Autonomous Robot Coverage Paths

Master's Thesis in Computer Science

Tommy Joe Lund

31st May 2017
Halden, Norway

Østfold University College www.hiof.no

Abstract

This thesis investigates the field of coverage path planning in two-dimensional space for the purpose of being used in autonomous robotic vacuum cleaners or similar mobile robots. Different path-planning algorithms were investigated, of which coverage heuristics proved most immediately pertinent due to their low cost. The problem that follows is how to ensure the best possible coverage rate, which is achieved by finding an optimal size for the robot, and exploring how scenes can be manipulated to best exploit the coverage algorithm.

A platform-independent model and a platform-specific model were developed to be employed on the robot, and a simulation environment was created to test the robot's coverage performance and to track statistics.

Various scenes were made to test the coverage performance under different conditions. These scenes reflect that the robot is assumed to be in an indoor environment containing obstacles.

Test results show that a circular robot's radius size has major effects on both coverage speed and maximum coverage potential. Small sizes are slow to get anything done, but eventually perform better than large sizes. Larger sizes are very good at quickly covering large open surfaces, but cannot reach all parts of the various scenes, and are easily outperformed by moderate sizes. The best coverage rate after a specific duration was shown to move towards smaller radiuses as the robot was given more time.

Obstacles introduce variance by dividing the scene into sections, creating areas that are difficult to reach, and areas where the robot is more likely to meander and repeatedly bump into walls and obstacles.

The maximum coverage rate is limited by obstacles. Some space is occupied by the obstacle itself, but they also make sections of the scene inaccessible for radiuses above a threshold. Obstacles also introduce areas which cannot be covered because of the robot's geometry.

A set of guidelines have been defined to assist in selecting the best possible robot size for any scene. The key takeaway from these guidelines is to choose a moderately sized radius if both time and coverage completion are important factors, and it is safer to select a slightly too small robot that will perform better over time, than a slightly too large robot that will comparatively do worse over time.

Keywords: autonomous robot, coverage path planning

Acknowledgements

I would like to thank my supervisor, Øystein Haugen, for all his help and support.

Contents

List of figures

List of tables

List of listings

Chapter 1

Introduction

We are interested in autonomous mobile robots that cover surfaces, which is a problem known as Coverage Path Planning (CPP). There are many applications for CPP algorithms, such as lawn mowing, cleaning, painting, and mine clearance, however the main consideration in this thesis are indoor floor-covering robots like vacuum cleaners, and other robots which routinely traverse the same environment.

Because the environment we have in mind is indoors, we can make some assumptions about boundaries, surfaces, and navigation. There is no need for additional measures to limit the areas the robot can access, because the perimeter is defined by walls and it is impossible for the robot to exit the site. The surface is generally friendly without too many challenges, as it is largely flat. There are soft obstacles like doorsteps, carpets, and wires that the robot must be able to handle, but the robot does not require an elaborate system of transportation or safety features like cliff detection. Nor does the robot require an expensive method of navigation. It is not reliant on GPS or other satellite systems for localisation. It can instead use cheaper sensors in combination walls or other local landmarks to navigate, and use a basic floor plan as a map. Walls can make environments rectilinear in nature, a feature the robot can take advantage of whenever possible: covering long stretches in straight lines. All of these assumptions can be used to improve performance and reduce costs.

Various obstacles are present in the environment. These are objects such as chairs, tables, and shelves—things one typically find in homes and offices. It could also be practical to create a category of virtual obstacles as done in [8]. These obstacles are created from areas in which the robot has become stuck. The problem areas can then be ignored and covered separately after the other areas are completed.

Many of the commercial products available to date are based on heuristics and the idea that if the robot covers the surface randomly, it will eventually cover the surface in its entirety. Examples of such robots are the Roomba by iRobot, RC3000 by Karcher, and Trilobyte by Electrolux [16].

We want to investigate how CPP algorithms can be used in situations with various obstacles, and how the robot, the obstacles, and the environment could be fit with sensors and actuators to facilitate the task at hand. There are three main topics of literature required to achieve this goal:

Algorithms. We need to know what kinds of CPP algorithms exist, and which ones are best suited for our purposes.

Hardware. The actual sensors we use are dependent on the algorithms we want to use, and what other kinds of features we want the robot to have.

Testing. It is necessary to have sound testing environments and procedures that can be used for consistent assessments of different robot models and versions.

As for the rest of the report, Chapter 2 provides a literature review and a background for what methods have been chosen, including the choice of algorithms, hardware, and test procedures. Chapter 3 contains a description of the project's methodology, Chapter 4 describes the software in detail, experiment results are found in Chapter 5, the discussion is in Chapter 6, the report is finally concluded in Chapter 7.

Chapter 2

Background

2.1 ThingML

The program has been written in ThingML[1]. ThingML is a domain-specific modeling language named in reference to the Internet of Things. It is based on state machines, and is an excellent tool for modeling systems that are intended to be predictable and target multiple platforms. Each state machine is called a Thing, and they have an assortment of features like different states, properties, and ports. Things work concurrently together, and communicate with messages sent through ports.

ThingML provides an automatic generation of mock interfaces which allows direct manipulation of messages. Examples can be found in Figure 4.2 and Figure **??**.

Not all functionality is implemented directly in ThingML because it is a domain-specific language, but other languages can be used in conjuction with it, in this case Java. There are two ways to do this: writing Java code directly inside ThingML source code using kickdown, or by writing Java code, installing it on your system as a Maven project, and importing it into ThingML as a dependency.

ThingML excelled at programming the PIM and the PSM which were primarily dependent on ports and messages, however the Simulation Environment was slightly trickier. Functions and arrays in particular behave somewhat unexpectedly, and it would perhaps be recommended to outsource most of these tasks to Java by using a Thing that functions as an interface between the PSM and a Simulation Environment coded in Java.

[1] http://thingml.org/

2.2 Search methods

A few methods have been used to find literature, and they can be summarised as follows:

1. Search. Most references have been found by searching an online database. Search queries can be found in Table 5.2.

2. Bibliographies. Relevant sources from the References section of papers have been selected and included here.

3. Cited origin. Google Scholar provides a way to see which works have cited a specific paper. No new literature was found using this method.

4. Author. When a reference has been very relevant, the authors' other works have been investigated because it is often the case that they have several publications in the same field. No new literature was found using this method.

Every reference has been provided with a source of origin in Table 2.2. The reference itself is listed in the first column. The second column contains the source where it was found, which can be a search label, an author's name, or another paper. The third column describes the method which was used to find the reference.

Table 2.1: Search queries

Search	Database	Recent	Results	Search query
S1	Google Scholar		62400	robotic cleaning
S2	Google Scholar		113000	robot coverage
S3	Google Scholar		3570	roomba coverage
S4	Google Scholar		80100	robot area coverage
S5	Google Scholar		484000	complete coverage navigation
S6	Google Scholar		496000	coverage path planning
S7	Google Scholar		33600	robotics coverage survey
S8	Google Scholar	✓	2950	cleaning robot coverage
S9	Oria	✓	31	cleaning robot coverage
S10	Google Scholar	✓	33	"cleaner robot" "machine learning"
S11	Google Scholar		7410	"mobile robot navigation" survey

Table 2.2: References and how they were found

Paper	Origin	Method
[15]		
[5]	S2	search
[4]	S7	search
[20]	S8	search
[13]	S8	search
[8]	S8	search
[10]	S9	search
[19]	S9	search
[6]	S9	search
[11]	S9	search
[21]	S10	search
[3]	S11	search
[12]	[5]	bibliography
[2]	[5]	bibliography
[7]	[5]	bibliography
[14]	[5]	bibliography
[16]	[5]	bibliography
[17]	[5]	bibliography
[18]	[5]	bibliography
[22]	[5]	bibliography
[23]	[5]	bibliography
[1]	[4]	bibliography
[9]	[8]	bibliography

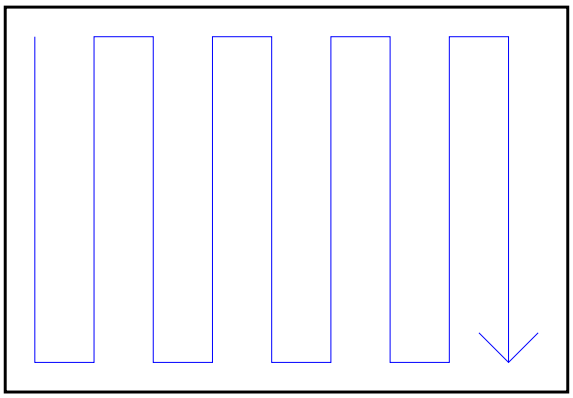

Figure 2.1: Zigzag motions

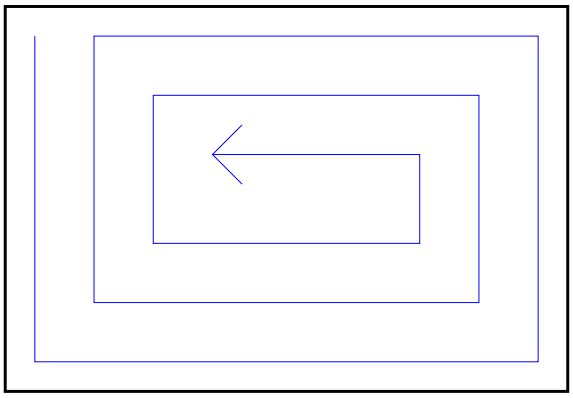

Figure 2.2: Spiral motions

2.3 Algorithms

We need to know what algorithms exist within the field of Coverage Path Planning (CPP) to get an idea of exactly what improvements can be made. Two surveys have been published in the field [4, 5]. Choset's survey from 2001 summarises CPP algorithms for mobile robots operating in the plane, and Galceran and Carreras' survey focuses on achievements made in the decade leading up to 2013, including work done in three-dimensional space. But because our robot exclusively operates on floors and similar surfaces, only planar algorithms have been considered.

There are two types of coverage methods, heuristic and complete [4, 5]. Heuristics are methods which do not rely on any kind of maps and cannot guarantee complete coverage. The archetypal example of a coverage heuristic is random search. This is a looping two-step process where the robot moves in a straight line until it reaches an obstacle, then turns in a random direction (Figure 2.3). Complete coverage methods decompose the environment into smaller segments called cells. In many cases, these cells can be covered with simple motions. Two typical patterns are the zigzag pattern in which the robot covers an area in parallel lines (Figure 2.1), and the spiral pattern where the robot covers the area in a motion resembling a spiral (Figure 2.2).

Choset [4] suggested four categories of CPP-algorithms and his taxonomy has been used widely in the literature. The categories are heuristics, approximate cellular decomposition,

semi-approximate cellular decomposition, and exact cellular decomposition. However, Galceran and Carreras [5] argue that there are qualitatively different approaches within these categories, and further categorisation can be made. Choset's taxonomy has been used in this chapter, but Galceran and Carreras' considerations have been taken into account.

Choset [4] further proposed independently categorising CPP methods as either offline if the environment is known and the planning can be done in advance, or online if the environment is fully or partially unknown and sensor-data must be used to understand the environment. Online algorithms are also known as sensor-based coverage algorithms. In our case the environment is only fully unknown the first time the robot discovers the area, and is subsequently partially unknown because of dynamic obstacles. In practice, this means we could map the environment separately and use an offline algorithm, or we could use an online algorithm.

Simultaneous Localisation and Mapping (SLAM) is a technique in which the robot generates a map of the world whilst tracking its location within it. SLAM has been used successfully in some autonomous cleaner robots such as the Neato Botvac. It can be practical to use this method for purposes such as combining mapping and coverage the first time the robot navigates the environment, and classifying obstacles in subsequent runs. It could make it possible for the robot to learn where dynamic obstacles are likely to occur, or if changes to static obstacles have been made. More details on SLAM can be found in Chapter 2.4.

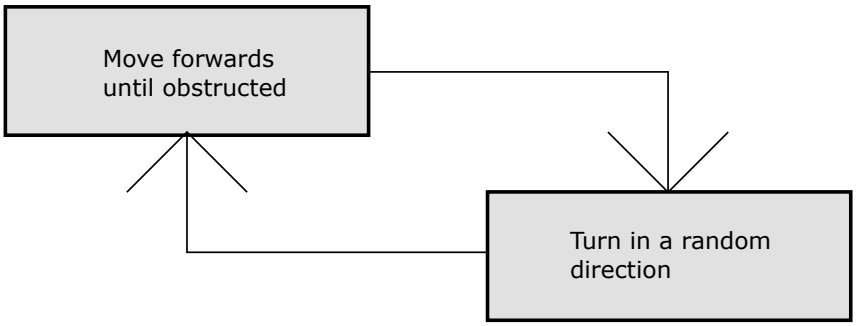

Figure 2.3: The process for random search

2.3.1 Heuristics

Coverage heuristics are algorithms which follow a set of simple behaviours. They do not guarantee complete coverage of the environment, but are still useful because they provide a lot of utility. The advantage is they do not need all the sensors which more advanced algorithms require, and so they are cheaper to produce.

Examples of heuristics include following walls, moving in spirals, avoiding obstacles, etc. The most notable example of a heuristic is random search.

Random search

Random search is a simple approach where the robot moves in a straight line until an obstacle is found, then it will turn in a random direction, and the procedure is repeated. Even though this method does not guarantee complete coverage, it is a reasonable solution in cases where the area is small—a typical situation for domestic cleaning robots.

The greatest advantage of this method is that it is unnecessary to have a lot of sensors or to have any expensive computational resources [5], because it only needs sensors for collision detection as seen in robots like RoboNet-1 [16]. However, the main disadvantage of this method is that it is very repetitive, and it will cover the same areas repeatedly [16].

Many commercially available cleaner robots use this method, such as the Roomba by iRobot, RC3000 by Karcher, and Trilobyte by Electrolux [16].

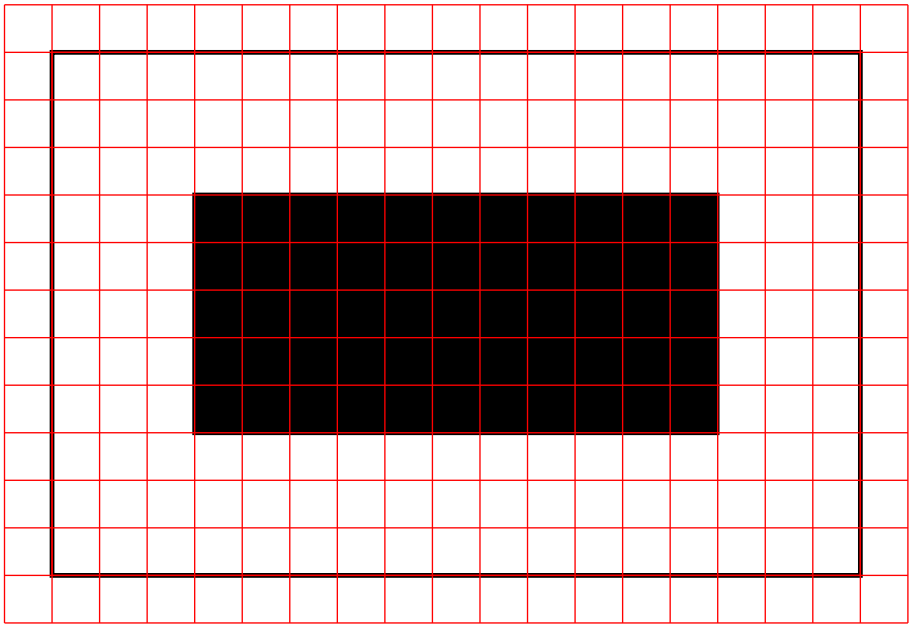

Figure 2.4: Approximate cellular decomposition

2.3.2 Approximate cellular decomposition

Approximate cellular decomposition methods create a map of the environment and divide it into a grid of uniform cells, typically squares [5], as shown in Figure 2.4. The grid can easily be represented by an array where each element of the array is a cell in the grid, and contain information such as if the cell has been visited or if it contains an obstacle. This method requires the robot to know where exactly it is located on a map at all times. The biggest problem with this method is the exponential growth of memory usage.

Galceran and Carreras [5] note that this category of methods is suited for indoor mobile robots because of the relatively small size of the area.

A common approach is to assume that once the robot has entered a cell, the cell has been covered. When this is used, the size of the grid cells must reflect the robot's effective coverage range.

Most grid-based methods use rectangular cells, but other options such as triangular cells [14] and hexagonal cells [17, 18] are available. Oh et al. [14] find that their paths are shorter and more flexible with triangular cells, however Galceran and Carreras [5] report that this difference is negligible and can be solved by increasing the resolution of the rectangular grid, and that robots typically are not able to make such fine movement adjustments. The improvements of the triangular cells comes from the fact they have 12 adjacent cells that can be used as navigational directions, whereas rectangular cells have 8. Hexagonal cells were not included in the study, however they have only 6 adjacent cells.

The first grid-based method introduced in the literature, according to Galceran and Carreras [5], uses distance-transforms [23].

Distance transforms

The distance transform algorithm, also known as the wavefront algorithm, works by first setting a cell as a goal, and giving every other cell a distance from this goal cell [23]. Every

Listing 2.1: Distance transform algorithm [23]

```
Set Start Cell to Current Cell
Set all Cells to Not Visited
Loop
        Find unvisited Neighbouring cell with highest DT
        if No Neighbour Cell found then
                Mark as Visited and Stop at Goal
        if Neighbouring Cell DT <= Current Cell DT then
                Mark as Visited and Stop at Goal
        Set Current cell to Neighbouring cell
Loop End
```

cell which is adjacent to the goal cell will get a distance of 1, every cell which is adjacent to those cells and not already assigned a distance will get a distance of two, and so on. A starting location is then chosen, and the robot will travel to the cells that are farthest away from the goal cell until the goal has been reached. The algorithm does not ensure an optimal path, but it still makes very few secondary visits to cells. The article provides pseudo-code describing the algorithm, and can be seen in Listing 2.1.

Genetic algorithms

An evolutionary approach using genetic algorithms (GA) was suggested in [21]. Genetic algorithms are algorithms which optimise problems via of natural selection. The parameters they used to optimise paths were the number of turns, the number of revisited cells, and the trajectory distance.

