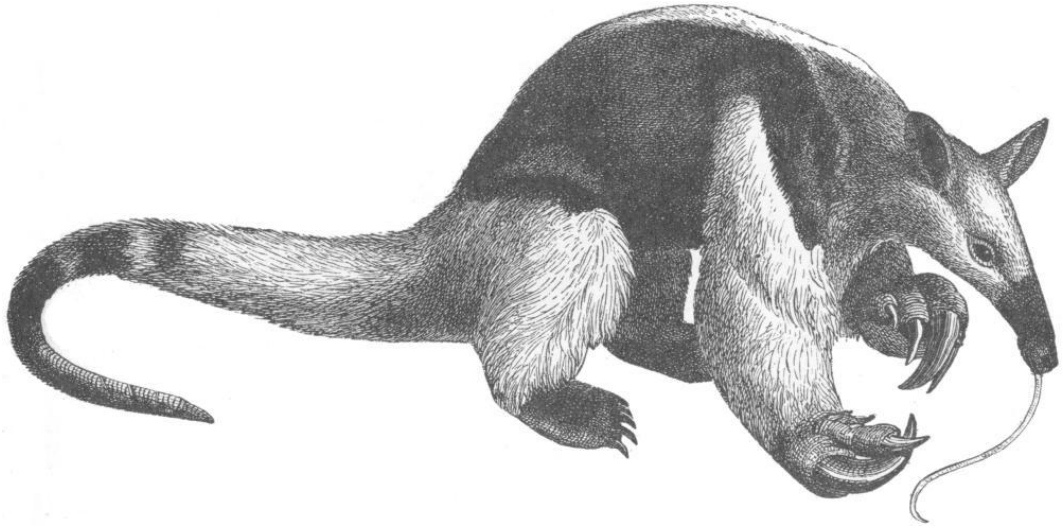
Department of Electrical and Computer Engineering
The University of Auckland
New Zealand



Qualitative Topological Coverage of Unknown Environments by Mobile Robots

Sylvia Wong

February 2006

Supervisors: *Dr Bruce A. MacDonald*
Dr George Coghill

A thesis submitted in partial fulfillment of the requirements of
Doctor of Philosophy in Engineering

Abstract

This thesis considers the problem of complete coverage of unknown environments by a mobile robot. The goal of such navigation is for the robot to visit all reachable surfaces in an environment. The task of achieving complete coverage in unknown environments can be broken down into two smaller sub-tasks. The first is the construction of a spatial representation of the environment with information gathered by the robot's sensors. The second is the use of the constructed model to plan complete coverage paths.

A topological map is used for planning coverage paths in this thesis. The landmarks in the map are large scale features that occur naturally in the environment. Due to the qualitative nature of topological maps, it is rather difficult to store information about what area the robot has covered. This difficulty in storing coverage information is overcome by embedding a cell decomposition, called *slice decomposition*, within the map. This is achieved using landmarks in the topological map as cell boundaries in slice decomposition. Slice decomposition is a new cell decomposition method which uses the landmarks in the topological map as its cell boundaries. It decomposes a given environment into non-overlapping cells, where each cell can be covered by a robot following a zigzag pattern. A new coverage path planning algorithm, called *topological coverage algorithm*, is developed to generate paths from the incomplete topological map/slice decomposition, thus allowing simultaneous exploration and coverage of the environment.

The need for different cell decompositions for coverage navigation was first recognised by Choset. Trapezoidal decomposition, commonly used in point-to-point path planning, creates cells that are unnecessarily small and inefficient for coverage. This is because trapezoidal decomposition aims to create only convex cells. Thus, Choset proposed boustrophedon decomposition. It introduced ideas on how to create larger cells that can be covered by a zigzag, which may not necessarily be convex. However, this work is conceptual and lacking in implementation details, especially for online creation in unknown environments. It was later followed by Morse decomposition, which addressed issues on implementation such as planning with partial representation and cell boundary detection with range sensors. The work in this thesis was developed concurrently with Morse decomposition.

Similar to Morse decomposition, slice decomposition also uses the concepts introduced by boustrophedon decomposition. The main difference between Morse decomposition and slice decomposition is in the choice of cell boundaries. Morse decomposition uses surface gradients. As obstacles parallel to the sweep line are non-differentiable, rectilinear environments cannot be handled by Morse decomposition. Also, wall following on all side boundaries of a cell is needed to discover connected adjacent cells. Therefore, a rectangular coverage pattern is used instead of a zigzag. In comparison, slice decomposition uses topology changes and range sensor thresholding as cell boundaries. Due to the use of simpler landmarks, slice decomposition can handle a larger variety of environments, including ones with polygonal, elliptical and rectilinear obstacles. Also, cell boundaries can be detected from all sides of a robot, allowing a zigzag pattern to be used. As a result, the coverage path generated is shorter. This is because a zigzag does not have any retracing, unlike the rectangular pattern.

The topological coverage algorithm was implemented and tested in both simulation and with a real robot. Simulation tests proved the correctness of the algorithm; while real robot tests demonstrated its feasibility under inexact conditions with noisy sensors and actuators.

To evaluate experimental results quantitatively, two performance metrics were developed. While there are metrics that measure the performance of coverage experiments in simulation, there are no satisfactory ones for real robot tests. This thesis introduced techniques to measure effectiveness and efficiency of real robot coverage experiments using computer vision techniques. The two metrics were then applied to results from both simulated and real robot experiments. In simulation tests, 100% coverage was achieved for all experiments, with an average path length of 1.08. In real robot tests, the average coverage and path length attained were 91.2% and 1.22 respectively.

Acknowledgements

It has been a long time since I first arrived at the E&E department in the University of Auckland as a wide-eyed undergraduate. I am happy to have this opportunity to thank some of the people who have helped make it an enjoyable experience.

First of all, I would like to thank my supervisor, Dr Bruce MacDonald for his support and guidance throughout my PhD years. Without his encouragement and occasional stern looks, I would not have been able to make it to the end.

I would also like to thank my parents. They provided much financial support, which is very appreciated. I hope they feel proud when they see this 200 page masterpiece.

The technical staff in the department have also been of tremendous help. Lance Allen and the workshop designed and built the wooden enclosure for the Khepera robot. Jamie Walker and Evans Leung helped me with all sorts of computer and networking problems. Bev Painter, Nichola Kavacevich and Grant Sargent have also been very helpful.

I greatly enjoyed the interactions with other PhD students in the department, from inspiring technical discussions to playing age of empires. They are (in alphabetical order): Geoff Biggs, Toby Collett, Barry Hsieh, Lee Middleton, Adrian Pais, Russell Smith, Brad Sowden, Chris Waters, Joseph Wong and David Yuen.

Lastly, I would like to thank Jorge Cham, the creator of piled higher and deeper. His comic strips provided me with laughter when I sorely needed it.

Contents

New ideas pass through three periods: 1. It can't be done. 2. It probably can be done, but it's not worth doing. 3. I knew it was a good idea all along!

Arthur C. Clarke

1 Introduction

1.1 Overview of problem domain

A recent survey released by the UN Economic Commission for Europe [1, 41] reported that robot orders for the first half of 2003 were the highest ever recorded. The survey also predicts the worldwide growth rate of the robotic industry will average at 7.4% annually for the period 2003 to 2006. Also, by the end of 2006, a tenfold increase in domestic service robots is predicted. These statistics and predictions show that robots have moved out of science fiction and into everyday life. Nowadays, it is already common to find industrial robots working in hazardous environments or space rovers surveying Mars. In the foreseeable future, domestic and service robots may also become a common sight. Examples include domestic robots mowing lawns and vacuuming floors autonomously, or professional service robots assisting in surgeries and surveillance.

To be useful, a robot has to be skilled in the specific task it is designed for. For example, a lawnmowing robot needs to know how to operate the grass cutting tool it carries; or a rubbish collecting robot needs to know how to pick up soft drink cans and cigarette butts. However, these robots should also be equipped with more general abilities such as obstacle avoidance, map building, path planning and localisation. These abilities enable a robot to move around its environment to do its job efficiently and with minimum human intervention.

One such general ability that is very important to all autonomous robots is path planning. Path planning in robotics is the *intelligence* of finding a path in a map that leads from a start configuration to a goal configuration. This area of artificial intelligence has received a great deal of attention in the robotics research literature [66, 76, 85, 91]. Path planning is an essential component of robot manipulator controllers, as it is the the basis for describing and controlling the manipulator tip [42, 54]. It is also important for mobile robots, as it is the basis for describing and controlling the varying robot positions in space [46, 98].

Most path planning algorithms are for point-to-point path planning. This type of algorithm usually attempts to find the shortest or quickest path to get from one point to another. Though sometimes criteria other than path lengths are used [71]. However, in some applications, a *coverage* path is needed instead. The aim of a coverage path planner is to create a path that covers all surfaces in an environment. In other words, given an initial location, it does not matter where the final location is, as long as the journey visits all surfaces in the environment. Examples of robotic tasks that require a coverage path are cleaning [38], surface coating [86, 87], humanitarian demining [69] and foraging.

Coverage path planning is similar to exploration, but not the same. When exploring, a robot sweeps its long range sensors, moving so as to sense all of its environment, often to build a map. In a coverage application, the robot or a tool it carries must *pass over* all the floor surface.

Compared to point-to-point path planning, the coverage path planning problem has not received as much attention. However, as robots move into service roles and interact with humans in more varied environments for a wider range of tasks, coverage will become more important. The ability to fully cover an environment will be a key capability for all mobile systems. For example, domestic robot assistants can spend their idle time cleaning, collecting items and storing them away.

Apart from generating different types of output, path planners also differ in their formats of input (the map). As path planning is essentially a search on a map of the environment, the data structure used to store this map naturally influences the operations of path planning algorithms. Also, depending on the application domain, the environment the robot operates in maybe known or unknown beforehand. If a map is created by a human operator and fed to the path planner, the environment is *known* to the robot. If no map is provided, the environment is *unknown*, and the robot has to construct a map for path planning using sensor information. There are two distinct ways to handle this situation. The first method is to carry out an exploration phase to construct an accurate map [95, 107] before any path planning is done. The alternative is to make assumptions concerning the unknown areas in the map in order to commence path planning, and then update the planned path whenever new environmental information becomes available [58]. In other words, path planning is done on a *partial map* of the environment.

1.2 Applications of coverage path planning

Coverage path planning is needed in a variety of mobile robot applications. The focus of this thesis is on the coverage of flat, indoor, unknown surfaces populated with obstacles. It is also assumed that the robot has to stay within the region. Typical applications that fit these criteria are vacuum cleaning and floor scrubbing.

Lawn mowing is very similar, but the restriction on staying within bounds is relaxed. For example, it is perfectly acceptable to push the lawn mower over the footpath while cutting grass on the kerb. Compared to a typical home or office, the average lawn is relatively free from obstacles. Also, being an outdoor application, global positioning systems can be used to aid localisation and landmark matching.

Intuitively, humanitarian demining should also allow the robot to stray outside the area to be covered. However, since it is unknown whether the region outside is free of landmines, it is safer and smarter to restrict movement within bounds of the environment. Also, due to the dangers of the mines, the robot should not move into surfaces not scanned by the landmine detection tool yet. Therefore, localisation must be very accurate. Otherwise, due to dead reckoning[1] error, the robot might move into an area it believes to be covered, but is not in reality.

In window cleaning the target surface is vertical, instead of horizontal. Other than this minor difference, the coverage requirement is essentially the same as vacuum cleaning.

In machine milling, it might be desirable to mill only in one direction (either in the spindle direction, or against). This means the coverage path should be a sequence of, say, right-to-left movements, instead of alternate left-to-right and right-to-left movements. This is because milling in only one direction gives better surface quality [51].

1.3 Description of the thesis

1.3.1 Scope of the research

The purpose of this research is to develop robust coverage algorithms for mobile robots working in unknown environments. I do not assume known environments because it can be costly and inflexible to require a complete map of the environment the robot operates in. In certain types of robots, for example, domestic vacuum cleaners, owners generally lack the expertise to enter detailed maps to the robots and will therefore require professional help for such tasks. Also, the map will need to be updated whenever the owner re-arranges the furniture.

[1] The estimation of a mobile robot's position from the distance it has travelled and the direction it is heading.

Map building is a key issue in this thesis because of the requirement of unknown environments. The robot should simultaneously construct a map with sensor information while covering the environment at the same time. This means that the coverage path planner has to make its decision based on a partial map of the robot's environment. Using a separate exploration phase for map building purposes is considered inefficient because a coverage path already requires the robot to visit all surfaces.

An integral part of developing a robotic coverage algorithm is to measure how well the algorithm performs in experiments. Performance metrics allow quantitative evaluation of implementations. They also permit comparisons between different algorithms. Despite the importance of quantitative metrics, this is an area that has received very minimal attention. Development of suitable performance metrics is therefore another aim of this research.

1.3.2 Overview of the thesis

The thesis is divided into the following chapters:

Chapter 2 presents a literature review of algorithms for coverage using mobile robots. The review includes algorithms for both known and unknown environments. The chapter also discusses existing performance metrics for evaluating coverage algorithms in both simulation and real robot experiments.

Chapter 3 presents the events and algorithms for slice decomposition. Two versions of the decomposition are presented. The first one is for known environments, and is produced using a normal line sweep process. The second one is for unknown environments, where the sweep line will be limited to within free space.

Chapter 4 introduces the topological coverage algorithm. The algorithm creates a slice decomposition of any environment online, without a known map. It explains methods for detecting landmarks used to form the decomposition. It also talks about how slice decomposition is stored in a topological map, how the map is maintained and updated, and how to determine if the map is completed and the environment is completely covered. It also explains why travelling between regions is robust. Completeness and complexity of the decomposition are also discussed.

Chapter 5 introduces two new performance metrics for coverage experiments. The chapter also includes practical methods for evaluating and measuring parameters needed to calculate these metrics.

Chapter 6 focuses on the implementation of the topological coverage algorithm. It describes the simulation and real robot environment and platform. It also talks about the implementation of the topological map and the robot controller.

Chapter 7 shows results from experiments in simulation and with a real robot. The metrics introduced in Chapter 5 are used to evaluate the performance of the experiments. The results are also discussed and compared with existing coverage algorithms.

Chapter 8 presents a list of potential future work and some conclusions drawn from the research in this thesis.

Appendix A describes an alternative method for recognising and classifying the landmarks used in the topological map. Two types of neural networks are trained to learn the different landmarks. This appendix gives a brief introduction to the two neural networks, followed by results from landmark recognition tests.

Appendix B covers the computer vision techniques used in Chapter 5 for creating composite images in greater detail. Topics discussed include Canny edge detection and Hough transforms.

1.4 Contributions of the thesis

This thesis makes several significant contributions. A study in the existing literature provides the basis for the identification of areas where contributions can be made to the field.

First is the development of an online coverage algorithm that uses a partial qualitative topological map for planning. Previously, qualitative maps based on simple landmarks have only been used in point-to-point path planning. This is because nodes and edges of topological maps do not correspond to specific locations in space. This qualitative nature makes it difficult to store coverage information. The problem is overcome by using the landmarks as cell boundaries of slice decomposition. In other words, the topological map embeds a slice decomposition of the environment. As a result, even though individual nodes in the map are not associated with specific areas of space, a combination of nodes now defines a cell of the decomposition. Coverage information is then stored with cells in slice decomposition.

Second is the introduction of slice decomposition, a cell decomposition for covering unknown environments. It can handle a larger variety of environments than existing cell decomposition based coverage algorithms. The concept of using the split and merge of the sweep line by obstacles as cell boundaries was first introduced in boustrophedon decomposition [30]. This approach creates maximum sized cells that can still be covered by a simple zigzag pattern. However, boustrophedon decomposition lacks detailed algorithms or implementation details. Slice decomposition extends the split and merge concepts in boustrophedon decomposition. New cell boundary types are added to simplify boundary detection in online decomposition with range sensors. Similar to slice decomposition, Morse decomposition [8] is also for covering general unknown environments using range sensors. The main difference between Morse

decomposition and slice decomposition is in the choice cell boundaries. Morse decomposition uses surface gradients. As obstacles parallel to the sweep line are non-differentiable, rectilinear environments cannot be handled by Morse decomposition. In comparison, slice decomposition uses topology changes and range sensor thresholding as cell boundaries. Due to the use of simpler landmarks, it can handle environments with polygonal, elliptical and rectilinear obstacles.

Thirdly, due to the use of simple landmarks as cell boundaries, the topological coverage algorithm employs a shorter navigation pattern to cover each cell in the decomposition than Morse decomposition. Wall following on side boundaries of cells is needed in Morse decomposition to discover connected adjacent cells. This is because cell boundaries can only be detected when they are the closest point on the obstacle surface from the robot compared to all other points on the obstacle surface. Therefore, a rectangular pattern that includes retracing is used to cover each cell in the decomposition. On the other hand, due to the use simpler landmarks and a more general technique for landmark detection, the topological coverage algorithm allows a robot to detect events in slice decomposition from all sides. As a result, a simple zigzag path that does not include any retracing can be employed instead. Due to the use of a shorter navigation pattern to cover individual cells, the topological coverage algorithm generates coverage paths that are shorter and more efficient.

Lastly, new performance metrics for evaluating real robot coverage experiments are developed. Currently, results from real robot experiments are mostly presented qualitatively, showing pictures of the routes taken by the robots. The only metric available is the coverage factor [22]. However, it does not measure effectiveness or efficiency of experiments properly. The two metrics proposed in this thesis measure the effectiveness and efficiency of any coverage experiment using data collected with computer vision techniques. The methods used to collect the data are very general and are not limited to the experimental setup used for this thesis.

Only in our dreams are we free. The rest of the time we need wages.

Terry Pratchett, "Wyrd Sisters"

2

Coverage navigation and path planning

Path planning for mobile robots generally involves a search on a spatial representation (map) of the environment. Therefore mapping and path planning are two related issues and cannot be examined in complete isolation. Robotic maps are data structures that store information about the environment. For any given data structure, there are multiple ways to conduct a search. For example, there are numerous search algorithms for graphs, such as A* search and depth-first search [92]. In summary, there are many types of robotic maps, and even more path planning algorithms.

This chapter first introduces several common robotic maps. They are Voronoi diagram (Section 2.1), cell decomposition (Section 2.2) and grid map (Section 2.3). These maps all employ some form of space decomposition, where a complex space is repeatedly divided until simple subregions of a particular type are created. Grid based maps are a special case of space decomposition where the environment, both free space and obstacles, is decomposed into uniform grid cells. Emphasis has been placed on space decomposition based maps as they are the most common data structure used in coverage path planning. This is because coverage algorithms usually use the strategy of "divide and conquer". Basically the environment is segmented into simpler subregions, and each subregion is then covered in turn. Cell decomposition is favoured by Choset, the leader in the area of robot coverage algorithms [8, 9, 11]; while grid maps are the most common choice among researchers in mobile robot coverage [44, 97, 109].

Two other robotic mapping and path planning methods are also discussed. Even though they are not based on space decomposition, they are included here because of their importance in mobile robot navigation. The two methods are topological mapping (Section 2.4) and reactive robotics (Section 2.5). Topological maps do not form precise geometric representations of space. Instead the map describes spatial relationship in a qualitative way, much like the way humans describe their environments. Reactive robotics approaches the navigation problem from a non symbolic AI perspective. As such, no maps are used and there is no planning of paths in the traditional sense.

Section 2.8 contains a survey on the performance metrics used in simulated and real robot experiments of coverage algorithms. In also includes a brief discussion of the complexity, or upper bound, of the length of a coverage path.

Lastly, this chapter finishes with a discussion that identifies areas where contributions can be made.

2.1 Voronoi diagram

Perhaps the most popular space decomposition is the Voronoi diagram. It is used in a wide range of disciplines including biology, computational geometry, crystallography and meteorology [81].

Given a finite set of distinct points (called reference vectors) in the Euclidean plane, an ordinary Voronoi diagram is formed by associating all locations in that space with the closest member of the point set with respect to the Euclidean distance. Thus a Voronoi diagram partitions the space into a set of non-overlapping regions. Figure 2.1 shows a set of reference vectors with its Voronoi diagram.

More formally, let the set of n reference points be $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, and their Cartesian coordinates be $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Then the ordinary Voronoi region associated with reference point p_i is given by

$$V(p_i) = \{\mathbf{x} \mid \|\mathbf{x} - \mathbf{x}_i\| \leq \|\mathbf{x} - \mathbf{x}_j\| \text{ for } i \neq j\}$$

And the set given by

$$\mathbf{V} = \{V(p_1), \ldots, V(p_n)\}$$

is the ordinary Voronoi diagram of the reference point set P.

By applying a distance measure other than Euclidean distance, the ordinary Voronoi diagram has been extended or generalised in many directions [81]. One of the most useful generalisations for robotics is the area Voronoi diagram. This is because a typical robots' environment consists

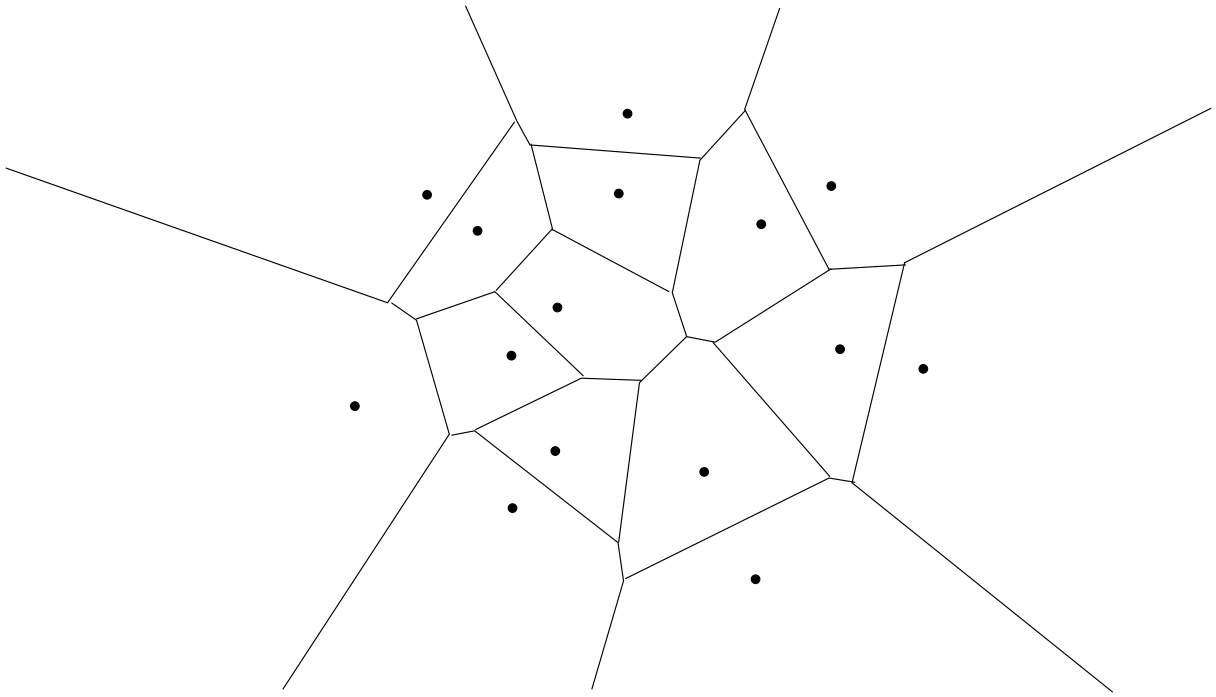


Figure 2.1: Voronoi diagram for a set of reference points.

of a set of n obstacles $A = \{A_1, \ldots, A_n\}$, rather than a set of points. The area Voronoi diagram is calculated using the following equation for the shortest distance from point p to obstacle A_i:

$$d_s(p, A_i) = \min_{x_i}\{\|\mathbf{x} - \mathbf{x}_i\| \mid \mathbf{x}_i \in A_i\}$$

In other words, an area Voronoi diagram (or simply generalised Voronoi diagram) is formed by associating all locations in the free space of the robot's environment with the closest obstacle. Figure 2.2 shows an example of a generalised Voronoi diagram for an environment with two obstacles.

In robotics, generalised Voronoi diagrams have been used in path planning [66], topological mapping [84] and localisation with visual landmarks [106]. A feature of maps based on generalised Voronoi diagrams is that they maximise clearance between robot and obstacles. A robot following the map will be staying far from obstacles. As a result, the Voronoi diagram is undesirable for coverage tasks. On the other hand, this characteristic makes it perfect for exploration navigation with long range sensors. For example, Acar, Choset and Atkar used a map based on the generalised Voronoi diagram for a robot equipped with an extended range landmine detector [6].

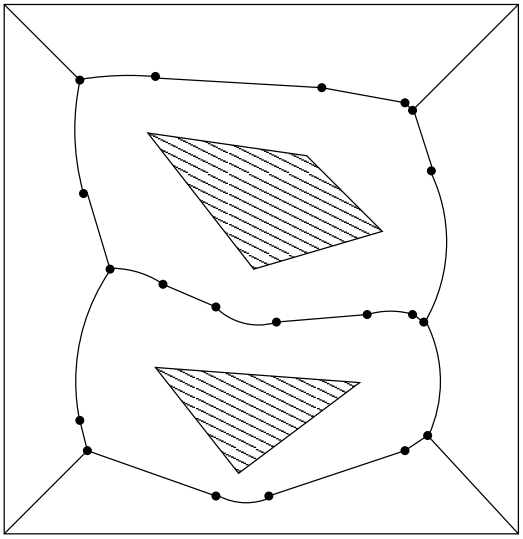


Figure 2.2: The Voronoi diagram maximises clearance between robot and obstacles. (Reproduced from page 172 of [66]).

2.2 Cell decomposition

A cell decomposition divides a complex structure S into a collection of disjoint simpler component cells. In exact cell decomposition, the union of the component cells is exactly S. While in approximate cell decomposition, the union of the component cells is approximately S. The boundary of a cell corresponds to a criticality[1] of some sort. The most common example of cell decomposition is trapezoidal decomposition [66]. It is formed by sweeping a line across the environment, and maintaining a list D of cells that intersects with the sweep line. The history of list D, ie all the cells that have appeared in D, forms the decomposition. A cell boundary is created whenever a vertex is encountered. Due to the use of vertices as criticality, obstacles are limited to polygons. Also, each cell of a trapezoidal decomposition is either a trapezoid or a triangle. The algorithm for trapezoidal decomposition is shown in Algorithm 2.1.

Figure 2.3 shows an example of trapezoidal decomposition for an environment with one polygonal obstacle. Originally, the list D consists of only one cell, and thus $D = (c_1)$. At the first vertex, cell c_1 is split into two parts, and D changes to (c_2, c_3). For the next two vertices in the environment, cells are replaced only. The list D changes to (c_2, c_4) and then again to (c_5, c_4). Lastly, cells c_4 and c_5 are merged into one cell, and D becomes (c_6).

The trapezoidal decomposition was originally proposed by Chazelle to partition a 3D polyhedron into a collection of convex polyhedra [29]. Other usages outside of robotics include decomposing complex polygons in 2D computer graphics [88].

For path planning, the trapezoidal decomposition is first reduced to a connectivity graph that

[1] Criticalities in cell decompositions are conditions of the sweep line where, if satisfied, a new cell boundary is created.

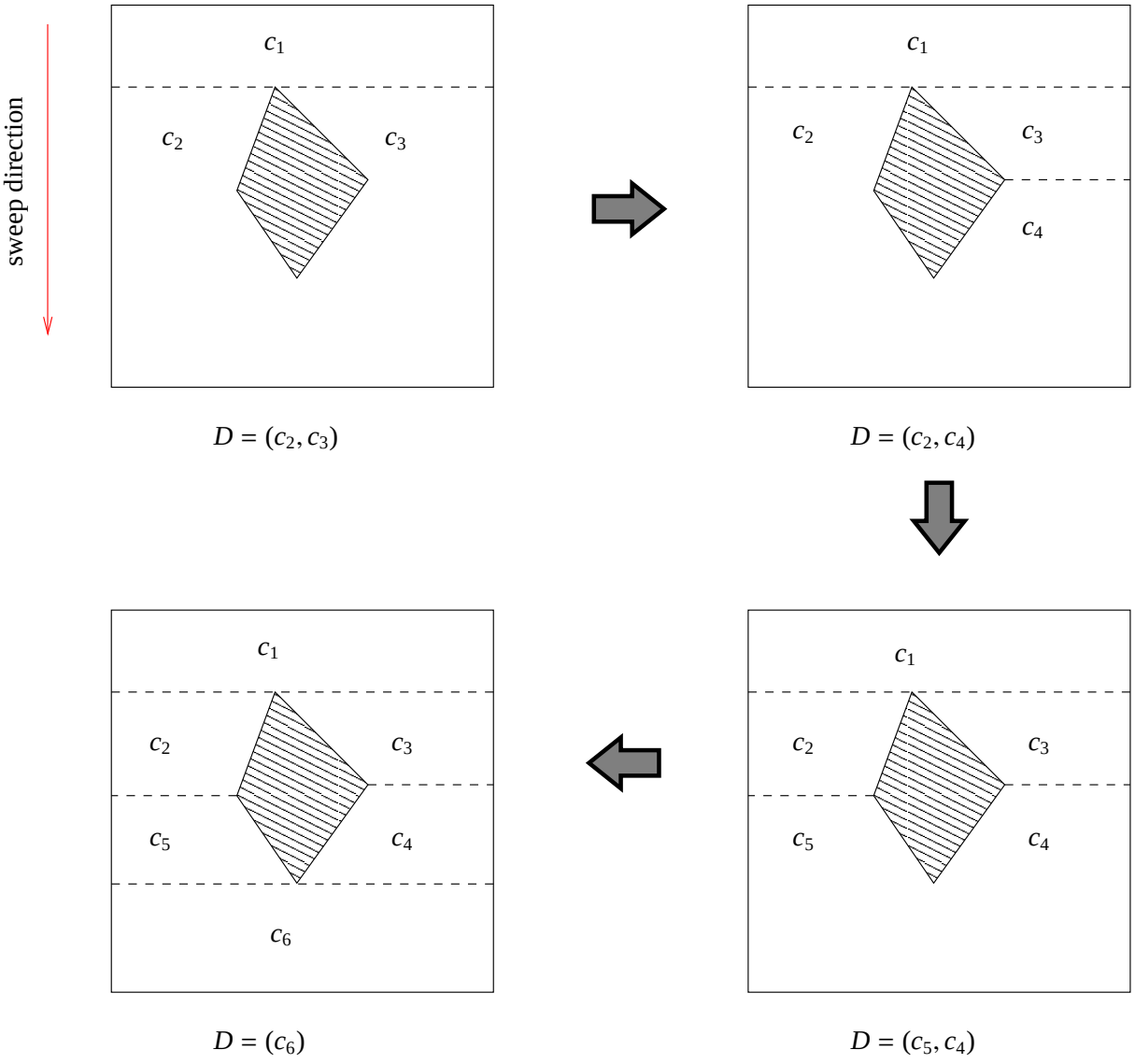


Figure 2.3: Trapezoidal decomposition is formed by sweeping a line over the environment. A new cell boundary is created whenever the sweep line encounters a vertex. Cell boundaries are shown as dotted lines.

Algorithm 2.1 Trapezoidal Decomposition

y_L: position of sweep line
$\{y_1, \ldots, y_n\}$: sorted list of y coordinates of all vertices in environment
$D = (\ldots, c_{i-2}, c_{i-1}, c_i, c_{i+1}, c_{i+2}, \ldots)$
for $y_L = y_1$ to y_n **do**
  if vertex splits cell c_i into two **then**
    $(c_i) \leftarrow (c_d, c_{d+1})$
    $D = (\ldots, c_{i-2}, c_{i-1}, c_d, c_{d+1}, c_{i+1}, c_{i+2}, \ldots)$
  else if vertex merges two cells c_i and c_{i+1} **then**
    $(c_i, c_{i+1}) \leftarrow (c_e)$
    $D = (\ldots, c_{i-2}, c_{i-1}, c_e, c_{i+2}, \ldots)$
  else if vertex replaces cell c_i with a new cell **then**
    $(c_i) \leftarrow (c_f)$
    $D = (\ldots, c_{i-2}, c_{i-1}, c_f, c_{i+1}, c_{i+2}, \ldots)$
  end if
end for

represents the adjacency relation among the cells [66]. Then this associated connectivity graph is searched to find paths between any two cells. Figure 2.4 shows the connectivity graph for the trapezoidal decomposition formed in Figure 2.3.

Choset first recognised that trapezoidal decomposition creates cells that are unnecessarily small, and therefore inefficient, for coverage purposes [30, 32]. Trapezoidal decomposition creates cells that are convex polygons only. However, non-convex cells can also be covered completely by simple coverage patterns. A decomposition that creates more cells are less efficient because for each cell, additional motion along the cell boundary maybe required. For example, the two cells on each side of the obstacle in Figure 2.5(a) can be merged and a simple zigzag pattern can still cover the combined cells (Figure 2.5(b)).

Based on this concept of merging multiple cells in trapezoidal decomposition, Choset proposed the first exact cell decomposition specifically designed for coverage navigation [30]. The resulting decomposition is called boustrophedon[2] decomposition, signifying the relationship between the decomposition and the zigzag. Boustrophedon decomposition introduces the idea of using the split and merge of the sweep line by obstacles as criticality. This is explained in the example in Figure 2.6. However, [30] does not provide a detailed algorithm for the decomposition, nor does it define the criticality precisely. Moreover, it is unclear if, or how, concave obstacles are handled.

Huang attempted to reduce the cost of coverage by minimising the number of turns in the coverage path [55]. He introduced the Minimal Sum of Altitude (MSA) decomposition. The decomposition works on known polygonal environments. The basic premise behind MSA de-

[2] Alternately from right to left and from left to right, like the course of the plough in successive furrows (Oxford English Dictionary).

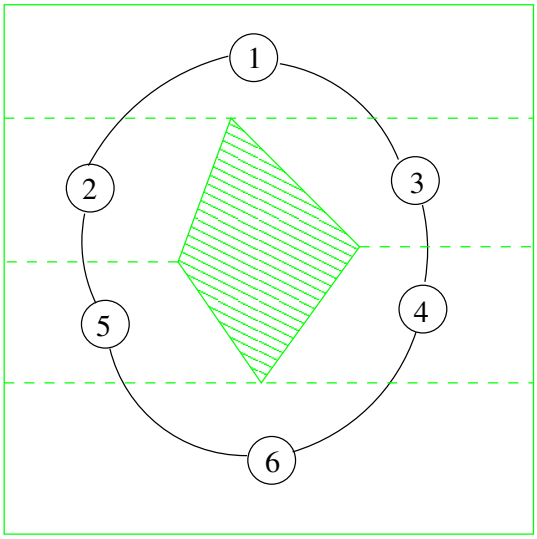


Figure 2.4: To form the associated connectivity graph, each cell is labelled by a distinct integer and connected to its neighbouring cells.

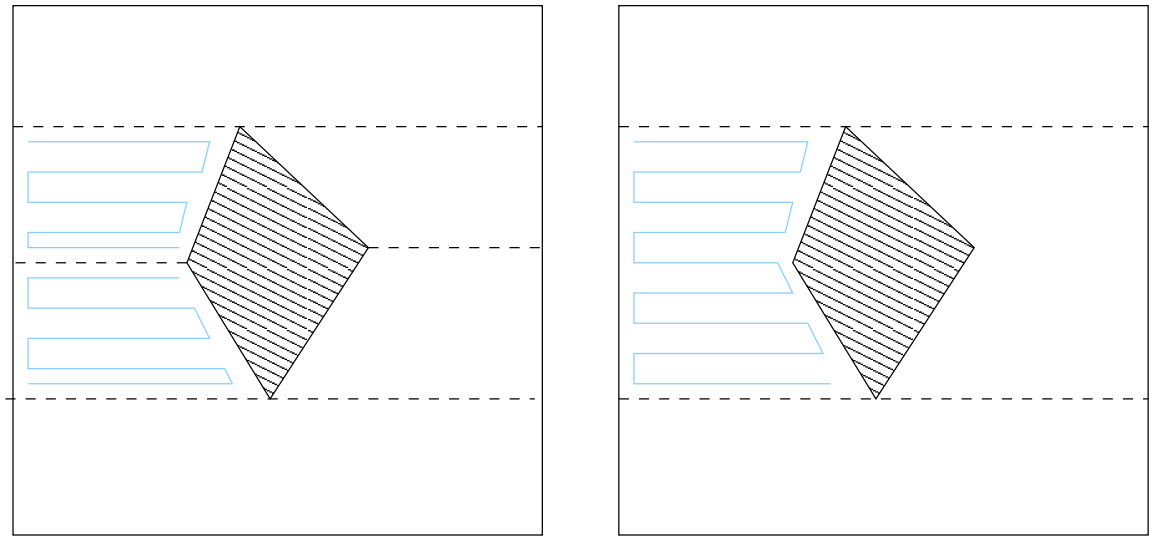


Figure 2.5: (a) Trapezoidal decomposition creates cells that are unnecessarily small for coverage tasks. This is because some non-convex cells can be covered by a simple zigzag path. (b) Boustrophedon decomposition reduces the number of cells in trapezoidal decomposition. This is achieved by combining multiple cells that can be covered by a zigzag.

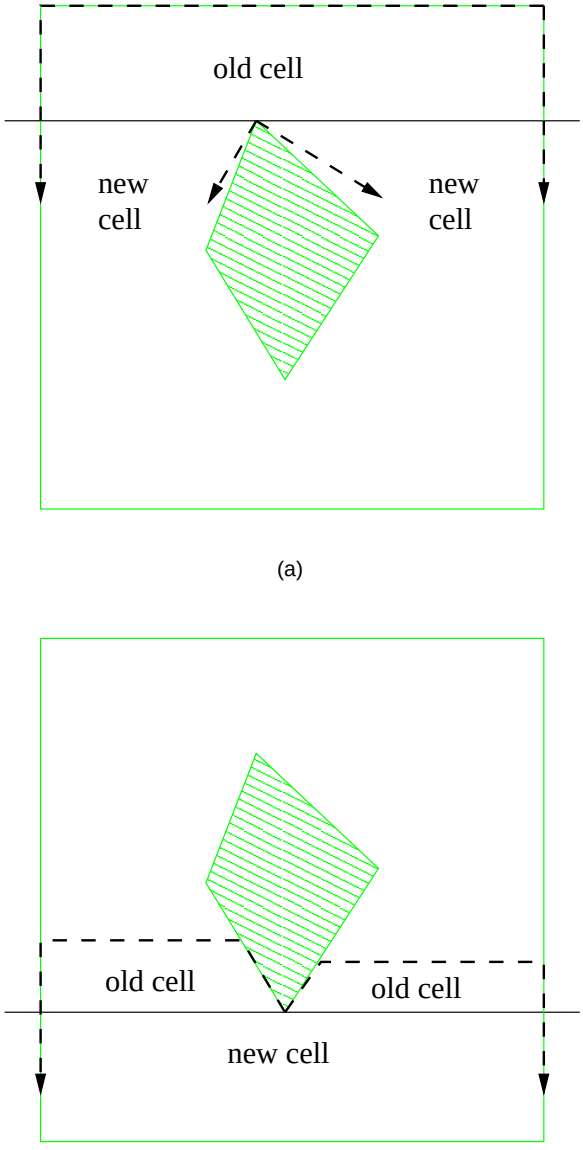


Figure 2.6: Criticalities in boustrophedon decomposition: (a) In event, (b) Out event.

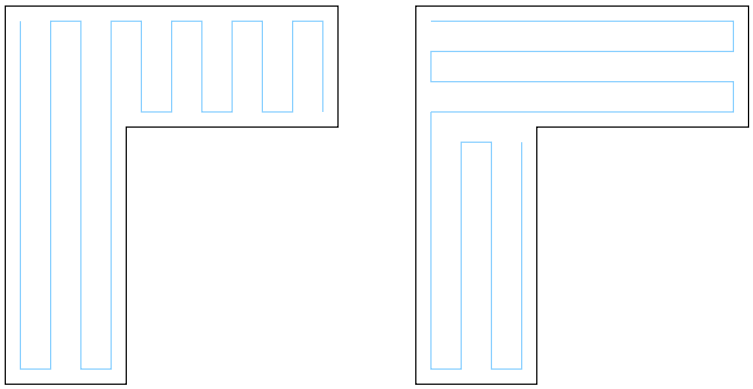


Figure 2.7: Assigning different sweep directions to cells can produce coverage paths with fewer turns. (Reproduced from [55]).

composition is that cost of coverage is lower when there are fewer turns in the path. This is because a robot must slow down to make a turn. Huang assumed that the cost of travelling between cells in the decomposition is significantly lower than the cost of turning. By choosing different sweep directions for different cells, the cost of a coverage path can be lowered. This is illustrated in the example in Figure 2.7.

The MSA decomposition is created with a two step process – multiple line sweeps followed by dynamic programming. For each edge orientation (of both the boundary of the environment and all obstacles), a line sweep is performed. A cell boundary is created for each vertex at split and merge events.[3] The decompositions from all edge orientations are then overlaid upon each other. Figure 2.8 shows an example of this multiple line sweep decomposition process.

Once the initial decomposition from multiple line sweeps is formed, an adjacency graph is created to represent the decomposition. An example of this graph is shown in Figure 2.9. The basis of the dynamic programming step is to either split this graph in two, thus creating two smaller subproblems; or to try to unite all cells and cover them as one large region. The minimum sum of altitudes of graph G is defined as:

$$S(G) = \min\left\{C(G), \min_i S(G_1^i) + S(G_2^i)\right\} \tag{2.1}$$

where i iterates over all possible ways to split the graph G into two connected subgraphs G_1 and G_2. $C(G)$ is the cost of covering all cells as one subregion. Figure 2.9 shows an example of the first level of decomposing a problem. Figure 2.10 shows the final MSA decomposition for a simple environment.

[3] An event in a cell decomposition occurs when the sweep line encounters a criticality. Therefore, a duality exists between criticalities and events.

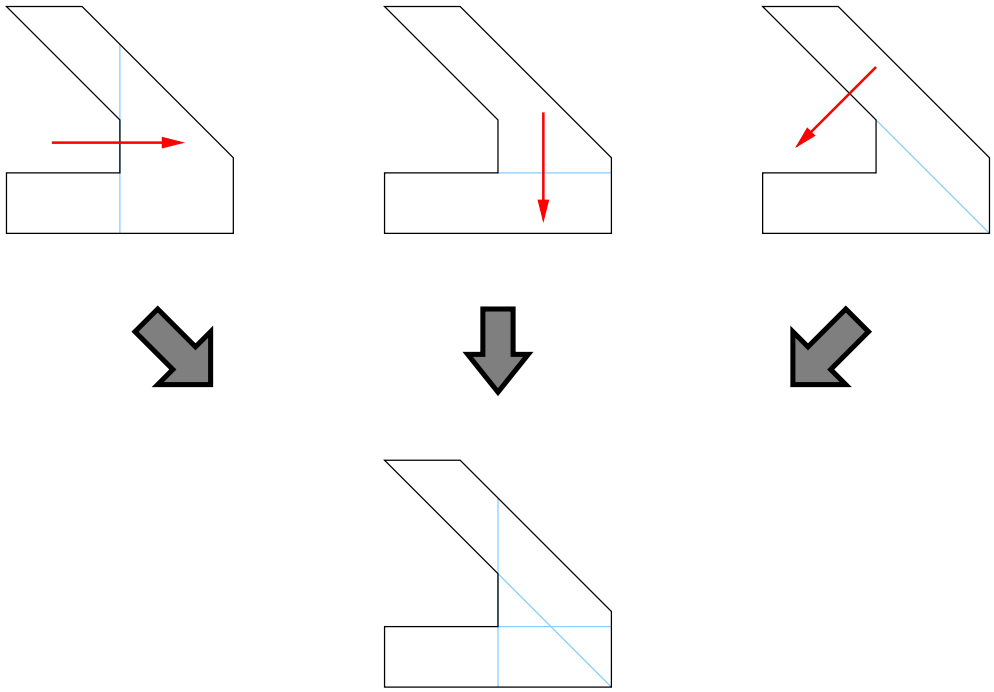


Figure 2.8: The first step in creating a MSA decomposition is multiple line sweep. A line sweep is performed for each edge orientation. The decomposition of all the line sweeps are then overlaid on top of each other. (Reproduced from [55]).

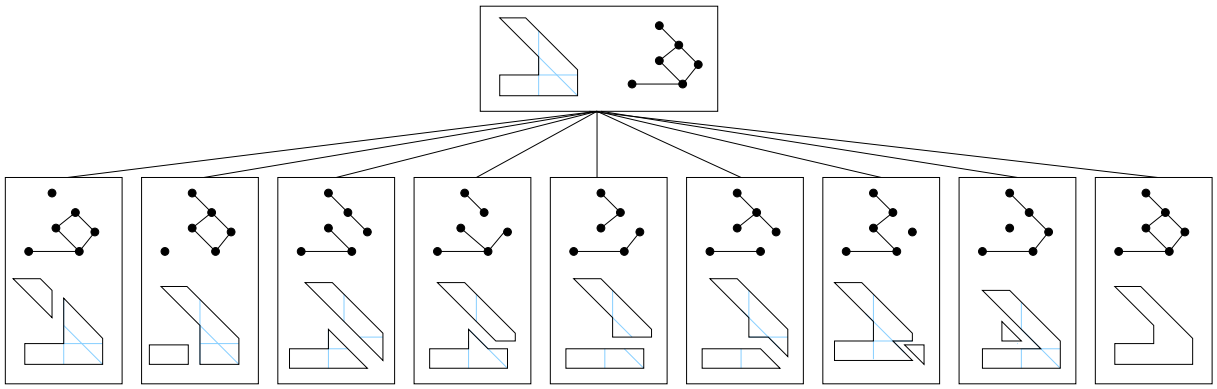


Figure 2.9: First stage of dynamic programming in MSA decomposition. The top box shows the initial decomposition from multiple line sweep, and its adjacency graph. There are 8 ways this graph can be split into two connected graphs. The rightmost box in the bottom row represents the choice of covering all cells as a single region. (Reproduced from [55]).

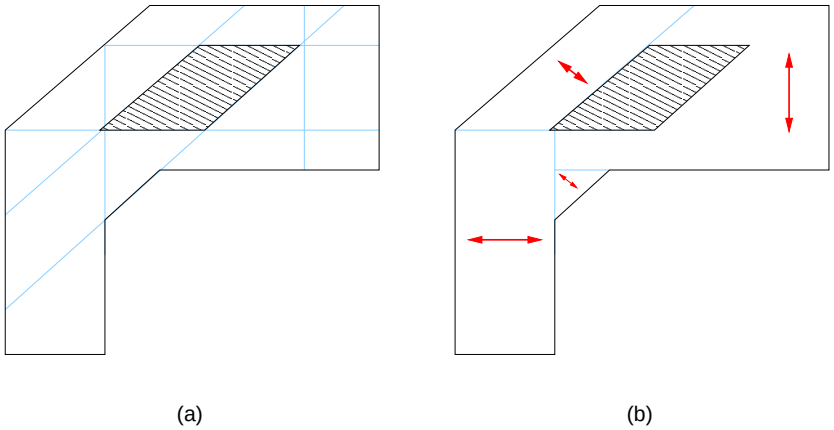


Figure 2.10: MSA decomposition of a simple environment. (a) Initial decomposition from multiple line sweeps. (b) After dynamic programming. (Reproduced from [55]).

However, MSA decomposition is limited in the complexity of environments it can handle. This is because of an exponential complexity for the algorithm. Firstly, each sweep direction contributes a dividing line that divides many cells. This produces a large number of cells in the adjacency graph. Secondly, the dynamic programming phase must examine all connected subgraphs of 1 to n nodes for a graph of n nodes.

Both boustrophedon and MSA decompositions are defined only for known environments. The idea of using split and merge events of the sweep line as criticality was extended to unknown environments in the works of Butler [22, 23] and Acar [8, 12] [4].

For a cell decomposition to be used for covering unknown environments, the following issues need to be addressed. Firstly, mobile robots can only move within the free space region of the environment. As a result, the sweep direction can no longer be from top to bottom only. For example, the area underneath the U-shaped obstacle in Figure 2.11 can only be swept in the reverse (bottom to top) direction. Secondly, planning of the coverage path has to be done using a partial cell decomposition of the environment. This is because the cell decomposition has to be created simultaneously with the coverage process. Thirdly, criticalities can occur between sweep line positions. An example of which is shown in Figure 2.12, where the vertex is positioned between strips of the zigzag. Lastly, the criticality chosen has to be realistically detectable by robot sensors.

Butler *et. al.* proposed CC_R, an exact cell decomposition for contact sensing robots[5] covering unknown rectilinear environments [23]. Cell boundaries are formed whenever an obstacle boundary parallel to the sweep line is encountered. An example of CC_R is shown in Figure 2.13.

[4] Butler and Acar were PhD students at the Robotics Institute at Carnegie Mellon University. Choset is a professor at the same institute, and is the supervisor of Acar.

[5] Robots that have no range sensing capabilities.

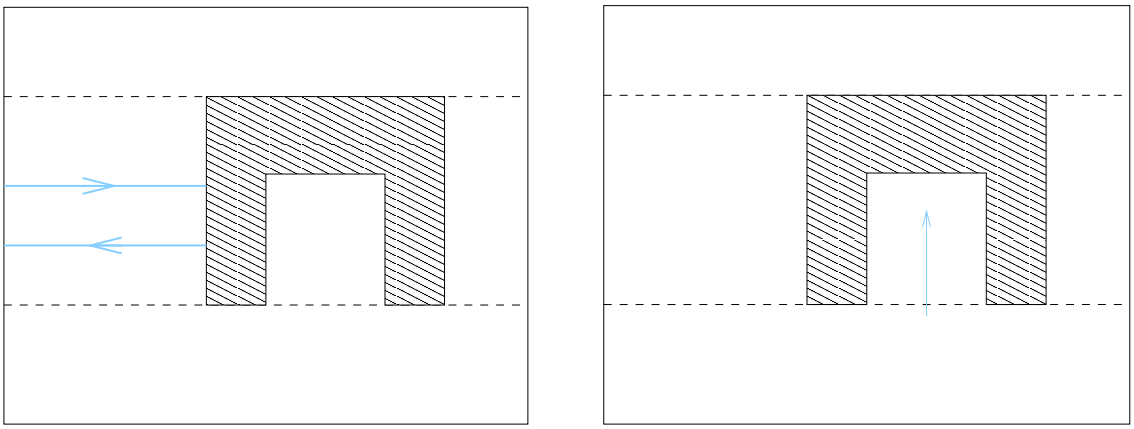


Figure 2.11: Mobile robots cannot move inside obstacle space. (a) Sweep line is limited to the current free space cell. (b) Some cells must be swept in the reverse direction.

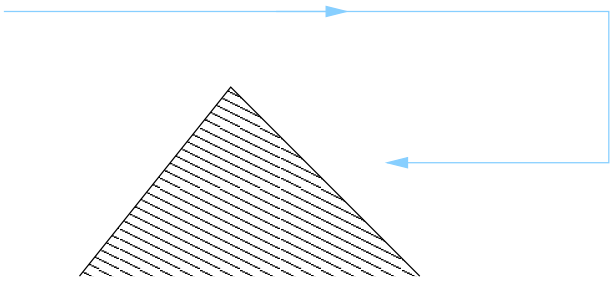


Figure 2.12: The vertex falls between two consecutive strips of a zigzag.

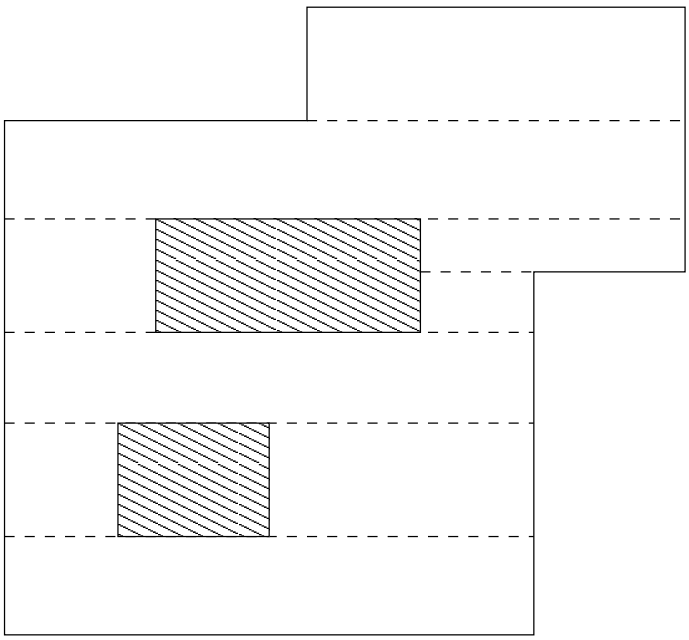


Figure 2.13: CC_R uses an exact cell decomposition for rectilinear environments. (Reproduced from page 16 of [22]).

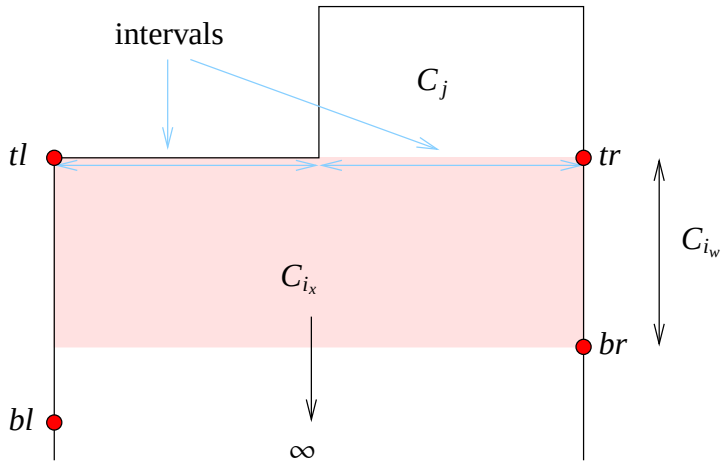


Figure 2.14: The partial decomposition in CC_R is stored as a list of cells $\mathbf{C} = \{C_0, \ldots, C_n\}$. This diagram shows the data structure associated with an individual cell C_i, which is a member of $\mathbf{C}$, ie $C_i \in \mathbf{C}$. Cell C_j is a neighbour of C_i and is shown here for clarity. (Adapted from page 17 of [22]).

The partial decomposition constructed is stored as a list of cells $\mathbf{C} = \{C_0, \ldots, C_n\}$. Figure 2.14 shows the data structures associated with each cell in CC_R. C_{i_x} is the maximum possible extent of cell C_i, and is represented simply by a rectangle. C_{i_n} is the cell's minimum known extent, and is given by four points – two on the cell's right boundary (*tr* and *br*), and two on the cell's left boundary (*tl* and *bl*). When the robot begins coverage with no knowledge of the environment, $\mathbf{C}$ will contain a single cell C_0 in which the minimum known extent C_{0_n} has zero size and the maximum possible extent C_{0_x} is infinite in all directions. As the robot covers this cell, its minimum known extent C_{0_n} will increase in size, while its maximum possible extent C_{0_x} will be limited with the discovery of each boundary. In addition to the minimum and maximum extents of the cell, the width of the portion of the cell that has been covered by the robot is also represented (C_{i_w}). Associated with each of the edges of the cell is a linked list of intervals which explicitly denote the cell's neighbours at each point along the edge. Each interval is represented as a line segment together with a neighbour ID. A cell is complete when its edges are at known locations, it has been covered from side to side, and all sides have been completely explored. In addition to the cell decomposition $\mathbf{C}$, CC_R maintains a list $H = \{H_0, \ldots, H_m\}$ of placeholders. A placeholder is an element of the boundary of any cell in $\mathbf{C}$ that is not a boundary of the environment, and thus are entrances to to free space cells. Coverage of an environment is complete when no placeholders remain.

Normally, the robot follows the U-shaped pattern in Figure 2.15 to cover individual cells. Segments α and δ are sliding movements against the side boundaries of the cell. An event (criticality) occurs whenever the robot is prevented from successfully executing the U-shaped coverage pattern. Also, all the events defined in CC_R can be detected without the use of any range sensing. Figure 2.16 shows the five events defined. In Figure 2.16(a), the robot's path is interrupted

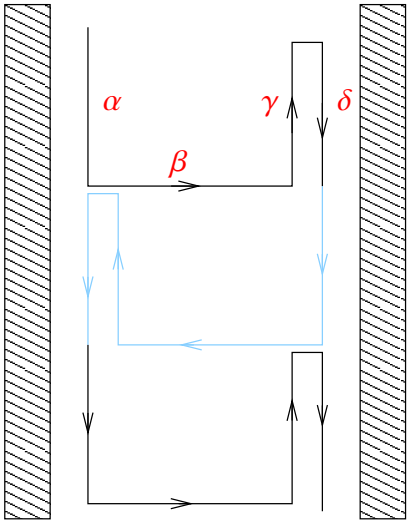


Figure 2.15: U-shaped coverage pattern used in CC_R and Morse decomposition. The pattern consists of four segments, α to δ. Distances between consecutive β segments should be small enough that no area is left uncovered when a robot follows this pattern.

by an obstacle while executing the α segment of the U-shaped pattern. In Figure 2.16(b), the side boundary the robot follows in the α segment disappears. In Figure 2.16(c), the robot's path in β is obstructed by an obstacle. Figure 2.16(d) shows an unexpected non-collision in segment β, where the robot does not encounter the side boundary as expected. In Figure 2.16(e), the robot loses contact with the side boundary. This is distinct from the situation in Figure 2.16(b) in that during the α segment the robot is outside the covered portion of the cell, while during segment δ it is not.

When the robot encounters any of the five events, it attempts to fully explore the new cell boundary. The maneuvers used depend on the event. The intervals associated with the current cell are updated. After the cell boundary is completely explored, the algorithm searches the list of placeholders H for any uncovered cells. Travelling between cells is done by moving into each cell in between, and moving first in one direction, for example x, then the other direction, y, in order to reach the next cell.

Unlike the zigzag (Figure 2.17), the U-shaped pattern contains retracing. This retracing is added to include wall following on both side boundaries. This is because a contact sensing robot cannot detect obstacles except when wall following. Therefore, if the robot is following a zigzag pattern and an opening occurs as shown in Figure 2.18, the robot will miss it.

Acar *et. al.* introduced Morse decomposition [8, 9] for range sensing robots covering unknown environments. Cell boundaries in Morse decomposition are critical points of Morse functions. The decomposition is based on a roadmap method originally proposed by Canny [26, 27]. Given a real-valued function $h : \mathbb{R}^m \rightarrow \mathbb{R}$, its differential at $p \in \mathbb{R}^m$ is $dh_p = [\frac{\partial h}{\partial x_1}(p) \cdots \frac{\partial h}{\partial x_m}(p)]$. A point $p \in \mathbb{R}^m$ is a critical point of a Morse function if $\frac{\partial h}{\partial x_1}(p) = \cdots = \frac{\partial h}{\partial x_m}(p) = 0$ and its Hessian

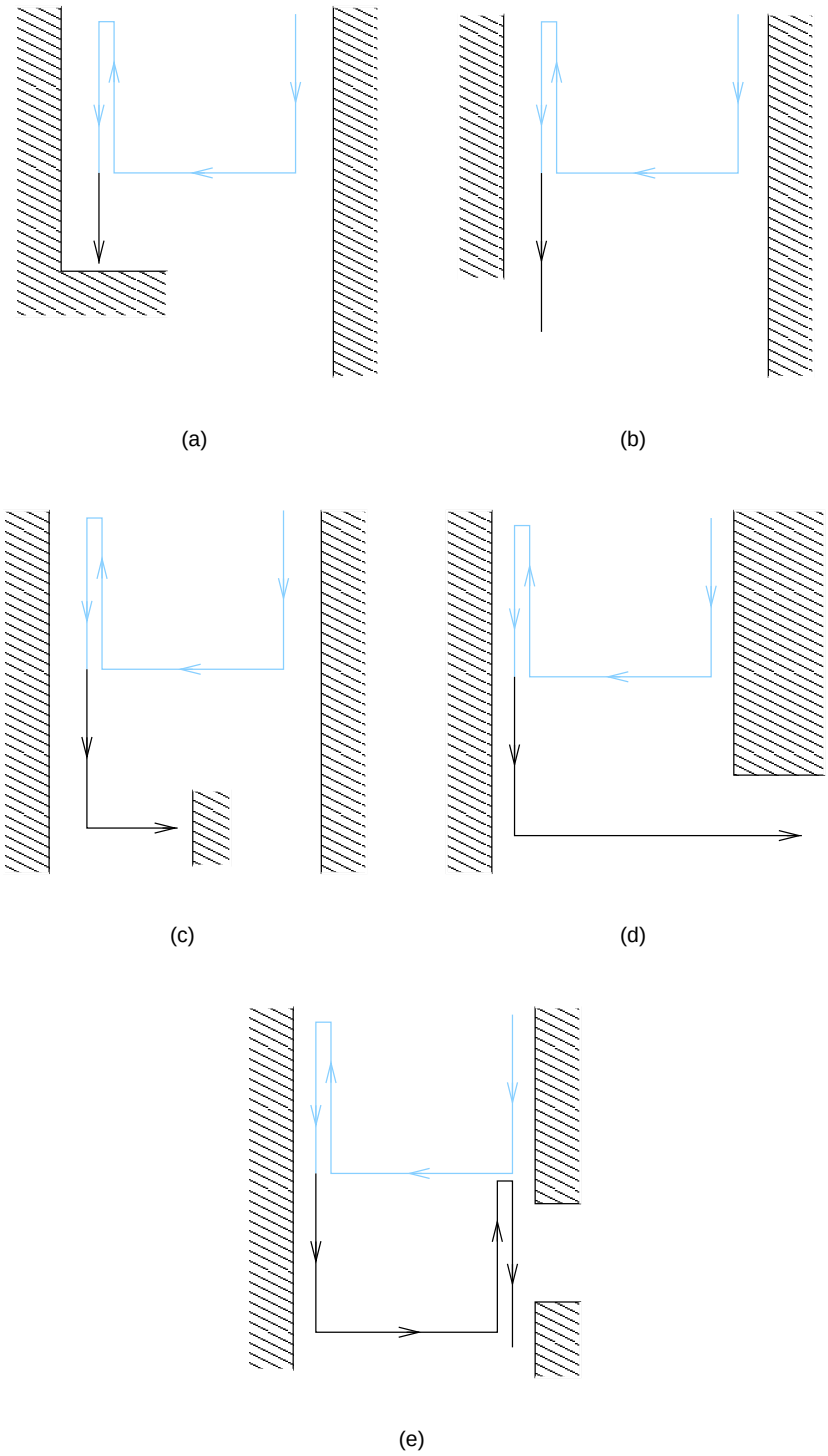


Figure 2.16: Events that occur at cell boundaries in CC_R.

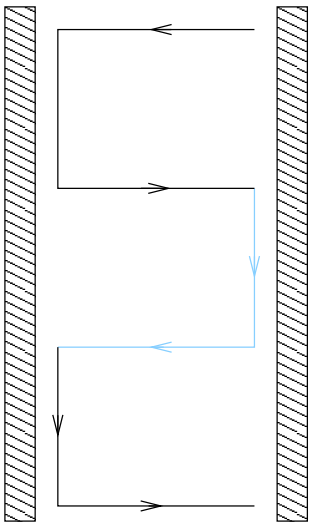


Figure 2.17: Compared to the pattern in Figure 2.15, a zigzag includes wall following on only one side boundary.

