

# **ABSTRACT**

Title of Document:                   DISTRIBUTED MAZE SOLVING BY  
COOPERATIVE ROBOTIC PLATFORMS

Brian William Beisel  
Master of Science, 2014

Directed By:                       Professor Gilmer Blankenship  
Department of Electrical and Computer Engineering

Problem solving based on sensor created area maps is a challenging problem that can benefit from a multi-robot approach. Cooperative problems are most eloquently designed through distributed services and systems. This thesis designs and implements a full distributed maze solving solution using simulated robotic sensor platforms. A distributed spatial communication system was developed and tested as a contributing element of the maze solving solution. Autonomous algorithms for communication, cooperation, and navigation were constructed and tested through simulation in maze solving tests. Working with an assumed map creating technology in tandem with the aforementioned developed technologies resulted in an effective complete solution. Although a great deal of future work is recommended to address imperfect mapping complications, it was found through simulation and mathematical analysis that multiple cooperative robotic platforms can result in significant performance improvements.

# **DISTRIBUTED MAZE SOLVING BY COOPERATIVE ROBOTIC PLATFORMS**

By

Brian William Beisel

Thesis submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park, in partial fulfillment  
Of the requirements for the degree of  
Master of Science  
2014

Advisory Committee:

Professor Gilmer Blankenship, Chair  
Professor Christopher Davis  
Associate Professor Nuno Martins

© Copyright by  
Brian William Beisel  
2014

## **ACKNOWLEDGEMENTS**

I would like to thank the many people who helped during academic career. First I would like to thank Professor Gilmer Blankenship who provided me with ideas, tools, and connections that I could not have gotten through these years without. At the end of my sophomore year of undergraduate education, Professor Blankenship sent me to TRX Systems where I found a home among some very wonderful people. The time I spent there was filled with learning, curiosity, and fun. I express my deepest gratitude to everyone at TRX System for their time and effort in nurturing a developing mind. I would also like to thank my parents who supported, encouraged, and listened to me whenever I needed it. I would not have gotten this far without them. Finally I would like to thank the faculty and staff at the University of Maryland that helped and inspired me to learn and succeed. To my friend and advisor Jermaine Jackson, you will not be forgotten and you will forever be missed.

# TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	ii
LIST OF TABLES.....	v
LIST OF FIGURES.....	vi
1 INTRODUCTION.....	1
1.1 Research Context.....	1
1.2 Technology Overview.....	1
1.2.1 Map Building.....	1
1.2.2 Global Position Estimation.....	3
1.2.3 Distributed Spatial Database.....	4
1.2.4 Autonomous Functionality.....	5
1.3 Project Overview.....	7
2 DISTRIBUTED SPATIAL QUERY.....	9
2.1 Introduction.....	9
2.2 Design.....	9
2.2.1 Coverage Polygon.....	10
2.2.2 Spatial Coverage Update.....	12
2.2.3 Query Coverage.....	13
2.2.4 Message Passing.....	15
2.3 Query Simulation Results.....	17
2.3.1 Message Passing and Query Time.....	17
2.3.2 Environment Data.....	20
3 DISTRIBUTED AUTONOMOUS SYSTEM.....	22
3.1 Introduction.....	22
3.2 Minimum Exit Path Calculation.....	22
3.2.1 Visibility Polygon.....	23
3.2.2 Vector Collection Breadth First Search.....	24
3.2.3 Edge Weight Calculation.....	26
3.2.4 Navigation Point Calculation.....	27
3.2.5 Exit Plan Stabilization.....	28
3.3 Autonomous Query Creation.....	30
3.4 Optimal System Path Selection.....	34
3.5 Platform Navigation Controls.....	37
4 MAZE SOLVING SIMULATION.....	41
4.1 Goal.....	41
4.2 Description.....	41
4.3 User Interface.....	42
4.4 Results.....	44
5 EXPECTED ESCAPE COST/TIME CALCULATION.....	50
5.1 Introduction.....	50
5.2 Problem Space.....	50
5.2.1 Non-Brute Force Problem Space.....	52
5.3 Process Example Walkthrough.....	54
5.4 Single Robot Calculation.....	56
5.4.1 Expected Edge Estimation.....	57
5.4.2 Maze Complexity Coefficient.....	60

5.5	Multiple Robot Complications.....	62
5.5.1	Query Success Probability.....	63
5.5.2	Query Attempt Expected Cost.....	65
5.5.3	Physical Exit Cost.....	67
5.6	Results.....	67
5.6.1	Maze Size vs. Number of Robots.....	68
5.6.2	Communication Cost vs. Number of Robots.....	71
6	CONCLUSIONS AND RECOMMENDATIONS .....	76
	REFERENCES.....	79

## LIST OF TABLES

Table 1 – Progression of a distributed spatial query.....	15
Table 2 – Exit distances of example 2 test.....	46
Table 3 – Exit distances and distance averages of robot number varied tests.....	48
Table 4 – Robot exit distances and averages of tree maze tests.....	49
Table 5 – Characteristics of exit boarder cell types.....	54
Table 6 – Best and worst case results of the expected value calculation on a 2x2 grid.....	56
Table 7 – Best and worst case expected values of a 2x2 grid using the per step expected number of edges and exits for exit probability estimation.....	60
Table 8 – Minimum exit times and the ranges of robots that are able to achieve those times.....	70

## LIST OF FIGURES

Figure 1 – Collector platform (Blue) with for proximity sensors (black) and their visibility polygons (yellow).....	11
Figure 2 – Expansion of visibility polygon (blue) through rotated union (green) then polygon buffer (yellow) to create the final coverage polygon.....	12
Figure 3 – Coverage polygons of same quality can be merged (blue). Overlapping sections must take the higher quality value (green).....	13
Figure 4 – The polygon relational logic for spatial coverage update. Database update transactions are omitted for simplicity.....	14
Figure 5 – (a) Collector Distribution of message passing test (b) Final query results showing updated coverage database of center green collector.....	18
Figure 6 – (a) The state of the system at the start of a query (originating from red node). The query region is shown as a red circle. The steel blue lines indicate environment wall data that is not currently in view of any node. (b) The final state of the system after the completion of the previous query. All red lines indicate the environment data known to the originating node. The gray regions indicate the red node's coverage database contents. All other colored lines are the immediate view of the remaining system nodes.....	21
Figure 7 – Examples of two visibility polygons originating from the green circles..	24
Figure 8 – Red lines connect visibility points that form a non-minimal path. Green circles represent waypoints for a minimum distance path. The black line shows the possible robot executed path using the minimum waypoints...	26
Figure 9 – The red lines connect visibility points that are the origin of the visibility polygons. The green circles are the navigation point locations calculated using the wall vertices (black dots) and the two visibility points connected by the red line intersected by the wall vertex.....	28
Figure 10 – The red lines and paired with a green circle show the two possible exit paths. The green circle centered in the gold diamond is the exit gate that results from combining the two redundant exit path options.....	30
Figure 11 – The robot (solid purple rectangle) located all exit gates (gold diamonds) using its partial map of the environment (purple walls inside of its coverage polygon (gray region)). Query regions (purple unfilled polygons) were calculated using the separate exit gates and the visibility polygons from those gates (blue regions).....	32
Figure 12 – Steps through sequential queries by a fresh robot to a system with full knowledge of the maze. The query regions are created using the exit gates of each step and when data is returned (regions of data are indicated by translucent green polygons) new exit gates are created from the query updated map.....	33
Figure 13 – An example of how each robot settles on which route to take based on the route decisions and locations of other robots around it.....	34
Figure 14 – Example of distributed optimal path selection. Each robot can see all possible surrounding route options. The distributed path selection service ensured that	



each robot picked a different route which is the most efficient way to explore the maze.....	36
Figure 15 – Example of starting system making sequential decisions divide and conquer. The result is that all possible routes were explored as efficiently as possible.....	37
Figure 16 – Visual example of actual detected environment features verses the assumption that the control algorithm must make based on that detection..	38
Figure 17 – The red circle describes the minimum turning radius of the robot’s most exterior point that will prevent a collision with the worst case obstacle detected by the robot’s front sensor. The blue circle shows the robot’s interior turning radius.....	40
Figure 18 – Initial solution path (purple) Query improved solution path (red).....	45
Figure 19 – Results of a four robot solution origination from the center of a ‘plus sign’ maze.....	46
Figure 20 – The solutions of a ‘plus sign’ maze with bottom branch exit using 1, 2, and then 3 robots.....	47
Figure 21 – Two tests run on a tree-like maze structure.....	49
Figure 22 – Discretized maze serves as representation of the problem environment used for mathematical analysis.....	51
Figure 23 – Progression of expected value calculation of a single robot in a 2x2 maze.....	55
Figure 24 – Results of number of edges expected value estimation equation 17 (black). Function lays in-between upper and lower bounds and trends to the lower bound dictated by the confining maze structure.....	58
Figure 25 – Escape probability estimate (blue) using edge and exit edge expected value approximations.....	59
Figure 26 – Worst (red) and best (blue) case expected exit cost/time values over a range of NxN maze grid sizes.....	60
Figure 27 – Relationship between maze size and complexity with respect to average and best case expected exit costs/times.....	62
Figure 28 – Expected cost of exploration options dictate which option is chosen. Logic contributes to expected cost/time calculation.....	66
Figure 29 – Time dependence on the number of robots used to solve a maze. Varies with grid size.....	68
Figure 30 – Exit time trend changes as more robots are added to the solution .....	71
Figure 31 – Dependence of expected exit time on a varied query cost with respect to the number of robots. Average movement exploration costs are assumed.....	72
Figure 32 – Dependence of expected exit time on a varied query cost with respect to the number of robots. Worst case movement exploration costs are assumed.....	72
Figure 33 – Reflection of the rate of change in efficiency with respect to query cost and the number of robots in a solution configuration.....	74
Figure 34 – Exponential increase in cost with respect to a linear change in time scale...	75

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 Research Context**

The generic focus of this project is to search an unknown region for a specific target as quickly and efficiently as possible. The searching is to be done with mobile robotic sensor platforms. The key idea is that the desired target could be found faster if more than one robot searched for it, faster still if participating agents worked together. The problem becomes, how to design a system that searches and collaborates autonomously and what is the most effective and efficient method of doing so? This Master's project attempts to solve this generic search problem and verify that its solution is valid. The generic task is made specific by assuming the unknown region to be a maze (area filled with complex obstacles) and taking the target to be an exit.

### **1.2 Technology Overview**

Simply exiting a maze is a trivial task as the exit is eventually reached by mindlessly following along the exterior wall. However, to exit a maze quickly and efficiently, several technologies and abilities are required.

#### **1.2.1 Map Building**

Accurate maps are essential in most areas of research involving navigation planning through obstacle filled environments [2] [9] [15] [16] [17] [18], and the solution developed in this project is similarly reliant. However, a robot starts without knowledge

of the surrounding environment; therefore, a map of the maze must be built as the solution progresses.

The field of sensor based area mapping is of high interest in both academia and industry. The objective is to explore and describe an unknown region or environment through a sensor/actuator platform. All sensor data is directly related to the position of the platform and thus a location estimate for the platform must be calculated and linked to the collected sensor data. The sensor data can then be converted to spatial data using its position estimate, which is then added to a map. A popular technique for area mapping is simultaneous location and mapping which uses sensor data for both location estimation and map building [12]. Other methods employ disjoint sensor sets which estimate location and build a map separately [14]. The reason that robotic mapping has been such a large topic of research is because of its high degree of difficulty [14]. It may be many years before mapping becomes a solved problem.

Environment data collection is an integral part of map building. Any object that inhibits platform movement can be considered as valid map data. Typically, only stationary obstacles are desired to describe a region; thus, data collected from obstacles in motion should be filtered out. Sensor noise and outlier filtering techniques such as those found in the Robust Kalman Filter have been found to improve map building performance in the presence of erroneous observations [11]. Common sensors used for obstacle data collection are ultrasonic sensors and LIDAR, the latter being the vastly more accurate and expensive option [10][13].

Cooperative map building is a key technology in multiple robot mapping solutions and requires the ability to merge data from multiple sources. Existing research has

created effective methods of map data merging through the use of occupancy grids and more advanced techniques [2][5]. The accuracy results are directly correlated to the accuracy of the maps prior to merging.

### **1.2.2 Global Position Estimation**

Any method of map building relies heavily on the location of the sensors collecting environment data. This is because any data collected must know its location relative to all other data in order for it to be correctly placed in the map. Multiple data collector systems must share a common location reference point or have a reference point in a coordinate system common to all participating collectors. It is required in a mapping application for shared spatial data to be useful to those it's shared.

Dead reckoning is the most common method of position estimation calculation and can be used in cooperative mapping applications if the starting location is globally referenced. The previously stated method of SLAM is one technology that produces an estimate position in conjunction with map creation [12]. Other methods fuse global referencing updates such as GPS with local position estimates to create an improved global position estimate. Inertial sensors such as gyroscopes and accelerometers can be used to create change in position estimates that serves as input to a dead reckoning algorithm. Robotic tracking systems often use wheel encoders in addition to gyroscopes to provide much more accurate path scaled location solutions [2][10][11]. Like robotic area mapping, location estimation is a very hard problem and may not be solved for quite some time. Location estimation and mapping are very closely related, as the solution for one may provide the solution for the other.

### 1.2.3 Distributed Spatial Database

Mapping large or complex areas could be made much more feasible when multiple collectors are used. However, to do so efficiently requires communication of spatial data. Collectors may require map data that is owned by another collector in the system or is not yet known to any participating robot. If a robot is able to define a region that could contain data critical to its solution path, it would be beneficial to have the ability to ask other participants for data in that region. Providing this ability may seem trivial when thinking of centralized communication requests, but when communication is transported into a distributed environment, complications arise. The option of centralizing the location of all spatial data is made unavailable when communication limiting constraints are applied and unrealistic when the solution requires scaling to large numbers of physical contributors. Since both communications constraints and the requirement of scaling are assumed for this project, a centralized solution is not possible.

There are many existing technologies that realize a generic distributed database system, such as Spanner by Google or the open source database RethinkDB.

Unfortunately, these systems do not provide a method of querying distributed spatial data. There has been limited research on distributed spatial data query systems and what there has been relies on connectivity guaranteed communication networks [7].

Realistically, many regions that are desirable to explore and map using a cooperative system will not have access to a complete communication network such as cell data towers. Robots can be mounted with range limited communication devices such as routers, Wi-Fi direct cards, or other range limited communicators. Therefore, a robust non-centralized communication network comprised of range limited router hubs is a

requirement for an effective data sharing solution. Fortunately, mesh communication networks that fulfill these requirements have been previously developed and implemented [6]. Although existing mesh networks satisfy the project's direct communication requirements, the mesh network's connectivity constraints make previously developed distributed spatial communication systems useless. Other distributed mapping applications have simply shared data between robots that are in direct communication proximity from one another [2]. It becomes apparent that a distributed query system that takes into account the application at hand and proximity connectivity constraints could be extremely beneficial to the overall maze escape search solution.

#### **1.2.4 Autonomous Functionality**

Using the previously listed technologies every robot has the ability to record surrounding environment data in the form of a globally referenced map. The distributed spatial database enables the system to complete spatial queries given a region of interest. Additionally, data received by spatial query result can be effectively integrated into a robot's partial map. There are several steps yet left to take before a distributed maze solving solution can be realized. Map data queries must be autonomously created and executed in order to utilize a multiple platform solution and minimize the effort of solving the maze. Each robot must use their partial view of the maze to calculate possible navigation plans. Ideally, the paths that constitute an optimal exit solution will be selected. Finally, a controls algorithm must use the selected plans to physically guide the robots out of the maze.