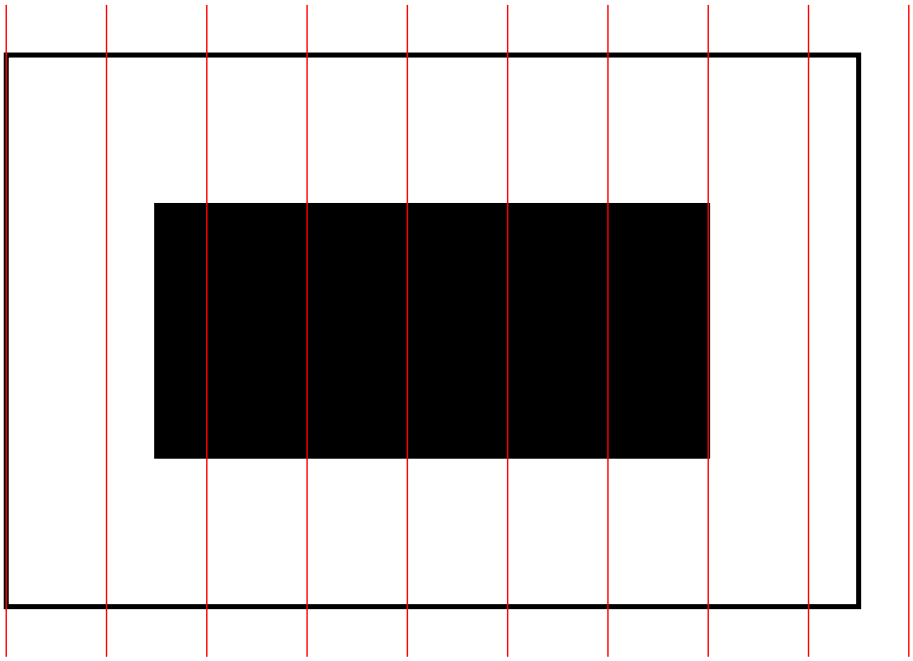

Figure 2.5: Semi-approximate cellular decomposition

2.3.3 Semi-approximate cellular decomposition

This type of decomposition divides the environment into cell slices of equal width as shown in Figure 2.5. The top and bottom of these cells can take any shape. These cells can be traversed with simple motions such as the zigzag pattern (Figure 2.1).

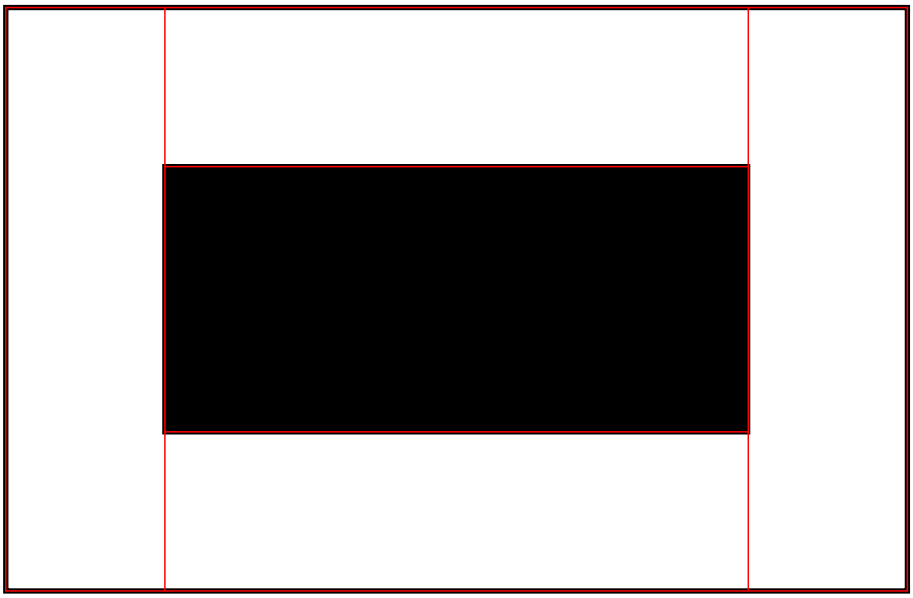

Figure 2.6: Exact cellular decomposition

2.3.4 Exact cellular decomposition

Exact cellular decomposition methods break all the free space of the environment into non-overlapping cells as shown in Figure 2.6. These cells can be represented by an adjacency graph. In this graph, the nodes represent cells, and the nodes are connected by an edge if the cells are adjacent to each other. In order to achieve coverage, a path through the nodes needs to be determined, however it is also necessary to create a path through each cell. This can easily be achieved because the cells do not contain any obstacles, which means the cells can be covered by a simple zigag motion (Figure 2.1).

Two offline algorithms in this category of methods are trapezoidal decomposition and boustrophedon decomposition.

2.3.5 Algorithms summary

We have looked at the different methods and want to explore how it relates to us and how we might be able to take advantage of them.

Heuristics are interesting because they only require basic sensors, but are still effective for the situations our robot will find itself in. Affordability is an important factor for products in the consumer market, so it is sensible to find improvements in the less costly alternatives.

Approximate cellular decompositions would let us mark virtual obstacles on the map as seen in [8]. This type of decomposition is particularly good for this use because the shapes of the cells would not be altered in any way. Approximate cellular decompositions would also combine well with SLAM algorithms that represent the map as a grid. Grids also form a good basis for genetic algorithms as seen in [21].

Semi-approximate cellular decompositions could allow us to optimise individual cells. However, they are less practical than the other methods because obstacles can prevent cells from being completely covered without being revisited or introducing significant travel time. In contrast, approximate cellular decompositions have the advantage that cells are small and can be marked as covered as soon as the robot enters it, and the cells of exact cellular decompositions do not contain obstacles at all.

Exact cellular decompositions could also allow us to optimise individual cells. For example, some cells could in practice be better to traverse in a zigzag motion or a spiral motion, and it might be better to use a heuristic fallback if a cell frequently contains dynamic obstacles. Another advantage of using this decomposition method is that the cell order can be traversed in a different order in a computationally efficient manner without overlap.

2.4 Hardware

The hardware of the robot must offer all the functionality the algorithms require. Some algorithms need more computing power or memory than others, or they need specific sensors or accurate localisation abilities. It is also dependent on scale. The hardware requirements of an algorithm can scale with the complexity of the environment or the size of the environment.

There is, of course, also an engineering aspect to hardware. The shape and size of the chassis is an interesting problem. It can be good for a cleaning robot to be quite flat in order to gain access to more areas, and it could be wide so it can clean a larger surface at a time. However, a flat robot will have less carrying capacity and less room for hardware components, a wide robot will lose some of its flexibility, and a heavy robot will require more energy to operate than a light one. There is also the issue of cost. Domestic robots cannot use the most expensive hardware on the market, but have to accept cheaper components, and as a result they do not have the same capabilities as an industrial robot with a larger budget would have.

It is also pertinent to see what hardware has been used for robots in other studies. This allows us to see what problems might occur with some approaches. Some examples are included here:

- A study from 1988 [22] details a basic cleaning robot. It uses 10 ultrasonic sensors to detect walls and obstacles. There are 6 of them at the front at 3 different heights to get an approximate height of the obstacles in front of them, and 2 on either side to detect obstacles on their respective side. It also has 3 bumpers for collision detection—2 in the front and 1 in the back.

- The RoboNet-1 is a proprietary robot developed by Palacin et al. [16] based on random movement after a collision. It has redundant methods for collision detection, with one big frontal bumper, an accelerometer, and motor current monitoring. It drives using two DC-motors.

2.4.1 Sensors

The different features of the robot require sensors and actuators, and they are determined by what features they cover and can be found in this section.

Collision detection

Bumpers are useful for collision detection and can be placed in various places on the chassis. The RoboNet-1 [16] also uses an accelerometer and motor current monitoring to detect collisions.

Obstacle detection

Obstacle detection is typically done by using distance sensors such as infrared or ultrasonic sensors, but can also be done in more elaborate ways such as using two cameras to create a depth map. Our robot only needs to move in two dimensions, so a simple approach is likely sufficient.

Yasutomi et al. used ultrasonic sensors at multiple height tiers to detect the height of their obstacles [22].

Orientation

In some cases, it can be helpful to know the physical orientation of the robot. Gyroscopes and accelerometers can be used for this.

Localisation

Algorithms may be dependent on the robot knowing its location within a certain margin of error. Zelinsky et al. [23] suggest using beacons. A GPS, compass or magnetometer can be used for navigating the environment, however our robot will stay indoors and can use the perimeter of the environment for navigation. A bumper can be used to detect walls, and then the robot can turn and move in a zigzag motion (Figure 2.1). Another option is to use distance sensors to calculate the robot's position.

Some autonomous lawnmowers define a perimeter with a wire one can lay down, and use the area within as a working surface.

Mapping

Mapping is a necessity unless a heuristic is used. The map can be generated by the robot itself, or it can be provided by another entity. A basic map can be created by the robot if it drives along the walls of the room [14].

A robot which doesn't generate its own map can be provided with one. For example, some smartphone apps are able to generate floor plans.

Simultaneous localisation and mapping

Simultaneous localisation and mapping (SLAM) is a method that can be used for mapping the environment whilst keeping track of the robot's location within it. The problem with this method is that accurate mapping requires accurate localisation, and accurate localisation requires an accurate map.

SLAM suffers from the problem that it is often computationally expensive and requires large amounts of memory [9]. Kuo et al. [9] developed a lightweight SLAM-algorithm for indoor use, based on Rao-Blackwellized particle filters (RBPF). Their maps use line-segments rather than grids, which helps reduce the computational cost.

Optional features

Some features can be useful, but not entirely necessary. Safety features such as cliff detection can prevent the robot from harm, and will allow the robot to classify more types of obstacles.

2.4.2 Prototyping platform

We have explored prototyping platforms for the robot, and have specifically investigated the Arduino, Raspberry Pi, and the iRobot Create, but other options are available such as fully custom robots.

The Arduino platform is an open-source microcontroller. The Raspberry Pi, on the other hand, is a small computer, but it does have similar capabilities for programming as the Arduino. Various kits for creating mobile robots are available on both platforms. Both the Arduino and the Raspberry Pi would require a chassis and additional components like sensors and actuators to make a complete prototype, but kits include various sensors, and it might not be necessary to add more.

The iRobot Create is a programmable robot based on the Roomba. It does not come with any advanced sensors, but it can be used in conjunction with an Arduino or a Raspberry Pi.

2.5 Testing

We need a good testing environment which can be used repeatedly to test with different robot models and different iterations of our robot. The most central thing to test for every type of robot is its ability to cover a surface, and is detailed in Section 2.5.1. Robots also have a practical purpose they need to fulfil, but testing this is out of scope for this report. Some examples are still included:

- A cleaning robot must be tested to see how good it is at cleaning, and this could be done by using simulated rubbish [15].
- Lawnmower robots would need to test their ability to cut grass. In this case, it is not only important to get full coverage, but it is also important not to damage the grass more than necessary. In some cases it may also be of interest to cut the grass in specific patterns.

2.5.1 Coverage

The two main things to consider when testing coverage are completeness and overlap. In some cases it may be acceptable for the robot to cover the surface incompletely if it is more important for the robot to be finished as quickly as possible, or it may be acceptable for the robot to do some overlapping because the obstacles in the environments have moved.

Some algorithms may work better than others in different environments. For example, random path planning is usually quite effective in small areas.

In the case of cleaners and lawnmowers, there is an obvious way of testing basic coverage. If an area is clean or cut, it has been covered at least once. A study using this method was published in 2002 [15].

Palacin et al. developed a system to evaluate the coverage performance of cleaning robots, based on machine vision [16]. They set up a basic test scenario, with no obstacles, for the robots to clean, and mounted a camera above it to record the robot as it cleaned the surface. This setup allowed them to find various metrics like the speed of the robot and the number of collisions and turns, but more importantly, they were able to create a map to see the number of times an area had been cleaned. The robots used in the study were based on random path planning.

2.5.2 Battery life

The battery life of mobile robots is a problem, and it is ideal to have a robot which can operate for as long as possible.

2.5.3 Simulation

CPP algorithms can be simulated, and they often are before they are tested on an actual robot. An example where this has been done is [11].

A big advantage of simulation is that it makes it possible to optimise for specific variables. CPP algorithms can be shown to be mathematically optimal, but their performance is affected by the robot's parameters. In a simulation, individual parameters can be more easily altered, and the impact can be measured without elaborate setups. Changes which

are difficult or expensive to test on a prototype can more easily be changed in a simulation, and simulated tests can be executed more quickly than their real-world counterpart.

There are, of course, disadvantages as well. Real-world results can differ significantly from simulations if there are variables which are not accounted for. For example, the turn rate of a robot affects how quickly it is able to complete the coverage, and moving in straight lines allow the robot to move at greater speeds. If such variables were neglected, the simulated model would not be a complete test.

2.6 Other related topics

This chapter contains works that may be of interest for the thesis, but which are not the main focus. The works featured here are only be tangentially related.

2.6.1 Multiple robots

Multiple robots [7, 2] can be used for purposes such as speeding up a task, or combining multiple tasks. One can imagine a swarm of specialised cleaning robots that work in unison, where e.g. one type of robot does vacuum cleaning, another type does rough cleaning, and a third type of robot cleans areas that are difficult to reach for the other robots. Or there may be multiple of the same robot that try to finish a coverage job faster than if there was only one.

2.6.2 Autonomous underwater vehicles

Much work in CPP has been done on the topic of autonomous underwater vehicles (AUVs) [17, 18]. They appear to be a major focus of research done in the field of CPP in recent years, and it might be worth looking deeper into this. However, the research is mostly focused on three-dimensional space, which is less relevant to us than work done in two-dimensional space.

2.6.3 Chaotic algorithms

Most algorithms move in very predictable ways, which is a major concern for some applications such as security patrolling. Chaotic paths [20] afford the ability to move in unpredictable patterns whilst still guaranteeing complete coverage.

2.6.4 Social robots

Social robots that talk to its users exist in Japan. The Cocorobo is a singing cleaner robot by the Sharp Corporation. Sharp teamed up with Yamaha and released a new Vocaloid in 2016. Vocaloids are voice synthesizers that takes the form of singing idols.

Voice recognition software has become a popular trend in recent years, with products from Amazon, Apple, Google, and Microsoft. It would be interesting to explore what effects voice communication has on coverage performance in domestic and office environments. Smart phones or smart speakers like the Amazon Echo could be used to instruct the robot to initiate a coverage run, or to give more specific instructions. Allowing users to micromanage the coverage algorithms would also serve as a way to gather data which could be used to improve coverage performance locally or to be used in a larger machine learning dataset. The robot could also prompt the user for instructions. For example, if the robot knows there is an area it cannot access underneath a desk, it can ask the user to move aside.

2.7 Research Questions

So far we have seen what types of algorithms exist within the field of coverage path planning (CPP), we have looked at what kinds of hardware we might want to use for a prototype robot, and we have looked at different kinds of testing approaches.

Coverage path planning (CPP) algorithms can be classified as either heuristic or complete. Complete algorithms can further be broken down into three categories of decomposition after Choset's taxonomy [4]: approximate, semi-approximate, and exact. Heuristics afford the lowest cost of all the options, and are thus the primary focus for the rest of the thesis.

In terms of hardware, we have looked at different functionalities the robot could have, and what kind of sensors can be used to implement them. The choice in algorithms means a sensor to detect collision in the front of the robot is definitely needed, and a bumper is a good choice due to the nature of the indoor environment.

For testing, we have decided to simulate the problem and focus on total coverage over time. To do this, we need a software application which can run on both a physical robot, and in a simulation. A great programming language of choice for this is the state machine-based ThingML which provides excellent options for developing for multiple platforms. Java can supplement ThingML in the areas of the simulation and graphics that ThingML is not so good at natively.

Finally, we want to present the research questions for the thesis:

RQ1 What is the optimal radius to cover the surface of a given scene?

RQ2 What impact do obstacles have on coverage rates?

RQ3 What guidelines can be created to find a good radius size for any scene?

Chapter 3

Methods

A program has been written in ThingML and Java, and is explained in detail in Chapter 4.

3.1 Scenes

We would like to test the robot in different situations, and various scenes have been created for this reason. The scenes have been created with real rooms as their basis, and each scene is designed to give the robot different challenges. In terms of scale, centimetres have been chosen as the base unit for all experiments, which is reflected in the size of the robot, the obstacles, and the rooms. Five main scenes have been created as seen in Figure 3.1.

Scene 0 is an empty room containing no obstacles. Its purpose is to be used as reference for the other scenes, and to isolate the effect the radius has on coverage.

Scene 1 is a small office of 3 by 4 metres, containing a small set of obstacles. There is one table, one desk, and shelving along one wall. This scene poses the challenge of having a significant portion of the room easily blocked by the chair.

Scene 2 is a conference room with one table, and six chairs. In this case, the challenge is to see how the coverage is affected by a large amount of small obstacles in the middle of the room.

Scene 3 contains small obstacles along the walls. These obstacles are imagined to be three tables in an office or workshop environment.

Scene 4 contains three rows of obstacles in a manner you might see in a warehouse or storage facility.