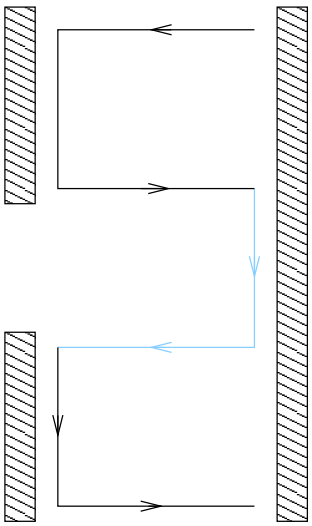


Figure 2.18: Contact sensing robots (as used in CC_R) will miss an opening in the side boundary unless it is wall following.

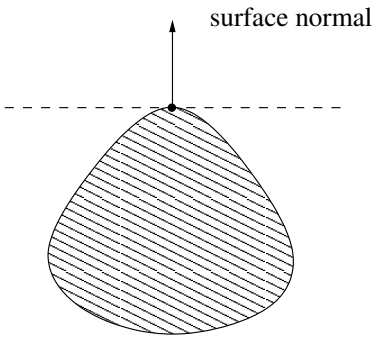


Figure 2.19: A cell boundary in Morse decomposition occurs at this position because the surface normal of the obstacle is perpendicular to the sweep line.

$(\frac{\partial^2 h}{\partial x_i \partial x_j}(p))$ is non-singular. To put it more simply, a critical point occurs when the sweep line encounters an obstacle whose surface normal is perpendicular to the sweep line. This definition of criticality for cell decomposition is explained graphically in Figure 2.19. Figure 2.20 shows an example of Morse decomposition.

By using surface normals as criticality, the environment is limited to differentiable functions only. Therefore, polygons cannot be handled as their vertices are non-smooth boundary points. To overcome this limitation in Canny's roadmap, Acar *et. al.* use Clarke's generalised gradients [33, 67] to calculate surface normals at these non-smooth boundaries [9]. The generalised gradient of a point x is the set of vectors within the convex hull of the surface normals of the adjacent smooth surfaces around x (see Figure 2.21). The generalised gradient can be used on any point x that is not differentiable, given that the function is Lipschitz around x. A function is locally Lipschitz for a bounded subset B if there exists a constant K such that

$$|f(x_1) - f(x_2)| \leq K\,|x_1 - x_2|$$

for all points x_1 and x_2 of B. However, since any function with a discontinuity is not Lipschitz around the discontinuity, rectilinear environments such as those used in CC_R are not covered by Morse decomposition.

Critical points in Morse decomposition can be detected using omni-directional range sensors. Given a robot is at point x, then the closest point on the surface of obstacle C_i to point x is c_0

$$c_0 = \arg\min_{c \in C_i} ||x - c||$$

Now, let $d(x)$ be the shortest distance between point x and the obstacle C_i. Then the gradient[6] can be determined by

$$\nabla d(x) = \frac{x - c_0}{||x - c_0||}$$

[6]By definition a gradient is a unit normal vector to a surface at a point [63].

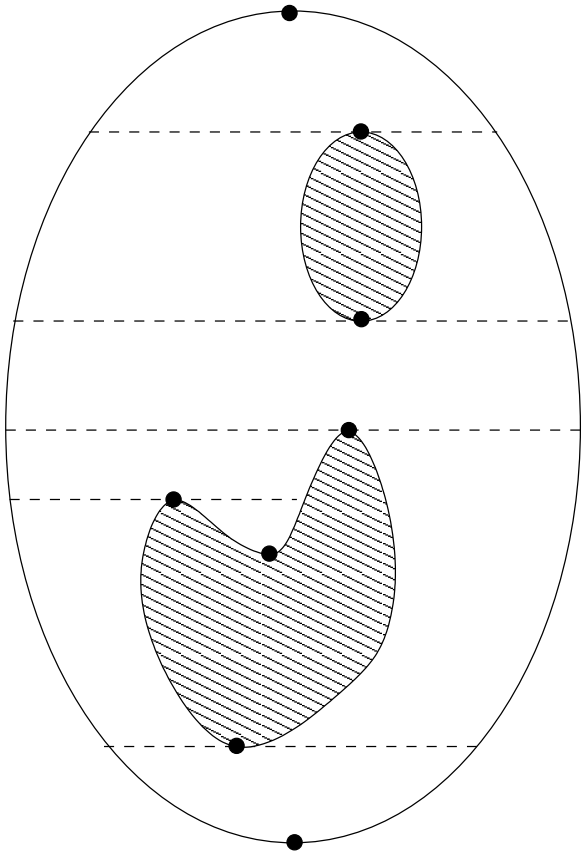


Figure 2.20: Criticality of Morse decomposition occurs at the positions of the black dots in the diagram. The dotted lines are the cell boundaries. Three of the critical points have no cell boundaries drawn through because free space is of zero width at those positions.

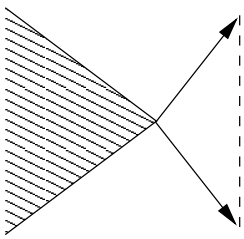


Figure 2.21: The generalised gradient is the convex hull of the set of gradients around the non-smooth boundary point.

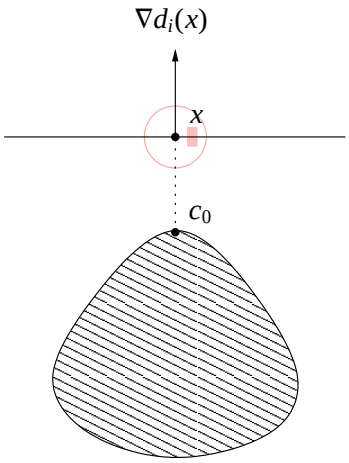


Figure 2.22: Detecting critical points in Morse decomposition.

This equation can be explained as follows. As c_0 is a point on the surface of the obstacle C_i, then $x - c_0$ is a vector which points from c_0 towards x. However, because c_0 is the closest point on the surface from x, it is thus normal to the obstacle surface. Division by $\|x - c_0\|$ turns the result into a unit vector.

The robot has detected a critical point if the gradient $\nabla d_i(x)$ is perpendicular to the sweep line. Figure 2.22 explains how critical points can be detected by robots equipped with omni-directional range sensors.

The Morse decomposition is stored as a Reeb graph[7], with the critical points as nodes. Figure 2.23 shows the graph corresponding to the decomposition in Figure 2.20. Note that the edges in the graph represent the cells in the decomposition.

Similar to CC_R, Morse decomposition has an associated algorithm for creating the decomposition online. It also employs the U-shaped coverage pattern in Figure 2.15. The wall following offered by the U pattern is needed because critical points occurring on the side boundary, such as those in Figure 2.24, cannot be detected even with unlimited range sensors except when wall following [8]. This is because the robot can only detect critical points of Morse functions if the critical point is closest to the robot compared to all other points on the obstacle surface.

An event occurs whenever the robot encounters a critical point while following the U-shaped pattern. Figure 2.25 shows the two events defined in Morse decomposition. In Figure 2.25(a), the robot is following the side boundary of the current cell when it encounters a critical point. The next lap position is moved to where the critical point is. In Figure 2.25(b), the robot encounters a critical point while moving along the length of the U-shaped pattern.

Information about uncovered cells is associated with nodes in the Reeb graph (ie the critical points). When the robot finishes executing a U pattern which is interrupted by critical points,

[7] A Reeb graph is a topological graph of a Morse function

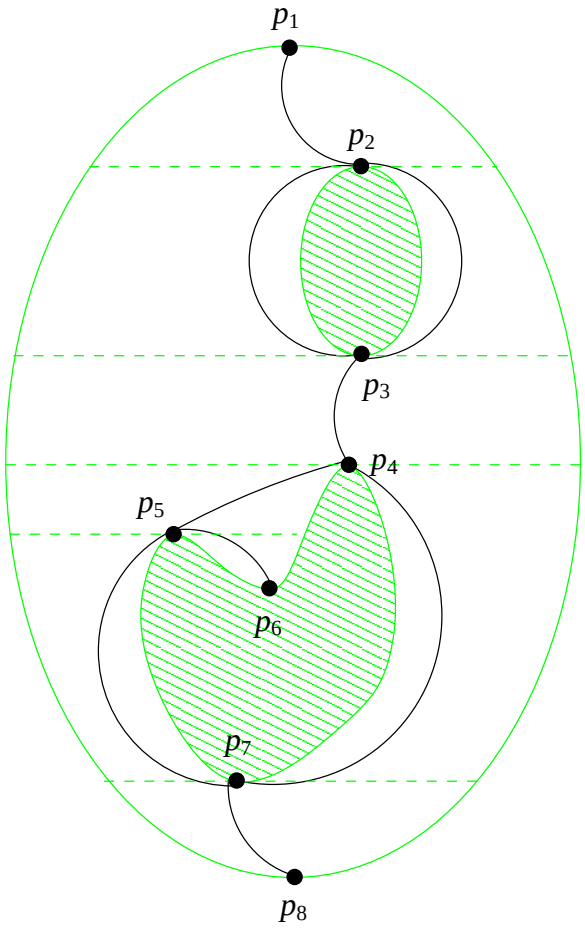


Figure 2.23: Reeb graph for the Morse decomposition in Figure 2.20.

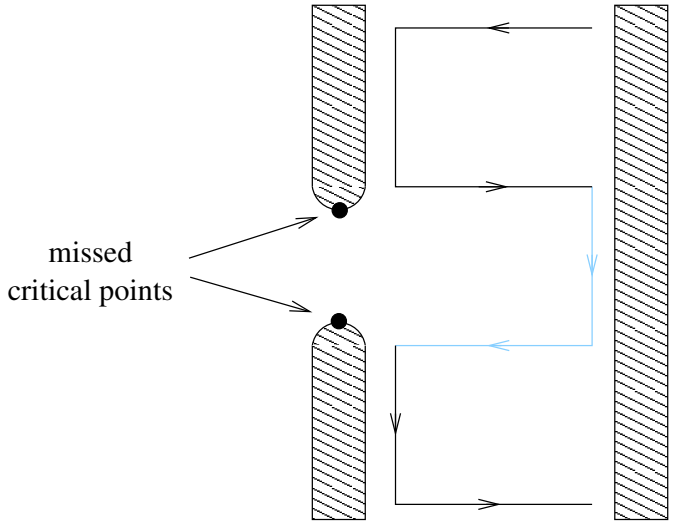
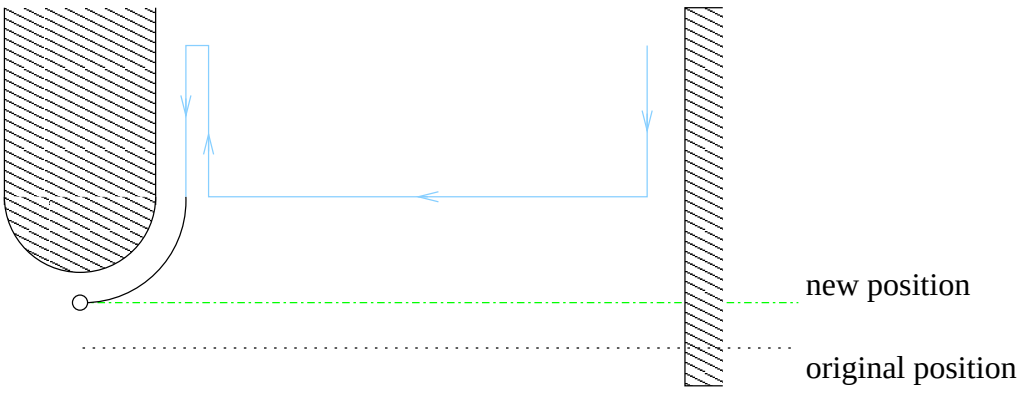
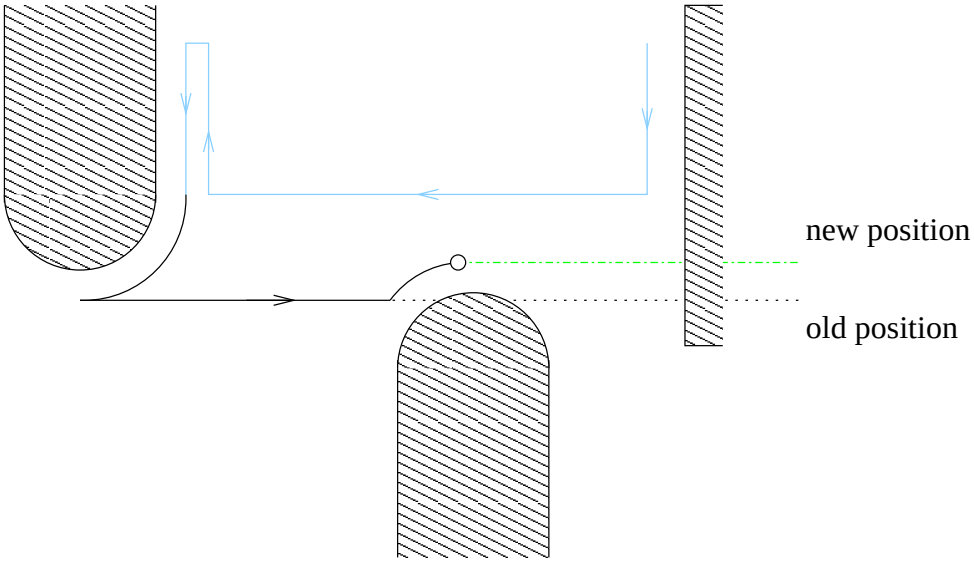


Figure 2.24: With Morse decomposition, a range sensing robot can only detect cell boundaries on its side. As a result, it will miss the critical points in this figure because there is no wall following on that portion of the side wall.



(a)



(b)

Figure 2.25: Events in Morse decomposition: (a) when wall following the side boundary, (b) when traversing the length of the coverage pattern. The two dotted lines in the diagrams show how the next strip position is changed.

it first looks for uncovered cells at the last encountered critical point. If the critical point is associated with two uncovered cells (eg in Figure 2.25(b)), the robot picks one of the cells associated as the next cell to cover. If there are no uncovered cells associated with the last encountered critical point, a depth-first search is performed on the Reeb graph. To travel to the selected uncovered cell, the robot follows the Reeb graph and plans a path that passes through cells and critical points. The environment is fully covered when no uncovered cells are associated with any of the nodes in the Reeb graph.

Although the starting point of this thesis is to investigate coverage with landmarks in topological maps (Section 2.4), the method proposed ultimately creates a cell decomposition similar to the split and merge concept in boustrophedon decomposition. However, unlike boustrophedon decomposition, this thesis deals with coverage of unknown environments. CC_R is especially designed for robots with no range sensing ability. Similar to this thesis, Morse decomposition is for range sensing robots working in more general unknown environments. Compared with Morse decomposition, the method proposed in this thesis can handle a larger variety of environments (rectilinear, polygonal and non-polygonal). Moreover, wall following on both side boundaries is not needed due to the use of more general landmarks as criticalities, or events, of the decomposition. Since retracing is eliminated from the coverage pattern, the proposed algorithm generates shorter coverage paths.

Boustrophendon decomposition was first published in a conference in 1997 [32], and later as a journal paper in 2000 [30]. CC_R was first published in 1999 [23]. Morse decomposition was first published in 2000 [7, 10], and thus was developed in parallel with the work in this thesis, which was also first published in 2000 [100, 101].

2.3 Grid map

In grid maps, environments are decomposed into a collection of uniform grid cells. This uniform grid includes both free space and obstacles. Each cell contains a value stating whether an obstacle is present. The value can either be binary or a probability [37]. Figure 2.26 shows an example of a grid map.

The major advantage of this type of map is the ease in creating one. It is essentially an array containing occupancy information for each cell of the map. However, grid maps require accurate localisation to create and maintain a coherent map [28, 95]. They also suffer from exponential growth of memory usage because the resolution does not depend on the complexity of the environment [94]. Also, they do not permit efficient planning through the use of standard graph searches [94].

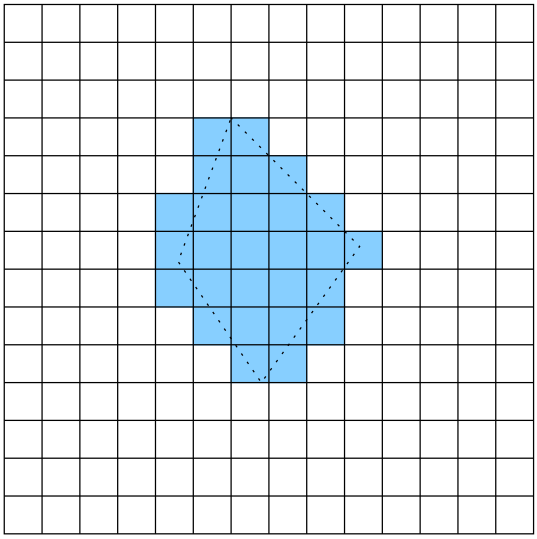


Figure 2.26: A grid map. Grid cells with obstacle present are shaded.

Grid based maps are the most widely used spatial representation for coverage algorithms. This is due to the simplicity of marking covered areas in a grid map.

Zelinsky *et. al.* proposed an offline coverage algorithm [108] based on the distance transform of a known grid map [58]. Figure 2.27 shows the distance transform of a simple environment with one obstacle. Here, S represents the initial (or starting) position of the robot, and G is the desired finishing position. The distance transform thus represents a wavefront that propagates from the goal cell G to the initial cell S. The algorithm for calculating the distance transform is shown in Algorithm 2.2. Lines 1 to 3 shows the initialisation needed before the execution of the main loop in lines 4 to 11. In line 1, the distance transforms (DT) of all cells in the grid are set to -1, which mark the cells as unprocessed. Execution starts with the goal cell G, which has a distance transform of 0 (lines 2 and 3). In each iteration of the main loop, unmarked cells who are neighbours of marked cells are assigned a DT one higher than their neighbours'. This continues until all cells in the grid map are marked.

Once the distance transform for the environment is calculated, the coverage path can then be formed by selecting the neighbouring cell with the highest DT and is unvisited, starting from the initial cell S. If two or more unvisited neighbours have the same transform, one of them is selected randomly. Figure 2.28 shows the coverage path generated for the example in Figure 2.27. The algorithm for generating the coverage path is explained in Algorithm 2.3.

Unlike other coverage algorithms, this distance transform based method requires the selection of a goal location.

Ulrich *et. al.* [97] also used a grid map for their online coverage algorithm. The algorithm starts with an exploration of the boundary of the environment. Afterwards, the robot moves in a straight line until it reaches an obstacle in front. When it is stopped, a new direction

13	12	11	10	9	8	7	7	7	7	7	7	7	7
13	12	11	10	9	8	7	6	6	6	6	6	6	6
S	12	11	10	9	8	7	6	5	5	5	5	5	5
									4	4	4	4	4
9	8	7	6	5	4	3	3	3	3	3	3	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	1	G	1	2	3	4
9	8	7	6	5	4	3	2	1	1	1	2	3	4
9	8	7	6	5	4	3	2	2	2	2	2	3	4

Figure 2.27: Distance transform for the selection start position (S) and goal position (G). (Reproduced from [108]).

Algorithm 2.2 Distance Transform for a Grid Map

Require: goal cell G

```
1:  DT(all cells) ← −1
2:  DT(G) ← 0
3:  n ← 0
4:  while there exists a cell c where DT(c) = -1 do
5:      for all x such that DT(x) = n do
6:          if y is a 8-neighbour or x and DT(y) = -1 then
7:              DT(y) ← n + 1
8:          end if
9:      end for
10:     n ← n + 1
11: end while
```

Algorithm 2.3 Coverage Path Planning using Distance Transform

```
c ← start cell
visited(all cells) ← false
repeat
    visited(c) ← true
    c ← neighbouring cell with highest DT
until c = goal cell G
```

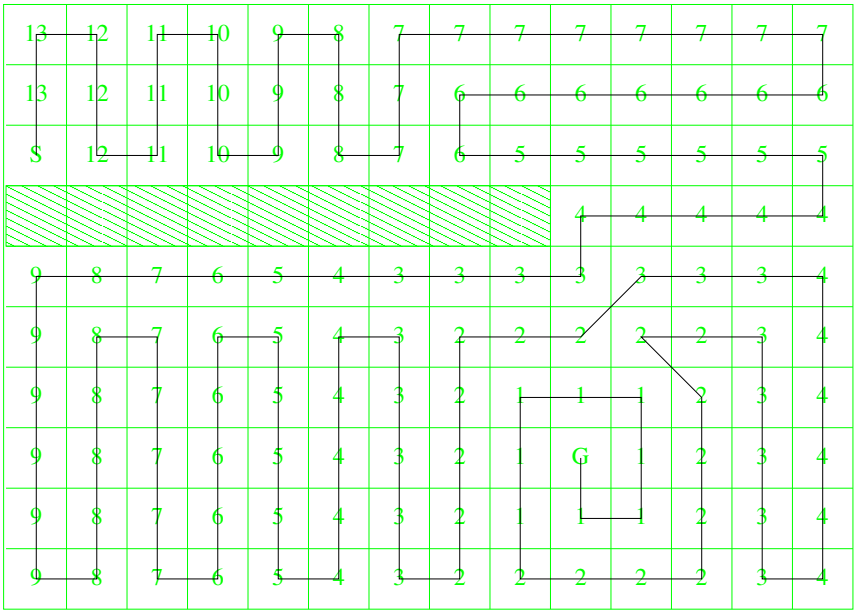


Figure 2.28: Coverage path generated from the distance transform. (Reproduced from [108]).

of travel is chosen. One of the criteria for the new direction is a high number of uncovered grid cells. The algorithm also attempts to generate a path that ends successively with mutually perpendicular walls (see Figure 2.29). This is done so the robot can alternately re-calibrate the *x* and *y* coordinates of its odometry estimation, with values obtained from the initial exploration phase. Since only a partial map is available, it may not be possible for the robot to reach the target wall in the chosen direction due to the presence of unknown obstacles. In this case, the robot updates the grid map and then selects a new direction of travel again. The path planning used in this work is a naïve approach and results in highly redundant paths, as can be seen in Figure 2.29.

Unlike the almost random approach taken by Ulrich *et. al.*, Gabriely and Rimon tackle the problem of coverage path planning on a partial grid map with a systematic spiral path. This is achieved by following a spanning tree of the partial grid map [45]. Two different sizes of grid cells are used. The smaller grid cell is the same size as the robot. Four of these smaller grid cells then form a mega cell. These concepts are shown in Figure 2.30. The details of this spanning tree approach is shown in Algorithm 2.4. The two parameters to the function STC are the current cell *x* and its parent cell *w*. The algorithm is started by executing STC(NULL, start cell). NULL is used for the parent cell because the start cell has no parent. A mega cell is *old* (line 1), if at least one of its four subcells is covered; otherwise, it is *new* (line 2). At each mega cell, the robot picks a new direction of travel by selecting the first uncovered free space mega cell in an anti-clockwise direction (line 3). A spanning tree edge is also grown from the current mega cell to the new one (line 4). The algorithm is recursive (line 6), and the recursion is stopped only when the current cell *x* has no *new* neighbours. This recursion moves

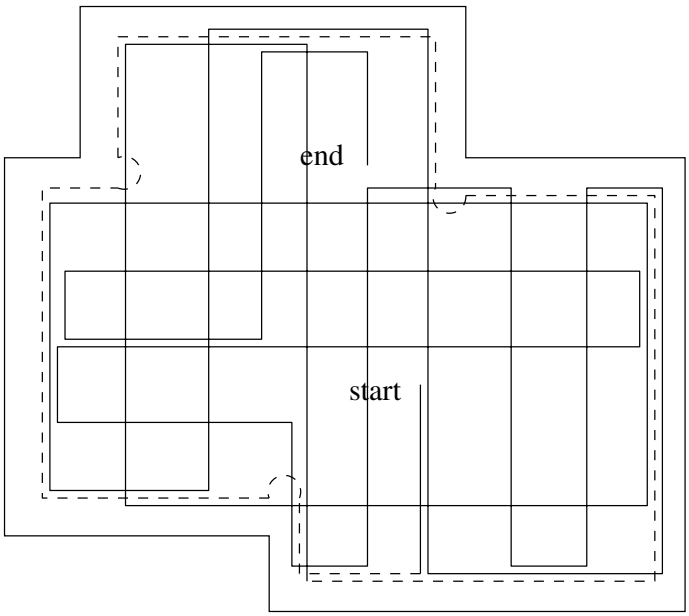


Figure 2.29: The robot path in the work of Ulrich *et. al.* is highly inefficient. The dotted line represents the path taken by the robot during boundary exploration, while the solid line represents the path taken during coverage. (Reproduced from [97]).

the robot along one side of the spanning tree (line 5) until it reaches the end of the tree (line 2), at which point it turns around to move along the other side of the tree (line 8). Figure 2.31 shows the spanning tree and the coverage path for the environment in Figure 2.30. Notice that when coverage is completed, the robot returns to the same mega cell as the initial location.

Algorithm 2.4 Spiral spanning tree coverage STC(w, x)

1: Mark the current cell x as *old*
2: **while** x has a *new* obstacle-free 4-neighbour cell **do**
3: Scan for the first new neighbour of x in anti-clockwise order, starting with the parent cell w. Call this neighbour y.
4: Construct a spanning-tree edge from x to y.
5: Move to a subcell of y by following the right-side of the spanning tree edges.
6: Execute STC(x, y).
7: **end while**
8: **if** $x \neq$ start cell **then**
9: move back from x to a subcell of w along the right-side of the spanning tree edges.
10: **end if**

Normally, cells in a grid map are square in shape and the same size as the robot. Oh *et. al.* proposed the use of a grid with triangular cells in their coverage algorithm instead [80] (see Figure 2.32). The rationale behind using a triangular grid is that it has a higher resolution compared to a rectangular one with similar sized cells. The use of alternative tiling arrangements to increase resolution is well known in image processing [74]. Oh *et. al.* showed that this increase in map resolution enables the robot to plan shorter coverage paths. However, the resolution

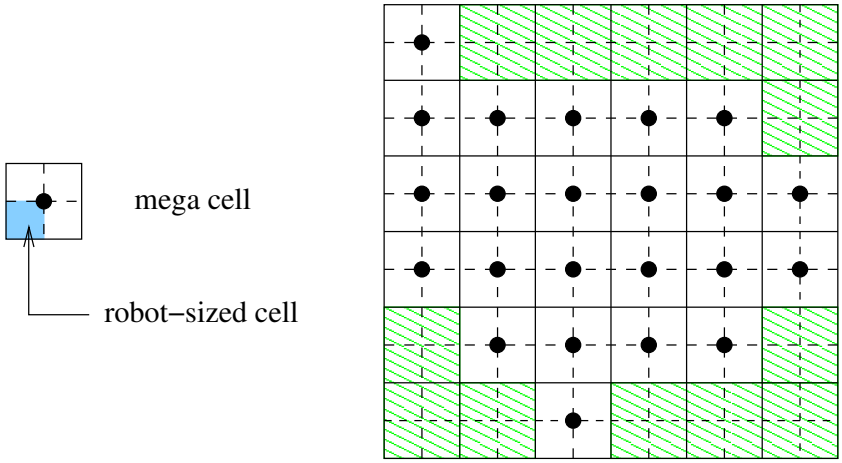


Figure 2.30: Gabriely and Rimon use mega cells that are formed from four grid cells in their coverage path planning algorithm.

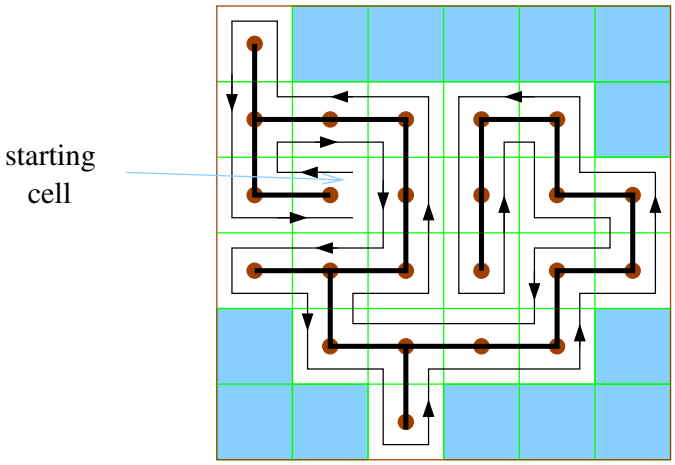


Figure 2.31: In Gabriely and Rimon, the robot circumnavigates the spanning tree formed from free space mega cells.

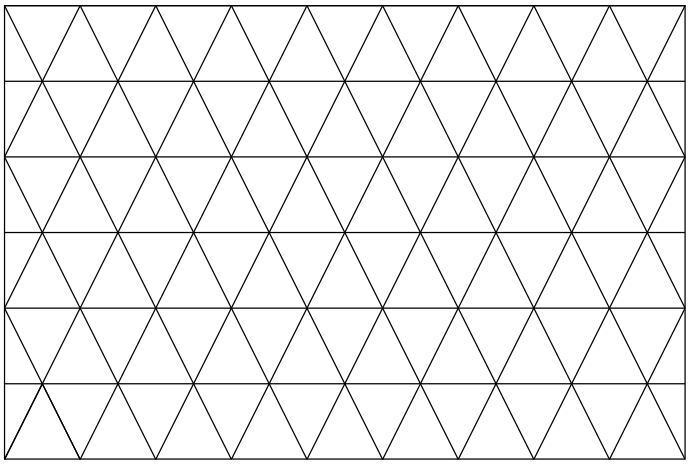


Figure 2.32: Oh *et. al.* used a triangular grid in their coverage algorithm.

of the grid can also be increased by using smaller square cells. Unlike in image processing, there is no need for ultra high resolution in robot path planning. This is because mobile robots are incapable of making very fine movement adjustments. Therefore I believe the extra effort needed in implementing an alternative grid arrangement is not worthwhile.

2.4 Topological map

Traditional approaches to robot mapping (such as the ones discussed previously in this chapter) are based on accurate metrical descriptions of environments. As a result, global metric consistency must be maintained to form a coherent and useful representation [59, 60, 64]. Therefore, these metric maps are very vulnerable to inaccuracies in sensors, actuators and odometry.

On the other hand, humans perform well in spatial reasoning tasks despite a lack of precise localisation. For example, I can easily plan a path that will take me from the engineering school to other buildings on campus. And I can do this without knowing the exact distances between the buildings or their precise locations. This is because humans approach the mapping problem in a more qualitative way by using topological relationships between landmarks. For example, we say that the engineering school is opposite the recreation centre and next door to the architecture school. Biologists have also found that animals store spatial information topologically as well [77, 82].

Kuipers and Byun introduce the topological map for storing robots' environments in this qualitative manner [64]. Figure 2.33 shows an example of their topological map. It consists of a set of nodes and a set of edges. The nodes are landmarks, or distinctive places, in the environment; the edges represent connectivities between the nodes. Kuipers and Byun use features like intersections and corners for their nodes. However, other features can be used as well. For example,

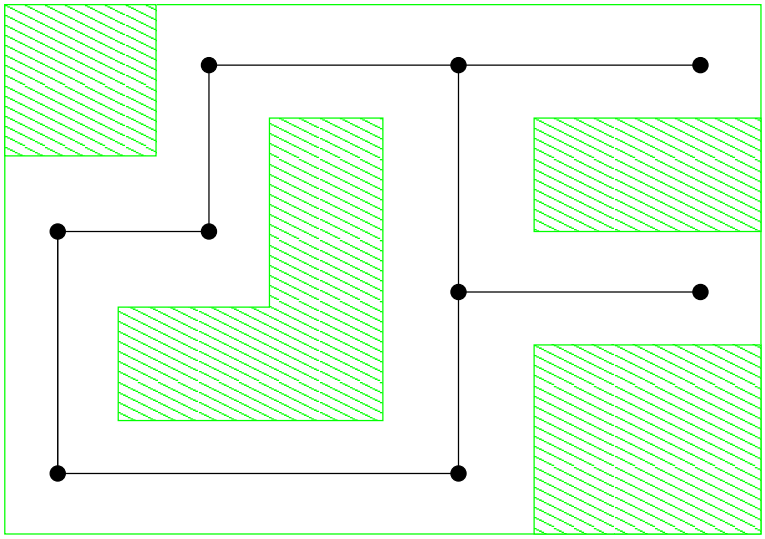


Figure 2.33: Kuipers and Byun use natural landmarks in an environment to create a topological map. The map attempts to capture the topological relationship of landmarks.

Mataric uses larger scaled features such as walls and corridors [72]. The use of graphs to show topological relationship is very common outside robotics. For example, most train and subway maps show stations as nodes, with edges indicating services between stations.

Topological maps are robust against sensor and odometry errors because only a global topological consistency, rather than a metric one, needs to be maintained [64, 94]. As an example, Zimmer successfully implemented a topological navigation and mapping system for a low budget platform with only light and touch sensors [111].

A problem with this qualitative approach is the very low resolution of topological maps. To combine the best of both worlds, hybrid maps have been introduced to unite the preciseness of metric maps and robustness of topological maps. An example is the work of Tomatis *et. al.* that uses a global topological map to connect local metric ones [96]. The low resolution of topological maps is also the reason why it is difficult to use them for coverage path planning. A node in a topological map is a landmark and does not correspond to a precise position or area in space. This makes it rather difficult to mark covered regions.

2.5 Reactive robots

The classical AI approach to robot navigation is to break the problem up into functional modules such as sensory perception, environmental mapping, path planning and execution of those plans [18]. Brooks argued that this encapsulation of knowledge in symbolic representations makes the AI approach inflexible to deviations in the real world [20, 21]. Brooks introduces reactive robotics, where no explicit internal representation is built [14, 18]. Instead, actions of the

robot are simply reactions to its sensory inputs. Usually, the reactions are organised as a set of behaviours and arranged in layers [18]. Examples of successful implementation of behaviour-based architecture include a six-legged walking robot [19] and a swarm of robotic bulldozers [83].

Behaviour-based systems are designed to overcome inaccurate sensors and imperfect control through a tight coupling between sensors and actuators. As a result, it is extremely robust and simple to implement. However, the principle of avoiding an explicit representation of goals has also limited their achievements to very simple tasks [68]. To put it simply, a purely reactive robot lacks purpose and plan.

The commercial vacuuming robot Roomba uses a purely reactive architecture [2]. Roomba follows a spiral path when it is in open space and turns to a random direction when its path is stopped by obstacles. However, since it does not retain a notion of what area has been covered, the algorithm cannot guarantee completeness in a finite amount of time. And in most operations, time is never unlimited. By following a pseudo random path, a reactive coverage robot will take a long time, if ever, to achieve complete coverage.

2.6 Coverage with multiple robots

Some researchers approach the coverage problem with large teams of robots, using dynamic roadmaps to coordinate robots' behaviours over the desired region [16], or partitioning an environment dynamically without the need for global communication [57]. This thesis focuses on coverage with a single mobile robot, which is more appropriate for tasks such as vacuuming. A team of vacuuming robots would be impractical and uneconomic in a domestic environment for example, whereas a team of inexpensive robots may be appropriate for a foraging task in a large environment.

2.7 Coverage of 3-dimensional surfaces

Most work on robot coverage path planning makes the assumption that the environment can be modelled as a simple planar surface. This is a valid assumption for floors, windows, lawns, etc. However, some surfaces are 3-dimensional in nature, and 3-dimensional path planning is required instead. For example, an autonomous underwater vehicle covering the seabed [52] or a robot painting motor vehicle panels [15] would need to move over 3-dimensional surfaces. This thesis focuses only on coverage of simple planar surfaces.

2.8 Performance metrics

It is important to have a quantitative measure on how well an algorithm performs. Generally, the measurement of performance is well studied in the area of localisation [35, 78]. However, this is often neglected in discussions of research on robot navigation (including coverage navigation). Most commonly, results are presented qualitatively, showing pictures of the route taken by the robot from simulation [32, 80, 108], or from real robot experiments [8].

2.8.1 Simulation

Gabriely and Rimon used two different metrics for their simulated experiments [45]. Their testing environment was modelled as a uniform grid. The amount of coverage was measured as the ratio of the number of grid cells that were visited at least once over the total number of unoccupied grid cells. The other metric is the number of repeatedly covered grid cells. Repeated coverage is a measure of the efficiency of the algorithm. This is because it is undesirable to repeatedly cover any cell, and thus a good solution will minimise the amount of repeated coverage. A problem with using number of repeatedly covered cells as a metric is that a repeatedly covered cell maybe covered more than twice. Also, there is no comparison with the total number of grid cells to be covered. For example, 2 repeatedly covered cells out of 10 is very different from 2 out of 1000.

2.8.2 Real robots

Ulrich *et. al.* [97] covered the test area with sawdust and estimated the amount of sawdust left afterwards. The amount of sawdust left serves as an indication of how much of the total area is covered. However, this method requires the robot to be equipped with a dust buster. It is also error prone as estimating the amount of dust is not an easy task.

Butler introduces the coverage factor (cf) [22], which is defined as

$$cf = \frac{d \times w}{\text{Total area to be covered}} \tag{2.2}$$

where d is the total distance travelled and w is the width of the robot. However, using $d \times w$ is not a reliable way to estimate area covered by the robot, except in the rare case where the robot does not cross over its own path. If the robot revisits any previously covered area, then $d \times w$ gives no indication on either the total surface area covered, or the amount of re-coverage. Therefore, the coverage factor in (2.2) is a poor measure of performance.

2.8.3 Complexity of coverage navigation

Even with complete knowledge of the environment, it has been shown that finding the shortest coverage path is an NP-hard problem [13]. This is not hard to verify as computing a coverage path contains the travelling salesperson problem (TSP). In [13], Arkin *et. al.* showed that a TSP tour for a grid map with N cells within a simple rectilinear polygon[8] will be at most $\frac{6N-4}{5}$ in length. For polygons with holes (ie environments with obstacles), Arkin *et. al.* showed a bound of $1.325N$.

In unknown environments, there is no representation available to perform searches. The cost of "search" is thus measured in terms of the length of the path taken [56]. For example, if a robot is searching for a goal location, it must move around and explore the environment while looking for the goal. An efficient algorithm will thus attempt to minimise the length of this path.

Searching for a goal location in an unknown environment is therefore generally more expensive (in terms of path length) than the optimal path. This can be explained with the following example. Suppose a robot is facing a very long wall. There is a door along the wall to the other side. If the robot knows where the door is, it can move directly from its current location to the door. On the other hand, if the robot does not know the location of the door, it has to either explore one side completely first (depth first search); or both side alternately, each time increasing the exploration depths (breadth first search).

In [56], Icking *et. al.* proved an upper bound of $2N$ for finding a coverage path in an unknown grid map with N cells. This upper bound is provided by depth first search, which makes exactly $2(N-1) = 2N-2$ steps in any environment with N cells.

An algorithm for coverage of a graph $G(N, E)$ with depth first search is shown in Algorithm 2.5. The algorithm visits every node in the graph at least once before returning to the start node (lines 8-11). Line 13 directs the robot to retrace its footstep along the graph until it reaches either the start node, or a node that has a neighbouring unvisited node. For every iteration of the loop, the robot moves to one of its neighbouring nodes (line 17). During the execution of the `repeat..until` loop, all nodes other than the start node are visited once with the condition at line 8, and the second time with the retracing at line 13. Therefore, for a graph with N nodes, the total number of moves is $2(N-1)$.

The algorithm can be explained with the graphs in Figure 2.34. The node highlighted in each graph is the start node. For the example in Figure 2.34(a), the robot visits each of the four branches in turn. Each visit requires 4 moves to reach the furtherest node and back. Therefore, a total of 16 ($2 \times (9-1)$) moves is taken with the depth first search. For the example in

[8] A polygon P is simple if the only points of the plane belonging to two polygon edges of P are the polygon vertices of P.

Algorithm 2.5 Coverage with Depth First Search

Require: $G(N, E)$, start node

```
1:  for each node n in N do
2:      predecessor(n) ← −1
3:      visited(n) ← false
4:  end for
5:  current ← start node
6:  visited(current) ← true
7:  repeat
8:      if there exists a neighbour x of current where visited(x) = false then
9:          visited(x) ← true
10:         predecessor(x) ← current
11:         current ← x
12:     else
13:         current ← predecessor(current)
14:     end if
15:     move to current
16: until current = start node
```

Figure 2.34(b), the robot first visits the only node connected to the start node (1 move). It then visits the two nodes connected to this centre node in turn (4 moves). Finally, it returns to the start node (1 move). Therefore, a total of 6 ($2 \times (4 - 1)$) moves is taken.

2.9 Discussion

Most existing coverage planning algorithms favour metric based grid maps for spatial representation. However, purely metric based maps need accurate localisation to be useful as they require global metric consistency. On the other hand, the biologically inspired topological maps are robust against such errors due to their use of topological relationships. However, it is difficult to use topological maps for coverage navigation as the nodes and edges do not directly correspond to any specific area in space.

A solution to this problem of direct representation in topological maps is to use a hybrid metric topological hierarchy. For example, a cell decomposition can also be represented by an undirected graph. By using detectable landmarks in an environment as events of a cell decomposition, a topological map that embeds a cell decomposition can be built [102]. This is the approach taken in this thesis. A coverage algorithm that uses easily detectable landmarks to build a topological map, and implicitly, a cell decomposition, is developed in Chapters 3 and 4.

The two performance metrics proposed by Gabriely and Rimon evaluate the effectiveness (percentage coverage) and efficiency (number of repeatedly covered cells) of coverage experiments in simulation. The efficiency measure has the problem that it is not normalised (with the total

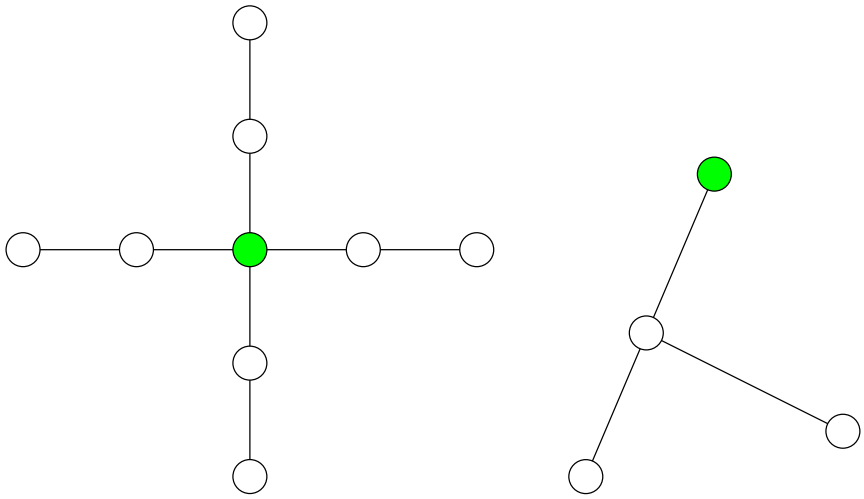


Figure 2.34: The coloured node is the start node of depth first search. A complete exploration of the graph in (a) requires 16 moves ($= 2 \times (9 - 1)$). A complete exploration of the graph in (b) requires 6 moves ($= 2 \times (4 - 1)$).

number of cells in the grid map). Moreover, the metrics are limited to simulation because real robots do not move in a uniform grid. For example, the movement of a real robot is continuous and therefore the robot is forced to "re-cover" surfaces it has already covered. Chapter 5 develops two new metrics for evaluating effectiveness and efficiency that are applicable to both simulations and real robot experiments.

2.10 Summary

This chapter reviewed existing coverage path planning methods. It includes coverage algorithms that use a variety of spatial representations for mapping, such as grid map and cell decomposition. It also discussed performance metrics used to evaluate results from robotic coverage experiments.

Newton's First Law of Graduation: A grad student in procrastination tends to stay in procrastination unless an external force is applied to it.

Jorge Cham, "Piled Higher and Deeper", www.phdcomics.com

3

Slice decomposition

This thesis introduces an online coverage algorithm that uses a hybrid topological/metric map for coverage path planning. The landmarks in the topological map are also criticalities of a cell decomposition, called *slice* decomposition [103]. Events in slice decomposition use and extend the concept of split and merge events first introduced by boustrophedon decomposition. As a result, cells in slice decomposition can be covered by a robot following a zigzag pattern. This topological coverage framework will be described in Chapters 3 and 4. Chapter 3 introduces the concepts, events and algorithms of slice decomposition; while Chapter 4 describes the online coverage algorithm that constructs the hybrid topological map/slice decomposition.

This chapter is organised as follows. Section 3.1 presents the first version of slice decomposition. It improves on boustrophedon decomposition in three ways. Firstly, it provides precise definitions for split and merge events in boustrophedon decomposition. Secondly, it adds two new events for handling concave obstacles. Thirdly, it presents a detailed algorithm for the decomposition algorithm. In Section 3.2, the second version of slice decomposition is presented. The first version of slice decomposition is modified to handle the situation where the sweep line is limited to free space regions only. This modification is necessary for online creation of slice decomposition in unknown environments by robots. This is because robots cannot perform a line sweep within obstacles of the environment. Then, in Section 3.3, the effect of step size on the overall decomposition formed is discussed. Finally Section 3.4 briefly discussed how slice

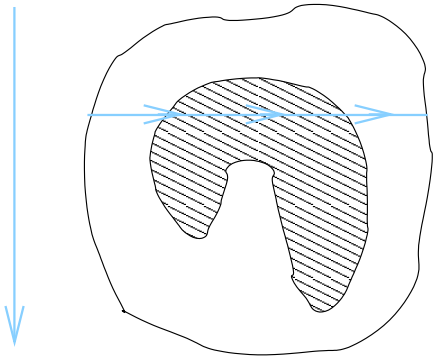


Figure 3.1: At any time (and sweep position), the sweep line acts as a ray travelling from left to right through the environment.

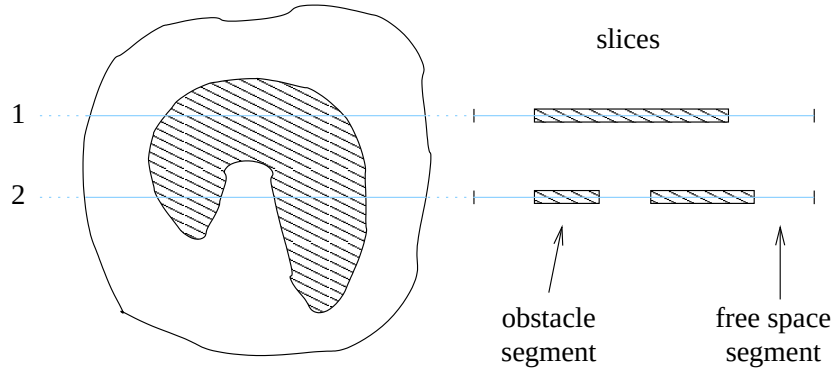


Figure 3.2: The arrangement of segments in slices made by the sweep line changes as it sweeps through the environment.

decomposition can be applied to coverage navigation with tethered robots.

3.1 Slice Decomposition I

Slice decomposition is formed by a line sweep process, where a cell boundary is created whenever a criticality is encountered by the sweep line [29]. Criticalities, or events, in slice decomposition are defined by the number of times the sweep line intersects with obstacle boundaries. Consider a sweep line that passes through an environment from top to bottom. At any time, the sweep line can be viewed as a ray that travels from the left boundary to the right boundary, crossing over a series of free space and obstacle regions depending on the topology of the environment. This is shown in Figure 3.1. At any time, the sweep line creates a *slice* of the environment, and the series of regions within the slice are called *segments*. Figure 3.2 shows two slices from two different sweep line positions.

The crossings test (or even-odd test) [49, 89] can be used to determine if a segment belongs to a free space region or an obstacle region. In computer graphics, the crossings test is used to

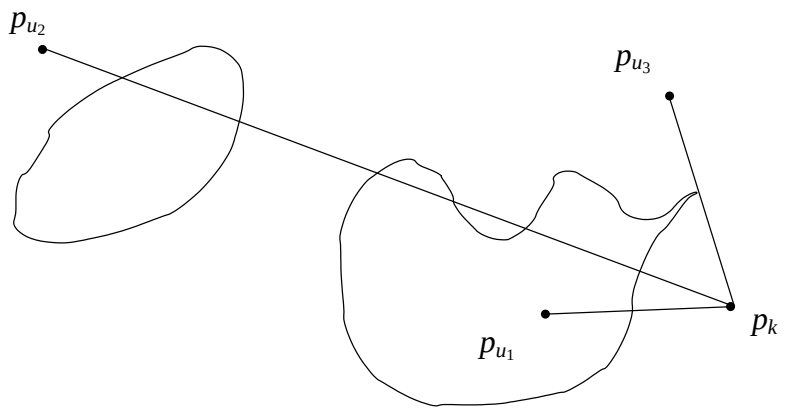


Figure 3.3: Crossings test for determining whether a point is within general polygons.

determine quickly whether a point p_u is inside a general polygon[1]. The test involves shooting a ray from the point p_u to a known point p_k and counting the number of crossings the ray makes. If the number of crossings is even, then point p_u is in the same type of region as p_k; if the number of crossings is odd, then point p_u is in an opposite type of region from p_k. This is illustrated in the example shown in Figure 3.3. Given a known point p_k outside any objects, we know that point p_{u_1} is inside an object because the line connecting points p_k and p_{u_1} intersects only once with an object boundary. On the other hand, the line connecting point p_k and point p_{u_2} intersects with objects 6 times, therefore point p_{u_2} must be outside of any objects.

The crossings test thus guarantees that region type changes within a slice occur only at object boundaries. This means that all spaces within a segment are always of the same type. Also, there is always an even number of crossings within a slice. This is because a slice starts from and finishes in the obstacle region outside the boundary of the environment.

In the case where the sweep line passes through a vertex, the vertex is classified as being infinitesimally away from the the sweep line [49]. This concept is explained in Figure 3.4. For example, consider the line connecting point p_{u_3} and point p_k. If the vertex is classified as being on the right of the line, then the line crosses the object twice (Figure 3.4(a)). If the vertex is classified as being on the left of the line, then the line does not cross the object at all (Figure 3.4(b)). Either way, the crossings test concludes that the point p_{u_3} is outside of any objects. Figure 3.5 shows an example of a sweep line passing through the vertex at the bottom of the obstacle. The vertex is classified to lie either infinitesimally above or below the sweep line.

3.1.1 Events

A cell boundary is created when a criticality occurs during the line sweep process. In slice decomposition I, this criticality occurs when the number of crossings the sweep line makes

[1] General polygons have no restrictions on the placement of vertices.

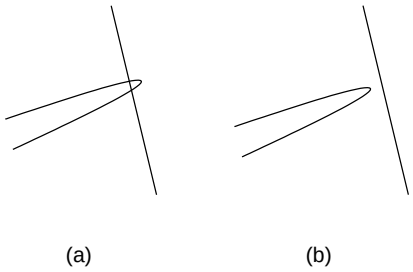


Figure 3.4: If the ray passes a vertex, it is always classified as being infinitesimally away from the ray. This diagram magnifies the vertex touched by the line connecting point p_{u_3} and point p_k in Figure 3.3. (a) Vertex classified to be on the right of the line. (b) On the left.

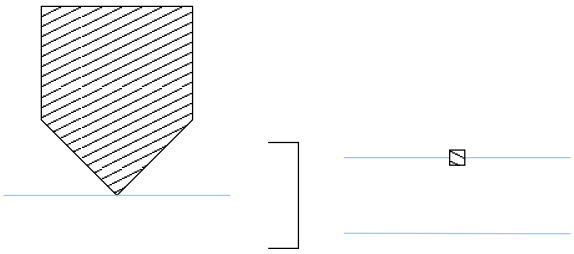


Figure 3.5: A vertex is classified as either infinitesimally above or below a sweep line that passes through it.

with the environment changes between two consecutive slices. The change in the number of crossings signifies an abrupt change in the topology among segments. Two slices S_a and S_b are consecutive if the two slices are from sweep line positions one time step apart. If the sweep line moves by a distance of Δx for each time step, and the slices S_a and S_b are from positions x_a and x_b respectively, then slice S_a and slice S_b are consecutive slices if and only if $| x_a - x_b | = \Delta x$.

There are four events in slice decomposition I:

1. *Obstacle emergence*: A free space segment in the previous slice is split into two by the emergence of a new obstacle segment in the current slice. This is equivalent to the split event in boustrophedon decomposition. The number of crossings made by the sweep line is increased by two. The current slice also contains one more free space segment and one more obstacle segment compared to the previous slice. An example of this is shown in Figure 3.6(a).

2. *Free space emergence*: An obstacle segment in the previous slice is split into two by the emergence of a new free space segment in the current slice. The number of crossings made by the sweep line is increased by two. The current slice also contains one more free space segment and one more obstacle segment compared to the previous slice. An example of this is shown in Figure 3.6(b).

3. *Obstacle disappearance*: An obstacle segment from the previous slice disappears in the

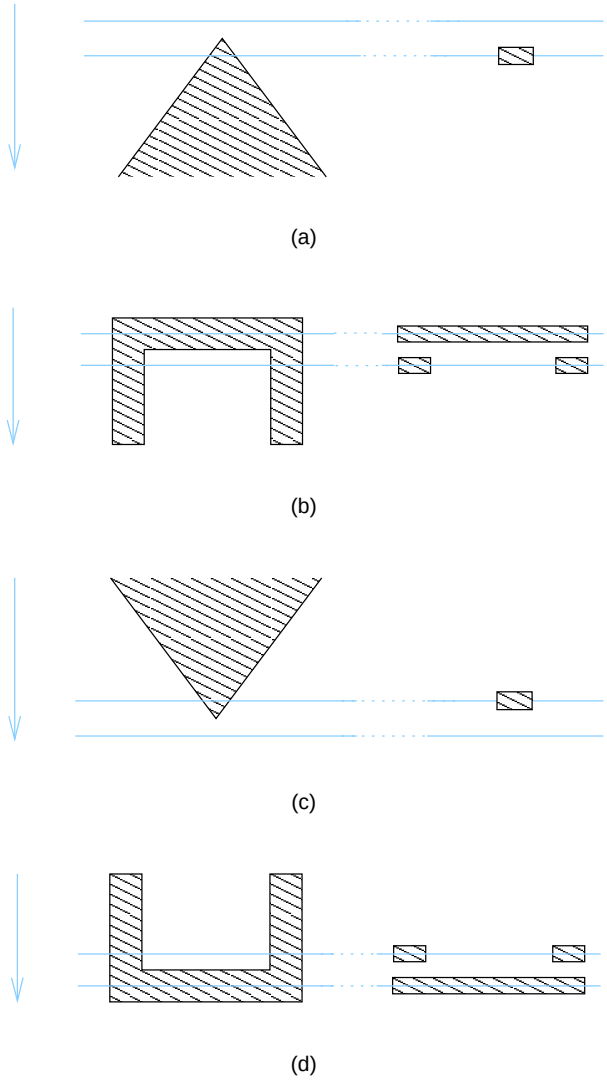


Figure 3.6: Events in Slice Decomposition I: (a) obstacle emergence, (b) free space emergence, (c) obstacle disappearance and (d) free space disappearance.

current slice. This is equivalent to the merge event in boustrophedon decomposition. The number of crossings made by the sweep line is decreased by two. The current slice also contains one less free space segment and one less obstacle segment compared to the previous slice. An example of this is shown in Figure 3.6(c).

4. *Free space disappearance*: A free space segment from the previous slice disappears in the current slice. The number of crossings made by the sweep line is decreased by two. The current slice also contains one less free space segment and one less obstacle segment compared to the previous slice. An example of this is shown in Figure 3.6(d).

Free space segment emergence and disappearance are new events in slice decomposition I. They are not covered by the split and merge concepts introduced by boustrophedon decomposition. These two new events are added to handle concave obstacles.

3.1.2 Algorithm

Slice decomposition is formed by maintaining a list D of active obstacle and free space cells with segments present on the slices created by the sweep line as it sweeps through the environment. The history of list D, ie all the cells that have appeared in D, forms the decomposition. The sweep stops to process and update list D whenever an event occurs in the current slice. The algorithm for slice decomposition is summarised in Algorithm 3.1.

Algorithm 3.1 Slice Decomposition I

```
 1: c ∈ {free space cell, obstacle cell}
 2: for all time t do
 3:    Move sweep line downwards by Δx
 4:    D_{t−1} = (..., c_{i−2}, c_{i−1}, c_i, c_{i+1}, c_{i+2}, ...)
 5:    for all segments in D_{t−1} do
 6:       if emergence inside c_i then
 7:          (c_i) ← (c_{e−1}, c_e, c_{e+1})
 8:          D_t = (..., c_{i−2}, c_{i−1}, c_{e−1}, c_e, c_{e+1}, c_{i+1}, c_{i+2}, ...)
 9:       end if
10:       if c_i disappears then
11:          (c_{i−1}, c_i, c_{i+1}) ← (c_d)
12:          D_t = (..., c_{i−2}, c_d, c_{i+2}, ...)
13:       end if
14:    end for
15: end for
```

The algorithm consists of two loops, one for moving the sweep line from top to bottom through the environment (line 2), the other for inspecting segments in the previous and the current slice for topology changes (line 5). It starts at line 1 by specifying that all cells are either free space cells or obstacle cells. Within the first loop, line 3 shows that the sweep line is moved by a very small distance Δx for each time step. Line 4 gives the format of the list D at the previous time step D_{t-1}. Lines 6 and 10 within the inner loop correspond the two main categories of events (emergence and disappearance). For obstacle or free space segment emergences (line 6), the segment that is split into two halves is replaced by three separate segments (line 7). The three segments belong to new cells and are therefore given new cell IDs, c_{e-1}, c_e, c_{e+1}. In other words, these new cell IDs identifying this slice contain a cell boundary. Line 8 shows the list D_t after the changes. The updates that occur for obstacle or free space segment disappearance are shown in lines 10 to 12. The cell that contains the disappeared segment, along with its two neighbours, are replaced in D by a single new cell (line 11). Line 12 shows the list D_t after the changes.

An example explaining the slice decomposition algorithm is shown in Figure 3.7. Here, f_n are free space cells and o_n are obstacle cells. Initially, the sweep line intersects only with the first free space cell f_1. Hence the decomposition is just that one space cell, $D_t = (f_1)$. At the first event, an obstacle segment emerges and the first cell f_1 is split. The decomposition D_t then

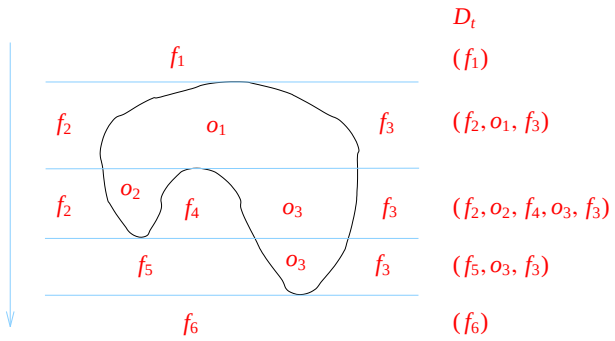


Figure 3.7: An example of slice decomposition I with an infinitesimally small step size Δx.

changes to contain three cells – a free space cell, an obstacle cell and another free space cell, $D_t = (f_2, o_1, f_3)$. Then the obstacle cell o_1 is split into two obstacle cells when a free space cell emerges. The decomposition D_t changes to contain five cells, $(f_2, o_2, f_4, o_3, f_3)$. Next D_t changes to three cells, (f_5, o_3, f_3), as the left side bulge is passed. Finally the decomposition D_t contains only one free space cell f_6 when the sweep line exits the obstacle.

3.2 Slice Decomposition II

Slice decomposition I keeps track of both free space and obstacle cells. However, robots cannot move inside obstacles. This means that the sweep line is limited to the cell that the robot is in, as shown in Figure 2.11(a) on Page 18. Sweeping in both forward and reverse directions is also needed because some free space cells cannot be accessed except from the bottom. For example, the cell under the U-shaped obstacle (Figure 2.11(b) on Page 18) can only be swept in the reverse direction. This is because the upper boundary of the cell is shared with an obstacle and the robot can therefore only enter from the bottom boundary.

3.2.1 Events

Events in slice decomposition I have to be updated for the situation where the sweep line is limited to the current free space region. For example, free space segment emergence (Figure 3.6(b)) cannot occur because it involves a sweep line moving from an obstacle to a free space cell. Figure 3.8 summarises the abrupt topology changes that are classified as events in slice decomposition II. Since line sweep is done in both forward and reverse directions, the arrows in Figure 3.8 indicate the current sweeping directions when the events are encountered. It does not mean that sweeping can only be done in the top to bottom direction.

There are five events defined in Slice Decomposition II:

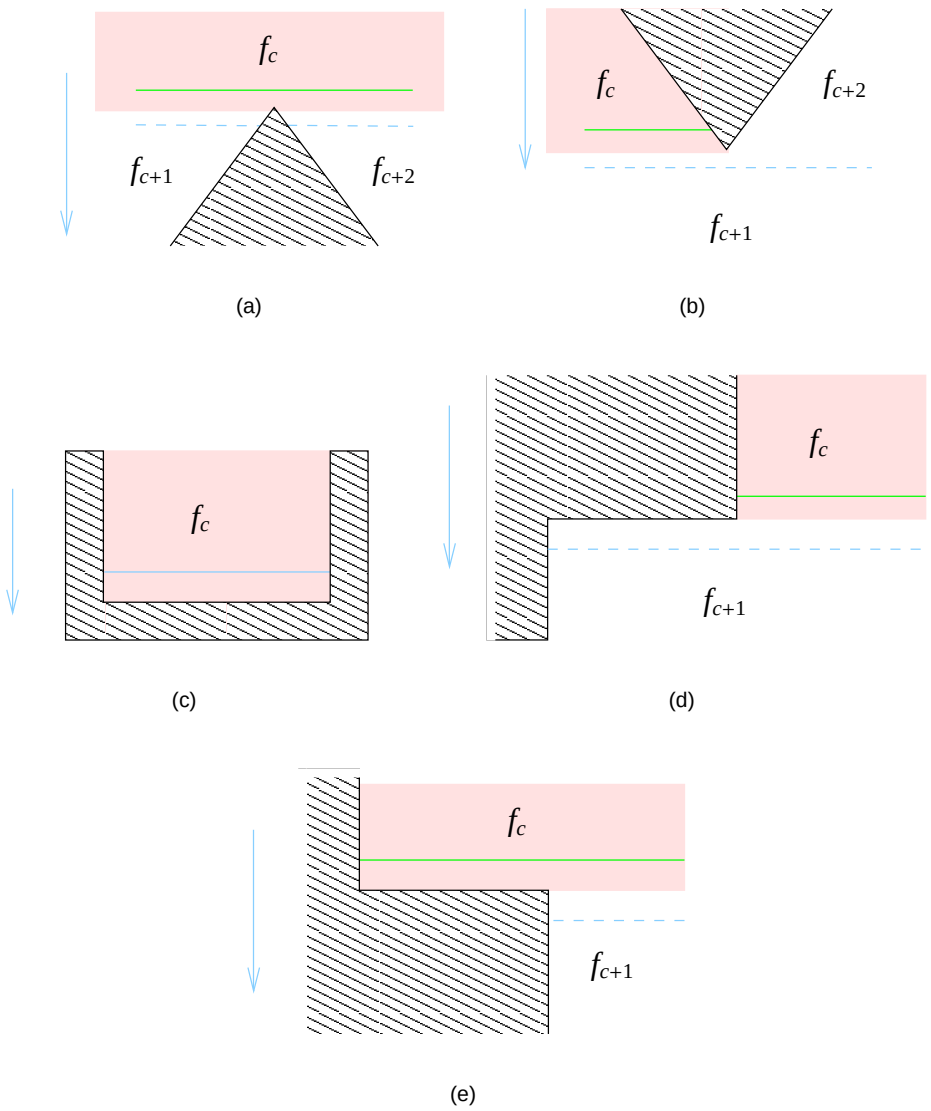


Figure 3.8: Events in Slice Decomposition II: (a) split, (b) merge, (a) end, (d) lengthen, and (e) shorten.

