Circle Packing as a Space Decomposition Method for Robot Path Planning and Coverage

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Andrew Kenneth Messing

May 2018

© by Andrew Kenneth Messing, 2018
All Rights Reserved.

Dedication

To my parents

who provided for me through the years and put up with my constant talk about robots

Acknowledgments

I would like to acknowledge and thank my advisor Dr. Lynne Parker. Your guidance and support has been invaluable through this process and through the prior years of both my master's and undergraduate careers. I would also like to thank Dr. Lonnie Love who helped me get into robotics and into additive manufacturing and continued to guide me throughout the years.

Abstract

One of the most important tasks for an autonomous robot is figuring out how to move from its current position and orientation to a new position and orientation. Despite significant progress in this area, active research is constantly looking for better ways to plan paths and traverse them. A specific type of path planning called coverage creates a path so that when the robot traverses the path its footprint covers the entire space. Most classical path planning and coverage path planning algorithms have some form of decomposition method for the target space. This work aims to provide a general way to decompose and represent the space so that it can be used both for classical path planning and for coverage. This work accomplishes the aim through using circle packing to tile a space with tangent circles that are the same size as the robot that will traverse the path. These circles are then converted into a planar graph which can be used for path planning and for coverage. The representation created by this decomposition uses less memory than traditional decomposition methods in practice and allows for shorter paths with less curvature to be created in most situations. These statements are shown to hold true for both planning a path between two points in an environment and planning a path to cover the entire environment.

Table of Contents

1 Introduction **1**

1.1 Contributions 3

2 Background **5**

2.1 Geometry 5

2.2 Graph 7

2.3 Routing Algorithms 7

2.3.1 Traveling Salesman 8

2.3.2 Chinese Postman 8

2.4 Spline 9

2.4.1 Smoothness 9

3 Related Work **12**

3.1 Coverage 12

3.1.1 Heuristic 13

3.1.2 Exact Cellular Decomposition 14

3.1.3 Semi-Approximate Decomposition 16

3.1.4 Approximate Decomposition 17

3.1.5 Optimal Coverage 18

3.2 Circle Packing 19

3.2.1 Additive Manufacturing 19

3.3 Summary 20

4 Approach **21**

4.1 Environment Complexity . . . 22

4.2 Circle Packing . . . 23

4.3 Path Planning . . . 27

4.3.1 Wavefront . . . 27

4.3.2 Chinese Postman . . . 28

4.4 Path Constraints . . . 29

4.4.1 Robot Model . . . 29

4.4.2 Continuity in heading . . . 30

4.4.3 Continuity in curvature . . . 30

4.4.4 Correlation between shape and parameters . . . 31

4.4.5 Spline Selection . . . 31

4.5 Summary . . . 35

5 Evaluation **37**

5.1 Analytical Results . . . 37

5.2 Experiments . . . 42

5.2.1 Wavefront . . . 48

5.2.2 Coverage using Chinese Postman Algorithm . . . 51

5.3 Summary . . . 56

6 Conclusion **57**

6.1 Limitations . . . 57

6.2 Contributions . . . 58

6.3 Future Work . . . 58

Bibliography **60**

Vita **64**

List of Tables

4.1 Common Path Planning Spline Properties 32

List of Figures

1.1 Coverage 2

2.1 Polygons 6
2.2 Concavity 6
2.3 Turning Angle 6
2.4 Undirected, Unweighted Graph 7
2.5 Spline 9
2.6 Parametric Continuity 10
2.7 Geometric Continuity 11

3.1 Sweeping Motion 14
3.2 Trapezoid Decomposition 14
3.3 Boustrophedon Decomposition 15
3.4 Morse Decomposition 16
3.5 Semi-Approximate Decomposition 16
3.6 Grid Decomposition 17
3.7 Circle Pack 19

4.1 Environment Complexity Cross-Sections 22
4.2 Circle Packing 24
4.3 Circle Packing with Obstacles 24
4.4 Circle Pack with Edges 25
4.5 Circle Pack with Boundary Edges 26
4.6 Close up of Boundary Edges 26

4.7 Wavefront Algorithm . . . 28
4.8 Hermite Spline . . . 33
4.9 Bezier Spline . . . 35

5.1 Two Types of Square Grid Decomposition . . . 38
5.2 Square Flower . . . 38
5.3 Single motion in a square flower . . . 39
5.4 Circle Flower . . . 39
5.5 Single motion in circle flower . . . 40
5.6 Spline Average Curvature . . . 40
5.7 Circle Pack Move Options . . . 41
5.8 Square Grid Move Options (Top: Path; Bottom: Footprint) . . . 42
5.9 Experiment Environments . . . 43
5.10 Packings (Left: Circle Packed; Right: Square Grid) . . . 44
5.10 Continued . . . 45
5.10 Continued . . . 46
5.11 Boundary Edges - Odd Vertices . . . 47
5.12 Number of cells and edges . . . 48
5.13 Example Wavefront Runs . . . 49
5.14 Circle Packed Wavefront Stats . . . 50
5.15 Square Grid Paths as Percentage of Circle Pack Paths . . . 50
5.16 Example Vertical Wavefront Run . . . 51
5.17 Chinese Postman Footprint . . . 52
5.18 Chinese Postman . . . 53
5.19 Continued . . . 54
5.19 Continued . . . 55
5.20 Coverage Path Stats . . . 55
5.21 Area vs Length . . . 56

Chapter 1

Introduction

The importance of robots to society is ever increasing. Companies invest enormous amounts of money and resources in automation and robotics to enter new markets, decrease operation costs, increase efficiency, increase safety, or solve problems that are impossible for humans to complete. There are numerous applications that benefit from this increased support of automation.

One of the most important tasks for an autonomous robot is figuring out how to move from its current position and orientation to a new position and orientation. Despite significant progress in this area, active research is constantly looking for better ways to plan paths and traverse them. This has been extensively looked into for mobile robots with wheels on 2D surfaces. The goals for 2D classical path planning are to avoid obstacles and find the optimal path, which is usually the path with the shortest length or time needed to traverse it. Another popular constraint for the optimal path is the number of turns. Each turn that a robot has to make when traversing a path causes an increase in time for completion of the path. For each turn, the robot must decelerate, turn, and then accelerate. Each turn also causes the wheels to slip some. This slippage causes sensor readings to not be as accurate and in turn reduces the accuracy of the robot's localization, or where it thinks it is.

In addition to the classical path planning between two points, there is specific type of path planning called *coverage* (Figure 1.1). The coverage problem is as follows:

Given a robot with footprint $\mathcal{F}$ in a target space $\mathcal{S}$, possibly with holes or obstacles, the coverage problem is computing a path $\mathcal{P}$ contained in $\mathcal{S}$ such that traversal of $\mathcal{F}$ along the path results in each point in $\mathcal{S}$ being covered by $\mathcal{F}$.

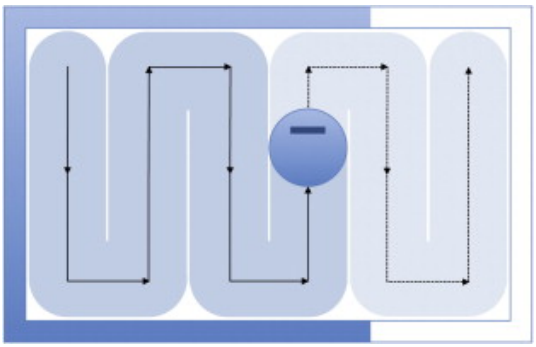


Figure 1.1: Coverage

Some examples of some highly important applications that fall under coverage include search and rescue, demining, natural disaster monitoring, and surveillance. Some other examples include floor cleaning, crop health monitoring, and ship hull inspection.

In the ideal case, when there is no uncertainty in the robot's state, the coverage problem can be framed as a classical path planning problem. There are several existing algorithms that can solve this problem and produce a complete coverage path. As the challenge of finding a solution that provides complete coverage has been solved, the current challenge is producing a "good" solution.

Each "good" solution is application specific. For example, a "good" coverage for floor cleaning could result in a clean floor with the least amount of time taken. For a painting robot, it would result in the most uniform paint layer on the surface. For crop monitoring a good solution would be measured by the quality of the capture of the entire field. These different qualifications for the "good" solution cause the works in literature that focus on one application to offer solutions that specialize to that application. Some of these solutions tend to pose undesirable properties when used for other applications such as an excessive amount of turns that makes the implementation challenging in practice.

While there are numerous algorithms for the various applications, most involve some form of decomposition method for the target space. A decomposition method is a way to break

up the target space into smaller pieces which can be used in both the classical path planning process and the coverage process. This work aims to provide a general way to decompose the space so that it can be used both for classical path planning and for coverage.

The specific problem this work attempts to solve is generating a representation of a target space that has properties that will cause generated paths for both classical path planning and for coverage to have less curvature and be shorter than paths created on current common space representations.

This work accomplishes the aim through using circle packing to tile a space with tangent circles that are the same size as the robot that will traverse the path. These circles are then converted into a planar graph which can be used for path planning and for coverage. The square grid is the most common representation for a target space and is compared against the circle packed representation. Both representations are used for classical path planning and for coverage and are compared on the path's properties. The paths planned on this new representation are shown to most of the time have less curvature and be shorter than paths planned on a square grid.

Chapter 2 provides background on some of the relevant subject areas used in this research. Chapter 3 contains a literature review of robot coverage and circle packing. Chapter 4 describes the methodology and approach for decomposing the space and for evaluating the space representation in path planning and coverage. Chapter 5 discusses the analytical results and the experiment results for validating the representation of the space. Chapter 6 concludes this report.

1.1 Contributions

The contributions of this thesis are the discoveries of the advantages, over other decomposition methods, of using circle packing to decompose a target space into a graph for both path planning and for coverage. The representation created by this decomposition uses less memory than traditional decomposition methods in practice and allows for shorter paths with less curvature to be created in most situations. These statements hold true for both

planning a path between two points in an environment and planning a path to cover the entire environment.

Chapter 2

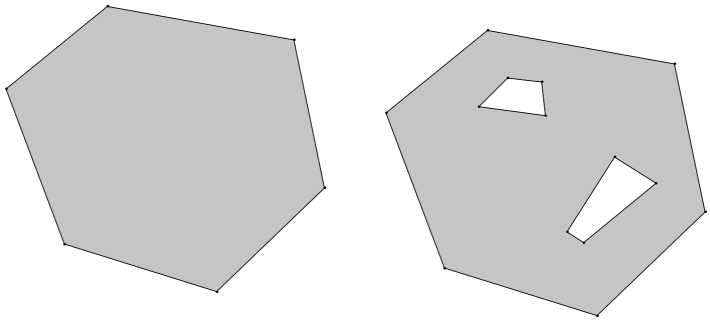
Background

For understanding this thesis there are several subject areas that the reader should know. Here these are briefly discussed. For each subject area more can be learned from other referenced sources.

2.1 Geometry

There are several elements of computational geometry that are dealt within this thesis. More information can be gathered from [6], but everything needed for understanding this thesis will be explained here.

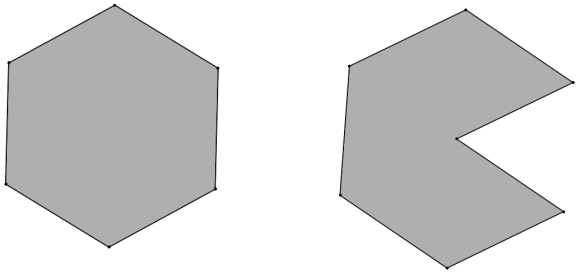
Only 2-dimensional environments are used with polygonal objects (e.g., objects that consist of segments and points). A segment is a line between two points in space. A *simple polygon* is a closed chain of segments which do not self-intersect (Figure 2.1a). A *non-simple polygon* might also have some simple polygons (holes) within its area (Figure. 2.1b).



(a) Simple Polygon **(b)** Polygon with holes

Figure 2.1: Polygons

A polygon is *convex* if it is a simple polygon in which all interior angles are less than or equal to 180° (Figure 2.2a). A polygon is *concave* if at least one interior angle is a reflex interior angle, or in other words, between 180° and 360° (Figure 2.2b).



(a) Convex **(b)** Concave

Figure 2.2: Concavity

A *polygonal path* is a chain of incident segments between two points and can also be considered as a sequence of points.

The *turning angle* between two incident segments is the minimum angle, θ, between them as shown in Figure 2.3.

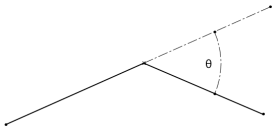


Figure 2.3: Turning Angle

2.2 Graph

Graphs are used throughout much of the work contained within this thesis and the reader should have some basic knowledge. A basic explanation is presented, but more is offered in [25].

An *undirected graph* $\mathcal{G}(\mathcal{V}, \mathcal{E})$ consists of a set of vertices V and a set of edges $\mathcal{E}$ where an edge $e = \{u, v\}$ connects the vertices $u \in \mathcal{V}$ and $v \in \mathcal{V}$. A *weighted graph* $\mathcal{G}(\mathcal{V}, \mathcal{E})$ additionally has a weight w assigned to each edge. In the case that a graph is unweighted then a default weight of 1 is assigned to each edge. A directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ contains edges (also called arcs) that are directed ($e = (u, v) \in \mathcal{E}$ allows transition from u to v, but not from v to u).

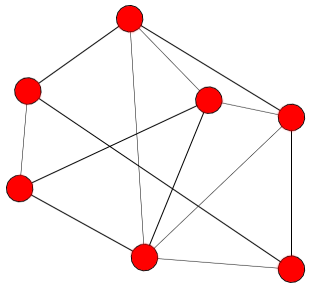


Figure 2.4: Undirected, Unweighted Graph

2.3 Routing Algorithms

Routing algorithms aim to determine the least-cost traversal of a specified subset of a graph, with or without constraints. These problems are found in a variety of practical contexts and have long been an object of attention. Specific versions have become fairly famous. One of the earliest is the Konigsberg bridge problem [9] in which the goal is to determine if there exists a closed walk traversing each of seven bridges once. A *closed walk* consists of a sequence of vertices starting and ending at the same vertex. Each two consecutive vertices in the sequence are adjacent to each other and connected by an edge in the graph. This problem led to Euler working on the constraints for the existence of a closed walk [9].