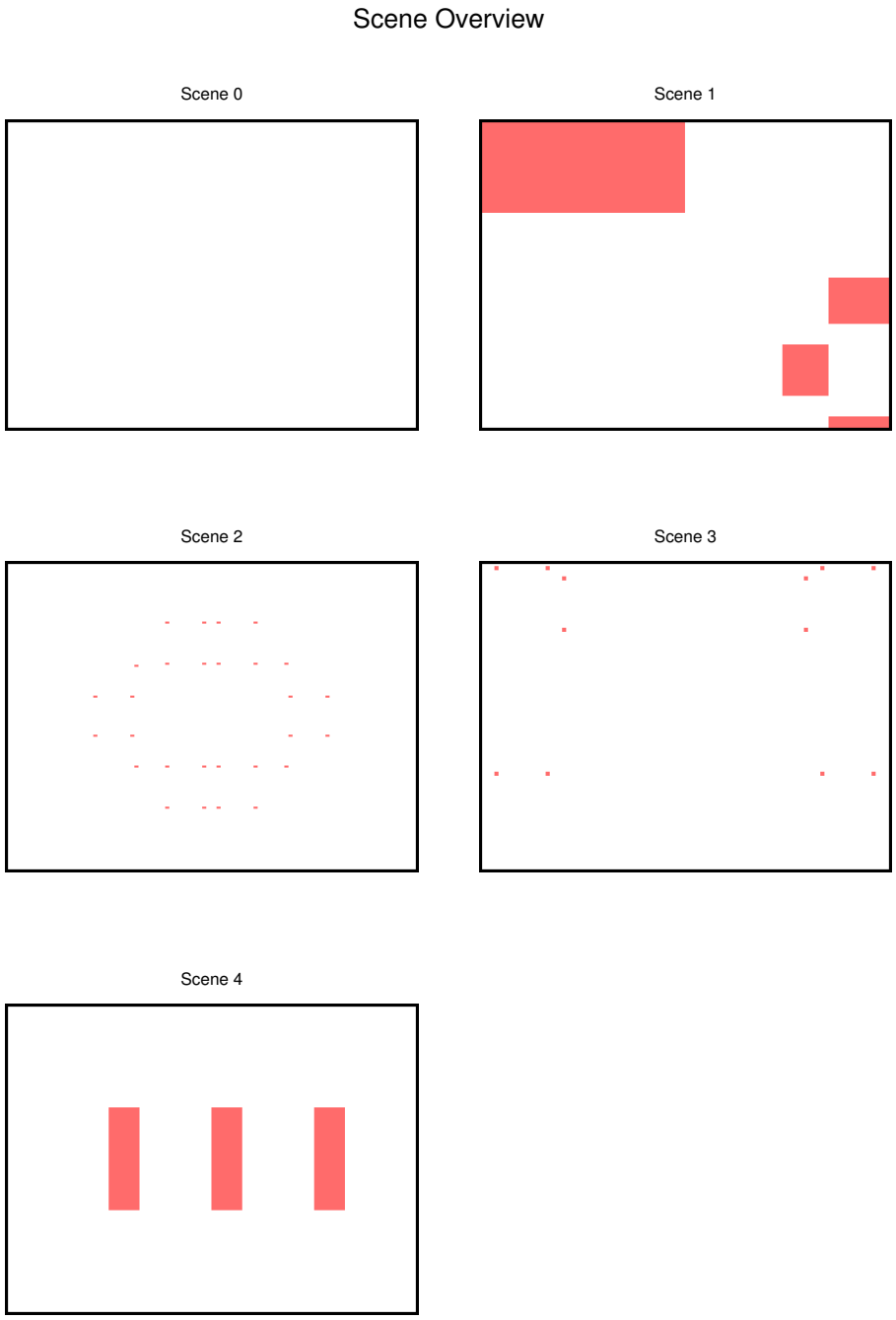

Figure 3.1: An overview of the various scenes used in experiments.

Chapter 4

Virtual laboratory

Robotic vacuum cleaners on the market today are often based on random search, though the higher end models are equipped with more advanced sensors which allow the robot to map the environment and use an elaborate algorithm. These higher-end models provide better user-satisfaction because of the intelligent algorithms based on maps, however they are much more costly. Affordability is one of the most important attributes when considering a purchase for the consumer market, and therefore we believe it is desirable to find improvements in the lower-end spectrum.

Simulation affords a flexibility beyond what is possible in the real world, and the experiments have been performed in a simulated environment. Some of the benefits of simulations are that specific parameters can be isolated, and experiments can be executed much more rapidly than they could in the real world.

4.1 The program

The robot's software has primarily been written in ThingML (described in more detail in Section 2.1), which is a modeling language based on state machines. This language gives us the flexibility of programming for multiple platforms. ThingML can be used in conjunction with other programming languages—in this case Java—which gives the programmer the ability to achieve results which are not possible in pure ThingML, such as the graphical elements of the simulation.

ThingML can use Java in two ways. It can embed Java code with kickdown directly in the code, or it can import a library. Both methods have been used prolifically, where kickdown is used for smaller tasks and to make use of the libraries, and the library is used to do the heavy lifting of Java coding. This is primarily the visualisation aspects of the program, but some more lengthy snippets have also been delegated to a library.

4.2 Architecture

The software model is split in three parts: the platform-independent model (PIM), the platform-specific model (PSM), and the Simulation Environment. How these parts are connected is displayed in ThingML configuration in Figure 4.1.

The PIM describes the logic of the robot. This is where the algorithms for movement and operation reside. An interface with the PSM provides the specific features it can utilise.

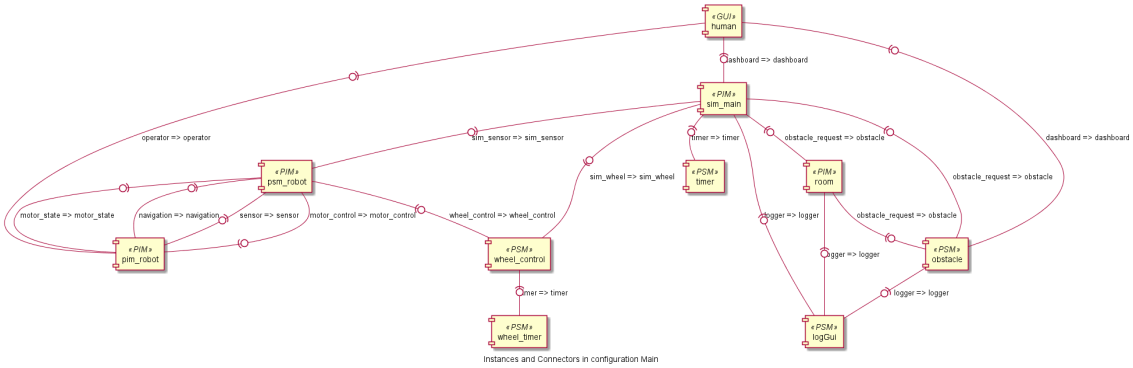

Figure 4.1: The ThingML architecture contains Things connected in this configuration.

The PSM controls the hardware directly, and provides an interface for the PIM. The wheel control system is the main actor in the Simulation Environment and is where the steps originate. When the robot is moving, the wheel control system sends messages to the Simulation Environment that the wheels are active, and the Simulation Environment in turn updates the location of the robot, checks for collisions, and gives sensory feedback.

The Simulation Environment describes the world outside the robot's logic. It is an additional layer which could be considered part of the PSM, but it has been separated for the sake of clarity. The PSM describes what the robot can physically do, and the simulation environment is more about the environment, physics, and a simulated map of the world and its contents. This is where, like in the real world, sensory input originates. When the robot bumps into something, a message is sent to the PSM to be processed. The philosophy is that when the robot acts upon the world, the Simulation Environment sends messages back only when a sensor has been triggered. Similarly, if the world acted upon the robot (not programmed), the Simulation Environment would send messages without polling from the PSM. Polling sensors would originate in the PSM, be sent to the Simulation Environment, and returne a message containing the update to the PSM, but none have been implemented.

There is also a graphical element which merely displays what happens in the Simulation Environment and has no effect on the robot or the world. A more elaborate description can be found in Section 4.2.4

4.2.1 Robot

The PIM and the PSM are intended to be deployed on the robot, but the Simulation Environment is intended to replace the real world. Effectively the robot's physical properties are located in the Simulation Environment, meaning the physics—such as location or velocity—exist in the Simulation Environment, but hardware controls exist in the PSM. It's beneficial for the robot to be programmed this way because the logic is completely separated from the rest of the code in the PIM, and the hardware control mechanisms and other interfaces can be placed in the PSM where changes can be made per platform (simulation, different physical robot versions, etc).

The robot's properties in the Simulation Environment are its location and its velocity. Location is updated in steps, and the robot moves at a constant velocity that can easily

be adjusted, but is set to be the same for all the experiments.

The focus of the simulation is the centre point of the robot. It is used to give the robot a physical location, for collision detection, and for visualisation purposes. The robot is circular in shape as seen from above, which means a single point can be used for collision detection. Every time the robot makes a step, a check is made to see if the centre point is inside any obstacles. If it is, then the intersection of where the robot would have collided is found, and the robot is only placed at the edge of the obstacle. To prevent the rest of the robot from overlapping the actual obstacles, an additional obstacle is generated to reflect the robot's physical size. This generated obstacle expands the original polygon

In this simulation, the robot's location is updated in steps of 75 milliseconds, and it moves at a constant speed of 7.5 centimeters per second. Every time a step is made, a collision detection check is made against the room and every obstacle. The collision detection procedure works as follows: A new location is calculated by adding vector $\vec{v}$ to the robot's current location (point C). A ray-casting algorithm is then used to check if the robot is on the outside of the room or on the inside of any obstacles. If it is, then a scalar value k between 0 and 1 is calculated to scale v. Finally, the robot's current location is updated: $C' = C + k\vec{v}$

4.2.2 Scenes

A scene consists of a room and a set of obstacles. There must be one and only one room, however, there can be as many obstacles as one would like.

The room can have an arbitrary number of walls, and take any shape. A new obstacle is created from the two points of every wall which gives a visual outline of what areas the robot can reach, and they are used for collision detection.

Obstacles can have an arbitrary number of sides, but their shape should be convex. If concave obstacles are desirable, they should manually be split into multiple convex obstacles to make use of the radius expansion described in the next paragraph.

Each obstacle can be expanded based on the robot's radius. A new set of points is created by radially adding new points one radius distance away from the points of an obstacle's polygon.

4.2.3 User interface

The graphical user interface is split in two parts. The first part is a mock interface, which is a part of ThingML. There is a mock interface made to adjust settings/parameters on the fly for testing or experimental purposes (Figure 4.3). This is where you can move the robot to a new location, or adjust the size of the radius. Another mock interface (Figure 4.2) exists to give the robot commands, in place of what might be a remote control or buttons on a physical robot.

The second part is a visualisation of the simulated environment and is detailed in Section 4.2.4.

Some optional settings or features require the source code to be edited, or can be adjusted in a file containing globally available read-only properties.

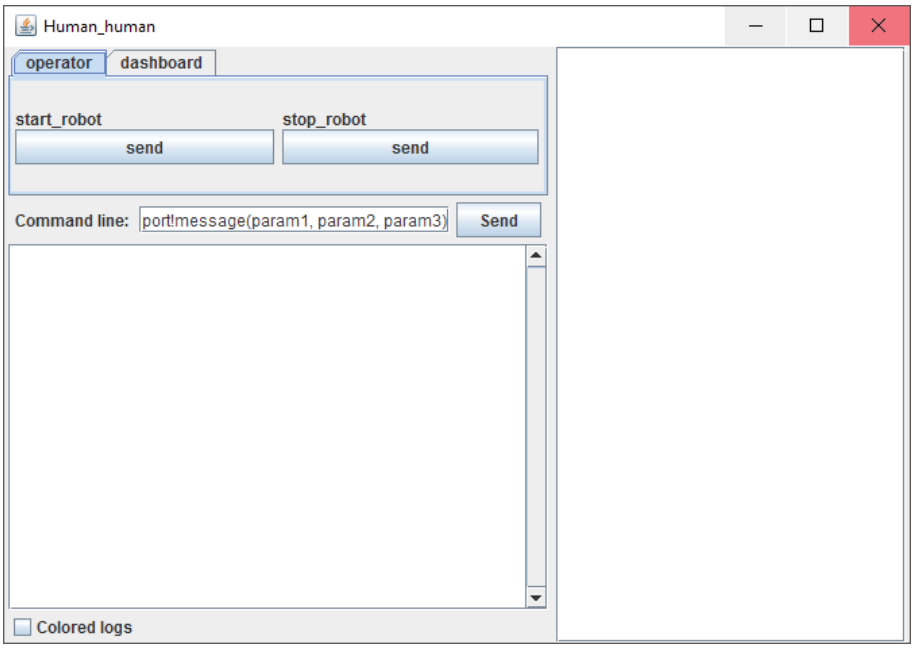

Figure 4.2: This ThingML mock interface gives the human operator controls that would be present on a physical robot.

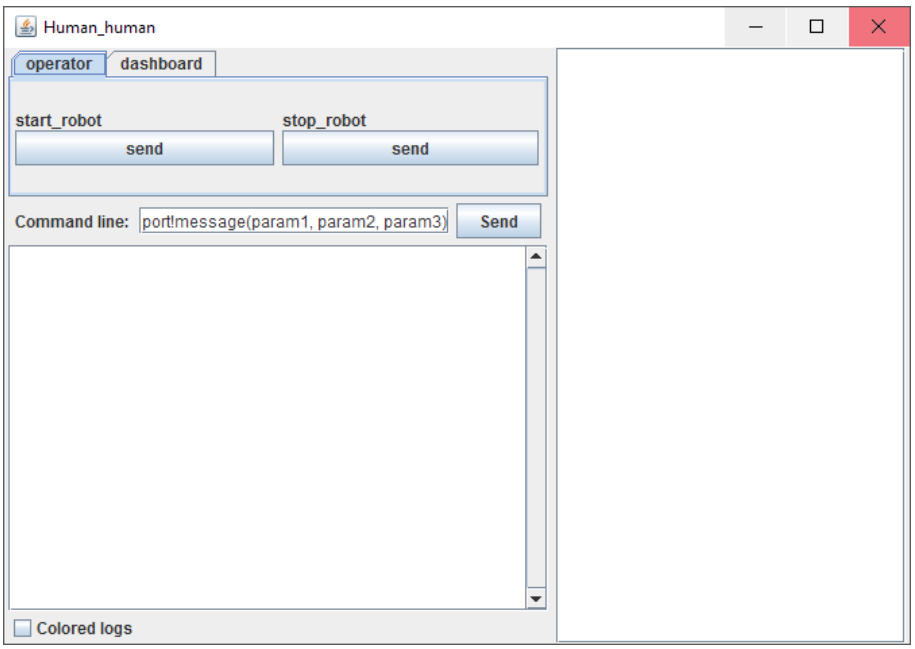

Figure 4.3: Some simulation options can be adjusted from the mock interface.

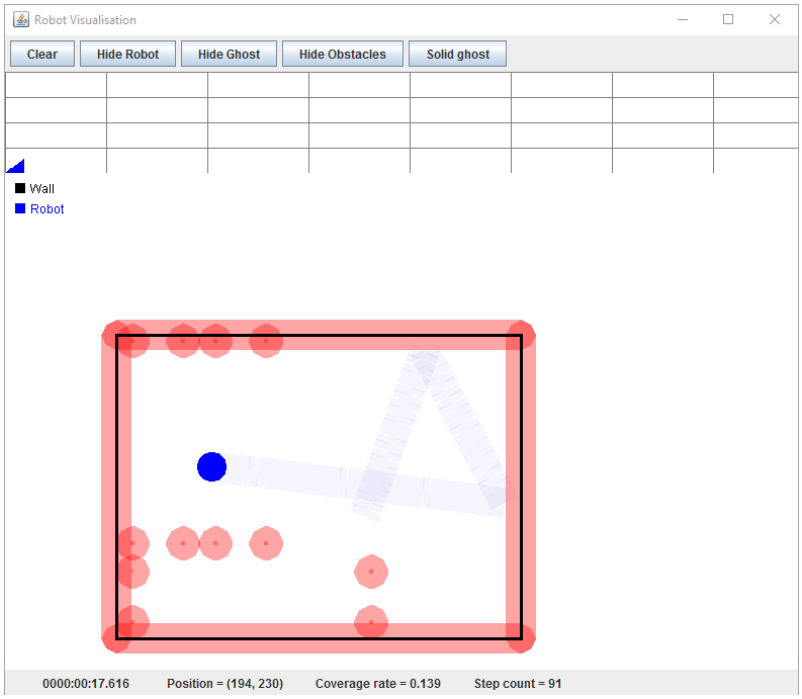

Figure 4.4: The Swing GUI contains a visualisation, display options, and various statistics.

4.2.4 Visualisation

The visualisation is written in Java, and Swing has been selected as the graphics package because ThingML already uses Swing to provide mock interfaces. There are four main aspects to the visualisation: the robot, the ghost, the obstacles, and the room. The robot is displayed as a blue circle with a one pixel white outline to contrast it aganst the ghost and the obstacles. Following the robot is a trail of where the robot has previously been, known as the ghost. The ghost has been given transparency to visually indicate to the user where paths have overlapped. Obstacles are shown in red, and are also transparent for similar reasons. Every obstacle generates an outer perimeter which is one radius greater in size, which the robot's centre point collides against. Effectively this means the robot cannot reach areas that are not connected by white space. Finally, the walls of the room are displayed in pure black.

As for the rest of the GUI, there is a button bar on top, a coverage profile underneath it, and there is a statistics bar at the bottom. The button bar gives the user various display options and the ability to clear the current run by deleting the ghost and resetting all statistics. Three buttons exist to hide or reveal different aspects of the simulation, and a final button can be activated if the user does not want a ghost with transparency. A coverage profile is generated in real-time under the button bar. It has horizontal lines at 25%, 50%, and 75% total coverage, and vertical lines for every 500 steps. The statistics bar contains some useful real-time information like a stopwatch, the position of the robot, the coverage rate, and the step count. Data and statistics are logged in a separate directory, where a new file is created—and given a timestamp in Unix time—each time the program is executed.

Screenshots of the GUI is found in Figure 4.4 and Figure 4.5.

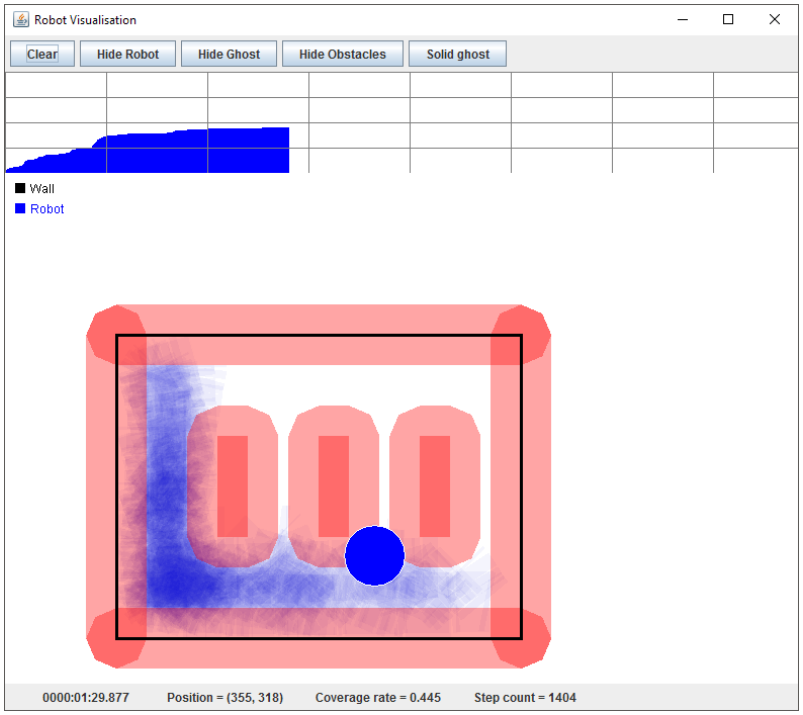

Figure 4.5: Notice the transparency where the ghost overlaps itself, and the real-time coverage profile present in the GUI.