1. *Split*: Free space segment in the previous slice is split into two by the emergence of an obstacle, as in Figure 3.8(a). This is equivalent to obstacle segment emergence in offline slice decomposition.

2. *Merge*: Free space segment in the current slice neighbours free spaces other than the free space segment in the previous slice in the direction of the previous slice. This is equivalent to obstacle segment disappearance in normal slice decomposition. An example is shown in Figure 3.8(b).

3. *End*: The previous free space segment is the final one in the current cell. This is equivalent to free space segment disappearance in the normal version. An example is shown in Figure 3.8(c).

4. *Lengthen*: Free space segment in the current slice neighbours an obstacle segment in addition to the free space segment in the previous slice in the direction of the previous slice. Another way to view this situation is that the current slice is much longer than the previous slice. An example is shown in Figure 3.8(d).

5. *Shorten*: Free space segment in the previous slice neighbours an obstacle segment in addition to the free space segment in the current slice in the direction of the current slice. Another way to view this situation is that the current slice is much shorter than the previous slice. An example is shown in Figure 3.8(e).

Lengthen and *shorten* are new events and do not correspond to any events defined in slice decomposition I. These two events will affect how some environments are decomposed. Take the example in Figure 3.9. In slice decomposition II, it is decomposed into three cells; in slice decomposition I, the entire environment is "decompose" into one large cell. Boustrophedon decomposition will also classify the entire environment as one large cell. The two new events are added to allow for simpler detection of cell boundaries. This is because boundary walls and free standing obstacles may appear identical to a robot when they are first encountered (Figure 3.10). Cell boundary detection with range sensors will be discussed in more detail in Chapter 4.

3.2.2 Algorithm

Algorithm 3.2 explains how slice decomposition II is created. The algorithm maintains two lists, O (open) and F (finish). The open list O stores all free space cells that have been discovered, while the finish list F remembers all cells that are visited. The algorithm loops until the open list O becomes an empty set (line 3). This happens when all cells discovered have been visited. In line 4, a cell in the open list O is picked as the current cell f_c. This cell is then swept systematically from one cell boundary (line 5) to the other (line 7) until an event is encountered

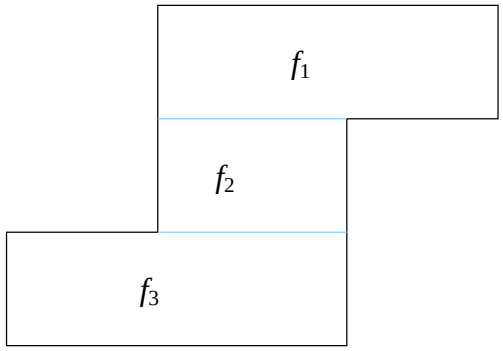


Figure 3.9: In slice decomposition II, this environment is decomposed into three different cells. In slice decomposition I, this same environment is classified as one cell only.

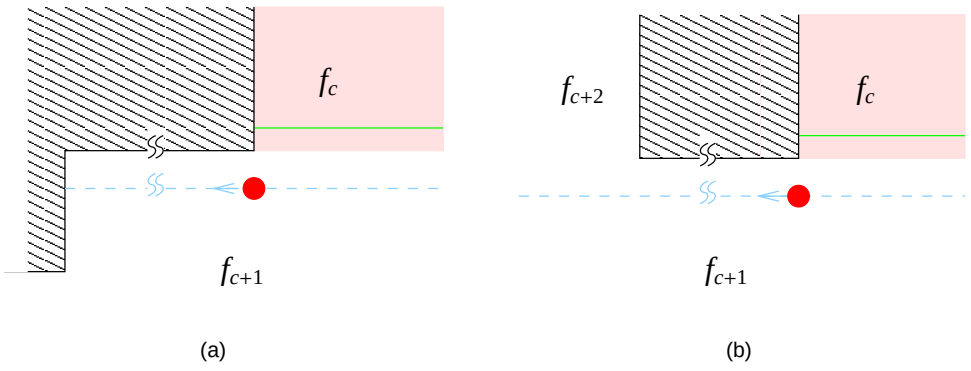


Figure 3.10: At the position marked with a large dot, these two situations appear identical locally to a robot. Only if the robot continues along the path will it discover the differences. (The obstacles are assumed to be very long). Event (a) lengthen, and (b) split of slice decomposition II.

(line 18). The events referred to in line 8 and 18 are those defined in Section 3.2.1. When an event occur, the current cell f_c is removed from the open list O (line 10), and added to the finish list F (line 9). If the event is a split or a merge, other cells are sharing the same boundary as the current cell f_c are added to the open list O, provided that they are not already in either list O or F (line 12). Similarly, in lengthen or shorten events, the neighbouring cell f_{c+1} is added to the open list O if it is not in either list O or F (line 15).

Algorithm 3.2 Slice Decomposition II

```
 1: O ← initial cell
 2: F ← ∅
 3: while O ≠ ∅ do
 4:    f_c ← f ∈ O
 5:    move to one (of two) cell boundary of f_c
 6:    repeat
 7:       move sweep line by Δx towards the opposite cell boundary
 8:       if event occur then
 9:          F ← F + f_c
10:          O ← O − f_c
11:          if event = split or merge then
12:             O ← O + f_{c+1}, f_{c+2} if f_{c+1}, f_{c+2} ∉ (O ∪ F)
13:          end if
14:          if event = lengthen or shorten then
15:             O ← O + f_{c+1} if f_{c+1} ∉ (O ∪ F)
16:          end if
17:       end if
18:    until event occur
19: end while
```

Figure 3.11 shows an example of slice decomposition II. Here, f_1 is the initial cell of the decomposition. When the sweep line arrives at the obstacle, a split event occurs, and free space cell f_1 is split into f_2 and f_3 (Figure 3.11(a)). Both the open O and finish F lists are updated. Then, f_2 is selected as the next current cell. Selection criteria is based on distance from the current cell and will be discussed in Chapter 4. When the sweep line arrives at the bottom cell boundary (Figure 3.11(b)), cells f_4 and f_5 are discovered and added to the open list O. Cell f_2 is moved to the finish list F because the sweep line has completely passes over the cell. f_5 is chosen as the next current cell in Figure 3.11(c). When the sweep line arrives at the top cell boundary, no new cells are added to the open list O. In Figure 3.11(d), f_4 becomes the current cell. When the sweep line reaches the bottom cell boundary of f_4, a merge event occurs. Two other cells, f_3 and f_6, share this cell boundary with f_4. However, only f_6 is added to the open list O because f_3 is already in the list. In Figure 3.11(e) and Figure 3.11(f), f_6 and f_3 becomes the current cell respectively. When the sweep line finishes its pass over f_3, the open list O becomes empty because all cells in the decomposition have been visited. The finish list F now contains all cells in the decomposition.

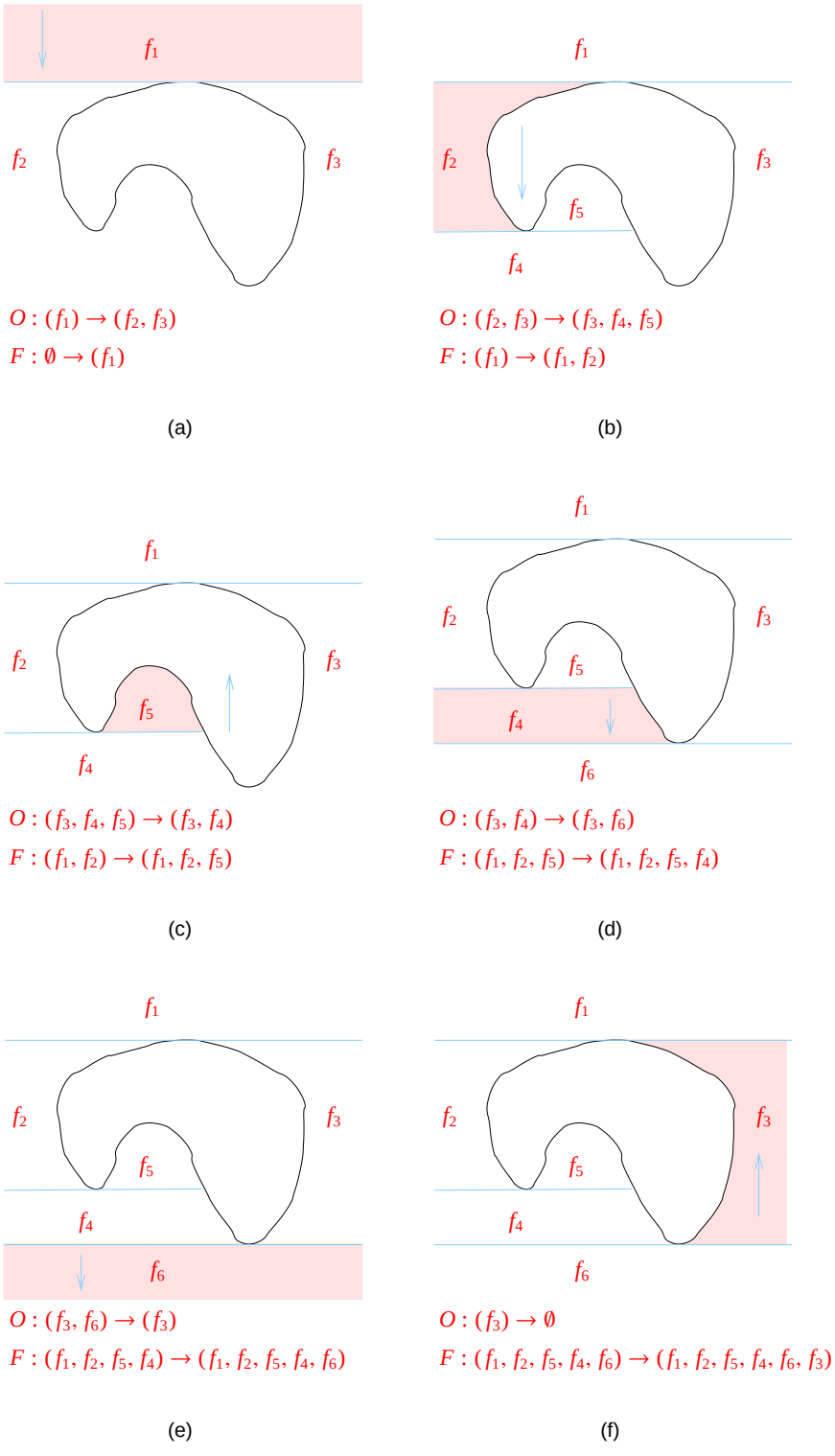


Figure 3.11: Creating a cell decomposition using slice decomposition II with an infinitesimally small step size Δx.

3.3 Effects of step size and sweep direction

Since slice decomposition is defined with a discrete line sweep process, the step size between consecutive slices therefore affects the decomposition yield for a given environment. If the step size is reduced to be infinitesimally small, $\lim_{\Delta x \to 0} \Delta x$, then the sweeping process becomes a continuous sweep, and slice decomposition forms an exact cell decomposition of the environment. So far in this chapter, it is assumed that the sweep is continuous.

However, slice decomposition also works for step sizes larger than infinitesimal. In such cases, slice decomposition forms an approximate decomposition of the environment instead. To capture all cells in a particular environment, the maximum step size has to be smaller than the height of the smallest cell

$$\Delta x = \min_i h(c_i) \tag{3.1}$$

Here, Δx is the step size of the line sweep and $\min_i h(c_i)$ is the height of smallest cell. Equation (3.1) guarantees that all cells will be present in at least one slice.

Figure 3.12 illustrates the effect of varying step size and the resulting decomposition created. When the step size is small, all cells in the environment are captured. For example, in Figure 3.12(a), the step size is small enough to guarantee a sweep line to pass through the small cell between the two lobes at the top of the obstacle. When the step size is increased to the height of the smallest cell, ie $\Delta x = \min_i h(c_i)$, the second sweep position in Figure 3.12(a) just barely touches the cell. If the step size is further increased, the smallest cell may be missed entirely, as is the case in Figure 3.12(c).

When (3.1) is satisfied, the decompositions created are independent of differences in step size. Compare the slice decomposition in Figures 3.12(a) and 3.12(b). Although the cells are discovered at different positions, the overall transitions of the list D are the same.

The slice decomposition created is the same whether the sweeping is done in the forward (top to bottom) or the reverse (bottom to top) direction. This is because the decomposition is dependent only on the position of the sweep lines. Figure 3.13 illustrates this concept. It shows the same sweep line positions as Figure 3.12(a), but the obstacle is upside down. It can be seen that the topology changes in the list D_t are essentially the same in both figures. The only change is to the numbering of cells, since the cells are discovered in a different order.

However, if the environment is rotated, the decomposition created will be different. (3.1) guarantees the same decomposition being created only for a particular sweep angle. Figure 3.14 shows the same obstacle as in Figure 3.12, but rotated 90^o. It can be seen that the decomposition is different from that given in Figure 3.12 regardless of how small the step size is. This is not a shortcoming of a discrete sweep algorithm because continuous sweep based exact cell decomposition, such as trapezoidal decomposition, is also affected by rotational transforms.

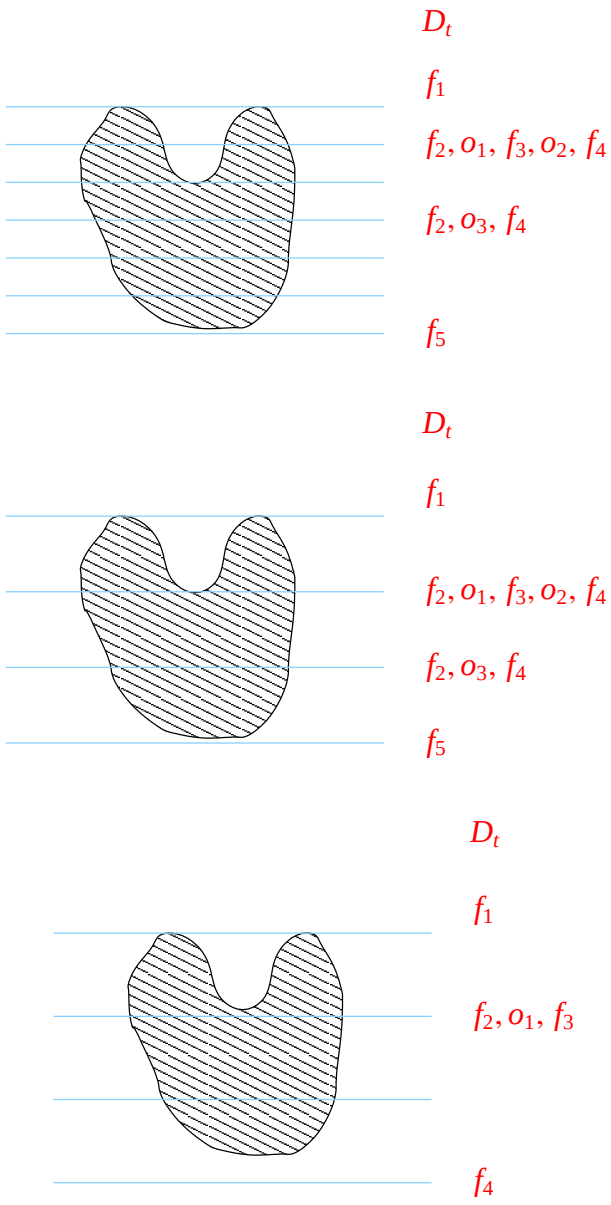


Figure 3.12: Effect of step size on decomposition produced. Slice decomposition I is used in this example. All sweep lines are assumed to be slightly above the obstacle surface they are touching. The list of cells on the right shows where events occur. (a) $\Delta x = \frac{1}{2} \times \min h(c_i)$, (b) $\Delta x = \min h(c_i)$, (c) $\Delta x > \min h(c_i)$.

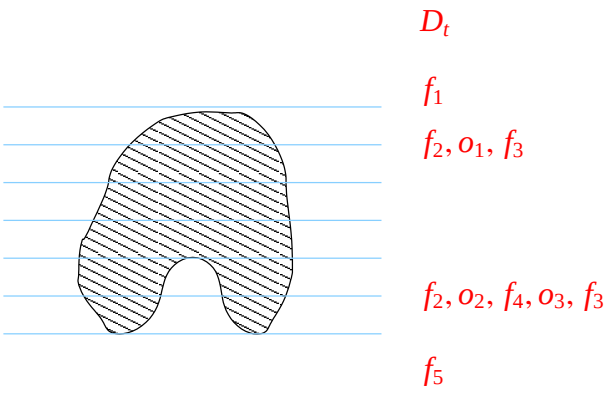


Figure 3.13: Forward and reverse sweep yield the same slice decomposition. Slice decomposition I is used in this example.

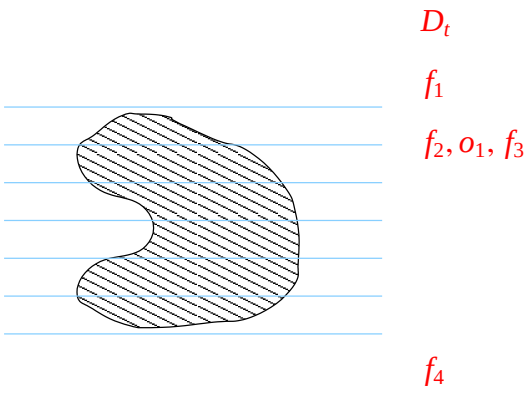


Figure 3.14: Rotation changes slice decomposition. Slice decomposition I is used in this example.

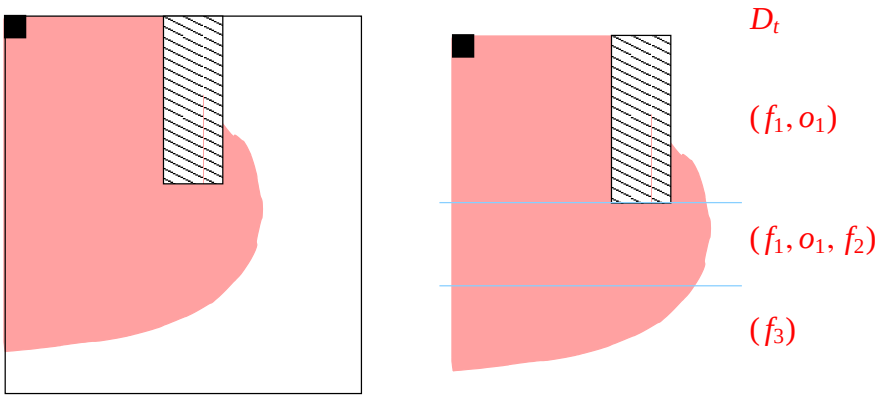


Figure 3.15: (a) The shaded area shows the reachable surface for a robot tethered to the top right corner of the environment. (b) To calculate its slice decomposition, the boundary of the environment is reduced to where the robot can reach.

3.4 Tethered robots

If the robot is tethered, its movement becomes restricted. The resulting reachable surface is dependent on the position where the tether is anchored and the length of the tether. Figure 3.15(a) shows the reachable surface area for a robot tethered at the top left hand corner. The situation can be viewed as a change of the boundary of the environment. To calculate the slice decomposition for this robot, only the reachable surface is considered. Figure 3.15(b) shows the resulting environment when restricted movement is taken into account. The slice decomposition created is shown on the right of the diagram.

3.5 Discussions

Slice decomposition extends the split and merge concepts introduced by boustrophedon decomposition. In addition to split (obstacle emergence) and merge (obstacle disappearance), slice decomposition I has two new events that handle topology changes associated with concave obstacles.

Ordinarily, the sweep line moves through the environment from top to bottom, passing both obstacle and free space regions. However, online cell decomposition methods have to use a different line sweep process because mobile robots can only move within the free space regions. Since the aim of this thesis is coverage in unknown environments, slice decomposition is thus modified to handle this restriction. Firstly, the event free space emergence is removed. Secondly, the lengthen and shorten events are added to simplify cell boundary detection in online

decomposition with range sensors. Lastly, instead of using a single list, the decomposition algorithm maintains two lists to remember cells that are detected (open list O) and visited (finish list F).

Events in exact cell decompositions are usually defined with small features in space. Examples are vertices in trapezoidal decomposition [29] and critical points in Morse functions in Morse decomposition [9]. In comparison, criticalities in slice decomposition are defined using large features, segments, in the environment. These large features have physical attributes that are detectable over time. Spurious sensor errors are filtered out through averaging. As a result, the detection becomes robust against noisy and inaccurate sensing [72].

Due to the use of topology changes as events, slice decomposition can handle a larger variety of environments compared to existing online cell decomposition based coverage algorithms. CC_R is designed for contact sensing robots working in rectilinear environments. Morse decomposition is more general and can handle obstacles with smooth surfaces. However, it is only defined for non-rectilinear environments because boundaries parallel to the sweep line are degenerate cases for Morse functions. In comparison, slice decomposition II can handle environments with polygonal and smooth-surfaced objects, including rectilinear ones.

3.6 Summary

This chapter introduces the events and algorithms of slice decomposition. It also discusses the effect of step size and sweep direction have on the decomposition formed.

Newton's Second Law of Graduation: The age, a, *of a doctoral process is directly proportional to the flexibility*, f, *given by the advisor and inversely proportional to the student's motivation*, m.

Jorge Cham, "Piled Higher and Deeper", www.phdcomics.com

4

Topological Coverage Algorithm

Chapter 3 introduced slice decomposition II, which creates a cell decomposition of an environment using a sweep line that is restricted to free space regions only. This restriction of the sweep line reflects the inability of mobile robots to move within obstacle regions of environments. This chapter continues the discussion by explaining how a mobile robot with range sensors can construct slice decomposition II *online*, while simultaneously covering the unknown space. This is achieved by creating a partial topological map using sensor information collected, and generating a coverage path using this partial map. The topological map embeds the slice decomposition of the environment by using the events in slice decomposition II as landmarks for its nodes. The map is updated whenever relevant new information becomes available. The path planner then generates a new path based on the updated partial topological map.

Specifically, the topological coverage algorithm implements the algorithm for slice decomposition II on Page 51 as a finite state machine (Section 4.1). The open and finish lists (O and F) are stored implicitly within the topological map, introduced in Section 4.3. Event detection is covered in Section 4.2, and the map updates associated are explained using examples in Section 4.3.3. The selection of the next cell to cover, and how a path is planned to reach it is described in Section 4.4. Finally, the chapter concludes with a discussion on the completeness (Section 4.5) and complexity (Section 4.6) of the topological coverage algorithm.

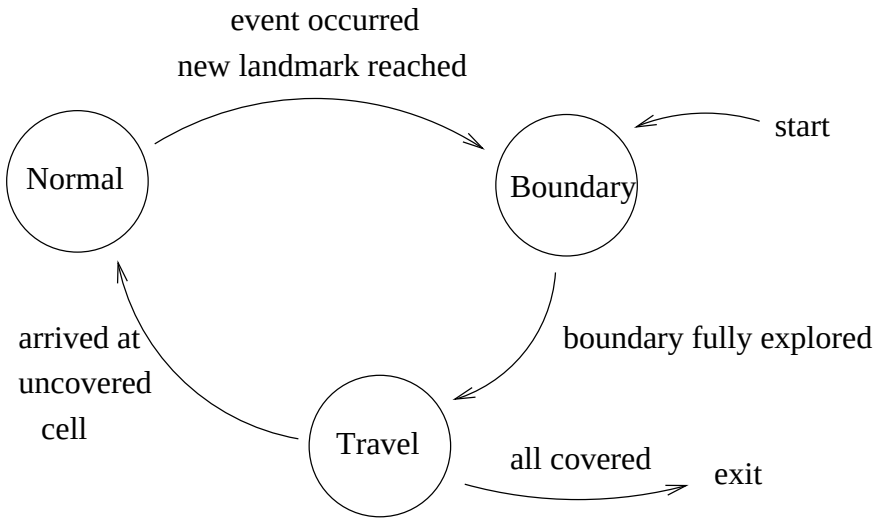


Figure 4.1: State transition diagram for the topological coverage algorithm.

4.1 Finite State Machine

The topological coverage algorithm is organised as a finite state machine with three states – boundary, normal and travel. Figure 4.1 shows the state transition diagram. The boundary state handles the situation where the robot is on a cell boundary. The algorithm always starts in this state. This is because the coverage process begins from a corner of the environment, which is a cell boundary of the initial cell. This restriction on initial condition is not a shortcoming because it is easy to program a robot to seek a corner by using simple forward and wall following movements. The first corner found will then become the initial cell boundary.

When the robot finishes exploring the cell boundary, execution of the topological coverage algorithm switches to the travel state. The robot searches its topological map and moves to the selected uncovered cell. When it arrives at the selected cell, the algorithm enters the normal state. This state controls the robot to follow a zigzag path to cover all the surface area in the cell.

4.1.1 State – Normal

The normal state handles the coverage of individual cells in slice decomposition. This corresponds to line 7 inside the `repeat .. until` loop in Algorithm 3.2 on Page 51. Its operation is summarised in Algorithm 4.1.

Normally, the robot follows a zigzag path to cover all surface area in the current cell. This continues until it arrives at a landmark, which signifies arrival at a cell boundary. The topological map G is updated with the information that the current cell is completely covered. Details on

Algorithm 4.1 Normal State

```
1: repeat
2:     follow zigzag pattern
3: until at landmark
4: update G
5: state ⇐ boundary
```

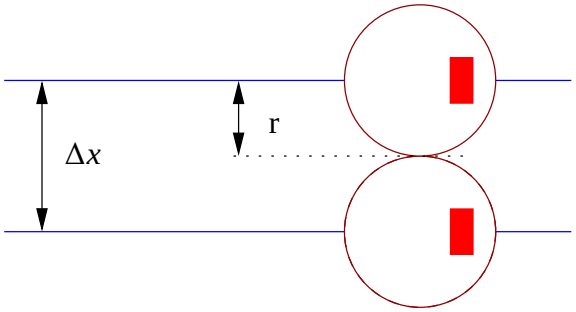


Figure 4.2: To cover all surfaces between two consecutive strips, the inter-strip distance Δx cannot be larger than the diameter of the robot $2r$.

how landmarks and cell boundaries are detected can be found in Section 4.2, and description of the map update process can be found in Section 4.3.

The step size Δx of the line sweep needs to be small enough to cover all surfaces between neighbouring strips in a zigzag. In Figure 4.2, the width of the robot is the same as the step size, ie $\Delta x = 2r$. In this case, the robot can cover all surfaces between strips, without any overlapping. However, it is usually better to allow for an error margin and the area covered should slightly overlap instead, as shown in Figure 4.3.

The following equation summarises the choice of step size Δx for a robot with radius r:

$$\Delta x \leq 2r \tag{4.1}$$

This equation states that the step size Δx should not exceed the diameter of the robot $2r$. If the robot is rectangular, then $\frac{1}{2}w$ should be used instead of r, where w is the width of the robot.

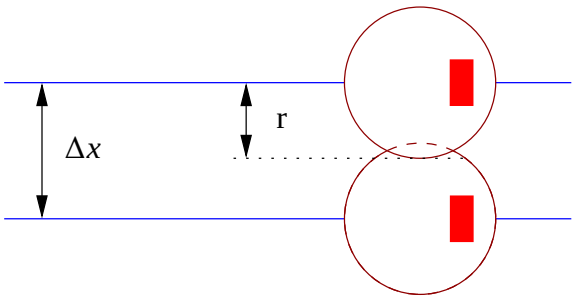


Figure 4.3: The area covered by neighbouring strips should overlap to provide a good coverage.

4.1.2 State – Boundary

The boundary state handles the situation where the robot is on a cell boundary. The topological coverage algorithm always starts in this state. This is due to the assumption that the initial position of the robot is at a corner of the environment. In other words, the initial position is on a cell boundary of the initial cell. This restriction on initial position is not a shortcoming because it is easy to program a robot to seek a corner by using simple forward and wall following movements.

The operation of the boundary state is summarised in Algorithm 4.2.

Algorithm 4.2 Boundary State

```
 1: loop
 2:    move forward along boundary
 3:    if at landmark then
 4:       update G
 5:    end if
 6:    if arrive at end of strip then
 7:       update G
 8:       if boundary fully explored then
 9:          state ⇐ travel
10:       else
11:          turn around 180^o
12:       end if
13:    end if
14: end loop
```

The aim of the boundary state is to direct the robot to move along the boundary to expose all cells neighbouring the current border. The topological map G is updated whenever the robot arrives at a landmark. It is also updated when the robot reaches either end of the cell boundary. When the robot reaches one end of the boundary, it checks to see if it has been to both ends (line 8). If it has, then the cell boundary is fully explored, and execution of the algorithm switches to the travel state. Otherwise, it turns around and moves towards the other end of the boundary. Section 4.2 describes the operation of the boundary state in detail.

The boundary state corresponds to line 8 - 18 in Algorithm 3.2 on Page 51.

4.1.3 State – Travel

The travel state is responsible for generating and following paths that move the robot from one cell to another. It implements lines 3 - 5 of Algorithm 3.2 on Page 51. Algorithm 4.3 summarises the operation of the travel state:

Algorithm 4.3 Travel State

```
1:  T(n) ⇐ search G
2:  if T(n) = ∅ then
3:      exit algorithm
4:  end if
5:  while T(n) ≠ ∅ do
6:      move towards T(0)
7:      if at T(0) then
8:          T(n) ⇐ T(n) − {T(0)}
9:      end if
10: end while
11: state ⇐ normal
```

First a graph search is done on the topological map G to find the closest uncovered cell, represented by a node in map G, from the current location. The search returns a list of nodes, $T(n)$, leading from the current node to the selected uncovered node. Here, $T(0)$ denotes the first node in the list $T(n)$. If the search returns an empty list, the environment is completely covered and the algorithm exits (lines 2 - 3). To reach the selected uncovered cell, the robot moves from one node in $T(n)$ to the next. A node is removed from $T(n)$ when the robot reaches the corresponding area (lines 7 - 8). When the robot arrives at the last node in $T(n)$, ie the uncovered node, operation switches to the normal mode. More details on path following and landmark matching in the travel state can be found in Section 4.4.

4.2 Cell boundaries

A cell boundary occurs when the robot approaches a change in topology of segments. This change is detected by monitoring obstacles around the robot. The conditions at which the robot is considered to have arrived at a landmark and thus a cell boundary, and the action it takes during the boundary state is determined by the type of topology change, the step size Δx and the detection range of the robot.

This section assumes that landmark detection is achieved using a combination of simple thresholding, temporal sequence comparisons[1], and odometry (comparing length of consecutive strips). An alternative solution that uses a neural network is presented in appendix A.

All the diagrams in this section assume the sweep direction for the current cell is from top to bottom.

[1] comparing current sensor reading with previous ones

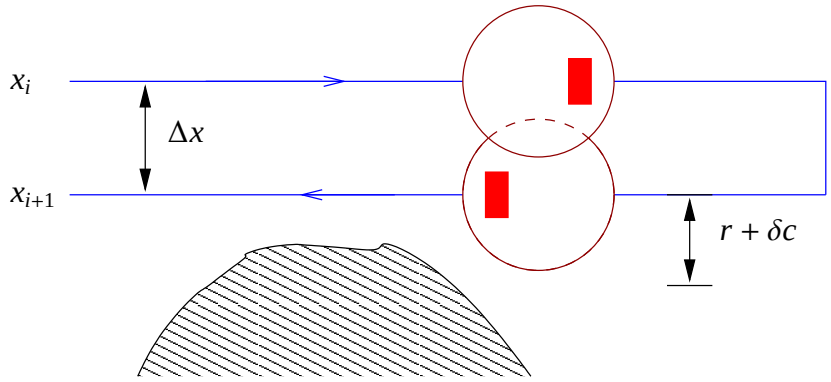


Figure 4.4: Around a split event.

4.2.1 Event – Split

Figure 4.4 shows an example of a robot approaching a cell boundary caused by a split event. This diagram shows the two closest sweep line positions around the criticality. The sweep line at x_i is the last strip where the robot can move freely along the strip unobstructed. At the next sweep position x_{i+1}, the robot's path will be blocked. This is because the distance between the strip to the obstacle is smaller than the minimum required for a robot to pass unobstructed:

$$d_{\text{min}} = r + \delta c \tag{4.2}$$

r is the radius of the robot and δc is the minimum clearance the robot kept from obstacles.

A split event can first be detected at the sweep position x_i. In Figure 4.5(a), the distance to the obstacle dropped below a threshold. At this point, the execution of the topological coverage algorithm switches to the boundary state. As the robot continues to move forward, the distance to the obstacle will eventually rises above the threshold, as shown in Figure 4.5(b). Between the two positions shown in Figure 4.5, the distance to the obstacle stays below the threshold. Therefore, if the robot missed the first transition where the distance drops below the threshold (for example, due to sensor error), the split event can still be detected anytime while the distance stays below the threshold.

If the robot leaves the current cell immediately after exploring the strip at x_i, the surface along the emerging obstacle will remain uncovered. This is illustrated in Figure 4.6. The surface along the dotted line in the diagram will be missed if the robot follows only the original sweep positions at x_i and x_{i+1}. How much of the surface is left uncovered depends on the length of the obstacle segment.

To address this problem, the robot will need to execute an extra strip that lies between the sweep lines at x_i and x_{i+1}, as shown in Figure 4.7. The extra sweep position is labelled as $x_{i'}$. It is positioned so that the robot can get as close to the obstacle as possible.

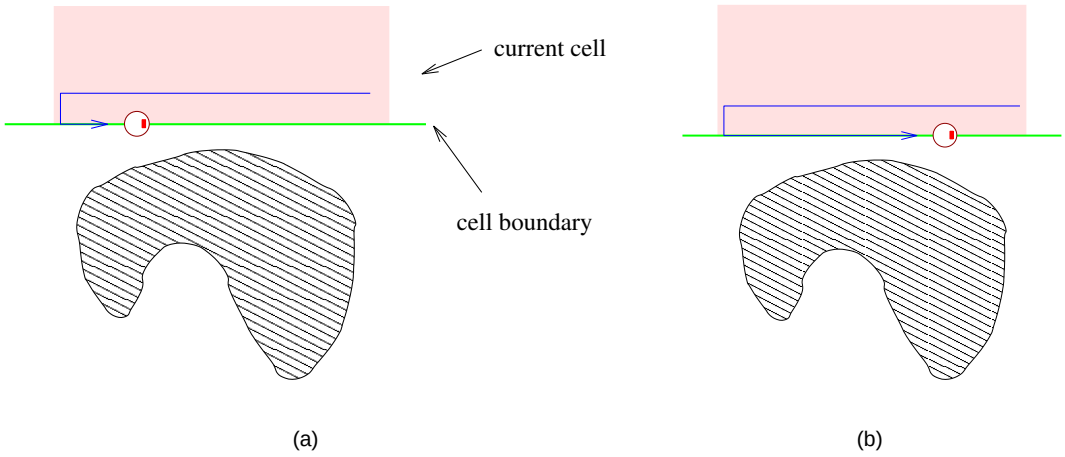


Figure 4.5: A split event occurs when there are obstacles close to the side of the robot in the direction it is sweeping towards.

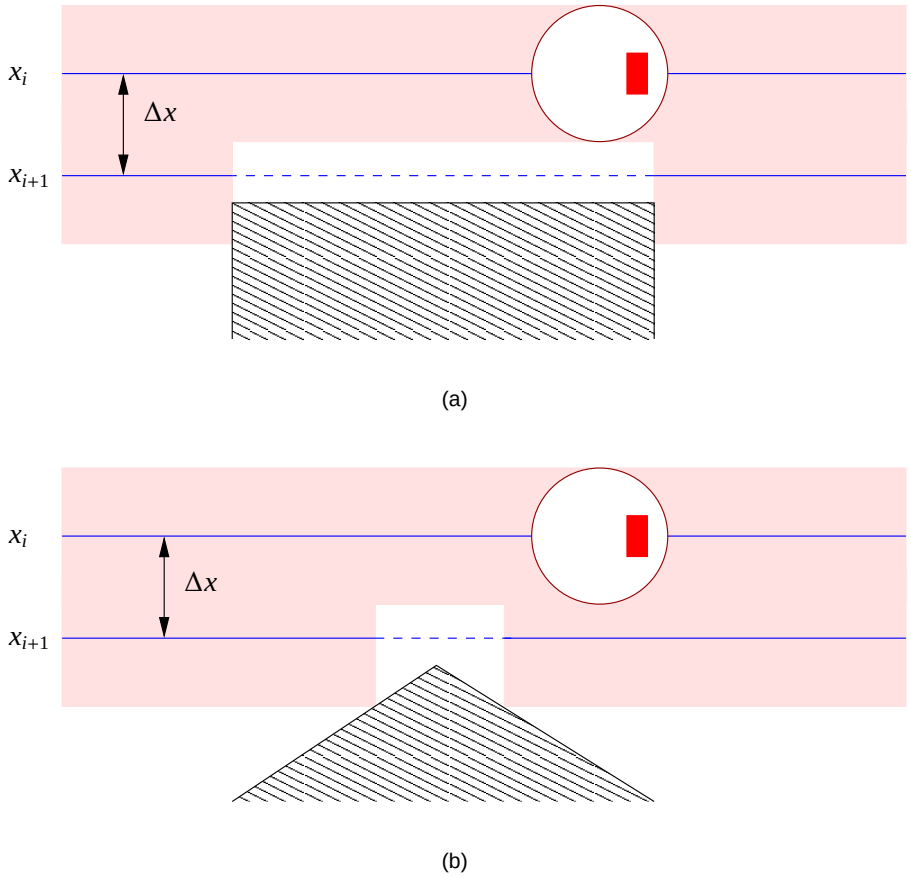


Figure 4.6: By following only the original sweep line positions, surfaces between the boundary strip and the obstacle will be missed. The area covered by the robot is shaded.

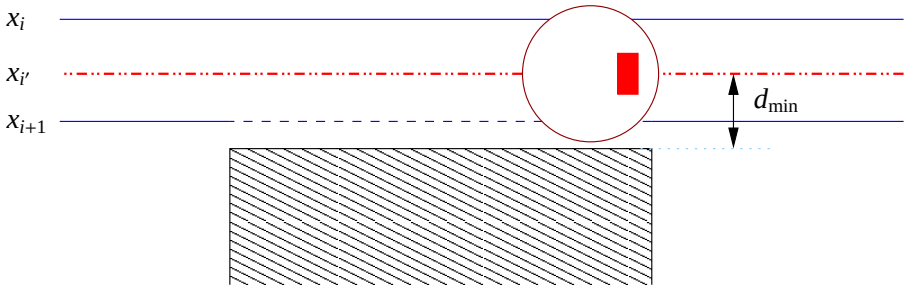


Figure 4.7: To cover the surface next to the obstacle, an extra strip is added between the normal sweep line positions.

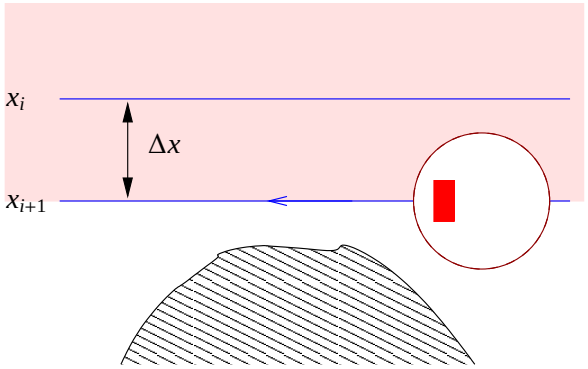


Figure 4.8: Detecting a split event when the robot's path is obstructed at x_{i+1}.

After the robot covered this extra strip, the current cell is declared to be completely covered, and execution of the topological coverage algorithm enters the travel state to find a suitable uncovered cell and travel there.

In the event that the robot fails to detect the split event while following sweep line position x_i, it will continue with the zigzag path and move on to the next strip at position x_{i+1}. Eventually, the robot's path will be blocked by the obstacle, as shown in Figure 4.8. There are two different ways to detect a split event from the strip position at x_{i+1}. The first is by the presence of an obstacle in the lower front of the robot, as shown in Figure 4.8. The other is by the sudden reduction in length of the current strip compared to the previous one. Also, since a split event can be detected via odometry (by comparing lengths of consecutive strips), it can be detected by mobile robots equipped with bump sensors only.

Since the path along the strip is blocked, the robot will need to go around the obstacle to fully explore the cell boundary. This is shown in Figure 4.9. This is achieved using a wall following move around the obstacle. When the robot reaches the end of adjusted strip, there will be no need for the addition of an extra strip since the surface near the obstacle will already be covered.

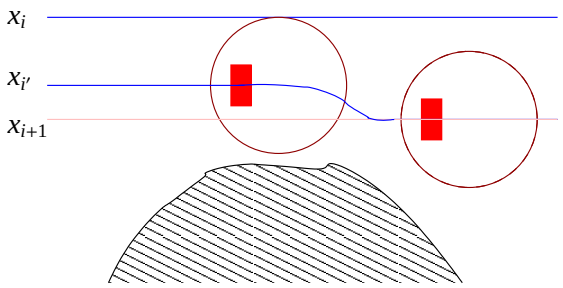


Figure 4.9: This obstacle blocks the movement of the robot along a strip. The robot moves around the obstacle to fully explore the cell boundary.

4.2.2 Event – Merge

Figure 4.10 shows the two closest sweep positions around a merge event. At x_i, the robot's path is blocked by the obstacle which forms one of the side boundaries of the current cell. At the next strip position x_{i+1}, the boundary disappears, and the robot can move freely underneath it. In other words, the strip at x_{i+1} is non-trivially longer than the strip at x_i because of the opening caused by the disappearing obstacle. More precisely, the length of the strip at x_{i+1}, $L_{x_{i+1}}$, is non-trivially longer than L_{x_i} if

$$L_{x_{i+1}} > L_{x_i} + 2r \tag{4.3}$$

where r is the radius of the robot.

In Figure 4.11, the robot travels on the last strip of the current cell in the direction approaching the disappearing side boundary. While still moving along the strip of its zigzag path, the robot can first detect the opening in the side boundary by the absence of an obstacle in its lower front (Figure 4.11(a)). Alternately, it can detect the event when moving from the strip at x_i to the next strip at x_{i+1}, with the obstacle disappearing from its side (Figure 4.11(b)). Similar to the situation with the split event, an extra sweep position between x_i and x_{i+1} is needed to cover the surface immediately underneath the disappearing obstacle. This extra sweep position $x_{i'}$ is put as close as possible to the obstacle. Since the robot arrives at the cell boundary in the middle of the strip, it has to travel in both directions from this entry point to fully explore the boundary. In other words, the robot has to first travel until the end of the strip in one direction, turn around, and then travel until the end of the other direction. Only after both ends of the strip was visited can the execution of the algorithm leaves the boundary state.

Figure 4.12 shows the reverse situation where the robot moves away from the criticality on the last strip of the cell. The robot can first detect the merge event in this situation as an opening behind itself while moving along the sweep position at x_i. When it arrives at the end of the strip, it moves down to the extra sweep position $x_{i'}$ to start exploring the boundary. The move to the extra strip position does not have to be very accurate. The robot can simply move a distance less than the normal step size Δx. When the robot approaches the obstacle segment, it adjusts

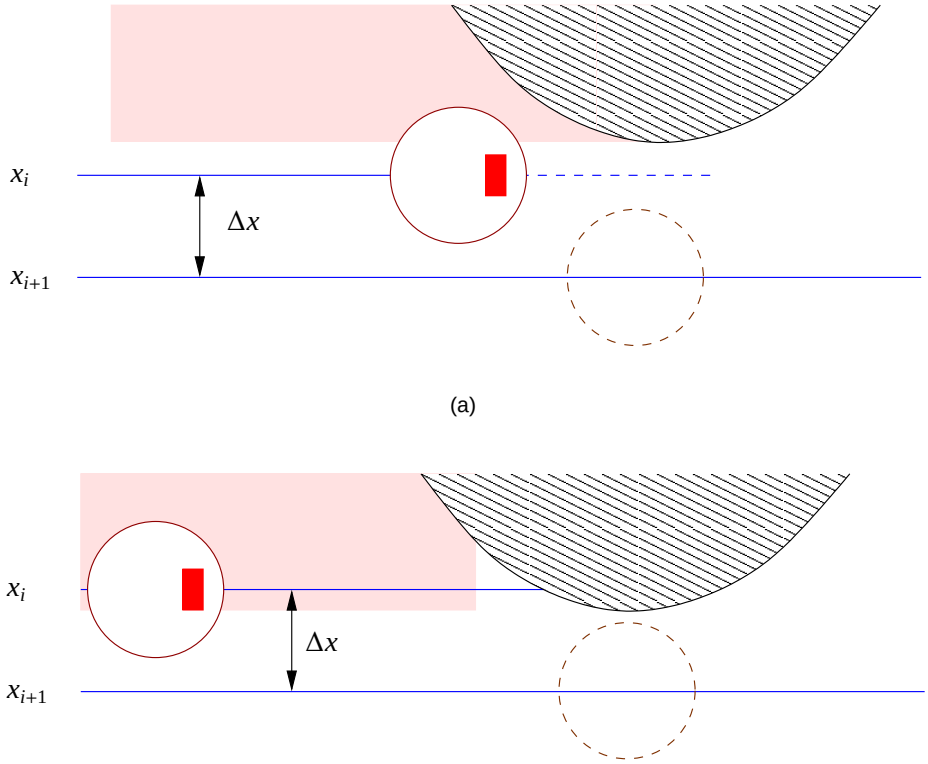


Figure 4.10: Around a merge event.

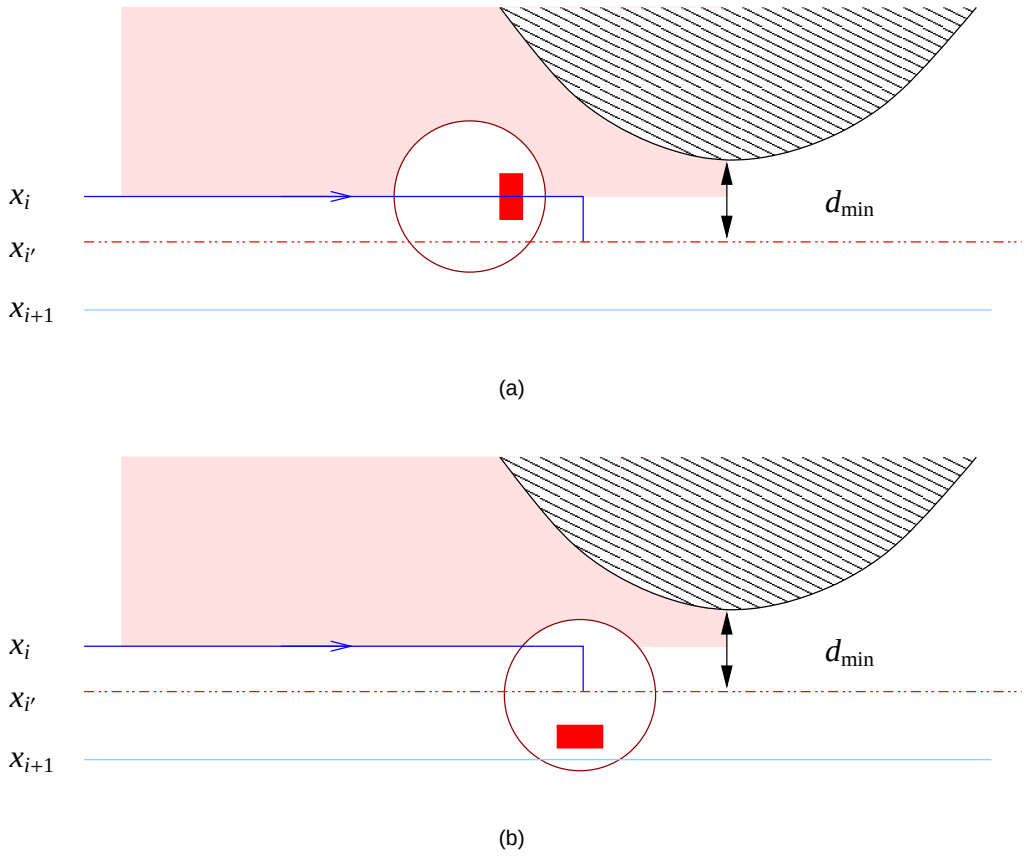


Figure 4.11: Cell boundary handling for robots moving towards a merge event.

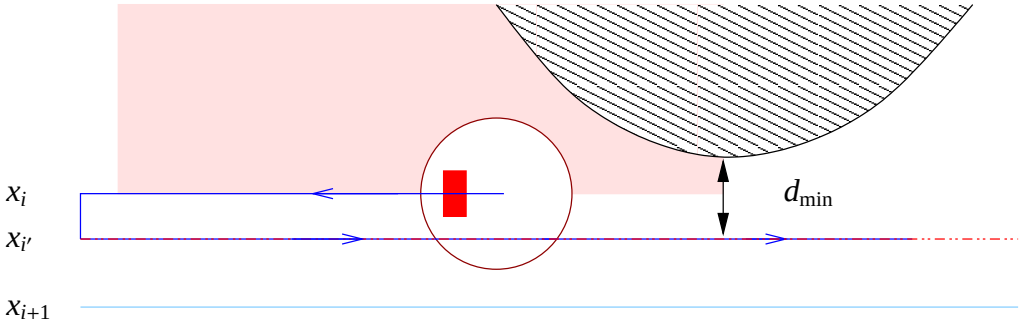


Figure 4.12: Cell boundary handling for robots moving away from a merge event.

its distance to the obstacle. Because the robot enters the extra strip from one of its ends, the robot enters the travel state when it reaches the other end.

If the robot fails to detect the merge event while at the strip position x_i, it will follow its normal zigzag path and move on to the next strip position x_{i+1}. This is shown in Figure 4.13. There are two ways to detect the merge event in this situation. Using range sensors, the robot can detect an obstacle in the direction where it has just swept. Otherwise, it can detect the event by comparing the length of the strips at x_i and x_{i+1}. This is because the strip at x_{i+1} is a lot longer than the one at x_i. After the robot has travelled a certain distance longer than the previous strip, it will declare that it has entered a cell boundary. Since a merge event can be discovered by comparing strip lengths, it can be detected by contact sensing robots. To cover the surface close to the disappearing obstacle, the robot adjusts its path to be closer to the obstacle, indicated as the extra sweep position $x_{i'}$ in Figure 4.13.

4.2.3 Event – End

An end event occurs when there are no more strips left from the current position for the robot to continue. Figure 4.14 shows a situation where this happens. Here, the robot finishes covering the current cell and reaches the bottom cell boundary. This cell boundary does not lead to any other free space cell. Therefore, this is the last free space segment left from the current sweep position.

The end event can be first detected by the robot while travelling along x_i. Similar to the split event, an end event is caused by the presence of obstacles in the direction the robot is sweeping towards. A sweep position becomes the last strip in the cell when the distance from the strip to the obstacle falls below a threshold. The difference between the two events is that in an end event, the cell boundary is not shared with any other free space regions. To cover the remaining surface left in the free space cell, an extra strip is added as shown in Figure 4.15.

If the robot fails to detect the end event from x_i, it will continue with its zigzag path and attempts

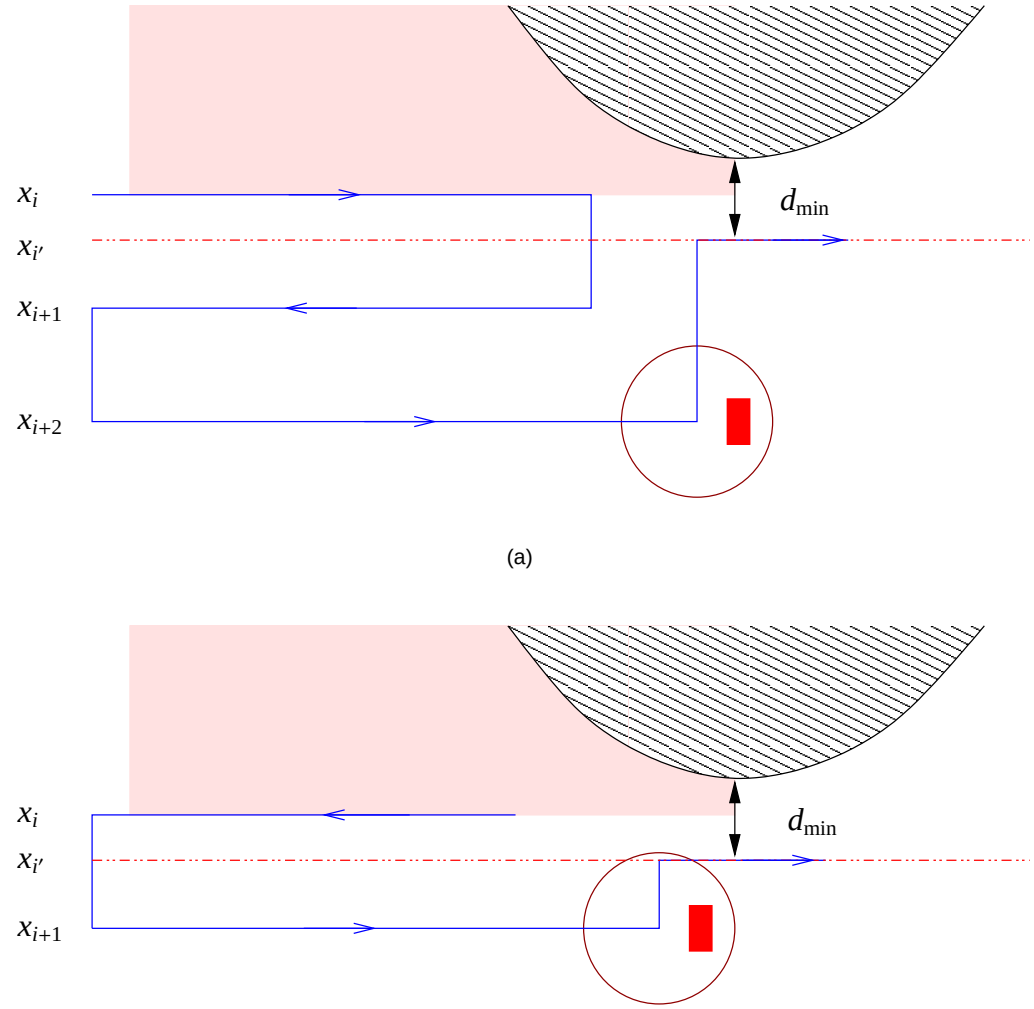


Figure 4.13: Detecting a merge event after the side boundary has disappeared.

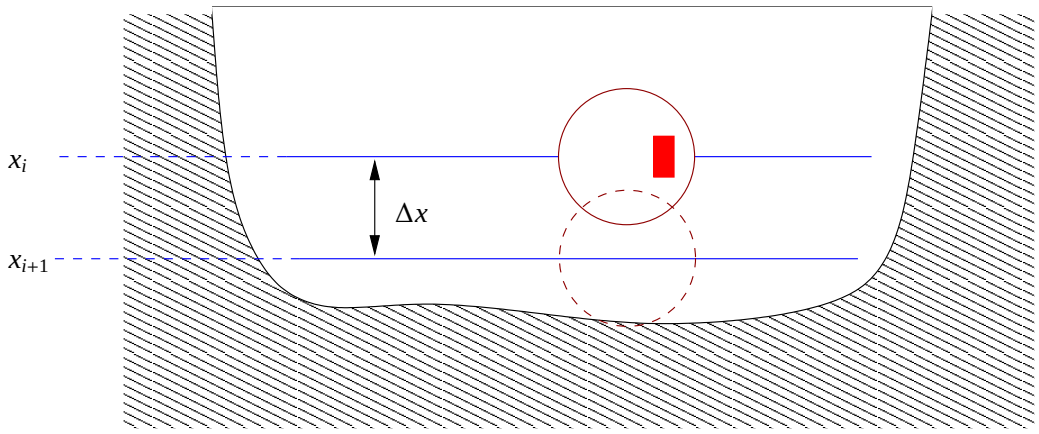


Figure 4.14: Around an end event.

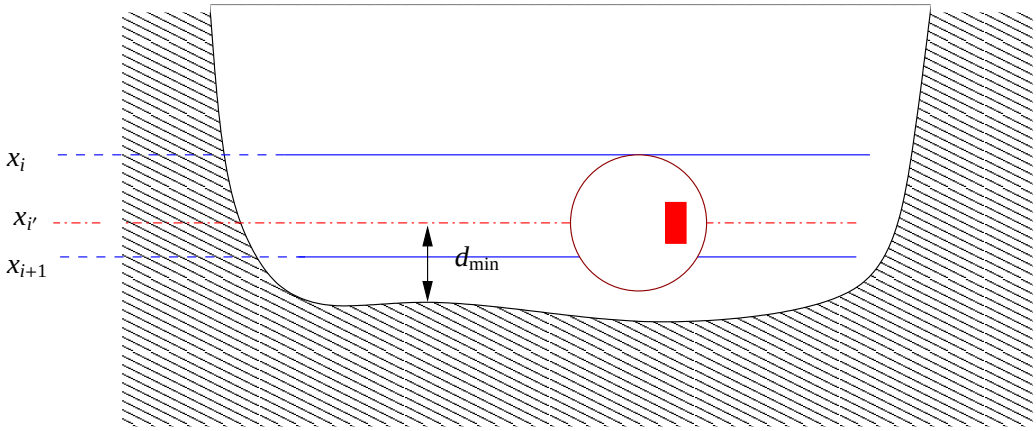


Figure 4.15: An extra strip is added between the last strip in the cell at x_i and the obstacle to cover the remaining surfaces in the cell.

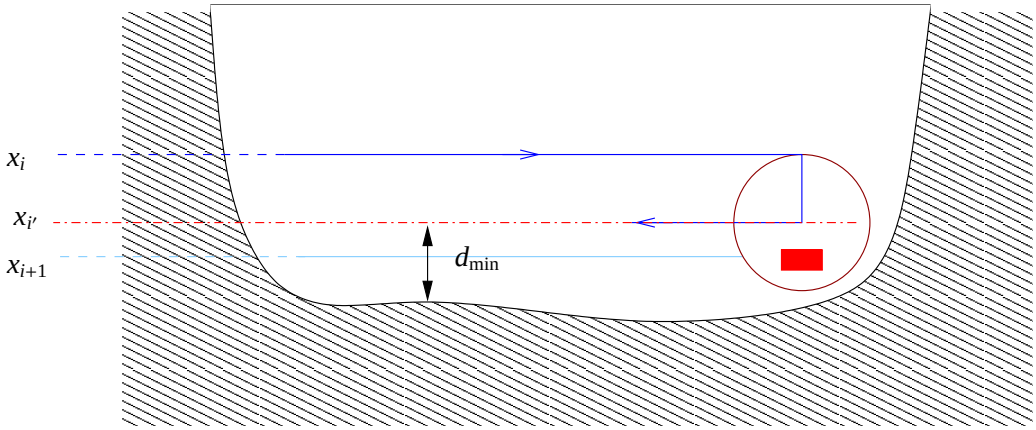


Figure 4.16: The cell boundary blocks the robot's movement from x_i to x_{i+1}.

to move into the next strip position at x_{i+1}. However, the robot will not be able to move into the strip position at x_{i+1} because it is too close to the obstacle, as shown in Figure 4.16. When the robot's path is blocked while it is moving from one strip to the next, it declares that it has entered a cell boundary. In other words, an end event can be detected by a contact sensing robot. To cover the remaining surface in the cell, it can simply follow the obstacle from its current position to the end of the strip ($x_{i'}$ in Figure 4.16).

4.2.4 Event – Lengthen

Figure 4.17 shows the sweep positions around a lengthen event. Detection of lengthen events is identical to that of merge events. As explained in Figure 3.10 on page 50, the merge and lengthen events appear identical locally to a robot when the event is first detected. Only when the robot continues along the path will it discover the differences. With a merge event, the robot

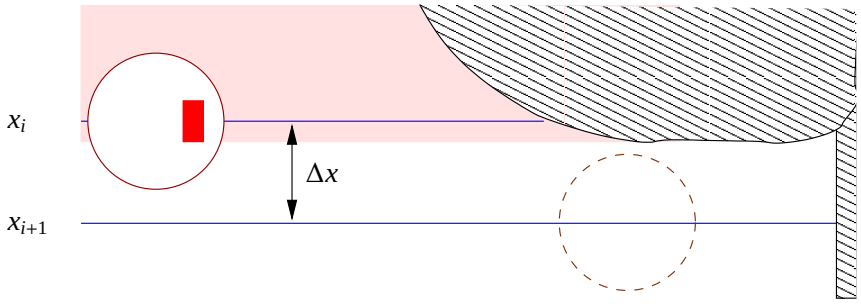


Figure 4.17: Around a lengthen event.

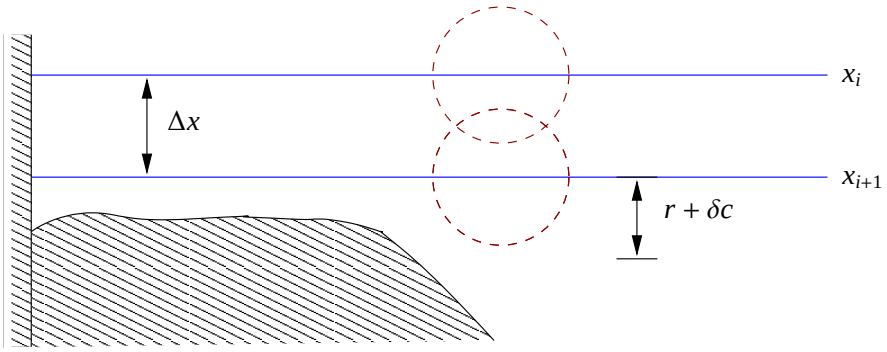


Figure 4.18: Around a shorten event.