Theorem 2.1 (Euler's Theorem)**.** *For a connected multi-graph G, G is Eulerian if and only if every vertex has even degree.*

Corollary 2.2. *A connected multi-graph G is semi-Eulerian if and only if there are exactly 2 vertices of odd degree.*

A *graph* is connected when there is a path between every pair of vertices, meaning there are no unreachable vertices. A *multigraph* is a graph which is permitted to have multiple edges between two vertices. A vertex has an even degree when there are an even number of edges coming out of it and has an odd degree when there are an odd number of edges coming out of it. Euler in his theorem (2.1) proved that an *Eulerian* or *Semi-Eularian* graph is needed to create an Eulerian path (or trail). An *Eulerian path* is sequence that visits every edge exactly once. An Eulerian path can be computed from a Semi-Eulerian graph, while a *Eulerian circuit*, which starts and ends on the start vertex while being a Eulerian path, can only be computed from an Eulerian graph.

2.3.1 Traveling Salesman

The traveling salesman problem (TSP) [15] searches for a tour of a graph that visits every vertex in a graph exactly once while minimizing the edge weights. The problem can simply be stated as:

> If a traveling salesman wishes to visit exactly once each of a list of m cities (where the cost of traveling from city i to city j is c_{ij}) and then return to the home city, what is the least costly route the traveling salesman can take?

The TSP is NP-hard and without additional constraints on the edge weights, no approximation algorithm can exist [15].

2.3.2 Chinese Postman

Another well-known routing problem is the Chinese Postman Problem (CPP) [9] which is formulated as:

A mailman has to cover his or her assigned segment before returning to the post office. The problem is to find the shortest walking distance for the mailman.

For the CPP the property of being Eulerian or Semi-Eulerian becomes very important. The algorithms for solving it contain two main stages. The first is a minimum matching algorithm. This stage determines what edges with the least total cost need to be added in order to have a Eulerian graph by converting each odd vertex into an even one. This can be done in polynomial time. Once these edges have been added to the graph, the traversal can also be found in polynomial time [9].

2.4 Spline

Splines have been described and presented in a number of books and papers [22] so only a quick overview will be given. A two-dimensional curve is created by combining two splines: $x(t)$ and $y(t)$. Each spline consists of one or more segments that are formed as polynomial parametric curves (Figure 2.5). All segments of the spline are required to be polynomials of the same degree n. The point of tangency of two adjacent segments is called a knot. Each segment has a series of control points that shape the spline. When the knots are set, the spline's control points can be obtained by solving systems of linear equations.

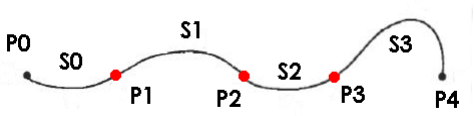


Figure 2.5: Spline

2.4.1 Smoothness

In mathematical analysis, a function's smoothness is measured by the number of derivatives it has which are continuous. A smooth function has derivatives of all orders everywhere in its domain. Functions are classified by their differentiability classes which are based on their derivatives. Higher order differentiability classes correspond to the existence of more derivatives.

2.4.1.1 Parametric Continuity

Parametric continuity is the smoothness of the curve and its parameterization (Figure 2.6).The function f is said to be of differentiability class C^k if the derivatives f', f'', ..., $f^{(k)}$ exist and are smooth. Another explanation is the class C^0 contains all continuous functions. Class C^1 contains all functions whose derivatives are continuous.

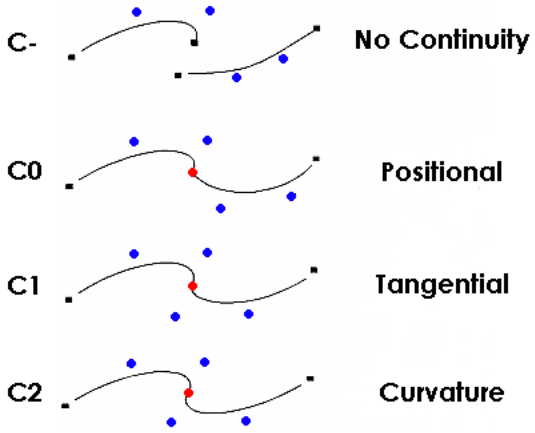


Figure 2.6: Parametric Continuity (Red: Knots; Blue: Control Points)

2.4.1.2 Geometric Continuity

Geometric continuity is the smoothness of the join point of two curves. Consider the segments on either side of a point on a curve (Figure 2.7). For G^0 continuity the curves must touch at the join point. For G^1 continuity the curves must share a common tangent direction at the join point. For G^2 continuity the curves must share a common center of curvature at the join point.

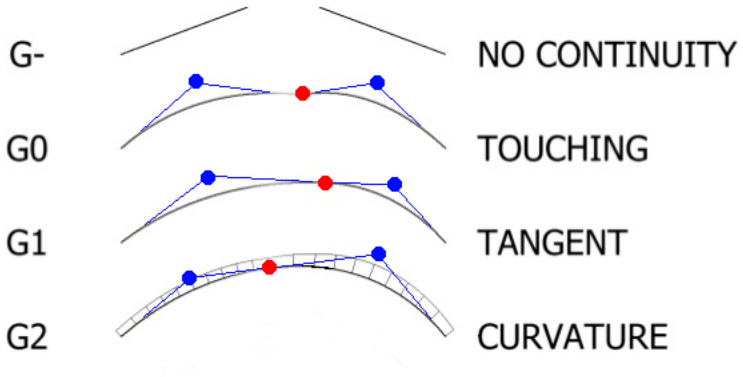


Figure 2.7: Geometric Continuity

Chapter 3

Related Work

There is a vast set of literature on robotics path planning and coverage. Presented here is a brief explanation of the existing work on decomposing the target space for these domains. Also, circle packing has been used for numerous applications and thus has its own large set of literature. Circle packing is briefly explored here.

3.1 Coverage

Robot coverage involves determining a path that passes over all points of an area or volume of interest while avoiding obstacles. As such it is a specific type of the robot path planning problem. There are many applications that make use of robot coverage such as cleaning robots, automated lawnmowers, and land mine detection robots.

Choset in [3], one of the earliest works on robot coverage, defines the criteria needed for coverage as:

1. The robot must move through all the points in the target area.
2. The robot should fill the region without overlapping paths.
3. The robot must avoid all obstacles.
4. Simple motion trajectories (e.g., lines or circles) should be used.
5. An "optimal" path is desired under available conditions.

It is not always possible to satisfy all of these criteria in complex environments.

The coverage problem is related to several other well known NP-hard problems [3] such as the covering salesman problem, a variant of the traveling salesman problem; the lawnmower problem, and the piano mover's problem. The covering salesman problem involves an agent visiting a neighborhood of each city. The lawnmower problem involves finding a path to cut all grass in a given region covered by grass. The piano mover's problem involves finding a collision-free path from a start configuration to a goal configuration. The coverage problem involves an agent passing over all points in a target space.

Coverage algorithms can be classified as offline or online. Offline algorithms rely only on static information, and assume that the information is known in advance. Online algorithms utilize real-time sensor measurements to cover the target space. This work falls into the category of offline algorithms; however, it can be modified to work as an online algorithm.

Choset in [3] and Galceran in [12] have presented surveys on the coverage body of research and have classified different types of coverage algorithms. Most coverage algorithms decompose the space into subregions called cells and so most of these categories are based on the type of decomposition used. Choset in [3] divided algorithms into heuristic, approximate, semi-approximate, and exact whereas Galceran in [12] breaks them up further.

3.1.1 Heuristic

Heuristic methods do not rely on any kind of maps and cannot guarantee complete coverage. These methods follow a set of simple behaviors and do not require as many sensors, reducing the cost of creating a robot to implement these methods.

3.1.1.1 Random Search

The most common heuristic method is the random search in which a robot moves in a straight line until an obstacle is found. It then turns in a random direction. This is an approach that several floor-cleaning robots use. The idea is that if the floor is swept randomly long enough then it should become clean. The advantages to this approach are the lack of need for sensors or expensive computational hardware. However, this is not as successful for large

spaces or 3D spaces as the cost of operating the vehicle would be too high. Also there are many cases in which the robot will cover the same area repeatedly.

3.1.2 Exact Cellular Decomposition

Exact cellular decomposition breaks the space down into simple, non-overlapping regions called cells. The union of these cells fills the entire target space. These regions where there are no obstacles can be covered through a sweeping motion such as a zigzag motion as shown in Figure 3.1. Generation of these motions called Seed Spreader motion are well documented in [4, 18]. This reduces the path planning for coverage to planning of motions from one cell to another. There are 3 common off-line approaches for exact cellular decomposition: Trapezoid, Boustrophedon, and Morse.

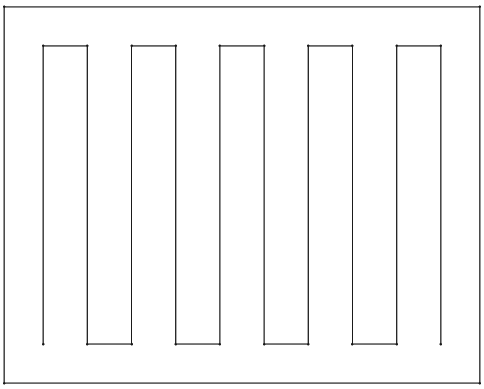
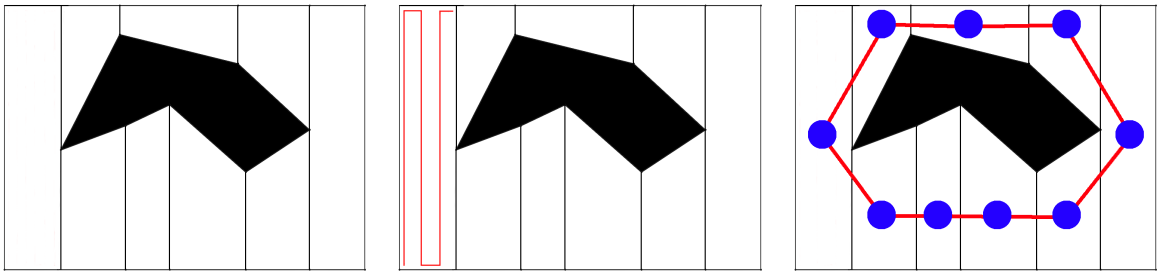


Figure 3.1: Sweeping Motion

3.1.2.1 Trapezoid Decomposition

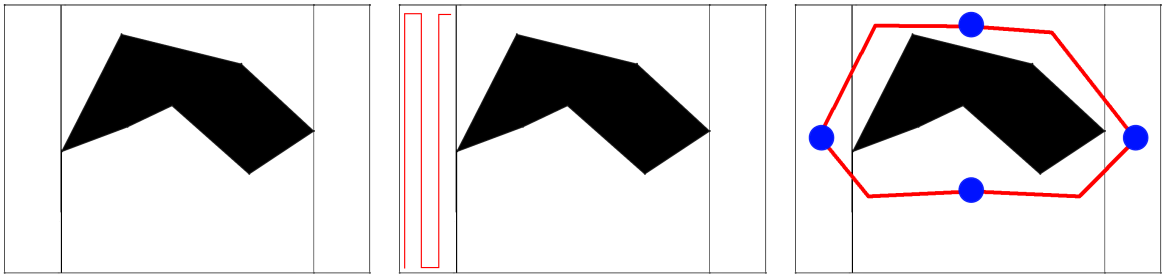


(a) Decomposition **(b)** Individual Cell Path **(c)** Cell Order

Figure 3.2: Trapezoid Decomposition

The trapezoid decomposition works on planar polygonal spaces and creates trapezoid cells as shown in Figure 3.2a. Each vertex has a line segment extended vertically up, down, or both to create the trapezoids. The trapezoid shaped cells allow for simple zigzag motions to cover each cell (Figure 3.2b). If each cell is treated as a vertex and edges connect adjacent cells (Figure 3.2c), then finding an exhaustive walk through the graph associated with the decomposition guarantees complete coverage.

3.1.2.2 Boustrophedon Decomposition

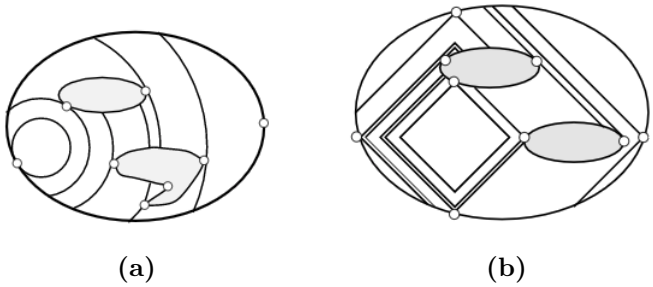


(a) Decomposition **(b)** Individual Cell Path **(c)** Cell Order

Figure 3.3: Boustrophedon Decomposition

The boustrophedon decomposition improves on the trapezoid decomposition by merging some of the cells together as shown in Figure 3.3a [4].The boustrophedon decomposition only considers vertices where a vertical segment can be extended above and below the vertex and thus can create non-convex cells. This reduces the number of cells and so shorter coverage paths are obtained.

3.1.2.3 Morse Decomposition



(a) **(b)**

Figure 3.4: Morse Decomposition [1]

The boustrophedon decomposition was later proven by [1] to be specific case of morse decomposition (Figure 3.4). Morse decomposition uses the critical points of a morse function to define the cells and thus can handle non-polygonal spaces. A morse function $f : M \to \mathbb{R}$ is defined as a function in which all its critical points are non-degenerate and M is a manifold.