Chapter 5

Results

Three experiments have been performed in the virtual laboratory. The first experiment explores the differences between radius sizes in the various scenes. The goal of the second experiment is to see how coverage rates are affected when obstacles are moved aside to make way for the robot. The third experiment explores what effects can be seen if the human operator moves the robot to areas that have not yet been covered. Unfortunately, the third experiment could not be completed in time to meet the thesis' deadline, therefore only a small part of it is included.

5.1 Experiment 1: Radius exploration

In the first radius experiment, the robot was allowed to run in the first scene for 5000 steps, or 375 metres, which was enough time to let the coverage rate converge in almost all cases—the exception being the smallest radius in scene 0. This procedure was repeated multiple times for each radius size, and the results are presented for each scene as a profile curve, and as a direct comparison of the coverage rate at specific times.

We can see that a larger radius produces a smaller total area covered over a long time. There are a few reasons for this. A larger robot will not reach all the way into the corners of the room, and the area under the desk has been blocked off. The coverage statistic has also only accounted for areas that have been covered up to the centre of the robot, which means that some spots have been missed in areas where the robot has not run parallel to the wall.

There is some minor variation of ± a percentage point of coverage for each radius size, but this is to be expected due to the random nature of the coverage algorithm.

5.1.1 Experiment summary

The empty rooms in scene 0–1 and scene 0–2 showed that a large radius is an advantage if speed is of concern. However, their early advantage becomes irrelevant as time goes on when smaller sizes are able to capitalise on the fact they have better reach. Similar results were found in the other scenes. Comparisons at specific intervals demonstrated that the optimal size changes over time, but only towards smaller sizes. There were no instances where the optimal radius moved to a larger size, but the optimal radius remained 15 cm for the duration of scene 2–1.

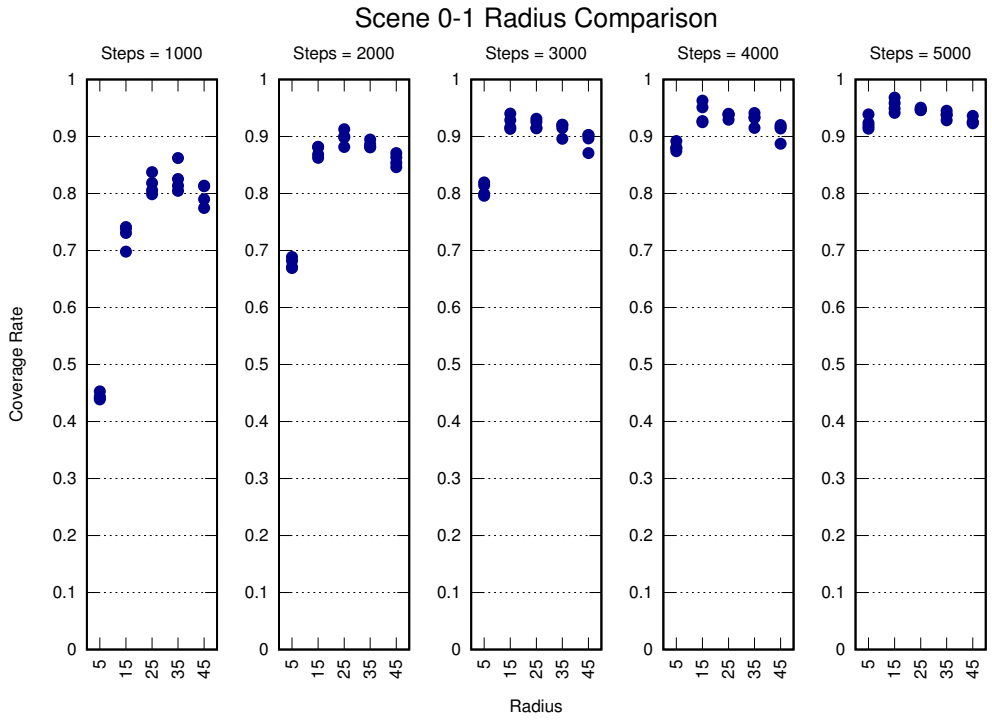

Figure 5.1: The coverage rate comparison for different radiuses at set intervals highlight the variance for a given radius.

5.1.2 Scene 0–1: No obstacles

Figure 5.2 shows the profile curve for scene 0–1, and a direct radius comparison is seen in Figure 5.1. It shows that when no obstacles were present, a larger radius was beneficial, but it is perhaps surprising that the 35 cm robot performs better than the larger 45 cm robot. The surface was covered much faster with a large radius, however the size and shape prevents the robot from reaching the corners of the room. This allows the smaller radiuses to overtake total coverage after an extended period. It is also worth noting that the smallest radius is too small to efficiently cover the entire surface in the given timeframe. The profile curve also shows that the initial linear curve

The radius comparison figure also shows that the size that performs best slowly drifts to a smaller size. The 35 cm radius has the greatest coverage rate at first, but the 15 cm radius outperforms it after 5000 steps.

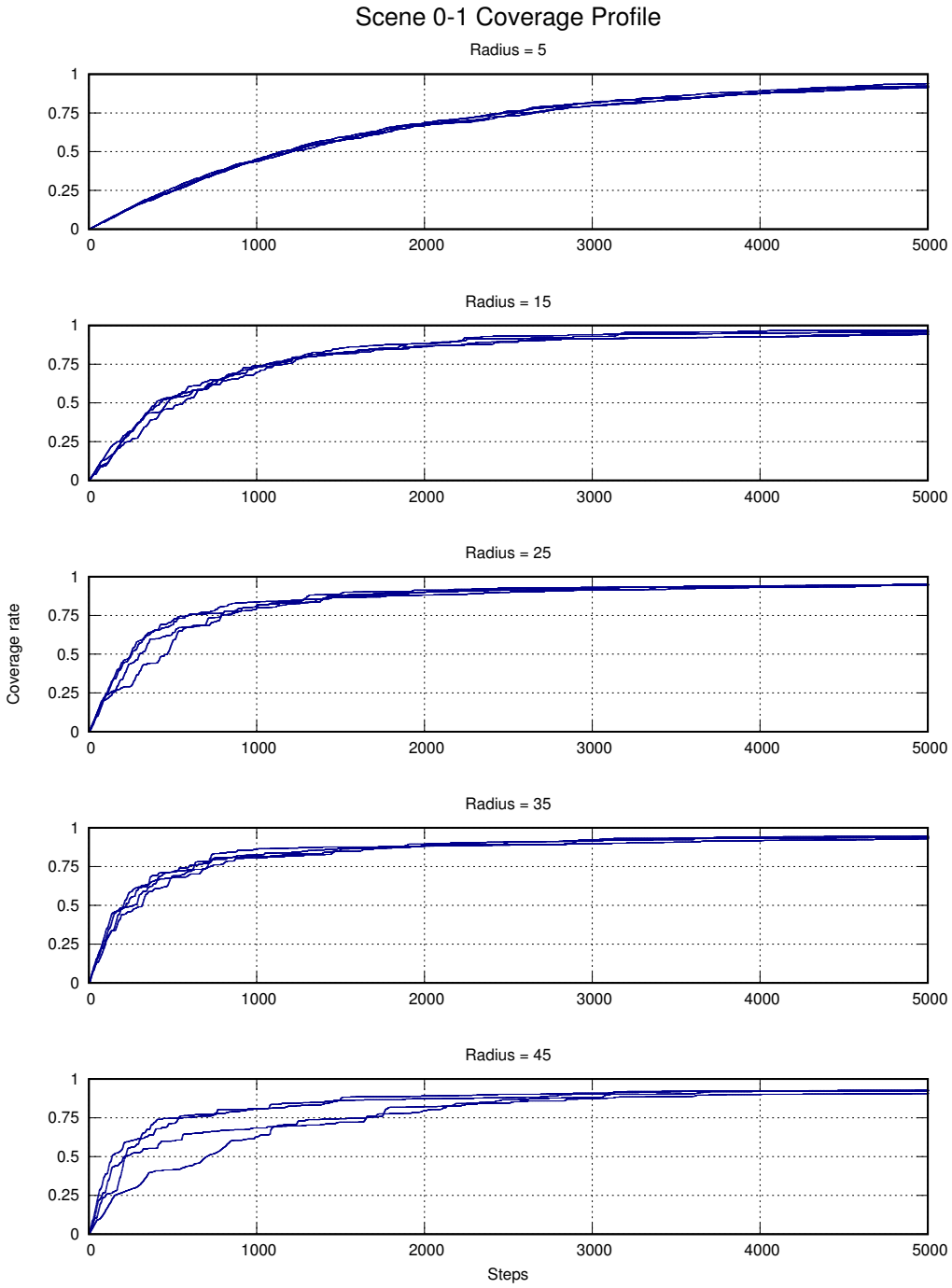

Figure 5.2: The profile curve shows how the coverage rate evolves as the robot moves through the scene. Each radius is given its own plot, and the plots are stacked on top of each other for ease of comparison.

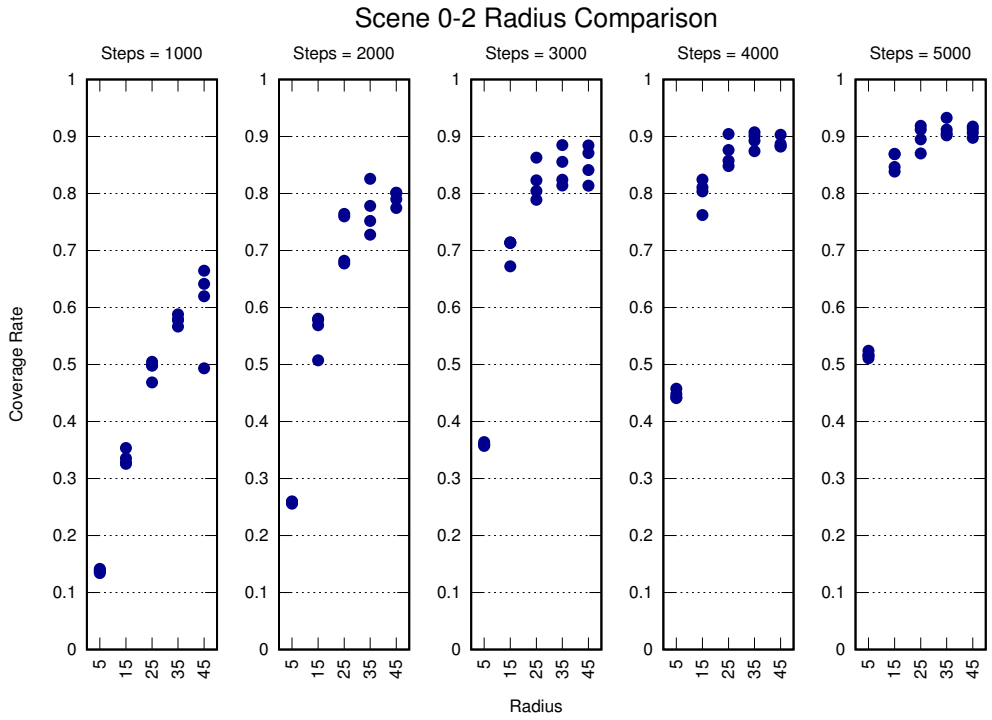

Figure 5.3: 5000 steps is not quite enough time for the coverage rates to converge in the large room in scene 0–2.

5.1.3 Scene 0–2

In scene 0–1 we saw how the robot performs in a room free of obstacles, and scene 0–2 features the same scene, but it is scaled to be twice as large at 8 metres by 6 metres.

Based on the previous results, we should expect the larger radius to perform better. The optimum, however, is expected to change for two reasons: there is more surface to be covered, and unreachable areas have less impact. This is indeed what we observe, and the larger room shows a more pronounced effect which is clearly visible in Figure 5.3. At the start, a larger radius suggest vast improvements, and it is not until the robot has moved 3000 steps that a radius of 35 cm and 45 cm perform equally well, which it continues to do for another 1000 steps. As we expected, the size of the room is a significant factor in how well the robot covers the empty room.

The profile curve in Figure 5.4 shows the increased early variance with a radius of 45 cm that we saw in the previous scene. It is also clear that the initial linear part of the curve gets significantly steeper as the size of the robot increases.

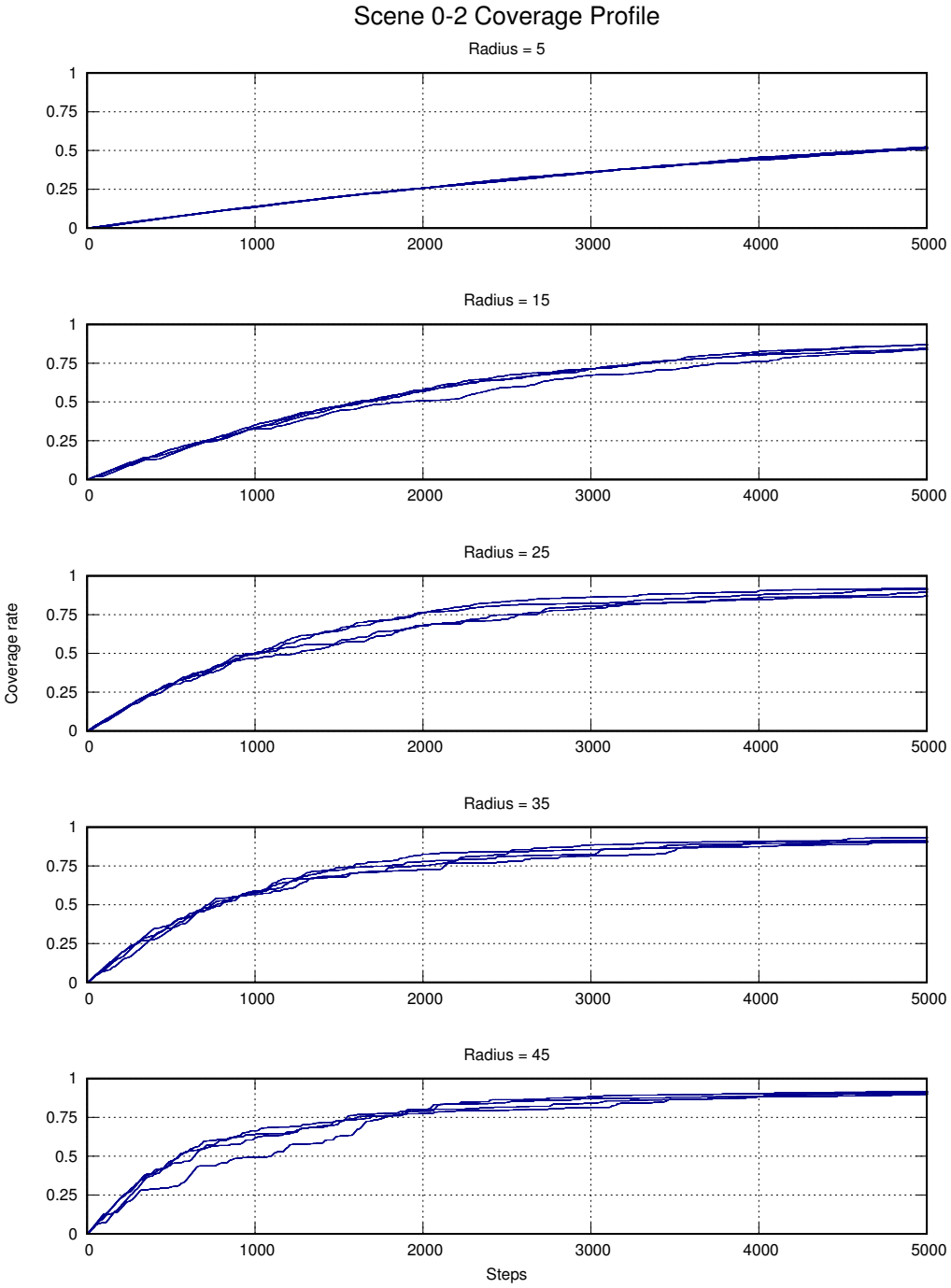

Figure 5.4: The profile curve for scene 0–2 is less steep than scene 0–1 due to its larger size.