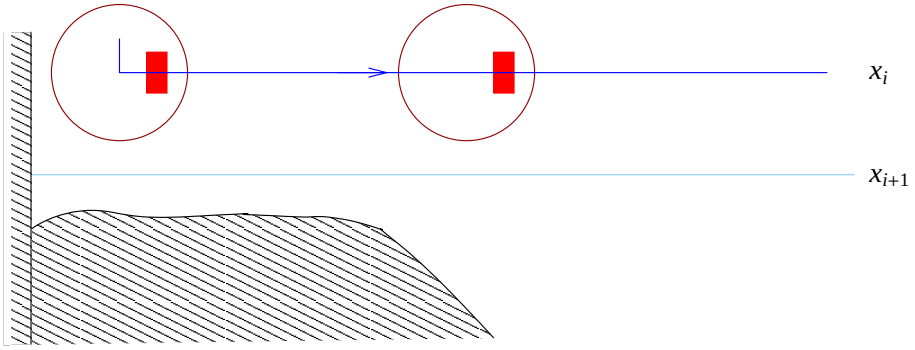
will detect another cell in the direction of the obstacle that causes the event originally. With a lengthen event, the obstacle will stay on the side of the robot until the robot reaches the end of the strip.

4.2.5 Event – Shorten

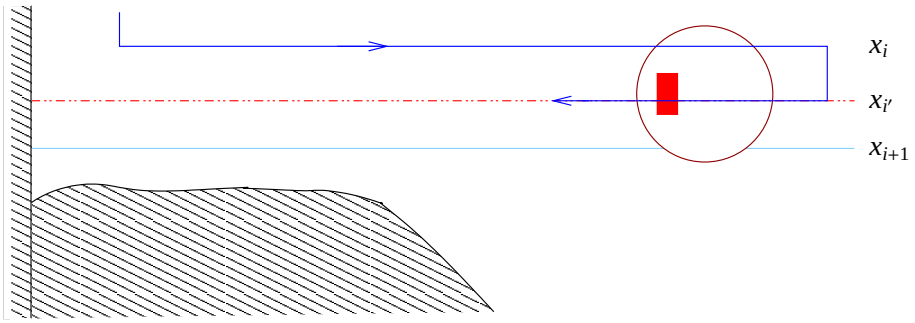
Figure 4.18 shows the sweep positions around a shorten event. In Figure 4.19(a), the robot enters the strip at x_i above the obstacle emergence. Therefore, the robot can detect that the distance to the obstacle has dropped below the threshold when it enters the strip. This distance stays below the threshold until the robot reaches the second position shown in Figure 4.19(a). To cover the remaining surface in the current cell, an extra sweep position $x_{i'}$ is added, as shown in Figure 4.19(b).

If the robot fails to detect the obstacle emergence at x_i, it will continue and move to the next strip at x_{i+1}. At this position, the robot's path will eventually be blocked by the emerging obstacle, as shown in Figure 4.20(a). A contact sensing robot will detect the shorten event at x_{i+1}. To completely explore and cover the remaining boundary, the robot will need to follow the obstacle, as shown in Figure 4.20(b).

Figure 4.21(a) shows the reverse situation, where the robot moves along the strip at x_i in the direction towards the obstacle emergence. At the position shown in the diagram, the distance

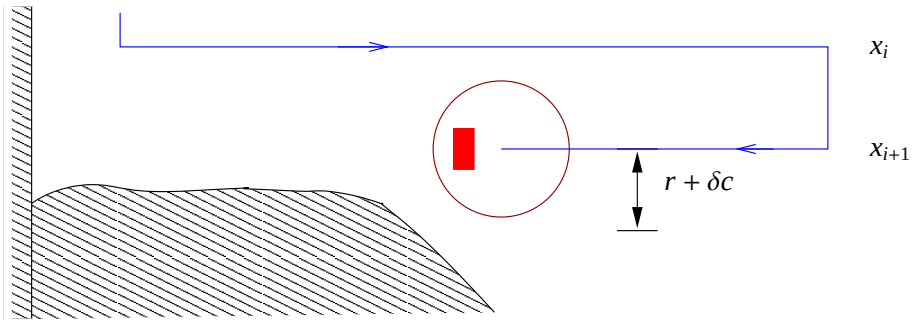


(a)

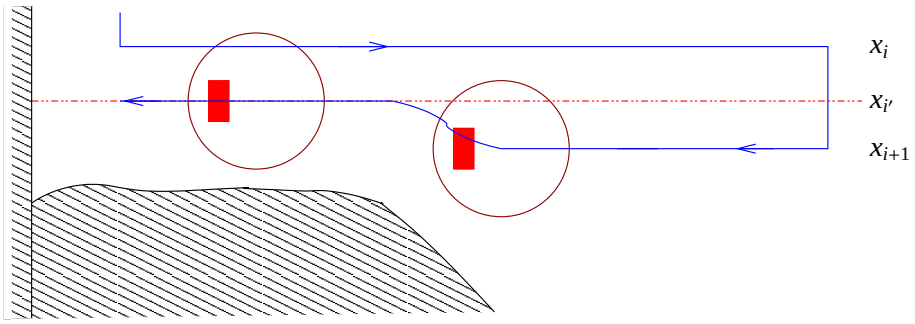


(b)

Figure 4.19: Detecting shorten events by monitoring obstacles in the direction the robot is sweeping towards.



(a)



(b)

Figure 4.20: The obstacle emergence in a shorten event blocks the path of the robot.

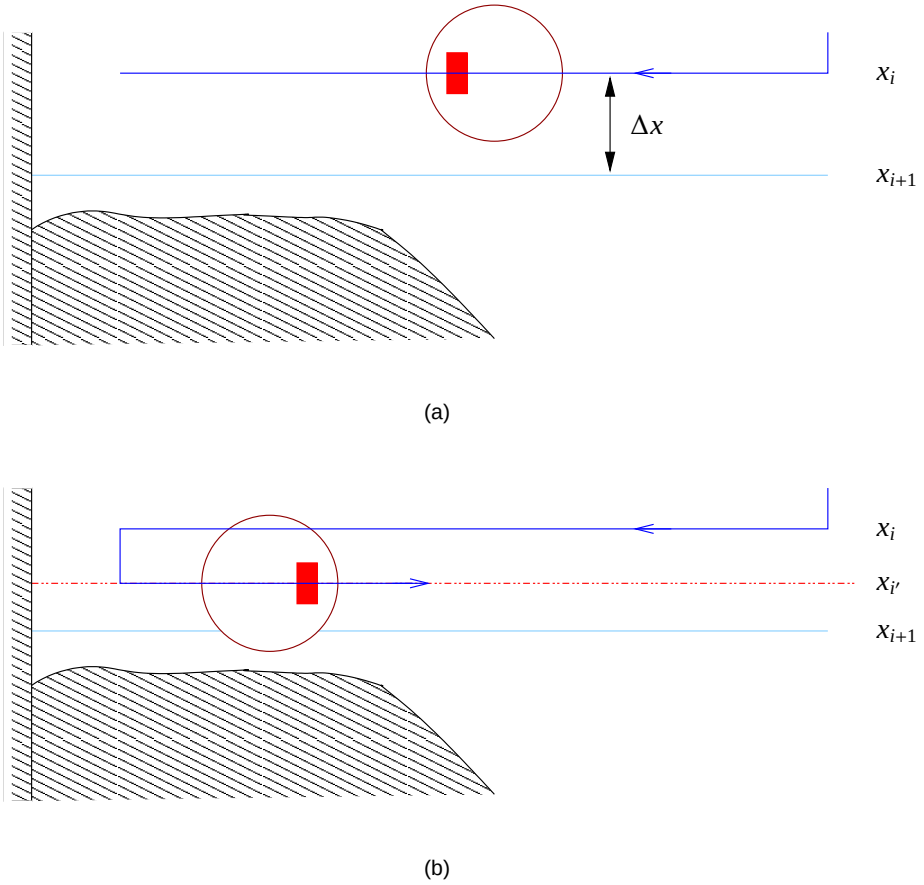


Figure 4.21: Detecting shorten events by monitoring obstacles in the direction the robot is sweeping towards.

to the obstacle drops below the threshold, and execution switches to boundary state. For the remaining of the strip, this distance stays below the threshold. To cover the remaining area of the current cell, an extra sweep position $x_{i'}$ is added, as in Figure 4.21(b).

If the robot fails to detect the obstacle emergence at x_i, it will continue with its zigzag path. However, the robot will not be able to move to its next strip at x_{i+1} since it is too close to the obstacle, as shown in Figure 4.22. When the robot's path is blocked while it is moving to the next strip, it declares that it has entered a cell boundary, and start exploring the rest of the boundary (same as Figure 4.21(b)).

4.2.6 Combination of split and merge events

The discussion for merge (Section 4.2.2) event assumes the first sweep line after the disappearing obstacle extends beyond the obstacle. However, if there is another obstacle underneath the disappearing one, no sweep line might pass underneath the disappearing obstacle segment at all. This situation is shown in Figure 4.23. Here, the two obstacles form a small gap of width $2d_{\min}$,

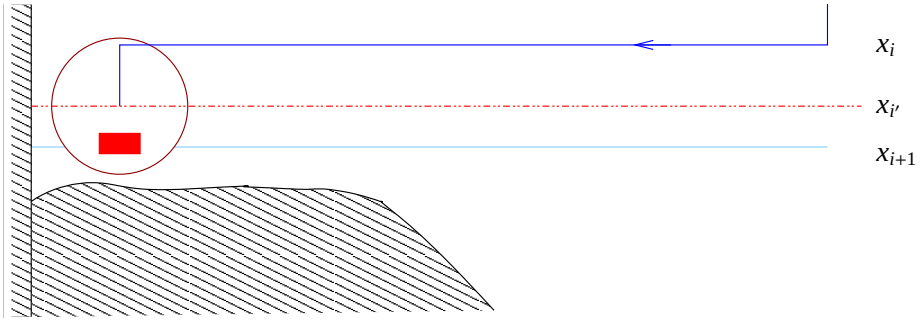


Figure 4.22: The cell boundary blocks the robot's movement from x_i to x_{i+1}.

which is just wide enough for the robot to pass through. Only if a sweep line is positioned in the exact middle of the gap will the robot be able to pass through the gap.

In Figure 4.23, the strip at x_i is the last one before the right side boundary disappears. The strip at x_{i+1} would have extended underneath the top obstacle if the bottom obstacle does not exist. In Figure 4.23(a), the robot moves along the strip at x_i in the direction approaching the disappearing side boundary. In Figure 4.23(b), the robot moves along the strip at x_i in the direction away from the disappearing side boundary. In both cases, the gap is detected with on board range sensors while the robot is at either x_i or x_{i+1}. To detect this gap, the range sensors must have a detection range larger than the minimum clearance the robot kept from obstacles δc. Therefore, if $d_{\text{range-min}}$ is the minimum sensing range needed, then

$$d_{\text{range-min}} > \delta c \tag{4.4}$$

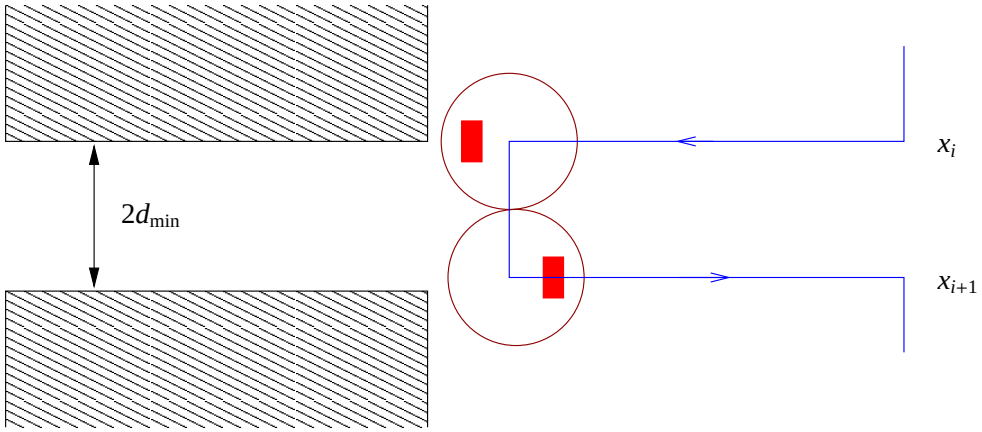
Since this combination event cannot be detected using odometry only methods, it cannot be detected by a contact sensing robot. Unlike the merge event described in Section 4.2.2, there is no sudden change in strip length before and after the event.

To explore and cover the gap between the obstacle, an extra sweep position is added midway between the gap, as shown in Figure 4.24.

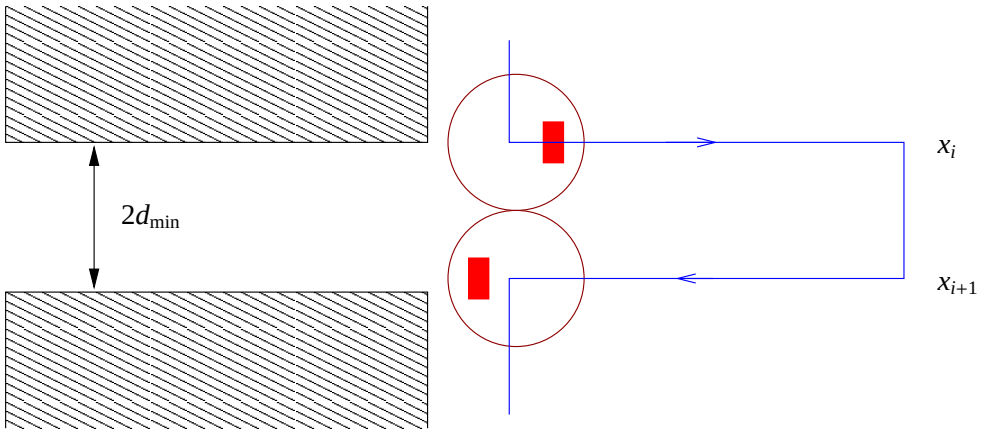
4.3 Topological Map

A planar graph representing the slice decomposition can be created by assigning nodes to the thresholds described in Section 4.2, and connecting these nodes with edges. Figure 4.25 shows examples of *incomplete* planar graphs around the five events in slice decomposition. The planar graphs are incomplete because only nodes and edges around the events are shown. Moreover, some edges are shown with only one endpoint.[2]

[2]The two nodes that an edge connects are its endpoints.



(a)



(b)

Figure 4.23: The two obstacles create a very narrow gap. Unless there is a sweep line position in the exact centre of the gap, no sweep line will extend to the region between the two obstacles.

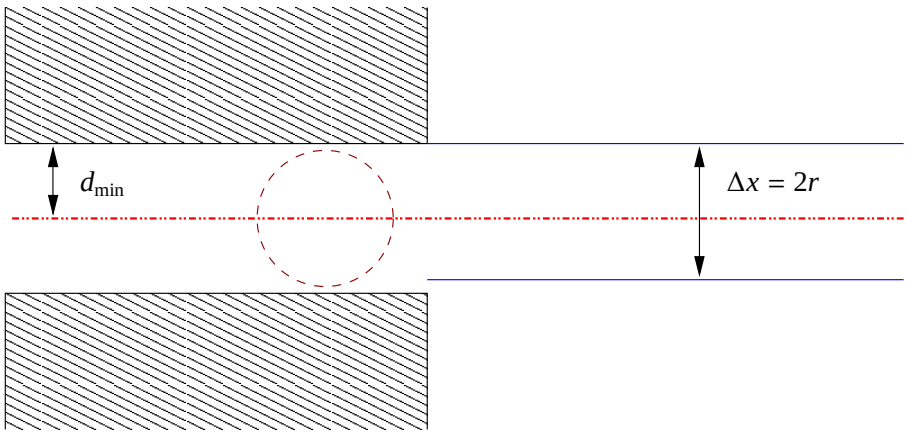


Figure 4.24: Similar to the merge event, an extra strip $d_{\min}$ below the obstacle is added.

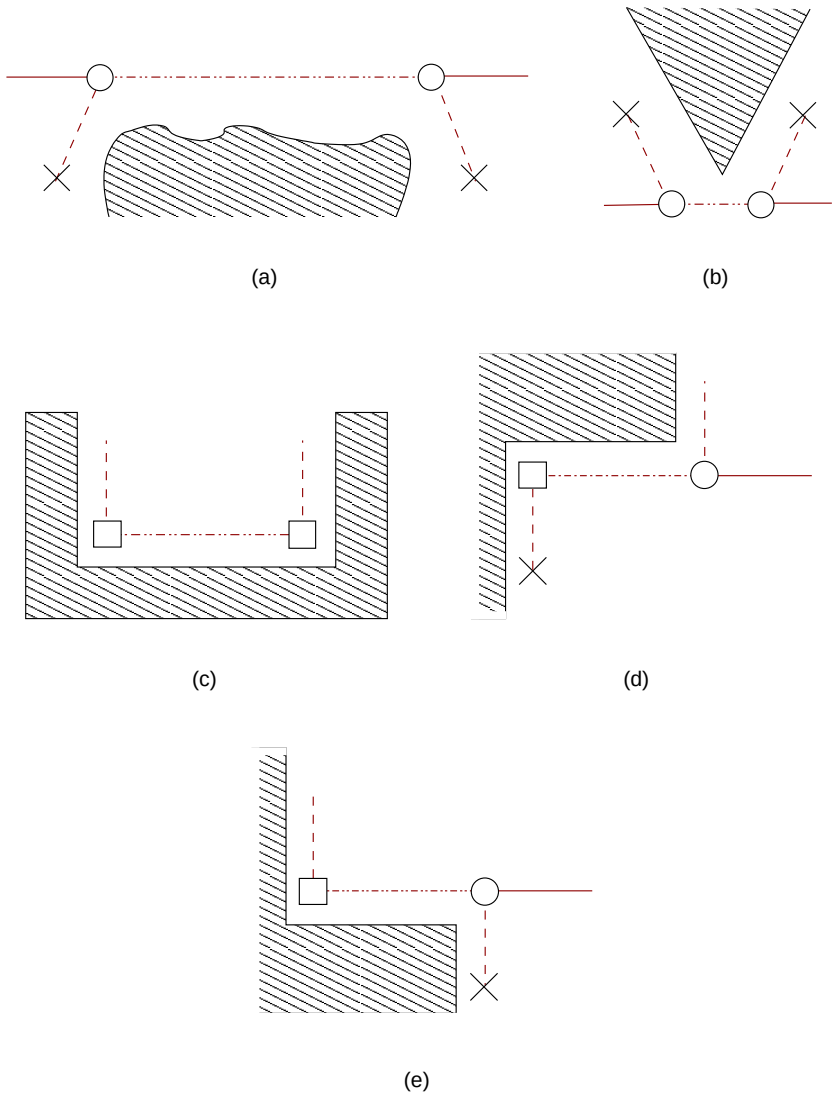


Figure 4.25: Events in slice decomposition can be represented with planar graphs: (a) split, (b) merge, (c) end, (d) lengthen, (e) shorten.

The planar graph constructed is not only a representation of the slice decomposition, but also a qualitative topological map [64] of the environment. This is because the landmark nodes are also recognisable[3] places of the environment. Here, the term landmark is used to refer to large features in the environment such as walls and corridors. These landmarks are not features independent of any particular coordinate system. This use of landmark is similar to Mataric in [72].

4.3.1 Nodes

Two pieces of information are stored with every node in the topological map — the node type and the edges incident to it.[4]

Four node types exist in the topological coverage algorithm — free space, obstacle uncovered and joint. Free space and obstacle nodes are landmark nodes and are directly related to events in slice decomposition (see Figure 4.25). Free space nodes have obstacles on two adjacent sides. They are called free space nodes because corners are related to free space segment disappearance. Obstacle nodes are associated with threshold drops and rises in the middle of strips. They are called obstacle nodes because of their relationship to obstacle segment changes. Uncovered nodes represent uncovered cells in the environment. Joint nodes connect the landmark nodes (free space and obstacle nodes) to the existing map. In other words, they join the graph segments shown in Figure 4.25 to the existing map. Section 4.3.3 explains the functions of the different types of nodes in more details using four examples.

Other than its type, each node also remembers the edges incident to it. An uncovered node has only one edge incident to it, a free space node has two, a joint node has three, and the obstacle node can have three or four.

4.3.2 Edges

Edges in the map are cell boundaries in slice decomposition. Each edge in the map has a type and an approximate distance between its two endpoints. The five types of edges are open, vertical, north, south and corridor. Open edges are horizontal edges that have no obstacles above or below them. Vertical edges are for the right and left boundaries of a cell. North and south are horizontal edges that are immediately below or above an obstacle respectively. Corridor edges are ones with obstacles on both sides.

[3] Can be reliably detected by robots using sensors.
[4] An edge is *incident* to its endpoints.

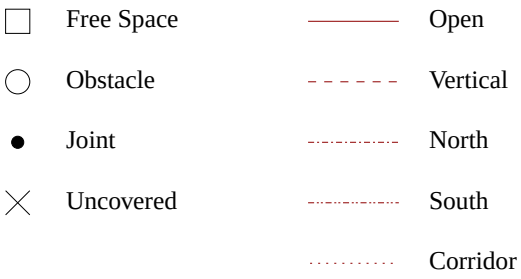


Figure 4.26: Symbols used for the different types of nodes and edges.

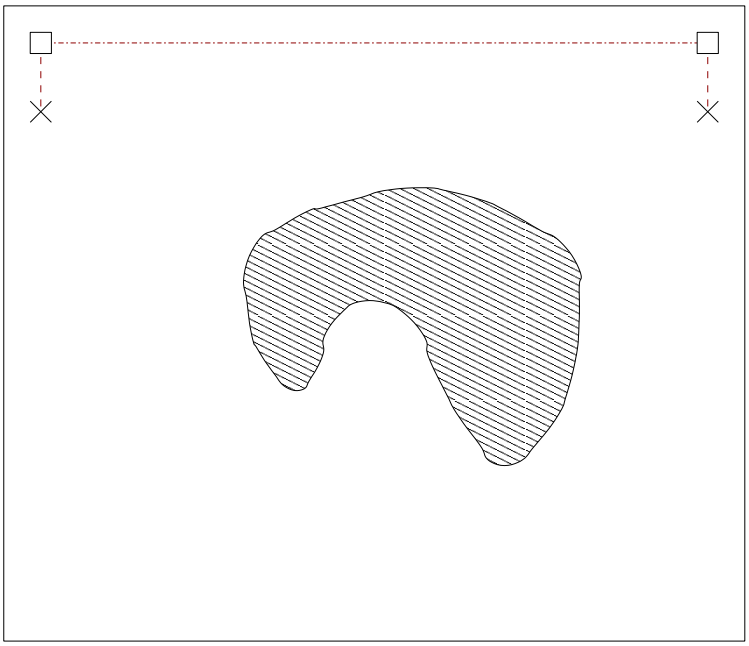


Figure 4.27: Sample environment.

4.3.3 Map updates

This subsection is arranged into four examples explaining how the topological map is updated. It also explains the functions and properties of the various types of nodes and edges. Figure 4.26 shows the symbols used in the diagrams in this subsection.

Example 1

The first example uses the obstacle seen in the examples in Chapter 3. This is shown in Figure 4.27 and there is only one obstacle in the environment. Cells that are detected but uncovered are represented on the topological map as a pair of uncovered nodes (×). Therefore, the map in Figure 4.27 contains only the uncovered free space cell at the top.

In Figure 4.28, the robot has covered the top free space cell and has arrived at the cell boundary. The distance to the obstacle drops below the threshold. A new obstacle node (○) is created.

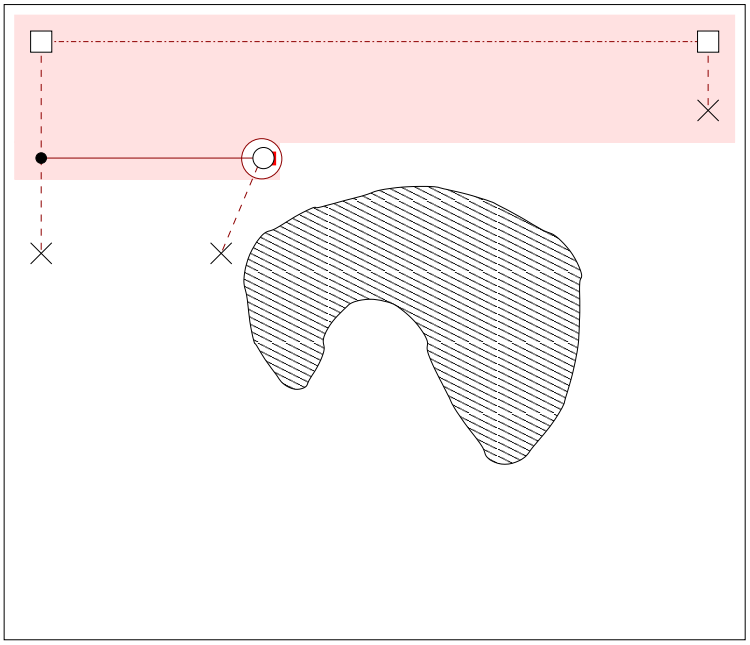


Figure 4.28: Detects a split event.

Since the obstacle node (○) is located in the middle of a strip, a joint node (●) is added to connect the obstacle node (○) to the rest of the map. Also, uncovered nodes (×) are connected to both the obstacle node (○) and the joint node (●). This is because the two nodes lead to the two side boundaries of the new free space cell underneath. Note that the newly found uncovered cell is also represented by a pair of uncovered nodes (×).

The type of edge used depends on the situation between the two nodes it connects. For example, in Figure 4.28, an open edge is used between the obstacle node (○) and the joint node (●) because there are no obstacles on either side of the edge. Vertical edges are used to connect the new uncovered nodes (×) as these edges are the side boundaries of the uncovered cell.

As the robot moves along the the cell boundary and reaches the point where the distance to the obstacle rises above the threshold, the map is updated as shown in Figure 4.29. Note that a south edge is used to connect the new obstacle node (○) to the existing map because of the obstacle below the edge. When the robot reaches the end of the strip, the uncovered node (×) representing the right boundary of the top free space cell is converted to a joint node (●) to complete the cell boundary. This is shown in Figure 4.30. It can be seen from the diagram that cell boundaries are represented as a horizontally connected series of nodes and edges. Also, the map updates that occur during the boundary exploration have converted the two uncovered nodes (×) in Figure 4.27. Now the nodes surrounding the boundaries of the top free space cell form a cycle [5] in the topological map.

[5] A cycle in a graph has positive length, its origin and terminus are the same, and its origin and internal nodes are distinct.

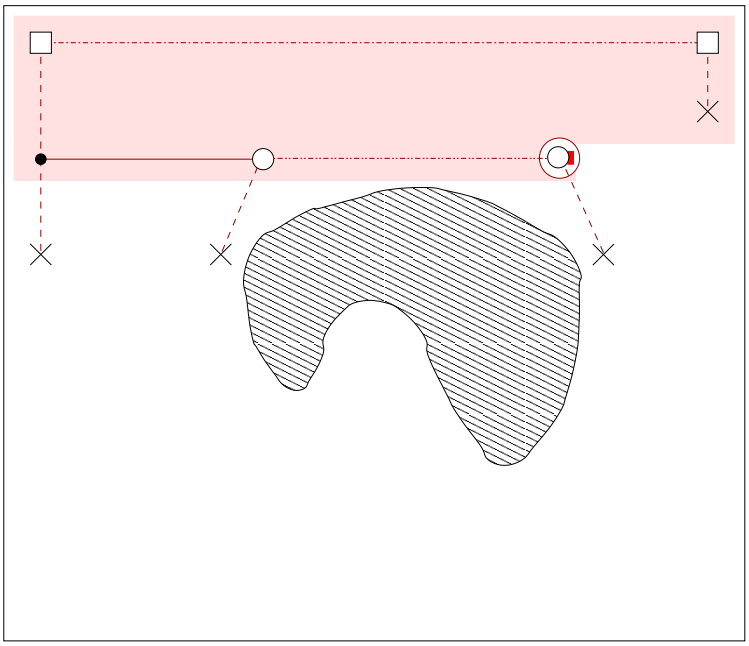


Figure 4.29: Reaching the other end of the obstacle segment in a split event.

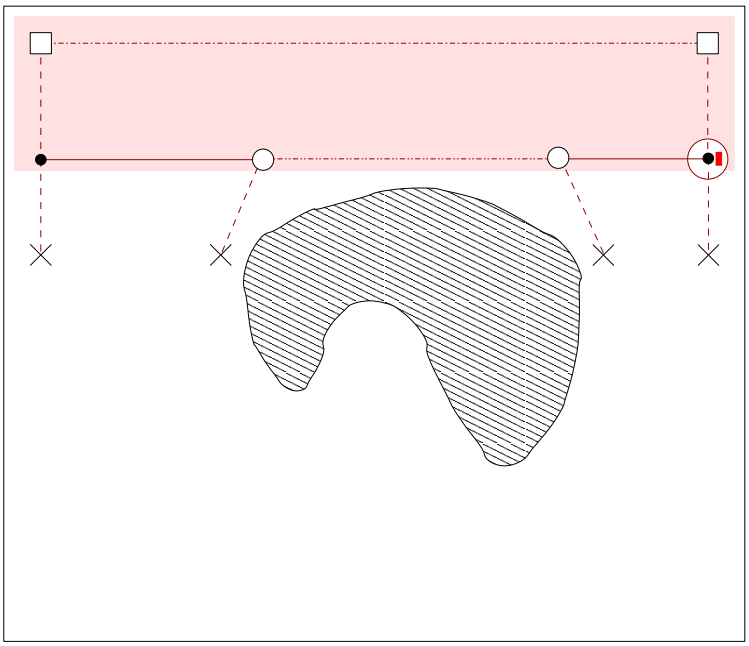


Figure 4.30: Finishing the cell boundary exploration. The nodes representing the top and bottom boundaries of the first free space cell form a cycle in the map.

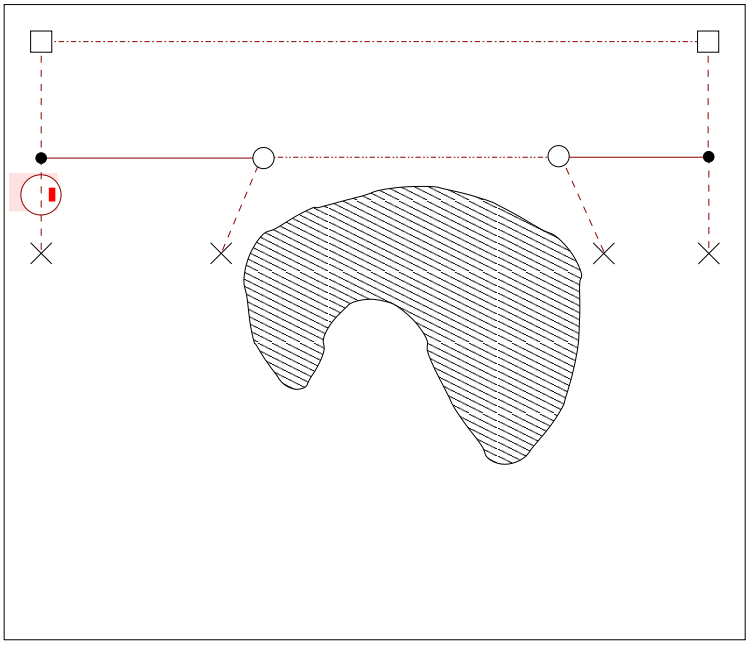


Figure 4.31: Covering another cell in the environment.

Assume that the robot now chooses to cover the cell to the left of the obstacle, as shown in Figure 4.31. It can be seen that this uncovered cell is represented by a pair of uncovered nodes (×). This pair of nodes are connected via a horizontal edge that is the top boundary of this cell. This arrangement is similar to the one shown in Figure 4.27. It is this configuration of nodes and edges that indicates the existence of an uncovered cell in the topological map.

When the robot reaches the bottom of this cell, the two uncovered nodes are converted into other types of nodes, as shown in Figure 4.32. The status of the current cell is now changed from uncovered to covered, since the nodes representing it now forms a cycle in the map. As the robot explores the cell boundary, it will discover the other end of the disappearing obstacle segment that causes the merge event (Figure 4.33). An obstacle node (○) is added for this landmark. An uncovered node (×) is also added for the new free space cell above the cell boundary. Figure 4.34 shows the robot completing the exploration and map updating for this cell boundary. At this moment, there are three uncovered cells in the topological map.

Figure 4.35 shows the robot reaching the top boundary of the free space cell. A free space node (□) is added here because the landmark is surrounded by obstacles on two adjacent sides. Since free space nodes are corners by definition, it is therefore always located at ends of cell boundaries. This is in contrast to obstacle nodes, which are always in the middle of cell boundaries. As a result of this, free space nodes are only incident to two edges, while obstacle nodes are incident to three.

When the robot reaches the other end of this cell boundary, the map is updated to indicate this cell is completely covered. This is shown in Figure 4.36.

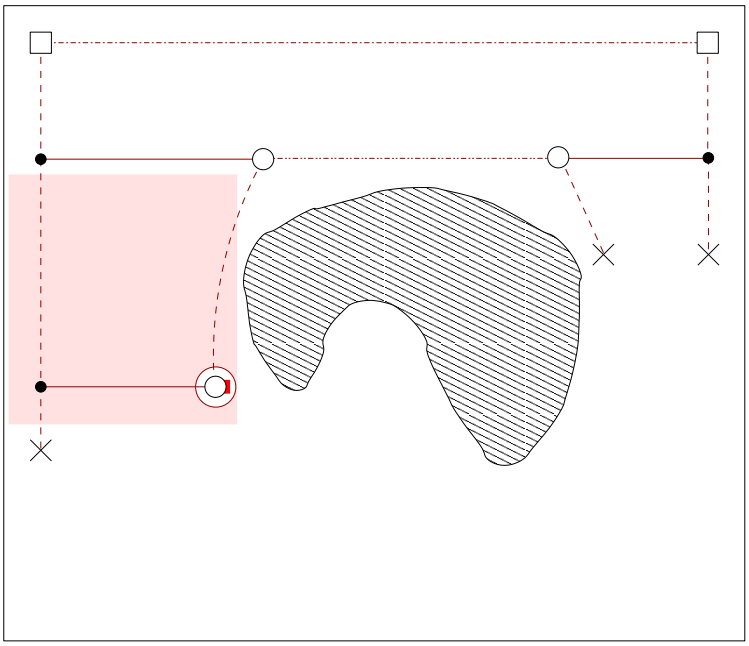


Figure 4.32: When the coverage of the current cell is finished, the uncovered nodes are converted to other types of nodes.

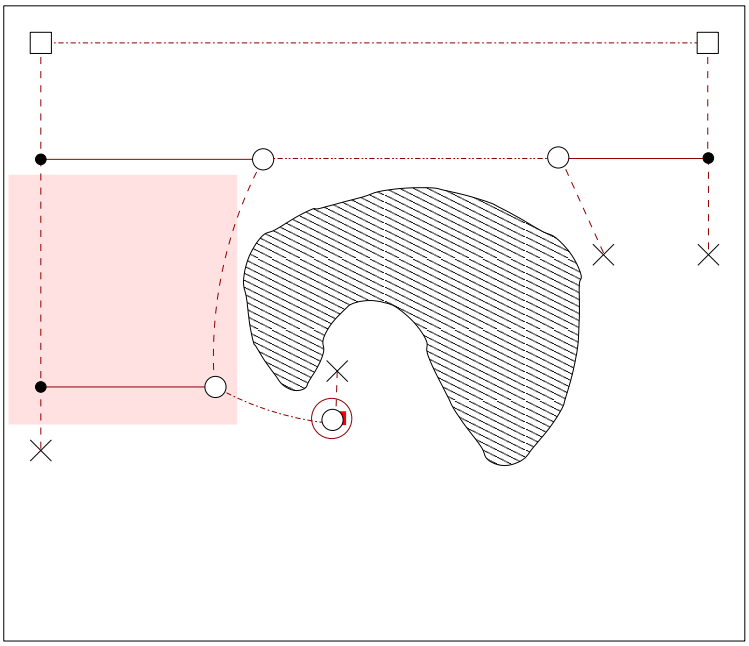


Figure 4.33: Discovering another free space cell.

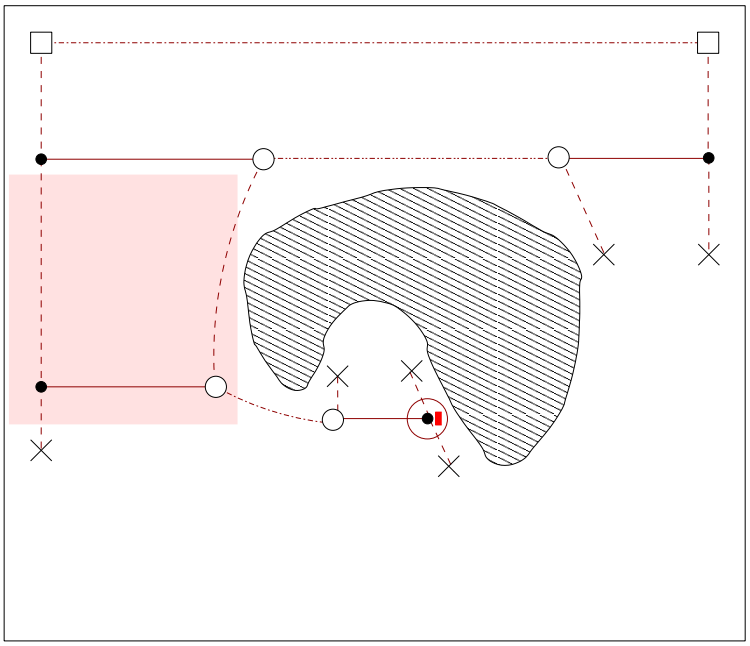


Figure 4.34: Just finished exploring the cell boundary.

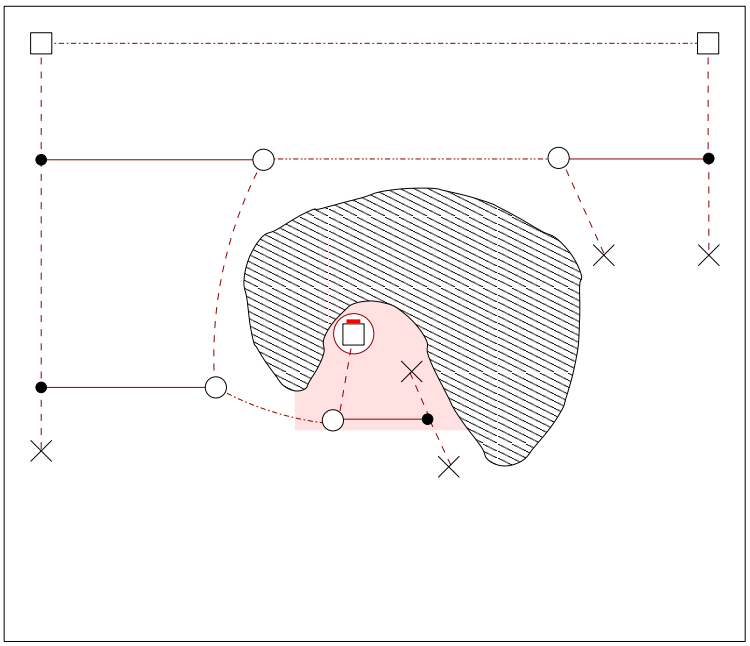


Figure 4.35: A free space node (□) is used here because the landmark is surrounded by obstacles on two adjacent sides.

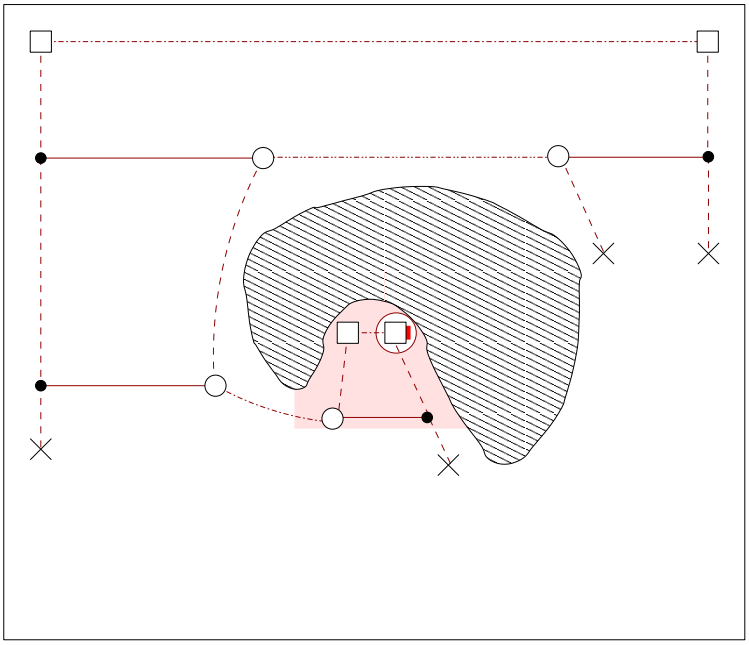


Figure 4.36: Completing the free space cell.

Example 2

Sometimes both the top and bottom boundaries of an uncovered cell have been exposed during two different boundary explorations. This happens around a free standing obstacle, when one side of the obstacles has been covered. Figure 4.37 shows a robot covering a cell where both top and bottom boundaries are known. Note that there are two sets of uncovered nodes (×), one for the top boundary, one for the bottom. This duplication of uncovered nodes (×) occurs because boundary exploration of both the top and bottom borders classify the cell as uncovered. The robot does not know that both borders belong to the same cell during the earlier boundary explorations. When the robot reaches the top border and arrives at the obstacle node (○), it is recognised as previously visited. The two sets of unexplored edges are merged to form a single cell as shown in Figure 4.38.

Landmark matching is done both metrically and topologically [64]. A list of possible matches is found by selecting nodes that are metrically close to the robot's position. Closeness is defined loosely as a region two to three times larger than the robot. If there is more than one possible candidate, a topological matching procedure is initiated. This involves comparing directions of obstacles around candidate nodes with the robot's current situation. Using the situation shown in Figures 4.37 and 4.38 as an example, lets assume the metrical match procedure returns the two obstacle nodes (○) above the obstacle. Using the knowledge that there are no obstacles on either side of the strip previously, it can be concluded that the obstacle node on the right is a match by analysing the topology around the two nodes. The node on the right is connected to an open edge in the direction behind the robot, while the one on the left is connected to a south

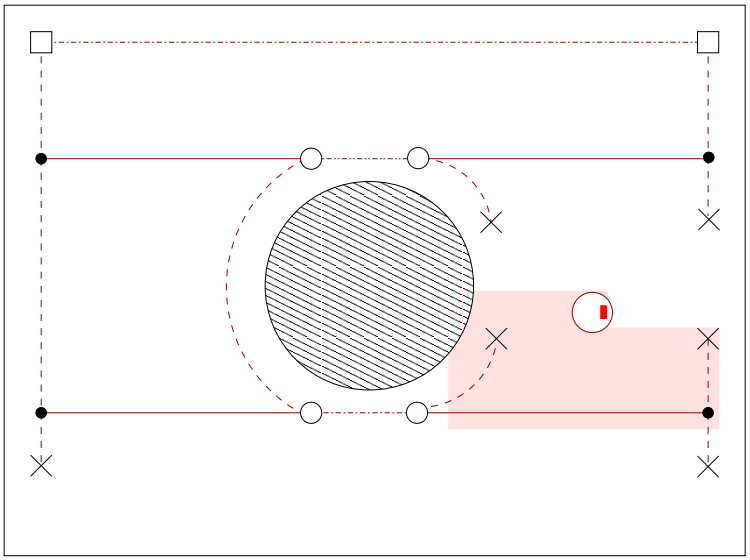


Figure 4.37: An uncovered cell where both top and bottom boundaries are known.

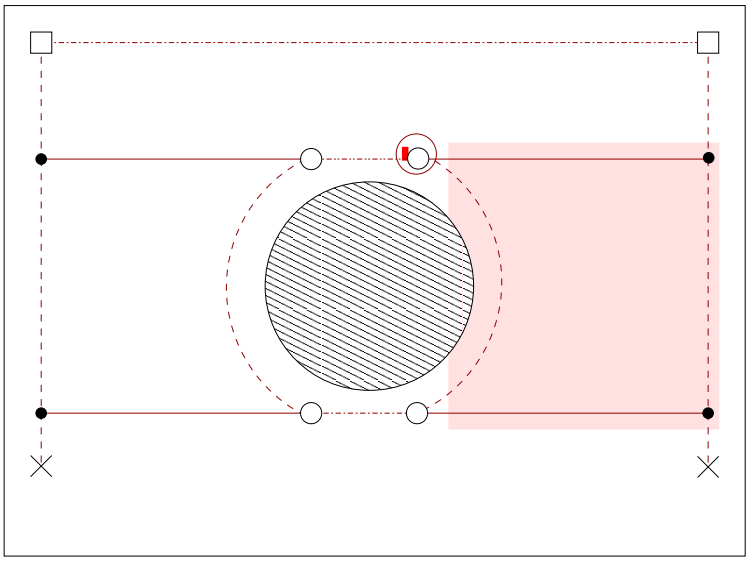


Figure 4.38: The obstacle node (○) is identified as the current landmark by matching metrically and topologically.

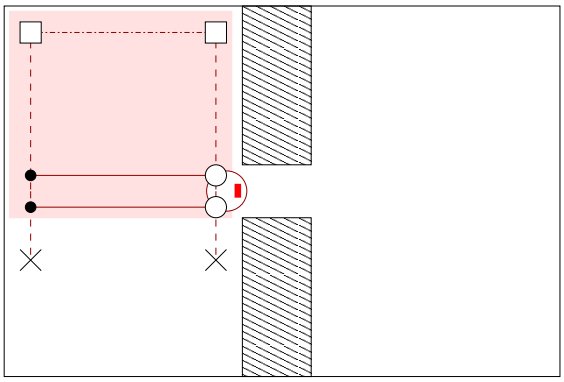


Figure 4.39: Two obstacle nodes (○) are added for the merge and split events. Because the two events occur at the same sweep position, the “cell” between the events is considered covered already.

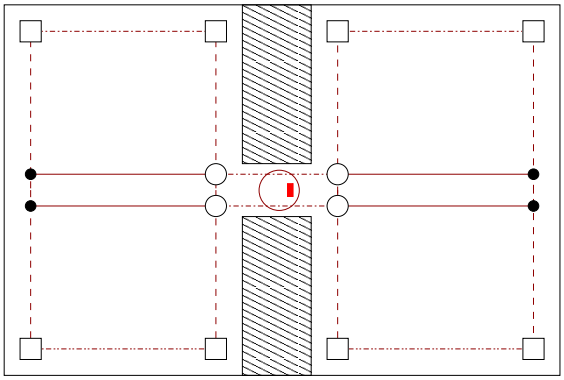


Figure 4.40: Completed topological map for this example.

edge.

Example 3

Section 4.2.6 discussed the situation where a merge and a split event happens in the proximity of one another. Figure 4.39 shows a robot encountering such a situation. Here, the gap between the top and the bottom obstacle is quite small, and the robot can just pass through. Since the robot detected both a split and a merge event, two obstacle nodes (○) are added. However, uncovered nodes (×) are added only for the lower set of joint (•) and obstacle (○) nodes. This is because the two events actually occur at the same sweep position (there is only one sweep position through the gap), and therefore the “cell” between the two events is already covered. Figure 4.40 shows the completed topological map for this example.

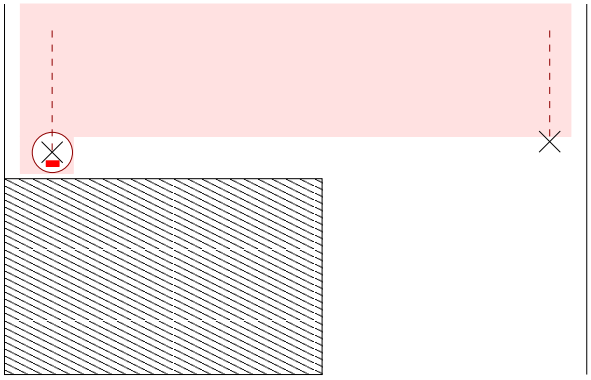


Figure 4.41: Reaching a new node that does not lead to any uncovered cells.

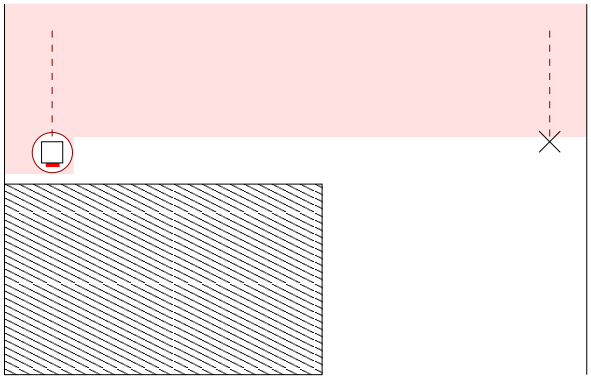


Figure 4.42: The uncovered node (×) is converted to a free space node (□).

Example 4

In the previous examples, all the free space nodes (□) share cell boundaries with only free space nodes (□). This example shows a situation where a free space node (□) shares a cell boundary with other types of nodes. This occurs in shorten and lengthen events. Figure 4.41 shows a robot arriving at a new free space node (□). The uncovered node (×) is converted to a free space node (□), as shown in Figure 4.41. No uncovered node (×) is added to the free space node (□) because it does not lead to any free space cell underneath. This is because the free space segment at that end becomes an obstacle segment in the next sweep line position. The obstacle segment ends and free space segment from the new uncovered cell underneath starts at the position shown in Figure 4.43. The representation of the completed cell boundary is shown in Figure 4.44.

4.4 Travel between cells

When the robot finishes covering the current cell, it searches the topological map for the closest uncovered node from its current position. Since the goal location is unknown, no heuristic

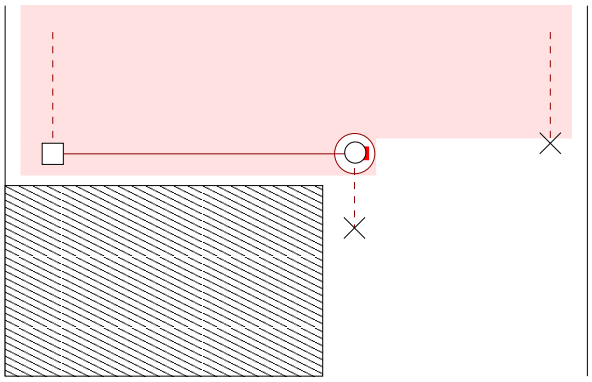


Figure 4.43: The free space cell underneath is connected from this position.

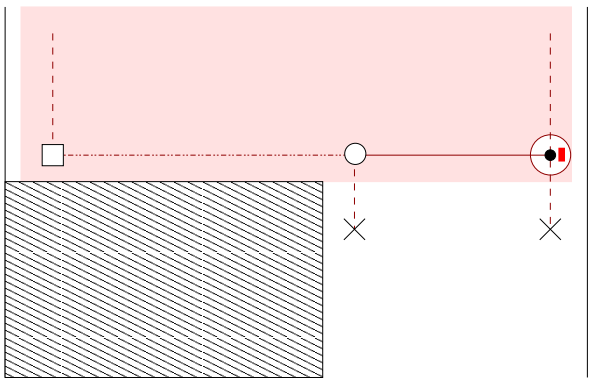


Figure 4.44: The cell boundary is completed.

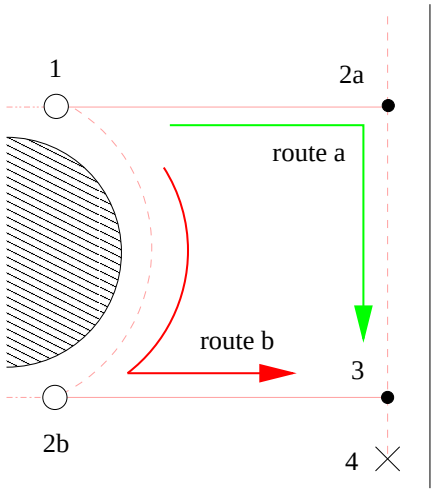


Figure 4.45: Results of a breadth-first search on the map in Figure 4.38. The current node is node 1, and the closest uncovered node found is node 4.

information is available relating each node to the goal node. As a result, a blind search[6] has to be used. A standard breadth-first search assumes that the length of a route from one node to another is equal to the number of edges in the path. To take advantage of the approximate distance information stored with the edges, a modified breadth-first search [92] can be used instead. This modified breadth-first search favours nodes with shorter distances from the start node.

The graph search returns a list of nodes from the current node to the selected uncovered node. The robot then moves from node to node until arriving at the chosen uncovered node. Similar to the landmark matching in the boundary state, the robot determines if it has arrived at the target node using a combination of metric and topological matching. When the robot moves into the region close to the target, it uses sensor and temporal information to decide when it has arrived at the node. Temporal information comes from previous sensor readings and actuator commands. The match is qualitative and there is no guarantee the robot will arrive at the exact location as previously. However, an exact match of locations from previous visits is not necessary. This is because cells in slice decomposition are bounded by obstacle boundaries, not the exact location of where these obstacles are first detected.

Using the situation in Figure 4.38 as an example, a standard breadth-first search will return either of the paths shown in Figure 4.45, depending on the search order. This is because both paths link the current node to an uncovered node ($\times$) via three edges. The only difference between the two routes is the second node (2a and 2b) on the route.

If route a is returned by the search, the robot turns around and moves forward until arriving at the left boundary of the environment. It then turns right (from a geocentric point-of-view) and

[6] A blind search does not use a cost or evaluation function to bias the search to move more quickly toward the goal.

follows the boundary. The length of the edge between node 2a and 3 is estimated by the number of strips it took the robot to arrive from the bottom to the top of the cell. As a result, it is not very accurate. The uncovered cell represented by node 4 is assumed to start at the position of node 3. Therefore, after the robot arrives at the approximate position of 3, it turns right again and starts moving towards the direction of the obstacle. The sweep position should be very close to, but underneath the obstacle. When the robot gets closer to the obstacle, it can adjust the sweep position accordingly and starts covering the chosen cell in the normal state of the topological coverage algorithm.

If route b is returned, the robot will follow the obstacle until it arrives at node 2b, which is at the bottom of the obstacle. It then turns around and moves towards the left boundary of the environment. When it arrives at the wall (node 3), it turns around again, switches execution of the algorithm to normal state and begins covering the cell.

4.5 Completeness

It is important to show that an online coverage algorithm is complete, that it can fully cover a given environment. This section attempts to prove the topological coverage algorithm can always find and cover all reachable surfaces in unknown environments.

Slice decomposition subdivides free space in the environment into a collection of disjoint cells. Two cells are said to be *directly* connected if they share a common cell boundary. Two cells are *indirectly* connected if there is a series of directly connected cells that link the two. A robot can always find a path between two directly or indirectly connected cells. Therefore, the reachable surface in an environment is the union of all cells directly or indirectly connected to the initial cell.

The topological coverage algorithm achieves complete coverage by creating a slice decomposition of the environment and covering all cells in the decomposition. Since cells in slice decomposition can be covered by a robot following a zigzag path, the problem of proving complete coverage can be simplified to showing that all reachable cells are guaranteed to be discovered and visited. It is assumed that the robot has range sensing ability.

Lemma 4.5.1. *All topology changes in the environment are always detected.*

Proof. The inter-strip distance Δx is always set to ensure a robot following two consecutive strips can fully cover all surfaces between the two strips. No surfaces are left uncovered between consecutive strips. Therefore, even robots equipped with only bump sensors can detect obstacles that appear within the length of the strips. In other words, all obstacles to the side of the robot

will be detected by following a zigzag path.

For all the events in slice decomposition, on top of detection by range sensors, there is a fall back to simple detection by odometry. In split (Figure 4.8) and shorten (Figure 4.20(a)) events, an obstacle appears in the middle of the strip and blocks the path of the robot. This causes a sharp decrease in the strip length compared to the previous strip. In end (Figure 4.16) and shorten (Figure 4.22) events, the robot's path is blocked while moving from one strip to the next. In merge and lengthen events, the obstacle appears on the side of the robot in the direction where it has already swept. The obstacle does not "block" the path of the robot, rather, it "opens" the area underneath. This creates an huge increase in the length of the current strip compared to the previous one. This increase is larger than can be attributed to sensor (odometry) error.

The only situation where the robot may not pass an obstacle by its side is when two events occur in the proximity of each other, creating a small opening. This is discussed in Section 4.2.6. Since the robot will only encounter the criticality on its front or back, there is no fall back to detection by odometry. In other words, detection has to be done purely via range sensing. If the robot can pass through the gap, it must be at least of width $2d_{\min}$. (4.1) gives the maximum value of Δx at $2r$. (4.2) states that r is always smaller than $d_{\min}$. Therefore, the step size Δx is always smaller than the minimum gap width $2d_{\min}$. For this reason, part of the robot will always pass in front of the gap independent of the sweep line positions. Therefore, a robot is capable of detecting such a gap even with short range sensors. □

The cell boundary where the robot enters a cell is called the *opening* cell boundary. The *closing* cell boundary is the boundary the robot reaches when it finishes covering a cell.

Lemma 4.5.2. *The closing cell boundary of any visited cell is always found.*

Proof. This follows directly from Lemma 4.5.1. A robot following the zigzag coverage path will always be able to detect all topology changes happening in the cell. □

When the robot arrives at the closing cell boundary, it is programmed to fully explore the boundary reached. This full exploration of cell boundary has two goals. First it completes the coverage of the current cell and removes all uncovered nodes associated. Secondly, it exposes all other cells that are connected to the same boundary. The following Lemma guarantees all cells sharing the closing cell boundary of a visited cell are added to the topological map.

Lemma 4.5.3. *All cells that share the same cell boundary to the closing cell boundary of a visited cell are detected.*

Proof. This also follows directly from Lemma 4.5.1. A robot moving along the strip of the closing cell boundary can always detect all topology changes at that sweep position. □

When the robot finishes exploring a closing cell boundary, it searches the topological map for any uncovered cells. Since the topological map is a connected graph, travelling to any uncovered node is a simple matter of finding a path between the current node and the chosen uncovered node in the connected graph.

Lemma 4.5.4. *All cells that are added to the topological map will be visited.*

Proof. Topological coverage algorithm continues until the map does not contain any uncovered cell. Therefore, all cells that are added to the map will be covered before the algorithm finishes. □

Proposition 4.5.5. *All reachable cells in slice decomposition are covered.*

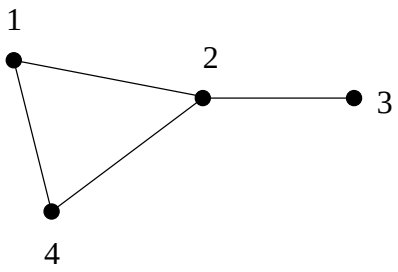
Proof. Starting from the initial cell, all reachable cells are either directly or indirectly connected. Lemma 4.5.3 guarantees that all the reachable cells are detected. Lemma 4.5.4 guarantees that all these reachable cells will also be visited. Lemma 4.5.2 shows the robot will always reach the closing boundary of the cell it is covering. Therefore, all reachable cells in the environment are detected, visited and covered. □

4.6 Complexity

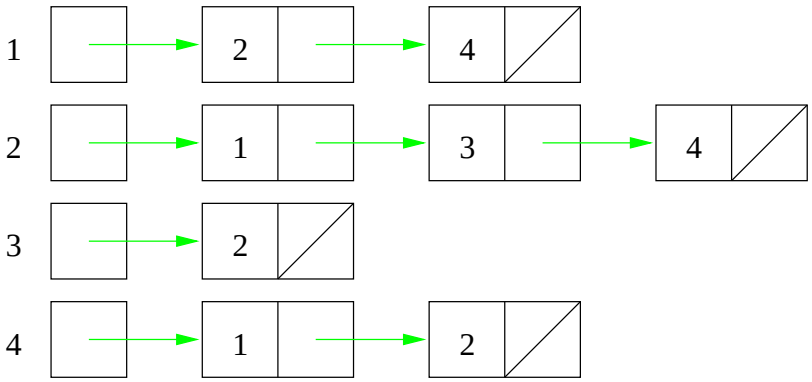
The topological map is a bi-directed graph $G(N, E)$, where N is a list of n nodes, and E is a list of e edges. Graphs are generally implemented as adjacency lists [53, 73, 93]. An example of representing a graph as adjacency lists is shown in Figure 4.46. The space complexity is therefore $O(n + e)$.

Topological map updates involve insertion and deletion operations on graphs. These are constant time operations because insertion and deletion on linked lists are constant time [73].

Searching for an uncovered cell on the topological map is done using breadth-first search. An implementation of breadth-first search adapted from Tanimoto [92] is shown in Algorithm 4.4. The algorithm requires the topological map $G(N, E)$, a start node, and a goal node as input. In the coverage algorithm, the start node is the current node the robot is situated at, while the goal node is any node of type uncovered. In the algorithm, *predecessor* is an array, while *open*, *closed* and *path* are all linked lists. Initialisation of the predecessor array in lines 1 - 3 takes $O(n)$. Initialisation of the linked lists in lines 4 - 6 is $O(1)$. There are two nested loops between lines 7 - 27, with the inner loop starting at line 21. The loop terminates either at line 9 when no goal node (uncovered nodes) is found; or at line 19, when the first empty node is opened. Therefore, the outer loop runs $O(n)$ times. The inner loop runs once for each node adjacent



(a)



(b)

Figure 4.46: (a) A directed graph. (b) Implementation as adjacency lists. Each node in the graph has a list of nodes it is connected to.

to the opened node n (line 21). From the adjacency lists, it can be seen that the inner loop iterates $2e$ times. Therefore the total time the nested loop iterates is $O(\max(n, e))$. Normally, there are more edges than nodes in a graph. Thus the time complexity of the nested loop can be simplified to $O(e)$. Combining the loop and the initialisation, the running time of the algorithm can be expressed as $O(n + e)$. (In the less common scenario where there are more nodes than edges, the running time is $O(n)$).

Algorithm 4.4 Standard Breadth-First Search

Require: $G(N, E)$, start node, goal node

```
 1: for each node n in N do
 2:    predecessor(n) ← −1
 3: end for
 4: open ← start node
 5: closed ← ∅
 6: path ← ∅
 7: loop
 8:    if open = ∅ then
 9:       return ∅
10:    end if
11:    n ← open(0)
12:    open ← open - n
13:    closed ← closed + n
14:    if n = goal node then
15:       repeat
16:          path ← path + n
17:          n ← predecessor(n)
18:       until n ≠ −1
19:       return path
20:    end if
21:    for each node m adjacent to n do
22:       if m ∉ open and m ∉ closed then
23:          open ← open + m
24:          predecessor(m) ← n
25:       end if
26:    end for
27: end loop
```

4.7 Summary

The topological coverage algorithm enables a robot to completely cover any unknown environment by incrementally constructing, updating, and storing coverage information in a topological map. The topological map shares the same set of landmarks with slice decomposition II.

Therefore, it is both a qualitative representation of landmarks in the environment, and a slice decomposition of the same space.

It is very difficult to mark particular regions in the environment as covered in a topological map. This is due to the qualitative nature of the representation. The nodes and edges of the map do not correspond to specific locations in space. This difficulty in storing coverage information within a topological map is overcome by embedding slice decomposition within the map. Even though individual nodes in the map are still not associated with specific areas of space, a combination of nodes now defines a region (a cell) bounded by obstacles.