There is quite a log of research involving optimal path planning in an obstacle filled environment. Research concerning multi-robot path planning has made assumptions that

do not align with this project. The most common assumption is that the environment is known [8] [4] [3]. Another assumption made is that of a centralized dataset and path calculation for robots exploring an unknown environment [2]. Since the desired solution is constrained to a distributed system in an unknown maze environment, existing path planning systems under conflicting constraints are insufficient. Distributed exploration attempting the completion of Voronoi partitions has been done and closely relates to the desired task. However, this method for optimal exploration does not work well in severely obstacle ridden environments such as mazes [1].

Multiple methods for single robot optimal path planning have been researched. Path planning on both vector and raster datasets has been accomplished with visibility graph calculation and flood fill algorithms, respectively [9] [2]. The map representation assumed in this project is vector based; therefore, the base method of shortest path calculation will utilize visibility polygons and relate to a visibility graph.

Intelligent spatial data communication with respect to exploration, problem solving, and optimal path calculation is a very specific topic and research concerning this area has not been found. This is most likely due to the fact that a spatial query system designed for the purpose of exploratory data sharing has not been integrated into optimal path planning technologies. This project's method of autonomous query creation relies on information derived from the optimal path calculation used.

### 1.3 Project Overview

Many of the technologies hinged upon by the solution presented in this project are largely imperfect. The focus of this research is to develop an effective distributed spatial communication system and create a distributed autonomous solution. These focal points must be verified for correctness and analyzed for performance metrics. In order to accomplish this, the technologies desired for research must be isolated. A few assumptions are made to limit the excessive complications created by map building and location estimation. The assumptions are listed below.

- Location estimate has a negligible global position and heading error.
- Sensor data contributing to environment data compilation has negligible error.
- Participating robots are able to effectively build accurate maps.

The second chapter of the paper describes the requirements, design, and results of a distributed spatial query system. All results are obtained through mathematical analysis and computer based simulation. Chapter three describes and analyzes all developed autonomous technology used for the controlling logic controlling maze escape. Minor verification of correctness is provided for each technology. Chapter four introduces the simulator used in correctness verification of every new technology produced in this project. The simulation program enables full maze escape configuration trials to be run and observed. Through the developed program, the correctness and performance of the system as a whole can be observed and analyzed. The fifth chapter contains an in depth description and outline of the theoretically based expected maze escape cost calculation. This chapter is very important as it attempts to determine if a beneficial distributed



solution is cost effective or even possible. Chapter six provides result conclusions, final remarks, and possibilities for future research.

## **CHAPTER 2**

### **DISTRIBUTED SPATIAL QUERY**

#### **2.1 Introduction**

Spatial databases are frequently used by fields involved in Geographic Information Systems (GIS). Most of the applications in these fields utilize spatial databases architected with a centralized data source. Some applications, however, which benefit from the use of a spatial database require the data to be distributed among participating nodes in the system, specifically the field of distributed area mapping by spatial feature collection requires the aforementioned database architecture. Spatial data collected by applications in this field is dependent upon characteristics of the node that performed the collection. This constraint adds complications to those involved with the standard distributed spatial database. This chapter steps through the development and analysis of a method for creating and completing data collection queries on a distributed spatial database.

#### **2.2 Design**

The goal of a distributed query is to reach the sources which contain the desired data and return said data to the part of the system that initially made the query. However, without a known or static mapping of data to system nodes, it is impossible to know which nodes contain the data that the query is requesting at the time of request. Unfortunately, a deterministic, or static spatial data mapping, is implausible with the distributed system of concern in this paper. Therefore the consequences of an unknown data mapping must

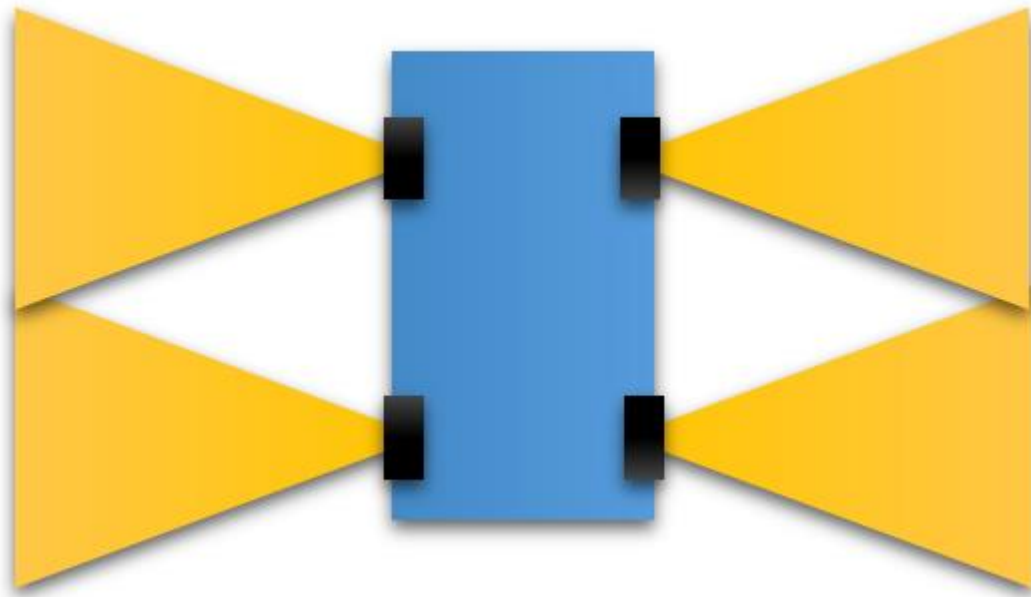
then be considered. The returning data related to a single query may be spread out over  $N$  nodes; hence the state of query completion is impossible to derive from the reception of result data sent from any given node. In addition, returned spatial query data only reflects the area that it occupies, not the area that was observed in order to initially collect it. As a result, in order for a node to acknowledge the scope of its collected data, additional information is needed in a distributed query system.

### **2.2.1 Coverage Polygon**

Each collector in the distributed system must know what portion of the world was explored to yield the environment data that it currently holds. This is achieved through the creation and maintenance of a spatial coverage database. The spatial database is comprised of coverage multi-polygons; each with a quality attribute labeling the collector's error bound density at the time of its most accurate update.

Each time a collector polls a sensor, thereby observing a portion of the environment, the coverage database must be updated. The exact portion of the environment observed is dependent on sensor characteristics. These characteristics dictate a region, referenced by the sensor's global location, where all environment data was detected. This region is known as a visibility polygon and is used to create coverage polygons. An example of a visibility polygon generated by an ultrasonic sensor is depicted in Figure 1. An ultrasonic sensor is able to detect the closest object within a field of view. Therefore, its visibility polygon is a cone with length equal to the distance value returned from polling. If multiple sensors of the same type are polled at the same time, their visibility polygons can be unioned together to begin the creation of a new coverage polygon. A coverage polygon differs from a visibility polygon in that it accounts for the accuracy of the

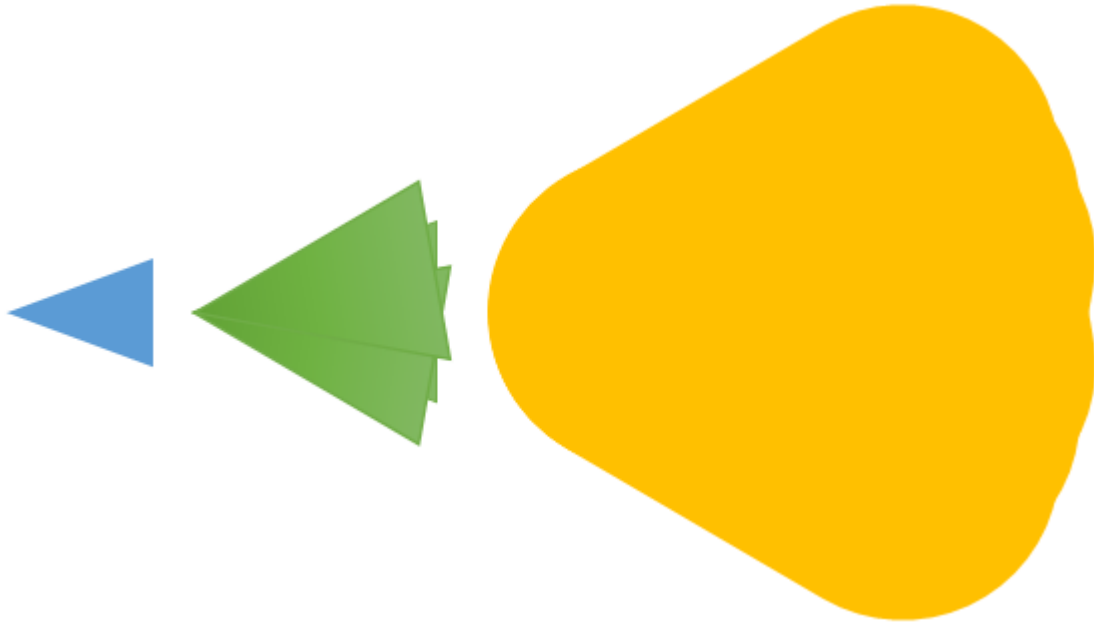
collector's global location estimate at the time the sensor data was collected. This is essential as the visibility polygon is referenced by a location estimate; therefore the exact region that could have been viewed is larger than the region defined by a sensor's characteristics. In addition, a coverage polygon must only represent data from a single sensor type. A sensor type defines the type of data that is collected by a given sensor. For example, ultrasonic sensors fall under the physical proximity sensor type. The spatial coverage database must have multiple layers for coverage polygons of different types. This way, the coverage maps of different data types will not conflict with one another.



**Figure 1: Collector platform (Blue) with for proximity sensors (black) and their visibility polygons (yellow)**

To create a coverage polygon, the obtained visibility polygon must be expanded by an error bound. The expansion depends on both the translational and rotational error parameters of the collector's location estimate. To account for rotational error, a finite set of polygons, describing the possible visibility polygon orientations is created. Each

member of the set is then buffered by the maximum radius of the collector's translational error. The complete coverage polygon is equal to the union of buffered polygons in the set (Figure 2). The final step is to determine the density or quality attribute of the coverage polygon. The quality attribute is a value ranging from zero to one. As the value approaches one, the assumed error in the observed area approaches null. This is expressed by the ratio of the visibility polygon's area over the area of the final coverage polygon.



**Figure 2: Expansion of visibility polygon (blue) through rotated union (green) then polygon buffer (yellow) to create the final coverage polygon.**

### **2.2.2 Spatial Coverage Update**

Every new coverage polygon must be used to update its respective collector's spatial coverage database. The coverage database has a layer for each sensor group in order to maintain the observation of different environment data disjoint. Upon update, the new coverage polygon is inserted into the spatial database layer corresponding to the sensor

group it represents. Each layer contains a disjoint set of multi-polygons to efficiently describe the observed regions. In result, any polygons that intersect the new polygon must be merged or updated to reflect the new coverage state. Figure 3 illustrates a merge update and figure 4 provides pseudo code that describes the complete relational logic.



**Figure 3: Coverage polygons of same quality can be merged (blue). Overlapping sections must take the higher quality value (green).**

```
foreach(Multipolygon newCMP in NewCoverageMultipolygonList)
{
    MergeList = new List<Multipolygon>();
    Intersections = GetCoverageIntersection(newCMP);
    foreach(Multipolygon intersecting in Intersections)
    {
        Multipolygon common = Intersection(intersecting, newCMP);
        if(intersecting.Density < newCMP.Density)
            intersecting = Difference(intersecting, common);
        else if(intersecting.Density > newCMP.Density)
            newCMP = Difference(newCMP, common);
        else
            MergeList.Add(intersecting);
    }
    newCMP = Union(newCMP, MergeList);
}
```

**Figure 4: The polygon relational logic for spatial coverage update. Database update transactions are omitted for simplicity.**

### 2.2.3 Query Coverage








A collector's coverage describes how well an area was observed in order to collect the data within that region. Most collectors in practice will have error in their respective

position estimate; therefore their description quality, at any given area, is likely to be imperfect. However, if multiple collectors observe the same region imperfectly, their collective knowledge of that region is greater than the individual parts. Ergo, a distributed spatial query's desire is to calculate the collective observation quality of the system in a relevant area. Once the quality of the entire query region is considered acceptable, the query is complete. To achieve this, spatial queries use collectors' coverage data to record what parts of the interest region have been answered and to what extent. A query message will sum coverage qualities together on subsequent collector visits.

New spatial queries are initialized with a single completion multi-polygon identical to the area of interest that has a quality attribute of zero. This reflects the query region and the fact that it has not been answered. Upon a query's arrival to a node, the collector adds the intersecting areas of its spatial coverage database to the query's completion polygon. The collector then processes the query based on the current region of interest and adds any results to a return query message. The result message contains a coverage polygon equal to the intersection of the original query polygon and the collector's spatial coverage. This ensures that any returned results are paired with the area that was observed in order to collect them. Any query region that attains a quality attribute greater than or equal to one is removed from the query multi-polygon. If the multi-polygon is not empty, the remnants of the processed query is passed on to other collector nodes, otherwise, the query is complete. When query result messages are received by the originating node, the node adds the result data to its collected database and the completion polygon is merged into its coverage database. The process of merging this

completion polygon is identical to that of the coverage merge described earlier. Table 1 demonstrates the progression of a query completion polygon as it is answered by multiple collectors.