3.1.3 Semi-Approximate Decomposition

Hert in [14] presents a coverage algorithm where the cells are fixed in width, but can have any length. Their algorithm works on both simply and non-simply connected environments through recursion. A robot following this algorithm can start at any point in the environment and will use a sweeping motion to cover the space. The main different between this and the exact decomposition is the exact decomposition uses the vertices of the obstacles to subdivide the space whereas in semi-approximate the cells simply are of fixed width.

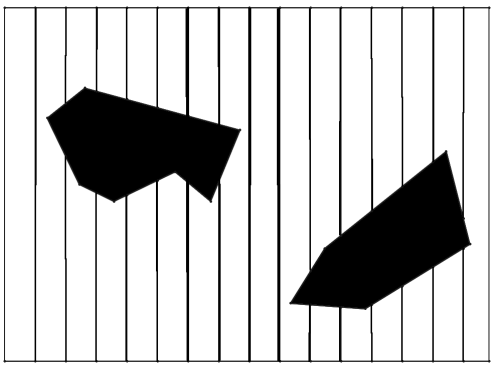


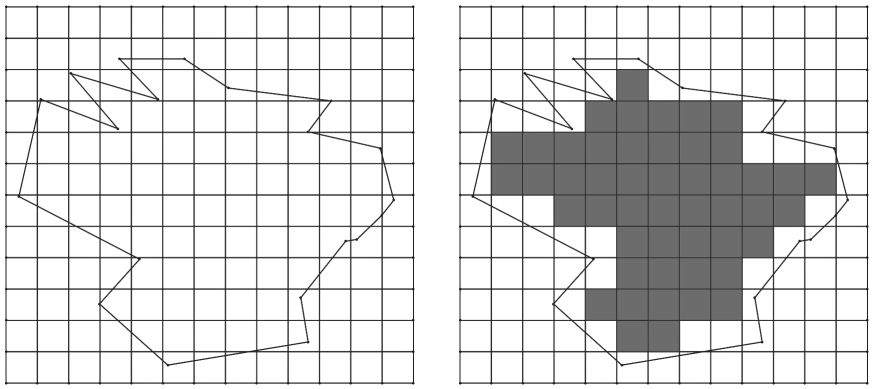
Figure 3.5: Semi-Approximate Decomposition

3.1.4 Approximate Decomposition

The third classification for coverage decompositions used by [3] is the approximate decomposition in which the space is broken down into cells with the same size and shape. The union of these cells only approximates the target space. It is assumed that once the robot enters a cell it has covered the entire cell. Approximate decomposition algorithms are considered resolution complete as they completely cover their cells; however, depending on the shape of the cells and the shape of the target space the union of the cells may not completely cover the space. This is the category of decomposition that the algorithm presented in this thesis falls into.

3.1.4.1 Grid-based methods

Most approximate decomposition methods create a map of the environment and then divide it into a grid and thus are known as grid-based (Figure 3.6). Each grid cell usually has an associated value of whether or not it contains an obstacle or is free, or the probability of it containing an obstacle [10]. Once a robot moves into a cell then it is considered covered, so the cell size must reflect the robot's coverage range.



(a) **(b)**

Figure 3.6: Grid Decomposition

It is easy to create a grid map as it can be represented as an array with each element containing the occupancy information. This makes it simple to mark covered areas. For these reasons, grid-based representations are the most widely used representation for coverage

algorithms and for classical path planning algorithms. However, they do have limitations in that they suffer from exponential growth of memory usage as the resolution remains constant regardless of the complexity of the environment [12].

Usually the cells are squares, but other shapes have been used such as triangles by [21]. The rationale behind the use of triangular cells is that they offer a higher resolution in comparison to the rectangular cells of the same size. However, the resolution of the grid can also be augmented by using finer-grained squared cells. Most mobile robots are not capable of making very fine movements adjustments and so there isn't need for ultra high resolution in coverage path planning [12].

3.1.5 Optimal Coverage

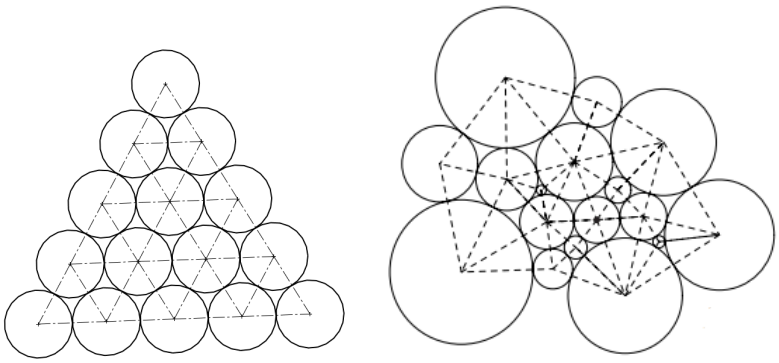
Many works have discussed optimal coverage using different factors for evaluation such as the number of turns, number of revisited cells, distance, and time taken. It is only possible to find an optimal solution for an a priori known environment since an antagonistic example can always be found for a sensor-based approach. Therefore, optimal coverage methods are classified as offline methods.

Huang in [16] presents an optimal line-sweep based method for cellular decomposition algorithms in planar spaces. This approach produces an optimal length coverage path by allowing different sweep directions in the zigzag paths used to cover each cell. The main idea is to minimize the number of turns in the path because each turn implies the additional cost of the robot having to decelerate before the turn and accelerate after the turn. The method intends to maximize the length of the laps in the zigzag pattern in order to minimize the number of turns.

Mannadiar in [19] proposes an algorithm based on the boustrophedon cellular decomposition that achieves complete coverage while minimizing the path of the robot. The optimal solution to the Chinese Postman Problem is used to calculate an Euler tour, which guarantees complete coverage of the available space while minimizing the coverage path length.

3.2 Circle Packing

A circle packing (CP) (Figure 3.7) is a configuration of circles having a specified pattern of tangencies, as introduced by [24]. Thus it enjoys both a combinatoric nature in the pattern of tangencies and geometric nature in the radii of the circles. There exists a rather extensive theory covering the existence and uniqueness of these radii [5]. Numerous uses of circle packings exist, both theoretical and practical such as discrete conformal mapping [13], graph embedding [20], and conformal tilings [2].



(a) Uniform Circle Pack **(b)** Non-Uniform Circle Pack

Figure 3.7: Circle Pack

Circle packing is used as the method of decomposing the space and converting it to a graph.

3.2.1 Additive Manufacturing

The path planning for Additive Manufacturing (AM) or 3D Printing (3DP) is a subset of the robot coverage problem as the goal is to fill a space with material without having to cross over where material has already been used. Recently circle packing has been used to decompose the space for Additive Manufacturing in order to allow for a planning scheme based on physical models instead of the traditional raster or spiral patterns. In [7] circle packing was used to create a graph which was used to create a path for 3D printing. In [8] a heuristic circle packing algorithm is proposed for creating a circle pack within a specified

polygon that can then be used for optimizing the toolpath for 3D printing. In [17] the previous work was used to incorporate internal stress distribution into the design of infill patterns for additive manufacturing. This work significantly influenced the current work; however, criteria used for AM was a path that increased the strength of the final printed part and the curvature of the path was not explored. Also, for AM a gantry robot was used to deposit material instead of a wheeled mobile robot.

3.3 Summary

Coverage is the process of determining a path that causes the footprint of a robot when traversed over the path to cover the entire space. Most coverage algorithms have a decomposition method. There are three main categories for decomposition: exact, semi-approximate, and approximate. This work deals with an approximate decomposition algorithm that can also be used for classical path planning.

Circle Packing is the process and theory behind creating a configuration of circles in which each circle is tangent to its neighbors. This has previously been used in Additive Manufacturing to solve the problem of how to plan the path for printing a layer. This work influenced the concept of using circle packing for robot coverage.

Chapter 4

Approach

This work seeks to determine the advantages of using circle packing to decompose a target space for both classical path planning between two points and coverage path planning. Specifically, this work explores the amount of memory used in the representation and whether most paths planned on it have less curvature and are shorter than paths planned on the commonly used square grid decomposition.

In order validate the claim that paths planned on the circle pack representation most of the time will be shorter and have less curvature than paths planned on the common square grid, a process composed of several steps was created. The first step selects multiple environments that vary in terms of size, shape, number of obstacles, and complexity. Each of the varied environments is then decomposed by packing the space with circles that are tangent to one another. These circles are then converted to a planar graph which can be used for classical path planning and coverage path planning. A similar process using squares is applied to each environment to create a square grid for comparison. The Wavefront Algorithm is then used to plan paths between two points on both the circle packed and square grid representations. This algorithm only outputs a series of points that the robot needs to pass through. Splines are then used to connect each point with the next point in the series. Certain criteria for which type of spline should be used are specified in order to select a spline that a real robot is able to follow. These splines are also needed in order to measure the curvature of the path. This process is used to test whether most of the classical

paths planned on the circle packed representation are shorter and have less curvature than a path planned between the same two points on the square grid.

For coverage path planning a similar process is used with the exception of the Chinese Postman Algorithm being used to plan a coverage path instead of the Wavefront Algorithm being used to plan a classical path.

In creating this approach there were several assumptions that were made. The robot is on a flat 2D surface. The robot is circular in shape and has a differential drivetrain. There are zero or more obstacles in the environment. The boundary and all obstacles are made of polygonal shapes. This work can be expanded upon to remove some of these assumptions.

4.1 Environment Complexity

In order to determine the advantages of the circle packing as a decomposition method, it needs to be tested on multiple environments with different complexities. There are several different ways that one can measure the complexity of an environment. For this thesis, a simple common method of calculating the complexity of an environment through averaging the number of intersections in a large number of 1D cross sections is used (Figure 4.1).

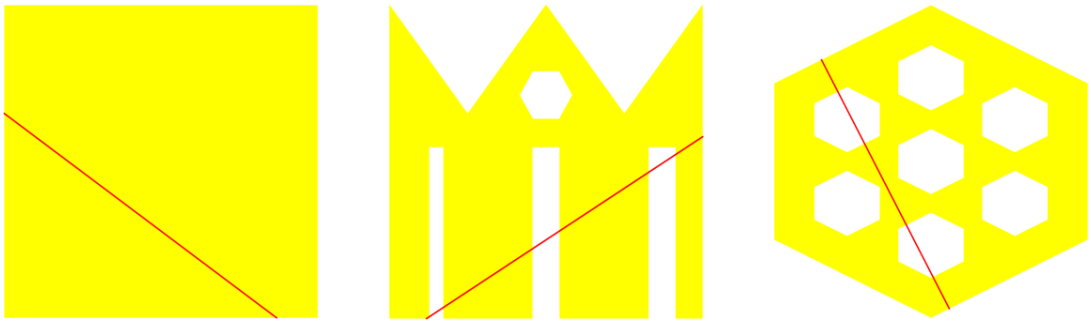


Figure 4.1: Environment Complexity Cross-Sections

Algorithm 1 selects two random points on the boundary of the environment. These random points are checked to make sure that they are not on the same boundary edge. A line is created between these two points and the number of intersections between the line and the obstacles/boundary is determined. This is done 1000 times with the number

of intersections being averaged. This average is used to represent the complexity of the environment. This shows how many obstacles the robot would have to go around on average when going traveling through the environment.

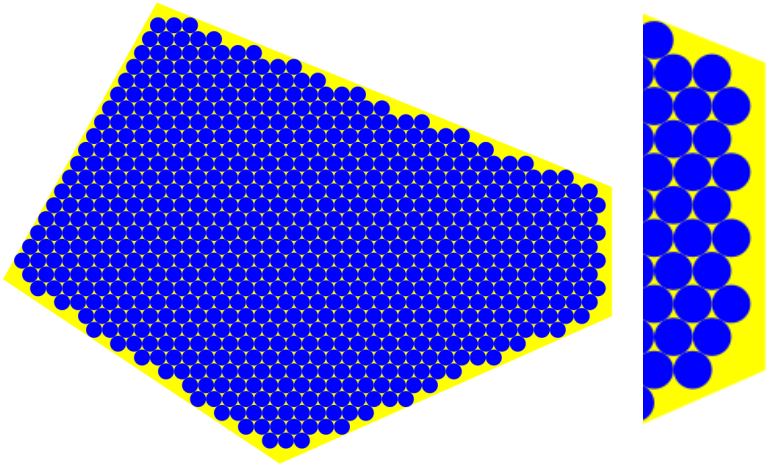
Algorithm 1 Measuring Environment Complexity

```
1: procedure
2:     boundary ← boundary(environment)
3:     sum ← 0
4:     for i ← 0 : n do
5:         p_1 ← random_point(boundary)
6:         p_2 ← random_point(boundary)
7:         while p_1 == p_2 or same_boundary_edge(p_1, p_2) do
8:             p_2 ← random_point(boundary)
9:         sum ← sum + num_intersections(environment, p_1, p_2)
10:    complexity ← sum/n
```

This algorithm is used to select environments with different amounts of complexity.