Completeness and complexity of the topological coverage algorithm are also discussed. Completeness is important because an online coverage algorithm must completely and fully cover all the reachable surface. On the other hand, complexity was not an important issue in practice. Empirical observation shows that the robot spends a lot more time moving through the environment than searching the topological map. There was never any noticeable reduction in performance due to computational requirements of the coverage algorithm.

Newton's Third Law of Graduation: For every action towards graduation there is an equal and opposite distraction.

Jorge Cham, "Piled Higher and Deeper", www.phdcomics.com

5

Performance metrics

Performance metrics allow quantitative evaluation of results from experiments. They also provide a way to compare different experiments, or algorithms, meaningfully. There are two questions commonly asked about coverage operations [104]. Firstly, how much of the environment is covered or missed? Secondly, how much time is wasted on revisiting area that is covered already?

The first question can be answered with a measure of the effectiveness of the operation. In simulation, this is commonly measured by calculating the percentage of grid cells covered [45]. In real robot experiments, existing approaches to estimating percentage coverage include sprinkling sawdust [97] and using the coverage factor [22]. Neither methods produce a good estimate of the percentage coverage for real robot experiments.

The second question is an inquiry about the efficiency of the operation. For simulated experiments, Gabriely *et. al.* uses the number of repeatedly covered grid cells [45]. There are two minor flaws with using repeatedly covered cells as a metric. Firstly, a repeatedly covered cell maybe covered more than twice. Secondly, the figure is not normalised against the total number of grid cells in the environment. There are no existing measure of efficiency for real robot coverage experiments.

This chapter presents the two metrics that will be used to measure the performance of coverage experiments in this thesis. Section 5.1.1 defines the effectiveness metric, which is the same

as the one used by Gabriely *et. al.* [45]. The efficiency metric defined in Section 5.1.2 is an improvement over the use of repeatedly covered cells. The metric not only estimates the area of re-coverage, it is also normalised against the actual area covered by the robot.

These metrics are useless if the data they require is difficult or impossible to obtain. Therefore, the rest of the chapter describes practical methods for obtaining and estimating the data needed. Section 5.2 presents methods for estimating the metrics with grid-based simulation environments. Section 5.3 describes estimation in real robot experiments using computer vision techniques.

5.1 Metrics

5.1.1 Effectiveness: percentage coverage

The effectiveness of a coverage algorithm is the amount of the total surface covered by a robot running the algorithm. Therefore,

$$C = \frac{\text{Area of surface covered}}{\text{Total reachable surface area}} \tag{5.1}$$

The coverage metric C calculates the percentage coverage of an experiment. It requires the estimation of the area of surface covered in an experiment and a measure of the area of reachable surface in the environment.

5.1.2 Efficiency: path length

The efficiency of a coverage algorithm can be measured by the length of the path taken to completely cover an environment. The path length is related to the time spent on the coverage task if the robot moves in a fairly constant speed.

To make the metric environment independent, it is necessary to normalise the actual path length $||P_a||$ travelled by the robot. An ideal solution would be to divide the actual path length $||P_a||$ by the length of the optimal path. The optimal path is the shortest path possible to cover an environment if the robot starts with a map. Any path generated by an online algorithm, such as the topological coverage algorithm in this thesis, will be longer than the optimal path.

However, Arkin *et. al.* has shown that finding the optimal coverage path is an NP-hard problem [13]. They proved this by formulating coverage as a travelling salesman problem, with each location in the reachable area as a city. As a result, the actual path P_a taken by the robot

is compared to the *minimal path* P_m instead. The minimal path P_m is the shortest coverage path for a mobile robot that can teleport with no cost associated with the teleport operation. All environments can be covered by such a robot with no retracing. The minimal path P_m is therefore always equal to or shorter than the realisable optimal path. In summary, the path length taken is normalised by

$$L' = \frac{||P_a||}{||P_m||} \tag{5.2}$$

where $||P_a||$ and $||P_m||$ are the Euclidean distances for the actual path and the minimal path respectively.

However, (5.2) does not take into account the amount of coverage C achieved in the experiment. This is important as the experiment may not attain 100% coverage, and P_m is the minimal path for covering the *entire* space. If the robot covers only 50% of a given environment, then P_a should be compared with 50% of P_m instead. The minimal path P_m of the area actually covered in an experiment is

$$P_m \times \frac{\text{Area of surface covered}}{\text{Total reachable surface area}} = P_m \times C$$

Therefore, (5.2) becomes

$$L = \frac{||P_a||}{||P_m|| \times C} \tag{5.3}$$

(5.3) is the path length metric used for measuring efficiency of coverage experiments. The length of the actual path P_a can be calculated from wheel encoders' readings. The minimal path P_m depends on the configuration of the environment.

5.2 In simulation

The simulation environment is assumed to be a uniform grid with square cells. Therefore, area in simulation can be measured in numbers of grid cells. (5.1) can be rewritten as

$$\begin{aligned} C &= \frac{\text{Area of surface covered}}{\text{Total reachable surface area}} \\ &= \frac{\text{number of grid cells covered}}{\text{number of reachable grid cells}} \end{aligned} \tag{5.4}$$

The number of grid cells reachable is calculated with the distance transform [58]. Figure 5.1 shows the distance transform of a very small and simple environment. It is assumed that the robot is the same size as a grid cell. Also, the robot keeps a distance of at least one grid cell from any obstacles. Starting from the initial cell (marked 0), all 8 neighbours of a marked cell are

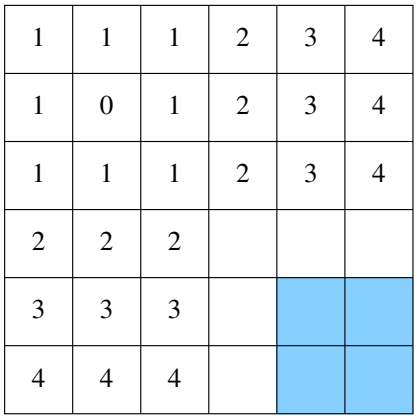


Figure 5.1: Calculating the total number of reachable grid cells using the distance transform.

tagged with a number one higher than itself. This tagging propagates until all cells not occupied by an obstacle, or a neighbour of an obstacle, are marked. The total number of reachable grid cells is then the number of cells marked, or numbered, by the distance transform.

Since the only piece of information needed is whether a path exists between any given grid cell and the initial grid cell, the distance transform is an overkill for this application. A simple filling algorithm [40], like the one shown in Figure 5.2, already suffices. However, both the distance transform and the filling algorithm require similar amounts of computation and coding time. Therefore, it does not make any difference which of the two methods are used to calculate the number of reachable grid cells.

The other parameter in (5.4), the number of grid cells covered, is found by counting grid cells after the simulation is finished. Only reachable grid cells covered are included in the total. If any non-reachable grid cells are covered, they are simply ignored in the count.

The path length measure from (5.3) is reformulated as

$$\begin{aligned} L &= \frac{||P_a||}{||P_m|| \times C} \\ &= \frac{\text{number of moves}}{\text{number of reachable grid cells} \times C} \end{aligned} \tag{5.5}$$

The actual path length is calculated as the number of moves the robot has made. This is similar, but not the same, as the number of cells in the robot's path. The difference is in the handling of multiply visited grid cells. Here, a grid cell that has been visited n times occupies n steps in the path.

The number of reachable grid cells is used as the length of the minimal path. This comes directly from the definition of the minimal path, which is the path length needed to achieve complete

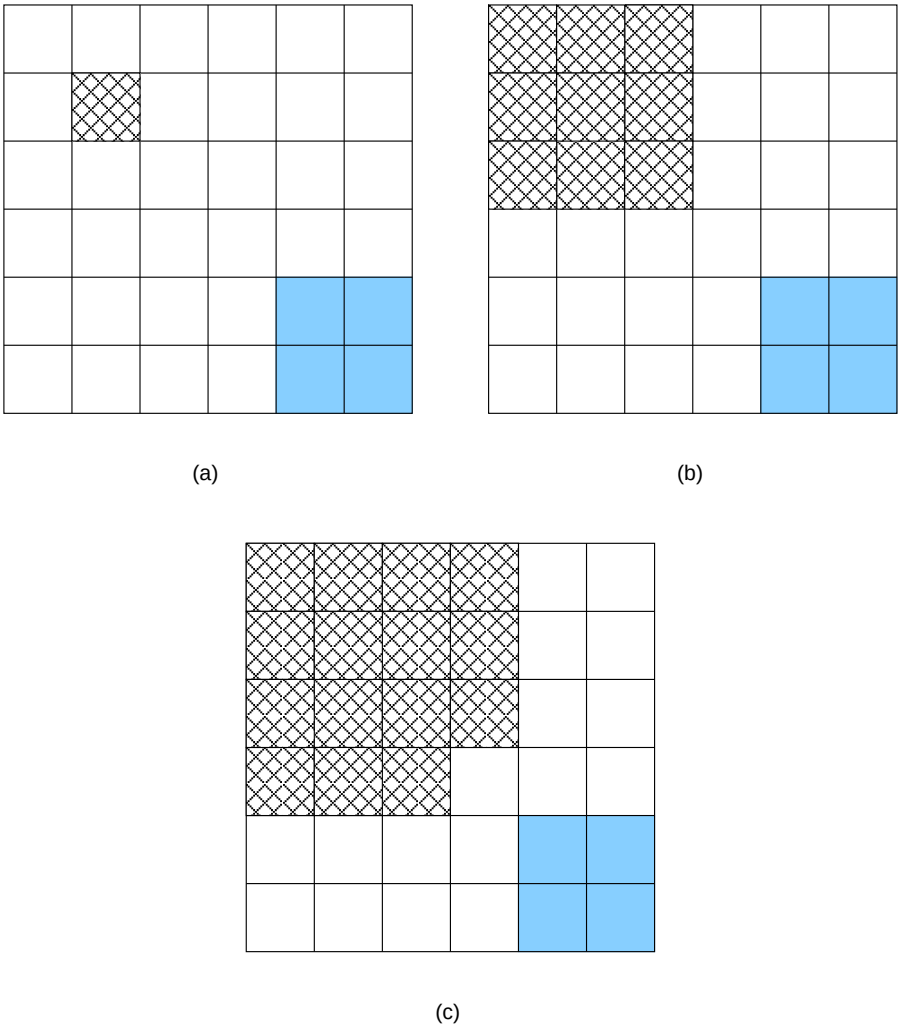


Figure 5.2: Calculating the total number of reachable grid cells with a a simple filling algorithm. (a) The algorithm starts with marking the initial cell. (b) In the next step, all the neighbours of the initial cells are marked. (c) Then all the neighbours of the grid cells marked in the previous steps are marked as well.

coverage with a teleport robot. With no cost to teleportation, such a robot can cover n grid cells in n steps. In other words, if the environment has n reachable cells to be covered, the minimal path would also be n.

5.3 In real robot experiments

5.3.1 Creating composite images

To calculate the amount of coverage C, an estimate of the area covered by the robot is required. It is important not to calculate this area from robot's positions estimated from dead reckoning. This is because dead reckoning attempts to estimate the robot's *position* based on the distance it travelled in its current direction from its previous position. Consequently, accumulation of errors, especially in the orientation, can lead to a larger and larger discrepancy between the estimated and the actual poses.

Therefore, an external, independent perspective of the experiment process is required. A simple and cheap solution is to use a wall mounted camera to capture a movie of the robot's progress. The experiment setup used in this thesis is shown in Figure 5.3 and a sample image from the camera is shown in Figure 5.4. Computer vision techniques can then be employed on the images captured to estimate the area covered by the robot. Figure 5.5 shows a flowchart explaining the process. First, the original frames captured by the camera are combined to form a single composite image. This composite image illustrates all the surfaces the robot covered. Since the original images contain perspective distortion, this artefact is removed using a deskewing operation. The percentage coverage is then estimated from the resulting deskewed composite image.

In this thesis, a composite image of an experiment refers to a single image that shows the trail or path of the robot during the experiment. To create such an image, the position of the robot is first extracted from each frame captured by the camera. These positions are then superpositioned to form a single image.

Two methods have been devised for creating composite images [105]. The first method, image subtraction, requires very little computation. However, it suffers from two major disadvantages. Firstly, it is not particularly reliable as artefacts frequently appear in the background subtraction process. This is because of environmental factors such as lighting. Additionally, in the case of a tethered robot (such as the Khepera) the cable motion is computed as part of the final composite image.

The other method, evidence gathering, improves tolerance to poor lighting by matching a model

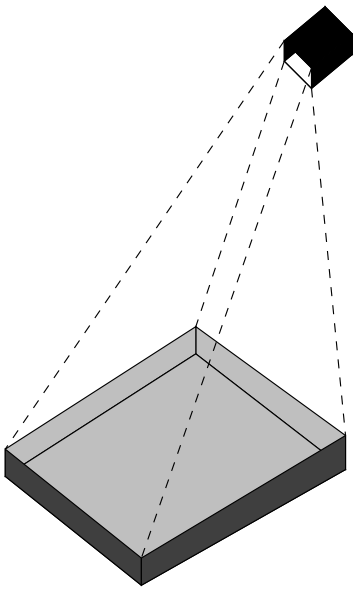


Figure 5.3: Mounted camera capturing frames of robot's movements.

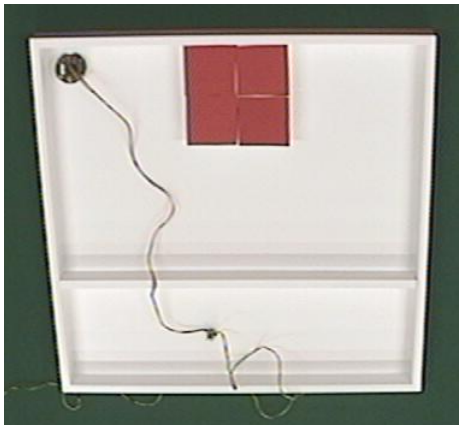


Figure 5.4: An image captured by the experimental setup shown in Figure 5.3.

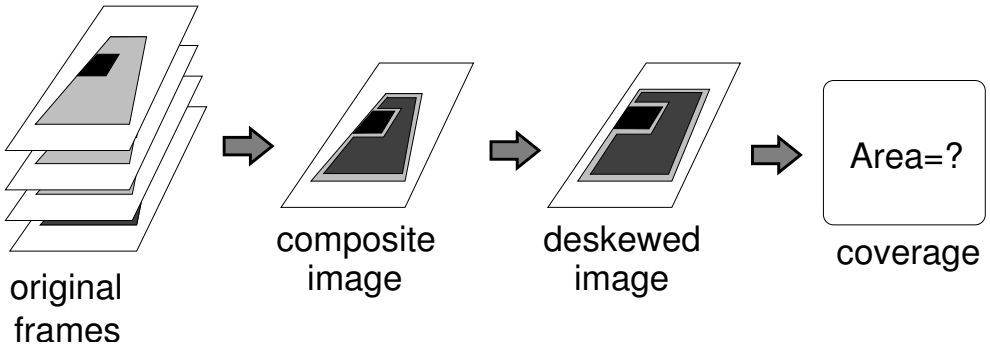


Figure 5.5: Estimating the percentage coverage from an image sequence.

of the robot using a Hough accumulator [79]. The major disadvantage of evidence gathering is the Hough transform used is very computation intensive. More details on the computer vision techniques employed in evidence gathering can be found in Appendix B.

Image subtraction

The first step in this method is to subtract a reference image r from each of the original images f_i in the sequence. Thus

$$s_i = f_i - r \tag{5.6}$$

As the images employed are full colour (RGB) images, this subtraction is computed for each colour channel separately. The difference s_i highlights the movement of the robot in the image sequence. This is because the robot is the only moving object in a static environment.

It is desirable that the reference image r be an accurate representation of the environment without the robot in it. Any error in the reference image will result in artefacts in s_i. If the experiments are very short, it is possible to use an image taken immediately before or after an experiment as the reference image r, as long as the robot is not in the scene.

However, if there are illumination changes within the scene, then a temporal average of the entire sequence should be used as the reference image r. A naïve approach would suggest using the average of the images. However, this will result in a ghostlike trail that corresponds to the motion of the moving object. Instead the median operator was employed. This computes the reference image as follows

$$r = \text{median}\Big(f_0, \cdots, f_{n-1}\Big) \tag{5.7}$$

for a sequence with n images f_0 to f_{n-1}.

In implementation, the median operation is computed via a histogram of image intensities at each point and finding the value corresponding to the 50th percentile.

After the subtraction in (5.6) is computed, the difference images s_i are thresholded pointwise with a step response. If the value of a pixel is greater than 127, the value is changed to one; otherwise if the value is smaller than 127, 0 is assigned instead. These thresholded difference images are then combined by performing a pointwise `OR` of the all the images. This methodology works as the main point of difference between successive frames is the robot. As the thresholded images are effectively a binary representation of this difference, then a logical `OR` will yield a resulting image which is a superposition of the robots locations.

Evidence Gathering - Hough Transform

The first step of evidence gathering is image subtraction using (5.6). The resulting difference images s_i are then edge detected to find the edge information for each colour channel. The edge map is computed using a Canny edge detector [25, 79], for two main reasons. Firstly, it is considered to have an optimal response for step edge responses, and secondly it can serve to reduce noise in the image. This noise reduction is useful as it can help remove small artefacts in s_i due to estimation problems in the reference image. The final edge map, e_i, is computed via weighted summation and thresholding of the individual colour channels. The weighting in the summation allows the emphasis of features in particular colour bands. The value of the threshold is tuned empirically.

The edge map e_i is then presented as evidence to a model of the robot. The specific model used depends on two factors — the shape of the robot; and how this shape changes because of perspective effects as the robot moves within the environment. For example, the Khepera robot was modelled as a circle. This is possible as at the extreme end of the enclosure, where the robot is at the maximum distance from the camera, the Khepera is still circular. In this case the model fitting must find the diameter that best fits the Khepera in each frame. A Hough transform is used to fit the model to the edge data. The specific model fitting algorithm is shown below:

Algorithm 5.1 Hough Evidence Gathering

```
for d = D_min to D_max do
  for all edge pixels (x, y) do
    for θ = 0 to 2π do
      x_c ← x − (d/2) cos θ
      y_c ← y − (d/2) sin θ
      if (x_c, y_c) is in image and (x_c, y_c) is an edge point then
        increment A(x, y, d)
      end if
    end for
  end for
end for
```

For a circular fit, the image is initially examined to find all the edge points. For each edge point, all the points are computed that are in a circle of diameter d and centred on this edge point. Diameter was employed instead of radius as it is easier to compute this from an image sequence. If the computed point is another edge point, then an array is incremented with a point indexed by the original edge point and the diameter.

After evidence is gathered from the entire image, the array $A(x, y, d)$ will contain peaks which correspond to likely circle centre points and diameters. Within this array the peak corresponds to the circle centre and diameter which occurs most often. This is the best candidate for the robot's

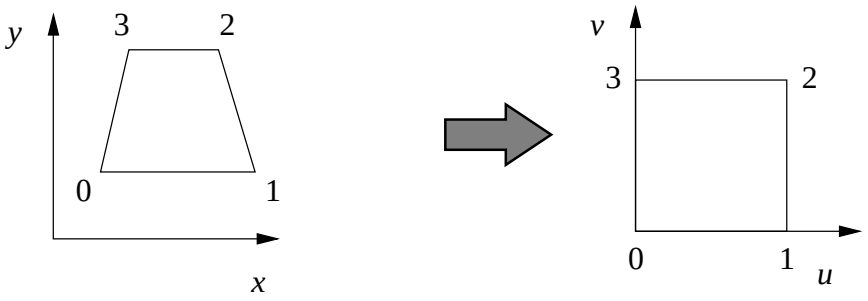


Figure 5.6: Quadrilateral to square mapping.

location within the frame. It can be computationally expensive to search all possible diameters so the possible search space is reduced by examination of the original image sequence. The purpose of this is to find estimates of the diameters of the robot at the maximum and minimum distance from the camera. This sets upper and lower bounds to the diameter parameter, $D_{\min}$ and $D_{\max}$.

The output of the model fitting stage will be a series of numbers which describe the central coordinates of the robot and the model chosen to describe the robot. By examining the results from all successive frames an accurate representation of the robot's path can be found. This, along with the model of the robot, can be used to produce a composite image for estimating the surface area covered by the robot.

5.3.2 Correcting perspective warp

The images captured by the camera show an effect known as perspective warping. This means that distant lines are shortened when compared to closer lines. This shortening effect makes it difficult to estimate the coverage area. Therefore, it is essential to correct for this effect. This can be performed using an inverse perspective transform to map the quadrilateral into a unit region as shown in Figure 5.6. This mapping makes the assumption that the original object, in this case the tray, is square in shape. The perspective transform described in this section follows the work of [99].

To make the mathematics simpler, this discussion will originally examine the reverse of the transform shown in Figure 5.6. The forward perspective can be written:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{5.8}$$

Here x, y, u and v are coordinates as shown in Figure 5.6. w is the perspective. It can be simply

written as $\mathbf{x}' = \mathbf{Au}$. Euclidean coordinates can be computed using equation (5.8) and performing the following substitutions :

$$x = \frac{x'}{w'} = \frac{a_{11}u + a_{12}v + a_{13}w}{a_{31}u + a_{32}v + a_{33}w} \tag{5.9}$$

$$y = \frac{y'}{w'} = \frac{a_{21}u + a_{22}v + a_{23}w}{a_{31}u + a_{32}v + a_{33}w} \tag{5.10}$$

For most applications w is generally 1.

For arbitrary systems (quadrilateral to quadrilateral) solving this equation will have 8 degrees of freedom (two for each point in the quadrilateral). However, for the situation here the uv plane is a unit square. Thus :

$$\begin{aligned}(0,0) &\rightarrow (x_0, y_0)\\ (1,0) &\rightarrow (x_1, y_1)\\ (1,1) &\rightarrow (x_2, y_2)\\ (0,1) &\rightarrow (x_3, y_3)\end{aligned}$$

This gives the following values for the elements of the transformation matrix :

$$\mathbf{A} = \begin{bmatrix} x_1 - x_0 + a_{31}x_1 & x_3 - x_0 + a_{32}x_3 & x_0 \\ y_1 - y_0 + a_{31}y_1 & y_3 - y_0 + a_{32}y_3 & y_0 \\ \frac{\Delta x_3 \Delta y_2 - \Delta y_3 \Delta x_2}{\Delta x_1 \Delta y_2 - \Delta y_1 \Delta x_2} & \frac{\Delta x_1 \Delta y_3 - \Delta y_1 \Delta x_3}{\Delta x_1 \Delta y_2 - \Delta y_1 \Delta x_2} & 1 \end{bmatrix} \tag{5.11}$$

To compute the reverse of this transform requires $\mathbf{A}^{-1}$ be found. This can be found by observing that $\mathbf{A}^{-1} = \frac{\operatorname{adj}\mathbf{A}}{\det\mathbf{A}}$. Now as $\det\mathbf{A}$ is a scalar quantity then $\operatorname{adj}\mathbf{A}$ can be used as an approximation of $\mathbf{A}^{-1}$ so long as suitable scaling is performed on the coordinates.

To employ this transformation on the image requires that the corner points of the quadrilateral be known. This could be computed via image processing and the methodology of edge detection. However, as they are fixed in all frames they were found by empirical examination. Once found, these points were used to generate a grid of points at which to sample the image. Neighbouring points yield a quadrilateral in the xy plane and once deskewed they form a rectangle in the uv plane. For each quadrilateral the intensity was computed using bi-linear interpolation (average of the intensities of the nearest pixels). So long as the density of points in the grid is sufficiently large then this method will yield an accurate image with the perspective warp removed.

5.3.3 Computing percentage coverage

After the perspective warping is removed in the composite image, it is very simple to compute the percentage coverage C. The covered area is estimated by counting pixels in the trail of the robot in the composite image. Since the covered area is calculated in pixels, the reachable surface area has to be in pixels as well.

$$\begin{aligned} C &= \frac{\text{Area of surface covered}}{\text{Total reachable surface area}} \\ &= \frac{\text{Number of covered pixels}}{\text{Total number of pixels in environment} - \text{Number of obstacle pixels}} \end{aligned} \tag{5.12}$$

For the experiments in this thesis, the tray shown in Figure 5.4 was used. A piece of software is written especially for calculating the number of pixels within the tray and the obstacles. These regions are marked manually inside this program, and the software will calculate the number of pixels in the two categories. The difference between the tray and the obstacles within gives the total area of reachable surface in number of pixels.

5.3.4 Calculating normalised path length

Unlike in simulation, efficiency cannot be estimated by the number of repeatedly covered pixels in the composite image. This is because a real robot does not move in a grid. It "re-visits" a significant number of pixels even when moving forward. Therefore, we measure efficiency as the distance the robot travels instead.

To obtain the path length metric in (5.3), estimations of the actual distance travelled by the robot and the minimal path to cover the environment are needed. The actual distance travelled can be obtained using the wheel encoders on robots. Wheel encoders are reasonably accurate in measuring distances travelled, with errors arising only when the robot slips, or if a wheel is not completely circular [39].

The minimal path is estimated using the assumption that the robot moves in a zigzag pattern. The area to be covered can then be divided as shown in Figure 5.7. The width of each division is the inter-strip distance Δx of the zigzag. The minimal path length $||P_m||$ to cover the entire region is the sum of the lengths of these divisions.

With an estimation of the lengths of the minimal path $||P_m||$, the actual path $||P_a||$ and the percentage coverage C (from the composite image), (5.3) can be used directly to calculate the path length metric L.

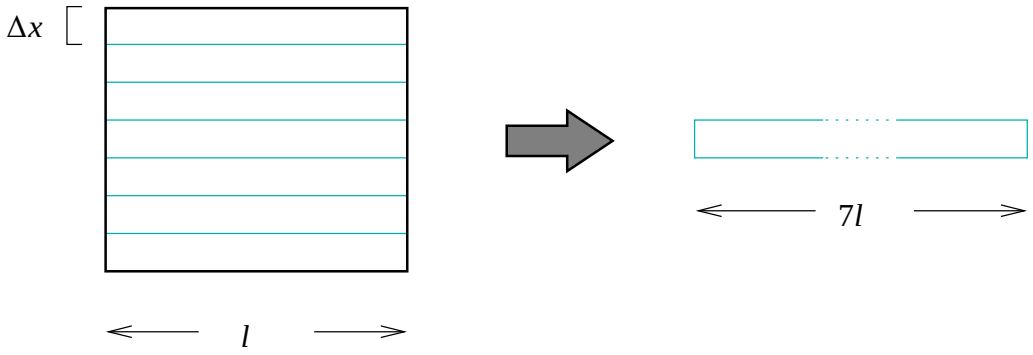


Figure 5.7: Estimating the minimal path length in real robot experiments. Δx is the step size of the sweep, which is also the distance between consecutive strips in the zigzag.

5.4 Summary

In simulation, the effectiveness metric C is the same as the one used by Gabriely and Rimon [45]. However, the path length L is an improvement over their use of repeatedly covered cells for measuring efficiency of coverage path. With L, the length of the actual path taken by the robot is compared with the minimal path of the environment. In other words, it shows how much longer the path generated by the algorithm is compared to the minimal path. Moreover, the metric also takes into account multiply covered cells and actual area covered.

The major contribution of these metrics however lies in the ability to evaluate them from real robot experiments. Previously, the only real performance metric for non-simulated robots was the coverage factor proposed by Butler [22]. The problem with the coverage factor is that it is a poor measure of both efficiency and effectiveness. This chapter outlines practical and simple solutions for calculating both metrics proposed from real robot experiments. A feature of the method used for extracting the parameters needed is that it is robot platform independent. The only on-board sensor required is wheel encoders for recording the actual distance travelled. Even this requirement can be relaxed since the path travelled by the robot is reconstructed when creating composite images using Hough transform (evidence gathering). As a result, the path length travelled can be calculated in terms of pixels from the captured movie of the experiment. Therefore, the only equipment necessary for evaluating the two proposed metrics is a camera to record the experiments.

The performance metrics will be employed in Chapter 7 to evaluate experimental results of the topological coverage algorithm.

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein

6 Implementation

The validity of the topological coverage algorithm was verified through testing in simulation and with the miniature robot Khepera [75]. This chapter describes the methods and tools used for developing, debugging and testing the topological coverage algorithm. The robot used and its command set is introduced first in Section 6.1. This is followed by a description of the simulated robot and environment, where most of the development took place (Section 6.2). The next two sections concentrate on the implementation of the coverage algorithm itself, with the topological map in section 6.3, and the the robot controller in Section 6.4.

6.1 Khepera robot

The robot used for testing the topological coverage algorithm is the miniature Khepera robot. A picture of it is shown in Figure 6.1. The robot is 53mm in diameter. The robot is equipped with eight infra-red proximity sensors for detecting obstacles. The sensors are placed around the robot in the layout shown in Figure 6.2. The infra-red sensors can detect objects up to 30mm to 40mm away. The sensors return an integer between 0 and 1023 depending on the distance between the sensor and the obstacle (with 1023 being the closest). The Khepera is also equipped with incremental optical wheel encoders for dead reckoning. The resolution of the

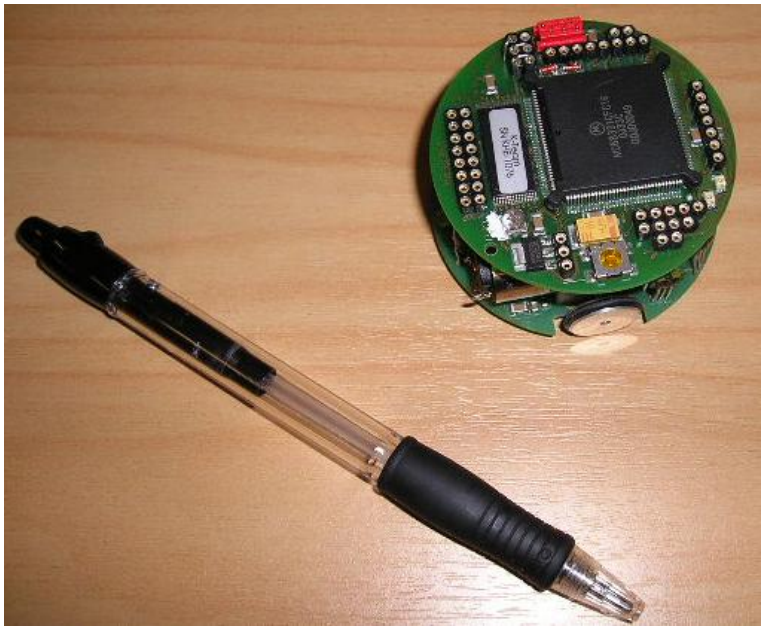


Figure 6.1: The Khepera miniature robot.

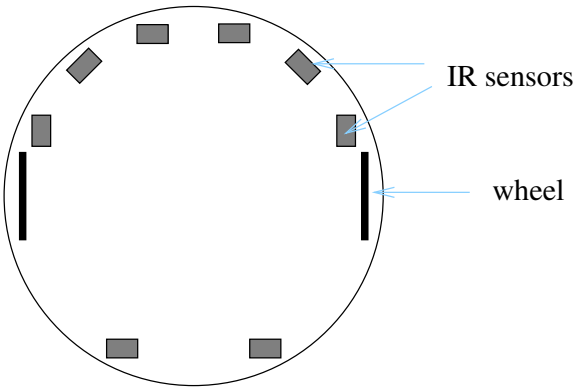


Figure 6.2: Layout of infra-red sensors on the Khepera.

wheel encoders is 12 pulses per mm of path of the robot.

The Khepera can be programmed and controlled using either the on-board Motorola 68HC08 microcontroller, or the serial communication protocol via an RS232 serial line. For the experiments in this thesis, control via the serial link is used. This is because implementing the algorithm on a PC allows for faster development, easier debugging and better code sharing between the Khepera and simulation experiments.

The serial communication protocol provides complete control of the functionalities of the Khepera. Table 6.1 shows a list of commands that are used in implementing the topological coverage algorithm. The protocol is in the form of commands and responses. Commands are sent from the host computer to the robot, and the responses are the answer the robot gives for the command sent.

Function	Command	Response
Set a position to be reached	C,left,right	c
Set speed	D,left,right	d
Read speed	E	e,left,right
Set position to the position counter	G,left,right	g
Read position	H	h,left,right
Read proximity sensors	N	n,sensor 0,$\cdots$,sensor 7

Table 6.1: Commands in the communication protocol of the Khepera. This list contains only commands used in the experiments.

The set position (C) command is used to instruct the robot to turn 90^o and 180^o. Normal navigation, such as move forward and wall following, is done using the set speed (D) command. The read speed (E) command sends the speed of the wheels back to the host computer, which can be used to check if the robot is moving. The set position counter (G) command is used to clear the counters of the wheel encoders; while the read position (H) command is used to read the values of these counters. The read proximity sensors (N) command is for obtaining range sensor data.

6.2 Simulation

The simulation environment is modelled as a 50×50 uniform grid. The robot is circular and has a diameter of one grid cell. It is programmed to keep a minimum distance of one grid cell from obstacles. It is equipped with 8 range sensors distributed uniformally around its circumference. The sensing range can be varied, but it is set to 10 grid cells by default. It is assumed that the sensors have no blind spots, and can detect everything around the robot up to the maximum sensing range. Also, the regions of detection for individual sensors do not overlap. The detection region therefore grows from the simulated robot in a pattern like the one shown in Figure 6.3. If an obstacle falls within the region of detection of one of the 8 sensors, the simulated robot will be notified of the index of the sensor that detected the obstacle, and the distance to the obstacle.

An environment editor was written to ease the creation of environments for testing. It imports and exports environment description in plain text files. Figure 6.4 shows a screenshot of the editor. Obstacles can only be drawn as straight lines and circles. By using a combination of these shapes, more complicated obstacles can be created.

Figure 6.5 shows the main simulator. The environment loaded is displayed twice on the screen. The left half shows the world view, where the robot moves; the right half shows the map view, which displays the topological map generated. Figure 6.6 shows another screenshot of the

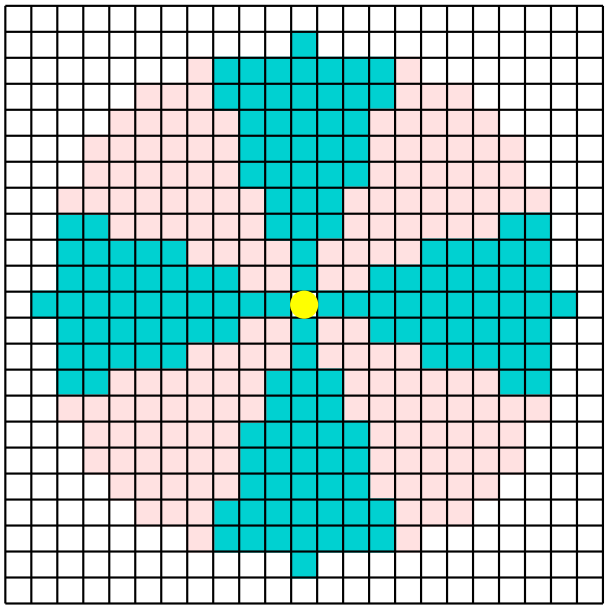


Figure 6.3: Region of detection for a simulated robot with a sensing range of 10.

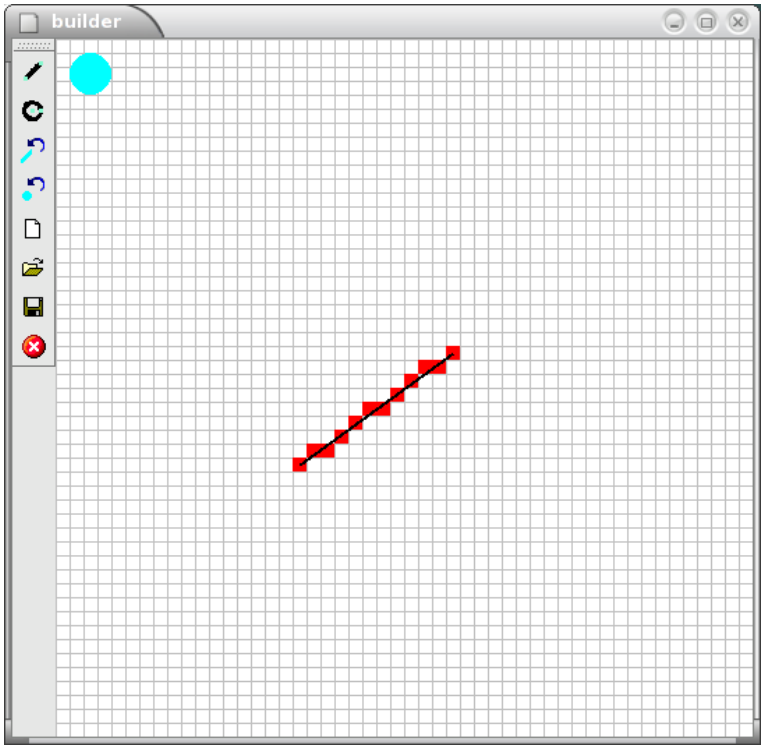


Figure 6.4: Screen shot of the environment editor. The environment in this figure contains only a single diagonal line.

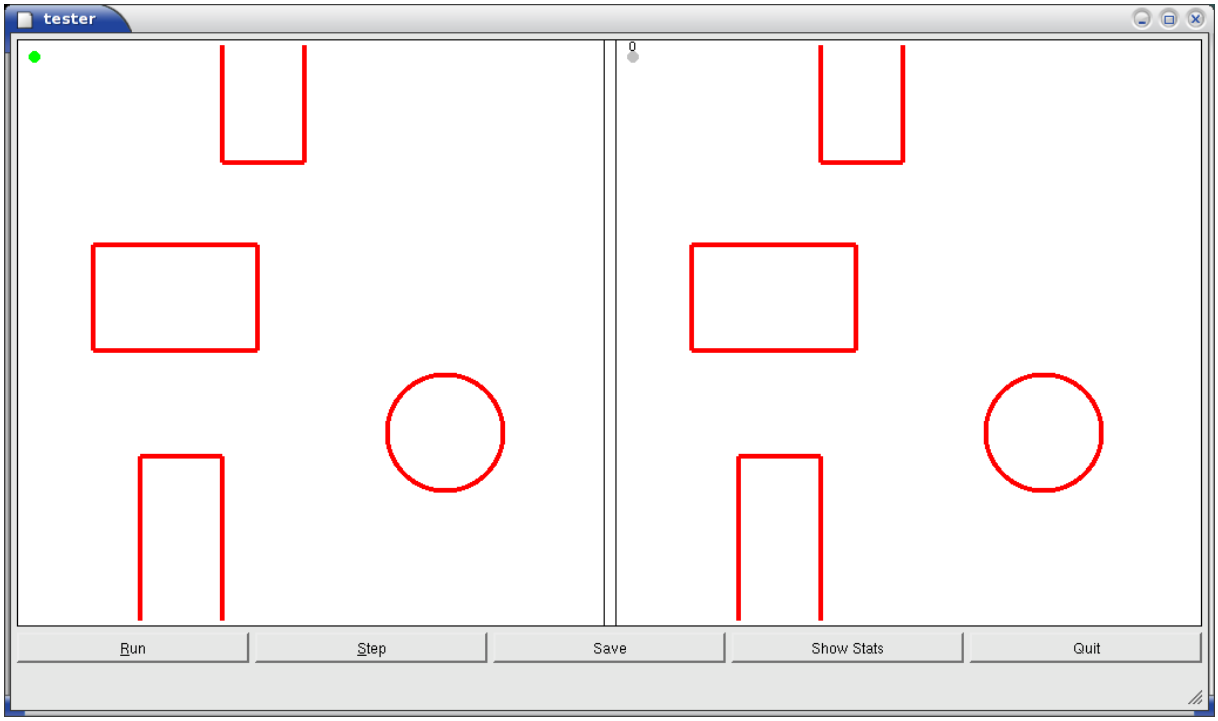


Figure 6.5: The simulator screen is divided into two halves. The world view on the left side displays the environment the robot moves in. The map view on the right shows the topological map created.

simulator, taken after the simulation has run for a little while. The world view now displays surfaces that the robot has covered, marked with a different colour. Grid cells that have been visited more are shaded with a darker colour.

The map view in Figure 6.6 shows the topological map created so far. The nodes are drawn in the grid cells where they are first detected, with the exception of uncovered nodes. These are initially displayed with their adjacent nodes and will be moved once they are converted to free space, obstacle or joint nodes. Also, all nodes are drawn as identical circles irrespective of their types. The numbers above the nodes are their internal tags within the topological map and are displayed for debugging purposes. If multiple nodes exist in the same grid cell, only the tag with the lowest numerical value is displayed. The edges are drawn in different colours depending on their types to aid debugging. Clicking anywhere on the map view of the simulator will bring up an information dialogue. This dialogue shows the nodes present at that position, and edges that are incident to them. An example of this is shown in Figure 6.7. Here, the grid cell has an obstacle node (13) and an uncovered node (14). Node 13 is connected to three different nodes, one of which is node 14 in the downward direction. There is a corresponding entry for node 14 that shows it is connected to node 13 in the upward direction. In the case where there are no nodes at the location selected, an empty dialogue box is shown.

Statistics for the performance metrics of simulated experiments can be obtained via the “Show Stats” button in the main window. Figure 6.8 shows the statistics for the simulation in Fig-

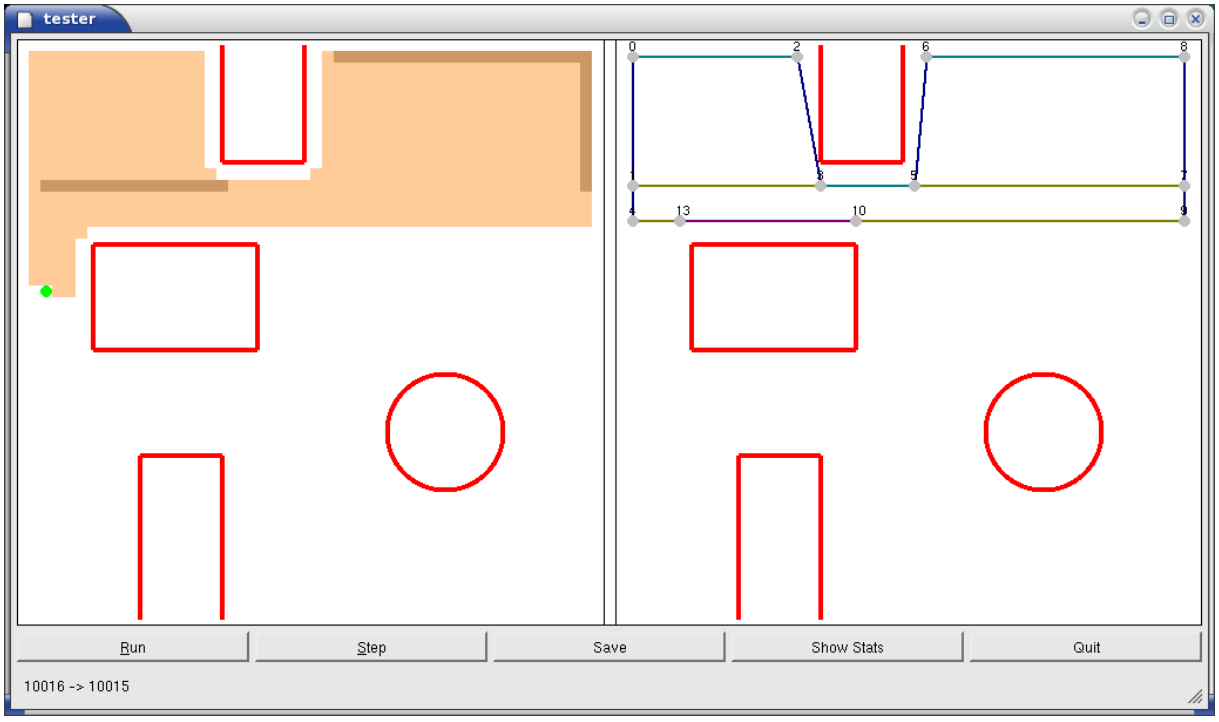


Figure 6.6: Covered area is shaded with multiply covered surfaces in darker colours.

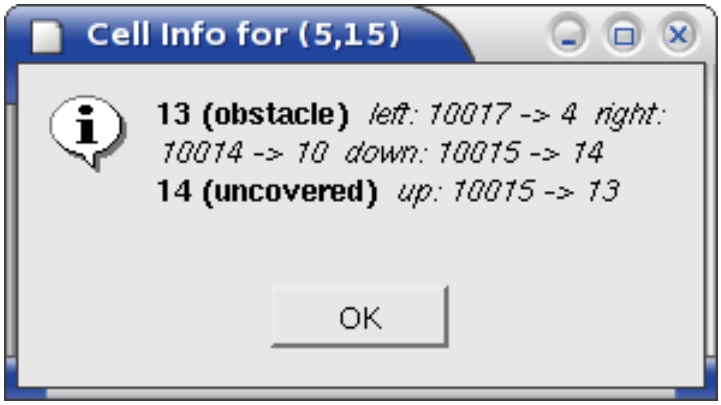


Figure 6.7: Information on nodes present at any location can be obtained by clicking on the map view of the simulator.

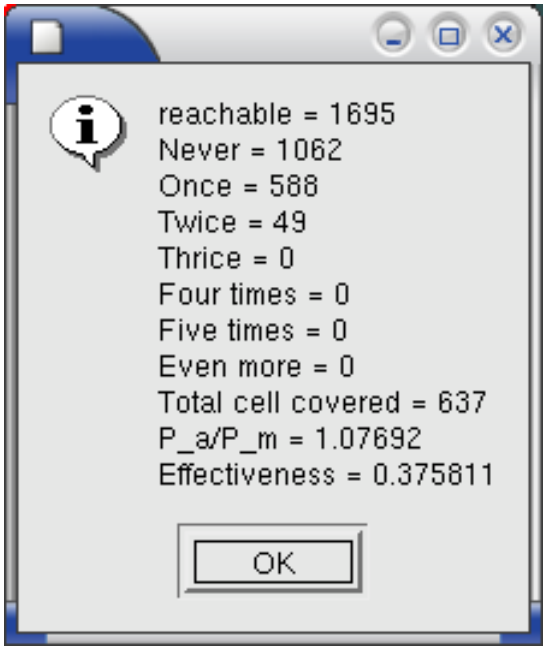


Figure 6.8: The statistics dialogue shows various statistics for the simulation.

ure 6.6. Since only a small portion of the environment is covered, a large number of cells are in the *never* category (1062 to be exact). The last line in the dialogue box shows the percentage coverage C, and the second to last shows the normalised path length without scaling L' (Section 5.1).

6.3 Topological map

The topological map is implemented using the Graph class in LEDA[1] [73]. However, the implementation is not LEDA dependent, and other graph libraries can be used instead. A suitable alternative is the open-sourced Boost graph library [4, 90]. Both LEDA and Boost libraries are written in C++. Of course, there is always the option of writing your own graph library.

The graph library should at least provide the functions listed below. The list is included here to illustrate what is required of the library, and to aid in selection of a suitable library, especially if the topological coverage algorithm is to be implemented in a language different from C++.

1. Retrieve adjacent and incident nodes or edges.
2. Obtain properties such as degree of a node[2], number of edges and nodes in graph, and if the graph is empty.

[1] Version 3.7.1 is used for this thesis. A few years ago, the LEDA library had a free license for research use in academic institutes. The newer versions are non-free only.

[2] The degree of a node is the number of edges incident with it.

3. Alter existing graph by addition and deletion of nodes and edges.

4. Store extra data with nodes and edges.

The usefulness of the last item on the list, the ability to store extra data with nodes and edges, may not be self evident. In robotics, the topological maps usually have information associated with their nodes and edges. For example, the nodes might have information about the landmarks they represent, such as sonar signatures [65] or panoramic images [106]; the edges might store appropriate behaviours or control strategies for traversal [64]. Therefore, it is important for the library to provide a mechanism for associating extra data of arbitrary types with graphs.

6.4 Robot controller

Based on current sensor inputs, and the internal representation (topological map), the robot controller issues appropriate commands to the actuators. The purpose of the controller is to guide the robot to carry out the topological coverage algorithm. The mechanism used to communicate with the robot differs between simulation and the Khepera. With the simulated robot, communications are achieved through function calls. For the Khepera, the controller communicates with the robot via a serial link.

The controller employs a hybrid deliberative/reactive architecture. This hybrid architecture incorporates the deliberative reasoning and planning in symbolic AI with the responsiveness of behaviour-based execution [24, 68]. The topological coverage algorithm is organised as a finite state machine with three states (normal, boundary, travel). For each state, a different set of behaviours is active. The behavioural control for each state is also implemented as a finite state machine, which is a common way of implementing behaviour-based architectures in sequential machines [61].

In the normal state, the robot moves in a zigzag path to cover the current cell. The state contains two behaviours, *forward* and *next strip*. Their interaction is shown in Figure 6.9. *Forward* moves the robot along the strip of a zigzag. When the robot encounters an obstacle in the front, it has arrived at the end of a strip. *Next strip* guides the robot to move into the next strip of the zigzag. It is composed of three sequential movements – turn 90^o, move forward for a fixed amount of time, and turn 90^o again. This series of actions brings the robot into the beginning of the next strip and faces the correct direction. If a landmark is encountered at any time during the execution of the normal state, the controller moves onto the boundary state (see Section 4.1.1).

The boundary state guides the robot along a cell boundary. Its operation is carried out by three behaviours, as shown in Figure 6.10. The *forward* and *wall follow* behaviours move the robot along the cell boundary for exploration. Every time the robot visits a landmark, it alternates

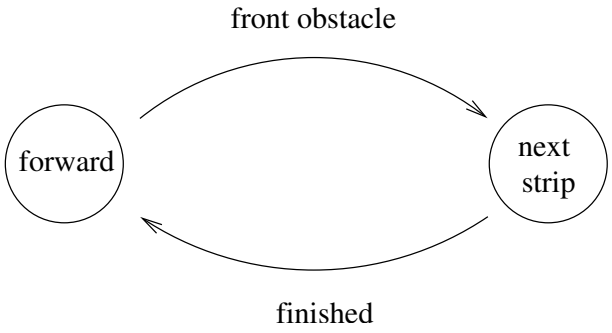


Figure 6.9: Following a zigzag path in the normal state.

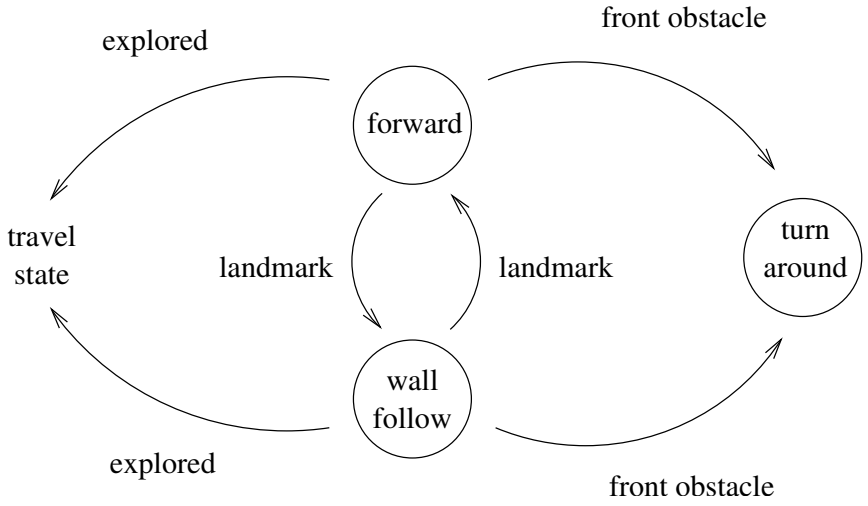


Figure 6.10: Behaviours for cell boundary exploration.

between the two states. This is because landmarks appearing in the middle of the strip arise from obstacle segment topology changes. When the robot reaches the end of the strip, it checks whether the boundary has been fully *explored*. If it has, the controller leaves the boundary state; otherwise the robot *turns around* and explores the cell boundary in the other direction.

In the travel state, the robot follows a path that leads it from the current location to a chosen uncovered cell. This path is generated from a search on the topological map. The behaviours that guide the robot along this path are shown in Figure 6.11. The edges in the path are followed using either *wall follow* or *forward*, depending on the edge type. When the robot arrives at the next node in the path, it turns to face the appropriate direction (*face direction*). Then it embarks on the edge in the path by choosing the appropriate control strategy. This continues until the robot arrives at the destination node, when the controller returns to the normal state.

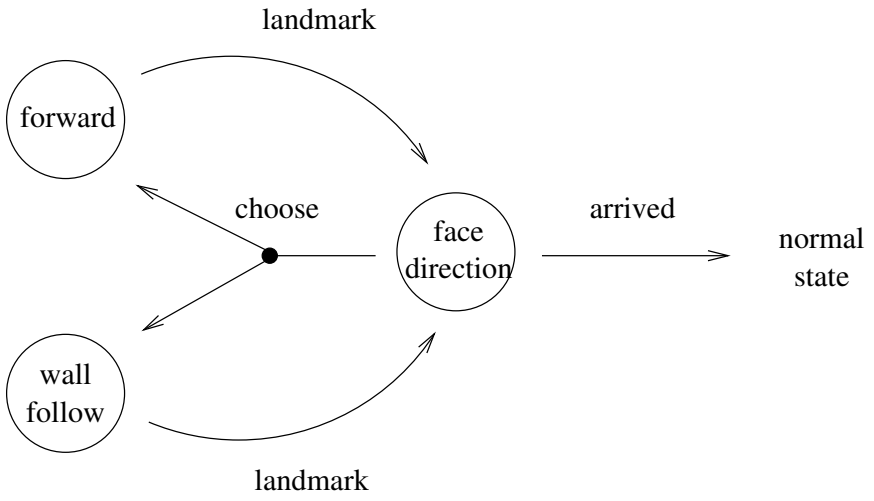


Figure 6.11: Behaviours guiding the robot to move along a path of nodes and edges in the travel state.

6.5 Summary

The development tools and implementation details described in this chapter serve as a link between the theory presented in earlier chapters and the experimental results in the next chapter. It explains how the ideas developed in this thesis are transformed into actual prototypes (on simulated and physical robots).

The topological coverage algorithm is the result of an iterative development process. Other than verifying the correctness of the algorithm, the experiments are also an integral part of the development process of the algorithm itself. This iterative process is the reason behind the large amount debugging help present in the simulation tools.

The information and details included in this chapter will also be of use to anyone wishing to implement the topological coverage algorithm.

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke

7

Results and discussion

This chapter presents results from experiments designed to test the proposed topological coverage algorithm. It begins with a description of the sensor detection tests in Section 7.1. These tests assess the ability of common range sensors in detecting the topology changes of events in the topological coverage algorithm. This is then followed by qualitative analyses of results from both simulation and real robot coverage experiments in Section 7.2. Section 7.3 discusses the use of zigzag as the coverage pattern in the topological coverage algorithm. In Section 7.4, the two methods for creating composite images are compared. The evaluation is done because the percentage coverage C is an important parameter for the two performance metrics. The metrics are then used in Section 7.5 to analyse quantitatively the experimental results first presented in Section 7.2. Section 7.6 explains how composite images can be created for robots that are not circular. Lastly, Section 7.7 tests the relationship between complexity of environment and path length empirically.

7.1 Landmark Detection

Three types of range sensors commonly used in mobile robots were tested for their abilities to detect landmarks in the topological coverage algorithm. They are laser scanner, ultrasonic transducers and infra-red proximity sensors. The tests on laser and sonar sensors were conducted

Figure 7.1: B21r robot.

with the b21r robot in Figure 7.1. The laser scanner is a SICK model LMS-200 and is mounted on the front of the b21r. For the experiments in this chapter, the resolution was set to 0.5°. Each scan returns 361 readings, thus covering a semi-circle in front of the robot. The sonar sensors are SensComp 6500 modules. They are evenly distributed around the b21r and there are 24 of them in total. The infra-red sensor tests were done with the Khepera robot. The infra-red sensors are Siemens SFH900s, and their positions on the Khepera are shown in Figure 6.2 on Page 114. They return an integer between 0 and 1023 depending on the distance to the closest obstacle (1023 being nearest).

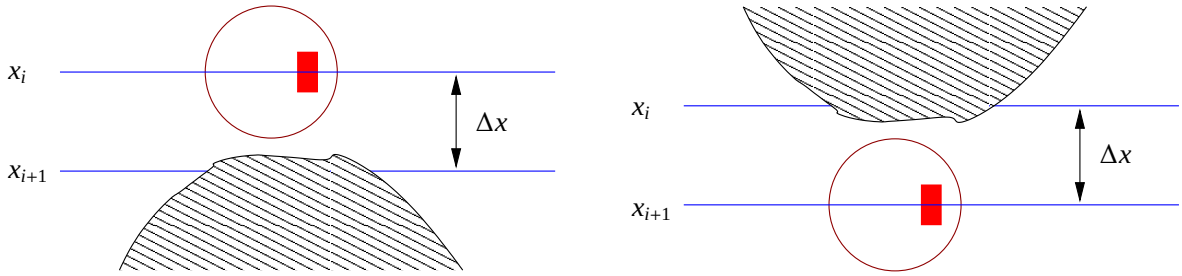
Two sets of tests were carried out. The first set tests the ability to discover discontinuities on the side of robots. This is used in the detection of split, end and shorten events at the sweep position x_i. It is also used to detect merge and shorten events at sweep position x_{i+1}. The second set of tests examines the ability of the robot to detect a gap or opening in the front or back direction. This is used to detect merge and lengthen events at sweep position x_i. It is also the only method to detect combined split and merge events. This is because combined split and merge events cannot be detected using odometry by comparing lengths of consecutive strips. In comparison, all the other events can fall back to detection by odometry in case range sensing fails.

7.1.1 Discontinuity on side of robot

Figure 7.2(a) shows the experimental setup used to test the sonar and laser sensors on the b21r. Two sets of bookcases were used to set up a series of obstacles for this experiment. The gap between the two set of bookcases was approximately 80cm. The b21r was driven parallel to the obstacles, in the direction shown in Figure 7.2(b). Laser and sonar sensor readings were

(a) (b)

Figure 7.2: Experimental setup for testing topology changes on the side with the b21r.



(a) (b)

Figure 7.3: (a) At the last strip before reaching the emerging obstacle. (b) At the first strip passing below the disappearing obstacle.

recorded. The experiments were repeated 12 times, with the b21r at varying distances from the obstacles.

In the first 4 experiments, the b21r was placed fairly closed to obstacles. The left edge of the robot was about 20 to 30cm away from the obstacles. (The b21r has a diameter of 55cm). This corresponds to strip position x_i in split, shorten and end events, where the robot is on the last strip before the emerging obstacle blocks its path. This is illustrated in Figure 7.3(a). In merge and lengthen events, this corresponds to strip position x_{i+1}, where the robot is on the first strip to pass underneath the disappearing obstacle. This is shown in Figure 7.3(b).

Figure 7.4 plots the distances measured by the laser scanner from two of the experiments. The measurements were recorded while the robot was moving along the path shown in Figure 7.2(b). The plots superimpose readings from the 11 leftmost beam positions (5°). It can be seen that the laser scanner can detect the gap between the two bookcases for 7 consecutive time steps. This

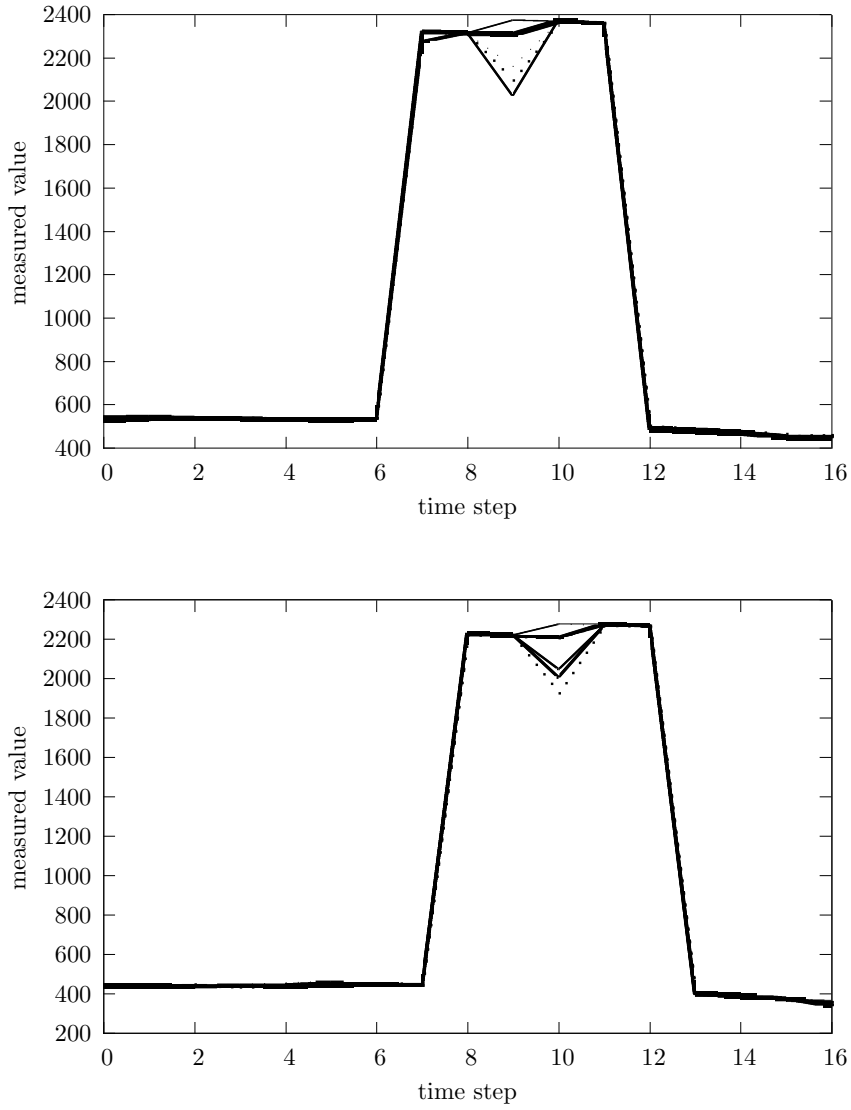
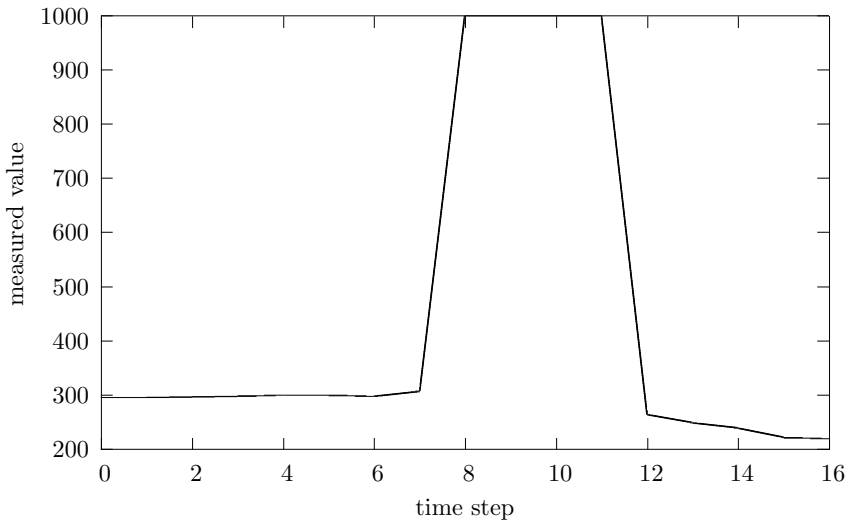


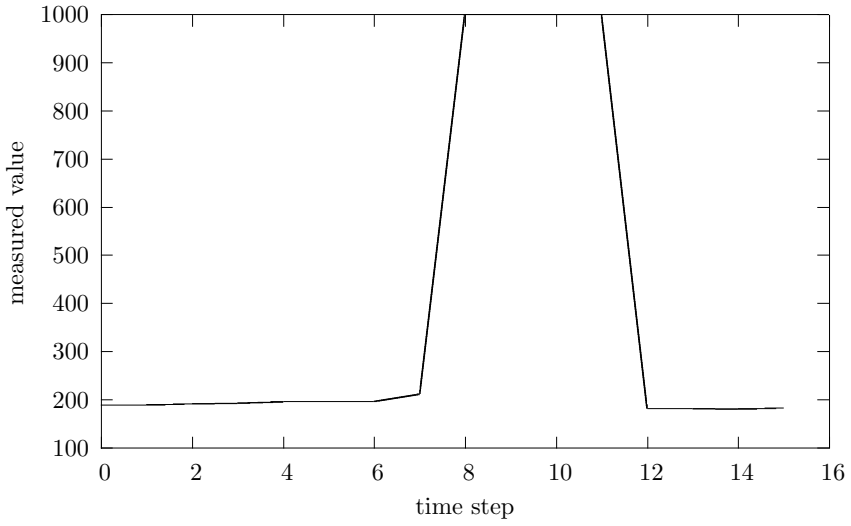
Figure 7.4: Distance to obstacles measured by the laser scanner on the b21r. The plots show measurements from the 11 leftmost beam positions of the sensor. The robot was fairly close to the obstacles. Distances were measured in millimetres.

confirms that changes in topology are large features and do not rely on detection from a single specific location. The slight drop in measured distance within the gap was due to the edge of the bookcase in the background behind the obstacles used for the experiments. Figure 7.5 shows distances measured by sonar sensors from the same two experiments as before. The plots show readings from the sonar sensor on the left of the b21r. It can be seen that sonar sensors had no problem detecting the gap either.