**Table 1: Progression of a distributed spatial query**

	Initialization	First Collector	Second Collector
Collector Coverage	N/A		
Completion Multipolygon			
Result Multipolygon			

#### 2.2.4 Message Passing

Connectivity between any two nodes over a mesh communication network cannot be guaranteed. Unfavorably, there is no guarantee that a query will ever get to a node with relevant information. Moreover, in the event that a query reaches a node with relevant data it cannot be ensured that the results will make it back to the original node. Since there are no connectivity guarantees and the connectivity of the network can change constantly, the available communication graph must be explored as quickly and thoroughly as possible. This will give query data the highest probability of reaching desired system nodes. Therefore, each node will broadcast all query messages to all other nodes in communication proximity. This is similar to the technique of optimistic replication. Companies such as Amazon and Google use optimistic replication to share



information quickly over large systems. Regrettably, this communication method produces a very large number of messages per query per node on the order of the expression displayed in equation 1.

$$M(t) = O(\tilde{E} * t + \tilde{E}^t) \quad (1)$$

The initial query message is sent by node  $Q$  to  $\tilde{E}$  nodes where  $\tilde{E}$  is the average number of directed edges per node in the communication graph. The second communication iteration results in each of the  $\tilde{E}$  nodes around  $Q$  receiving  $\tilde{E}$  messages. A fraction of  $\tilde{E}$  nodes times  $\tilde{E}$  receives their first message belonging to  $Q$ 's query. Upon the third iteration, the  $\tilde{E}$  nodes around  $Q$  each receive  $\tilde{E}^2$  messages while the next furthest group of nodes from  $Q$  each receives  $\tilde{E}$  messages. A fraction of  $\tilde{E} * \tilde{E}$  nodes times  $\tilde{E}$  receives their first message. This pattern shows a huge increase in the number of messages sent with respect to  $t$ , the number of communication iterations. If the pattern is analyzed to find the number of messages seen by a single node with respect to the number of communication iterations yields an expression of order equal to that of Equation 1. Therefore, the order of the number of messages generated must be decreased before this method can be considered practical. This is achieved by removing duplicate messages and merging messages originating from identical queries. This is equivalent to changing equation 1 to equation 2.

$$M(t) = O(1^t) \quad (2)$$

The number of messages per node per query is therefore of constant order with respect to time; however, the number of queries is unbounded. To keep the number of messages per node constant with respect to time and queries, the number of messages per node with respect to a single query must go to zero as time goes to infinity. Both limit

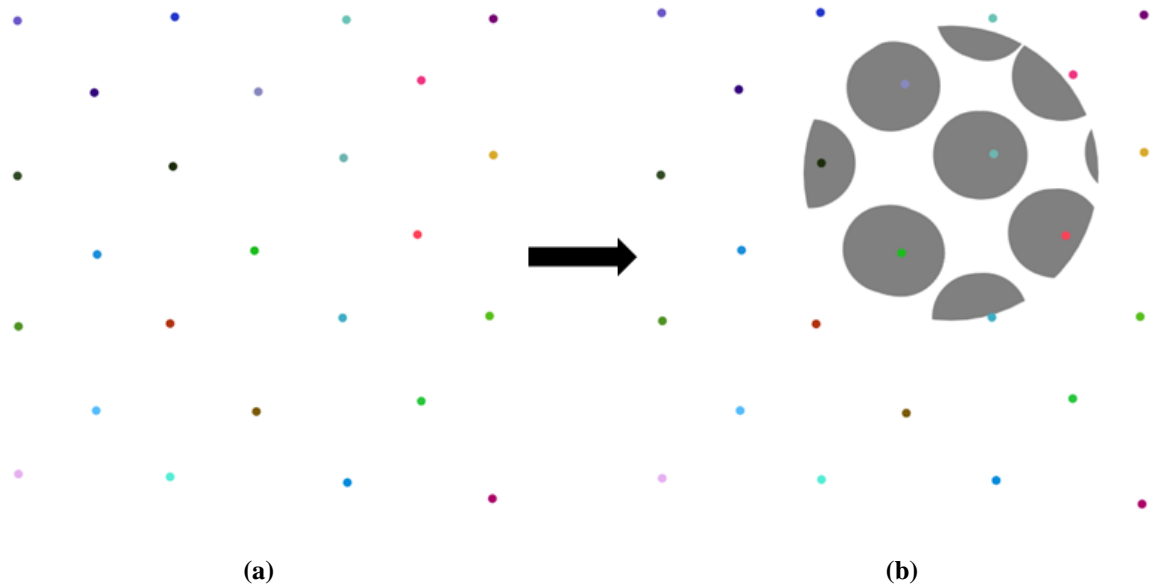
and message bound are realized with a query message lifespan. When a new query is produced at time  $t_0$  we have a guarantee that resulting messages will not exist at any node after some time  $t_0 + T$  where  $T$  is finite. This analysis indicates that this method is feasible in theory.

## **2.3 Query Simulation Results**

Performance analysis and correctness testing is required before the distributed mesh networked spatial query system can be used in the desired maze solving application. However, both the unknown data mapping and absent connectivity guarantee characteristics of the system make this difficult. The only thing accomplished with mathematical analysis was a bound on message generation. Determining the typical quality of query success is a task better suited for simulation. A graphical user interface along with a simulation program was written in order to perform query analysis and correctness testing. For organizational purposes, simulator details are discussed in chapter 4.

### **2.3.1 Message Passing and Query Time**

The goal was to see if the number of messages per node was seemingly bounded to a constant even when  $\tilde{E}$ , the average number of communication network edges was great. The test used a grid of 25 collectors distributed so every internal node would have 4 communication graph edges. The node distribution is shown below in Figure 5(a).



**Figure 5:**

**(a) Collector distribution of message passing test**

**(b) Final query results showing updated coverage database of center green collector**

The center green collector launched a query that contained a rather large region. The first of the results returned at the second communication iteration which is optimal given that the closest nodes with relevant information were in immediate communication proximity. Full results returned at the fourth communication iteration which is also optimal since the furthest node with relevant coverage was two steps away in the communication mesh network (Figure 5(b)).

The message passing performance did not have positive results initially. The test was first conducted before input queue messages were filtered for duplicates and redundant queries. The initial results showed the number of messages per node rose exponentially with each progressive communication iteration. After 10 iterations, the average number of input queue messages per node was about 12,000. There were so many coverage polygons that the program ran out of memory and crashed. Clearly this kind of optimistic

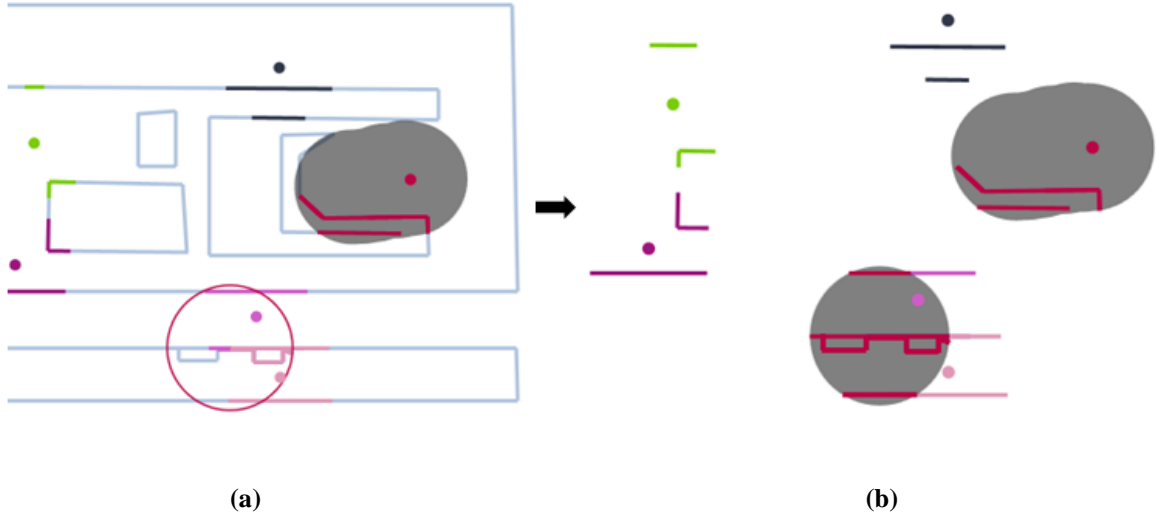
replication would not work with a system of this kind. The solution was to remove duplicate messages from the input queue and merge like query messages before processing them. Like queries messages are ones that are of the same type (query or result) and have the same query ID. The query ID is a 16 byte randomly generated unique query identification number generated and attached to a query on creation. After this filtering was implemented, the test was rerun. The result response times were identical as both message passing schemes still traversed the communication network graph as quickly and as thoroughly as before. However, this time the message passing results were much better. The maximum number of input messages any one node received upon new communication iteration was 9. That is the number of input messages received before filtering or merging. This means that filtering and merging input messages results in bounding the number of messages sent per node per query. This is the exact result that we were looking for. Both outcomes from the tests prior and post to input message filtering support our message passing mathematical analysis results. Since a message lifespan was implemented we know that if the number of active queries is finitely bounded then the number of input messages per node is also finitely bounded. From this result we can hypothesize that the number of input messages will increase linearly with respect to the number of live input queries.

In order to verify our previous claim another test is run with a near identical collector configuration. This time however, the center actor executes 4 unique queries simultaneously. The simulation results again showed that query results were received by the asking node in an optimal number of steps. The maximum number of messages received by any one node upon communication iteration was 32 while the average was

around 10. This supports our hypothesis that the number of input messages has a linear order or increase with respect to the number of queries. The summation of simulation results validates our claim that the message passing method developed is practical in application.

### **2.3.2 Environment Data**

The generic performance of the distributed system in simulation was shown. The final tests are used to verify that environment data and query coverage are effectively returned by the distributed spatial query. Figure 6(a) shows the initial coverage of the collector and the region it is about to query data from. When the query is completed, all environment data contained in the query region that is also known to the collective system should be known to the collector and the regions that were observed to complete the query should be present in the collector's coverage database. Figure 6(b) shows only the environment data known to the collector that originally executed the query in addition to the collector's complete coverage database. The results show that indeed all data known to the system in the area requested is now known by the query originating node. The coverage is of the complete area with quality equal to 1, therefore the data query has been fully answered.



**Figure 6:**

**(a) The state of the system at the start of a query (originating from red node). The query region is shown as a red circle. The steel blue lines indicate environment wall data that is not currently in view of any node.**

**(b) The final state of the system after the completion of the previous query. All red lines indicate the environment data known to the originating node. The gray regions indicate the red node's coverage database contents. All other colored lines are the immediate view of the remaining system nodes.**

The number of simulation scenarios is infinite and seemingly uncountable which makes automating simulations for a portion of the possibility space difficult. It is for future research to determine what subset of simulation configurations is useful in determining additional system performance characteristics. However, the simulations run thus far have yielded useful and positive results.

## **CHAPTER 3**

### **DISTRIBUTED AUTONOMOUS SYSTEM**

#### **3.1 Introduction**

Thus far, every robot has the ability to record surrounding environment data in the form of a globally referenced map. The distributed system is able to complete spatial queries given a region of interest. Data received by spatial query result can be effectively integrated into a robot's partial map. There are several steps yet left to take before a distributed maze solving solution can be realized. Map data queries must be autonomously created and executed in order to utilize a multiple platform solution and minimize the effort of solving the maze. Each robot must use their partial view of the maze to calculate possible navigation plans. It is required that a subset of these plans be chosen in order to optimize the maze solution of the system as a whole. Finally, the selected plans must be used to physically exit the maze.

#### **3.2 Minimum Exit Path Calculation**

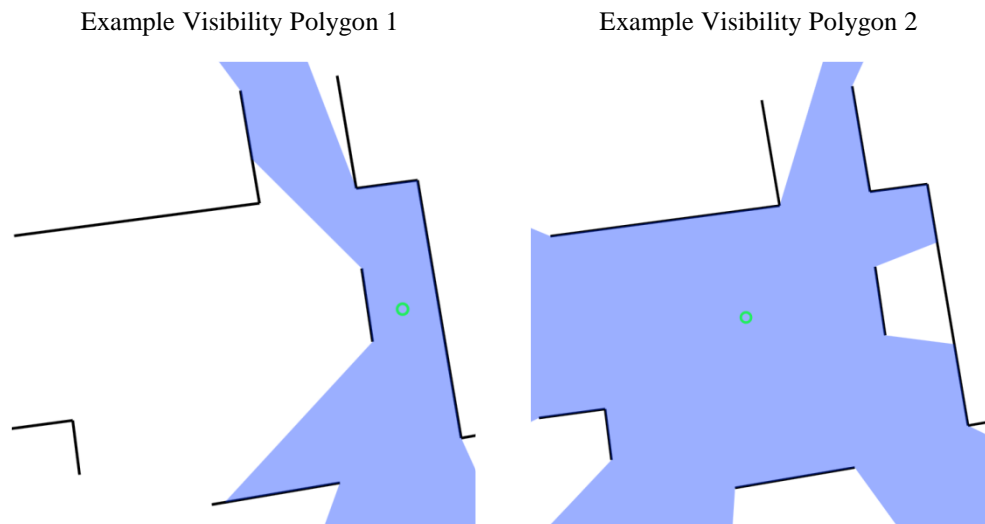
The process of exit path calculation is applied to each robot's partial map of the maze. The calculation must be performed whenever the map is updated, otherwise, a suboptimal exit plan will contribute to the solution. For the purpose of this research, the assumption that maps are represented as collections of globally referenced vectors will continue. Optimizing planned navigation through a vector field is difficult (REFERENCE: Smoothly Blending Vector Fields for Global Robot Navigation). In fact, in 3-Space, the shortest path problem is NP-hard (REFERENCE: Planning Algorithms). To solve the

problem it is desirable to treat our collection of vectors as a directed graph with weighted edges that represent distance. The goal is to perform a breadth first search on the graph starting from the node corresponding to the platform's start location. Our search is for graph leaves, which are exit nodes. There may be multiple paths to a single exit or multiple exits within a graph. Therefore, it is our goal to find the shortest path to each exit. The main question is how do we perform a BFS on a collection of vectors? The answer begins with visibility polygons.

### **3.2.1 Visibility Polygons**

Given a robot's placement inside a collection of obstacles, to determine a path past all walls it must be known which ones immediately block the robot from moving in any given direction. Whatever blocks the robot's line of sight in a current direction will also block its movement. Therefore, if it is determined the parts of walls that block our current view into infinity in every direction, the robot will know what to immediately avoid navigating into. The algorithm to create a visibility polygon provides exactly this information. By ray tracing at different angles around the current location and picking the first wall vector that is intersected, a polygon describing the robot's current line of sight in all directions can be created. Given a set of walls represented as vectors, figure 7 provides the visibility polygon constructed from two different starting locations.





**Figure 7: Examples of two visibility polygons originating from the green circles.**

The information derived from the visibility polygon tells a robot several about the surrounding. The directions of which view is completely blocked by obstacles, partially blocked by multiple obstacles, and not blocked by any obstacle. This information is directly used to help compute a path out of the vector field.

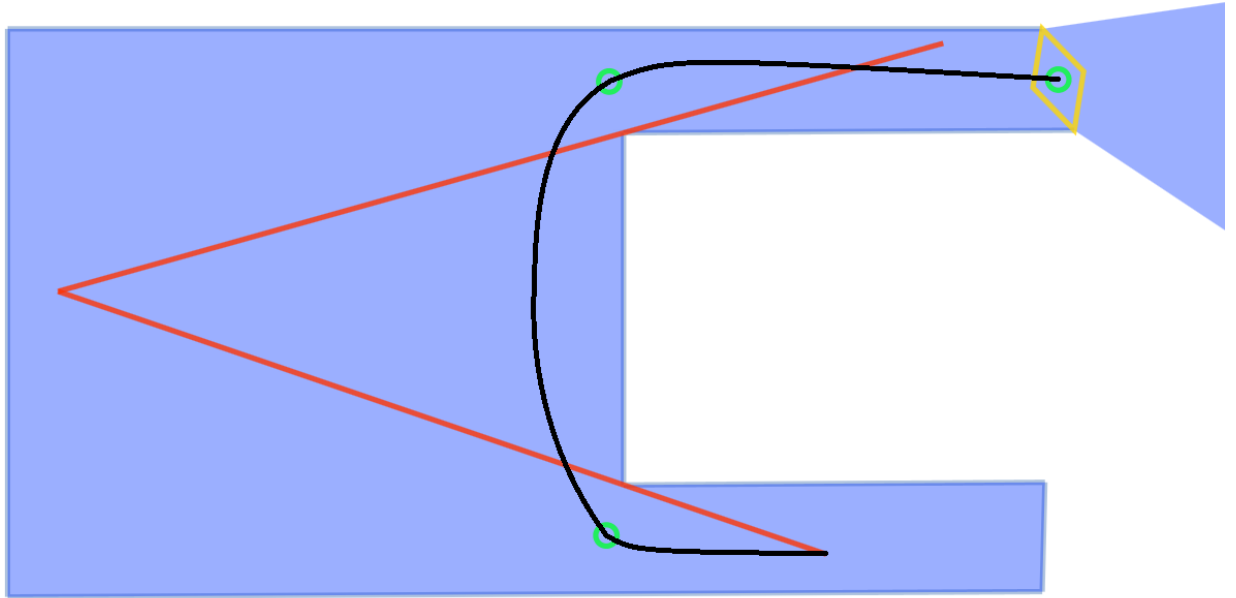
### **3.2.2 Vector Collection Breadth First Search**

The search starts by calculating a visibility polygon originating at the robot's current global location and creating the search graph's starting node at that point. The vertices of the resulting polygon are then analyzed to determine their significance. Each vertex can be one of three things. The vertex can be common to the vertex set comprising our wall collection. This indicates that our vision at this point is blocked by the adjacent wall. Sequential vertices in the visibility polygon that are common to our wall vertex set signify a dead end over that range of vision. Therefore, this category of vertex cannot be used to continue the search and is ignored. The second category of vertex is described as

a point along one of the walls in our collection. Such a point indicates that another wall has blocked the complete view of the observed wall. Hence, calculating a second visibility polygon originating from this point will reveal something more than what the current polygon can perceive. This category of vertex is referred to as a continuation point. A new node belonging to this point is created and by linking it to the node of the current visibility polygon, the search is able to advance. The third and final category is known as an infinity vertex, which signifies the situation where no wall data that is blocking our view in this direction. Therefore, we have found an exit to the map. To instill an exit path into the graph, a leaf node belonging to the infinity vertex is created then linked to the previous graph node. By maintaining a list of leaf nodes, every exit path can be found by traversing up to the start node. The BFS is specifically conducted by add all continuation nodes to a work queue and saving all infinity vertices as exit possibilities. An iteration of the search removes a single node from the queue and repeats the initial procedure with a new start location. The search finishes when the work queue becomes empty. However, the described method for BFS on our vertex collection is lacking one critical feature. Both BFS and DFS require nodes to be marked once visited in order to prevent infinite recursion. Since our nodes are represented through origin and visibility polygon, a simple Boolean marker is insufficient to prevent looping. To effectively ‘mark’ processed nodes, the union of each processed node’s visibility polygon is used to determine if new vertices have been visited before. If a vertex intersects with the union of all visibility polygons, it is not added the work queue. Once the map is fully covered by visibility polygons, it is impossible for work to be added to the queue, guaranteeing search completion.