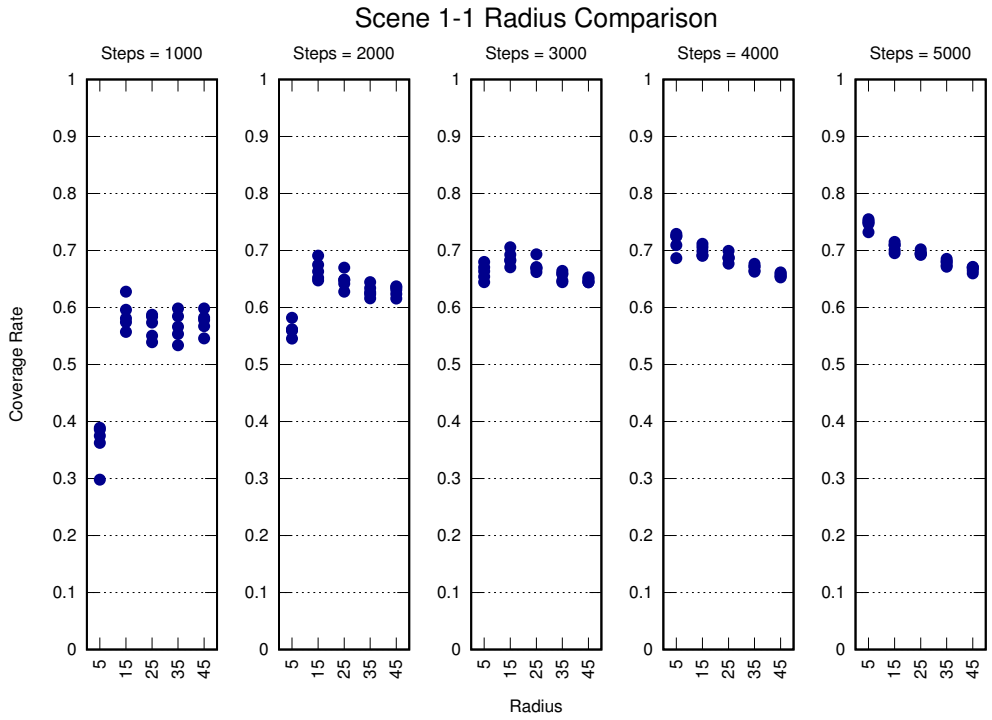

Figure 5.5: Similar performance across different radius sizes is seen in the radius comparison for scene 1–1.

5.1.4 Scene 1–1: Small office

When the robot has only run for a short distance of 1000 steps, or 75 metres, there is more variation in coverage rates within each radius size as seen in the coverage profile in Figure 5.6 and the radius comparison in Figure 5.5. In the case of a smaller radius, this can be because the robot has become stuck under the desk for a short period. However, in most cases the total coverage is not drastically different between each run.

The relatively low coverage rate numbers can be explained by the total area covered by obstacles which are quite large in this scene.

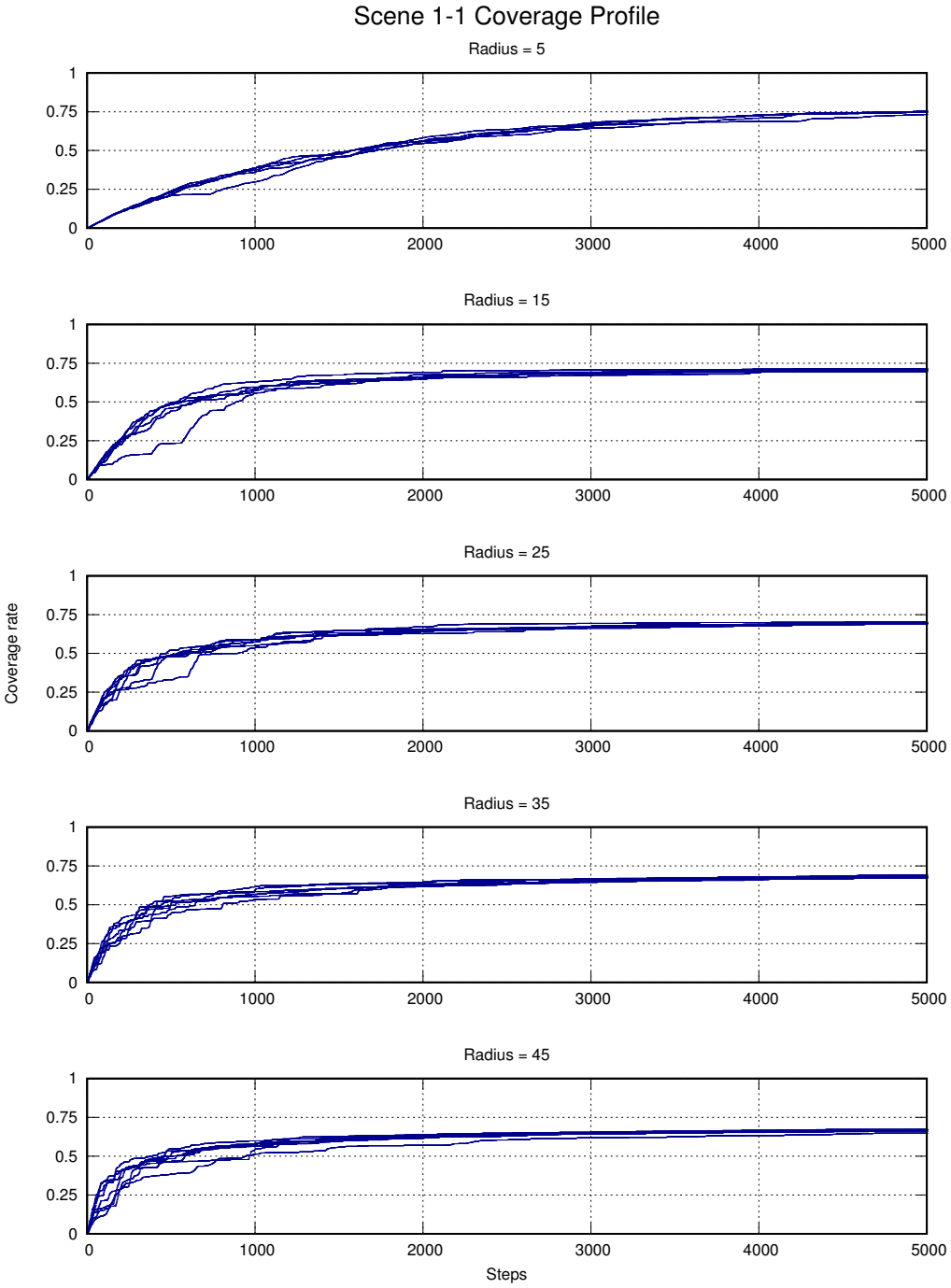

Figure 5.6: Large obstacles and an inaccessible area is reflected in the profile curve for scene 1–1.

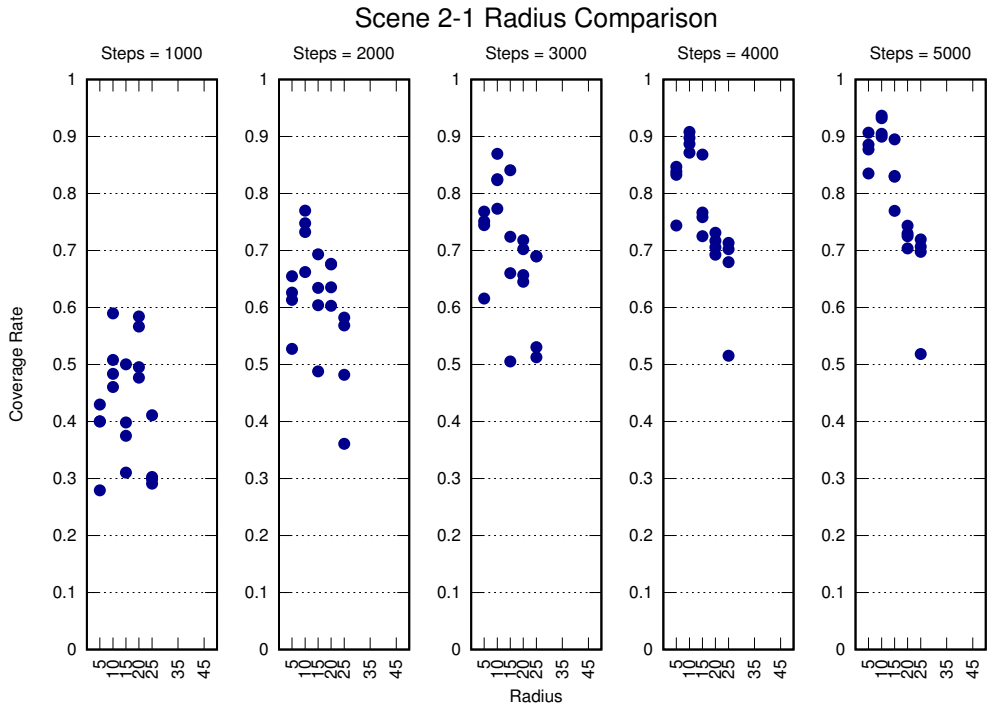

Figure 5.7: The 15 cm radius is a clear winner in this radius comparison.

5.1.5 Scene 2–1: Small conference room

The first thing that stands out in this scene is that when the radius exceeds 30 cm, most of the area becomes inaccessible to the robot. Results from this sizes larger than 30 cm are therefore uninteresting, and have been omitted in favour of two smaller radiuses.

The coverage profile (Figure 5.8 and Figure 5.9) of this scene looks quite different from the previous scenes where the curve is mostly smooth. Obstacles in this scene are situated such that the robot bounces back and forth in a particular region, or back to a previous region, until it enters a new place it has not yet been, resulting in very distinct steps in the profile curve that negatively affect the coverage rate.

Variance caused by these steps are also visible in the radius comparison (Figure 5.7). The variance is the lowest after 5000 steps because the coverage rates have had time to converge.

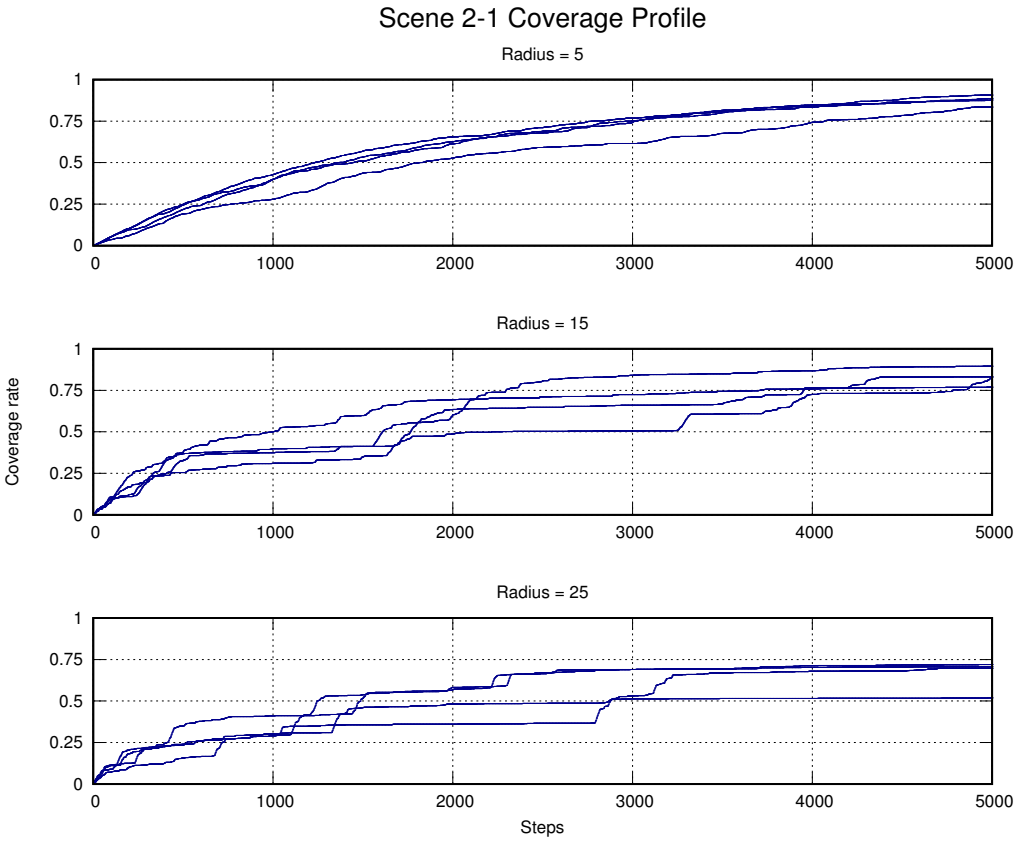

Figure 5.8: The robot had great difficulty finding its way around scene 2–1 as evidenced by the large jumps in coverage rates.

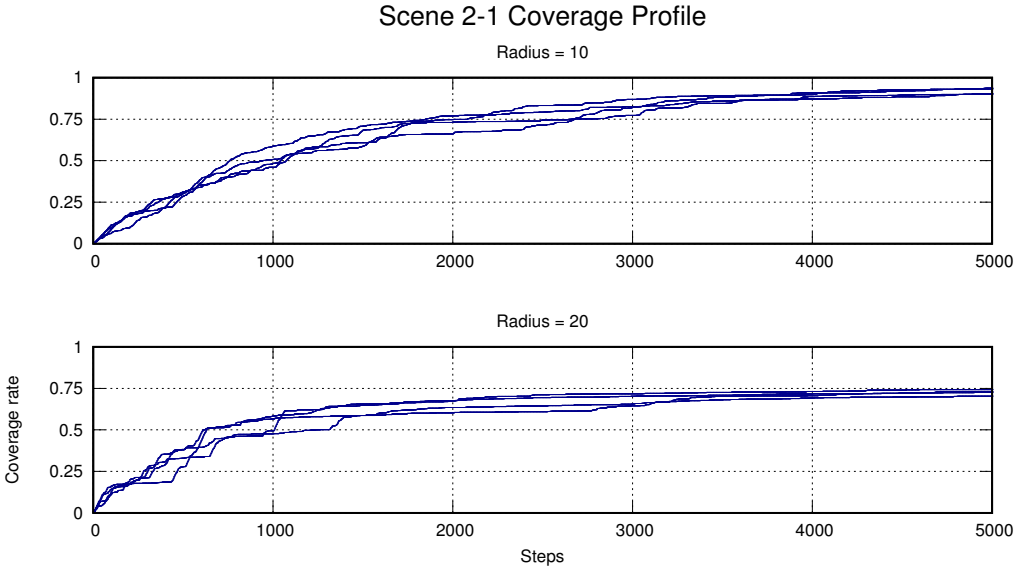

Figure 5.9: Two additional radius sizes are included because the two largest radius sizes were stuck in place.

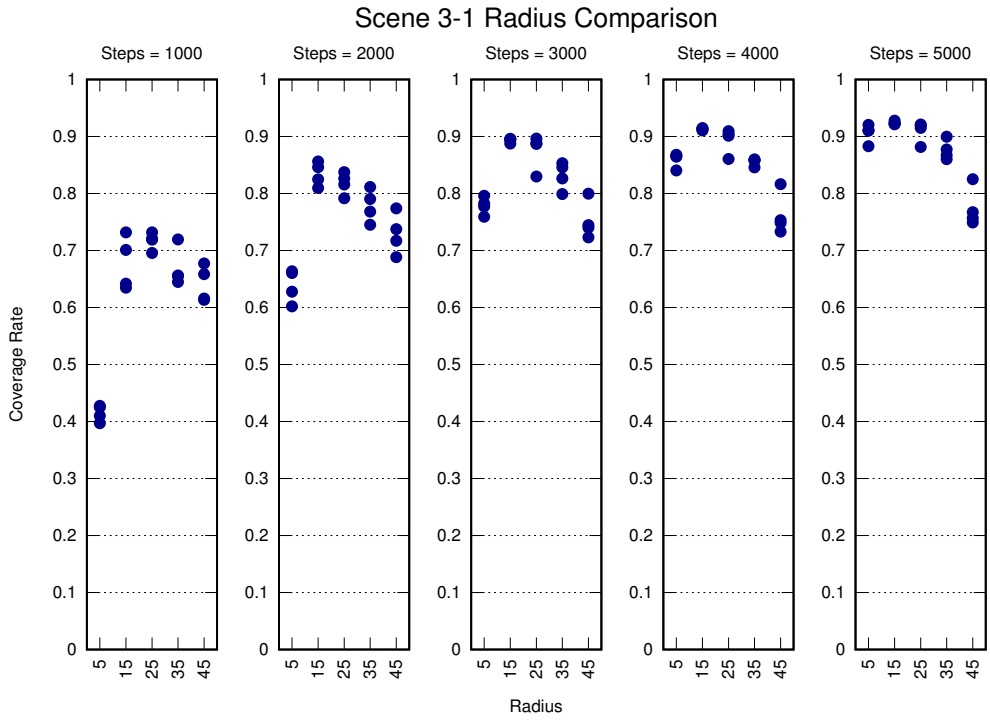

Figure 5.10: A performance comparison at set points in time.

5.1.6 Scene 3–1: Three tables

Scene 3–1 has three tables placed along the walls. Figure 5.11 shows a very similar profile curve to the one in scene 1–1, but less surface area is covered or inaccessible in this scene than in scene 1–1, so the maximum coverage rate should be greater than scene 1–1, and this is indeed what we see.

The smallest radius does far worse at covering the area quickly, but it covers more as time passes. Comparatively, the bigger radiuses perform slightly worse relative to the smaller ones. This can be seen in the radius comparison in Figure 5.10. Whereas the coverage difference between radius sizes in scene 1–1 were minor at 0–10 percentage points at the same step count, they go as high as 20 percentage points here in scene 3–1.