4.2 Circle Packing

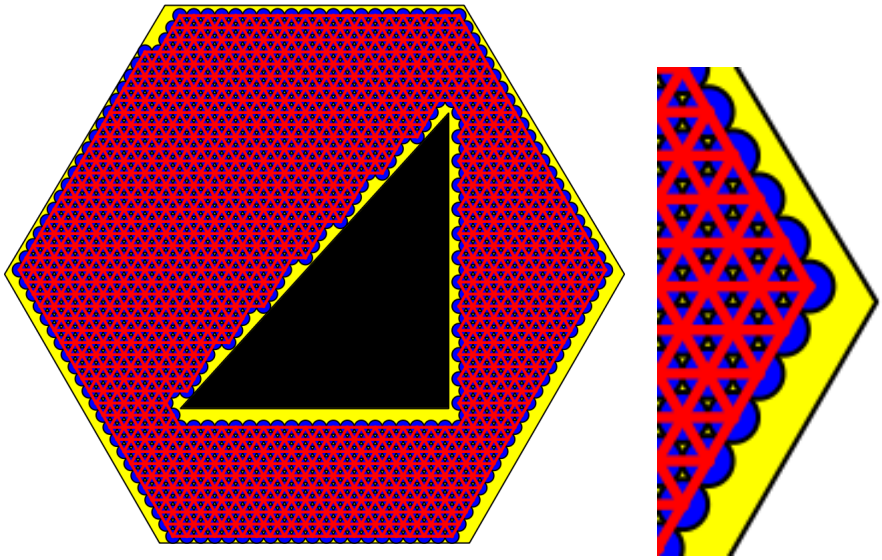
The target space $\mathcal{S}$ is filled with adjacent circles of equal size (Figure 4.2). There are multiple possible uniform circle packings. Lagrange and later Thue proved that in two dimensional Euclidean space the highest-density lattice arrangement of circles is the hexagonal packing arrangement, where the centers of the circles are arranged in a hexagonal lattice and each circle is surrounded by six other circles [11]. This hexagonal packing arrangement is thus used to maximize the amount of space covered by the circles. Approximate decomposition methods are resolution complete meaning that a coverage created from them will completely cover the representation, but the representation will not necessarily cover the entire target space. How much of the target space is covered depends on the resolution of the decomposing shape (e.g., square, triangle, circle, etc). This method is a specific type of approximate decomposition and is resolution complete.



(a) Full Shape **(b)** Close Up

Figure 4.2: Circle Packing

The circle packing can handle any space with a single exterior boundary and zero or more obstacles or holes in the interior (Figure 4.3). For the purposes of this thesis a polygonal shape was used for the exterior and the holes; however, any closed entity could be used such as a closed looped spline.



(a) Full Shape **(b)** Close Up

Figure 4.3: Circle Packing with Obstacles

This circle packing is then converted to a planar graph (Figure 4.4). The center of each circle is represented as a vertex. Tangency between two adjacent circles is represented as an edge between their centers.

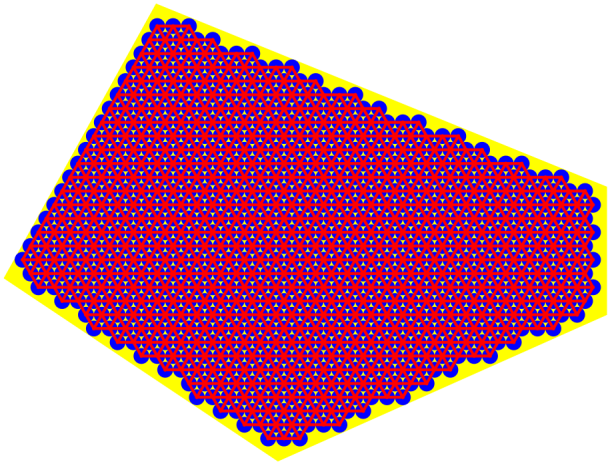


Figure 4.4: Circle Pack with Edges

Because of how the circle packing is formed all interior vertices have six edges making each interior vertex an even vertex. Some of the boundary vertices have an odd number of edges. For an Eulerian tour, which is needed for a coverage path, odd vertices have to be connected. In order to reduce the number of odd vertices in the graph some of the boundary vertices are connected to non-adjacent boundary vertices within a specified distance (Figure 4.6). Algorithm 2 loops through the vertices on the boundary. If a boundary vertex is odd then an attempt is made to connect it to another odd boundary vertex that is within twice the width of the robot. If it is close enough then it is checked whether the robot can travel there without colliding with an obstacle or the boundary of the environment.

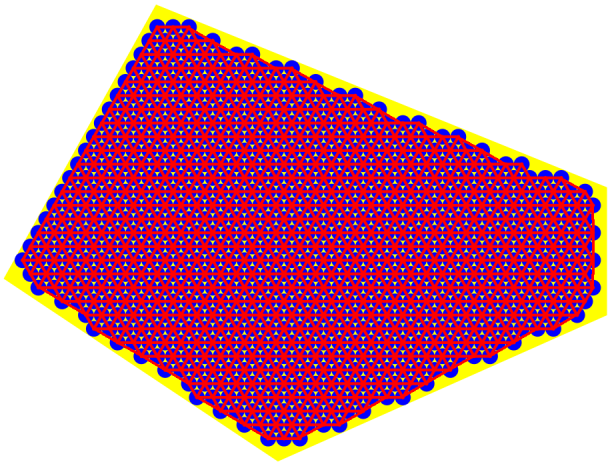
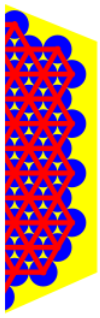
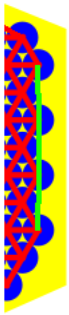


Figure 4.5: Circle Pack with Boundary Edges



(a) Circle Pack without Boundary Edges



(b) Circle Pack with Boundary Edges (Boundary edges in green)

Figure 4.6: Close up of Boundary Edges

Algorithm 2 Add Boundary Edges to Odd Boundary Vertices

```
1: procedure
2:     g ← graph
3:     exterior ← exterior_face(graph)
4:     exterior_iterator ← face_edge_iterator(exterior)
5:     for edge in exterior_iterator do
6:         vertex ← origin(edge)
7:         if odd(vertex) then
8:             continue
9:         target_edge ← edge_increment(exterior_iterator, 2)
10:        target_vertex ← origin(target_edge)
11:        if distance(vertex, target_vertex) < robot_footprint then
12:            for edge2 in exterior_iterator do
13:                valid ← 1
14:                edge2_origin ← origin(edge2)
15:                edge2_target ← target(edge2)
16:                if intersect(line(vertex, target_vertex), line(edge2_origin, edge2_target)) then
17:                    valid ← 0 break
18:            if valid then
19:                add_edge(g, vertex, target_vertex)
```

4.3 Path Planning

Once the graph is created, a path planning algorithm is applied in order to test the properties of the path. For a classical planning between two points the Wavefront Algorithm is used. For a coverage path the Chinese Postman Algorithm is used. Both of these algorithms output a series of points that the robot must go through.

4.3.1 Wavefront

The Wavefront Algorithm (Figure 4.7), also known as the Distance Transform Algorithm, is used in order to plan a path between two points. The Wavefront Algorithm finds a path by applying a breadth-first search (BFS) on the graph from the goal cell. As the BFS traverses the space each cell is assigned a value which corresponds to the number of moves required for

the shortest path from that cell to the goal. Upon reaching the start the number of moves needed is known. The path to the goal cell can be found by selecting the cell adjacent to the start cell that has the smallest value. The cell adjacent from that with the smallest value is then selected. This step is repeated until the goal is adjacent to the current cell.

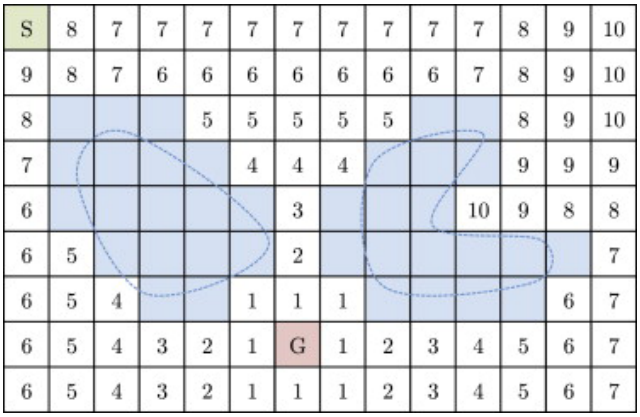


Figure 4.7: Wavefront Algorithm [12]

For the classical path planning experiments, the start and goal points were randomly selected from the target space. The containing circle or square was selected as either the start or the end cell. In the case that the random point was in the gap between circles the closest circle was selected. If a random point ended up inside one of the obstacles or outside the boundary then it was reselected. This was run for both the square grid and the circle pack. Each of the six environments was used 1000 times with different random start and goal points.

4.3.2 Chinese Postman

The Chinese Postman Problem (CPP) seeks to find the lowest-cost tour of a graph that includes each edge at least once [9]. This is known as a Eulerian tour. An odd degree vertex has an odd number of edges. An even degree vertex is a vertex that has an even number of edges. The general CPP starts by transforming all odd degree vertices into even degree vertices. This is done by finding a minimum weight matching. A Minimum Weight Matching is a list of the pairs of odd degree vertices that would have the minimum weight when traversing from one to the other. The new edge connecting the pair is then added

to the original graph. A Eulerian tour of the graph is then created using the End-Pairing algorithm. This Eulerian tour guarantees complete coverage of the available free space.

For the coverage experiments, a modified version of the Chinese Postman Algorithm is used to determine the order of the vertices to traverse in order to cover the entire space with the least amount of turning. As with the normal Chinese Postman Algorithm, a Minimum Weight Matching is applied in order to add new edges and make each vertex an even vertex. This Minimum Weight Matching uses the number of cells in between two vertices as the weight for the matching. This reduces the number of cells covered multiple times. The results from this Minimum Weight Matching are connected and the new edges are added to the graph making all vertices even. During the End-Pairing, the Eulerian tour that minimizes the amount of turning the robot would need is then selected as the coverage path.

4.4 Path Constraints

The output of each of the path planning algorithms is a series of points that the robot must go through. Splines are then needed to connect each of these points in way that the robot can actually traverse. For the robot to be able to traverse these splines they must have certain constraints. Without these constraints it cannot be guaranteed that a robot can actually follow the path due to a number of factors including slippage and loss of localization from odometry.

4.4.1 Robot Model

Both experiments are completed with a simulated differential drive robot of circular shape. The movement of the robot can be modeled with the following equations:

$$\dot{x} = vcos(\gamma) \qquad \dot{y} = vsin(\gamma) \qquad \dot{\gamma} = \omega \tag{4.1}$$

where x, y, γ stand for the position and orientation, v is the magnitude of the linear velocity, and ω is the angular velocity. If the wheels are not sliding then the velocities are close to

what has been sent to the robot. To assure that a real robot can follow the prescribed path it must not slide, i.e., its accelerations must be within certain limits.

In order to guarantee that the robot modeled can actually follow a planned path there are several properties that are necessary.

4.4.2 Continuity in heading

The path is required to be G^1 and C^1 continuous to make sure that the heading of the robot changes continuously. If this wasn't the case then the robot would have to be able to execute infinite rotational accelerations which introduces slippage and decreases the accuracy of the localization.

4.4.3 Continuity in curvature

A discontinuity in curvature also requires infinitely high accelerations. Even if the robot could cope with the accelerations, the discontinuity would imply strong abrupt forces to the robot itself and its payload.

The curvature κ is the inverse of a curve's radius and is defined in [23] as follows:

$$\kappa = |\frac{dT}{ds}| \tag{4.2}$$

where $\mathcal{T}$ is the tangent vector function to the curve and $\int$ the curve's arc length. Sprunk in [23] shows that for a curve $\mathcal{F}$ the parameterization of the curvature is

$$\kappa(t) = \frac{F'(t) * F''(t)}{|F'(t)|^3}. \tag{4.3}$$

As can be seen, control over the first and second derivatives of a parametric curve provides control over its curvature. Choosing first and second derivatives suitable for C^2 continuity implies curvature continuity of the spline.

When applied to a 2 dimensional spline the equation becomes as follows:

$$\kappa = \frac{|\dot{x}\ddot{y} - \dot{y}\ddot{x}|}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \tag{4.4}$$

4.4.4 Correlation between shape and parameters

When planning a path through an environment, a spline must be defined that passes through certain points (such as the vertices of the graph) and stays clear of certain areas (e.g., obstacles). This task is facilitated a lot if there are direct correlations between the spline's shape and its parameters. Otherwise complex inverse calculations would have to be employed to retrieve spline instances with given shape properties.

4.4.5 Spline Selection

In order to create the path that will allow comparison of the curvature properties of the space representations, a spline must be selected that fulfills the previously stated criteria. Path planning has a number of commonly used splines. These include Hermite, Bezier, Catmull-Rom, B-splines and many more. Hermite and Bezier Splines are defined on a per-segment basis. This causes them to require manual stitching. Manual Stitching involves actively connecting two spline segments so they both are touching and tangent at the end points. During the stitching, they can be connected so that they are C^2 continuous. On the other hand, Catmull-Rom and B-splines are defined so that they already include the stitching. While B-Splines provide C^2 continuity inherently, they lack the intuitive correlation between shape and control points. Catmull-Rom has the intuitive correlation between shape and control points, but is not guaranteed to be C^2 continuous.

Table 4.1: Common Path Planning Spline Properties
* Can be stitched so that it has C^2 continuity

Properties	**Hermite**	**Bezier**	**Catmull-Rom**	**B-Spline**
Stitching	Manual	Manual	Automatically	Automatically
C^1 **continuity**	Yes	Yes	Yes	Yes
C^2 **continuity**	Not inherently*	Not inherently*	No	Yes
Correlation between shape and parameter	Yes	Yes	Yes	No
Freely set first derivative	Yes	Yes	Yes	Yes

Of the common spline types discussed in Table 4.1, only the Cubic Hermite and the Cubic Bezier spline meet the required criteria. Both are cubic meaning they are made of segments S_i that are cubic polynomials, generally defined as:

$$S_i(t) = a_i t^3 + b_i t^2 + c_i t + d_i, \quad t \in [0, 1]. \tag{4.5}$$

4.4.5.1 Cubic Hermite Spline

The Cubic Hermite has four control points:

- The start point
- The goal point
- The tangent vector at the start point
- The tangent vector at the goal point



Figure 4.8: Hermite Spline

As the spline must be G^1, C^1, and C^2 continuous as shown in the above constraints, the end point of one segment must be the starting point of the next or:

$$S_i(1) = S_{i+1}(0), \quad i \in [0, n-2] \tag{4.6}$$

and the tangent of the end point of one segment must be the tangent of the starting point of the next or:

$$\dot{S}_i(1) = \dot{S_{i+1}}(0), \quad i \in [0, n-2]. \tag{4.7}$$

This makes using them easy to use for path planning. On a [0, 1] interval the equation for the 1 dimensional polynomial is:

$$p(t) = (2t^3 - 3t^2 + 1)P_0 + (t^3 - 2t^2 + t)M_0 + (-2t^3 + 3t^2)P_1 + (t^3 - t^2)M_1, \quad 0 \leq t \leq 1 \tag{4.8}$$

where P_0 is the starting point, P_1 is the end point, M_0 is the tangent at the starting point, and M_1 is the tangent at the end point.