### 3.2.3 Edge Weight Calculation

The sum of edge weights from the start node to any given exit should represent the distance that must be traveled to exit by those nodes that are along the way. The distance from the start node to each exit through all possible paths is necessary for the calculation of minimum exit paths. We will call the origin of a node's visibility polygon a visibility point. The distance between two node's visibility points is an incorrect representation of this distance as a robot may be able to take a more direct path while following the direction of these vertices to the exit. Figure 8 provides an example of this. We must calculate points that describe the minimal exit path within the found minimal exit node sequence. This point will be known as a navigation point and must be calculated for each node. The edge weight between two nodes will be the distance between their navigation points.



**Figure 8: Red lines connect visibility points that form a non-minimal path. Green circles represent waypoints for a minimum distance path. The black line shows the possible robot executed path using the minimum waypoints.**

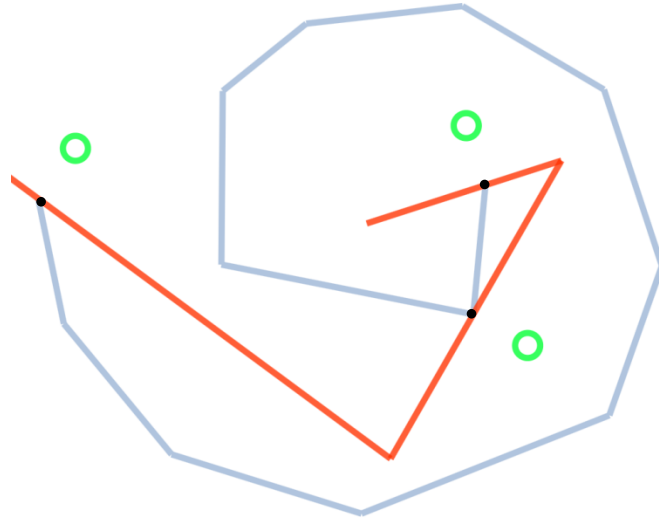
### 3.2.4 Navigation Point Calculation

An edge indicates that moving from one visibility point to the other will yield a cumulatively more complete view of the wall collection. This truth is a result of the fact that the previous view was blocked by a wall that the second visibility point is able to see past. The closest point to the first visibility point that is able to completely see around this blockage is a vertex belonging to the blocking wall. The vertex that we desire additionally belongs to the previous visibility polygon and is sequentially located to the second visibility point along the first visibility polygon's hull. This point is marked by a black dot in figure 9. This point can help to calculate the navigation point as it provides the shortest point around a critical obstacle. There should be some extra space between the wall and the navigation point to allow for effective robot navigation without collision. The final element is what direction to put the navigation point with respect to the shortest point. The direction is calculated by using the normalized vector perpendicular to the vector between the first and second visibility point  $\overrightarrow{v_N^T}$  and the set of walls common to the blocking vertex  $\overrightarrow{W}$ . If the dot product of that vector and any wall (with its origin equal to the common vertex  $\dot{c}$ ) is greater than zero, then the two vectors face towards the same side of  $\vec{v}$ , thus  $\overrightarrow{v_N^T}$  must point in the opposite direction in order to avoid collision during navigation. The logic is described below with the navigation point indicated by  $\dot{n}$ . Examples of navigation points are illustrated with green circles in figure 9.

$$if \exists \vec{w} in \overrightarrow{W} \text{ s.t. } \vec{w} \cdot \overrightarrow{v_N^T} > 0$$

$$then \dot{n} = \dot{c} - \overrightarrow{v_N^T}$$

$$else \dot{n} = \dot{c} + \overrightarrow{v_N^T}$$



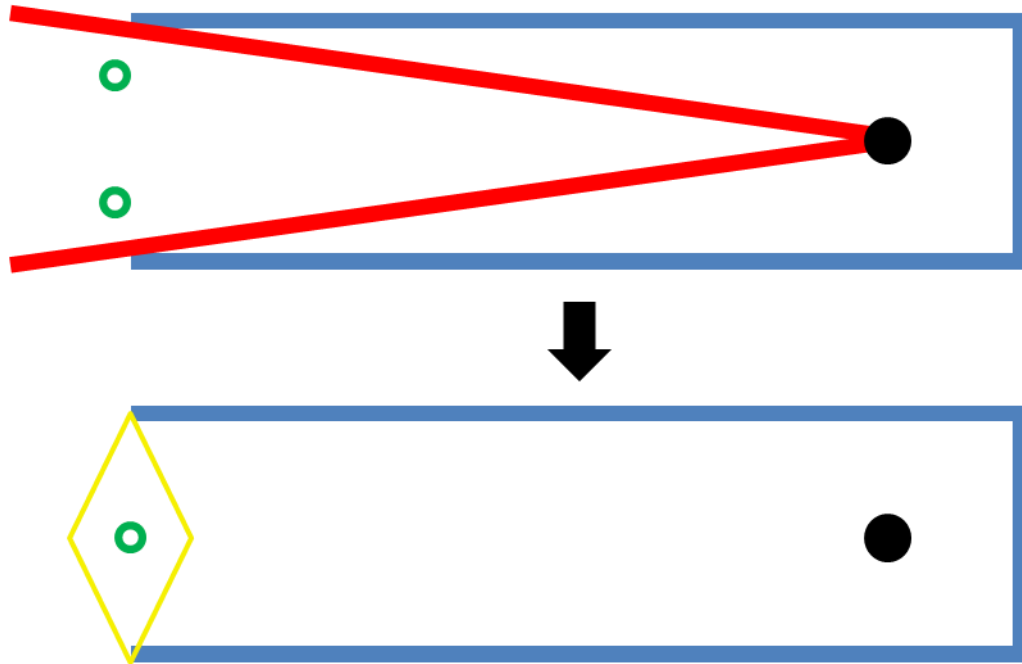
**Figure 9: The red lines connect visibility points that are the origin of the visibility polygons. The green circles are the navigation point locations calculated using the wall vertices (black dots) and the two visibility points connected by the red line intersected by the wall vertex.**

### 3.2.5 Exit Plan Stabilization

An exit node's visibility point can be located anywhere there exists a view out to an infinity point. This means that there are an unbounded number of possible exit nodes for any single exit. The only thing that prevents the BFS from infinitely generating exit nodes is the visibility union's intersect check. Unfortunately, the check does not give physical exits and exit nodes a one to one relation, or force a physical exit to have a unique exit node, independent of starting node location. The absence of these conditions results in possible loop conditions during runtime. To clarify, the loop does not occur in the BFS, the loop is embodied through an oscillation of navigation plan selection. The oscillation causes an indefinite suspension of a robot's exit progression. The error is a direct result of exit distance calculation variation but is a symptom of a greater problem, exit node instability.

To assist with the solution, the concept of an exit gate is created. An exit gate combines the two possible exit nodes of any physical exit together. When viewing an exit from inside the maze, two different wall vertices will bound the view span of infinity. Together, they serve the same purpose and consequently should be grouped together since exit uniqueness is part of the upcoming goal. Additionally, exit gates are later used for querying purposes. Figure 10 shows the consolidation.

The stabilization solution is reached through iteratively updating an exit node's position until no change in position is seen. When an exit node is found, the calculated navigation point is used to create another visibility polygon. If any exit node extracted from this polygon results in an identical navigation point, the previous exit node is considered to be stable and is subsequently added to the list of possible exit plans. This stabilization technique creates the possibility of search looping which are easily filtered by a recursive check. Now that all possible exit plans are stable, it is a simple task to return the shortest path for each physical exit.



**Figure 10: The red lines and paired with a green circle show the two possible exit paths. The green circle centered in the gold diamond is the exit gate that results from combining the two redundant exit path options.**

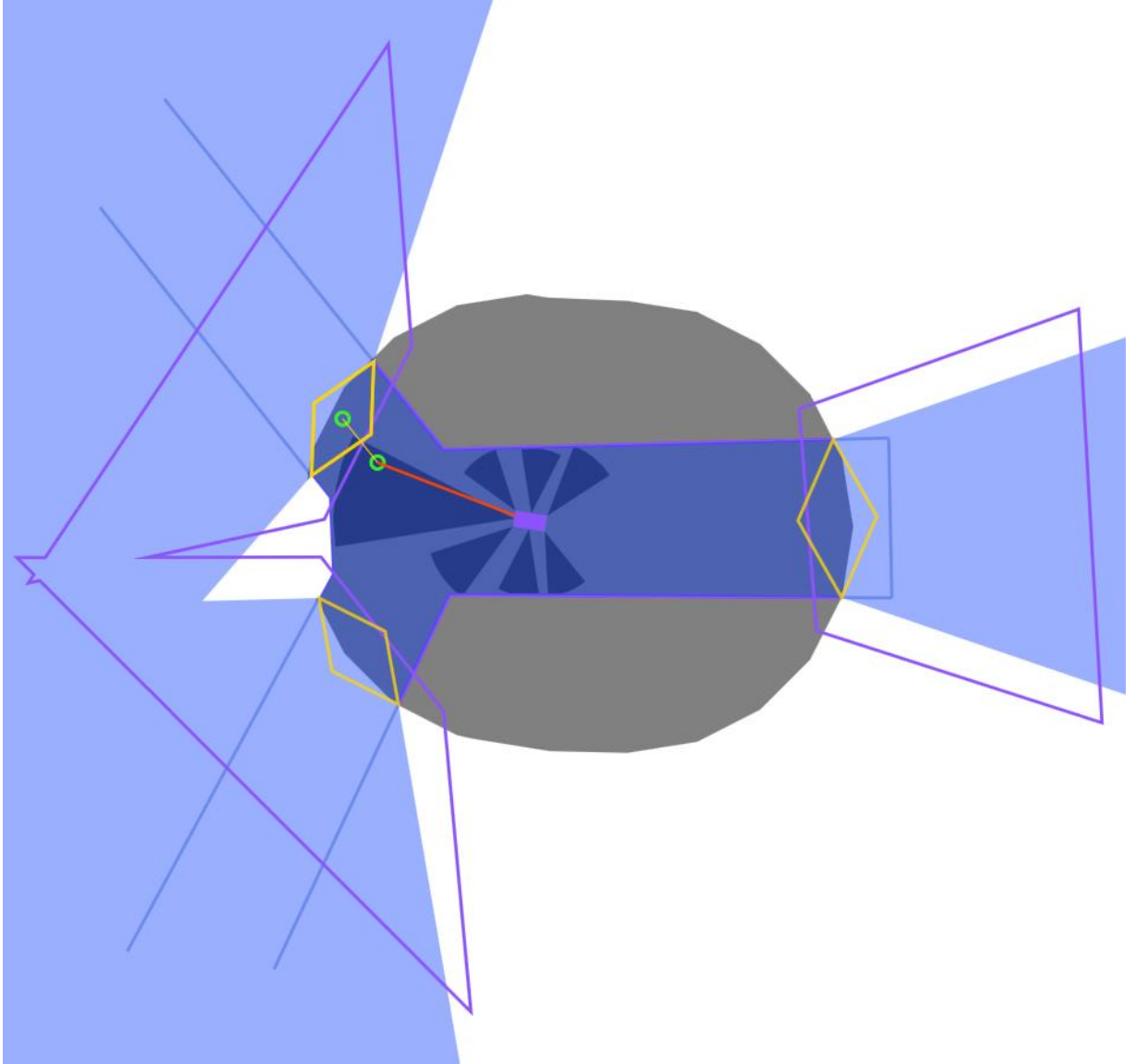
### 3.3 Autonomous Query Creation

The goal of the system is to get each robot out of the maze with minimal effort and in minimal time. To do this, redundant work must be minimized. The function of map data queries is to discover what work has already been done by the system. By spending communication cost in place of physical movement cost, the overall cost of gaining redundant information is lowered. The current problem is to determine what areas should be queried and when.

It is possible to query for map data over the entire world. The effect would be to pull in the entire system's knowledge every time a query is executed. Even though this would fulfill the desired purpose of the query, it is impractical with large systems and may drive

the final solution cost above the non-cooperative solution. The situation is similar when querying for all unknown areas as the effect would be to constantly sync with system updates. The only regions of unknown map data that would progress the navigation toward an exit solution are those corresponding to exit gates. If the system has data in a region that a robot has scheduled for exploration, querying that data could save the physical effort of moving to that area. The area could be a dead end or a much longer route than previously known. Conversely, a region query yielding deprived results may indicate an area unknown to the system. Regardless, it is decided that each robot must only query the critical regions pertaining to their partial map. The unique exit gates returned from navigation path calculation indicate all critical regions. Spatial query polygons are created by expanding the exit gate and adjacent visibility polygon sides. Figure 11 shows examples of query polygons created from the critical regions indicated by exit gates.