The experiments were then repeated 4 times with the b21r placed slightly further away (about 50cm away). However, the distance between the robot and the obstacles remains smaller than the diameter of the robot. Therefore, the robot is still on the last strip before the emerging obstacle (Figure 7.3(a)), or the first strip after a disappearing one (Figure 7.3(b)).



(a)



(b)

Figure 7.5: Distance to obstacles measured by the ultrasonic sensor mounted on the left hand side of the b21r. The robot was fairly close to the obstacles. Distances were measured in millimetres.

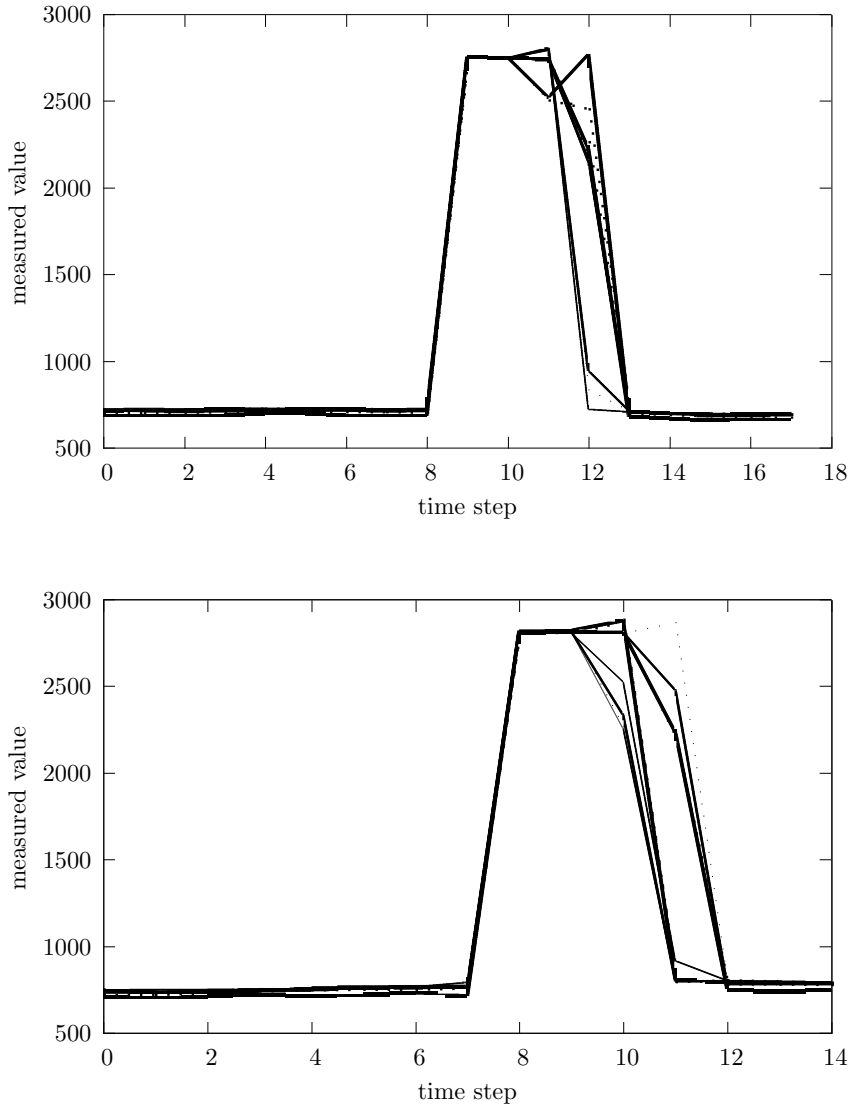


Figure 7.6: Distance to obstacles measured at about 50cm away with the laser scanner. The plots show measurements from the 11 leftmost beam positions of the sensor. Measurements are in millimetres.

Figure 7.6 plots the distances measured by the laser scanner from two of the experiments. The plots superimpose readings from the 11 leftmost beam positions (5°). The sonar sensor measurements from the same experiments are shown in Figure 7.5. It can be seen that both the laser and sonar range finders can detect the gap easily.

In the last 4 experiments, the b21r was placed even further away, at approximately 80cm from the bookcases. As a result, the robot is now situated at one strip away from the cell boundary. This is illustrated in Figure 7.8.

Figure 7.9 shows readings from the laser scanner from two of the experiments. Figure 7.10 shows readings from the sonar sensor from the same two experiments. Both sensors can still

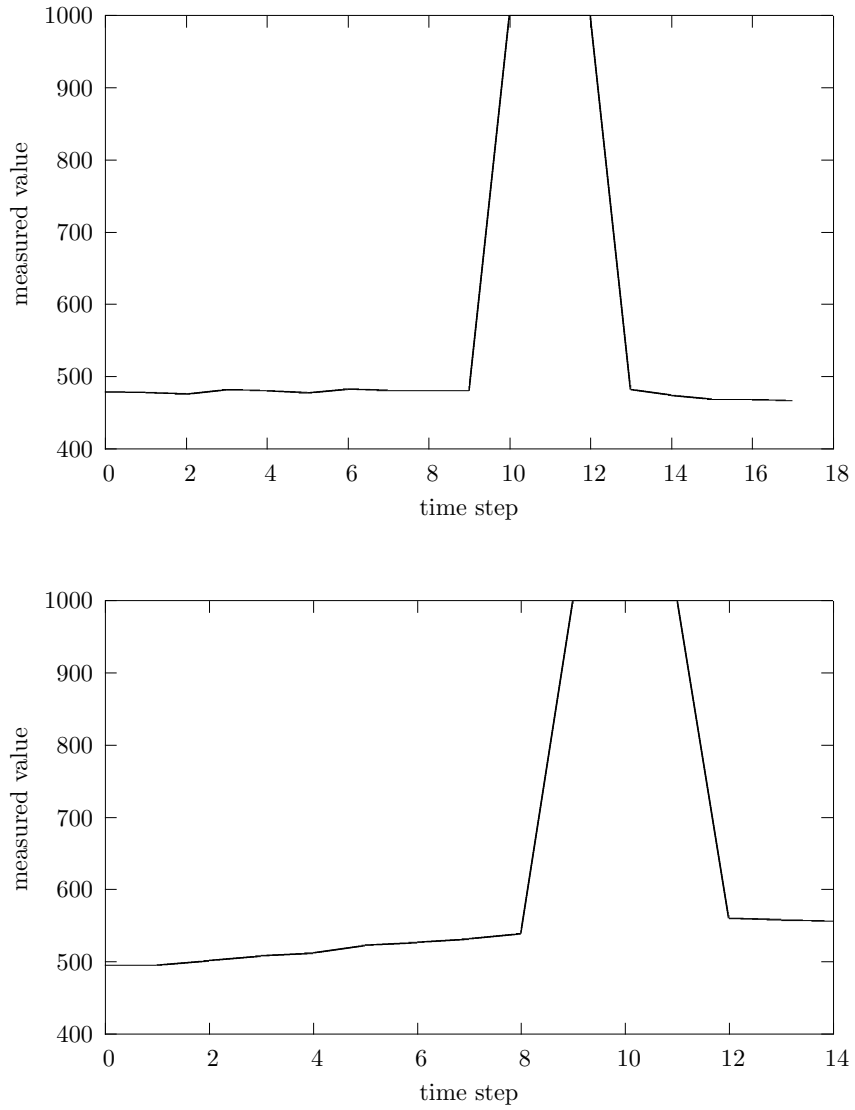


Figure 7.7: Distance to obstacles measured at about 50cm away with the sonar sensor mounted on the left of the b21r. Measurements are in millimetres.

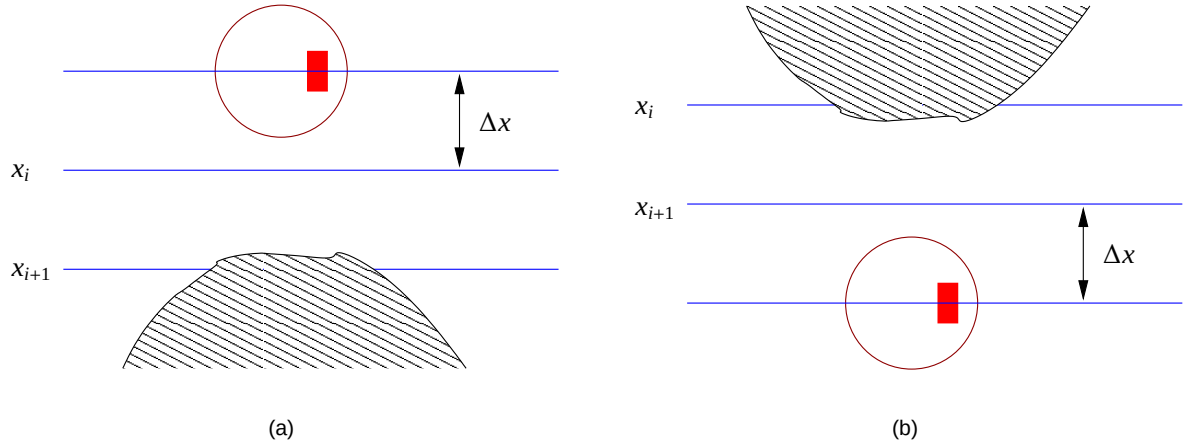


Figure 7.8: (a) At the second last strip before reaching the emerging obstacle. (b) At the second strip passing below the disappearing obstacle.

detect the gap easily at this distance.

The experiments were repeated with the Khepera robot to test the infra-red proximity sensors. Figure 7.11(a) shows the experimental setup used. A round plastic container and two square blocks are used to set up two sets of obstacles. The Khepera was driven parallel to the obstacles, in the direction shown in Figure 7.11(b). Two sets of experiments were carried out, at different distances from the obstacles. Each set of experiments was repeated 4 times.

In the first four experiments, the Khepera was placed fairly close to the obstacles, at about 20mm away. (The diameter of the Khepera is 53mm). This means the robot is on the strip nearest to the cell boundary, as in Figure 7.3. Figure 7.12 plots the readings returned by the infra-red sensor mounted on the left from two of these experiments. The first peak corresponds to the circular plastic container, and the second one corresponds to the square blocks. The infra-red sensor successfully detected the topology changes in all four experiments.

In the next four experiments, the Khepera was placed slightly further away at about 35mm from the obstacles. The distance is still smaller than the diameter of the robot. Figure 7.13 shows the infra-red sensor readings from two of the experiments. They show that the sensor successfully detected the topology changes.

No experiments were done at distances two strips away from the cell boundaries. This is because the diameter of the Khepera is larger than the reliable detection distance of the infra-red sensors (40mm). This means that if x_i in Figure 7.3(a) (or x_{i+1} in Figure 7.3(b)) is between 40mm and 53 mm away from the obstacle, the infra-red sensors will not be able to detect the emerging (or disappearing) obstacle. However, this does not imply that the discontinuity cannot be detected. All the events that lead to discontinuities on the side can also be detected with strip length comparisons with data collected from wheel encoders.

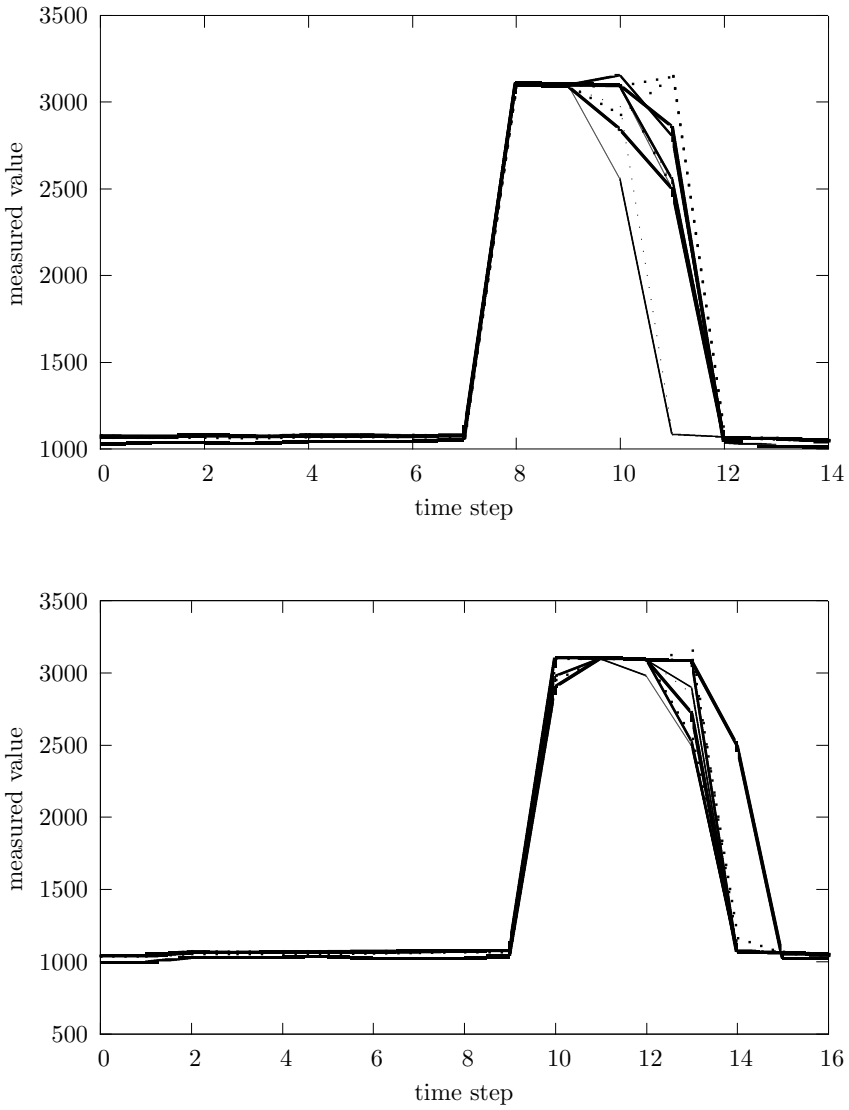


Figure 7.9: Distance to obstacles measured at about 80cm away with the laser scanner. The plots show measurements from the 11 leftmost beam positions of the sensor. Measurements are in millimetres.

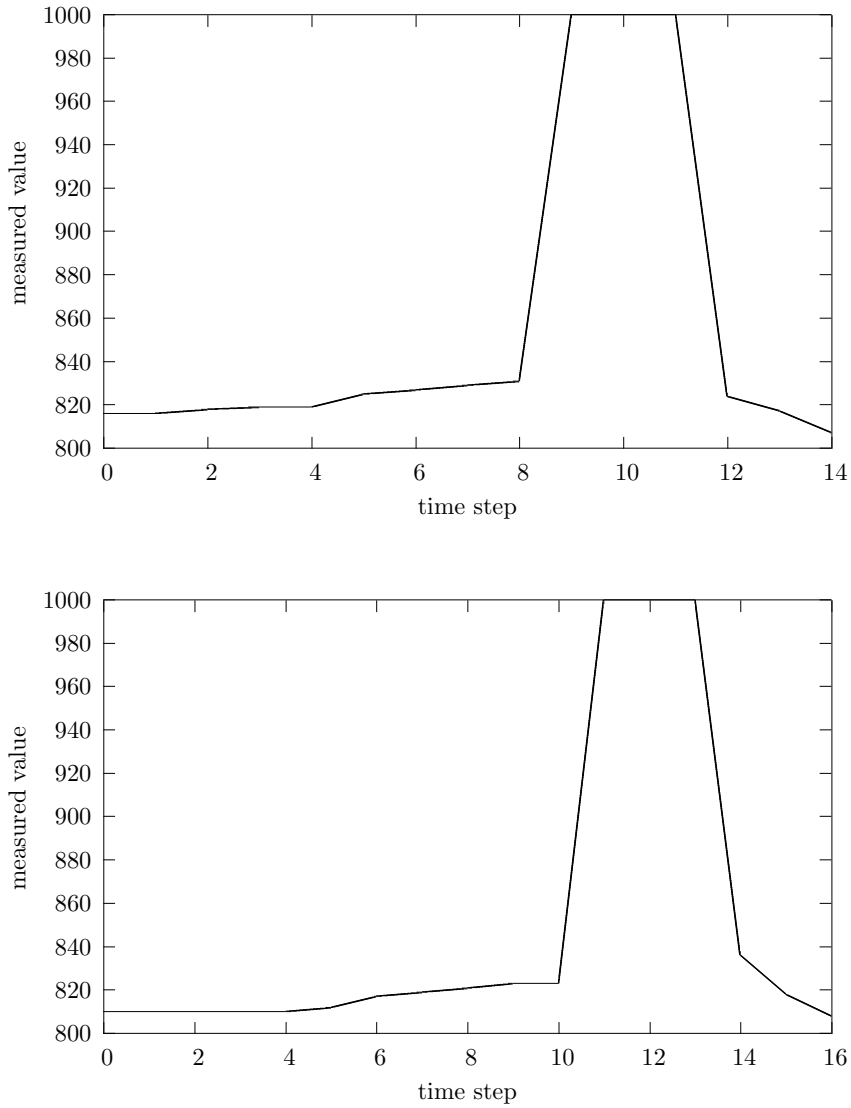
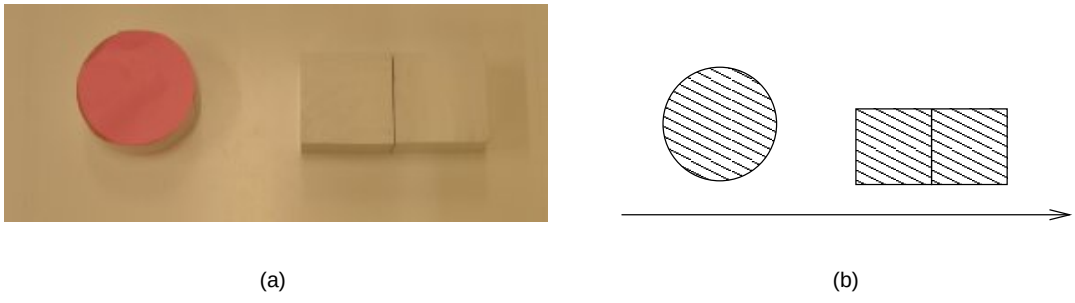


Figure 7.10: Distance to obstacles measured at about 80cm away with the sonar sensor mounted on the left of the b21r. Measurements are in millimetres.



(a) (b)

Figure 7.11: Experimental setup for testing topology changes on the side with the Khepera.

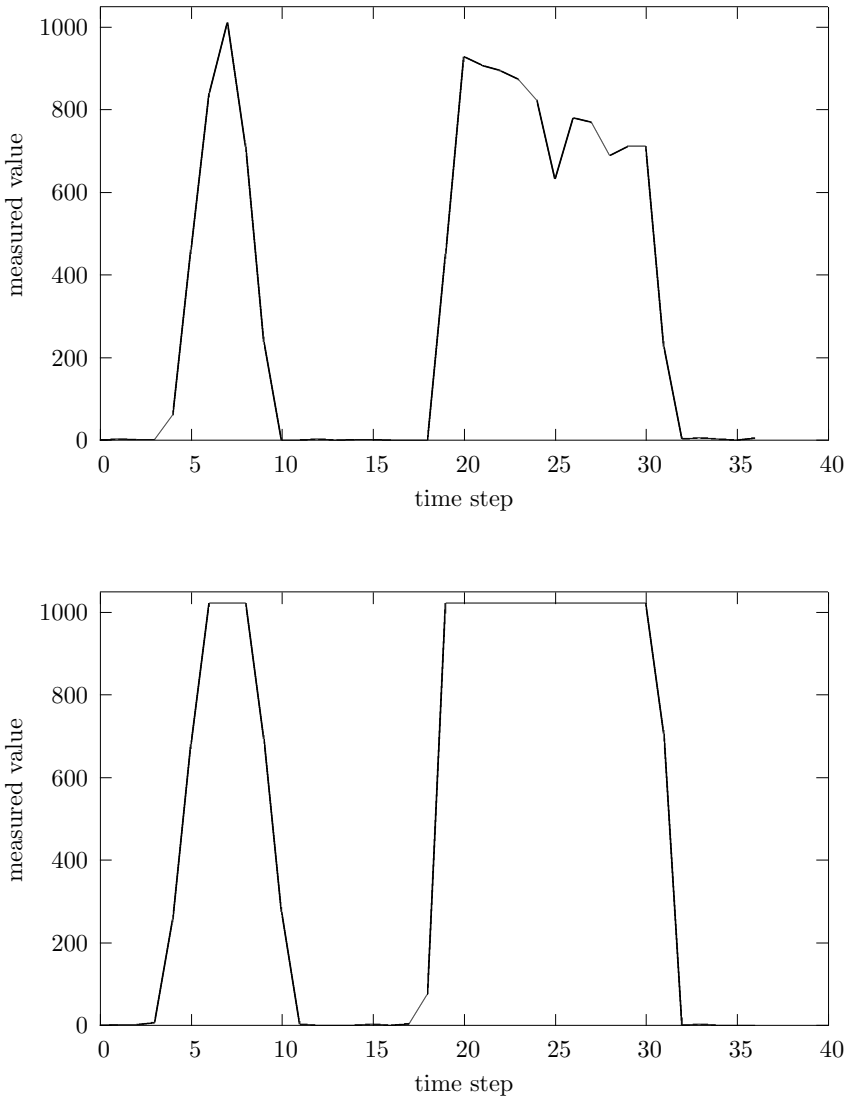


Figure 7.12: Distance to obstacles measured by the infra-red proximity sensor mounted on the left hand side of the Khepera. The robot was about 20mm away from the obstacles.

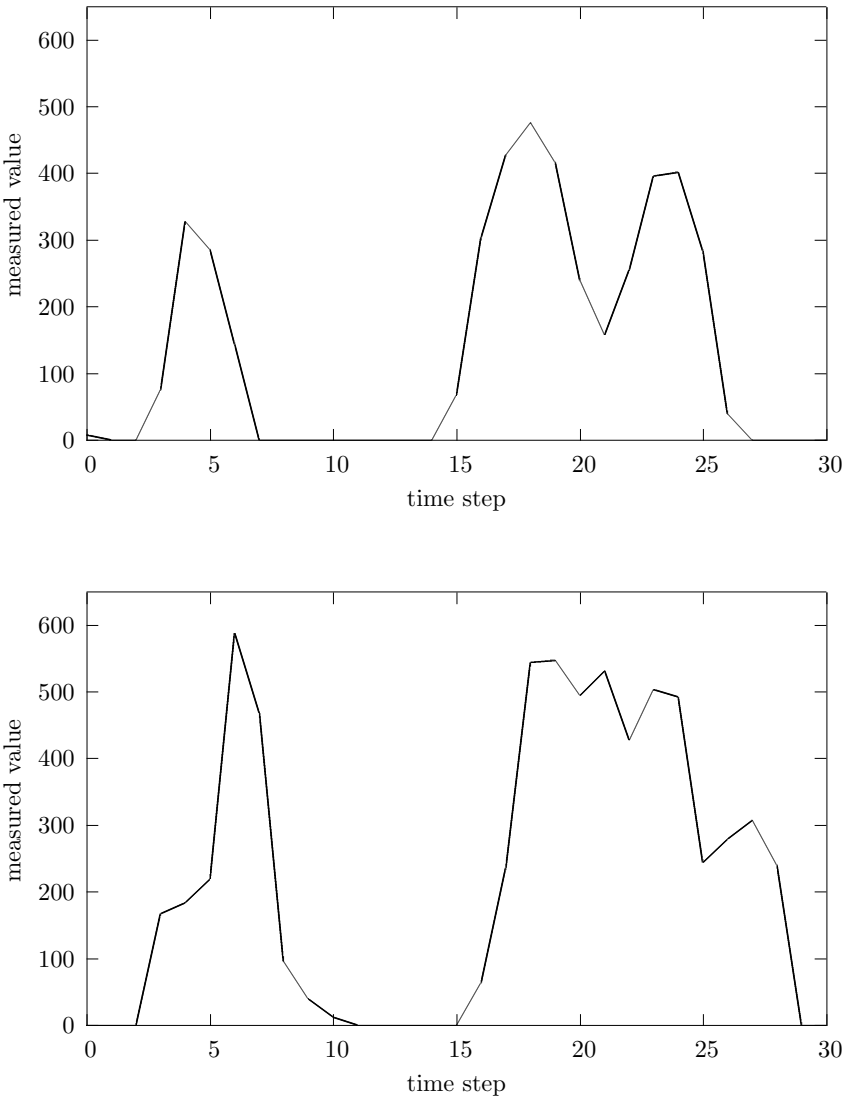


Figure 7.13: Distance to obstacles measured by the infra-red proximity sensor mounted on the left hand side of the Khepera. The robot was about 35mm from the obstacles.

(a) (b)

Figure 7.14: Experimental setup for testing topology changes in the front with the b21r.

7.1.2 Topology changes in front of robot

Figure 7.14(a) shows the experimental setup used with the b21r. Two partitions were used to create an opening of about 80cm wide. In the experiments, the b21r was driven towards the opening, as in Figure 7.2(b). While the robot was moving towards the partitions, laser and sonar sensor readings were recorded. Four of these experiments were carried out.

The experimental setup corresponds to strip position x_i in combined merge/split events. This is illustrated in Figure 7.15(a). In merge and lengthen events, the situation is slightly different from the experimental setup. This is because only one of the two obstacles are present, as in Figure 7.15(b). However, the sensor tests are still relevant as detection of merge and lengthen events at x_i relies on partial presence of obstacles in front of the robot. The emergence of another obstacles in proximity merely reduces the width of the opening.

When the robot is facing such an opening, some of the front range sensors will detect a much longer distance to obstacle than others. This is illustrated in Figure 7.16. Sensors pointing towards the opening will return longer distance readings than those facing the obstacles next to the opening. Figure 7.17 shows 3 sets of readings from the laser scanner as the b21r approached the partitions in Figure 7.14. The opening was in front of the robot, slightly towards the right hand side. As a result, there was a large difference in measured distances between the left and right fronts. This difference remained as the robot moved closer to the partitions.

Figure 7.18 shows distances measured by the laser scanner in two different experiments with the b21r. The values were recorded while the robot followed the path shown in Figure 7.14(b). For each graph, five sensor readings from beam positions 10° on the right of the centre front are plotted, together with five from the left. It can be seen that the distances measured on the right

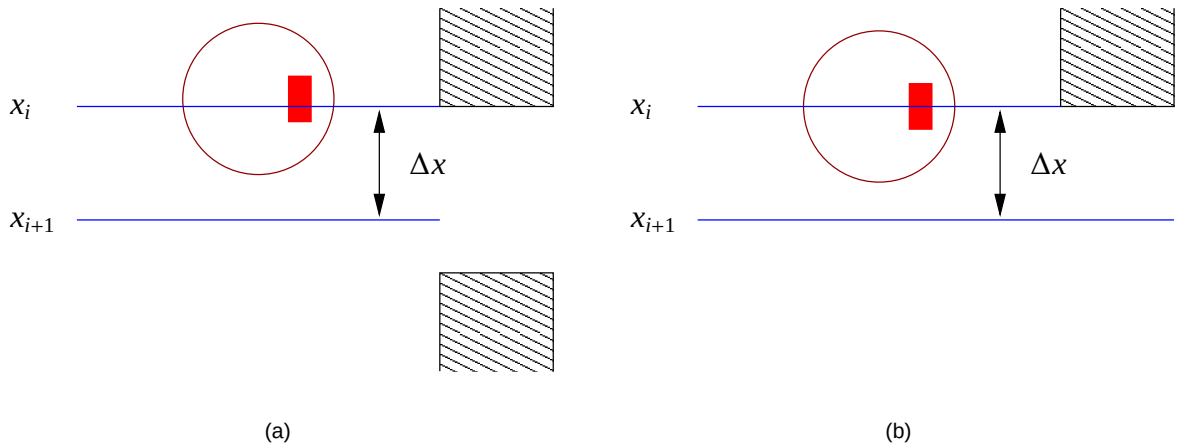


Figure 7.15: Moving towards an opening on the side boundary in (a) combined merge/split event, (b) merge and lengthen events.

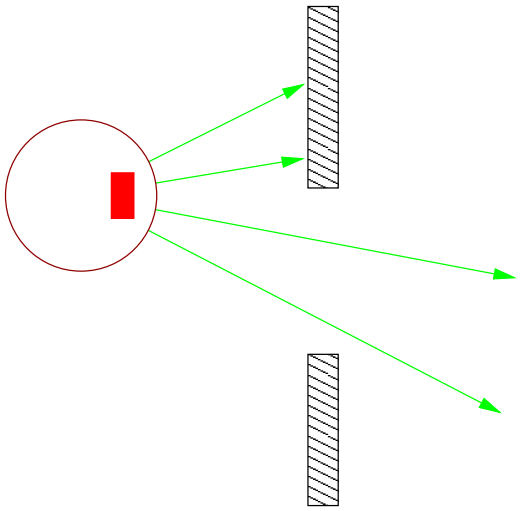


Figure 7.16: An opening in front of the robot is indicated by a large difference in measured distances.

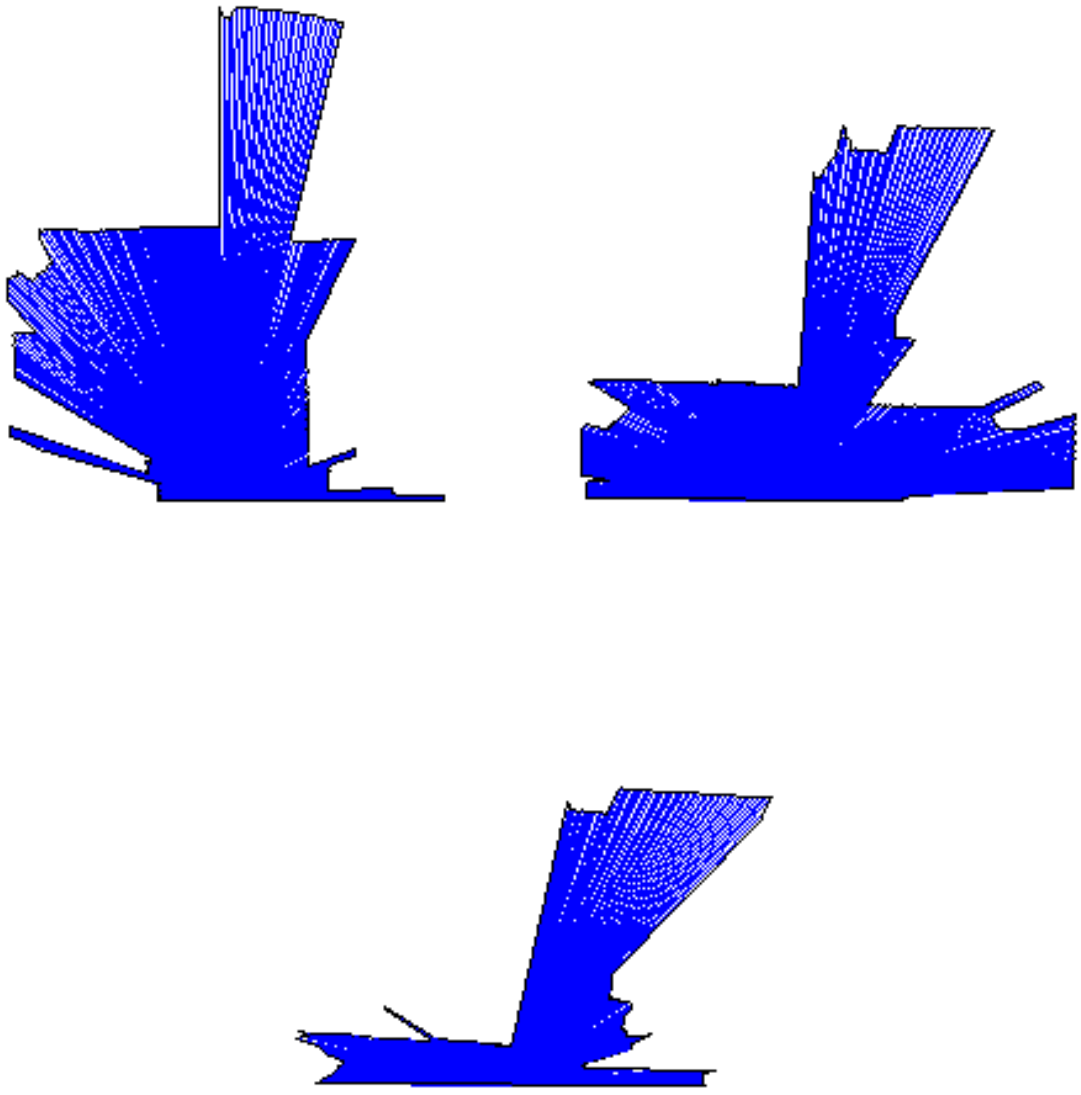


Figure 7.17: Laser sensor readings as the b21r moved towards and opening.

front of the robot are much larger than those from the left front. Therefore, the opening was successfully detected.

Figure 7.19 shows distances measured by sonar sensors as the robot moved towards the obstacles. For each graph, measurements from the first sensors to the left and right of the centre front are plotted. The distance measured by the left front sensor stayed at the threshold (1000mm). In comparison, the distance measured by the right front sensor dropped as the robot moved closer to the partitions. In other words, the sensor on the right front detected the obstacle, while the one on the left front did not. Therefore, the opening was successfully detected by the sonar sensors.

The experiment was repeated with the Khepera to test the infra-red proximity sensors. The experimental setup was the same as the one shown in Figure 7.14(b). Two square blocks were used to create the opening required. Four experiments were carried out.

Figure 7.20 shows the results from two of the experiments with the Khepera. For each graph, measurements from the sensors on the immediate left and right of the centre front are plotted. As the robot moved closer to the obstacles, the values returned by the right front sensor increased, indicating the appearance of an obstacle. On the other hand, the left front sensors never detected any obstacles throughout the experiments. In other words, the infra-red sensors successfully detected the opening between the obstacles.

7.2 Coverage Experiments

7.2.1 Simulation

To test the correctness of the proposed topological coverage algorithm, 11 environments populated with various standalone obstacles were created. The obstacles can be located either in the middle of the region or put against the boundary. The complexity of these environments ranges from 6 to 14 free space cells. Environments with standalone obstacles are the norm in testing coverage algorithms [8, 23, 30].

Figure 7.21 shows three screenshots from simulation with one of these normal environments. This figure shows how the topological map is incrementally constructed as the environment is being covered. Notice that the horizontal edges of the topological map correspond to the cell boundaries of the decomposition. Also, some of the vertical edges cross over the obstacles. This is because they are simply drawn as straight lines linking their incident nodes. The vertical edges merely represent the side boundaries of cells in the decomposition, and there are no restrictions on possible shapes of these boundaries.

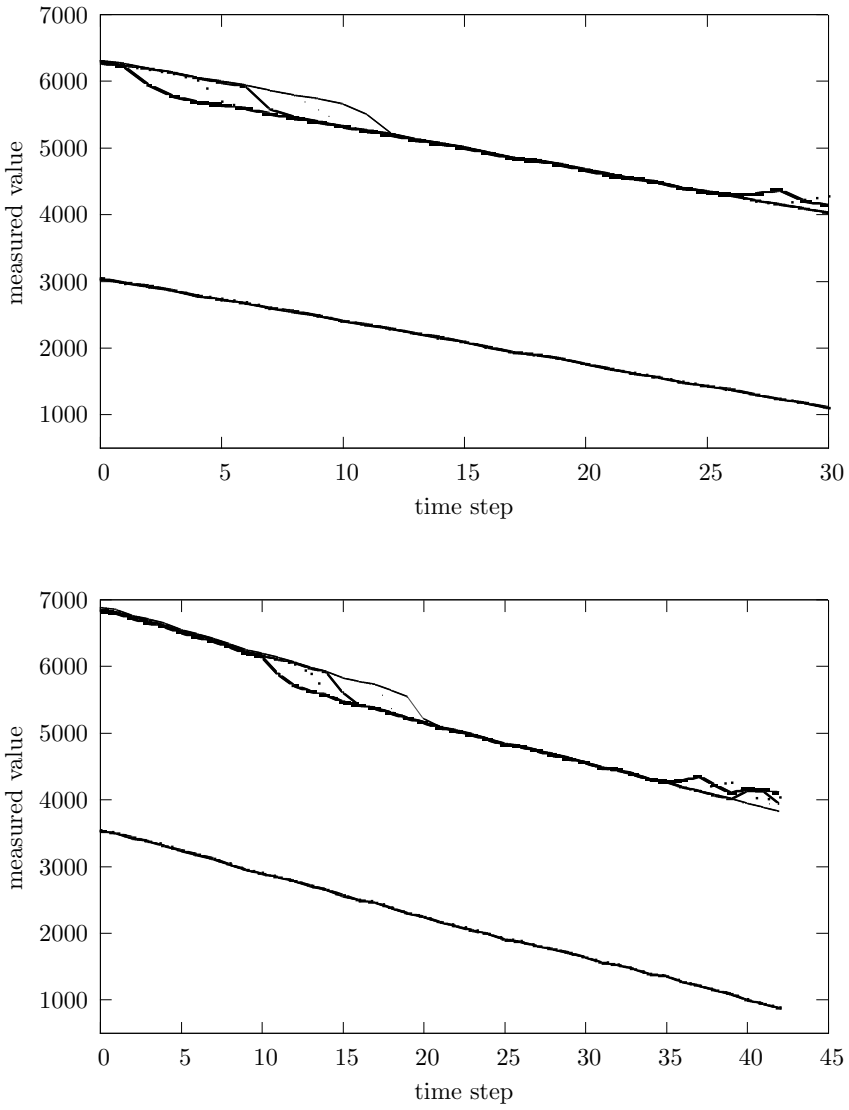


Figure 7.18: Distances measured by the laser scanner as the b21r moved towards a set of partitions. Values from 5 beam positions 10^o off the centre front on both sides are plotted. Distances to obstacles measured by beams on the right front are much larger than those on the left.

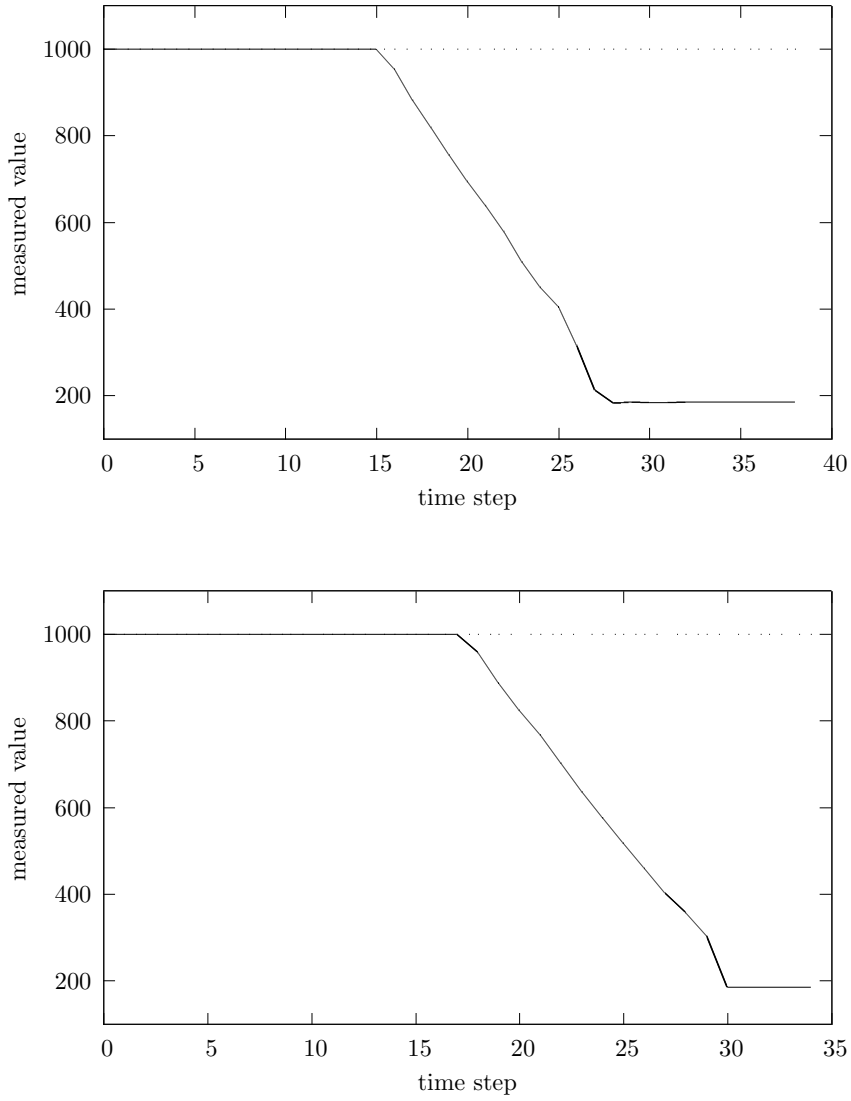


Figure 7.19: Distances to obstacles measured by sonar sensors as the b21r moved towards a set of partitions. The graphs show measurements from the sonar sensors on the immediate left and right of the front one. The two sensors are mounted at 15^o off the centre front.

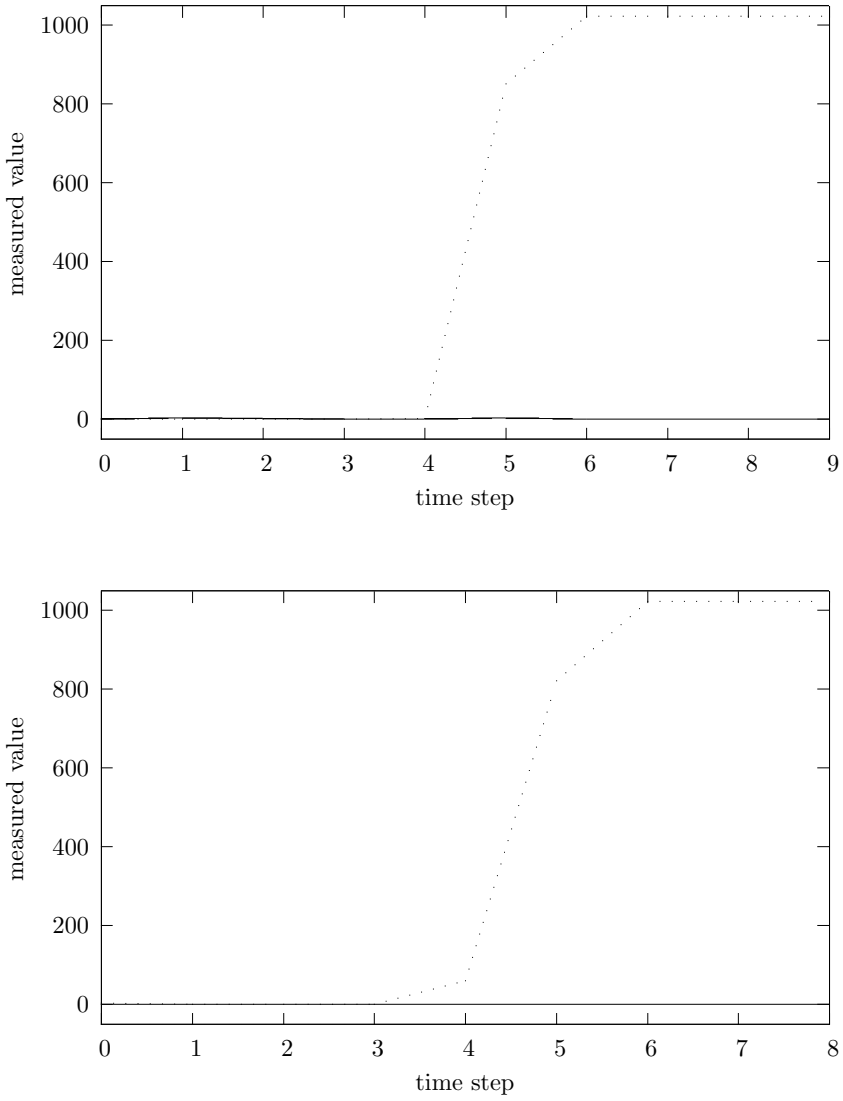


Figure 7.20: Distances to obstacles measured by infra-red sensors as the khepera moved towards two square blocks. The graphs show measurements from the two infra-red sensors mounted on the front (see Figure 6.2 on page 114).

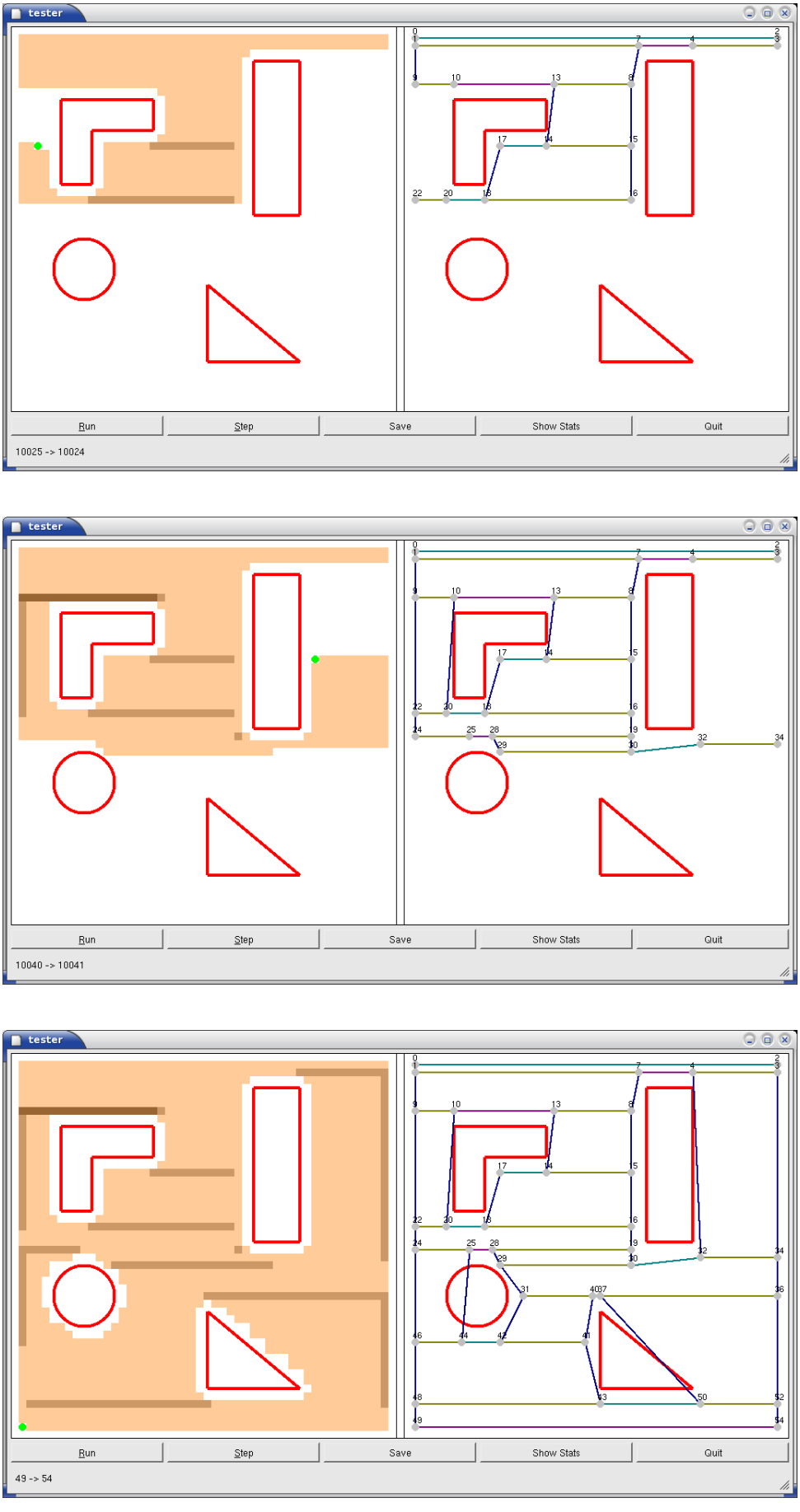


Figure 7.21: Coverage of one of the normal environments in simulation.

Figure 7.22 shows the coverage of another normal environment. Here two of the obstacles are put next to the boundary. As a result, the shape of the external boundary is altered.

The topological coverage algorithm was also tested in environments that differs from the normal ones. Figure 7.23 shows the coverage of a spiral. Figure 7.24 has a non rectilinear boundary. Figure 7.25 shows two office-like environments with long thin obstacles representing partitions. All of these were completely covered.

7.2.2 Real Robot

The correctness of the topological coverage algorithm was also tested with the Khepera robot. Four different environments were used. The environment in Figure 7.26(a) has a rectilinear boundary and no free standing obstacles. The environment in Figure 7.26(b) has a rectangular boundary and one square obstacle. The environment in Figure 7.26(c) has a rectangular boundary, one circular and one elliptical obstacle. Lastly, the environment in Figure 7.26(d) has a non-rectilinear boundary and a circular obstacle.

The pictures in Figure 7.26 are composite images displaying coverage results from the experiments. The composite images have been rotated so the robot always started at the top left corner and faced right. Moving on an average of 0.012 m/s (1.2 cm/s), the robot takes approximately 5 minutes on average to cover the environments inside the tray.

The output of the Hough accumulator used in evidence gathering contains a list of centres and radii of the robot throughout the video sequence. A composite image that shows the area covered (like the ones in Figure 7.26) is created by drawing a series of filled circles corresponding to the list is drawn over the reference image. If only the centres are plotted, a composite image showing the path taken by the robot is created instead. Examples of this are shown in Figures 7.27. These images show the route the robot took in the experiments. It can be seen that the strips in the zigzag path were not perfectly horizontal. However, the robot still found and covered all the free space cells.

7.3 Zigzag as coverage pattern

Compared to other online coverage algorithm based on cell decomposition, the proposed topological coverage algorithm employs a more general technique for boundary detection. This is because cell boundaries are detected through topology changes, which are large features. As a result, a robot with range sensors can detect new cells all around itself, not just on its sides.

Existing cell decomposition based coverage algorithms require wall following on both side

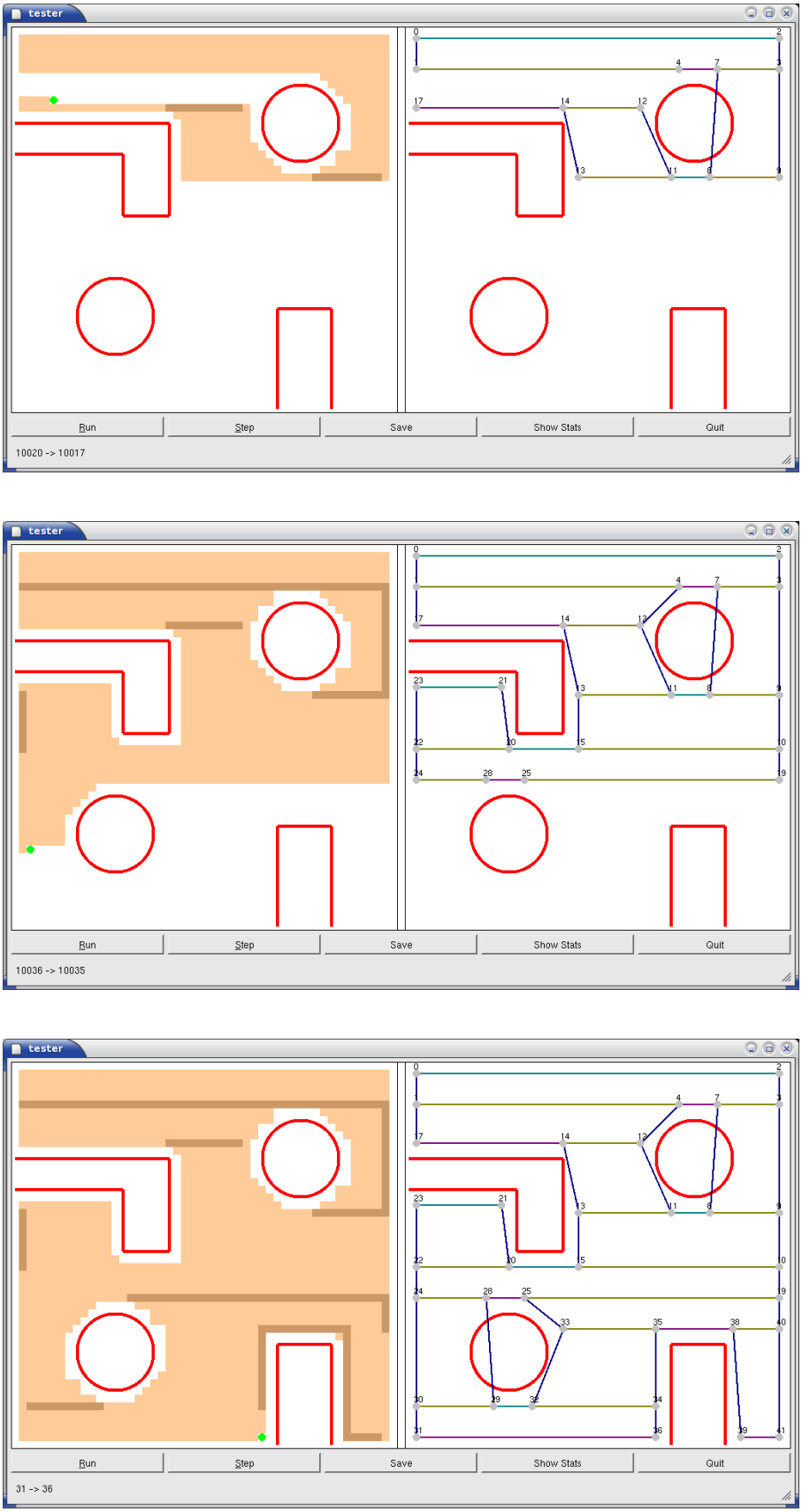


Figure 7.22: Coverage of another normal environment in simulation.

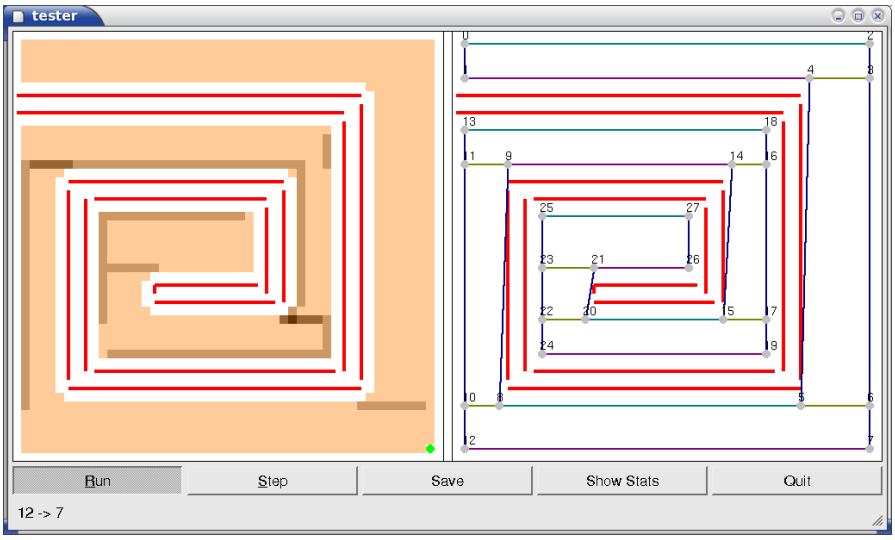


Figure 7.23: Coverage of a spiral.

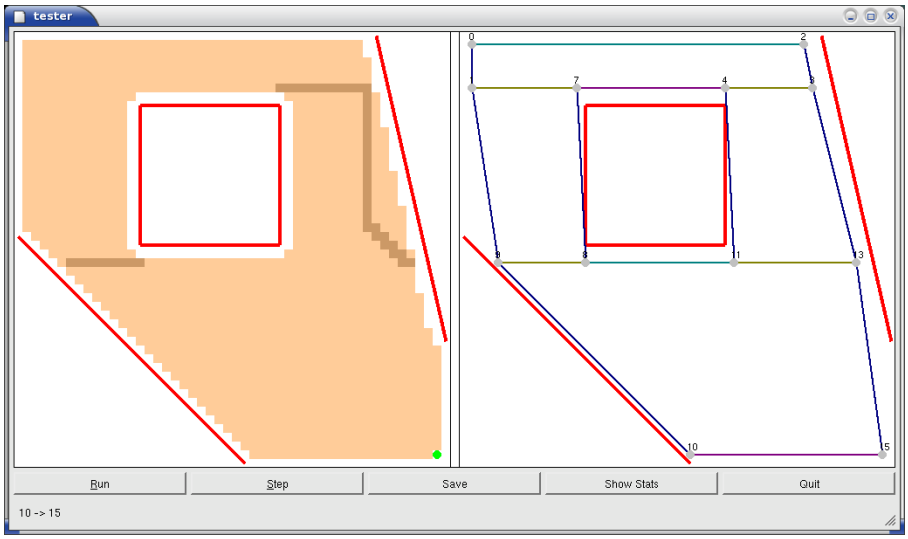


Figure 7.24: Environment with a non-rectilinear boundary.

