This turned out to be partly false. The snaking algorithm completed 95% of coverage with the fewest updates required, in both the square and elongated room. However, the snaking algorithm also required more turns than the random algorithm to complete the full 95% of coverage in the elongated room. For the square room both the snaking- and random algorithms required around the same number of turns. See Figure 5.4. When looking at the standard deviation, for both updates and turns, the random algorithm had less variation than both the other two algorithms. Despite requiring more updates than the snaking algorithm, the random algorithm deviated less from its mean. It is also interesting to discuss the performance at lower percentages of coverage than 95%. In both rooms the random algorithm required less turning than the snaking algorithm up until the last percentages of coverage. The same goes for updates up until 50% of coverage in the square room and up until 80% of coverage in the elongated room. It can be seen from our results that the random algorithm was better over all in the beginning of a run, while later on suffering. The snaking algorithm performed better in achieving the top percentages of coverage. See Figure 5.3.

Our belief that all of the tested algorithms would perform worse towards the end of the runs is supported by our results. We believe that this increases the probability of revisiting areas. Specifically, the spiral algorithm proved to be especially vulnerable to having large portions of the room already covered. This can be linked to the fact that the spiral algorithm had difficulties reaching the corners. The snaking- and the random algorithms also suffered towards the end of the runs, but at a lower degree.

The heat maps for both the random- and snaking algorithm showed a uniform coverage of the room. Most parts of the room had the same colour. For the snaking algorithm the colour was more blue, indicating fewer visits to the same spot. The part of the room the snaking algorithm visited the most was along the walls. See figures 5.1 and 5.2 in section 5.

The authors of [10] stated regarding the random algorithm and the spiral algorithm: *"... the manner in which the robot is moving depends strongly on its surroundings"*. However, it can be seen in Results (section 5) that the random algorithm had an almost identical average performance in a square and a rectangular room of the same size. On the other hand, both the spiral algorithm and the snaking algorithm performed better in the square room than in the elongated room. Refer to Figure 5.3 and Figure 5.4, section 5. However, the authors of [10] examined rooms with objects in them, which is something we have not tested. Hence we cannot discuss how a room with objects affects the performance of an algorithm.

The path transform algorithm and the genetic algorithm proposed by the authors of [12] and [11], respectively, assume that the robot knows its operating environment and that it always knows where it in the room. The robot is assumed to start at a optimal position and with a optimal angle. The environment is assumed to be represented as a grid, even if it contains obstacles. We believe that these assumptions make them hard to use in practice. Furthermore the performance is evaluated in terms of *complete* coverage, 100%, in the above mentioned papers and in [10]. As seen from our results it can be relevant to study the performance to achieve lower percentages of coverage than full coverage, something that neither of the authors of [12], [10] and [11] discussed.

The claim by the authors of [7] that the snaking algorithm performs a lot of turning is supported by our simulation results if we compare snaking with the random algorithm. A lot of turning means longer running times if the robot must stop and rotate. Because we assumed perfect motion, it is difficult to discuss the claim by the authors of [7] that errors accumulate fast over time for the snaking algorithm. It is however reasonable to consider that, in general, a robot with cheap sensors and poor build quality will generate larger errors as the robot turns. Also, if the path algorithm applies a sophisticated heuristic or relies on positional data, then errors will most likely have a negative impact on the robot's performance. Our tested algorithms rely on randomness to some degree, not the robots position. Since errors likely will affect the robot in random ways, the already used randomness in decision making will not be affected greatly.

6.1 Limitations

The drawings on the *HTML canvas* only have so many pixels. This means that the robot and the trace behind it can not be precisely drawn. Movement in every direction can not be correctly represented on the canvas, since the pixels are square. In each update the robot had a theoretical maximum number of pixels to cover. This maximum was sometimes exceeded due to imprecisions in the drawing of the trace.

The spiral algorithm in our simulations did not have a slower speed while turning. Depending on the implementation it is likely that turning will reduce speed in reality, since motors have to work on turning.