For a Cubic Hermite, the first derivative is :

$$p'(t) = (6t^2 - 6t)P_0 + (3t^2 - 4t + 1)M_0 + (-6t^2 + 6t)P_1 + (3t^2 - 2t)M_1 \tag{4.9}$$

and the second derivative is:

$$p''(t) = (12t - 6)P_0 + (6t - 4)M_0 + (-12t + 6)P_1 + (6t - 2)M_1. \tag{4.10}$$

These derivatives are used to calculate the curvature of the spline. This is used to compare the curvature properties of the proposed space representation to other common approximate space representations.

4.4.5.2 Cubic Bezier Spline

The Bezier spline uses the Bernstein polynomial. The Bernstein polynomial is defined as:

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}. \tag{4.11}$$

They have a number of useful properties including symmetry:

$$B_{i,n}(t) = B_{nm-i,n}(1-t),$$

positivity:

$$B_{i,n}(t) \geq 0,$$

and normalization:

$$\sum_{i=0}^{n} B_{i,n}(t) = 1.$$

The Bezier curve is defined as:

$$p(t) = (1 - t^3)P_0 + 3(1 - t^2)tP_1 + 3(1 - t)t^2P_2 + t^3P_3, \quad 0 \leq t \leq 1. \tag{4.12}$$

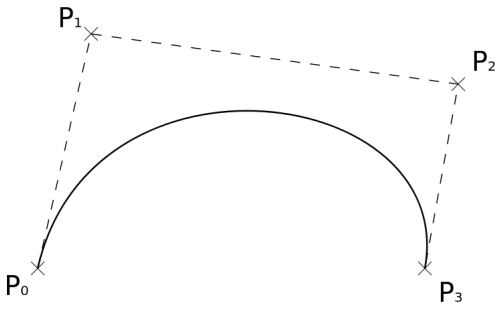


Figure 4.9: Bezier Spline

P_0 is the starting point and P_3 is the goal point. The vector from P_0 to P_1 is the tangent at P_0 and the vector from P_2 to P_3 is the tangent at P_3. The distance between P_1 and P_2 affects how far the curve moves towards P_1 before turning towards P_2. It also typically does not pass through P_1 or P_2, but just uses them for directional information. This is the most commonly used cubic spline.

For a Cubic Bezier, the first derivative is:

$$p'(t) = 3(1-t^2)(P_1 - P_0) + 6(1-t)t(P_2 - P_1) + 3t^2(P_3 - P_2) \tag{4.13}$$

and the second derivative is

$$p''(t) = 6(1-t)(P_2 - 2P_1 + P_0) + 6t(P_3 - 2P_2 + P_1). \tag{4.14}$$

These derivatives are used to calculate the curvature of the spline. This is used to compare the curvature properties of the proposed space representation to other common approximate space representations.

4.5 Summary

In order to determine the advantages of using circle packing to decompose a target space for both classical path planning between two points and coverage path planning, a process is developed. The hypothesis is that paths planned on the circle packed representation most of the time will be shorter and have less curvature than those planned on a common square

grid representation. The process used to validate this hypothesis was discussed. The first step selects multiple environments that vary in terms of size, shape, number of obstacles, and complexity. Each of varied environments is then decomposed by packing the space with circles that are tangent to one another. These circles are then converted to a planar graph which can be used for classical path planning and coverage path planning. A similar process using squares is applied to each environment to create a square grid for comparison. The Wavefront Algorithm is then used to plan paths between two points on both the circle packed and square grid representations. This algorithm only outputs a series of points that the robot needs to pass through. Splines are then used to connect each point with the next point in the series. Certain criteria for which type of spline should be used are specified in order to select a spline that a real robot is able to follow. Of the spline types looked at only the Cubic Hermite and the Cubic Bezier meet the criteria. These splines are also needed in order to measure the curvature of the path. This process is used to test whether most of the classical paths planned on the circle packed representation are shorter and have less curvature than a path planned between the same two points on the square grid.

For coverage path planning a similar process is used with the exception of the Chinese Postman Algorithm being used to plan a coverage path instead of the Wavefront Algorithm being used to plan a classical path.

Chapter 5

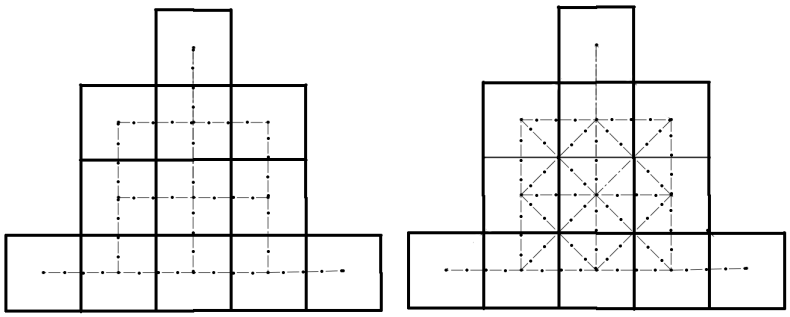
Evaluation

The hypothesis of this thesis is that a space representation generated through circle packing the space will have properties that cause most generated paths to have less curvature and be shorter than paths planned on other common approximate space representations such as the square grid. First this work analytically tested these claims by looking at moving from a single cell to an adjacent cell for each of the circle pack representation and square grid representation. Then the process described in the Approach was used to experimentally validate the hypothesis.

5.1 Analytical Results

Traversing a path on both a circle pack and a square grid can be simplified to a repetition of the same motions. The robot starts in a single cell and moves to one of the adjacent cells. This process repeats until the robot has fully completed the path. This fundamental part of the path is used to analytically determine the curvature properties of paths planned on both the circle pack and square grid.

The traditional grid cell decomposition has two types of representations. The first only allows horizontal and vertical movement between the centers of adjacent squares (Figure 5.1a). The second also allows for diagonal movements between the centers of adjacent squares (Figure 5.1b). For the comparison between a circle packed decomposition and the square grid representation, the version that includes diagonal movements will be used.



(a) **(b)**

Figure 5.1: Two Types of Square Grid Decomposition

In a traditional grid cell decomposition (Figure 5.2), there are eight adjacent cells to the one the robot is currently in; however, unless this is the very first cell then the robot just arrived from one of the cells, reducing the number of options to seven: straight ahead, 45° to the left or right, 90° to the left or right, and 135° to the left or right. The distance between the center of the current cell and the center of the adjacent cell depends on the cell the robot came from and the angle it is going. For example, if the robot is in the center cell of figure 5.2, then moves diagonally like in Figure 5.3b, then the distance between the center of the current cell and the next cell is $\frac{\sqrt{2}}{2}$ times what it would be if the robot went straight like in Figure 5.3a.

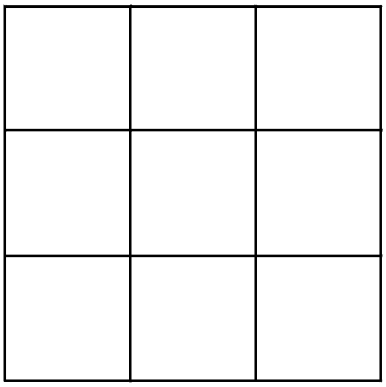
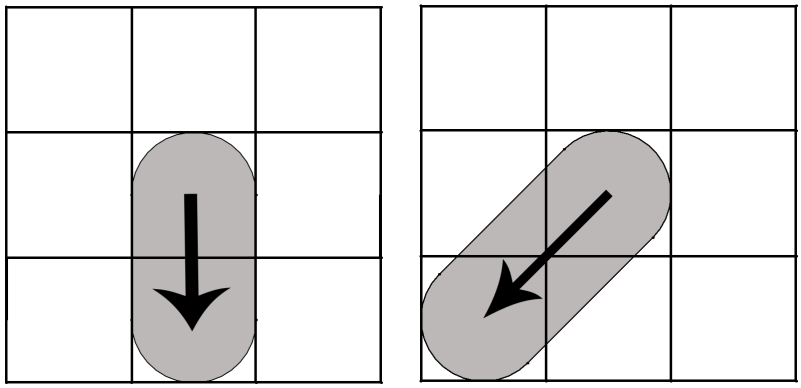


Figure 5.2: Square Flower



(a) Straight Movement **(b)** Diagonal Movement

Figure 5.3: Single motion in a square flower

For the circle packed representation there are six adjacent cells surrounding the one the robot is currently in (Figure 5.4); however, again unless this is the very first cell, then the robot just arrived from one of the cells, reducing the number of options to five: straight ahead, 60° to the left or right, and 120° to the left or right. The distance between the center of the current cell and the next cell is always the same.

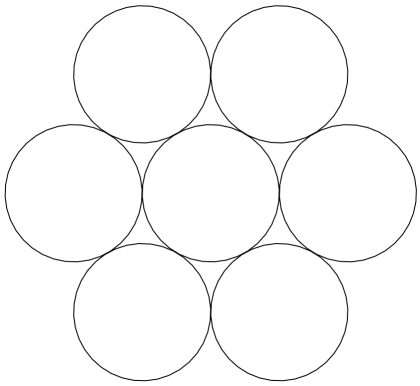


Figure 5.4: Circle Flower

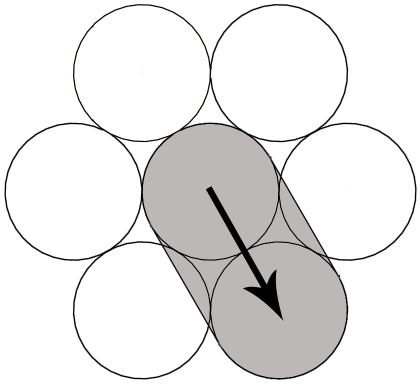


Figure 5.5: Single motion in circle flower

The two splines, the Cubic Hermite and Cubic Bezier, that meet the criteria in Section 4.4.5 are used to determine how the properties of the circle packed representation compare to the traditional square grid representation. Using the derivatives of the Cubic Hermite and Cubic Bezier spline shown in Sections 4.4.5.1 and 4.4.5.2 with Equation 4.4 lets one calculate the 2 dimensional curvature, κ, for a specific point on each spline. The average curvature, $\bar{\kappa}$, is calculated by selecting 1000 points with a uniform interval on the spline and averaging the absolute value of the curvatures calculated from those points:

$$\bar{\kappa} = \sum_{i=0}^{1000} |\kappa(\frac{i}{1000})|. \tag{5.1}$$

The absolute value is used so that opposite signed curvatures do not cancel each other out.

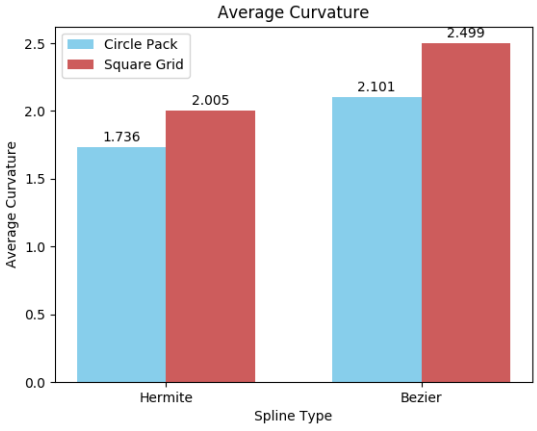


Figure 5.6: Spline Average Curvature

Of the two spline types that meet the path constraints required, the Cubic Hermite spline has less curvature as shown in Figure 5.6. This means the robot needs to turn less in order to follow this path. It is also simpler as the Bezier Spline requires tuning two of the control points; whereas, all the control points for the Cubic Hermite are built into the path. For these reasons, the path planning and coverage experiments both use a Cubic Hermite Spline to represent the path.

It is also shown that in making a single movement from a cell to an adjacent cell, the circle packed representation has properties that cause less curvature than the square grid representation for both sets of splines. This should lead to less curvature for the entire path on average.

In Figure 5.7 three of the five move options for the circle packed representation are displayed. The remaining two are simply mirrors of the 60° and 120° options. The footprint images show that the spline will still cover the gap between the circles.

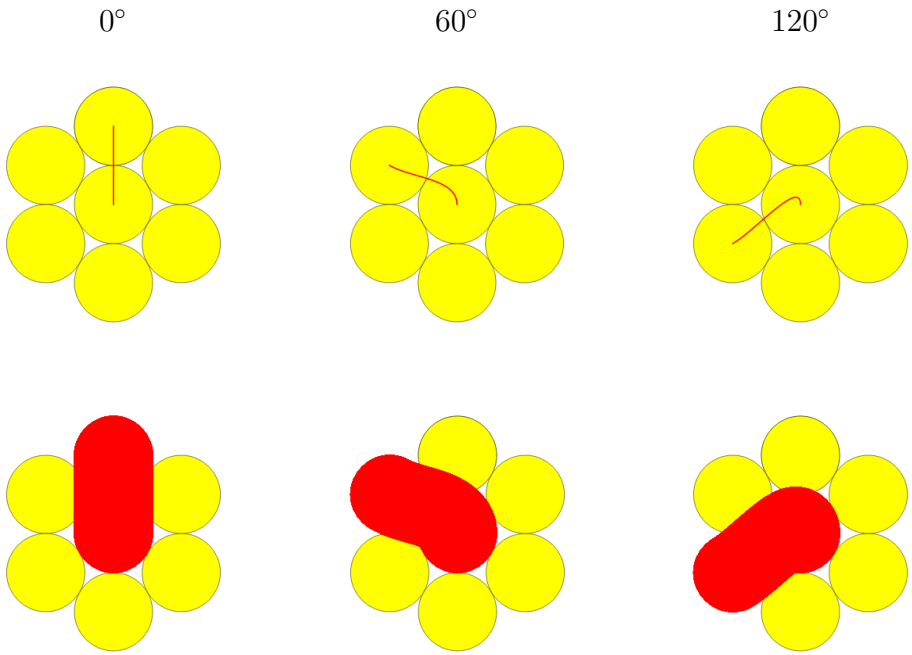


Figure 5.7: Circle Pack Move Options (Top: Path; Bottom: Footprint)

Similarly, Figure 5.8 shows four of the seven move options for the square grid. The remaining three are simply mirrors of the 45°, 90°, and 135° options.