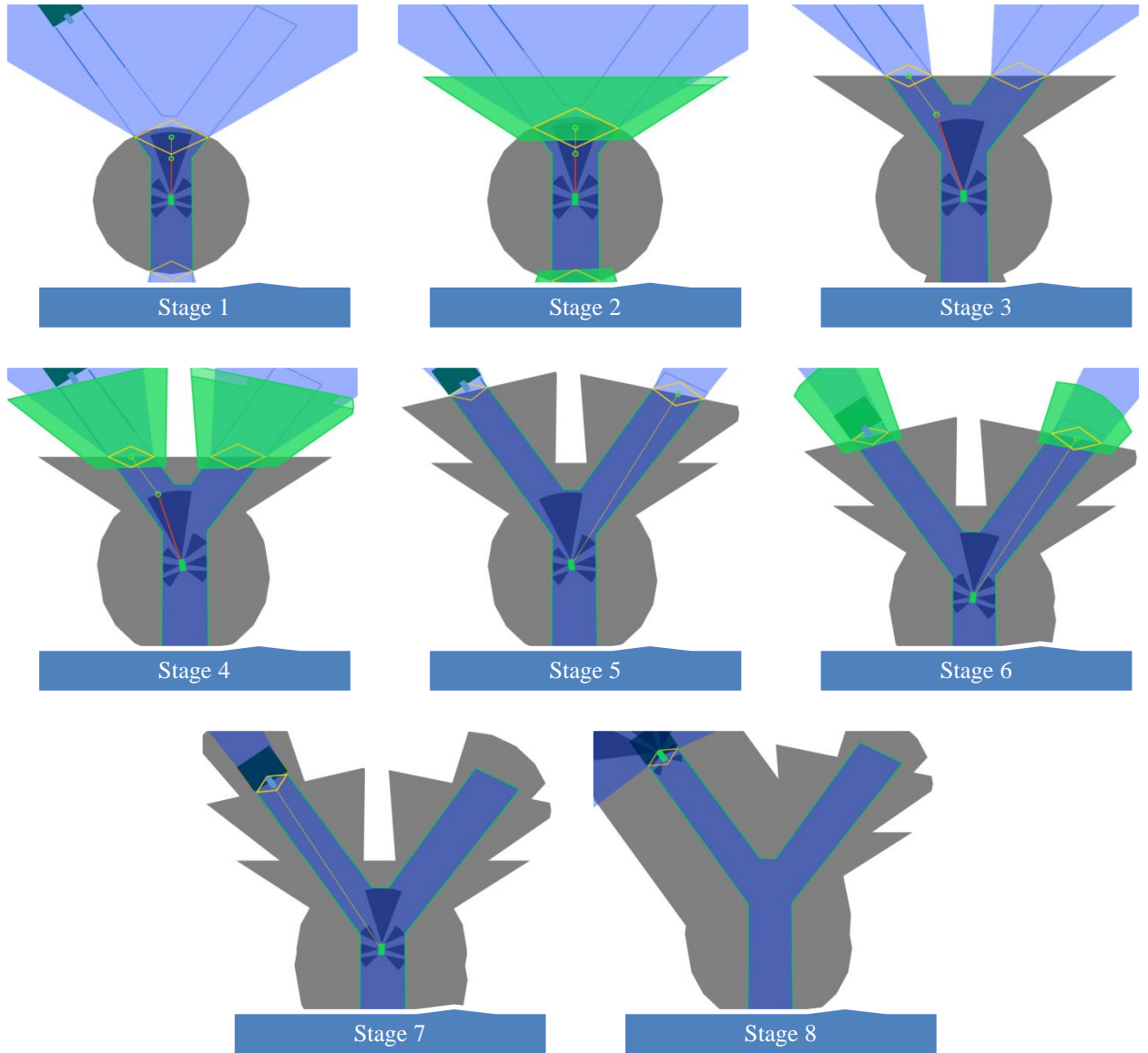




**Figure 11: The robot (solid purple rectangle) located all exit gates (gold diamonds) using its partial map of the environment (purple walls inside of its coverage polygon (gray region)). Query regions (purple unfilled polygons) were calculated using the separate exit gates and the visibility polygons from those gates (blue regions).**

Returning query results integrate into the current map, updating all the critical regions to which they apply. Sequential queries will update critical regions until the critical region is found to be a dead end or system knowledge does not include that region. Figure 12 illustrates the results of sequential queries with a system that knows the

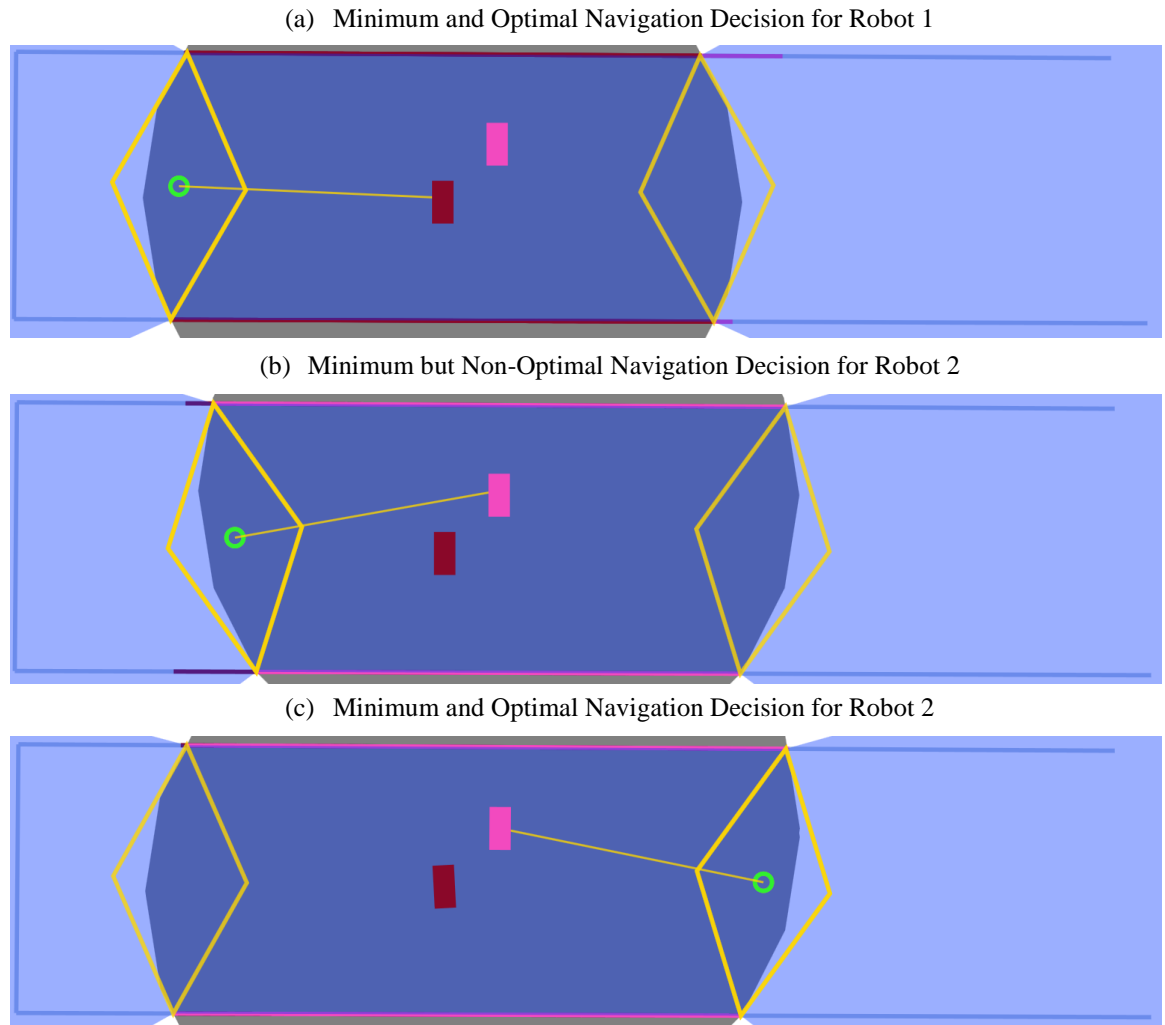
all desired critical regions. It can be seen that the robot finishes in minimum time since the parts of the maze critical to making an exit decision were filled in before the robot had to make a critical route decision.



**Figure 12: Steps through sequential queries by a fresh robot to a system with full knowledge of the maze. The query regions are created using the exit gates of each step and when data is returned (regions of data are indicated by translucent green polygons) new exit gates are created from the query updated map.**

### 3.4 Optimal System Path Selection

The collection of minimal paths to each robot's closest unknown area may not be the optimal solution for the system. For example, figure 13 shows a scenario where two actors have identical minimal paths when the ideal path selection would be to choose opposite routes. Thus a distributed optimal path selection solution is required.



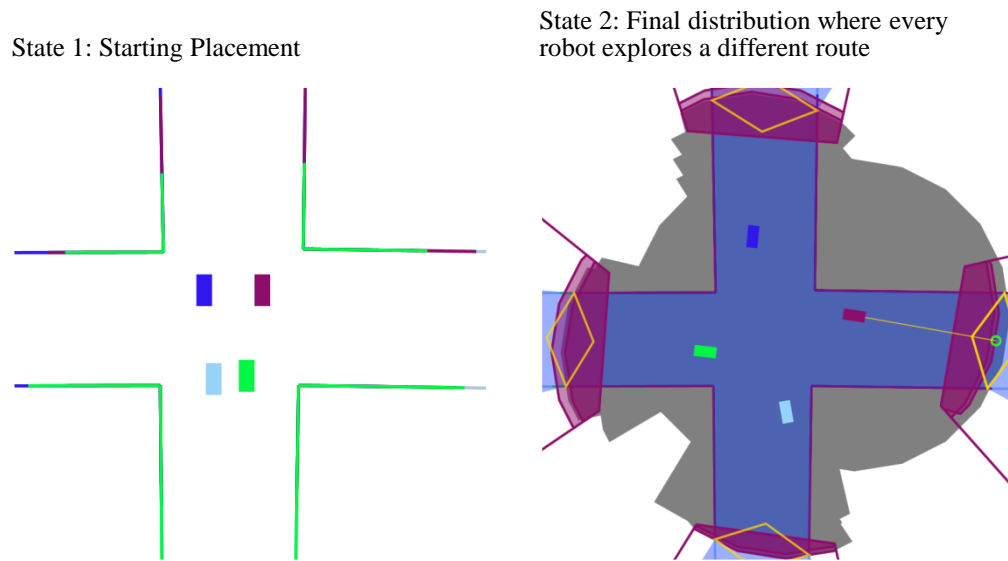
**Figure 13: An example of how each robot settles on which route to take based on the route decisions and locations of other robots around it.**

Each actor broadcasts their most recent navigation decision to those around them. Each agent maintains a time-stamped navigation decision list of those around them.

When making a new navigation decision, an agent uses this list to select an optimal path for itself. The selection process involves comparing pairs of navigation paths, checking for conflict. One member of the pair is a navigation possibility belonging to the local agent, while the other is from the foreign navigation decision list. If a conflict is found, a distance calculation is required to determine which agent is a better fit for the path. Two paths conflict when any vector connecting one path's navigation points intersects with the other path's exit gate polygon. The point where gate intersection first occurs is the starting point for distance calculation. Tracing back from that point to the start of both paths yields the distances of both respective agents. This is done so that navigation decisions made from different maps will have deterministic comparisons.

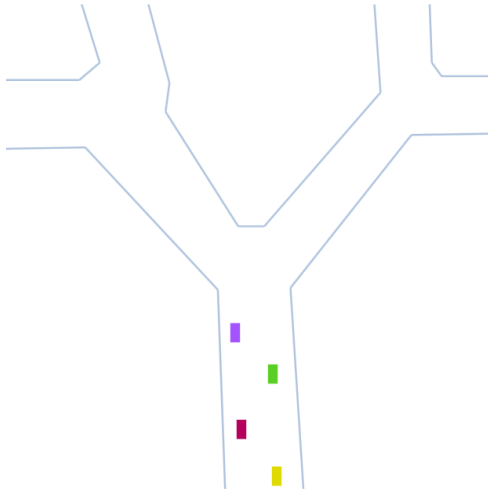
Given the conflict checking method of path comparison, the selection process can begin. Iterating from its minimal to maximal navigation decision, the local agent checks the foreign navigation list for conflict against the iterator's current navigation plan. If a conflict exists and the current agent is farther from the destination than the foreign agent, the iterator continues to check the agent's next shortest navigation decision. Conversely, if the current agent is closer to the destination than the foreign agent, it will set its navigation decision to the iterator's current and broadcast it. This will then update the foreign agent's list, in turn forcing that agent to select a different path since it now sees that it is farther than another agent and thus non-optimal. If each possible path is in conflict, the agent will take the shortest path. The consequences of this last policy are the acceptance of immediate redundant exploration/work, however it does allow for a faster divide and conquer scheme. The alternative would be to force the blocked agent to wait until an unblocked option is available. However, since it is known that the redundant

work is unavoidable, it would minimize execution time of exploration and escape if the redundant work was completed immediately. Since the possibility of multiple blockers exists, the optimal selection is the path that is blocked by the least number of foreign agents that are closer to the destination than the current actor. This way, the most even distribution of agents is selected. Figures 14 and 15 illustrate the effect this selection has on agent diffusion. Figure 14 walks through a scenario where the optimal route configuration can be immediately chosen where figure 15 demonstrates the system maintaining an even exploration distribution. An even distribution does not guarantee that the optimal configuration will be selected as only partial map views are available to make earlier decisions. However, an even distribution trends toward the average case of redundant effort when incorrect distribution decisions are made prior to complete maze knowledge.

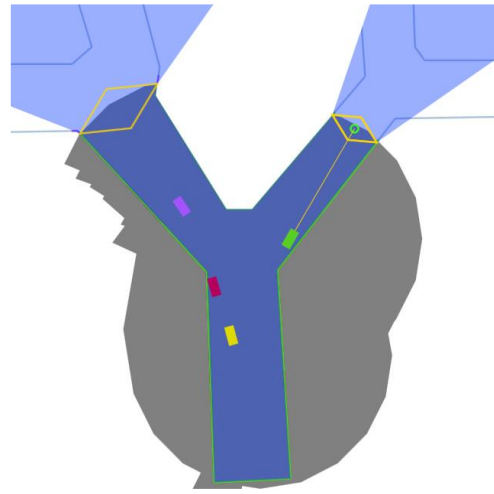


**Figure 14: Example of distributed optimal path selection. Each robot can see all possible surrounding route options. The distributed path selection service ensured that each robot picked a different route which is the most efficient way to explore the maze.**

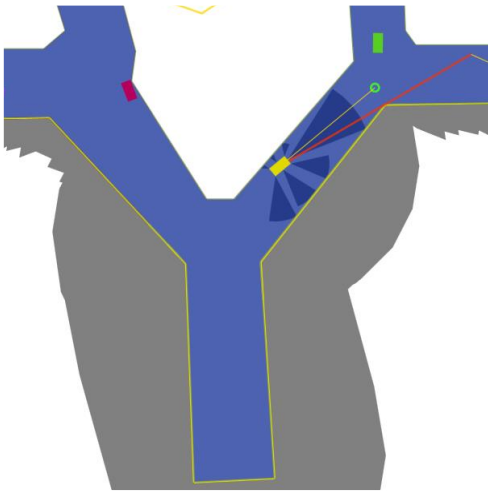
State 1: Starting robot placement



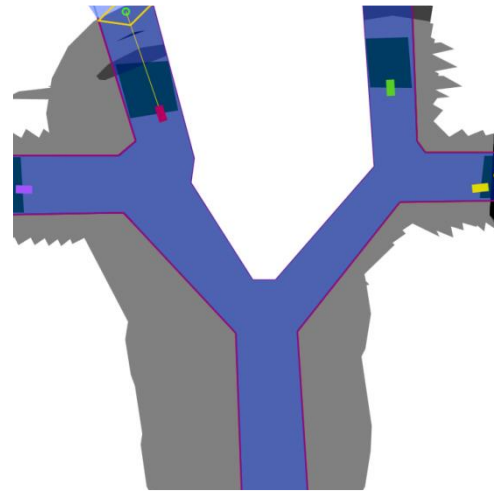
State 2: First two robots take opposite paths



State 3: Last two robots split even the distribution



State 4: All robots have split into separate paths resulting in the most efficient exploration



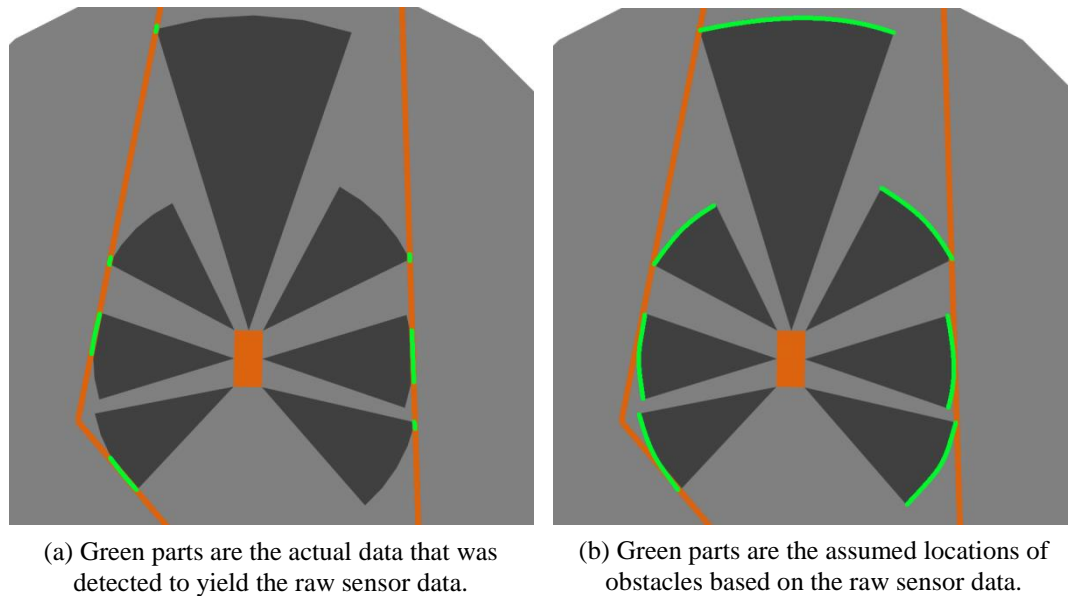
**Figure 15: Example of starting system making sequential decisions divide and conquer. The result is that all possible routes were explored as efficiently as possible.**

### 3.5 Platform Navigation Controls

The raw navigation plan is a sequence of waypoints that leads to a possible exit region. It is the goal of the platform control to follow the sequence of waypoints without colliding

into any obstacles present in the environment. Obstacles are detected with a platform's onboard proximity sensors. Although map data collection is assumed to be handled by LIDAR, this research uses a more inexpensive sensor set to collect data for navigation control input. The reason being, cheap sensors are feasible to acquire and use for physical experimentation. Hence, ultrasonic proximity sensors are fixed at useful points on the perimeter of a robot's hull.