When the robot hits a wall a new angle will be calculated according to the algorithm being tested. Since *hitting a wall* is rather *passing a virtual line* or in our software, some problems do occur. See Figure 6.1. If the robot is in the room in the beginning of one update, and outside a wall after, it is moved into the room again in the next update - the update that detected that the robot *bumped* into a wall. When the robot is moved into the room from outside a wall the position is changed. For simplicity, only the position in one dimension is changed. For example, if the robot passes the right wall, the x is changed to be just inside that wall. The y-position is not changed - meaning that the position of the collision will be slightly off form where the *real* collision should have happened. This error is evaluated as minimal, however the speed of the robot is kept relatively low to be on the safe side.

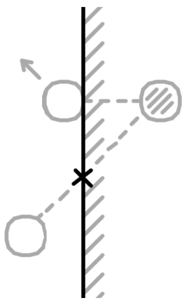


Figure 6.1: Picture of a robot (circle) in different stages of bouncing in our simulation. The cross marks the actual place of collision. The lined circle is where the robot is before it is detected that it has passed the wall. The circle with a direction of path marked is where it is move to after collision.

6.2 Future Research & Improvements

We did not simulate and test rooms of irregular shapes, and we did not have objects in our rooms. For more realistic results one could randomise the shape of the room and objects in it between runs, however keeping the room of the same total area all the time.

Studying the performance of different algorithms and coverage using our method - drawing on a canvas - was according to us a great choice.

Chapter 7

Conclusion

Our study and our results show that the random algorithm performed best in our simulations. The random algorithm and the snaking algorithm showed similar performance. The snaking algorithm required more turns, while the random algorithm required more updates. Since repeated coverage is sometimes desirable, but extra turns is not, the random algorithm is better. The spiral algorithm did not perform well in comparison. However, it would be interesting to combine the spiral algorithm with an algorithm made to cover corners and walls, such as the wall following algorithm. We believe that other solutions for path planning for robotic vacuum cleaners that have not been tested in this paper perform well in theory but might not be applicable in practice. This is due to the initial assumptions of the environment and capabilities of the robot.

Bibliography

[1] "Automower". In: *Husqvarna, website* (). Accessed: March 2018.

[2] Nai-Hua Chen and Stephen Chi-Tsun Huang. "Domestic technology adoption: comparison of innovation adoption models and moderators". In: *Human Factors and Ergonomics in Manufacturing & Service Industries* 26.2 (2016), pp. 177–190.

[3] Bill Gates. "A robot in every home". In: *Scientific American* 296.1 (2007), pp. 58–65.

[4] Kazi Mahmud Hasan, Khondker Jahid Reza, et al. "Path planning algorithm development for autonomous vacuum cleaner robots". In: *Informatics, Electronics & Vision (ICIEV), 2014 International Conference on*. IEEE. 2014, pp. 1–6.

[5] Han-Gyeol Kim, Jeong-Yean Yang, and Dong-Soo Kwon. "Experience based domestic environment and user adaptive cleaning algorithm of a robot cleaner". In: *Ubiquitous Robots and Ambient Intelligence (URAI), 2014 11th International Conference on*. IEEE. 2014, pp. 176–178.

[6] Julia Layton. "How Robotic Vacuums Work". In: *HowStuffWorks.com, website* (2005). Accessed: March 2018.

[7] Yu Liu, Xiaoyong Lin, and Shiqiang Zhu. "Combined coverage path planning for autonomous cleaning robots in unstructured environments". In: *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*. IEEE. 2008, pp. 8271–8276.

[8] Courtney Myers and Andy Greenberg. "Household Robots". In: *Forbes* (2008).

[9] "Press Release 20 december 2016". In: *The International Federation of Robotics, website* (). Accessed: February 2018.

[10] Krzysztof Skrzypczyk and Agnieszka Pieronczyk. “Surface covering algorithms for semiautonomous vacuum cleaner”. In: *ACMOS* 10 (2010), pp. 294–298.

[11] Mohamed Amine Yakoubi and Mohamed Tayeb Laskri. “The path planning of cleaner robot for coverage region using genetic algorithms”. In: *Journal of Innovation in Digital Ecosystems* 3.1 (2016), pp. 37–43.

[12] Alexander Zelinsky et al. “Planning paths of complete coverage of an unstructured environment by a mobile robot”. In: *Proceedings of international conference on advanced robotics*. Vol. 13. 1993, pp. 533–538.