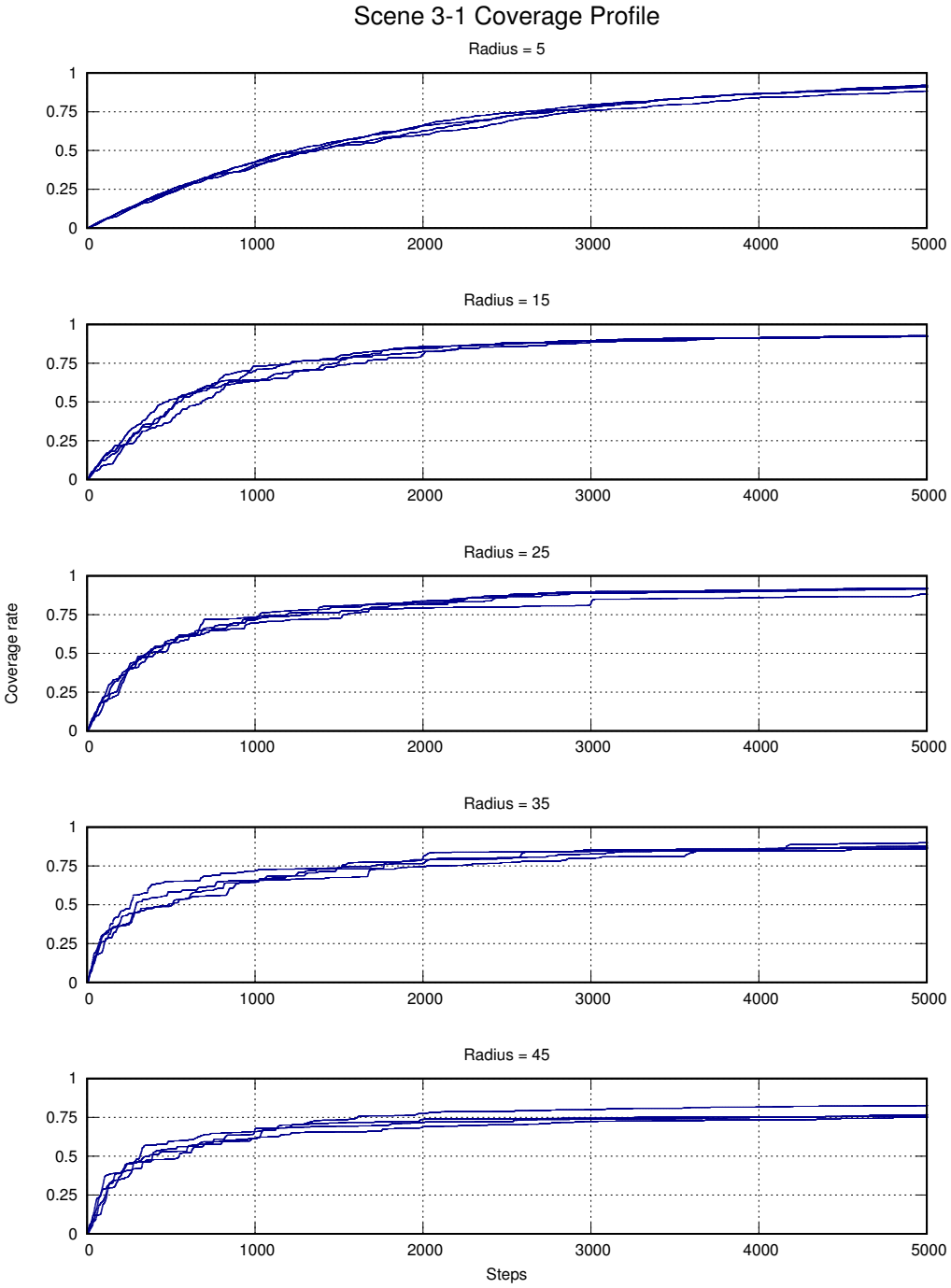

Figure 5.11: A profile of total area covered over time.

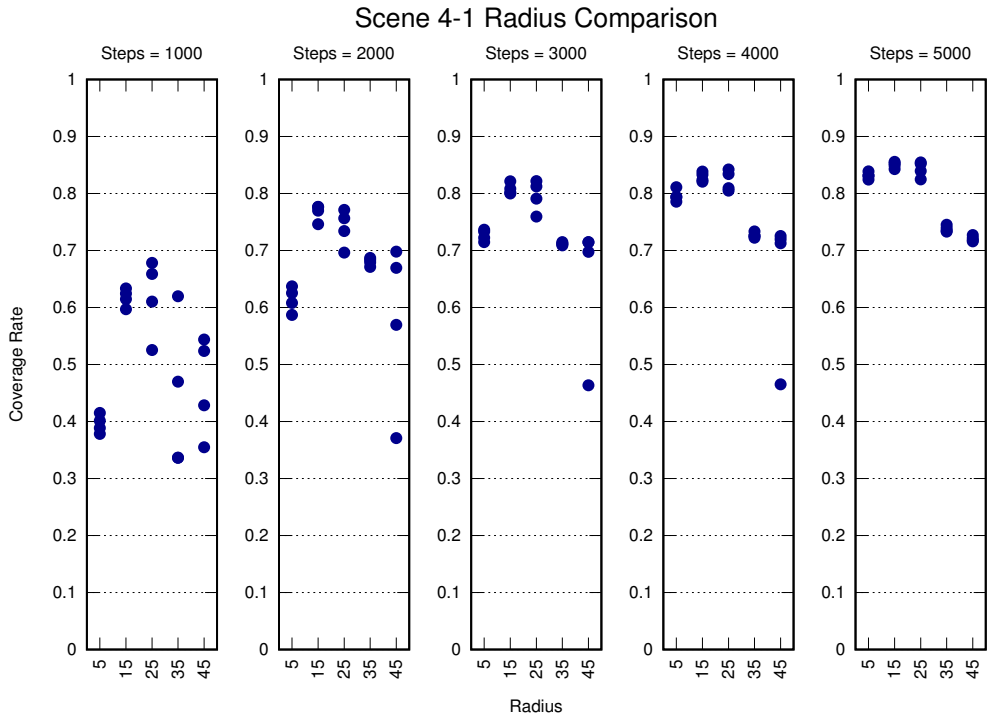

Figure 5.12: The variance in scene 4–1 flattens out over time as seen in the close-up radius comparison.

5.1.7 Scene 4–1: Storage shelves

There obstacles in scene 4–1 are three shelves in the middle of the room, which are expected to give the robot some trouble, and cause jumps in coverage rates much like in scene 2–1. The profile curve in Figure 5.13 shows exactly this. When the radius is sufficiently large at 25 cm and above, distinct jumps are seen. They cause variance and huge spikes in coverage rates. The variance does not last, though. Jumps are no longer visible after 2000 steps, except in the case of a 45 cm radius. This radius needed 4500 steps to ensure the coverage rate had converged.

The radius comparison in Figure 5.12 highlight the fact that radius sizes 35 cm and 45 cm cannot achieve coverage rate as high as the smaller radiuses can. Shelves in the centre of the room prevent the robot from accessing all areas.

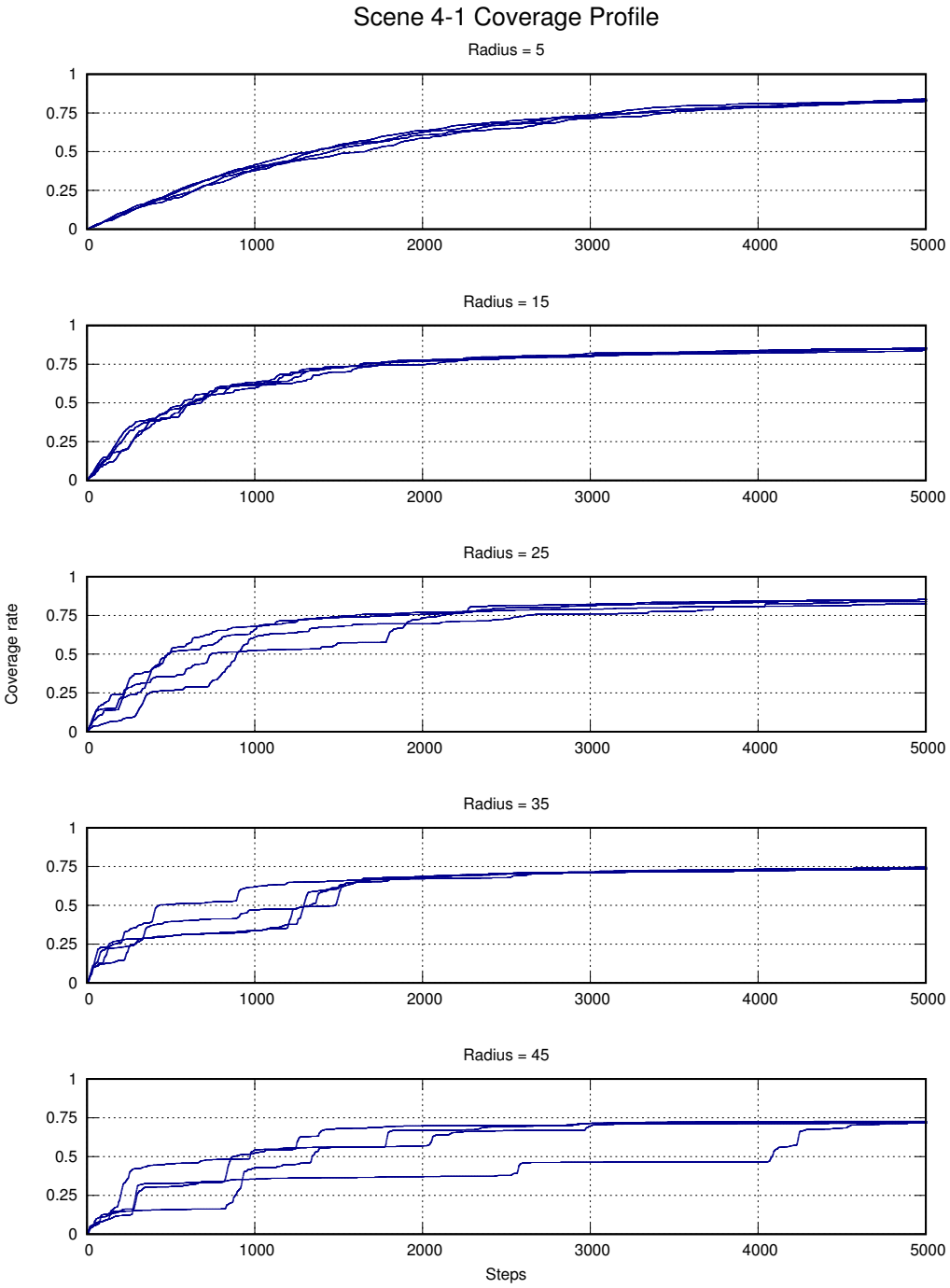

Figure 5.13: The profile curve in scene 4–1 shows jumps in coverage rates at radiuses 25 cm and above.

5.2 Experiment 2: Moving the obstacles

In this experiment, we take a look at how the scenes are affected if obstacles are moved out of the way to make room for the robot. The coverage profiles in scenes 2 and 4 showed very distinct jumps in coverage rate when the robot found its way to a new area. By moving the obstacles, we hope to see that we can reduce these jumps and achieve a better coverage rate given the same time.

Previously 5000 steps was used as the cut-off point, but we saw a major fall-off in coverage rate after 2000 steps. A new cut-off point has therefore been chosen at 2000 steps.

5.2.1 Experiment summary

The experiment has shown the importance of obstacle placement. We saw that obstacles can be placed strategically to reduce variance or increase coverage rates. Obstacles placed along walls have a smaller negative effect on coverage than obstacles in the middle of the room, and large robots are sensitive to areas that can be difficult to reach.

Scene 1–2 shows greater variance than before, but the actual coverage rate values are about the same. Scattered obstacles have made room for bigger robots in scene 2–2. Great variance was observed here too, but it fades off before 2000 steps and gives a greater performance for a radius of 25 cm or below. Scene 3–2 showed negligible differences, but there was a slight drop in performance for the largest radius caused by a hard-to-reach area.

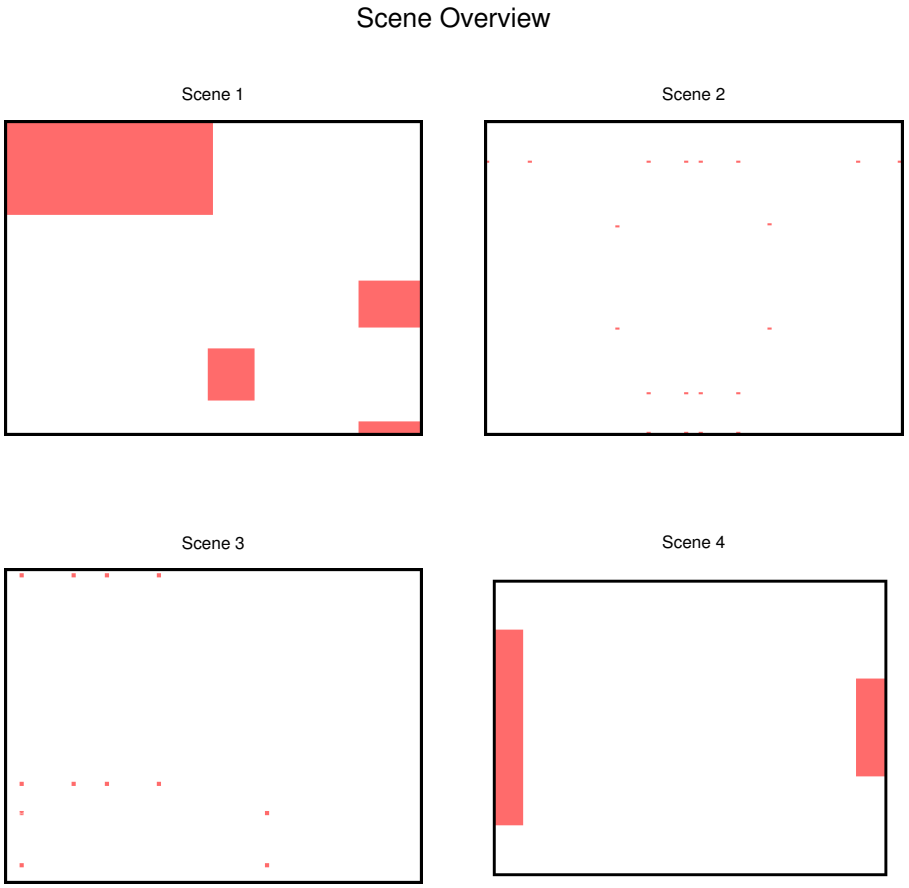

Figure 5.14: Scenes in experiment 2 have moved the obstacles in an attempt to improve coverage rates.

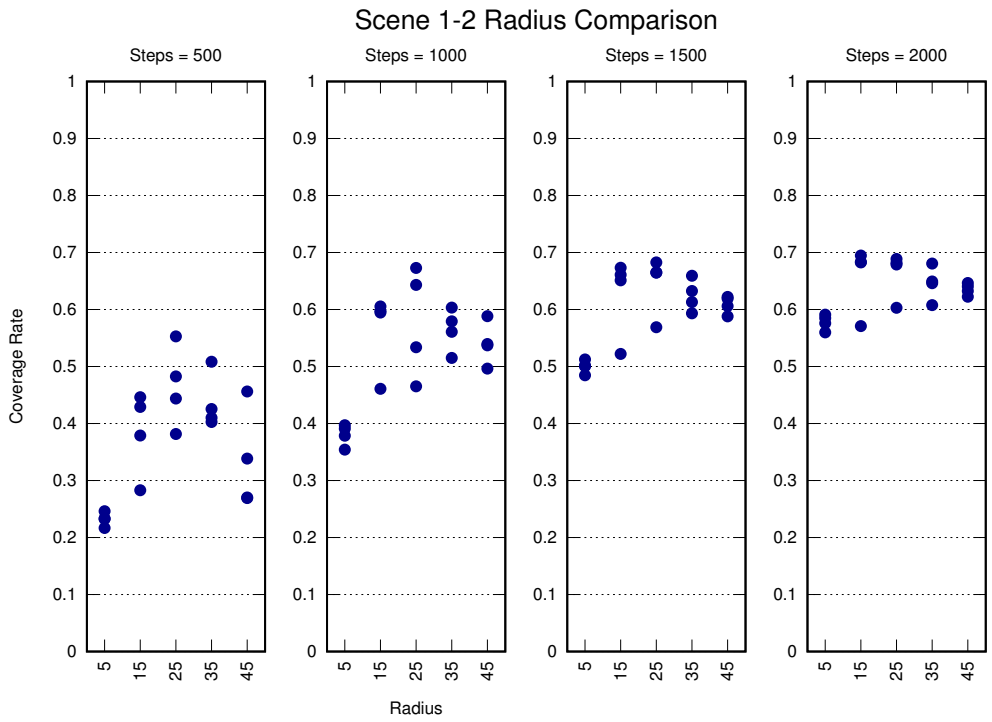

Figure 5.15: Major variance can be seen in the close-up of coverage rates for scene 1–2. The maximum step count is now 2000, down from the previous 5000.

5.2.2 Scene 1–2: Chair pulled back

The office in Scene 1–1 contained a desk and chair that blocked the area under the desk when the radius became large enough. To make space for the robot, the chair has been pulled back. The hypothesis is that this access will have a noticeable impact on the coverage rate.

What we actually observe in the coverage profile in Figure 5.16 is that there is more variance in the first 1000 steps than before. In experiment 1, the maximum and minimum coverage rates for a given radius was well within 10 percentage points of each other, but the difference is now up to 20 percentage points as the radius comparison in Figure 5.15 shows.

Aside from the increased variance, the coverage rate steps remains largely the same. The 5 cm radius looks almost identical at 1000 and 2000 steps, and the larger radiuses follow the same pattern but is more scattered.