Polling an ultrasonic sensor returns the distance to the object in the sensor's field of vision that is closest to the sensor. Consequently, a collision avoidance system must assume the worst case. That being, obstacles exist along the entire perimeter of the sensor's field of view at the distance indicated by the sensor. Figure 16 illustrates the difference between the data that resulted in the data point and the worst case assumption.



**Figure 16: Visual example of actual detected environment features verses the assumption that the control algorithm must make based on that detection.**

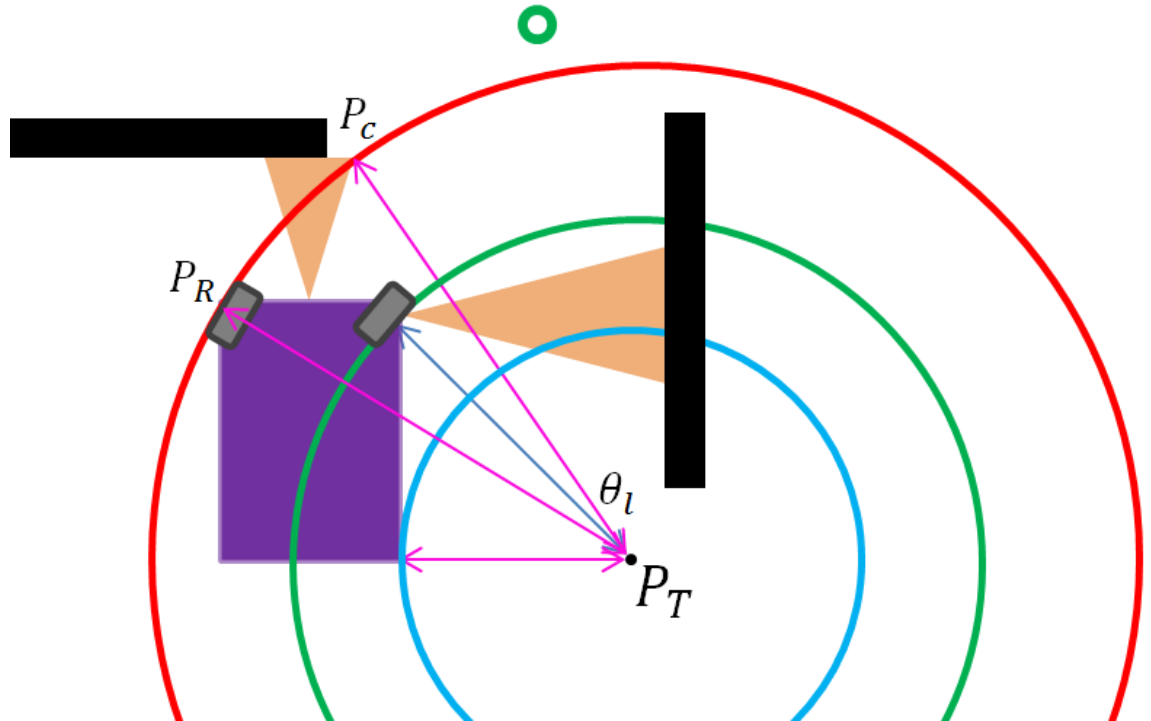
The model of robot mobility in this research is that of a four wheeled rover. The front two wheels are able to swivel in order to change the heading of the robot when all four wheels rotate forward or backward along the ground. Given this model, it is known that robot size, wheel base length, and front wheel swivel range dictate the maneuverability of the robot. Therefore, robot hull dimensions ( $width_{hull}, height_{hull}$ ), wheel base length ( $l_{wb}$ ), and max swivel angle ( $\theta_{max}$ ) serve as input to our control model.

The front sensor and the first point in the navigation path controls steering direction and magnitude while the side sensors determine if the desired steering instructions will result in a side collision. If the front sensor does not detect an obstacle then the wheels can turn directly toward the vector pointing from the robot's location to the navigation point location, provided a side collision will not result. If a side collision is predicted, the turn angle should be decreased until the side collision check passes. If the front sensor detects an obstacle, the worst obstacle assumption mentioned earlier must be made. Therefore, the navigation point determines which side of the assumed object we turn away from. Using the visibility polygon constricted from the front sensor field of view and its polled value, the point of possible collision is extracted,  $P_C$ . The opposite front corner of the robot  $P_R$ , with respect to the direction towards the navigation point, is selected. This is the point that defines the outer edge of the robot's turning radius circle. The radius of the turning circle is determined by the wheel base and the magnitude of the front wheel turn angle. The circle that passes through  $P_R$ ,  $P_C$ , and has a center point  $P_T$  that intersects a ray tracing perpendicular to the like navigation side of the robot's back tire, determines the minimum turn angle that will not collide with the obstacle. Figure 17 shows the turning radius circle just described. The angle  $\theta_l$  paired with the turning circle



described the arc length that the outer point of the robot must travel in order to miss the obstacle. This angle paired with the radii describing the exterior of the robot is used as input in conjunction with the side sensor values to check for an expected collision.

Full control details are contained in the computer simulation used to implement a working maze solving system.



**Figure 17: The red circle describes the minimum turning radius of the robot's most exterior point that will prevent a collision with the worst case obstacle detected by the robot's front sensor. The blue circle shows the robot's interior turning radius.**

## **CHAPTER 4**

### **MAZE SOLVING SIMULATION**

#### **4.1 Goal**

Now that all the technologies required to realize the maze solving solution are known, a complete system can be implemented and tested. However, the ideal map building and location estimation assumptions force the system to be implemented through computer simulation. The goal of the simulation program is to enable testing of any desired maze environment with any number of participating robots and robot starting placements. The simulation program is to be used for simple testing of the developed technologies since the size of the maze exit problem space is much too large to complete a statistically significant performance analysis. Chapters two and three were seen to heavily utilize the simulation program and the results of the separate technologies are seen in those sections. This chapter will briefly describe the simulation program and step through a few test execution examples.

#### **4.2 Description**

The state of the system and environment was held and maintained by the simulation program. Environment data, consisting of sensor detectable features, was globally available in a spatial database. Collector state was comprised of a real location, estimated location and error bound, communication range, visibility range, queues of input and output query messages, a database of observed environment features, and a spatial

coverage database. The mesh communication network is realized by using collector's actual locations and message queues.

Upon simulation iteration, a collector starts by sending actuator commands to platform controllers (motors, legs, etc.). The simulation intercepts these signals and updates the node's actual location while returning sensor data that the collector uses to estimate its own location. The location estimate provides error estimation parameters that are used in conjunction with the collector's visibility polygons to generate coverage polygons. The coverage polygons are then used to update the node's spatial coverage database. The simulator uses the collector's visibility polygons and actual location to query the environment spatial database and generate sensor data corresponding to any features the visibility polygons intersected. The sensor data is used by the collector to update its known environment features database. Nodes process all messages located in their input queue and write the appropriate messages to their output message queue. The simulated communication network then runs by sending all output queue messages to all nodes that are in range. Once the communication network has finished, the simulation is then ready to complete another iteration.

### **4.3 User Interface**

A functioning simulation program is essential for final results, but a graphical user interface is required to effectively inspect the state of the simulation at each iteration.

The GUI must enable the user to view and edit the environment.

The environment and collector spatial databases store everything in Spatialite (extension of Sqlite) utilizing the WGS84 standard coordinate system for Earth (latitude and longitude). Since the projection does not directly translate to pixels, everything that is

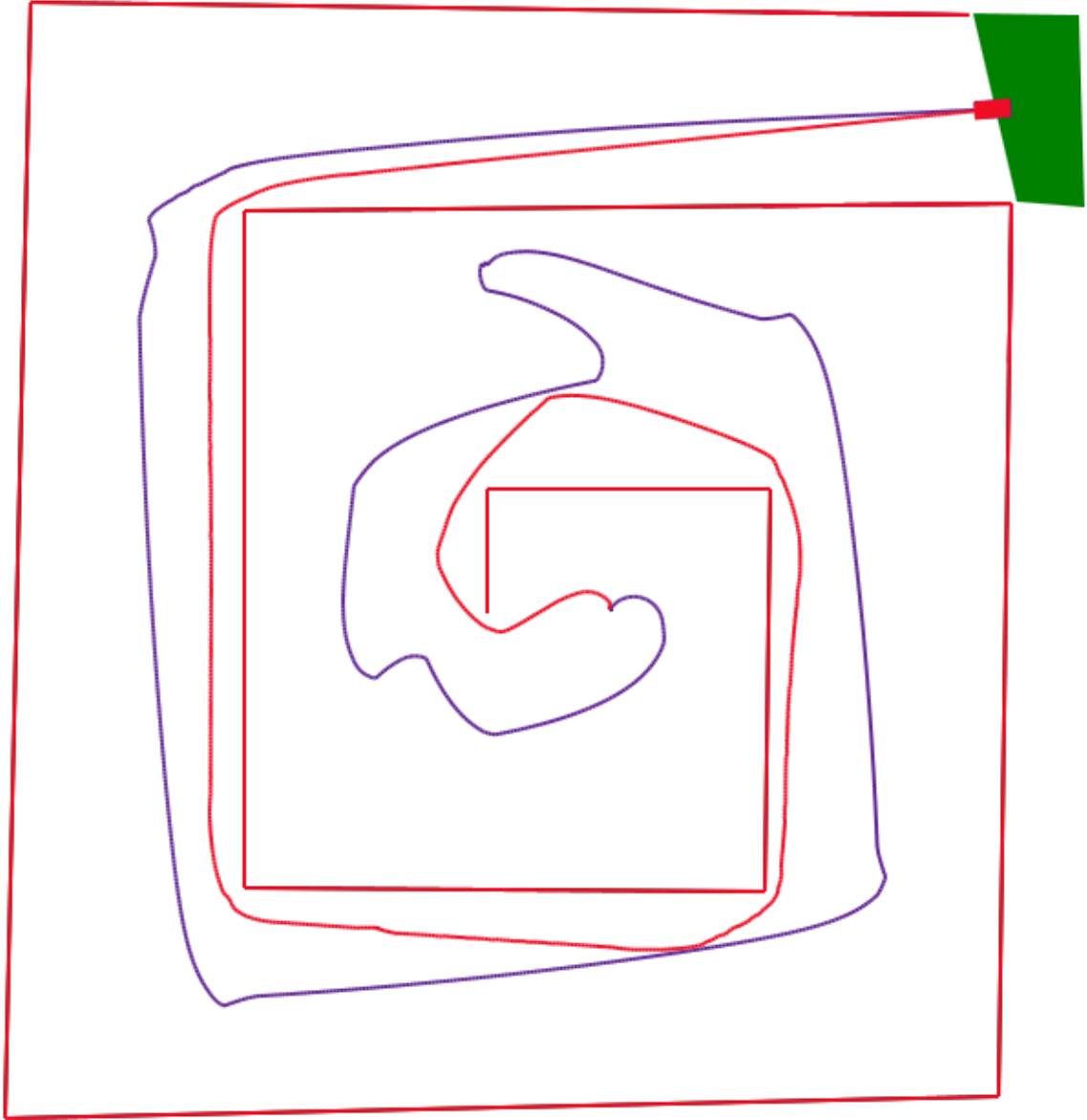
displayed must first be converted to Cartesian coordinates using a global reference point specific to the current view window. The coordinates describe the meter offset (north/south, east/west) from the given reference point. The Cartesian meter offset coordinates must then be converted to pixels using a translation and scale transformation matrix that is specific to the screen's viewing window. Whenever the viewing window is changed from a pan or a zoom user action, the transformation matrix must be recalculated and applied to every object that is drawn. This conversion must be done in reverse order whenever a user simulation edit occurs.

To add environment features to the database the user must simply right-click while holding down a particular key to draw a desired outline. The environment view can be hidden or displayed to allow for a more realistic collector view. Exits can be created using a polygon drawing tool that functions similar to the wall add function. Once an exit is reached by a robot, it considers its task complete and will no longer move but will answer query requests sent from other robots. Collectors can be added to the simulation at any time with any desired starting location to enable all possible collector configurations. Once a collector is present the simulation will be able to run. The user is able to play, pause, or step through simulation iterations which allows any single simulation state to be analyzed. When a collector is clicked on they become selected. When this occurs, a separate property window appears which displays in text that node's state. The UI will load and display the selected node's coverage polygons, message query polygons, exit gate, chosen and minimum paths, ultrasonic sensor visibility polygons, and all known environment data. Other nodes will simply display the environment data that currently intersects their visibility polygon. The user is able to

make a query on behalf of any selected node for testing purposes. When the simulation is run queries will be made autonomously by the different collectors in the system. Once a query is made, selecting different nodes will reveal the states and locations of that query's messages. The quality attribute of both coverage and completion polygons are represented through the UI by both its graphical alpha fill value and its textual representation in a collector's property window. For historical viewing purposes, each robot records all previous locations which are used to create a path history. The path history is used to show simulation test results through a single image.