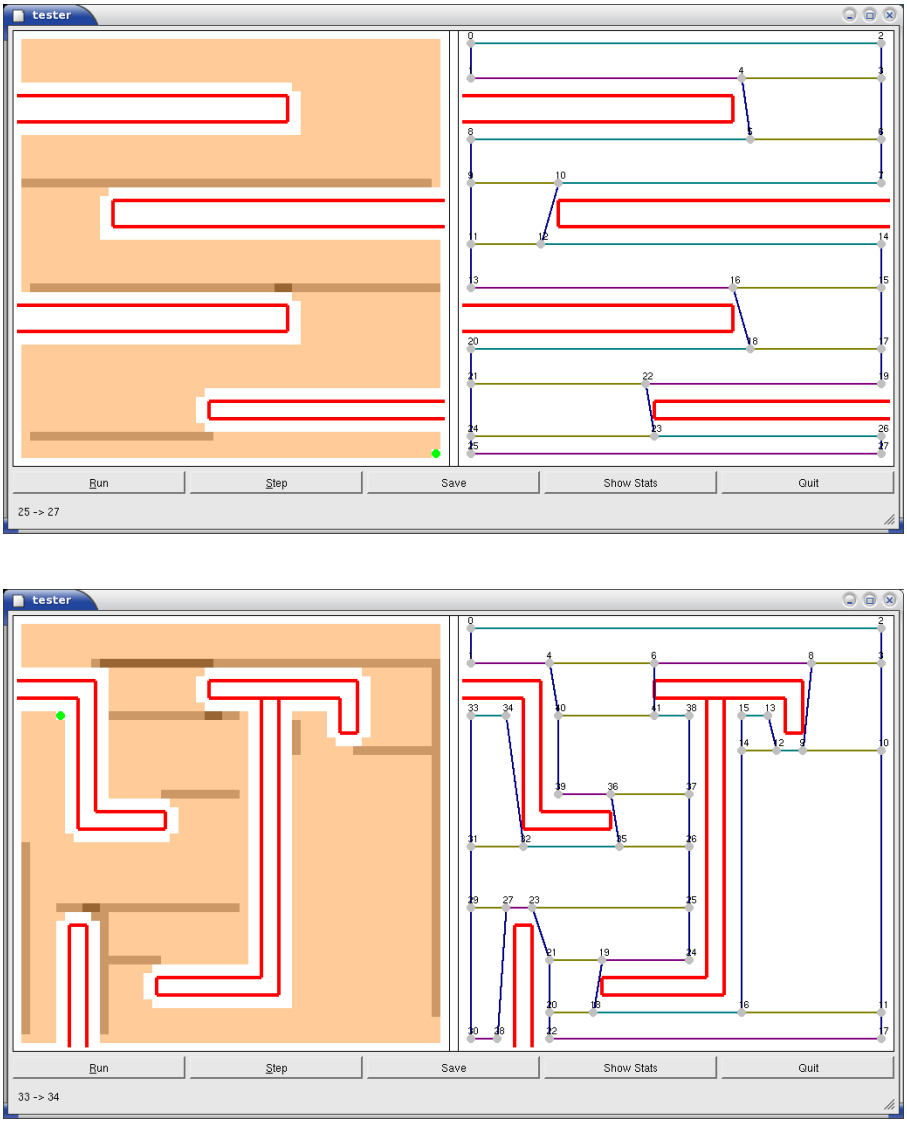
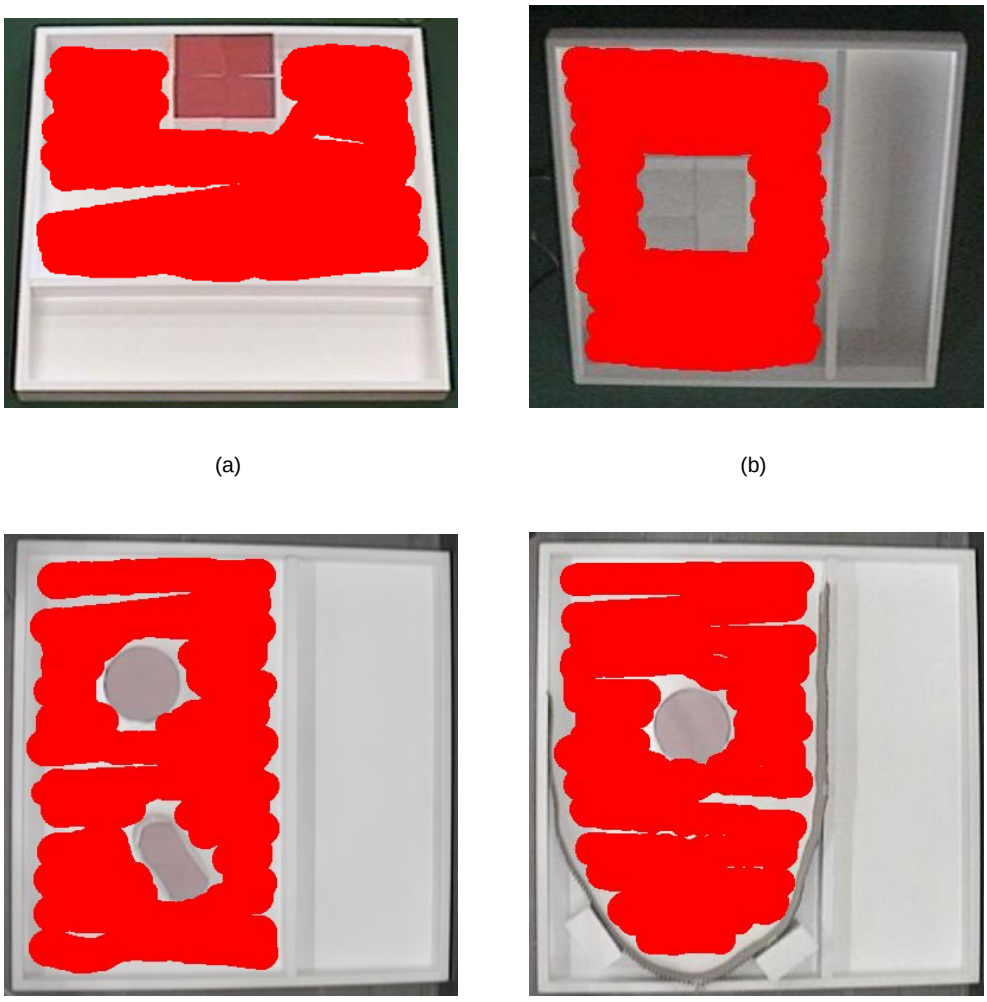


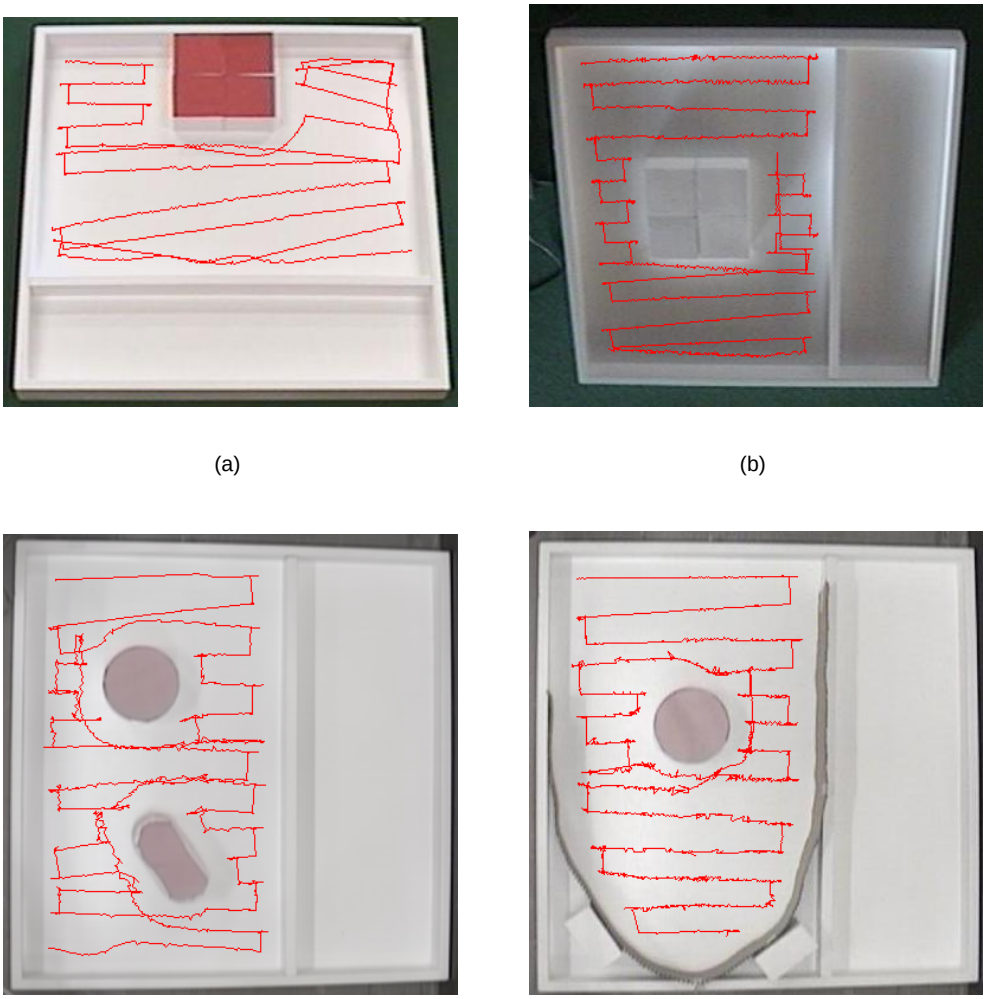
Figure 7.25: Two office-like environments with long thin obstacles.



(a) (b)

(c) (d)

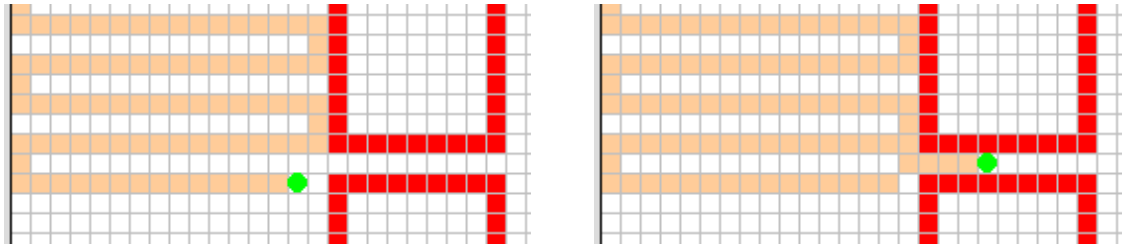
Figure 7.26: Coverage with the Khepera robot.



(a) (b)

(c) (d)

Figure 7.27: Path taken by the Khepera robot.



(a) (b)

Figure 7.28: Environment with a gap on the right wall that falls between strips of the zigzag, but large enough for the robot to enter: (a) approaching the opening, (b) gap found and entered.

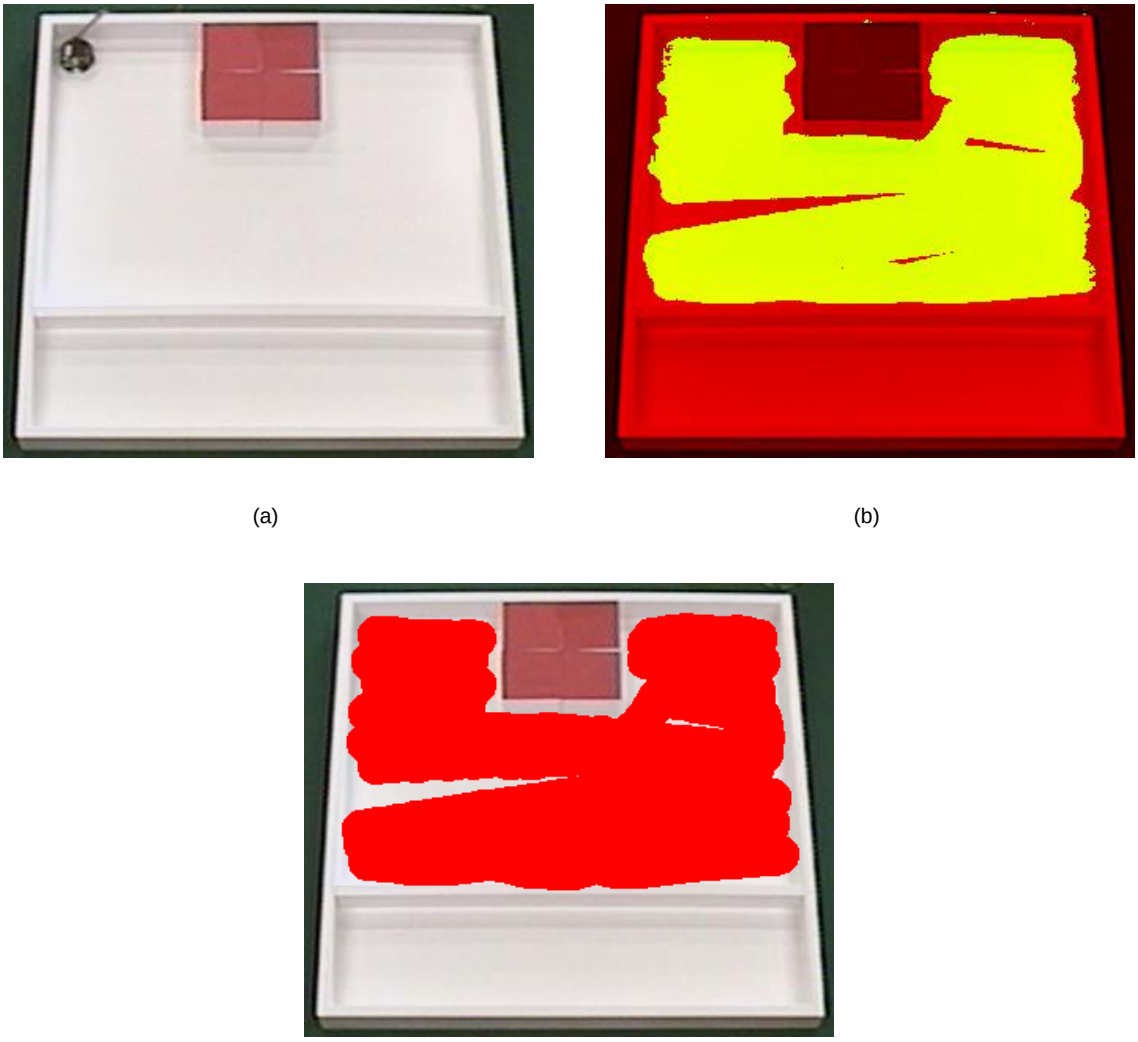
boundaries to detect openings such as the ones in Figure 2.24 on Page 26. In topological coverage algorithm, this kind of opening is handled as a combined merge and split event. Section 7.1.2 demonstrates the ability to detect this event with different range sensors.

Figure 7.28(a) shows the simulated robot approaching a gap that lies between consecutive strips of the zigzag. The robot detects the gap while moving towards the side boundary. When it arrives at the side boundary, operation switches to boundary state. The robot enters the gap to continue exploring the new cell boundary (Figure 7.28(b)). In the figure, the inter-strip distance Δx has been increased. This is because normally consecutive strips are on consecutive rows in the uniform grid of the simulated environment. Therefore, there are no gaps between strips. Only by increasing the inter-strip distance Δx, can an environment with an opening in the side boundary that lies between consecutive strips can be created.

As the proposed topological coverage algorithm does not require a coverage pattern that includes retracing, the path length required to cover a given environment is shorter than existing cell decomposition based online algorithms.

7.4 Evaluating composite images

Two methods for creating composite images are introduced in this thesis. The first method, image subtraction, is efficient and simple to perform, but suffers from two major disadvantages. Firstly, it is not particularly reliable as artefacts frequently appear in the background subtraction process; this is because of environmental factors such as lighting. Additionally, in the case of a tethered robot (such as the Khepera) the cable motion is computed as part of the final composite image. The second method, evidence gathering, includes more post-processing after the subtraction step. The images are edge detected and then a Hough transform is used to find the most likely match of the robot in each frame. As a result, evidence gathering is very tolerant to different lighting conditions. However, the Hough transform is computationally intensive.



(a) (b)

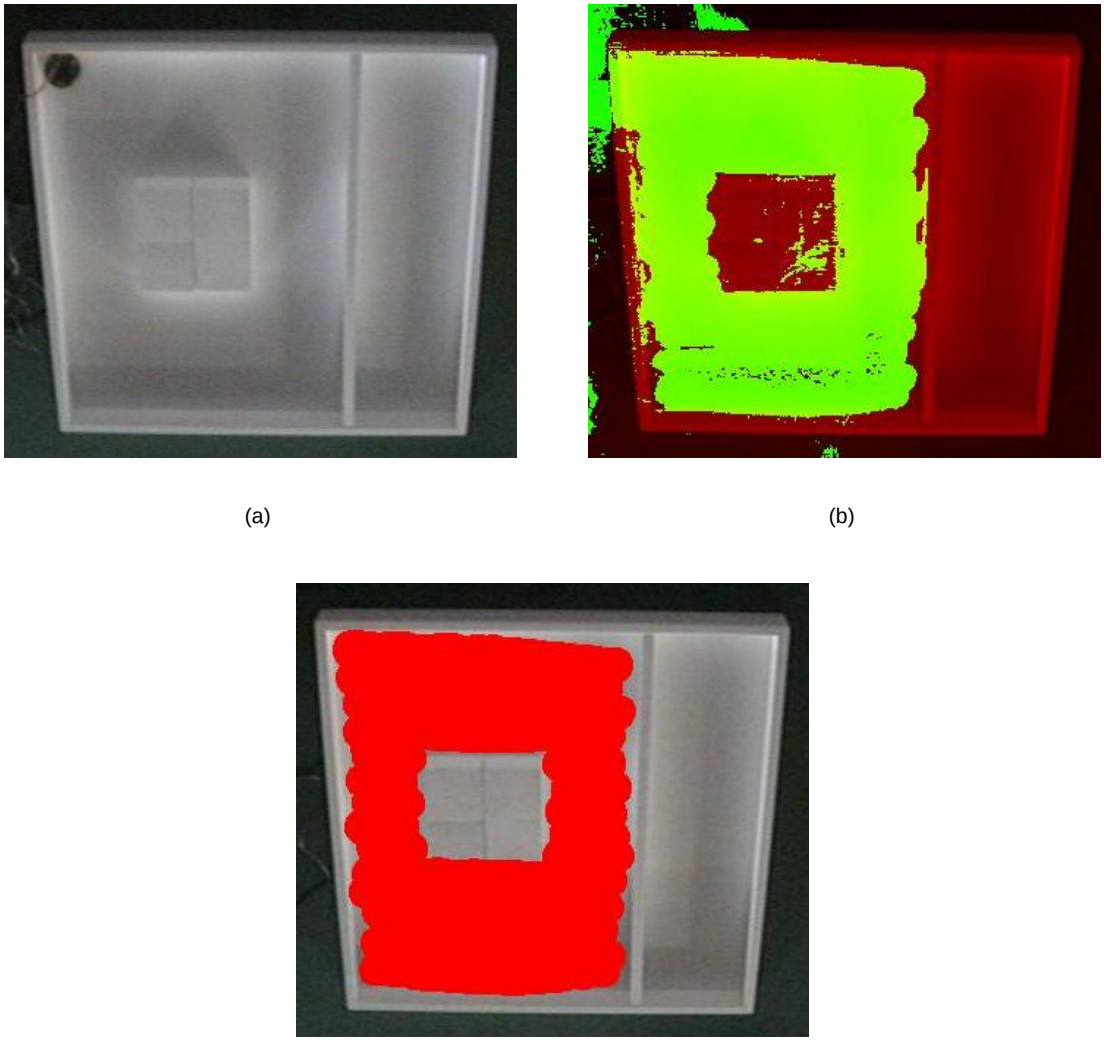
(c)

Figure 7.29: Frames taken under good lighting condition. (a) Reference background image. Composite image created using (b) image subtraction, (c) evidence gathering.

Therefore, the two methods are a tradeoff between speed and accuracy.

Figure 7.29 shows composite images created from frames taken on a sunny day with good lighting conditions. Figure 7.29(b) is created using image subtraction, and Figure 7.29(c) with Hough transform. The good lighting produces good contrast between the robot and the background. Therefore, the resultant composite images created from the two methods look similar.

Figure 7.30 shows composite images created from frames taken on a rainy day. With poor lighting conditions, it can be seen that the contrast in the original images (Figure 7.30(a)) is a lot lower. Figure 7.30(b) is created with image subtraction, and Figure 7.29(c) is produced with Hough transform. It is obvious that the resultant composite image created by evidence gathering is better. For example, in the composite image created using subtraction, areas that



(a) (b)

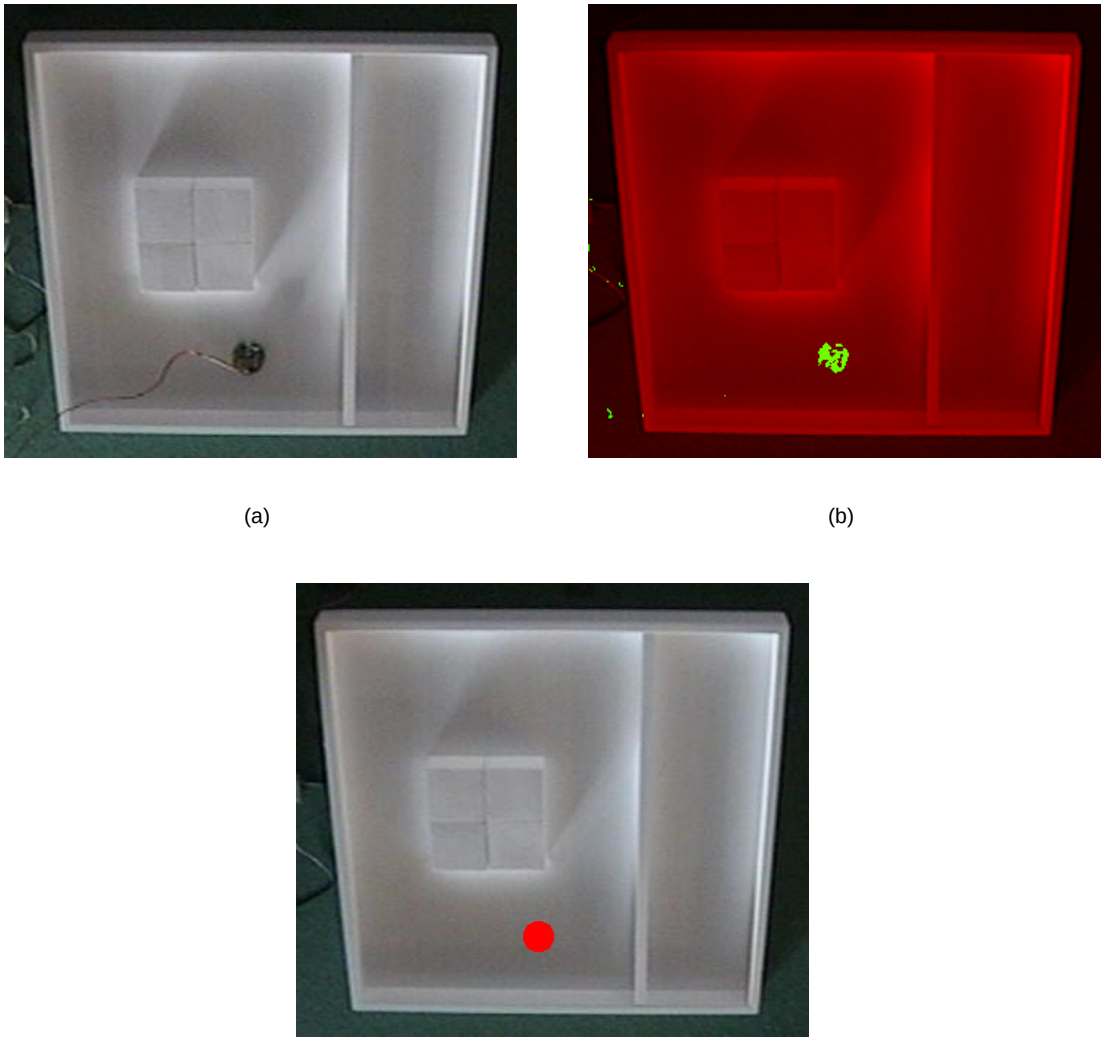
(c)

Figure 7.30: Frames taken under poor lighting condition. (a) Reference background image. Composite image created using (b) image subtraction, (c) evidence gathering.

the robot could not possibly reach are classified as covered, like the boundary of the tray or on top of the obstacles.

The tiny gaps present in the top part of the tray in Figure 7.30(b) are artefacts from the image subtraction method. Examining the original frames showed that the region was properly covered by the robot. This occurs because the robot extracted using image subtraction sometimes is incomplete. This effect is illustrated in Figure 7.31. Figure 7.31(b) shows the result of extraction using image subtraction. The robot is incomplete and does not appear solid. In comparison, Hough transform does not suffer from this problem, as seen in Figure 7.31(c).

To evaluate the two methods for creating composite images quantitatively, a sequence of 100 frames was hand marked to provide the ground truth. One image from this hand marked se-



(a) (b)

(c)

Figure 7.31: (a) Extraction from a single frame. (b) Robot extracted using image subtraction appears “porous”. (c) Hough transform does not suffer from this effect.

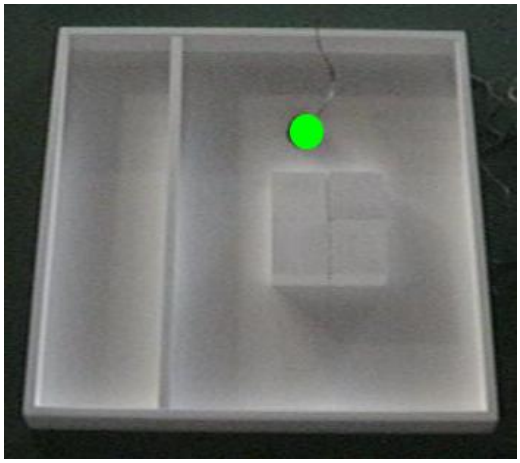


Figure 7.32: A sample image from the hand marked sequence. The position of the robot is coloured manually.

	Mean	Std. dev.	Combined
Subtraction	269.81	45.80	740
Hough transform	155.18	41.12	594

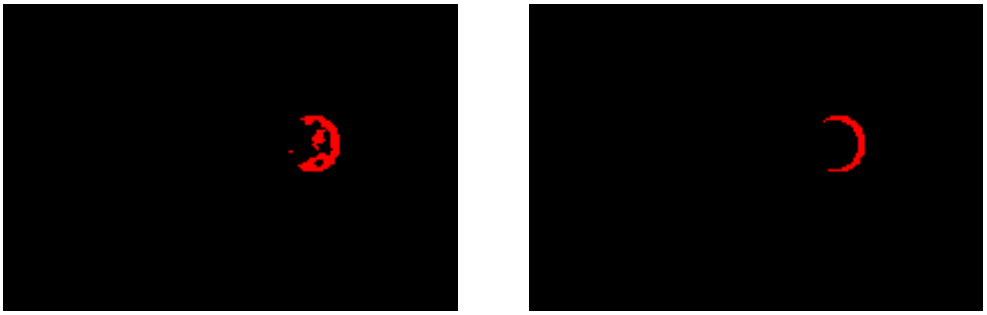
Table 7.1: Misclassification (in number of pixels) compared with a sequence of hand marked images. There are 100 frames in the image sequence. The table shows the mean number of misclassified pixels, its standard deviation, and the combined total in a composite image.

quence is shown in Figure 7.32. The position of the robot in each frame from the hand marked sequence was then compared to that extracted with background subtraction and Hough transform.

All pixels in the images belong to one of two classes - the robot or the background. Comparison was done by tallying the number of pixels classified into the wrong category. For example, if a pixel is classified as being part of the robot by the Hough transform (or background subtraction), but belongs to the background in the hand marked frame, then it is a wrong classification; similarly, a pixel classified as part of the background by Hough transform, but belonging to the robot in the hand marked frame is also a misclassification.

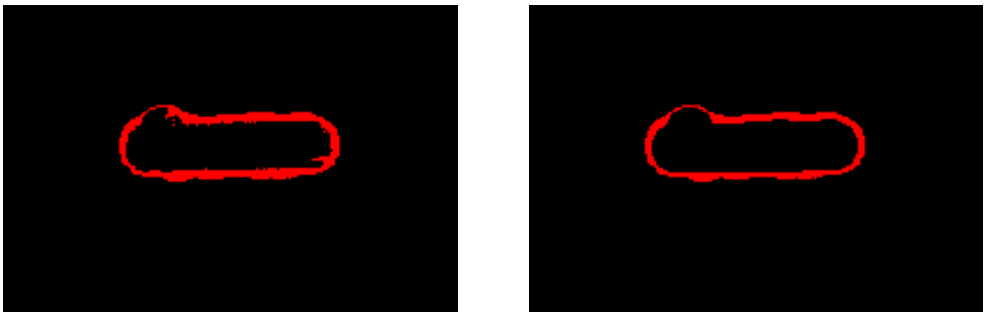
Table 7.1 shows the results of comparison. It can be seen that background subtraction has a higher average number of misclassified pixels. This is also evident in the difference images in Figure 7.33, which highlights the misclassified pixels in a frame. The last column in the table shows the total number of misclassified pixels in the composite images created with all 100 frames. The difference composite images are shown in Figure 7.34. The results in Table 7.1 prove that Hough transform is a more accurate method for extracting robot positions from images.

A disadvantage of the Hough transform is its high computational cost compared to the image subtraction method. For a sequence of around 3000 frames, image subtraction takes only a few



(a) (b)

Figure 7.33: Misclassified pixels are coloured to highlight the differences between the hand marked image and the robot position extracted using (a) image subtraction, (b) Hough transform.



(a) (b)

Figure 7.34: Misclassified pixels in the composite image of 100 frames using (a) image subtraction, (b) Hough transform.

	Complexity	Path length L	Std. dev.
Normal (11)	6 - 14	1.08	0.04
Spiral	9	1.12	-
Irregular Boundary	4	1.03	-
Office (2)	9, 13	1.10	0.03

Table 7.2: Results from simulation for the normal, spiral, irregular-shaped and office-like environments. The complexity refers to the number of free space cells. The path lengths listed for normal and office are the averages of all environments in the category.

minutes, while evidence gathering takes several hours. However, since performance metrics are used offline for data analysis, the speed of the evidence gathering method is still acceptable as processes can be left running on a computer cluster overnight.

There are two possible ways to improve computation efficiency. First, the current implementation used Python, a scripting language. The speed of computation may be improved by implementation using a compiled language such as C. Secondly, in the current implementation, three parameters (x,y and radius) are used for model fitting with the Hough accumulator. Since the robot size does not vary much from one side of the tray to the other side, the dimension of the model can be reduced by varying only the centre of the model (x,y). However, neither method will make Hough evidence gathering real time.

7.5 Performance Metrics

In this section, performance metrics proposed in this thesis will be used to evaluate the results from simulated and real robot experiments quantitatively.

7.5.1 Simulation

All 15 simulated environments tested were fully covered, with percentage coverage $C = 100\%$. This is calculated using (5.4) on page 101. Since the simulated robot is programmed to keep a minimum distance of one cell from obstacles, cells immediately next to obstacles are non-reachable, and thus ignored in calculating the percentage coverage C. Table 7.2 summarises the path length metric L for these experiments. The complexity in the Table refers to the number of free space cells in the environment.

Icking *et. al.* proved an upper bound of $2N$ for complete coverage paths for unknown grid maps with N cells [56]. Using (5.5) on Page 102, the path length L for the upper bound can be found:

	Average	Std. dev.
Subtraction	92.9	2.94
Hough evidence gathering	91.2	1.13

Table 7.3: Coverage percentages C for experiments with the Khepera.

$$\begin{aligned} L &= \frac{\text{number of moves}}{\text{number of reachable grid cells} \times C} \\ &= \frac{2N}{N \times 1} \\ &= 2 \end{aligned}$$

Therefore, the path lengths L achieved in the simulated experiments are all within the theoretical upper bound.

7.5.2 Real robot experiments

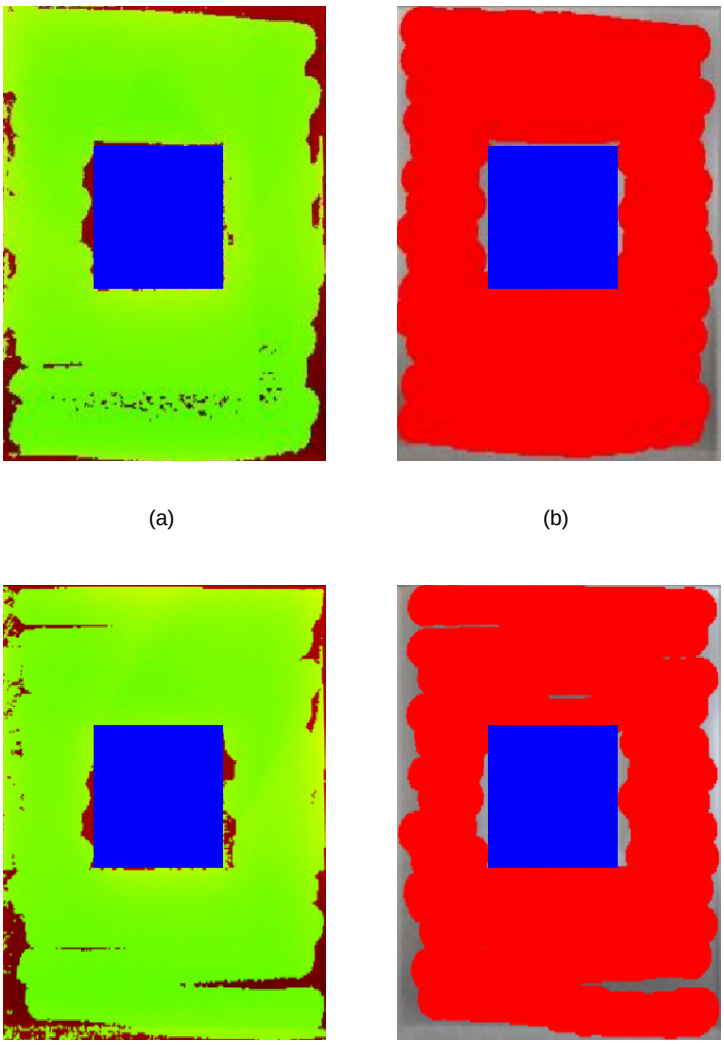
Percentage coverage

The amount of coverage achieved in an experiment is estimated from its composite image using pixel counts. Before doing a pixel count, the composite image is de-skewed and cropped, and the obstacles in the image are coloured. Figure 7.35 shows the result of such processing on the composite images. The obstacle is coloured blue because none of the original colours present (red, white and yellowish green) have blue in their RGB space.

Table 7.3 summarises the experimental results over 9 experiments. The figures in the table show the topological coverage algorithm performed the coverage tasks successfully. It can be seen that the average percentage coverage C calculated using image subtraction is slightly higher. This is evident from the composite images in Figures 7.30 and 7.35. Artefacts occurring from misclassification of background as part of the robot are more pronounced than incomplete extraction of the robot. However, this might not always be the case because it is not possible to predict the distribution of artefacts arising from the background subtraction process.

Path length

To calculate the path length metric L defined in (5.3) on Page 101, three parameters – coverage C, actual distance travelled $||P_a||$ and minimal path distance $||P_m||$ – are needed. The percentage coverage C is already calculated for the effectiveness metric. The actual distance travelled $||P_a||$



(a) (b)

(c) (d)

Figure 7.35: Before conducting a pixel count to estimate the percentage coverage, composite images are de-skewed, cropped and false coloured. (a) and (b) are created using image subtraction and evidence gathering from the same video sequence. (c) and (d) are from another video sequence.

Figure 7.36: The distance between consecutive strips can be found using a pair of images from the captured video sequence. The Hough accumulator returns the centre and radius of the robot extracted from the image. Since the real world radius of the robot is also known, the real world distance between the two centres can be calculated.

	Average	Std. dev.
un-scaled with percentage coverage (L')	0.99	0.04
subtraction	1.19	0.047
Hough evidence gathering	1.22	0.037

Table 7.4: Path length metrics for experiments with the Khepera.

by the robot is calculated using odometry information from the wheel encoders on the Khepera. The minimal path $||P_m||$ depends on both the total area of the environment and the inter-strip distance Δx. The total area can be obtained easily with a measuring tape. To estimate the inter-strip distance Δx, I use the list of radii and centres of the model fitted by the Hough accumulator. The inter-strip distance Δx is measured empirically because the Khepera is commanded to move between strip by turning on the motor for a specific amount of time. Two images from the ends of consecutive strips are used, as illustrated in Figure 7.36. The distance between the centres of the robot in the two images gives the inter-strip distance Δx in pixels. Since the radius of the robot is known both inside the image (from the radius of the model fitted) and in real life, the real world distance between the two strips can thus be calculated. This procedure was repeated with multiple pairs of images. It is found that the inter-strip distance Δx is the same throughout all the experiments, with only a variance of 1 pixel.

Table 7.4 shows the path length metric for the experiments carried out. The first row is the un-scaled path length measure $L' = \frac{||P_a||}{||P_m||}$ in (5.2) on Page 101. The average L' is smaller than unity, which implies that the actual path P_a is shorter than the minimal path P_m. This is of course impossible. It illustrates the importance of scaling the path length with the actual percentage coverage C achieved to obtain a meaningful metric. The next two rows are the path length metric L in (5.3), scaled with the percentage coverage C obtained from pure image subtraction and Hough evidence gathering respectively. The path length is now over unity, which means that the actual path taken is longer than the minimal path.

The path lengths L from experiments with the Khepera are longer than all the environments tested in simulations. There are two reasons behind this increase in path length. Firstly, the

percentage coverage C is below the 100% achieved in simulation. According to (5.3), a coverage C below unity will increase the path length metric L. Secondly, not all free space cells can be covered by an integer number of strips. In other words, the height of a free space cell may not be an integer multiple of the inter-strip distance Δx. As a result, the last strip in a free space cell maybe closer to the previous strip then normal. This can be seen at the bottom of Figure 7.27(b).

7.6 Composite image for non-circular robots

To extract a non-circular robot using Hough transform, a different model from the one used in Algorithm 5.1 on Page 107 is required. For example, for the B21r robot in Figure 7.37, an ellipse would be used as a model for the robot. The centres of the elliptical model fitted was drawn over each image in Figure 7.37. Note that the cylindrical robot does not appear circular because the camera is not mounted directly above the environment. If pure image subtraction is used to create the composite images, no changes are required because the method is not model based.

To extract an odd shaped robot or a tool carried by a robot, a marker has to be used. With Hough evidence gathering, a model is fitted for the marker, not the robot itself. With the image subtraction method, the entirety of the robot is extracted and centres are never found. This means that image subtraction cannot be used to create composite maps for coverage with tools that present partial views of the robot. In comparison, evidence gathering can fit a model to find the position of the tool instead of the robot position.

7.7 Path length L and complexity of environment

Figure 7.38 shows a plot of the path length metric L against the number of free space cells for the 15 environments in Table 7.2. The graph shows that environments with the same number of free space cells are not covered with the same path length L. For example, the path lengths L for the four environments with 9 free space cells range from 1.03 to 1.12. This difference arises because the re-coverage needed to completely cover an environment depends on the layout of the obstacles. However, it can be seen that there is a trend of increasing path length with an increase in the number of free space cells. This is because more travelling is required in environments with more free space cells.

The section investigates empirically the relationship between path length and the number of free space cells. Five simulated environments were created. The environments were essentially

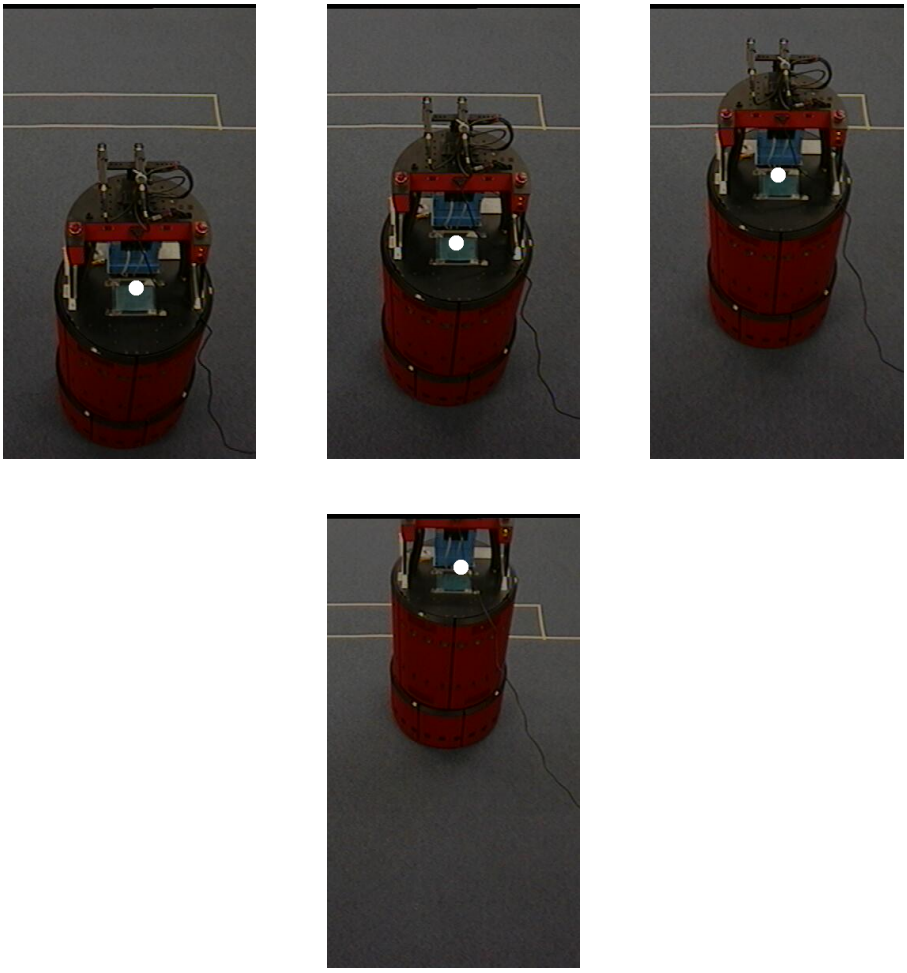


Figure 7.37: Extracting the location of a B21r robot using Hough evidence gathering. The white dots correspond to where the centres are found.

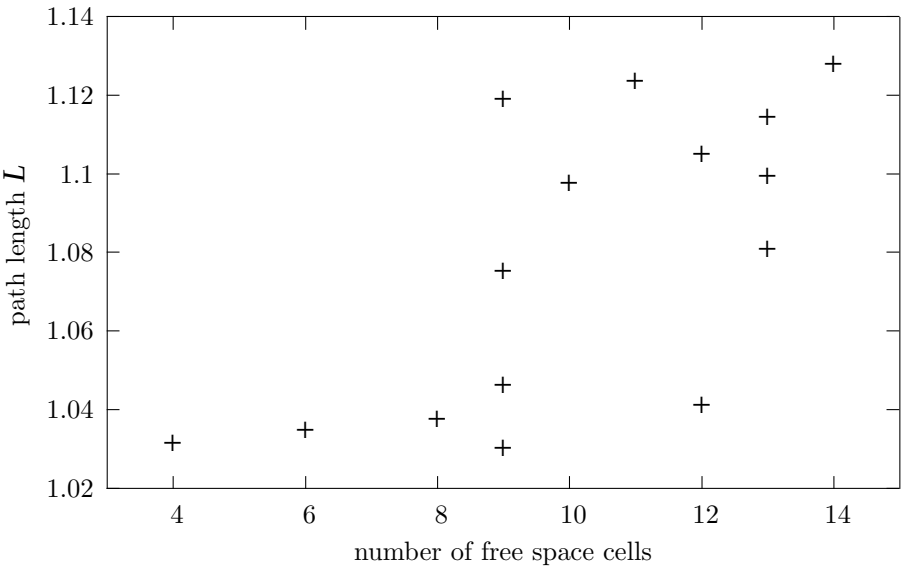


Figure 7.38: Path length L vs number of free space cells in random environments.

Figure 7.39: Simulated environments used to study effect of number of free space cells on path length L.

identical apart from the number of obstacles present. Rectangular obstacles were placed in a regular fashion as shown in Figure 7.39. The five environments have 2 to 10 obstacles, creating 5 to 21 free space regions. Results from these experiments are plotted in Figure 7.40. With nearly identical environments, the path length L appears to increase linearly with the number of free space regions.

The reason behind this linear increase can be explained using Figure 7.41. Consider the environment in Figure 7.41(a). Let's assume that the free space cells are covered in the sequence shown in the diagram. Also assume that the travel required to get from cell a and cell b is t_{a-b} in length. Then the total travel required in Figure 7.41(a) is

$$T = t_{1-2} + t_{2-3} + t_{3-4}$$

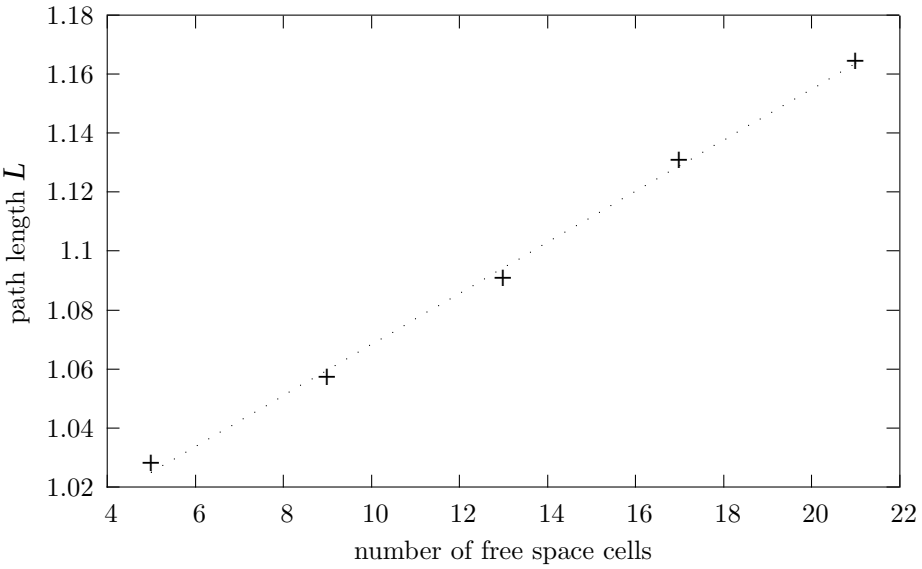


Figure 7.40: Path length L vs number of free space cells in environments with identical layouts.

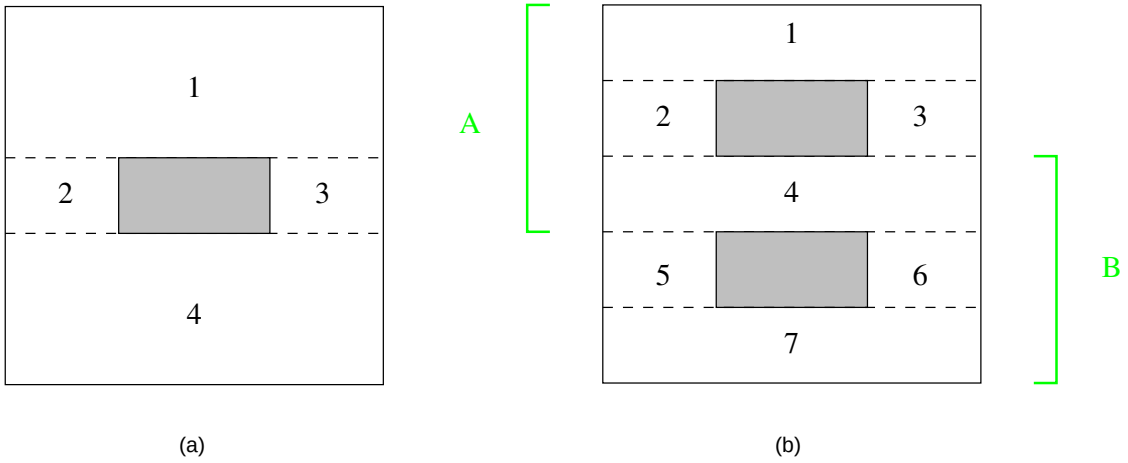


Figure 7.41: The two environments are essentially identical other than the number of obstacles present.

The environment in Figure 7.41(b) is essentially the same as that in Figure 7.41(a), but has two obstacles. Another way to view this is that the new environment contains two blocks — A and B. The two blocks are of the same configuration and share free space cell number 4. The total travel required is

$$\begin{aligned} T &= (t_{1-2} + t_{2-3} + t_{3-4}) + (t_{4-5} + t_{5-6} + t_{6-7}) \\ &= (t_{1-2} + t_{2-3} + t_{3-4}) + (t_{1-2} + t_{2-3} + t_{3-4}) \\ &= 2 \times (t_{1-2} + t_{2-3} + t_{3-4}) \end{aligned}$$

In summary, both the layout and the number of obstacles affect the length L of a coverage path. With more free space cells, there is an increase in the number of times the robot needs to move

between cells. However, the distance the robot need to travel to get from one cell to another depends on the placement of obstacles in the environment.

7.8 Summary

The landmark detection experiments show that events in the topological coverage algorithm are easily and accurately detected by range sensors. The tests also confirm that landmarks are large features that appear across multiple time steps.

Coverage experiments in simulation tested the correctness of the proposed topological coverage algorithm. The tests confirm that the algorithm can correctly and completely cover unknown environments, using a partial topological map created during the coverage process. The experiments also show that the topological coverage algorithm can handle a wider variety of environments than previous cell decomposition based coverage algorithms. This includes non-polygonal obstacles (not possible in boustrophedon decomposition and CC_R) and obstacles with surfaces parallel to the sweep line (not possible in Morse decomposition).

In addition, sensor tests and simulation experiments show that unlike existing online cell decomposition based coverage methods, the topological coverage algorithm does not require wall following on the side boundaries of cells. Therefore, a shorter coverage pattern that does not include retracing can be used. The overall coverage path generated from the topological coverage algorithm is thus shorter compared to existing algorithms.

Tests on the Khepera robot empirically demonstrate that the proposed algorithm is indeed viable under real, inexact conditions with sensor and actuator errors.

Accuracy of the composite map used in the proposed performance metrics was compared with a hand marked sequence of 100 frames. Hough evidence gathering was found to be the better method for extracting positions of the robot in a video sequence. The performance metrics were then used to evaluate the simulated and real robot experiments empirically. In the simulated experiments, all environments were completely covered (ie C = 100%), with an average path length L of 1.08. In the real robot experiments, the average coverage C was 91.2%, with an average path length of 1.22. The path length L achieved in both real and simulated experiments was lower than the upper bound of 2 proved by Icking *et. al.* .

The fool doth think he is wise, but the wise man knows himself to be a fool.

William Shakespeare, "As You Like It", V.i.31

8

Future Work and Conclusions

8.1 Future work

8.1.1 Tethered robot

Slice decomposition is defined for both free-moving and tethered robots. However, the current simulation environment can only handle free-moving mobile robots. Therefore, an obvious area for future development is to add support for tethered robots in the simulation. This extension would allow development and testing of the online topological coverage algorithm with such restricted movement.

One addition required to the simulation engine is the calculation of whether a robot can reach a target location given its present location and the configuration of its tether. A visibility graph can be used for this purpose. It allows for rapid calculation of the minimum length of the tether between the anchor point and the robot, around a given set of obstacles. This use of the visibility graph is slightly different from its normal use in navigation [66] to find the shortest path. This difference is illustrated in Figure 8.1. In Figure 8.1(a), the shortest path between the robot and the anchor point is highlighted. However, if the robot moves between the obstacle to reach the current point, the minimum tether length required to reach the current location is actually the distance shown in Figure 8.1(b). This is because the tether is "tangled" between

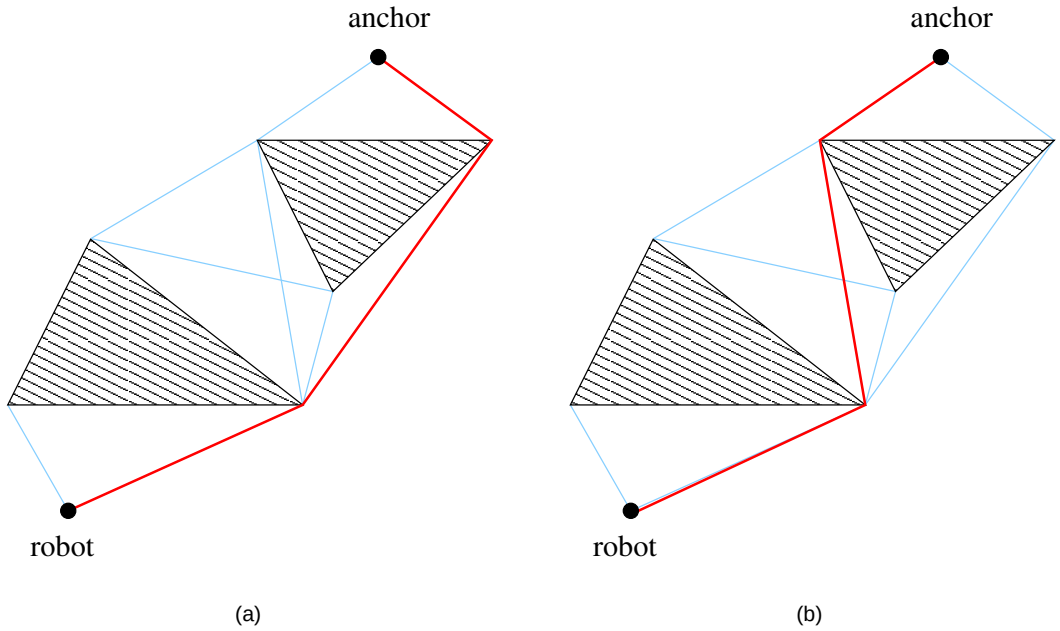


Figure 8.1: A visibility graph. (a) Shortest path between robot and anchor point. (b) Minimum tether length to reach current location if robot passes between the obstacles. This is different from the shortest path because the tether is ”tangled” between the obstacles.

the obstacles. To determine whether the robot can reach a certain location, the minimum tether length is compared with the actual tether length.

8.1.2 Simultaneous localisation and coverage (SLAC)

Although topological maps are tolerant to errors in pose estimation, the performance of the coverage operation is still affected. This is because the amount of coverage achieved within a cell is dependent on the direction of the zigzag. Therefore, incorporating localisation correction within the framework will improve coverage performance.

Recent advances in SLAM (simultaneous localisation and mapping) have greatly improved mobile robot localisation. SLAM uses statistical techniques to correct the robot’s pose (position and orientation) estimation. By adapting existing SLAM methods to coverage navigation, localisation in the topological coverage algorithm can be improved. Of particular interests are SLAM techniques designed for topological maps, such as the work on Generalised Voronoi graph by Choset and Nagatani [31], and the hybrid topological map with laser signatures (signatures stored with the nodes) by Tomatis *et. al.* [96].

8.1.3 Multi-robot coverage

Another interesting extension is to develop a multi-robot version of the topological coverage algorithm. For small domestic environments, it is more appropriate to use a single robot solution. For large scale foraging and demining, a multi-robot team could significantly shorten the time required to finish the task.

The major issue here will be the coordination of the robot team to perform this complex, global task (mapping and complete coverage). Simply increasing the number of robots in the team does not improve efficiency of achieving the operational goal.

There are three main approaches to coordinating multi-robot teams — fully centralised, fully distributed and auction/negotiation-based [34]. In fully distributed frameworks [43, 83], robots do not explicitly work together, but group-level cooperation behaviour emerges from their interaction with each other and the world. This type of architecture is popular in the field of swarm robotics. Since the effort of the team is not coordinated, complete coverage cannot be guaranteed until one member of the team has fully covered the environment. Thus a fully distributed approach is not suitable for this application.

In a fully centralised multi-robot team, a single robot or central computer acts as the leader of the team. Team members report their findings and status to the leader. The leader is completely responsible for planning the actions of all team members. A fully centralised architecture will be the easiest way to implement a multi-robot version of the topological coverage algorithm. This is because a single topological map can be maintained by the leader. This type of architecture is good for a small robot team, but does not scale well with increases in team size. The bandwidth and computational requirements of the leader grow very rapidly as the number of robots in the team increases. The leader also represents a single point of failure.

The most common approach to coordinating multi-robot teams is the auction/negotiation-based architecture [47, 112]. Here, tasks are traded among team members, but the individual robots are responsible for their own planning. Communications are limited to offers, bids and awards of tasks. To implement the topological coverage algorithm into an auction-based framework, further work needs to be done to find an appropriate task decomposition scheme, a suitable cost function and a negotiation protocol.

8.2 Conclusions

Topological maps represent features in the environment using topological relationships between landmarks. This is similar to the way animals represent their spatial environments [77, 82].

Topological maps are robust against sensor and odometry errors because only a global topological consistency, rather than a metric one, needs to be maintained [94]. However, due to their qualitative nature, it is difficult to store coverage information in a topological map. This is because nodes and edges in the map do not correspond to specific locations in space. This thesis tackles the representation problem by embedding a cell decomposition, called slice decomposition, within the topological map. This is achieved by using landmarks in the topological map as cell boundaries in slice decomposition. As a result, even though individual nodes in the topological map are not associated with specific areas of space, a combination of nodes now defines a region (a cell) bounded by obstacles.

Although the starting point of this thesis is to investigate coverage with landmark-based topological maps, the method proposed ultimately creates a cell decomposition similar to the split and merge concept in boustrophedon decomposition [30]. However, the work on boustrophedon decomposition is conceptual in nature. It does not provide a detailed algorithm for the decomposition, nor does it define the criticality precisely. It is also unclear if, or how, concave obstacles are handled. Lastly, boustrophedon decomposition is defined for known environments only.

Slice decomposition, CC_R [23] and Morse decomposition [8] all extend the use of split and merge events to unknown environments. CC_R is different in that it is designed for contact sensing robots operating in rectilinear environments. Similar to slice decomposition, Morse decomposition is for range sensing robots covering general unknown environments. The difference between Morse decomposition and slice decomposition is in the choice of cell boundaries. Morse decomposition uses surface gradients of obstacles as cell boundaries. An event occurs when a surface gradient is perpendicular to the sweep line. As obstacles parallel to the sweep line are non-differentiable, rectilinear environments cannot be handled by Morse decomposition. In comparison, slice decomposition uses topology changes in segments to define cell boundaries. Due to the use of simpler landmarks, slice decomposition can handle a larger variety of environments than Morse decomposition, including ones with polygonal, elliptical and rectilinear obstacles.

Moreover, as landmarks are large features, they can be easily detected via range sensor thresholding. As a result, events can be detected from all sides of the robot. This is confirmed with the landmark detection tests in Section 7.1. In comparison, events in Morse decomposition can only be detected if the critical point on the obstacle is closest to the robot than any other point on the obstacle. This is due to the difficulty in detecting surface gradients. Therefore, an U-shaped coverage pattern that includes wall following on both side boundaries is used to cover individual cells. This U-shaped pattern includes retracing. The topological coverage algorithm can use the simpler zigzag pattern for covering individual cells. Without any retracing in the coverage pattern, the topological coverage algorithm generates shorter, and thus more efficient,

coverage paths.

It is important to have a quantitative measure on how well an algorithm performs. While there are metrics that measure the performance of coverage experiments in simulation, there are no satisfactory ones for real robot tests. The only existing metric for real robot experiments is the coverage factor [22]. However, it is a poor measure of both effectiveness and efficiency. Thus, this thesis proposed two performance metrics for evaluating coverage experiments. The metrics are for both simulation and real robot experiments. The methods used to extract data needed are robot platform independent. The first metric is percentage coverage, which measures the effectiveness of an experiment. In simulation, this is the percentage of grid cells covered (same as the one used by Gabriely and Rimon in [45]). In real robot experiments, a composite image of the experiment is created using computer vision techniques. Then, the percentage coverage is estimated using the number of pixels the robot has appeared in the composite image. The second metric measures the efficiency of the experiment in path length. Since finding the optimal coverage path is an NP-hard problem, the concept of minimal path is introduced. The minimal path is the shortest coverage path for a mobile robot that can teleport with no cost associated with the teleport operation. The actual path taken by the robot is normalised against the minimal path and the percentage coverage. In simulation, the path lengths are measured in number of grid cells. This metric is an improvement over the use of repeatedly covered cells by Gabriely and Rimon because it takes into account multiply covered cells and actual area covered. In real robot experiments, the actual path length is taken from wheel encoder readings; while the minimal path is estimated using the area of the environment and the diameter of the robot.

The two performance metrics are applied to results from both simulated and real robot experiments. In simulation tests, 100% coverage was achieved for all experiments, with an average path length of 1.08. In real robot tests, the average coverage and path length attained were 91.2% and 1.22 respectively.

In summary, this thesis has made the following major contributions to the area of complete coverage path planning for mobile robots. Firstly, it developed an online coverage algorithm that uses a partial topological map of large features in the environment for path planning. Secondly, it introduced slice decomposition, a cell decomposition for covering unknown environments. It can handle a larger variety of environments than existing cell decomposition based coverage algorithms. Thirdly, due to the use of simpler landmarks as cell boundaries, the proposed coverage algorithm employs a shorter navigation pattern than existing methods. Lastly, new performance metrics for evaluating real robot coverage experiments are developed. These new metrics measure the experiments more reliably and accurately than existing metrics.

Oh, yes. The important thing about having lots of things to remember is that you've got to go somewhere afterwards where you can remember them, you see? You've got to stop. You haven't really been anywhere until you've got back home. I think that's what I mean.

Terry Pratchett, "The Light Fantastic"

A

Landmark Recognition using Neural Networks

This chapter outlines how supervised neural networks can be trained to correctly recognise and classify the topology changes used in the topological coverage algorithm. The chapter starts with a description of how classification works in the neural networks paradigm (Section A.1). Then the architectures and training algorithms for two supervised neural networks are explained. The two networks are multi-layer perceptron (Section A.2) and learning vector quantisation (Section A.3). Finally, Section A.4 presents the implementation and testing of these two networks in the landmark recognition task.

A.1 Pattern classification with Neural Networks

Classification is the problem of assigning new inputs to one of a number of discrete classes. A simple way to achieve this is to analyse sample data manually, and then establish a set of rules that captures the distinctive features of the different classes. For example, we might use the rules in Figure A.1 to distinguish between dogs, tables and vases.

In general, the classification problem is a non-linear mapping from several input variables to several output variables. In the previous example, the input could be video, sound and/or tactile

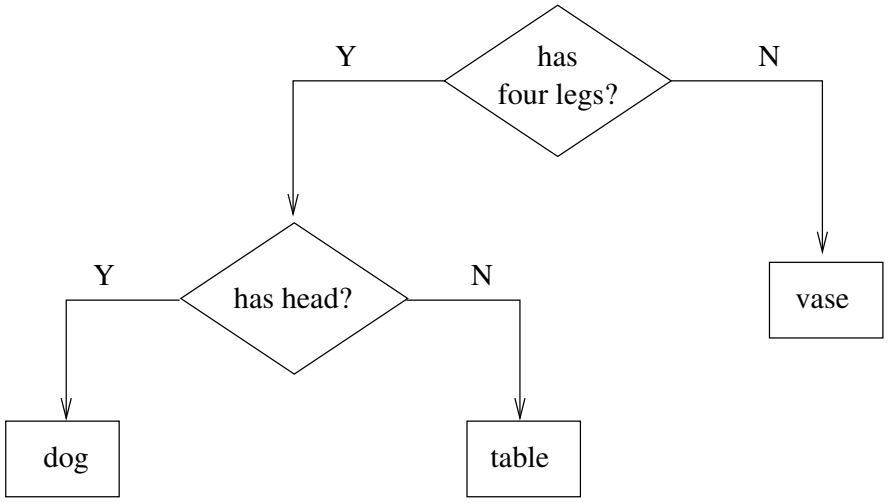


Figure A.1: Simple rules to classify dogs, tables and vases.

samples of the objects; the output variable has three values, one for each class.

Neural networks offer a very powerful and very general framework for mapping arbitrary input variables to another set of output variables, where the form of the mapping is governed by a number of adjustable parameters [17, 50]. Figure A.2(a) shows a hypothetical classification problem involving two independent input variables x_1 and x_2. Neural networks perform classifications by creating decision boundaries in the input space. For the example in Figure A.2(b), new samples that lie to the left of the decision boundary are classified as belonging to class C_1; while samples to the right of the decision boundary are classified as belonging to class C_2.

Training neural networks therefore involves creating decision boundaries to minimise misclassification errors. The decision boundary is altered with the adjustable parameters of the neural network. The training is done with appropriate machine learning algorithms, and is dependent on the type of network used.

By using a neural network, the designer of a classification system no longer has to manually devise a set of rules to separate the input samples into their designated classes. The computer can automatically extract the relevant features to create the classification system required. Examples of classification using neural networks in robotic systems include detecting defective areas in waste pipes and drains [36], and classifying outdoor road scenes with panoramic images [110].

A.2 Multilayer Perceptron (MLP)

Figure A.3 shows a picture of a multi-layer perceptron (MLP). It is the most common type of neural network used. The network is made up of several layers of artifical neurons. Each neuron is connected to every neuron in the immediately adjacent layers. The network can have many

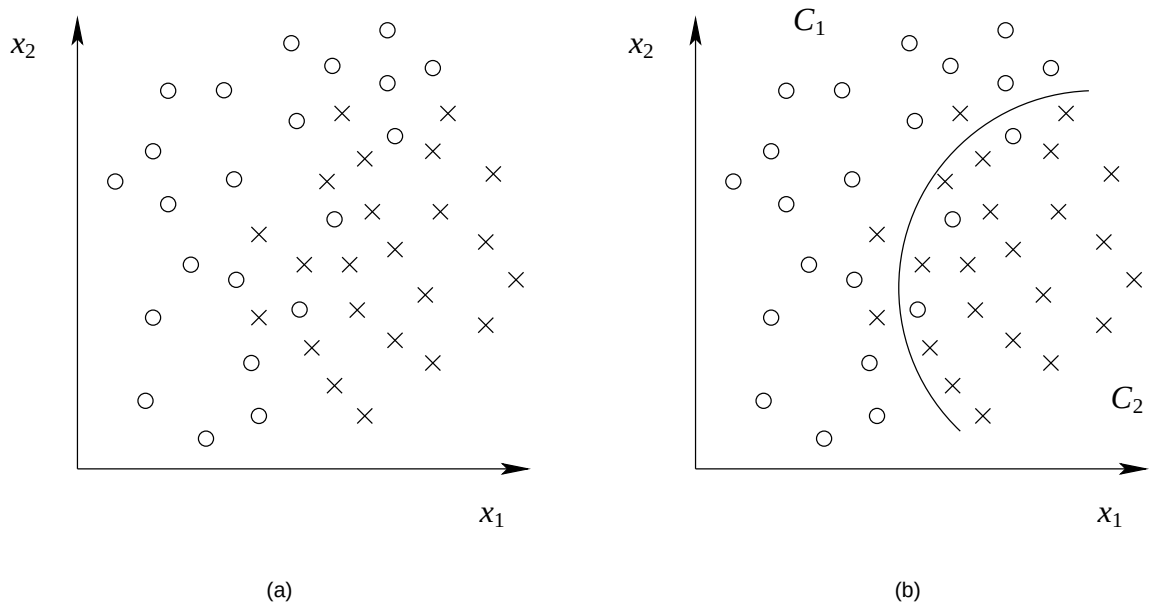


Figure A.2: (a) A classification problem with two variables. Circles (○) denote samples from class C_1 and crosses (×) denote samples from class C_2. (b) Neural networks perform classification by forming decision boundaries. (Adapted from [17]).

hidden layers, but only one input and one output layer.