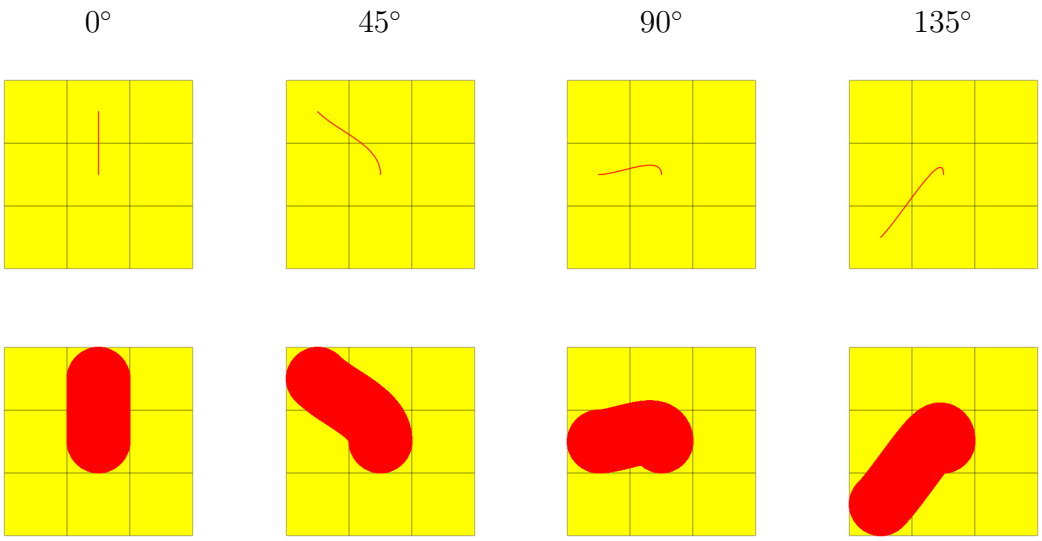


Figure 5.8: Square Grid Move Options (Top: Path; Bottom: Footprint)

5.2 Experiments

The process described in the Approach is used to validate the advantages of the properties of the circle packed space representation for both path planning and coverage. This process involves selecting six environments that vary in complexity. Each of these environments is then decomposed through circle packing and through a square grid. The Wavefront Algorithm is used to plan 1000 paths for each environment. The output of this algorithm is a series of points. The Cubic Hermite spline is used to connect each point with the next in the series. These paths are used to compare the properties of paths planned on the circle pack and the square grid.

A similar experiment is used for comparing the properties of coverage paths except that the Chinese Postman Algorithm is used.

The six environments of varying complexities selected are shown in Figure 5.9. The complexities were calculated using Algorithm 1 to average the number of intersections through 1000 1D cross sections.

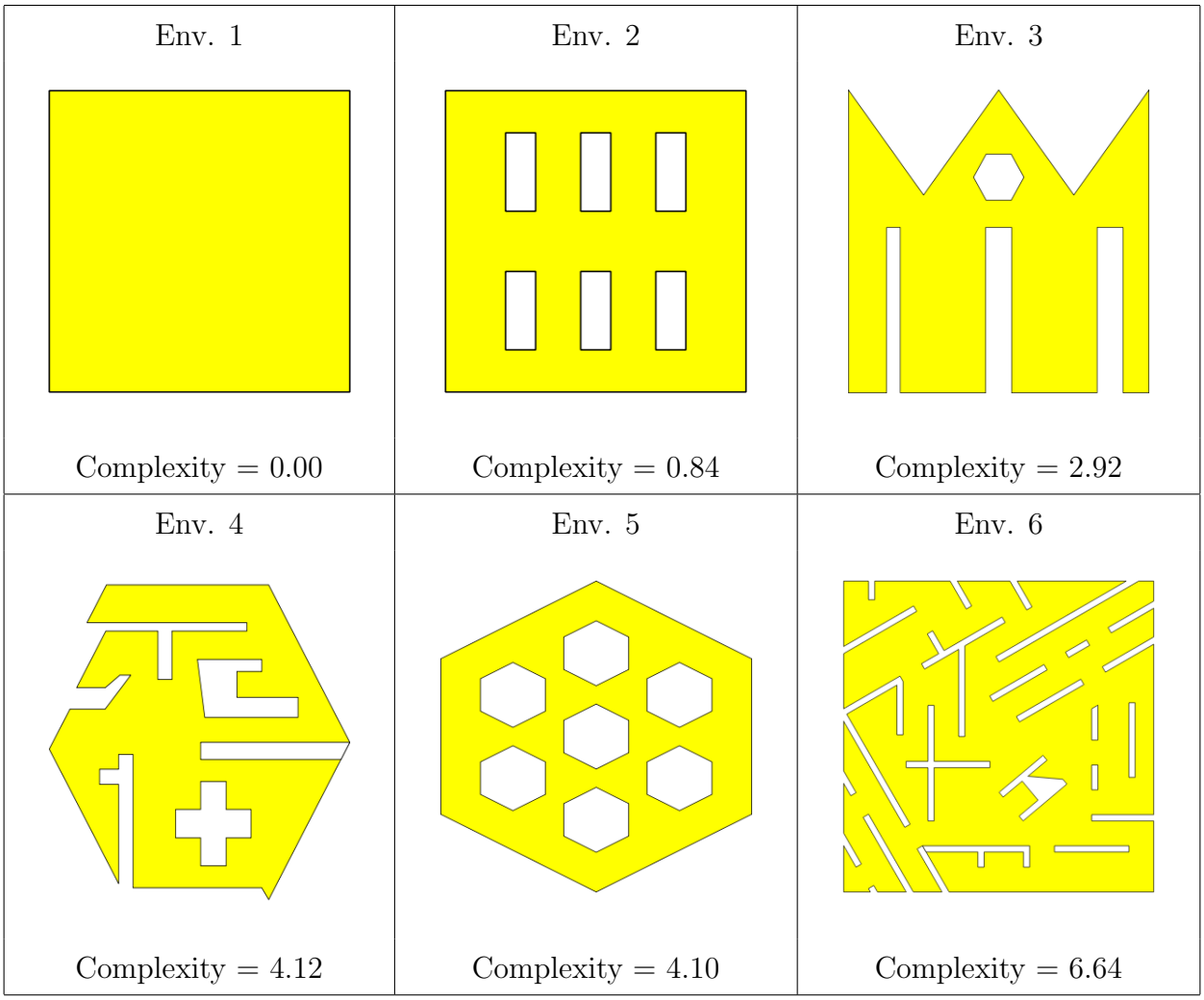
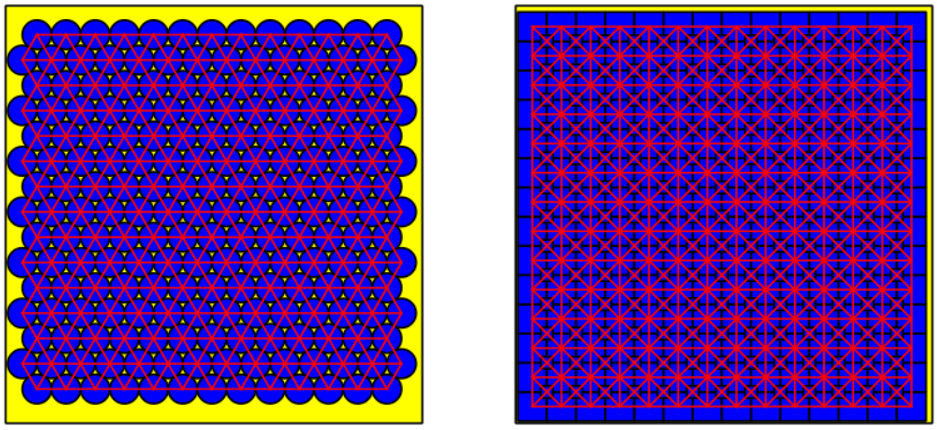


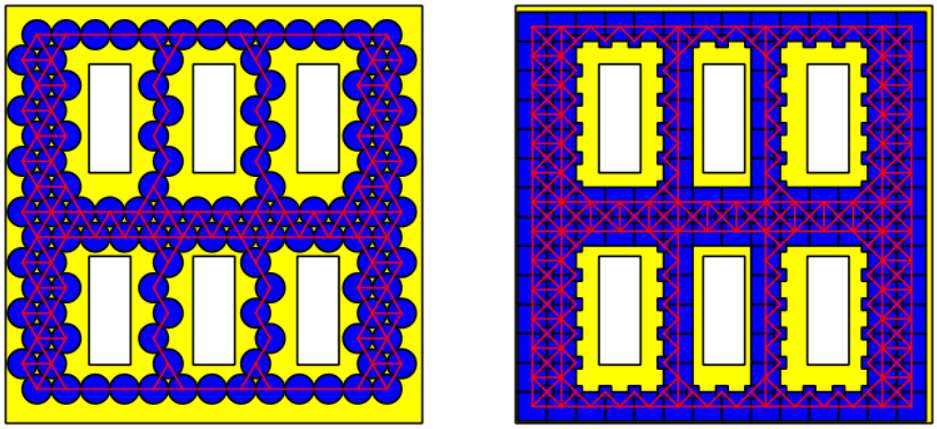
Figure 5.9: Experiment Environments

A circle packing and square grid was applied to each environment in order to decompose them into the respective representations (Figures 5.10). Then the boundary edges from Algorithm 2 were added.

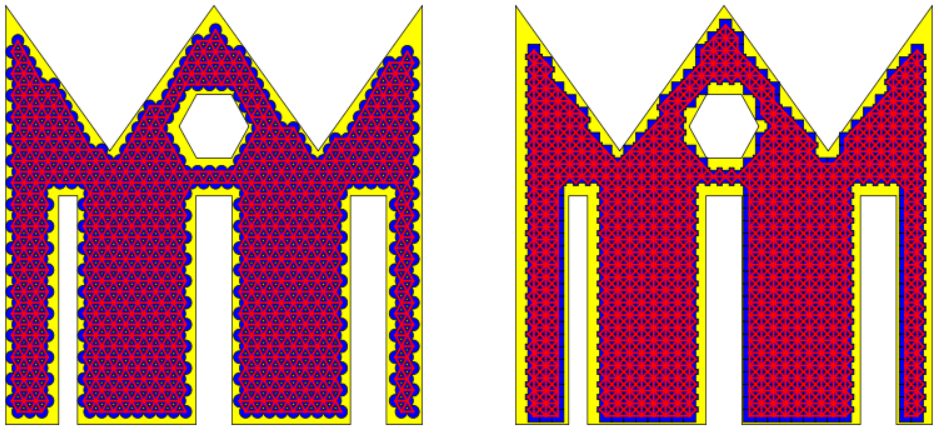
Figure 5.10: Packings (Left: Circle Packed; Right: Square Grid)



(c) Env. 1

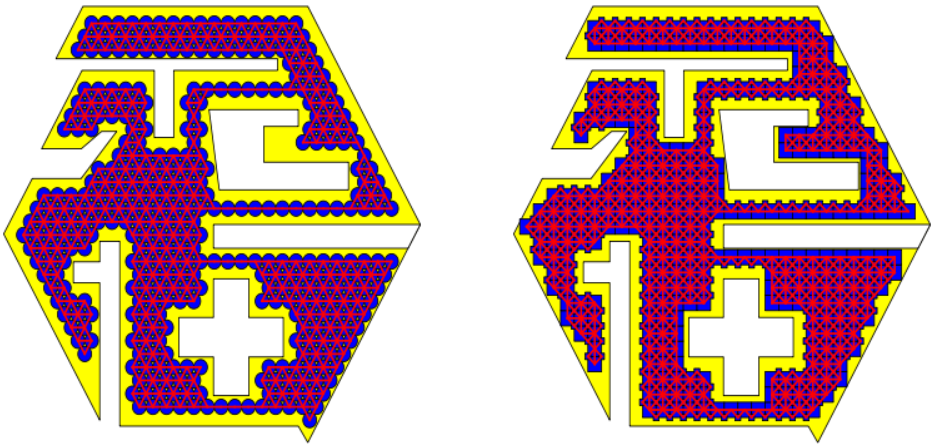


(d) Env. 2

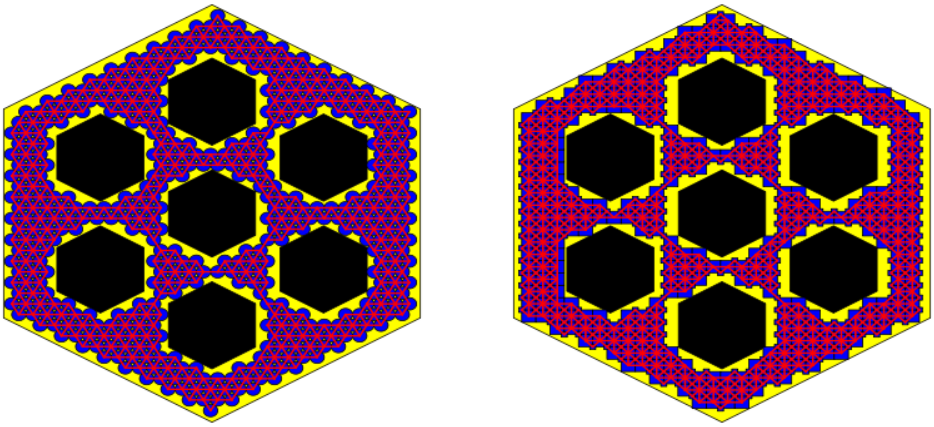


(e) Env. 3

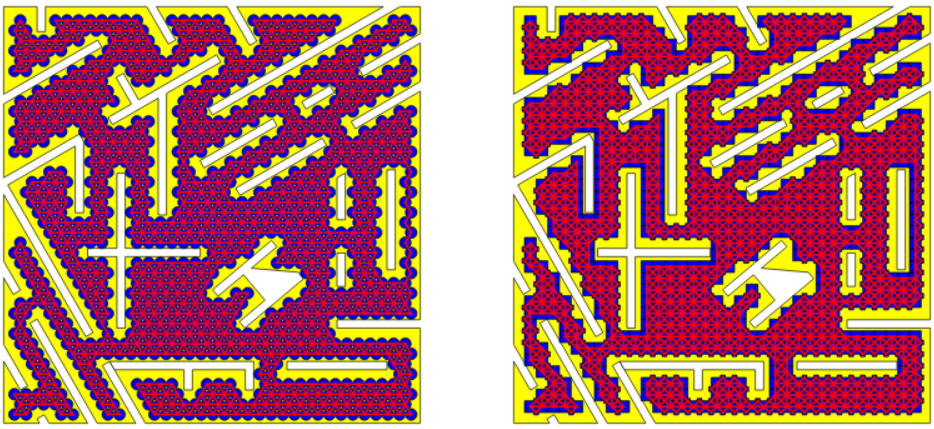
Figure 5.10: Continued



(f) Env. 4



(g) Env. 5



(h) Env. 6

Figure 5.10: Continued

The boundary edges are shown in Figure 5.11 to have reduced the number of odd vertices in the graph. The fewer odd vertices in the graph means that fewer circles are crossed multiple times. Reducing the amount of area that is crossed multiple times reduces the length of the path.