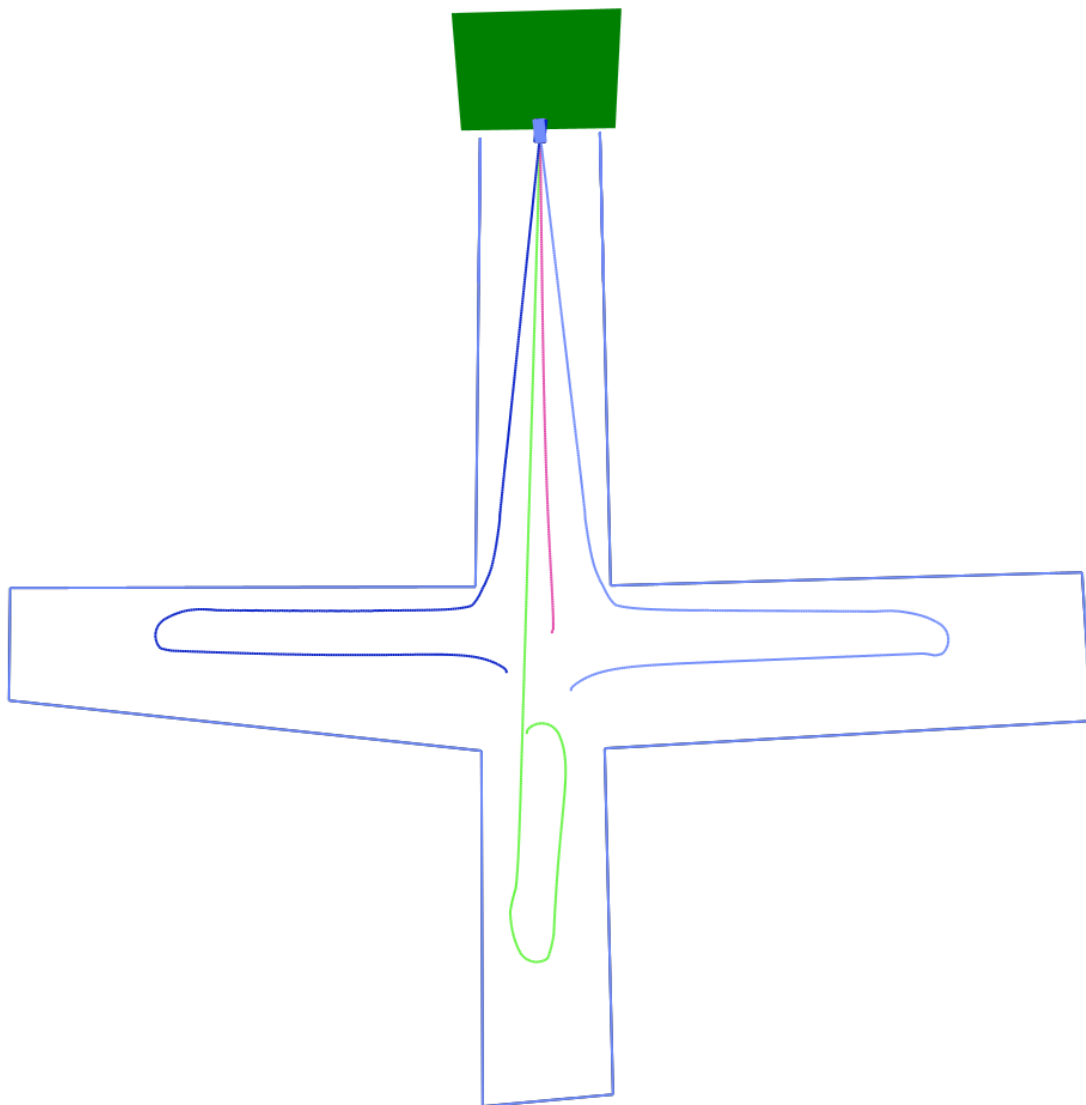
#### **4.4 Results**

The first example demonstrates improvement in path efficiency through autonomous spatial query. Figure 18 shows two different paths exiting a spiral maze environment. The purple path shows the results of a robot exploring the environment without any other robot present. The red path results from a second robot starting after the first had reached the exit at the same location as the first. The red robot was able to query the first robot for any information it had collected during its escape in autonomously generated regions of interest. The red path is clearly more efficient and had a length of 84.3 meters while the purple path had a length of 120.1 meters. The test shows that minimum path calculation, autonomous query, and navigation controls are all effectively working to help reach a solution.



**Figure 18: Initial solution path (purple) Query improved solution path (red)**

Figure 19 displays the results of the second example. Four robots started in the center of a multiple choice maze. Optimal path selection dictated that each robot explored a different initial branch option. Data sharing through query informed each robot which branch options were dead ends so that when returning robots were required to pick another branch, the non-dead end option was always selected. Table 5 shows the exit times of all participating robots.



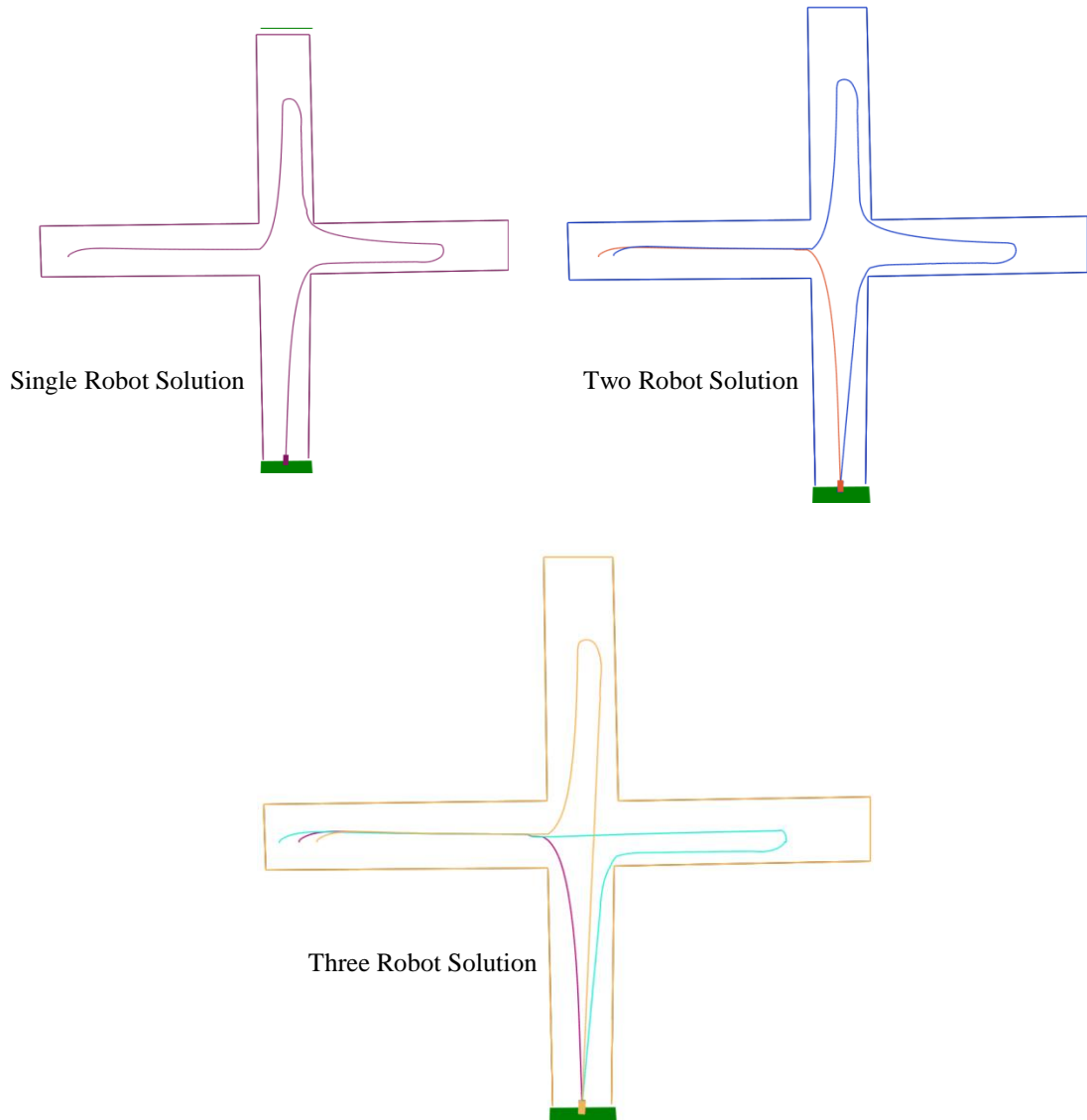
**Figure 19: Results of a four robot solution origination from the center of a ‘plus sign’ maze**

**Table 2: Exit distances of example 2 test**

	Pink	Dark Blue	Light Blue	Green	Average
Exit Path Distance	20.8	49.1	51.1	46.7	41.9

The third example is shown in figure 20 and used the same maze structure as the previous example with the exit located at the bottom branch. The number of robots used

to solve the maze is varied while the starting locations are held relatively constant. It is seen that the worst case travel time of a single robot is avoided when the work is evenly distributed among extra workers. Table 6 lays out exit distances of each solution. The average exit distance and therefore time is seen to decrease as robots are added.



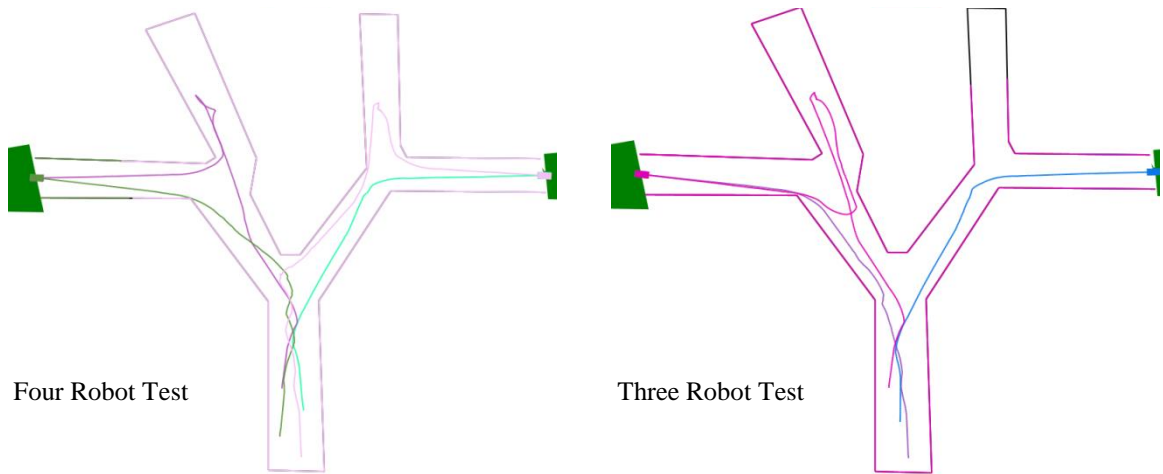
**Figure 20: The solutions of a 'plus sign' maze with bottom branch exit using 1, 2, and then 3 robots**



**Table 3: Exit distances and distance averages of robot number varied tests**

	Robot 1 Distance	Robot 2 Distance	Robot 2 Distance	Average Distance
Run 1	91.2	-	-	91.2
Run 2	90.2	38.7	-	64.5
Run 3	67.3	69.6	38.5	58.5

The final example attempts to demonstrate the need for theoretical mathematical analysis. Figure 21 is of two tests run on a three-like maze structure where sequential branch decisions are required to exit the maze. Table 7 shows the average distances required to exit given the different robot configurations. Unlike the previous example's tests, adding more robots did not decrease the exit time even though four robots would clearly be able to more efficiently explore the given region. The reason that the three robot test had a lower distance was due to the fact that the entire maze was not explored unlike the four robot test. However, since that branch was a dead end, its exploration was not required for maze solution. Branch decisions are up to chance and the simulation program created was not designed to control these decisions, it only calculates what they will be. Consequently, the simulation is unequipped to create every possible exploration outcome. However, it is desired to know how many robots will best solve any given maze. Determining the expected exit distance and cost of a solution is the only way to find the optimal maze solution configuration. The research will now turn to theoretical expected value calculation to determine the optimal configuration.



**Figure 21: Two tests run on a tree-like maze structure.**

**Table 4: Robot exit distances and averages of tree maze tests**

	Robot 1	Robot 2	Robot 3	Robot 4	Distance Average
Test 1	45.5	32.5	30.5	42.7	37.8
Test 2	48.5	33.0	31.3	-	37.6

## CHAPTER 5

### EXPECTED ESCAPE COST/TIME CALCULATION

#### 5.1 Introduction

The following chapter attempts to find the expected cost of exiting any unknown region, given a set of robots. The goal is to determine what robotic configuration optimizes the time and cost to escape a maze. The equation for the expected value is shown in equation 3. The variable  $Z$  is the size of the event space which is equal to the number of possible ways the system can exit the maze. The sum of all event probabilities must equal one as it is assumed that the system must eventually escape. From this it can be seen that the average event probability must decrease as  $Z$  increases. The cost and probability of each separate event must be known to evaluate the expression. Trial simulation and physical experiment are only practical for a small number of events, as execution time per event can be quite long. Therefore, the calculation of expected values must be approached mathematically.

$$E[X] = \sum_{i=1}^Z c_i p_i \quad (3)$$

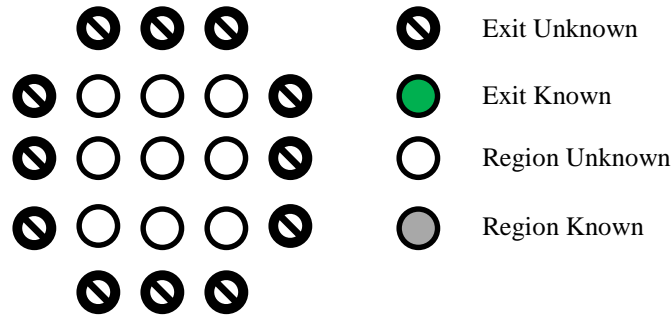
$$c_i = \text{cost of event } X_i$$

$$p_i = \text{probability of event } X_i$$

#### 5.2 Problem Space

The maze escape problem should be discretized in order to ensure a countable number of possible escape events. The discretized maze shown in figure 22 will be the base for the estimated value calculation. The maze space is broken up into grid cells of four types, known, unknown, exit unknown, and exit known. Each robot starts in one interior cell

that is set from unknown to known. Event possibilities branch out from there, one created for each possible exploration direction. Every new event recursively branches on its exploration possibilities. The act of taking a direction accrues exploration cost for the event and decreases the probability that it will happen. When an event branch reaches an exit cell, the product of the current probability and cost is added to the expected cost expression. Once all event branches have reached an exit, the expected cost calculation is complete. This is known as the brute force answer and although it seems very simple to calculate, the amount of work required is impractical.



**Figure 22: Discretized maze serves as representation of the problem environment used for mathematical analysis**

To calculate the expected cost for  $R$  robots, it is required that every scenario where all robots exit the region be found. The expected cost for every possible starting location configuration must be averaged. The number of possible starting location combinations is given by equations 4 and 5. The number of unique combinations is large but not large enough to make brute force calculate impractical for small grids and numbers of robots. However, the number of unique combinations is only the run time coefficient of the expected value calculation. The number of events given a single configuration is equal to the number of paths from a start location to each exit cell to the power of  $R$ . Consequently, the number of events required to discover optimal maze solving

configurations is far larger than what is possible to calculate. It is concluded that a brute force solution will not be used.

$$Total\ Combinations = N^{2*R} \quad (4)$$

$$Total\ Unique\ Combinations = O(N^2\ choose\ R) = \frac{N!}{R!(N-K)!} \quad (5)$$

### 5.2.1 Non-Brute Force Problem Space

Since the calculation of all possible events of  $R$  robots is too great a task, the state of the individual must be calculated and used to create an inference of the system state that is used to construct a solution. The new problem space for the expected value calculation assumes an unknown starting location configuration. This assumption is essential for a non-brute force calculation. The effects of particular starting location configuration characteristics are accounted for later. First, the components of an individual robot's state and the path toward a solution must be described before any complications can be addressed.

A robot starts in a randomly chosen cell with a known region consisting of only that cell and an unknown area of size  $N^2 - 1$ . The robot must attempt to exit the maze by exploring unknown areas. There are two methods of unknown region exploration, one physical and the other based on communication query. The details of both are laid out below.