An MLP estimates the decision boundary in terms of composition of the activation function(s) of the network. The hidden layer(s) usually uses a sigmoid function,

$$g(a) \equiv \frac{1}{1 + \exp(-a)} \tag{A.1}$$

while the output layer normally uses a linear activation function[1]

$$\tilde{g}(a) = a \tag{A.2}$$

However, other functions can also be used. This concept of estimating a non-linear function with a composition of simpler functions is not unique to the MLP. For example, the Fourier series represents arbitrary periodic functions with sines and cosines.

A.2.1 Forward propagation

A new data sample is classified with a forward propagation through the MLP. This subsection describes the operation of this forward propagation. Figure A.3 shows an example of a two-

[1]Using linear activation functions in the output layer does not restrict the class of functions that can be approximated by the MLP. Also, sigmoidal activation functions limit the range of possible outputs to the range attainable by the sigmoid. [17]

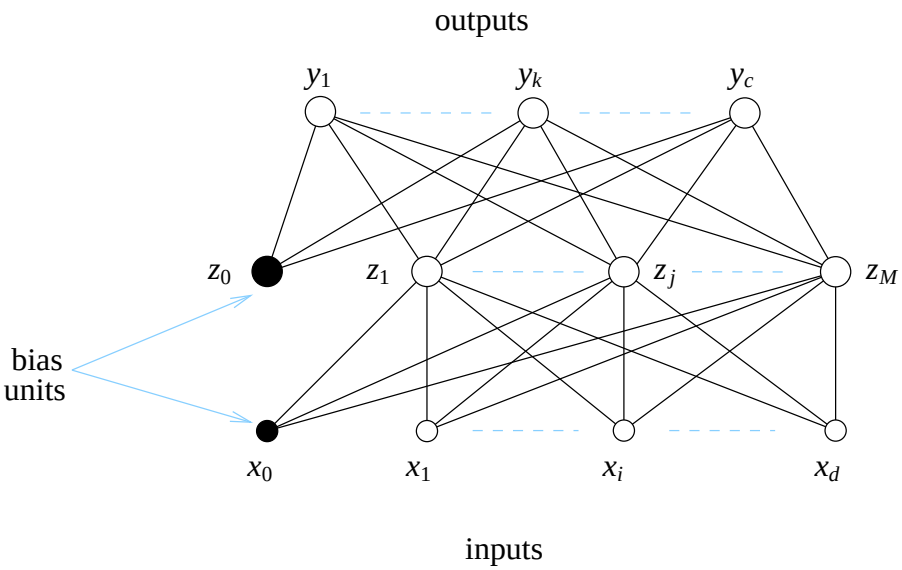


Figure A.3: A multi-layer perceptron with d inputs, M hidden units, and c outputs.

layer[2] MLP with d inputs, M hidden layer neurons, and c outputs.

The output of the jth hidden neuron is obtained by a weighted linear combination of the d inputs

$$a_j = \sum_{i=0}^{d} w_{ji} x_i \tag{A.3}$$

Here w_{ji} is one of the weights in the first layer, connecting input i to hidden neuron j. w_{j0} denotes the bias for hidden neuron j with x_0 permanently set at 1.

The activation of hidden neuron j is then obtained by transforming the linear sum in (A.3) with the sigmoidal activation function in (A.1)

$$z_j = g(a_j) \tag{A.4}$$

The outputs of the MLP are obtained similarly. For each output neuron k, the activation is given by

$$a_k = \sum_{j=0}^{M} w_{kj} z_j \tag{A.5}$$

$$y_k = \tilde{g}(a_k) \tag{A.6}$$

w_{kj} is the weight connecting hidden neuron j and output neuron k. z_j is the output of hidden neuron j. $\tilde{g}(a)$ is the linear activation function defined in (A.2).

[2] It has two layers of adaptive weights.

By combining (A.3), (A.4), (A.5) and (A.6), the result of forward propagation of input sample **x** through an MLP can be summarised as

$$y_k = \tilde{g}\left(\sum_{j=0}^{M} w_{kj} g\left(\sum_{i=0}^{d} w_{ji} x_i\right)\right) \tag{A.7}$$

A.2.2 Error back-propagation

MLP is trained using an algorithm known as error back-propagation. It involves propagating errors in the output layer backwards to adjust the weights in the network.

Initially, the weights in the network are set to some random values. For each iteration t, an input vector $\mathbf{x}(t)$ from the training set is applied to the network to find the activations in the hidden and output layers using (A.4) and (A.6). If $y_k(t)$ is the output of neuron k in the output layer, and $t_k(t)$ is the target (or desired) output of the same neuron, then the *local error gradient* of output neuron k is defined as

$$\delta_k(t) = y_k(t) - t_k(t) \tag{A.8}$$

The error gradient for the hidden layer neuron j is found using

$$\delta_j(t) = z_j(t)(1 - z_j(t)) \sum_{k=1}^{c} w_{kj}(t)\delta_k(t) \tag{A.9}$$

It can be seen that the error gradients from the output layer $\delta_k(t)$ are propagated backwards to the hidden layer.

Using the error gradients $\delta_k(t)$ and $\delta_j(t)$, the weights in the output and hidden layers are updated with

$$\Delta w_{kj}(t) = -\eta\delta_k(t)z_j(t) + \alpha\Delta w_{kj}(t-1) \tag{A.10}$$

$$\Delta w_{ji}(t) = -\eta\delta_j(t)x_i(t) + \alpha\Delta w_{ji}(t-1) \tag{A.11}$$

Here η is the learning rate, and α the momentum term. The update equations specify that the updates to weights w_{kj} and w_{ji} with the current input sample $\mathbf{x}(t)$ is dependent on the update from the previous sample.

Training continues until the stopping criterion is met, with repeated forward presentation of input samples (A.7), and backward propagation of error gradients (A.10) and (A.11).

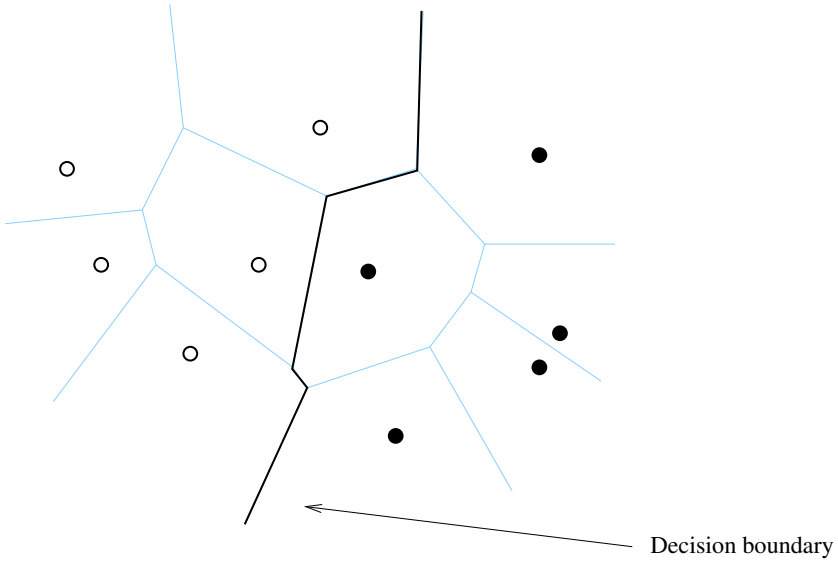


Figure A.4: Classification using LVQ.

A.3 Learning Vector Quantisation (LVQ)

A.3.1 Vector Quantisation

Classification in LVQ is done via a set of reference vectors. A subset of these reference vectors are placed into each of the classes of the data samples. A sample is considered to be of the same class as its closest reference vector. Figure A.4 shows an example of an LVQ for a set of samples with two classes. The reference vectors are divided into the two classes, labelled ∘ and •. The region that belongs to each reference vector is shown in Figure A.4. This is the same as the Voronoi tessellation, which is based on nearest neighbourhood on a set of vectors. The class region is the union of the Voronoi sets for the reference vectors belonging to the same class. Therefore, the decision boundary of the LVQ is the borders of the Voronoi tessellation that separate Voronoi sets into different classes. As a result, the decision boundary in LVQ is piecewise linear.

A.3.2 Learning the reference vectors

The reference vectors are initially distributed randomly within the classes, with an equal number of vectors in each class. The learning process thus involves using the training set to move these vectors to form a good decision boundary.

Let $x(t)$ be an input sample in the training set, and $m_i(t)$ represent sequential values of the m_i

reference vector. Let c be the index of the nearest m_i to x:

$$\begin{aligned} c &= \arg\min_i \|x - m_i\| \\ \|x - m_c\| &= \min_i \{\|x - m_i\|\} \end{aligned}$$

Then the following equations define the basic learning algorithm LVQ1 [62]:

$$m_c(t+1) = \begin{cases} m_c(t) + \alpha(t)[x(t) - m_c(t)] & \text{if } x \text{ and } m_c \text{ belong to the same class,} \\ m_c(t) - \alpha(t)[x(t) - m_c(t)] & \text{if } x \text{ and } m_c \text{ belong to different classes.} \end{cases} \tag{A.12}$$

$$m_i(t+1) = m_i(t) \quad \text{for } i \neq c \tag{A.13}$$

α_t is the learning rate and its value is limited to between 0 and 1. Also, the learning rate decreases monotonically with time during learning.

The update equations basically moves the nearest reference vector for a data sample either closer to or further away from the sample, depending on whether the sample and the nearest reference vector belong to the same class. All other reference vectors are unchanged.

If individual learning rates $\alpha_i(t)$ are assigned to each of the reference vector m_i, then the update equation (A.12) become:

$$m_c(t+1) = \begin{cases} m_c(t) + \alpha_c(t)[x(t) - m_c(t)] & \text{if } x \text{ and } m_c \text{ belong to the same class,} \\ m_c(t) - \alpha_c(t)[x(t) - m_c(t)] & \text{if } x \text{ and } m_c \text{ belong to different classes.} \end{cases} \tag{A.14}$$

This is known as the optimised learning rate LVQ1, or OLVQ1. It is the learning method used in the experiments in this thesis. The number of time steps needed to learn the LVQ is generally 30 to 50 times the number of reference vectors [62].

A.4 Landmark recognition

The Maxifander robot was used for the training and testing of the two neural networks in the landmark recognition task. A picture of the mobile robot is shown in Figure A.5. It has a single ultrasonic transducer on the top. A stepper motor is used to rotate the transducer around to

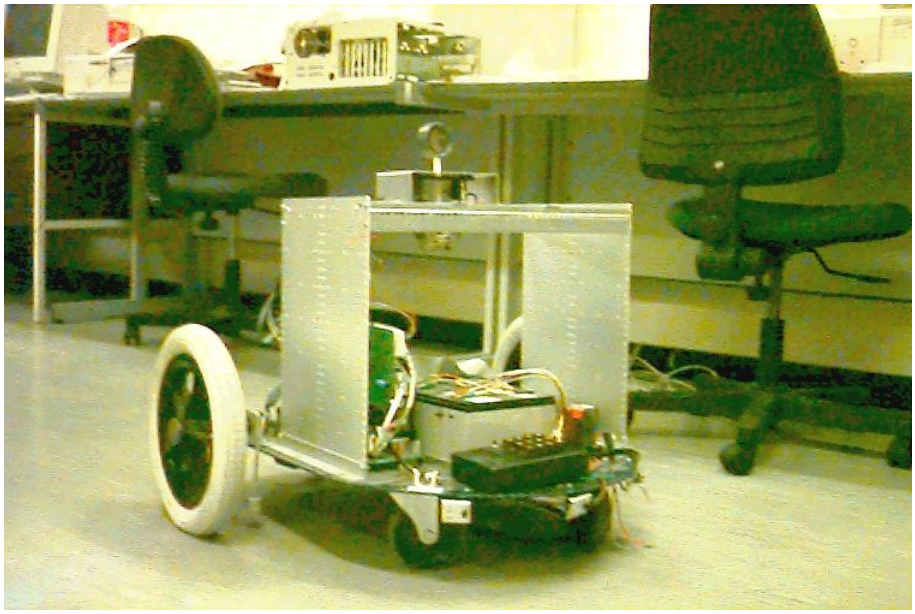


Figure A.5: Maxifander in a university laboratory.

detect obstacles from all directions of the mobile robot. A vector of 48 readings is returned from a single 360-degree scan.

Sonar data collected were categorised into three groups — free space nodes, obstacle nodes and everything else. The tasks of the two neural networks were thus to learn this classification and to predict which group a new sonar data sample belonged to.

A.4.1 Preprocessing

Both free space and obstacle nodes are local features. To reduce the influence of far away objects on the recognition process, the measured sonar range data was cut if it was over a certain threshold.

To make the classification independent of the orientation of the robot, each vector of 48 range readings was virtually rotated into the orientation most occupied by obstacles [65]. After this virtual rotation, index 0 of the vector would always be pointing towards the direction where the sonar range sensor measured the shortest distances. An example explaining this virtual rotation is shown in Figure A.6. This most occupied orientation was calculated using the following equation:

$$\vec{d}_{\mathrm{MOO}} = \frac{1}{n} \sum_{i=1}^{n} \vec{d}_i \tag{A.15}$$

where $n = 48$ is the number of readings in each vector, $\vec{d}_i$ is a vector originating from the centre of the robot denoting sonar sensor reading for direction i, and $\vec{d}_{\mathrm{MOO}}$ is the vector for the most

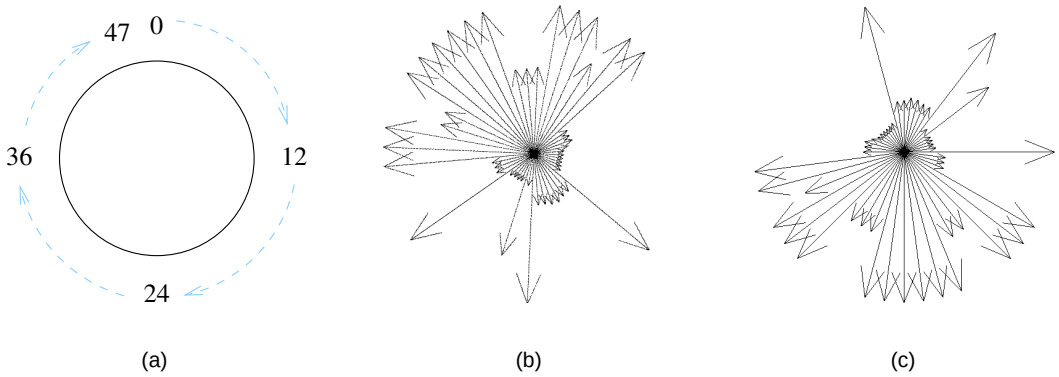


Figure A.6: Rotation of sonar reading to most occupied direction. (a) Index to the sonar data vector, (b) Original data, (c) Rotated most occupied direction to index 0.

occupied orientation.

Using (A.15), all 48 points in the vector were used to calculate the most occupied orientation. This made the process of finding the most occupied orientation more robust to noise than if only the shortest range in the vector was used.

A.4.2 Results

Multi-layer perceptron (MLP)

The training and testing of MLPs was done using the free Stuttgart Neural Network Simulator (SNNS) [3]. MLPs with various configurations were trained multiple times on this recognition problem using the training set to find the network that achieved the lowest mean square error on the test set[3]. The parameters that were varied in the networks were number of hidden layer neurons, learning rate, momentum term and initial weight values.

The lowest mean square error achieved was 0.0955 with 8 hidden neurons, learning rate $\eta = 0.4$ and momentum term $\alpha = 0.25$. Classification accuracy achieved in the test set is shown in Table A.1, where accuracy is defined as the number of accurate predictions divided by the total number of samples in the test set. The result of classification using MLP on a test environment is shown in Figure A.7(a).

[3] A test set contains input samples for testing the classification accuracy of neural networks. The samples in the test set are not in the training set.

	MLP	LVQ
concave	90%	90%
convex	60%	100%
others	95%	88%

Table A.1: Accuracy achieved on the test set for MLP and LVQ.

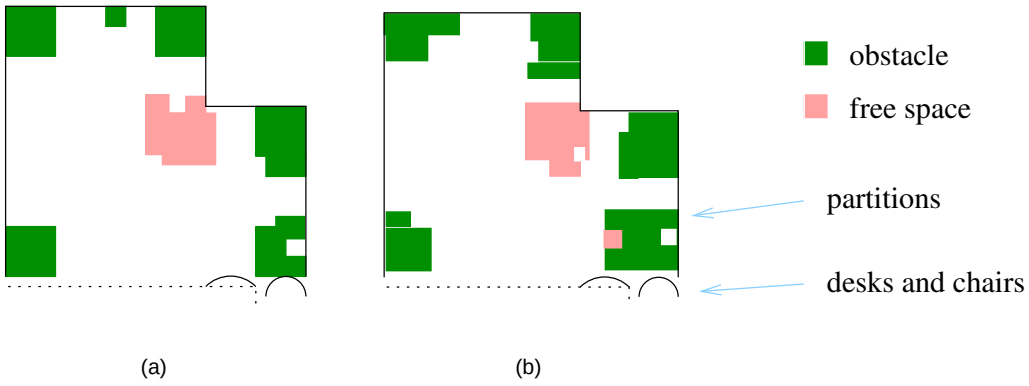


Figure A.7: Classification with (a) MLP (b) LVQ.

Learning vector quantisation (LVQ)

LVQs for testing the landmark recognition problem were implemented using the free LVQ_PAK [5]. Networks with different numbers of neurons, or reference vectors, distributed among the three classes were trained for 40 epochs[4]. A network with 30 neurons yielded good results. The accuracy achieved is shown in Table A.1. The result of classification using LVQ on the test environment is shown in Figure A.7(b).

A.5 Summary

Supervised neural networks were chosen for this task because the landmark types to be recognised were predefined. The neural networks were trained to generate rules statistically to classify sonar range data into three pre-defined classes. A different use of neural networks for topological maps is to let unsupervised neural networks partition environments into separate regions according to similarity of input sensory data [65, 111].

It can be seen from Table A.1 that the two neural networks give different accuracy rates for the three categories to be classified. Despite the difference in accuracies, the resultant classification is quite similar, as in Figure A.7. This is due to the fact that misclassification occurs mostly at the boundaries between different zones. Misclassification at boundaries is insignificant because

[4] An epoch is a complete presentation of the training set.

it does not affect the implementation of the topological coverage algorithm. This shows that accuracy alone is not a good indication on how well a neural network performs for a target application. Overall, both neural networks are capable of recognising free space and obstacle nodes in the environment.

One of the problems of taking things apart and seeing how they work – supposing you're trying to find out how a cat works – you take that cat apart to see how it works, what you've got in your hands is a non-working cat. The cat wasn't a sort of clunky mechanism that was susceptible to our available tools of analysis.

Douglas Adams, "Hitchhiker's Guide to the Galaxy"

B

Computer Vision

This chapter describes two of the computer vision techniques used for creating composition images in Chapter 5 in greater detail. Section B.1 explains the Canny edge detector, which is perhaps the most popular edge detection technique at present [79]. This is followed by a description of the Hough transform in Section B.2, which is used for locating and extracting shapes in images.

B.1 Canny Edge Detection

Edge detection is an important operation in computer vision applications. Edges are employed as primitive features which guide applications to make more complex suppositions about the visual information. Edges are also important in preprocessing an image to yield regions of interest.

Figure B.1 shows an image of a light box on a dark background. In this image, the edges are clearly defined as the external boundary of the box. Figure B.2 shows the intensity profile of the image across the row of pixels where the y-coordinate is 50. Note that there is a large change in intensity at the point where the box starts (at $x = 20$). This trend is reversed where the box ends (at $x = 70$). In summary, the change in intensity is directly related to the edges in the image.

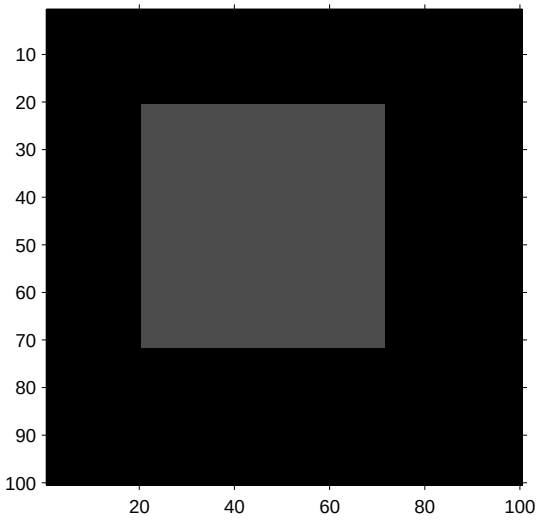


Figure B.1: An image with strong edges.

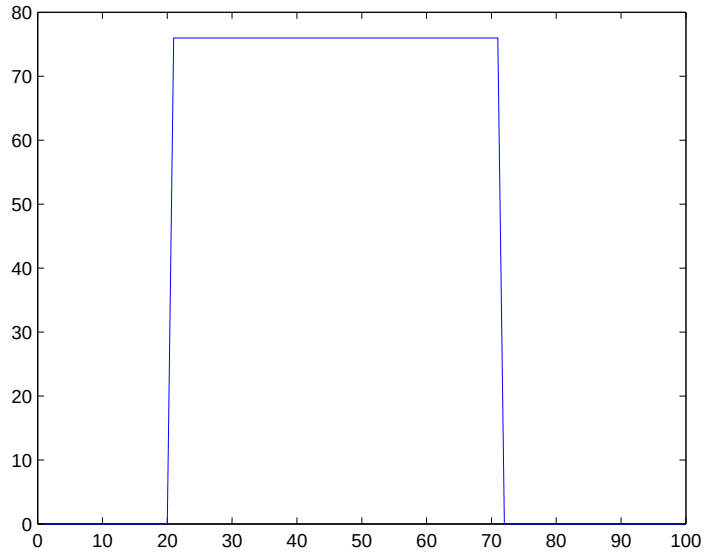


Figure B.2: The intensity profile of the image across y=50.

In computer vision terms, an edge is characterised by significant step changes in intensity. Detection of edges within images are based on mathematical methods which can be applied to detect such rapid changes in slope. For instance, the first derivative of the intensity function should have a maxima at the midpoint of the left edge and a minima at the midpoint of the right edge (Figure B.3(a)). The second derivative is zero at the beginning and end of each edge (Figure B.3(b)). The first and second derivatives are the basis for the design of many edge detection techniques [48, 70].

The Canny edge detector was designed to be an optimal edge detector. It has the following properties:

- *Optimal detection* By smoothing the image, spurious responses are reduced, and the edge map is made less noisy.
- *Good localisation* To improve accuracy in detection, edges must be found in the correct location.
- *Single response* A single response is found for each valid edge in an image. For instance, a edge detector based on the second derivative will have two responses for every edge in the image. This was considered non-optimal by Canny.

These three goals are achieved using the following steps:

1. Gaussian smoothing
2. Sobel edge detection
3. Non-maximal suppression
4. Hysteresis thresholding

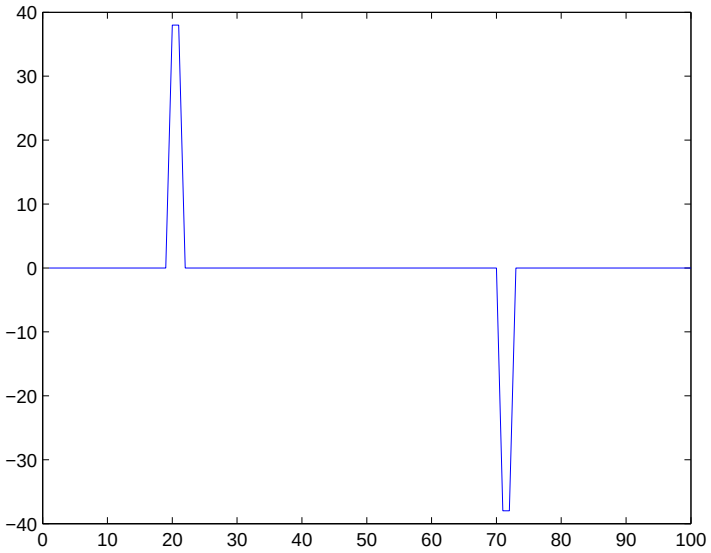
These steps will now be discussed in turn. The image shown in Figure B.4 will be used to aid in this discussion.

B.1.1 Gaussian smoothing

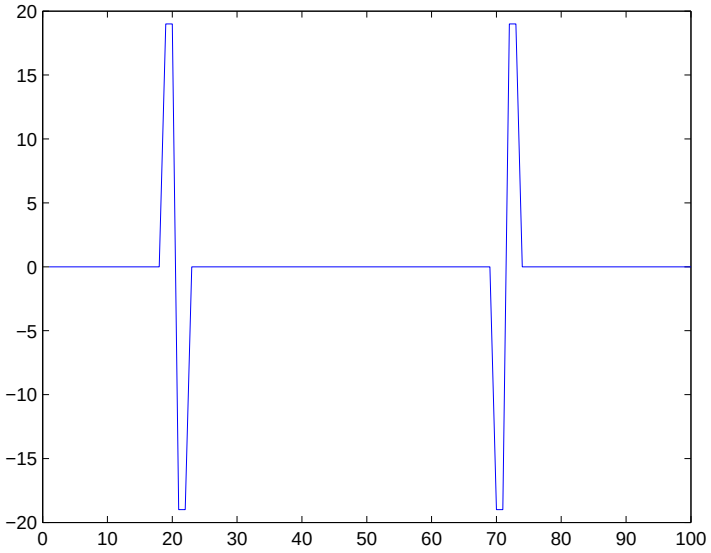
Canny demonstrated in [25] that Gaussian smoothing was the optimal method for image smoothing. Application of Gaussian smoothing requires convolution of the image $I(x, y)$ with a suitable Gaussian mask $g(x, y)$:

$$I * g = \sum_{j,k} I(j, k)g(x - j, y - k)$$

The coefficients of the Gaussian mask can be defined by the Gaussian function:



(a)



(b)

Figure B.3: (a) First derivative, and (b) second derivative of the edge profile in Figure B.2.

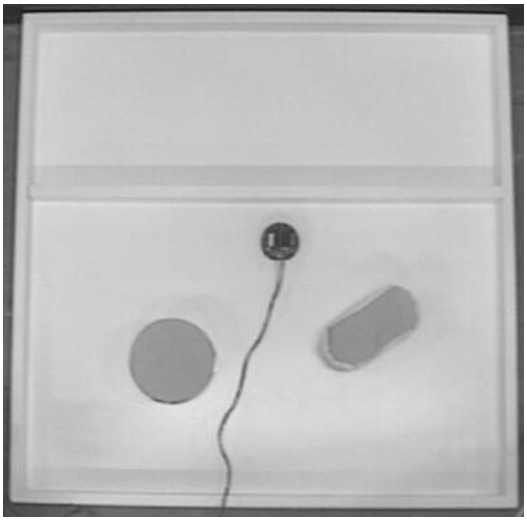
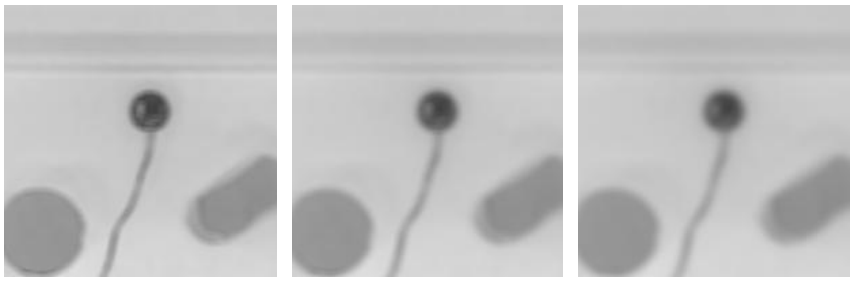


Figure B.4: An image of the Khepera robot moving in its environment.



(a) (b) (c)

Figure B.5: Gaussian blurring of an image (a) 3×3 (b) 5×5 (c) 7×7

$$g(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{B.1}$$

The larger the mask that is employed, the more accurately it fits the ideal Gaussian. However, this accuracy comes at a cost of increased computation. Generally, for real applications, a mask of size 5×5 or 7×7 is employed. The σ term in (B.1) is chosen to make sure the coefficients drop to 0 near the edge of the mask. Figure B.5 shows increasing Gaussian blur on a sub region of the image in Figure B.4.

B.1.2 Sobel edge detection

The Sobel edge operator is a widely used edge detection operator. It is an example of a gradient/derivative based edge detector. It approximates the gradient operation by a pair of 3×3 masks:

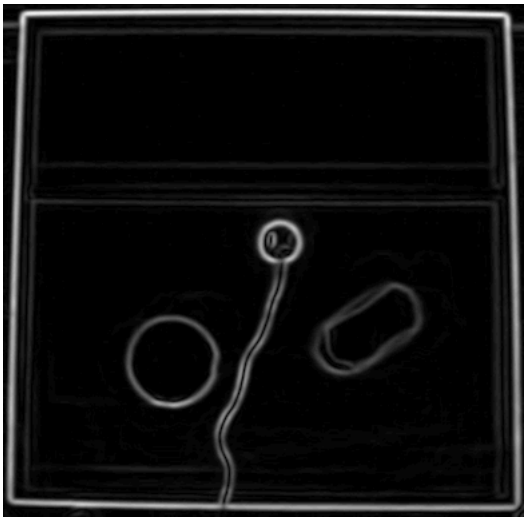


Figure B.6: The result of edge detection after Gaussian blurring.

$$M_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The masks are aligned in the horizontal (M_x) and vertical directions (M_y). Operation of the Sobel edge detector involves computing a pair of gradient images, S_x and S_y, by first convolving the image I with each of the masks:

$$S_x = I * M_x, \qquad S_y = I * M_y$$

Each point in the two gradient images, S_x and S_y are then combined using sum of squares[1] to generate a candidate edge map M for the original image I:

$$M = \sqrt{S_x^2 + S_y^2}$$

A candidate edge map is shown in Figure B.6. Brighter intensities in this image relate to stronger edges.

B.1.3 Non-maximal suppression

The next stage in the processing of the edge data is to use non-maximal suppression. Non-maximal suppression serves to find the highest points in the edge information and follow the

[1] Alternately, the sum of absolute values can be used.

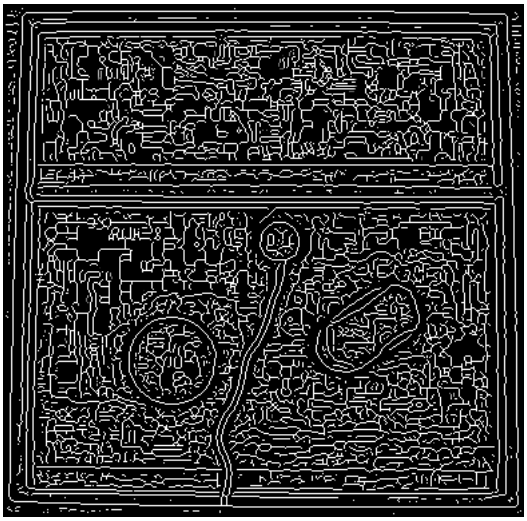


Figure B.7: The result of applying non-maximal suppression.

contours that they belong to. This process is performed via the use of the edge gradient images, S_x and S_y. For a 3×3 region, the gradient is maximal if the gradient on either side of it is less than the gradient at the centre point. Thus, in order to find this value the value of the gradient at points normal to the actual gradient are required. As the image is defined upon an integer grid, interpolation is required for this. Thus:

$$G_1 = \frac{S_y}{S_x} M(x+1, y-1) + \frac{S_x - S_y}{S_x} M(x, y-1)$$
$$G_2 = \frac{S_y}{S_x} M(x-1, y+1) + \frac{S_x - S_y}{S_x} M(x, y+1)$$

If the value of the gradient at the point $M(x, y)$ exceeds both G_1 and G_2, then the point is marked as a peak; otherwise it is set to 0. The result of non-maximal suppression applied to the Sobel edge image is shown in Figure B.7.

B.1.4 Hysteresis thresholding

The final step in the Canny edge detector is the use of hysteresis thresholding. This uses a hysteresis function to threshold the image. Figure B.8 shows an example of hysteresis thresholding. In the example, the underlying function (as given by the smooth curve), is hysteretically thresholded. A maximum value is kept until the value of the curve drops below a minimum value (lower threshold). This value is kept until the value of the data exceeds the upper threshold. In this way, values which exceed the upper threshold are considered to be definite edge points; while values below the lower threshold are definitely not edge points. Points which lie

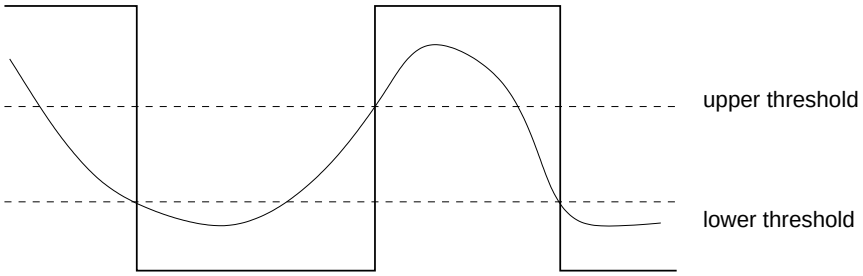


Figure B.8: Hysteresis thresholding. The smooth curve is the underlying function. The square curve is the output of hysteresis thresholding.

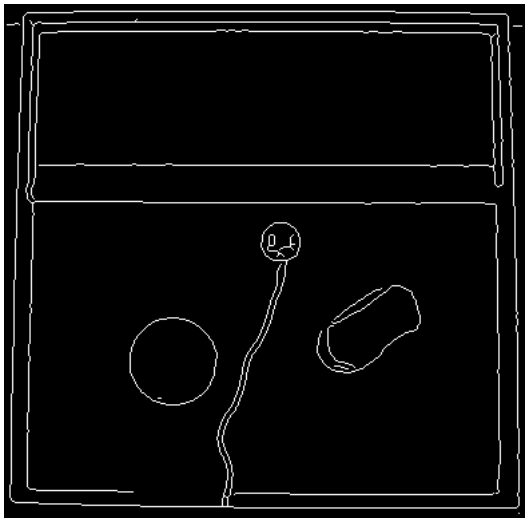


Figure B.9: Hysteresis thresholding applied to the non-maximally suppressed image.

in between are indeterminate and generally depend on their neighbours. In implementation, any edge point can be used as the starting seed. The neighbours of the seed are then searched to see if they exceed the lower threshold. If they do, then they are also labelled as an edge point and become a new seed. The process terminates when there are no neighbours above the lower threshold and no seeds remaining. Figure B.9 shows the result of applying hysteresis thresholding to the non-maximally suppressed image in Figure B.7.

The image in Figure B.9 is the final result of the Canny edge detector. The Canny edge detector is generally a very clean looking edge map with little noise. Furthermore, it preserves fine detail in the image. These features make the edge map very useful for subsequent processing.

B.2 Hough Transform

The Hough transform is a computer vision technique for finding shapes in images. Generally it is used to find lines, circles, and ellipses, though it has been extended to more arbitrary shapes

[79]. It owes its popularity to the fact that it achieves the same result as template matching[2] without the computation overhead. The reduction in computational overhead is due to the way in which it reformulates template matching as an evidence gathering technique where votes are cast in an accumulator array. To achieve this, it uses a mapping from the image space to the accumulator. The mapping is computationally efficient as it is based upon a description of the shape that is being searched for.

In this thesis, a circular object was required to be identified. This was the Khepera robot. For this reason, the Hough transform was employed as a circle finder. The general equation for a circle is:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

This defines a circle centred on point (x_0, y_0) with a radius r. This definition implies that the circle is a locus of points, centred on point (x_0, y_0), with a radius r. However, there is an alternate view in that it is also the locus of circles, centred at (x, y) with a radius r. The two cases are illustrated in Figure B.10. Geometrically, the two cases are equivalent. For the Hough transform the latter of the two views is used for the accumulator space. The reason it is employed is as follows. For any point that is on the circle in Figure B.10(a), a unique circle can be drawn in the accumulator space as shown in Figure B.10(b). Other concyclic points also result in another circle in the accumulator space. However, the key point to note is that in the accumulator space, all these circles pass through one point in common. This is the point which corresponds to the centre point of the original circle. In employing the Hough transform, a count is kept of the number of times a circle is drawn though this centre point for each candidate edge point in the original image. As only valid edges are used in this process, the computational requirements of the matching process is significantly reduced compared to template matching.

An example of this notion applied to the Canny edge image from Figure B.9 is shown for several different radii in Figure B.11. Figure B.11(a) shows the accumulator space where the radii is too small for the feature of interest. There is no main peak though the highest peak lies in the robots general vicinity. This is probably due to the fine details on the top surface of the robot. In Figure B.11(b), the radii is set to the same as the Khepera. Notice the distinct peak found centred on the robots' centre. In Figure B.11(c), the radii is set to be equal to that of the round obstacle. This gives a nice peak centred on the centre of the obstacle. Generally, the exact radius of the object to be found is not known and a range of candidate radii are employed. The peak finding process in this case requires a three dimensional search in (x, y, r).

[2] In template matching, a template is centred on an image point, and the number of points in the image that match the template is counted. The procedure is repeated for the entire image. The point which led to the best match, the maximum count, is deemed to be the point where the shape (given by the template) lies within the image [79].

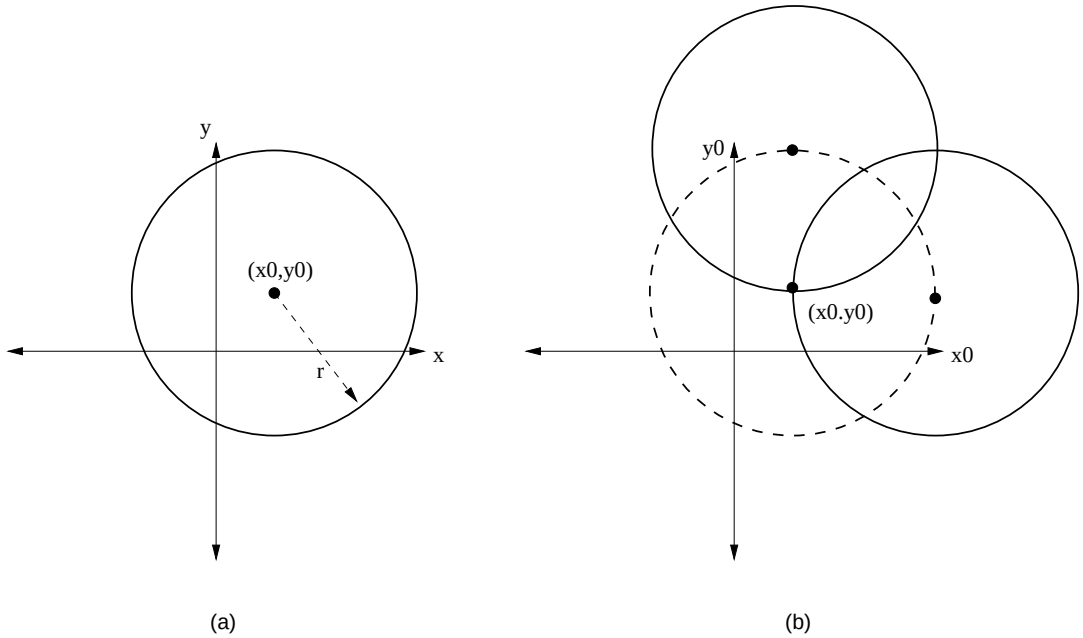


Figure B.10: Definition of a circle (a) a locus of points (b) a locus of circles.

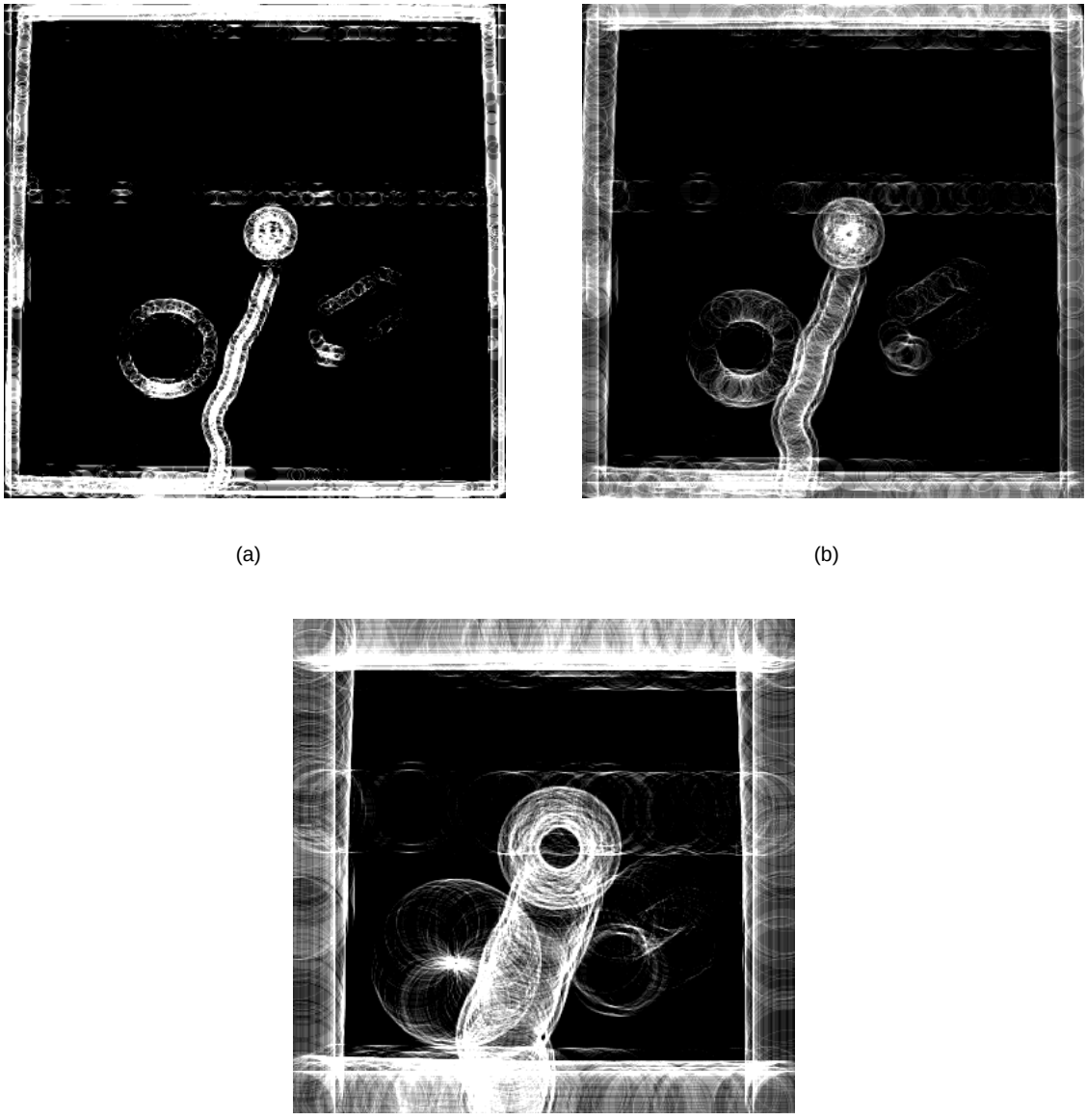
It is possible to use a parametric form of the circle as follows:

$$x = x_0 + r\cos\theta, \qquad y = y_0 + r\sin\theta$$

This is useful as it directly allows the parameters to be solved for :

$$x_0 = x - r\cos\theta, \qquad y_0 = y - r\sin\theta \tag{B.2}$$

These equations directly define the points in the accumulator space (as seen in Figure B.10). In (B.2), θ is not a free parameter but defines the trace, or point spread function, of the curve. These notions put together allow us to define the algorithm for the Hough transform (Algorithm B.1). Figure B.12 highlights the position and size of the robot in Figure B.9 as found by the Hough transform.



(a) (b)

(c)

Figure B.11: Scaled accumulator space for Figure B.9 with radii of (a) 5 pixels (b) 12 pixels (c) 32 pixels.

Algorithm B.1 Hough transform

$I \leftarrow m \times n$ image
$a \leftarrow m \times n$ array of zeros
$E \leftarrow$ set of edge points in I
for $(x, y) \in E$ **do**
  for $r \in [r_{min}, r_{max}]$ **do**
    for $\theta \in [0, 2\pi)$ **do**
      $x_0 \leftarrow x - r * \cos\theta$
      $y_0 \leftarrow y - r * \sin\theta$
      if (x_0, y_0) in image **then**
        $a(x_0, y_0) \leftarrow a(x_0, y_0) + 1$
      end if
    end for
  end for
end for

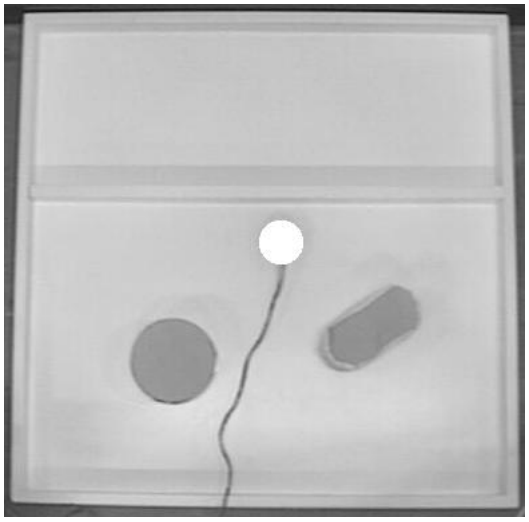


Figure B.12: The drawn circle on this image shows the position and size (x, y, r) of the robot as extracted by the Hough transform.

Bibliography

[1] Press release for World Robotics 2003. http://www.unece.org/press/pr2003/03stat_p01e.pdf.

[2] Roomba, robotic floorvac. http://www.roombavac.com.

[3] Stuttgart neural network simulator (SNNS). http://www-ra.informatik.uni-tuebingen.de/SNNS/.

[4] Boost C++ libraries. http://www.boost.org.

[5] LVQ_PAK for the learning vector quantization algorithms. http://www.cis.hut.fi/research/som-research/nnrc-programs.shtml.

[6] E. U. Acar, H. Choset, and P. N. Atkar. Complete sensor-based coverage with extended-range detectors: a hierarchical decomposition in terms of critical points and voronoi diagrams. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1305–1311, 2001.

[7] Ercan U. Acar and Howie Choset. Critical point sensing in unknown environments. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3803–3810, April 2000.

[8] Ercan U. Acar and Howie Choset. Sensor-based coverage of unknown environments: Incremental construction of morse decompositions. *International Journal of Robotics Research*, 21(4):345–366, April 2002.

[9] Ercan U. Acar, Howie Choset, Alfred A. Rizzi, Prasad N. Atkar, and Douglas Hull. Morse decompositions for coverage tasks. *International Journal of Robotics Research*, 21(4):331–344, April 2002.

[10] Ercan U. Acar, Howie Choset, Alfred A. Rizzi, and Jonathan Luntz. Exact cellular decompositions in terms of critical points of morse functions. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 2270–2277, April 2000.

[11] Ercan U. Acar, Howie Choset, Yangang Zhang, and Mark Schervish. Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods. *International Journal of Robotics Research*, 22(7–8):441–466, 2003.

[12] Ercan Umut Acar. *Complete Sensor-based Coverage of Unknown Spaces: Incremental Construction of Cellular Decompositions*. PhD thesis, Carnegie Mellon University, Pennsylvania, 2002.

[13] Esther M. Arkin, Sandor P. Fekete, and Joseph S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1-2):25–50, 2000.

[14] Ronald C. Arkin. *Behavior-based robotics*. MIT Press, 1998.

[15] Prasad N. Atkar, Howie Choset, and Alfred A. Rizzi. Towards optimal coverage of 2-dimensional surfaces embedded r3: Choice of start curve. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 4, pages 3581–3587, 2003.

[16] O. Burchan Bayazit, Jyh-Ming Lien, and Nancy M. Amato. Better flocking behaviors in complex environments using global roadmaps. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, December 2002.

[17] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[18] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[19] Rodney A. Brooks. A robot that walks; emergent behavior from a carefully evolved network. *Neural Computation*, 1(2):253–262, 1989.

[20] Rodney A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[21] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[22] Zack J. Butler. *Distributed Coverage of Rectilinear Environments*. PhD thesis, Carnegie Mellon University, Pennsylvania, 2000.

[23] Zack J. Butler, Alfred A. Rizzi, and Ralph L. Hollis. Contact sensor-based coverage of rectilinear environments. In *Proceedings IEEE International Symposium on Intelligent Control/Intelligent Systems and Semiotics*, pages 266–271, 1999.

[24] Lola D. Cañamero. Designing emotions for activity selection in autonomous agents. In Robert Trappl, Paolo Petta, and Sabine Payr, editors, *Emotions in Humans and Artifacts*, pages 115–148. MIT Press, 2002.

[25] J. Canny. A computational approach to edge detection. *PAMI*, 8(6):679–698, November 1986.

[26] John Canny. Constructing roadmaps of semi-algebraic sets i: Completeness. *Artificial Intelligence*, 37(1–3):203–222, 1988.

[27] John F. Canny and Ming C. Lin. An opportunistic global path planner. *Algorithmica*, 10:102–120, 1993.

[28] J. A. Castellanos, J. D. Tardós, and G. Schmidt. Building a global map of the environment of a mobile robot: The importance of correlations. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1053–1059, 1997.

[29] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K. Yap, editors, *Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, 1987.

[30] Howie Choset. Coverage of known spaces: The boustrophedon cellupdar decomposition. *Autonomous Robots*, 9(3):247–253, December 2000.

[31] Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17(2):125–137, 2001.

[32] Howie Choset and Philippe Pignon. Coverage path planning: The boustrophedon decomposition. In *Proceedings of the International Conference on Field and Service Robotics*, Canberra, Australia, 1997.

[33] Frank H. Clarke. Generalized gradients and applications. *Transactions of the American Mathematical Society*, 205:247–262, 1975.

[34] M. Bernardine Dias and Anthony Stentz. A comparative study between centralized, market-based, and behavioral multirobot coordination approaches. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2279–2284, October 2003.

[35] Tom Duckett and Ulrich Nehmzow. Mobile robot self-localisation and measurement of performance in middle-scale environments. *Robotics and Autonomous Systems*, 24:57–69, 1998.

[36] O. Duran, K. Althoefer, and L.D. Seneviratne. Pipe inspection using a laser-based transducer and automated analysis techniques. *IEEE/ASME Transactions on Mechatronics*, 8(3):401–409, September 2003.

[37] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

[38] N. Elkmann, T. Felsch, M. Sack, J. Saenz, and J. Hortig. Innovative service robot systems for facade cleaning of difficult-to-access areas. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 756–762, 2002.

[39] H. R. Everett. *Sensors for Mobile Robots: Theory and Application*. A K Peters, 1995.

[40] James D. Foley et al. *Computer graphics : principles and practice*. Addison-Wesley, second edition, 1990.

[41] United Nation Economic Commission for Europe. *World Robotics 2003 (Survey Report)*. ISBN No. 92-1-101059-4, 2003.

[42] J. J. Fox and A. A. Maciejewski. Utilizing the topology of configuration space in real-time multiple manipulator path planning. *International Journal of Robotics and Automation*, 16(1):1–13, 2001.

[43] Jakob Fredslund and Maja J. Matarić. A general, local algorithm for robot formations. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, 2002.

[44] Yoav Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31:77–98, 2001.

[45] Yoav Gabriely and Elon Rimon. Spiral-STC: An on-line coverage algorithm of grid environments by a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 954–960, Washington, DC, May 2002.

[46] S. S. Ge and Y. J. Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, 13(3):207–222, 2002.

[47] Brian P. Gerkey and Maja J. Matarić. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, 2002.

[48] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, second edition, 2002.

[49] Eric Haines. Point in polygon strategies. In *Graphics Gems IV*, pages 24–46. Academic Press, Boston, 1994.

[50] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, second edition, 1999.

[51] Martin Held. *On the Computational Geometry of Pocket Machining*. Springer-Verlag, 1991.

[52] Susan Hert, Sanjay Tiwari, and Vladimir Lumelsky. A terrain-covering algorithm for an auv. *Autonomous Robots*, 3:91–119, 1996.

[53] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[54] Arjang Hourtash and Mahmoud Tarokh. Manipulator path planning by decomposition: algorithm and analysis. *IEEE Transactions on Robotics and Automation*, 17(6):842–856, 2001.

[55] Wesley H. Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 27–32, 2001.

[56] Christian Icking, Thomas Kamphans, Rolf Klein, and Elmar Langetepe. On the competitive complexity of navigation tasks. In *Sensor Based Intelligent Robots, Lecture Notes in Computer Science*, pages 245–258, 2000.

[57] Markus Jäger and Bernhard Nebel. Dynamic decentralized area partitioning for cooperating cleaning robots. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 3577–3582, May 2002.

[58] Ray Jarvis. Distance transform based path planning for robot navigation. In Yuan F. Zheng, editor, *Recent Trends in Mobile Robots*, pages 3–31. World Scientific Publishing Co., 1993.

[59] Margaret E. Jefferies, Wenrong Weng, Jesse T. Baker, and Michael Mayo. A hybrid approach to finding cycles in hybrid maps. In *Processings Australasian Conference on Robotics and Automation (ACRA)*, Brisbane, Australia, 2003.

[60] Margaret E. Jefferies, Wenrong Weng, Jesse T. Baker, and Michael Mayo. Using context to solve the correspondence problem in simultaneous localisation and mapping. In *Proceedings PRICAI 2004: Trends in Artificial Intelligence: 8th Pacific Rim International Conference on Artificial Intelligence*, volume 3157/2004 of *Lecture Notes in Computer Science*, pages 664–672, Auckland, New Zealand, August 2004. Springer-Verlag Heidelberg.

[61] Joseph L. Jones, Bruce A. Seiger, and Anita M. Flynn. *Mobile Robots: Inspiration to Implementation*. A. K. Peters, second edition, 1999.

[62] Teuvo Kohonen. *Self-Organizing Maps*. Springer, second edition, 1997.

[63] Erwin Kreyszig. *Advanced Engineering Mathematics*. John Wiley and Sons, sixth edition, 1988.

[64] Benjamin J. Kuipers and Yung-Tai Byun. A robust, qualitative method for robot spatial learning. In *Proceedings Seventh National Conference on Artificial Intelligence AAAI-88*, pages 774–779, St Paul, Minnesota, August 1988.

[65] Andreas Kurz. Constructing maps for mobile robot navigation based on ultrasonic range data. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, 26(2):233–242, April 1996.

[66] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer, 1991.

[67] Philip D. Loewen. *Optimal Control via Nonsmooth Analysis*. American Mathematical Society, 1993.

[68] Amol Dattatraya Mali. On the behavior-based architectures of autonomous agency. *IEEE Transactions on Systems, Man, and Cybernetics: Part C - Applications & Reviews*, 32(3):231–242, 2002.

[69] L. Marques, M. Rachkov, and A. T. de Almeida. Mobile pneumatic robot for demining. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3508–3513, 2002.

[70] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W.H. Freeman and Company, 1982.

[71] Mohamed Marzouqui and Ray Jarvis. Covert path planning for autonomous robot navigation in known environments. In *Proceedings Australasian Conference on Robotics and Automation (ACRA)*, Brisbane, Australia, 2003.

[72] Maja J. Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, June 1992.

[73] Kurt Mehlhorn and Stefan Nähe. *LEDA : A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[74] Lee Middleton and Jayanthi Sivaswamy. A framework for practical hexagonal-image processing. *Journal of Electronic Imaging*, 11:104–114, 2002.

[75] F. Mondada, E. Franzi, and P. Ienne. Mobile robot miniaturisation: A tool for investigation in control algorithms. In T. Yoshikawa and F. Miyazaki, editors, *Proceedings of the Third International Symposium on Experimental Robotics*, pages 501–513, 1993.

[76] Luis Moreno and Eladio Dapena. Path quality measures for sensor-based motion planning. *Robotics and Autonomous Systems*, 44:131–150, 2003.

[77] Ulrich Nehmzow. Animal and robot navigation. *Robotics and Autonomous Systems*, 15:71–81, 1995.

[78] Ulrich Nehmzow. Quantitative analysis of robot-environment interaction - on the difference between simulations and the real thing. In *Proceedings Eurobot*, 2001.

[79] Mark S. Nixon and Alberto S. Aguado. *Feature Extraction & Image Processing*. Butterworth-Heinemann, 2002.

[80] Joon Seop Oh, Yoon Ho Choi, Bae Jin Park, and Yuan F. Zheng. Navigation of cleaning robots using triangular-cell map for complete coverage. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 2006–2011, 2003.

[81] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, 1992.

[82] J. O'Keefe and J. Dostrovsky. The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely moving rat. *Brain Research*, 34:171–175, 1971.

[83] Chris A. C. Parker, Hong Zhang, and C. Ronald Kube. Blind bulldozing: Multiple robot nest construction. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 2010–2015, Las Vegas, Nevada, 2003.

[84] P. Ranganathan, Hayet J. B., M. Devy, S. Hutchinson, and F. Lerasle. Topological navigation and qualitative localization for indoor environment using multi-sensory perception. *Robotics and Autonomous Systems*, 41:137–144, 2002.

[85] Elisha Sacks. Path planning for planar articulated robots using configuration spaces and compliant motion. *IEEE Transactions on Robotics and Automation*, 19:381–390, 2003.

[86] S. Sakamoto. Mechanical planning and actual test results of a robot for painting the exterior walls of high-rise buildings. *Advanced Robotics*, 5(4):457–466, 1991.

[87] C. Santos, N. Monteiro, J. Fonseca, P. Garrido, and C. Couto. Control of a robot painting system using the multi-resolution architectural principle-a summary. In *Proceedings IEEE International Symposium on Industrial Electronics (ISIE)*, volume 2, pages 672–676, 1997.

[88] Raimund Seidel. A simple and fast randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry Theory and Applications*, 1:51–64, 1991.

[89] M. Shimrat. Algorithm 112: Position of point relative to polygon. *Communications of the A.C.M.*, 5(8):434, 1962.

[90] Jeremy G. Siek Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2001.

[91] Thierry Siméon, Stéphane Leroy, and Jean-Paul Lauumond. Path coordination for multiple mobile robots: a resolution-complete algorithm. *IEEE Transactions on Robotics and Automation*, 18:42–49, 2002.

[92] Steven L. Tanimoto. *The Elements of Artificial Intelligence Using Common Lisp*, chapter 5. Computer Science Press, second edition, 1995.

[93] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[94] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99:21–71, 1998.

[95] Sebastian Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[96] Nicola Tomatis, Illah Nourbakhsh, and Roland Siegwart. Hybrid simultaneous localization and map building: a natural integration of topological and metric. *Robotics and Autonomous Systems*, 44:3–14, 2003.

[97] Iwan R. Ulrich, Francesco Mondada, and J. D. Nicoud. Autonomous vacuum cleaner. *Robotics and Autonomous Systems*, 19(3–4):233–245, March 1997.

[98] Steven A. Wilmarth, Nancy M. Amato, and Peter F. Stiller. MAPRM: a probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1024–1031, 1999.

[99] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1990.

[100] Sylvia Wong, George Coghill, and Bruce MacDonald. Landmark-based world model for autonomous vacuuming robots. In *Proceedings International ICSC Congress on Intelligent Systems and Applications (ISA)*, Wollongong, Australia, 2000.

[101] Sylvia Wong, George Coghill, and Bruce A. MacDonald. Natural landmark recognition using neural networks for autonomous vacuuming robots. In *Proceedings of 6th International Conference on Control, Automation, Robotics and Vision*, Singapore, 2000.

[102] Sylvia C. Wong and Bruce A. MacDonald. A topological coverage algorithm for mobile robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 4, pages 1685–1689, Las Vegas, Nevada, 2003.

[103] Sylvia C. Wong and Bruce A. MacDonald. Complete coverage by mobile robots using slice decomposition based on natural landmarks. In *Proceedings PRICAI 2004: Trends in Artificial Intelligence: 8th Pacific Rim International Conference on Artificial Intelligence*, volume 3157/2004 of *Lecture Notes in Computer Science*, pages 683–692, Auckland, New Zealand, August 2004. Springer-Verlag Heidelberg.

[104] Sylvia C. Wong, Lee Middleton, and Bruce A. MacDonald. Performance metrics for robot coverage tasks. In *Proceedings Australasian Conference on Robotics and Automation (ACRA)*, pages 7–12, Auckland, New Zealand, 2002.

[105] Sylvia C. Wong, Lee Middleton, and Bruce A. MacDonald. Creating composite images for estimating the effectiveness of mobile robot coverage algorithms. In *Proceedings Australasian Conference on Robotics and Automation (ACRA)*, Brisbane, Australia, 2003.

[106] David C. K. Yuen and Bruce A. MacDonald. Natural landmark based localisation system using panoramic images. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 915–920, 2002.

[107] David C. K. Yuen and Bruce A. MacDonald. An evaluation of sequential monte carlo technique for simultaneous localisation and map-building. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, pages 1564–1569, Taipei, September 2003.

[108] A. Zelinsky, R. A. Jarvis, J. C. Byrne, and S Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robots. In *International Conference on Advanced Robotics ICAR*, Tokyo, Japan, November 1993.

[109] Alexander Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6):707–717, December 1992.

[110] Zhigang Zhu, Shiqiang Yang, Guangyou Xu, Xueyin Lin, and Dingji Shi. Fast road classification and orientation estimation using omni-view images and neural networks. *IEEE Transactions on Image Processing*, 7(8):1182–1197, August 1998.

[111] Uwe R. Zimmer. Robust world-modelling and navigation in a real world. *Neurocomputing*, 13(2–4):247–260, 1996.

[112] Robert Zlot, Anthony Stentz, M. Bernardine Dias, and Scott Thayer. Multi-robot exploration controlled by a market economy. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 3016–3023, May 2002.