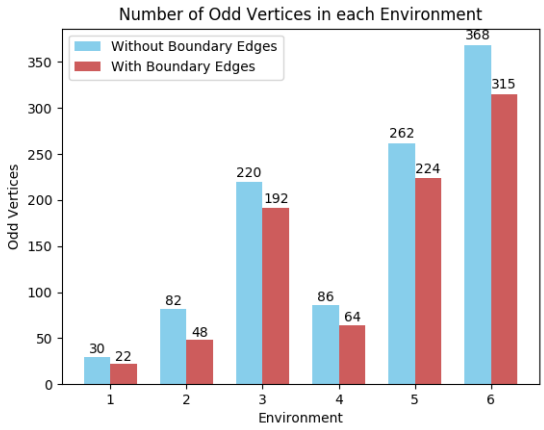


Figure 5.11: Boundary Edges - Odd Vertices

Asymptotically, they both have the same complexity ($O(n^2)$) and memory ($O(E+V)$). There is a constant time difference; however, that makes the amount of memory used in practice different. In each of the environments there were more cells on the circle packed version than from the square grid (Figure 5.12). This makes sense as the squares have more area than the circles, so more circles can be packed in. The square grid on the other hand with its more adjacent cells for each interior cell has more edges (Figure 5.12). The difference in the number of edges is larger than the difference in the number of cells. This makes the square grid take more memory in practice.

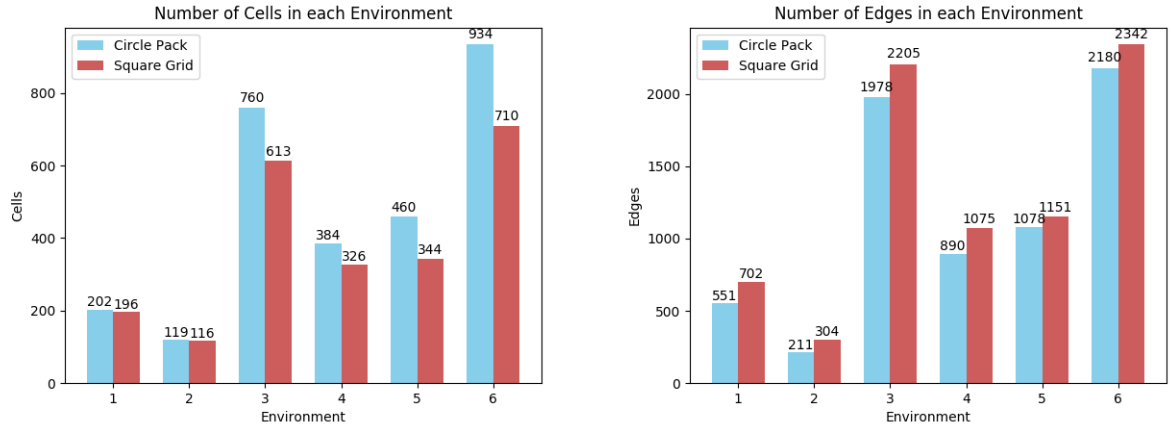
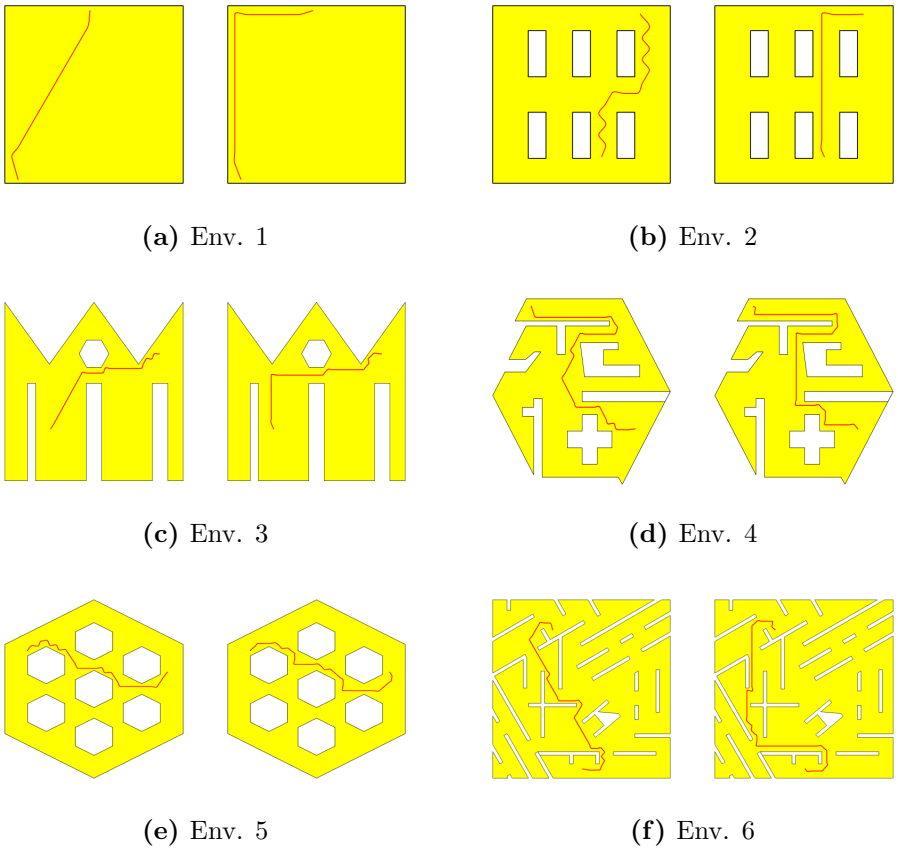


Figure 5.12: Number of cells and edges (Note: Different scales on on the y axis)

5.2.1 Wavefront

The graphs generated by the circle packing and the square grid have the wavefront algorithm applied to them to plan a path between a random start and a random goal point as explained in Section 4.3.1. Figure 5.13 shows an example from each of the environments.



(a) Env. 1 **(b)** Env. 2

(c) Env. 3 **(d)** Env. 4

(e) Env. 5 **(f)** Env. 6

Figure 5.13: Example Wavefront Runs (Left: circle packed; Right: square grid)

In each of the six environments most of the paths generated on the circle packed representation were shorter and have less curvature than their square grid counter-parts (Figure 5.14). Of the 6000 runs over the six environments, the path planned on the circle packed representation was shorter in 76.4 % of them. It also had less curvature in 72.7 % of them.

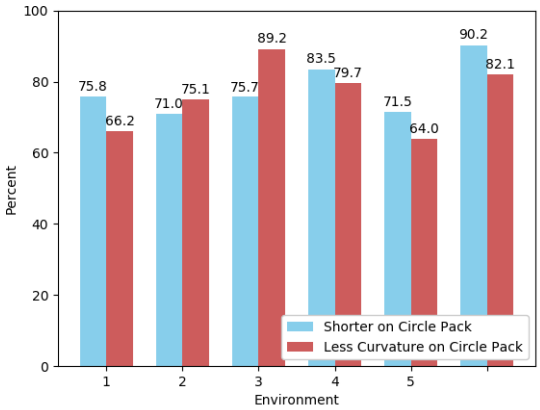


Figure 5.14: Circle Packed Wavefront Stats

On average the square grid paths were 5% longer and had 29% more curvature than their circle packed counter-parts (Figure 5.15).

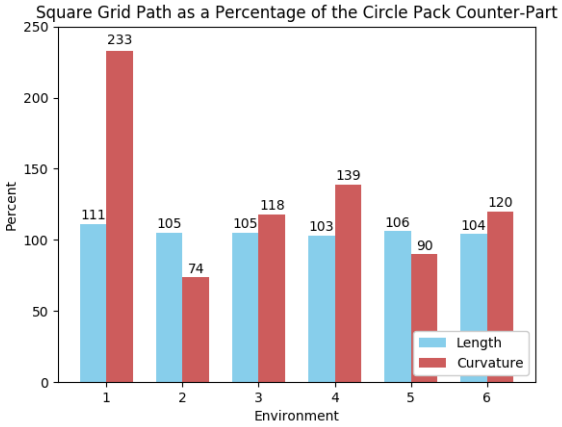
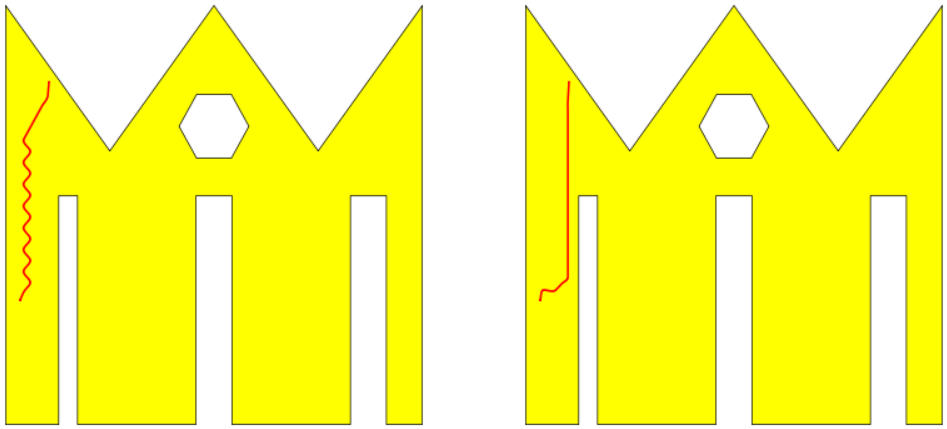


Figure 5.15: Square Grid Paths as Percentage of Circle Pack Paths

The paths planned on the circle pack wrapped tighter around obstacles and tended to create a more direct path to the goal. This led to them being shorter and having less curvature than their square grid counter parts. Of the paths on the circle packed representation that were longer many of them were mostly vertical paths like in Figure 5.16. For the square grid these paths are simple whereas the circle packed representation cannot create a vertical straight path. The zigzag that the circle pack causes ends up with more curvature than the straight path that can be generated for the square grid. The runs that were mainly horizontal did not provide this same result which leads to the belief that if the

tiling pattern were rotated by 90° then the direct vertical route can be created from the circle pack. Also some of the runs resulted in the path planned on the circle packed representation and the path planned on the square grid representation to be completely different routes.



(a) Circle Pack **(b)** Square Grid

Figure 5.16: Example Vertical Wavefront Run

Paths generated on the square grid had less curvature on average in environments 2 and 5. Environment 2 was selected as an environment that is more rectilinear with narrow hallway like areas. These type of environments are better for the square grid as there are lots of vertical narrow areas that would cause the vertical zigzag. Also the rectilinear nature of the environment leads to lots of 90° turns which better suits the square grid. Environment 5 also had many paths with the vertical zigzag causing the same effect with curvature.

5.2.2 Coverage using Chinese Postman Algorithm

The modified Chinese Postman discussed in Section 4.3.2 was run with the different environments. In each case both the circle packed and square grid representations were used and the total arc length and average curvature were recorded. An example from Environment 5 is shown in Figure 5.17.