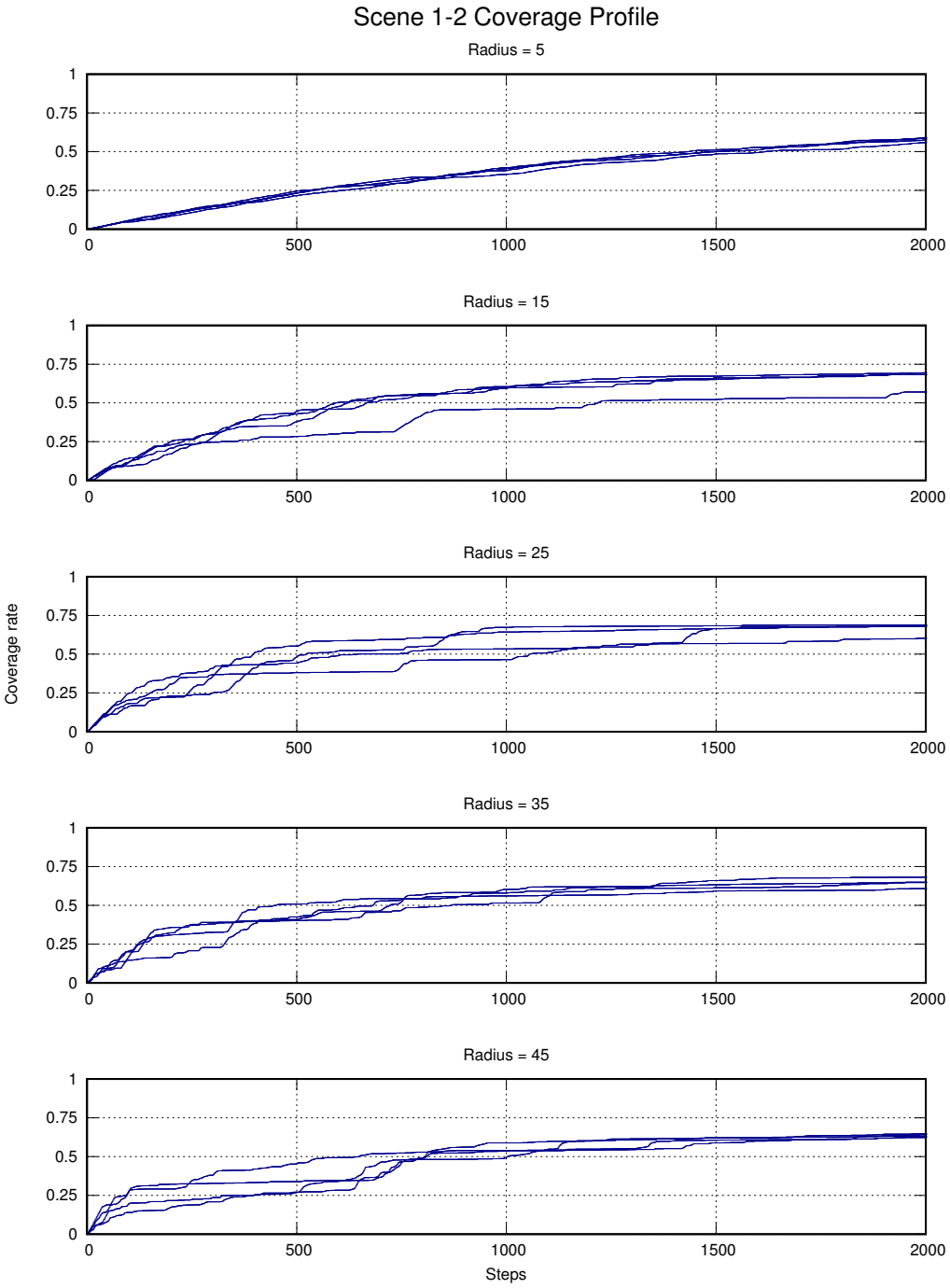

Figure 5.16: A greater variance than the previous experiments can be seen in the profile curve for scene 1–2.

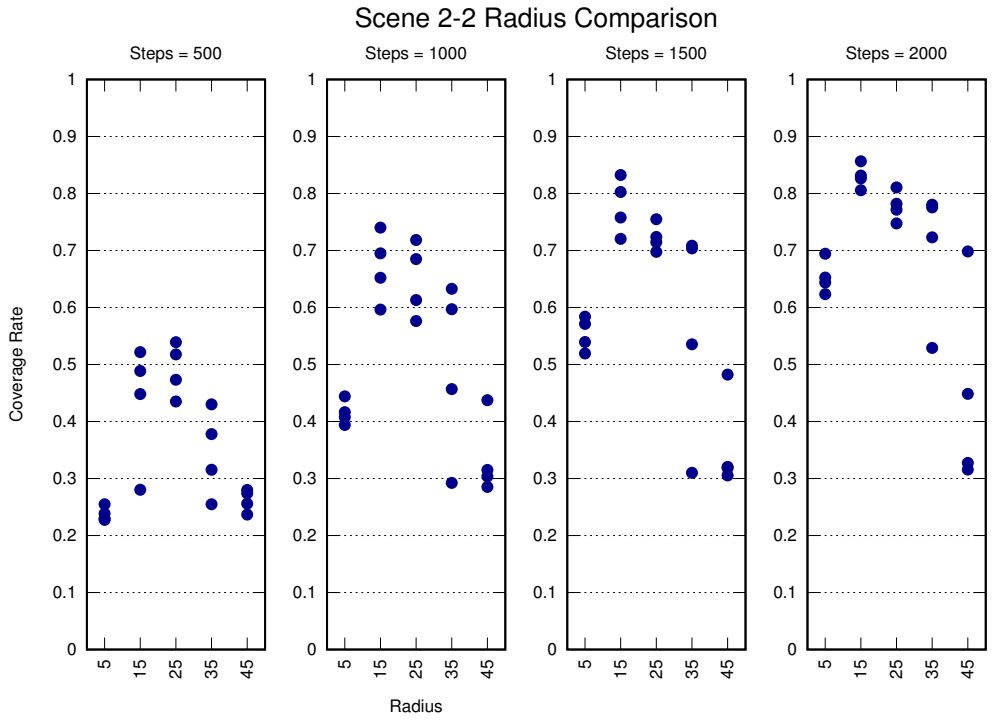

Figure 5.17: The coverage rates in the close-up for scene 2–2 show improvements in coverage rates as compared to scene 2–1.

5.2.3 Scene 2–2: Chairs moved to corners and sides

Scene 2–1 contained six chairs and a table which gave the robot a lot of trouble moving around, completely blocking the for radiuses greater than 30 cm. In a real-world situation, it would be sensible to move the chairs to make it easier to clean. This can be done by putting the chairs on top of the table, stacking them in a corner, or placing them strategically around the room. We are interested in seeing how the number of obstacles present in the scene affects coverage rates, and have used the latter method. Two chairs have been placed in corners, and the other four chairs have been placed along the walls. This has made more of the room accessible for larger robot radiuses, so the two biggest radius sizes are included this time in experiment 2.

The profile curve in Figure 5.18 shows a great variance shows a great variance just like in scene 1–2, but improvements have been made this time. A strategic placement of obstacles has made the robot less likely to meander in the same area for extended periods, effectively

When we look at Figure 5.17, we can see very clear improvements at 1000 steps. In experiment 1, the radiuses 15 cm and 25 cm varied from 0.3 to 0.6, but the coverage rates this time vary from approximately 0.6 to 0.75.

The two largest radiuses have a lot of trouble accessing the whole surface, but they are able to achieve a coverage rate of 0.75.

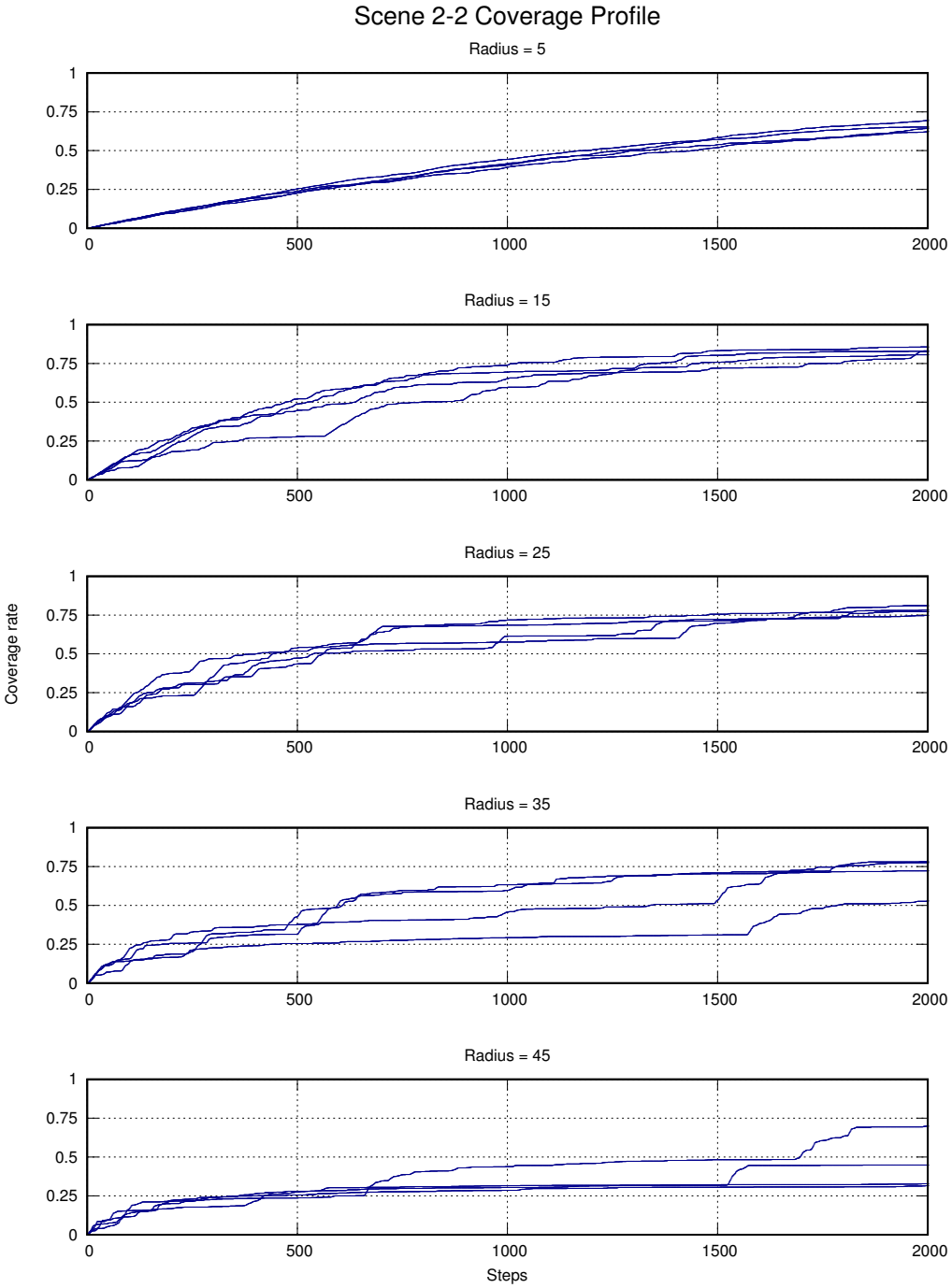

Figure 5.18: Fewer jumps in coverage rates are observed in the coverage profile for scene 2–2.

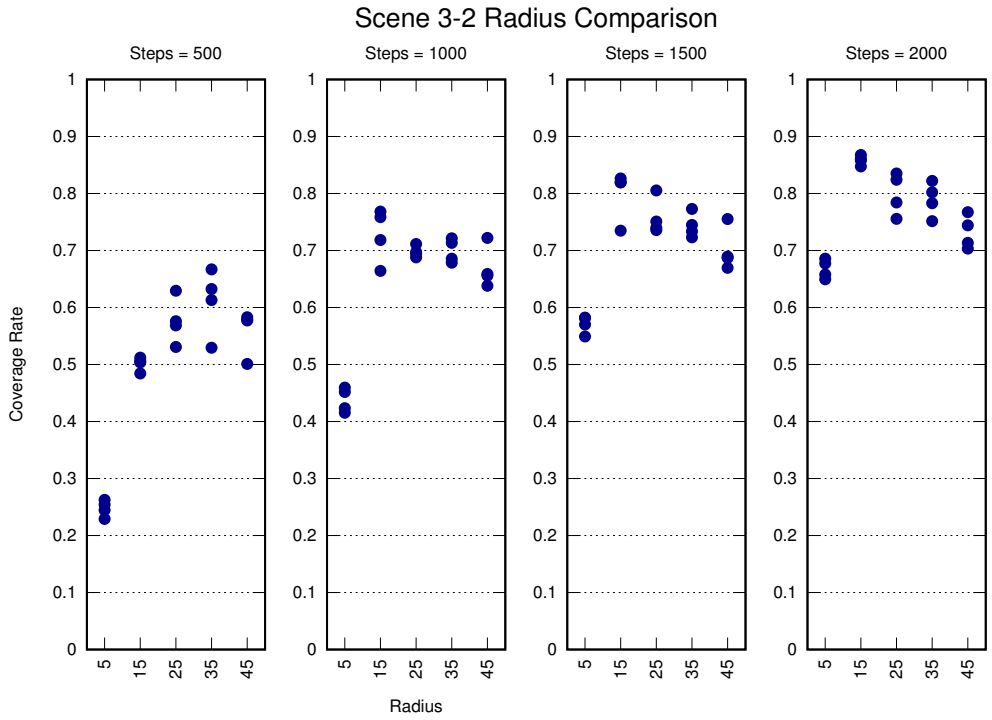

Figure 5.19: This radius comparison for scene 3–2 displays worse performance for the largest radius.

5.2.4 Scene 3–2: Grouped tables

The obstacles in scene 3–2 have been grouped together in a corner. These obstacles were aligned with the walls before they were moved.

This new setup shows very similar results as we had before when we look at the coverage profile in Figure 5.20. The radius comparison in Figure 5.19 reveals that the largest radius performs worse in the new setup, but the other sizes have nearly identical results. A section in the bottom left corner became difficult to access for the largest radius, resulting in this drop in performance.

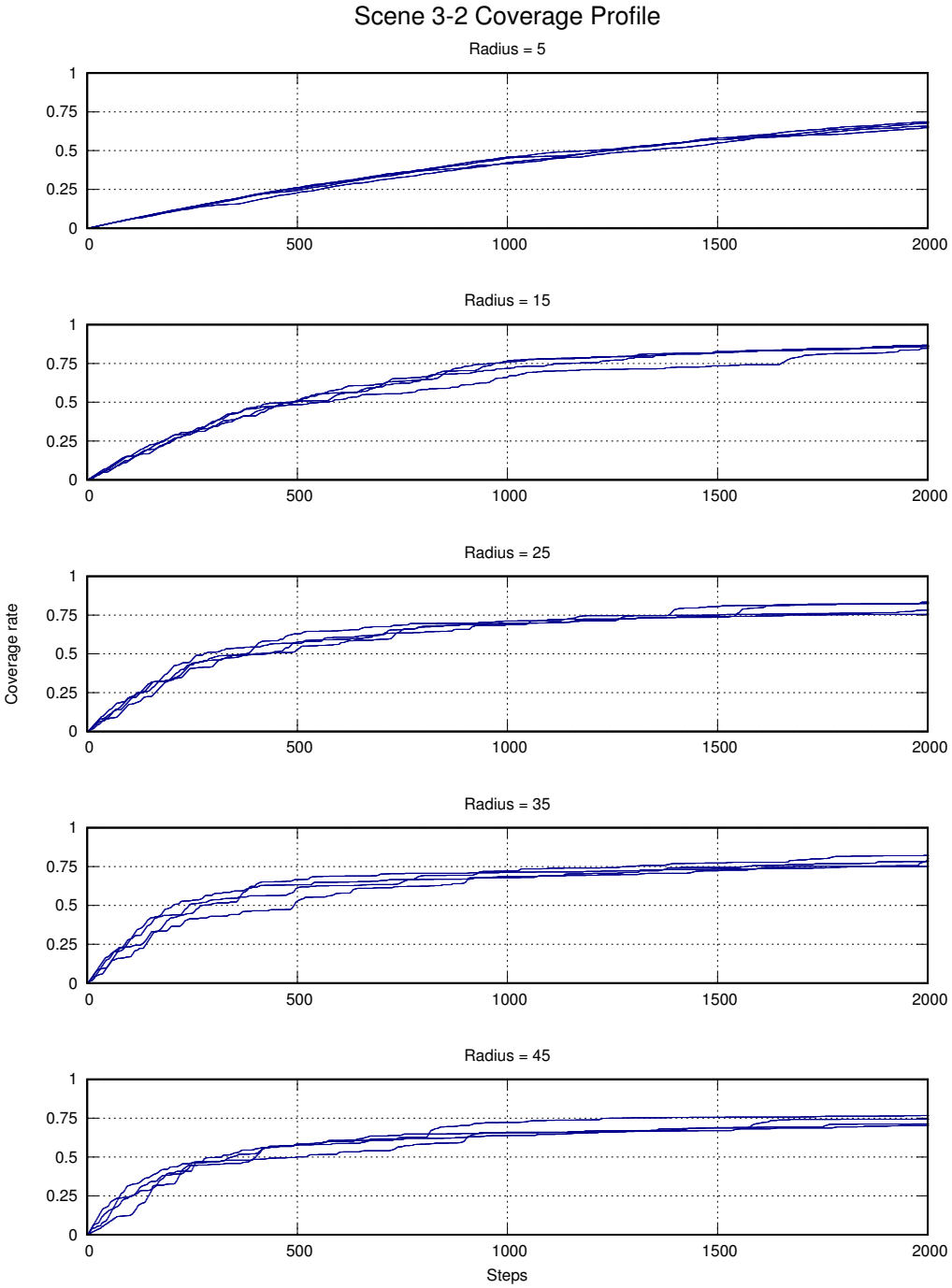

Figure 5.20: Scene 3–2 had similar results to scene 3–1 as seen in this coverage profile.

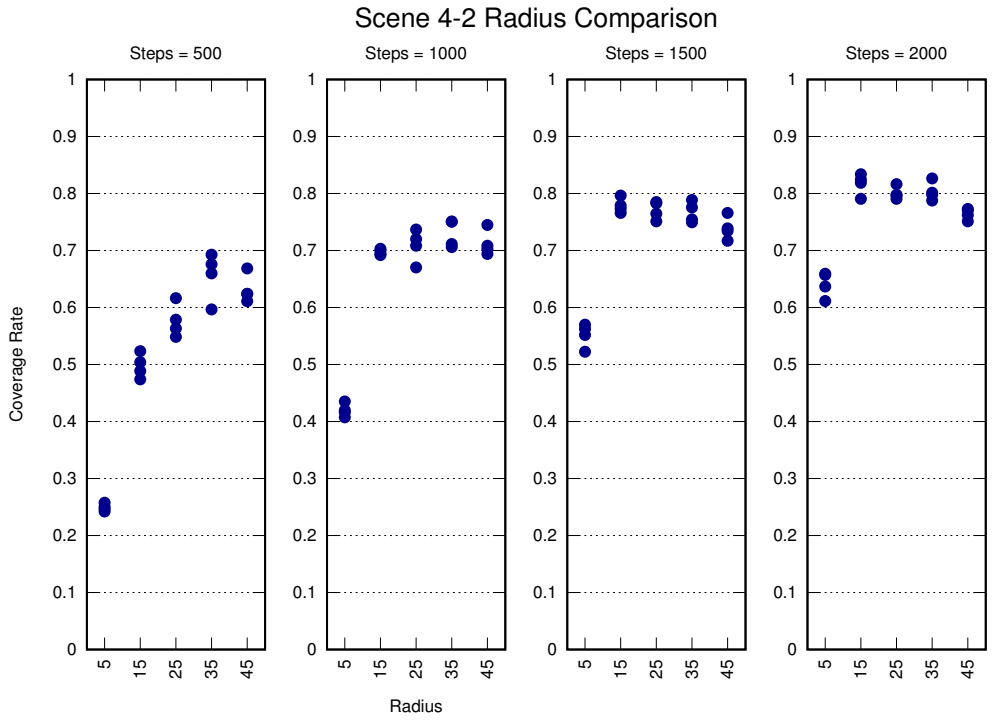

Figure 5.21: This radius comparison shows big improvements for all radius sizes in scene 4–2, but especially the two largest ones.