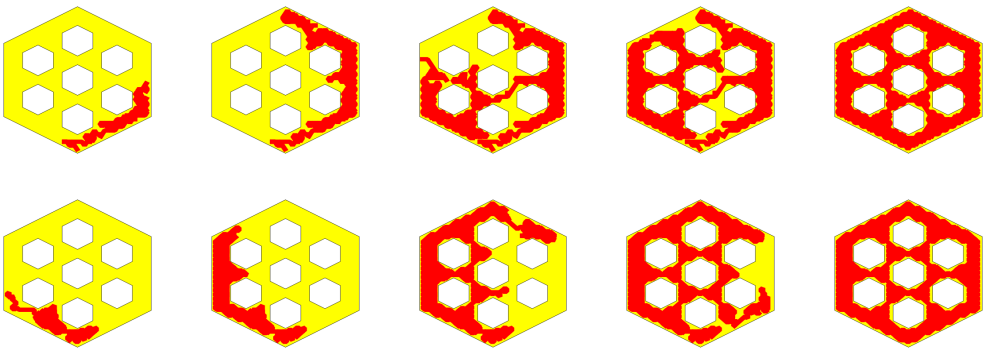


Figure 5.17: Chinese Postman Footprint (Top: Circle Pack, Bottom: Square Grid)

Figure 5.18: Chinese Postman

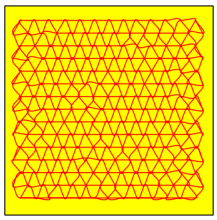
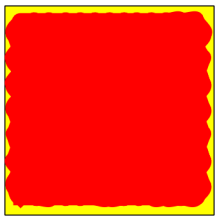
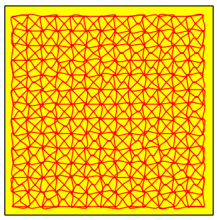
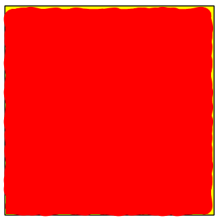
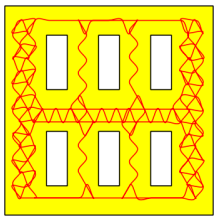
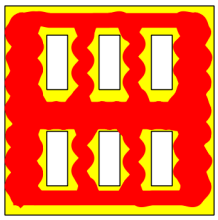
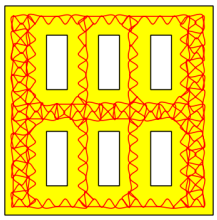
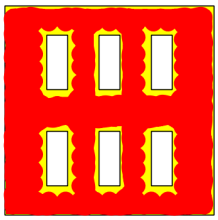
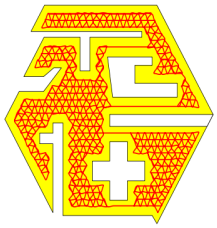
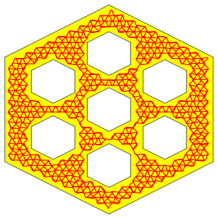
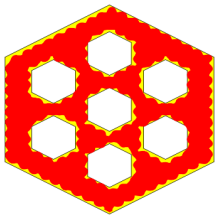
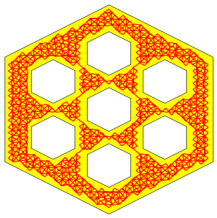
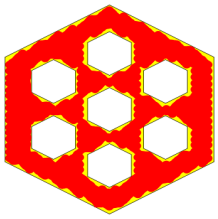
	Circle Pack		Square Grid	
Env.	Path	Footprint	Path	Footprint
1				
2				
3				
4				
5				

Figure 5.19: Continued

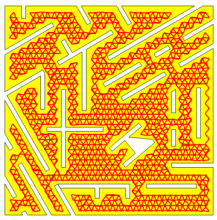
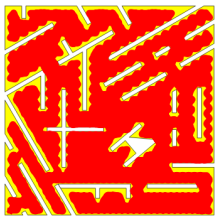
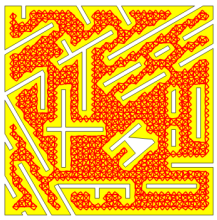
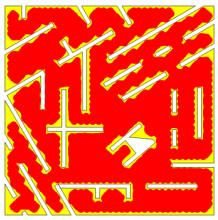
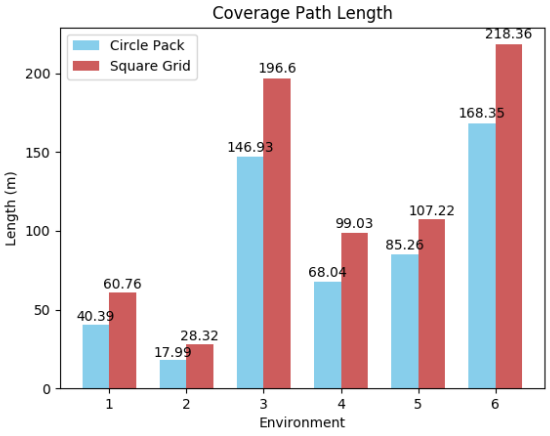
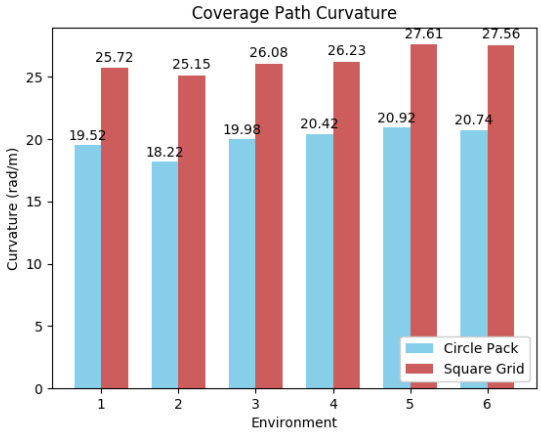
	Circle Pack		Square Grid	
Env.	Path	Footprint	Path	Footprint
6				

Figure 5.19: Continued

Each of the square grid coverage paths were 40.4% longer on average than their circle packed counterpart as shown in Figure 5.20c. This is expected due to the extra edges in the graph. The square grid paths also had a much higher average curvature as there was 32.3% increase in curvature from the circle packed paths as shown in Figure 5.20d.



(c) Coverage Path Lengths



(d) Coverage Path Curvatures

Figure 5.20: Coverage Path Stats

Each of the coverage paths for the square grid was longer than the coverage path for the circle pack on each environment. In order to determine that this was not caused by differences in the amount of area represented, the area of each representation was calculated. In Figure 5.21 it is shown that some of the environments had more area represented by the square grid (Environments 1, 2, and 4); however, there was a larger difference in extra length

from the coverage paths on the square grid than in the amount of area represented. It is expected that the relationship between the area to be covered and the path to cover it would be linear, so this suggests that the paths for circle pack would be shorter for the same area.

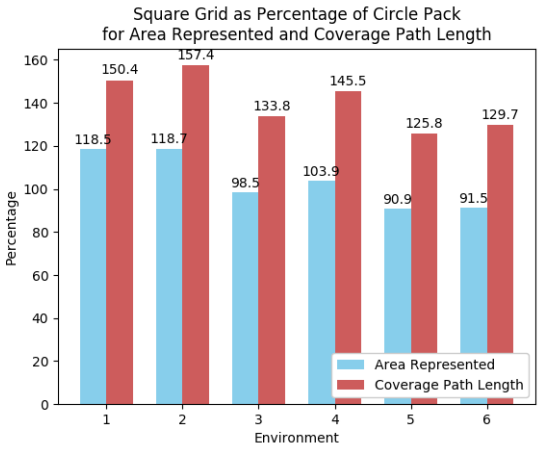


Figure 5.21: Area vs Length

5.3 Summary

The circle packed representation was compared to the common square grid representation at both the low and high level. First, this was done by analyzing the curvature of a single move on average in which it was shown that the circle packed representation causes less curvature. Then the amount of memory each representation took up on environments with varied shape and complexity was tested and showed that the circle packed representation uses less memory in each of these cases. The wavefront algorithm was used in order to plan paths on both representations. On each of the six environments, most of the paths planned on the circle packed representation were shorter and had less curvature than their square grid counter-parts. Using the Chinese Postman Algorithm, it was shown that all of the planned coverage paths were shorter and had less curvature on the circle packed representation than their square grid counterparts.

Chapter 6

Conclusion

In this thesis, a method for decomposing a space into smaller pieces was demonstrated. This method tiles the space with tangent circles in a uniformal hexagonal pattern. These circle are then converted into a planar graph which represents the space. This representation is then analyzed for how well it works for path planning and coverage on an individual move level. The representation was then compared against the common square grid approach on several different environments. These environments varied in shape and the number of obstacles. This comparison determined that for path planning for many environments the circle packed representation causes the paths planned to be shorter, more direct, and have less curvature. The representation was then also tested on its ability to be used in covering the space in each of the environments using a modified version of the Chinese Postman Algorithm. The paths generated from the circle packed representation were shorter and had less curvature than their square grid counterparts. These results validated that this is a useful method for decomposition and representation for path planning and coverage.

6.1 Limitations

While this method has has been shown to have various advantages, a number of limitations to this proposed method should be discussed. As with other approximate decomposition methods, the circle packing representation has an exponential growth in memory with larger spaces. Also, this representation is resolution complete meaning the representation can be

completely covered, but that the representation will not always cover 100% of the space. Currently this method is an offline method so it can only be used in a previously known environment. Certain environments that are more rectilinear in nature may cause worse performance out of the circle packed representation than the square grid representation.

6.2 Contributions

The contributions of this thesis are the discoveries of the advantages, over other decomposition methods, of using circle packing to decompose a target space into a graph for both classical path planning and coverage. The representation created by this decomposition uses less memory than traditional decomposition methods in practice and allows for shorter paths with less curvature to be created in most situations. These statements hold true for both planning a path between two points in an environment and planning a path to cover the entire environment.

6.3 Future Work

There are several ways to improve this method. For example, a step that determines the best size for the circle can be added. Currently the circle size is based on the robot size, but the planned path and the percentage of the space that can be covered varies pretty heavily based on the size of the circle. Smaller circles use more memory as more are needed, but will have a finer resolution and thus a higher percentage of the space will be represented. Another way to improve the method is to add a step that determines the translation and orientation that gives the packing that provides the best path or coverage. Depending on where the first circle is and how the rest are laid out the space space can have vastly different packings with different numbers of circles, different percentages of the space represented, and different paths that can be generated from it.

There are also several directions future research can take to improve the state of the art. More path planning algorithms can be tested on this representation to determine if certain ones perform better on it. For this work the robot was described to use be non-holonomic

and use a differential drive. This representation could be tested with other drive styles such as a holonomic vehicle or one with Ackermann steering. Currently this method is an offline planning algorithm, but an online version could be implemented that takes newly discovered areas and adds them to the graph by tiling and then converting them. This method can be expanded for 3D spaces by using sphere packing to convert a 3D space into a graph. While more complicated, this would be useful for drone research indoors.

Bibliography

[1] Acar, E. U., Choset, H., Rizzi, A. A., Atkar, P. N., and Hull, D. (2002). Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21(4):331–344. 16

[2] Bowers, P. and Stephenson, K. (1997). A regular pentagonal tiling of the plane. *Conformal Geometry and Dynamics of the American Mathematical Society*, 1(5):58–86. 19

[3] Choset, H. (2001). Coverage for robotics-a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1):113–126. 12, 13, 17

[4] Choset, H. and Pignon, P. (1998). Coverage path planning: The boustrophedon cellular decomposition. In *Field and Service Robotics*, pages 203–209. Springer. 14, 15

[5] Collins, C. R. and Stephenson, K. (2003). A circle packing algorithm. *Computational Geometry*, 25(3):233–256. 19

[6] De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O. C. (2000). Computational geometry. In *Computational Geometry*, pages 1–17. Springer. 5

[7] Dreifus, G., Goodrick, K., Giles, S., Patel, M., Foster, R. M., Williams, C., Lindahl, J., Post, B., Roschli, A., Love, L., et al. (2017a). Path optimization along lattices in additive manufacturing using the chinese postman problem. *3D Printing and Additive Manufacturing*, 4(2):98–104. 19

[8] Dreifus, G., Rapone, B., Bowers, J., Chen, X., Hart, A. J., and Krishnamoorthy, B. (2017b). A framework for tool path optimization in fused filament fabrication. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*, page 9. 19

[9] Eiselt, H. A., Gendreau, M., and Laporte, G. (1995). Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43(2):231–242. 7, 8, 9, 28

[10] Elfes, A. (1987). Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265. 17

[11] Fejes, L. (1942). Über die dichteste kugellagerung. *Mathematische Zeitschrift*, 48(1):676–684. 23

[12] Galceran, E. and Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276. 13, 18, 28

[13] He, Z.-X. and Rodin, B. (1993). Convergence of circle packings of finite valence to riemann mappings. *Comm. in Analysis and Geometry*, 1:31–41. 19

[14] Hert, S., Tiwari, S., and Lumelsky, V. (1996). A terrain-covering algorithm for an auv. *Autonomous Robots*, 3(2):91–119. 16

[15] Hoffman, K. L., Padberg, M., and Rinaldi, G. (2013). Traveling salesman problem. In *Encyclopedia of operations research and management science*, pages 1573–1578. Springer. 8

[16] Huang, W. H. (2001). Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 27–32. 18

[17] Kim, S. (2017). An integrated design approach for infill patterning of fused deposition modeling and its application to an airfoil. *Society for the Advancement of Material and Process Engineering*. 20

[18] Lumelsky, V. J., Mukhopadhyay, S., and Sun, K. (1990). Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation*, 6(4):462–472. 14

[19] Mannadiar, R. and Rekleitis, I. (2010). Optimal coverage of a known arbitrary environment. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 5525–5530. 18

[20] Mohar, B. (1993). A polynomial time circle packing algorithm. *Discrete Mathematics*, 117(1-3):257–263. 19

[21] Oh, J. S., Choi, Y. H., Park, J. B., and Zheng, Y. F. (2004). Complete coverage navigation of cleaning robots using triangular-cell-based map. *IEEE Transactions on Industrial Electronics*, 51(3):718–726. 18

[22] Schumaker, L. (2007). *Spline Functions: Basic Theory*. Cambridge University Press. 9

[23] Sprunk, C. (2008). Planning motion trajectories for mobile robots using splines. Unpublished. http://www2.informatik.uni-freiburg.de/ lau/students/Sprunk2008.pdf. Accessed on August 2017. 30

[24] Thurston, W. P. (1979). *The geometry and topology of three-manifolds*. Princeton University Princeton, NJ. 19

[25] West, D. B. et al. (2001). *Introduction to Graph Theory*, volume 2. Prentice hall Upper Saddle River. 7

Vita

Andrew Messing was born in Knoxville, TN. In 2016, he completed his Bachelor's degree in Computer Engineering at the University of Tennessee. He then proceeded into the Master's program where he worked in the Distributed Intelligence Laboratory studying Robot Path Planning. He also worked for the Manufacturing Demonstration Facility at Oak Ridge National Laboratory developing path planning software for the world's largest 3D printers. He will be graduating with his Master's in May 2018.