5.2.5 Scene 4–2: Shelves moved near walls

The shelves in scene 4–2 have been placed along the walls and out of the way. The profile curve in Figure 5.22 shows the tremendous impact this has had on coverage rates, which converge already at 1000 steps for radiuses of 15 cm and greater. Jumps seen in the previous experiment are also gone, which has reduced the variance, and the easier access and greater line of sight have improved the coverage rates from below to above 0.8 at 2000 steps as seen in the radius comparison in Figure 5.21.

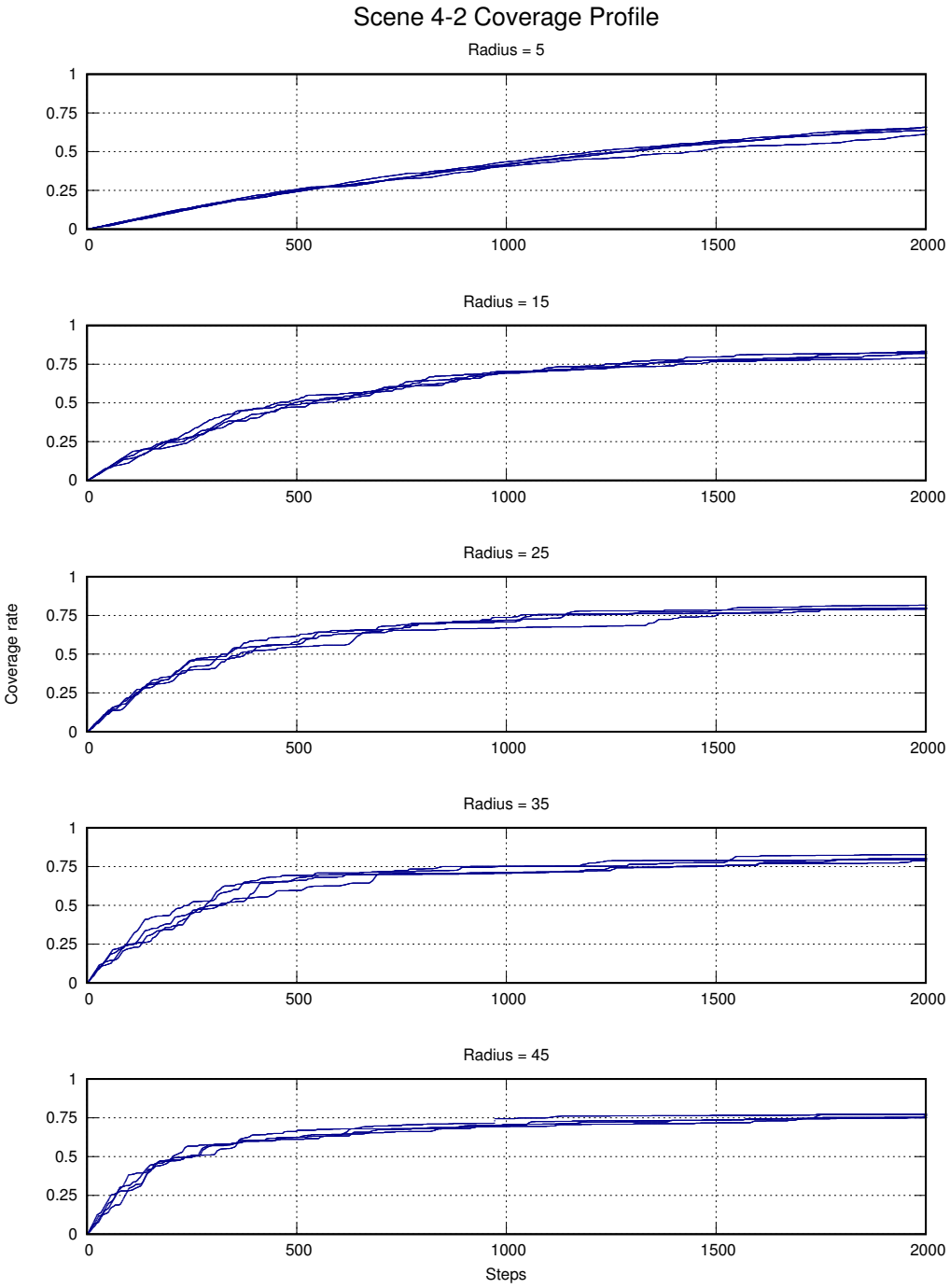

Figure 5.22: Moving the obstacles in scene 4–2 has removed most of the variance seen in experiment 1.

5.3 Experiment 3: Human-in-the-loop

When the robot moves randomly across the surface area, it naturally misses some spots. Some of these spots can be quite large, and seeking them out could have significant gains. In this experiment, the robot is manually placed in these spots by the operator.

5.3.1 Scene 1–3

Radius=15: The least covered area was typically the top right corner of the room. Naturally, the robot would be placed here first. The second placement would be under the desk. This space is covered quickly and does not require the robot to stay, so it was moved outside after a brief period.

Radius=25: A radius of 25 cm would give one meaningful re-placement at 500 steps. Most of the surface would already be covered at 1000 steps and beyond, with only small sections remaining.

General: The robot would often be headed in the direction where the least surface had been covered, causing the operator to wait a few steps to decide where to manually place it. In some cases, the most open area would be covered completely by the robot in such a situation, but typically, it was still the least covered area after the robot had bumped into a wall and was headed in another direction.

Table 5.1: Results scene 1–3, radius=15

Run Number	Step Count	Coverage Rate
4	500	0.4859
4	1000	0.6343
4	1500	0.7297
4	2000	0.7629
5	500	0.4088
5	1000	0.5688
5	1500	0.6860
5	2000	0.7019
6	500	0.4766
6	1000	0.5871
6	1500	0.6679
6	2000	0.6914
7	500	0.4527
7	1000	0.6240
7	1500	0.7078
7	2000	0.7308

Table 5.2: Results scene 1–3, radius=25

Run Number	Step Count	Coverage Rate
2	500	0.5083
2	1000	0.6073
2	1500	0.6287
2	2000	0.6383
3	500	0.5024
3	1000	0.5910
3	1500	0.6462
3	2000	0.6865
4	500	0.4773
4	1000	0.5720
4	1500	0.6724
4	2000	0.7008
5	500	0.4998
5	1000	0.6259
5	1500	0.7214
5	2000	0.7296

Chapter 6

Discussion

When the robot turns in a random direction when it bumps into a wall, it quickly covers the big open areas. After about 2000 steps or 150 metres, we begin to see more dramatic fall-off in new coverage caused by repeatedly going over the same areas. However the coverage along the walls is often less complete, with 'hot spots' in areas where the robot has bumped multiple times.

If we look at the first few steps in each scene, we see that the profile curve is initially linear. Linearity suggests the robot has little to no overlap, and is the best-case scenario. The curve becomes steeper as the radius increases, but so does the variance. Which makes sense, because the larger size can cover more area per time, and will have more difficulty avoiding spots that have been previously covered. The profile curve starts off linear, so when the robot does cover the same spots again, a more pronounced effect is observed with a larger radius.

If we believe it is ideal to quickly cover as much area as possible, then we can use the 2000 steps mark as a basis. The radius comparison charts from Section 5.1 show that a smaller radius of 15 cm has the highest coverage rate in most situations with obstacles. The two cases where it was not optimal were Scene 0 with no obstacles, and Scene 2 where the best coverage rate was found with a radius of 10. In Scene 0–2, the one scene where the room is much larger than the others (4 times the surface area), the highest coverage rate was scored by the second largest radius of 35 cm, but the 45 cm radius scored better overall and had less variance.

A robot of a different size for every scene in the real world can be impractical, therefore we want to investigate if it pays off to go up or down a size from the overall best scoring radius. In scene 1–1, 3–1, and 4–1, downsizing from 15 cm to 5 cm drastically reduces the coverage rate at 2000 steps, but only a small loss is seen if you go up a size. Further increasing the radius shows a linear reduction in coverage rate, at approximately a 10 percentage point loss in the range of a 15 cm to 45 cm radius. Scene 2–1 has a large number of obstacles and it shows that going up or down 10 cm in size both yield small coverage rate losses, but going up increases the variance. More generally speaking, a smaller radius will be able to cover more surface area when given enough time, whereas a larger radius covers more surface area when time is restricted.

We have seen that it would be beneficial for the robot to have sensors that would allow it to access areas that are difficult to reach. Obstacles divide the room into subsections where the robot meanders until it slips through a gap into the next subsection.

6.1 Further work

The software application has provided a good environment for experiments, but there are always expansions that can be made. The PIM and PSM can be further developed to support alternative or more advanced algorithms. Some examples of algorithms are heuristics such as following walls, or more advanced methods such as boustrophedon cell decomposition. It would also be interesting to see how alternative patterns for random coverage would affect coverage rates. For example, a bump could trigger the zigzag pattern. The robot turns 90 degrees twice per cycle in this pattern in order to move in parallel lines. If the robot bumped into something when covering the parallel line, it would not change directions, but it would change direction if it bumped during the short line that offsets the parallel line.

There are still many scenes that haven't been tested. Offices and storage facilities come in configurations of all kinds, and other types of rooms such as classrooms and kitchens are yet to be tested. Not to mention various room sizes.

The effects of physical parameters like rotation speeds were out of scope for this thesis, but it has a major effect on runtime in a real robot.

One can imagine robots that are able to adjust their radius size on-the-fly to achieve even greater coverage performance.

Only circular robots have been used here, but other shapes have been used with great success.

The robot can be equipped with more sensors to intelligently decide paths with a set of criteria. This would be especially helpful to assist the robot in reaching areas that are hard to access because of narrow openings, and could be used to avoid meandering in the same area by favouring longer travel paths.

Human-in-the-loop solutions could be explored. One example is having the robot people to move obstacles aside to give access to uncovered areas.

Chapter 7

Conclusion

We have seen that random coverage is generally reliable. It follows the same pattern every time, albeit with some variance. The differences mainly comes from obstacles that separate the room into subsections where the robot meanders without finding the way to the next subsection, but the random nature of the path-finding does also contribute.

A radius of 15 centimetres is a good choice in all scenes, but does not do quite as good a job as larger radiuses in big open spaces. Other moderate sizes like 25 cm also perform quite well, but they have a history of being slightly too large and will cause a lot of variance when obstacles are introduced in certain configurations where the robot can become trapped, or is otherwise apt to bump repeatedly in the same area. Generally speaking, a smaller radius is more consistent than a larger one, but the tradeoff is slower and less complete coverage. In scenes where the robot is prone to get trapped, a larger radius causes the robot to divide the room into subsections. To answer Research Question 1, the optimal radius to cover the surface of a given scene is 15 cm in most situations if we believe it is optimal to get the greatest coverage rate after a short period of time.

Research Question 2 asks for the impact of obstacles in scenes. Experiments showed that there is some, but not a whole lot of variance in rooms with no obstacles and in rooms with obstacles along the walls. It is only when obstacles are placed in the middle of the room, or when the obstacles extend into the room to effectively change the geometry of the room, that a great variance is introduced. Obstacles can make parts of the room completely inaccessible for radiuses above a certain size, or they can separate the room into sections that make it difficult for random pathfinding to enter. This sectioning has a greater effect the larger the radius, but is also significant for small radiuses. Obstacles can also make the robot meander in specific areas of a scene. Variance caused by obstacles can be minimised by strategic placement. Meandering can be reduced by placing the obstacles along walls. As long as the robot

The results of experiment 1 showed that a large radius is only good if time is the only important factor. If coverage completion is the most important factor, then a small radius is best. A smaller robot has access to areas that would otherwise be inaccessible because it cannot fit between obstacles, and it can reach farther inside corners. This means the smaller the radius, the greater the potential for coverage completion. A moderately sized robot is ideal if optimised for both time and completion, but the results show that the optimal radius size is always smaller when given more time, and larger radiuses comparatively always perform worse over time. Using this knowledge, we can settle on the following guidelines for Research Question 3:

- Select a large radius if only speed is important.
- Select a tiny radius if only coverage completion is important.
- Always select a moderately sized radius (10–25 cm) if both speed and coverage completion are relevant.
- Undersizing is safer than oversizing.

Bibliography

[1] Esther M Arkin and Refael Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218, 1994.

[2] Maxim A Batalin and Gaurav S Sukhatme. Spreading out: A local approach to multi-robot coverage. In *Distributed Autonomous Robotic Systems 5*, pages 373–382. Springer, 2002.

[3] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*, 53(3):263–296, 2008.

[4] Howie Choset. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1-4):113–126, 2001.

[5] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.

[6] Lorenz Gerstmayr-Hillen, Frank Röben, Martin Krzykawski, Sven Kreft, Daniel Venjakob, and Ralf Möller. Dense topological maps and partial pose estimation for visual control of an autonomous cleaning robot. *Robotics and Autonomous Systems*, 61(5):497–516, 2013.

[7] Noam Hazon and Gal A Kaminka. Redundancy, efficiency and robustness in multi-robot coverage. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 735–741. IEEE, 2005.

[8] 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*, pages 176–178. IEEE, 2014.

[9] Bor-Woei Kuo, Hsun-Hao Chang, Yung-Chang Chen, and Shi-Yu Huang. A light-and-fast slam algorithm for robots in indoor environments using line segment map. *Journal of Robotics*, 2011, 2011.

[10] Tae-Kyeong Lee, Sanghoon Baek, and Se-Young Oh. Sector-based maximal online coverage of unknown environments for cleaning robots with limited sensing. *Robotics and Autonomous Systems*, 59(10):698–710, 2011.

[11] Yuming Liang, Lihong Xu, Ruihua Wei, Bingkun Zhu, and Haigen Hu. Behavior-based fuzzy control for indoor cleaning robot obstacle avoidance under dynamic environment. In *2011 Third International Conference on Measuring Technology and Mechatronics Automation*, volume 1, pages 637–640. IEEE, 2011.

[12] Chaomin Luo and Simon X Yang. A bioinspired neural network for real-time concurrent map building and complete coverage robot navigation in unknown environments. *IEEE Transactions on Neural Networks*, 19(7):1279–1298, 2008.

[13] Chaomin Luo, Simon X Yang, Hongwei Mo, and Xinde Li. Safety aware robot coverage motion planning with virtual-obstacle-based navigation. In *Information and Automation, 2015 IEEE International Conference on*, pages 2110–2115. IEEE, 2015.

[14] Joon Seop Oh, Yoon Ho Choi, Jin Bae Park, and Yuan F Zheng. Complete coverage navigation of cleaning robots using triangular-cell-based map. *IEEE Transactions on Industrial Electronics*, 51(3):718–726, 2004.

[15] Yong-Joo Oh and Yoshio Watanabe. Development of small robot for home floor cleaning. In *SICE 2002. Proceedings of the 41st SICE Annual Conference*, volume 5, pages 3222–3223. IEEE, 2002.

[16] Jordi Palacín, Tomás Palleja, Ignasi Valgañón, Ramón Pernia, and Joan Roca. Measuring coverage performances of a floor cleaning mobile robot using a vision system. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4236–4241. IEEE, 2005.

[17] Liam Paull, Sajad Saeedi, Howard Li, and Vincent Myers. An information gain based adaptive path planning method for an autonomous underwater vehicle using sidescan sonar. In *2010 IEEE International Conference on Automation Science and Engineering*, pages 835–840. IEEE, 2010.

[18] Liam Paull, Sajad Saeedi, Mae Seto, and Howard Li. Sensor-driven online coverage planning for autonomous underwater vehicles. *IEEE/ASME Transactions on Mechatronics*, 18(6):1827–1838, 2013.

[19] Hoang Huu Viet, Viet-Hung Dang, Md Nasir Uddin Laskar, and TaeChoong Chung. Ba*: an online complete coverage algorithm for cleaning robots. *Applied Intelligence*, 39(2):217–235, 2013.

[20] Ch K Volos, Ioannis M Kyprianidis, and Ioannis N Stouboulos. Experimental investigation on coverage performance of a chaotic autonomous mobile robot. *Robotics and Autonomous Systems*, 61(12):1314–1322, 2013.

[21] Mohamed Amine Yakoubi and Mohamed Tayeb Laskri. The path planning of cleaner robot for coverage region using genetic algorithms. *Journal of Innovation in Digital Ecosystems*, 3(1):37 – 43, 2016.

[22] Fumio Yasutomi, M Yamada, and K Tsukamoto. Cleaning robot control. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 1839–1841. IEEE, 1988.

[23] Alexander Zelinsky, Ray A Jarvis, JC Byrne, and Shin'ichi Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proceedings of international conference on advanced robotics*, volume 13, pages 533–538, 1993.

Appendix A

Glossary

approximate cellular decomposition	a CPP decomposition where the environment is divided into a fine grid. 5
cell	a segment of a CPP decomposition. 5, 10
CPP	Coverage Path Planning (CPP) is path planning where a robot is assigned a path which covers every point of a surface. 1, 2, 5, 18
dynamic obstacle	obstacles that move. 1, 6
exact cellular decomposition	a CPP decomposition where cells cover all free space. 5
offline	CPP algorithms which can be executed in advance. 6
online	real-time CPP algorithms which rely on sensor data. 6
random search	a CPP heuristic based on turning in random directions. 5
semi-approximate cellular decomposition	a CPP decomposition where cells comprise parellel slices. 5
SLAM	Simultaneous Localisation and Mapping (SLAM) is a navigation method where the robot navigates whilst concurrently creating a map. 6
spiral pattern	a coverage motion which spirals from the perimeter towards the center or vice versa. 5
static obstacle	immutable obstacles. 1, 6
virtual obstacle	obstacles which are exclusively enforced by software. 1

zigzag pattern

a coverage motion consisting of parallel lines. 5