- Physical – A robot uses its motors to move from its current position within its known region to a bordering unknown region. The cost of such a move is dependent on the desired destination block and the current position of the robot. Since a brute force solution is inaccessible, there is no way to know the current location of the robot or which unknown block is desired for exploration. The

minimum movement is one block where the maximum is  $n$  blocks, where  $n$  is the current number of blocks in the robot's known region.

- **Communication Query** – A robot asks the collective distributed system if the robot's desired block to explore is shared with the system's set of known blocks. If it is, the desired block then becomes known to the asking robot. The cost of a communication query scales with the number of robots in the system. The probability that the system will have the requested data relies on the known regions of the other robots in the system. Sequential communication queries have changing success probabilities as the unknown region of the robot making the query changes each time useful information is returned. This probability is given by equation 6 while its calculation is addressed further along.

$$P_r[QuerySuccess] = \frac{Area[\cup_{i \neq r}^R E_r \cap S_i]}{Area[E_r]} \quad (6)$$

Assuming that at each step one cell is switched from an unknown to a known cell, there now exists a sequence of the number of known cells incrementing from 1 to  $N^2$ . Synchronously, the number of unknown cells becomes a decrementing sequence from  $N^2 - 1$  to 0. These sequences are state variables that describe a robot's progression to guaranteed maze escape. Each time an unknown area is explored, there is a probability that the exploration will take the robot to an exit. This exit probability is dependent on how many blocks have been previously explored and how many blocks have yet to be seen. When the number of explored blocks reaches  $N^2$ , the probability of escape becomes 1, as the number of unknown non-exit blocks is now 0. A simple example will help explain and show what extra information is required for a generic calculation.

### 5.3 Process Example Walkthrough

A two by two maze entrapping a single robot will serve as an example of the calculation process. Assuming that all blocks bordering the maze area are exit blocks, all possible starting locations have an equal likelihood of escape. This is because each block has two edges touching exit regions and two touching other maze blocks. Given a generic  $N \times N$  maze, table 5 lays out the probabilities of escape from different block types.

**Table 5: Characteristics of exit boarder cell types**

	Number of Cells in $N \times N$	Number of Escape Edges	Escape Probability
Corner Cell	4	2	.5
Edge Cell	$N^2 - (N - 2)^2 - 4$	1	.25
Inner Cell	$(N - 2)^2$	0	0

Taking a single randomly chosen start location, the probability that the next explored edge will be an exit is given by equation 7 with a cost of 1 move. There is no possibility of spatial query since there is only a single robot in the system. The calculation of the remaining escapes continues under the probability that the robot did not escape with the previous move. Currently, the robot's state consists of two connected known cells and two unknown cells. Both known blocks have three edges that can explored. Two out of the three of each would lead to an exit, which yields a final probability of 4/6 for exit at the next step with a worst case movement cost of 2. Moving to a third unknown block will give a total of eight possible moves to unknown space with size of them leading to an exit. The exit probability of this state is 6/8 with a worst case movement cost of 3.

After moving to the final block, the robot is guaranteed to exit with a maximum movement cost of 4. Figure 23 shows the escape progression.

$$P_{escape}[X_1] = \frac{4}{N^2} * .5 + \frac{N^2 - (N-2)^2 - 4}{N^2} * .25 \quad (7)$$

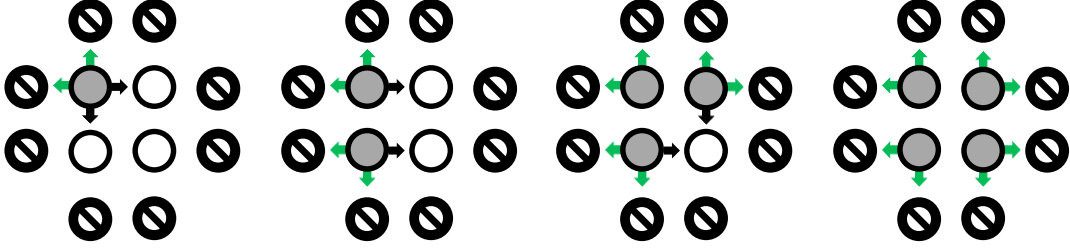


Figure 23: Progression of expected value calculation of a single robot in a 2x2 maze

To calculate the expected cost of escaping the problem space, the probability of reaching each state must be calculated. The probability of reaching state  $X_i$  is  $P[X_i]$ , which is the probability that the robot did not escape in any prior state. Equations 8 and 9 define aforementioned probabilities. The cost accrued at state  $i$  is between best and worst cases, 1 and  $i$  movement units respectively. However, the total cost of each state is the current state's cost plus the cost of reaching the previous state. State costs are defined by equations 10 and 11.

$$P[Esc(X_i)] = \frac{Exit\ Edges}{Total\ Edges} \quad (8)$$

$$P[X_i] = P[X_{i-1}] * (1 - P[Esc(X_{i-1})]) \quad (9)$$

$$M(i) = [1, i] \quad (10)$$

$$Cost(X_i) = Cost(X_{i-1}) + M(i) \quad (11)$$



Using the state probability and cost equations, an expected cost equation can be constructed. Equation 12 and table 6 illustrate the complete expected cost calculation for both best and worst case scenarios. To ensure the calculation is correct, the probabilities of all escape events is summed and verified to be 1 (Equation 13).

$$E[escape] = \sum_{i=1}^{N^2} P[X_i] * P[Esc(X_i)] * Cost(X_i) \quad (12)$$

**Table 6: Best and worst case results of the expected value calculation on a 2x2 grid**

	$P[X_i]$	$P[Esc(X_i)]$	$Cost_{best}(X_i)$	$Cost_{worst}(X_i)$	$Total_{best}$	$Total_{worst}$
$X_1$	1	.5	1	1	-	-
$X_2$	.5	.67	2	3	-	-
$X_3$	.17	.75	3	6	-	-
$X_4$	.042	1	4	10	1.71	2.67

$$P[Escape] = \sum_{i=1}^{N^2} P[X_i] * P[Esc(X_i)] = \frac{1}{2} + \frac{1}{3} + \frac{1}{8} + \frac{1}{24} = 1 \quad (13)$$

## 5.4 Single Robot Calculation

The example provided in the previous section had a known number of exit and non-exit edges at every step. The next goal is to calculate the expected number of total edges and exit edges at every step of the solution given an  $N \times N$  maze size so that the probability for escape can be effectively calculated. Future research could find an equation to describe this or a brute force method of calculation, but such information is not known at this time.

Consequently, the equation for escape probability changes from equation 8 to equation 14.

$$P[Esc(X_i)] = \frac{E_{N,n}[\text{number exit edges}]}{E_{N,n}[\text{number edges}]} \quad (14)$$

#### 5.4.1 Expected Edge Estimation

Starting with the expected total number of exploration edges, an upper and lower bound should first be found. All blocks in a robot's known region must be connected to each other by at least one side. The maximum number of edges in a continuous configuration of  $n$  same sized squares occurs when  $\exists$  a sequence of construction such that each block added to the formation increases the net sum of structure edges by 2, starting from the second block. Formula 15 gives the maximum possible number of edges given  $n$  blocks which is the upper bound for the expected number of edges. The minimum number of edges results from square block formations. Formula 16 provides the lower bound for the expected number of edges.

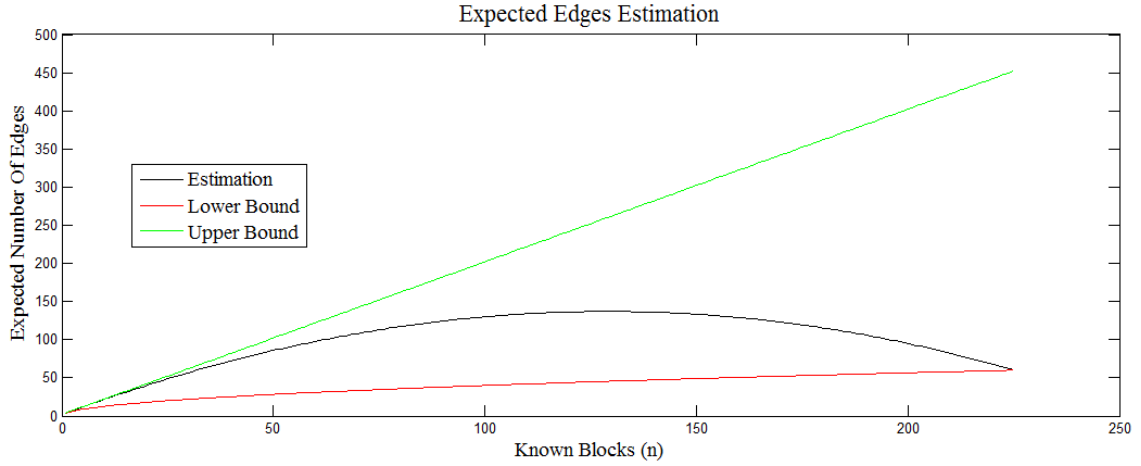
$$MaxEdges(n) = 2 + 2n \quad (15)$$

$$MinEdges(n) = n - (\sqrt{n} - 2)^2 + 4 \quad (16)$$

Now that the desired bounds have been found, they must be used to create an expected value function. Within a bounding  $N \times N$  region, as  $n$  approaches  $N^2$ , the maximum number of edges converges toward  $MinEdges(N^2)$ . This trend results from the fact that as free space within the bounded region decreases, more square construction patterns are forced to occur, eventually forming a complete  $N \times N$  grid. The trend is replicated by changing the weight each bound contributes to the function as a whole. The

expected value approximation function is given in equation 17 with its results shown in figure 24.

$$E_{N,n}[\text{num edges}] \approx \text{Edges}_N(n) = \frac{N^2 - n}{N^2} \text{MaxEdges}(n) + \frac{n}{N^2} \text{MinEdges}(n) \quad (17)$$



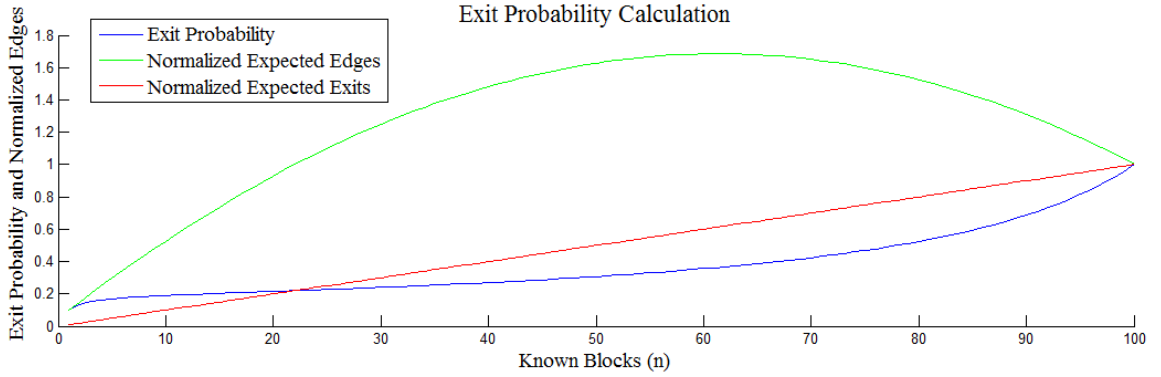
**Figure 24: Results of number of edges expected value estimation equation 17 (black). Function lays in-between upper and lower bounds and trends to the lower bound dictated by the confining maze structure.**

The expected number of exit edges is a much simpler to estimate. Each known block is treated as though its placement had been randomly chosen and the expected number of edges is added together with the others. Equation 18 depicts the expected number of exit edges given a grid size and the current number of known blocks. The equation assumes that each bordering block is an exit. However, to account for a more restrictive number of exits, the numerators of the expression must be changed to reflect the new configuration. In addition, the minimum edge limit must go to this new exit bound as well. The complication of consideration for the continuous blocks assumption is not included, as a way of effectively doing so is not known.

$$E_{N,n}[\text{number exit edges}] \approx Exits_N(n) = n \frac{4}{N^2} * 2 + n \frac{N^2 - (N - 2)^2 - 4}{N^2} * 1 \quad (18)$$

Now that the desired approximations are complete, they can be used to calculate the probability that a robot will escape at each step. Figure 25 displays the estimated escape probability for a ten by ten maze. The estimated probability serves as input to a program that calculates the expected escape cost for a single robot and any size square maze. The results of said program on a two by two maze are shown in table 4 and function as a comparison to the calculation previously performed by hand (table 7). Confidence in the estimate is strengthened resulting from the near identical solutions. Equation 19 describes the new cost calculation method implemented by the earlier program.

$$E[Work] = \sum_{i=1}^{N^2} P[X_i] * \frac{Exits_N(i)}{Edges_N(i)} * Cost(X_i) \quad (19)$$

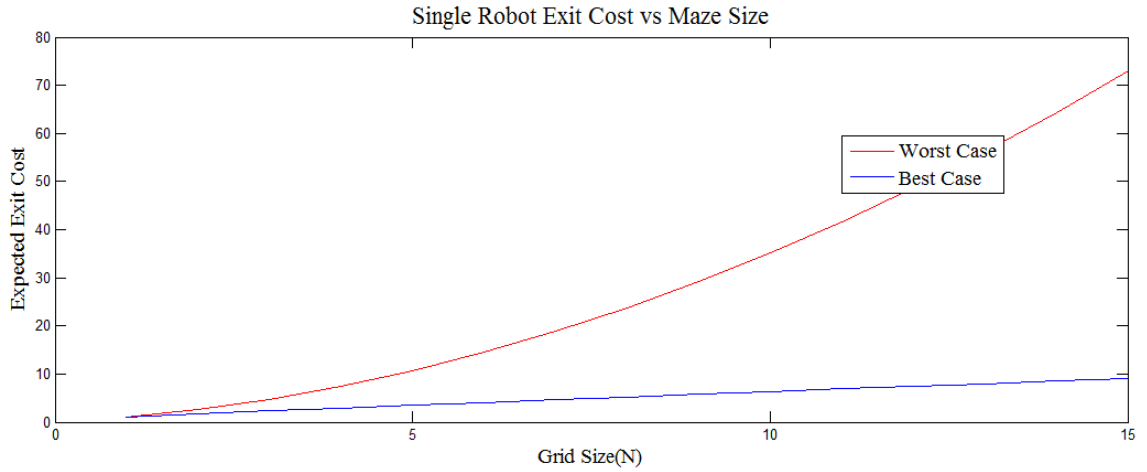


**Figure 25: Escape probability estimate (blue) using edge and exit edge expected value approximations.**

**Table 7: Best and worst case expected values of a 2x2 grid using the per step expected number of edges and exits for exit probability estimation**

	$P[X_i]$	$P[Esc(X_i)]$	$Cost_{best}(X_i)$	$Cost_{worst}(X_i)$	$Total_{best}$	$Total_{worst}$
$X_1$	1	.5	1	1	-	-
$X_2$	.5	.69	2	3	-	-
$X_3$	.16	.83	3	6	-	-
$X_4$	.026	1	4	10	1.68	2.57

Before moving onto a more complex solution, equation 19 will be used to observe the cost pattern as  $N$  increases. Both best and worst case expected escape cost calculations for mazes with  $N$  ranging from 1 to 15 are shown in figure 26.



**Figure 26: Worst (red) and best (blue) case expected exit cost/time values over a range of NxN maze grid sizes.**

#### 5.4.2 Maze Complexity Coefficient

Although upper and lower bounds for maze completion cost and time are critical for expected value estimation, it must be known what dictates cost estimations that lie between these bounds. More specifically, what influences the average movement length

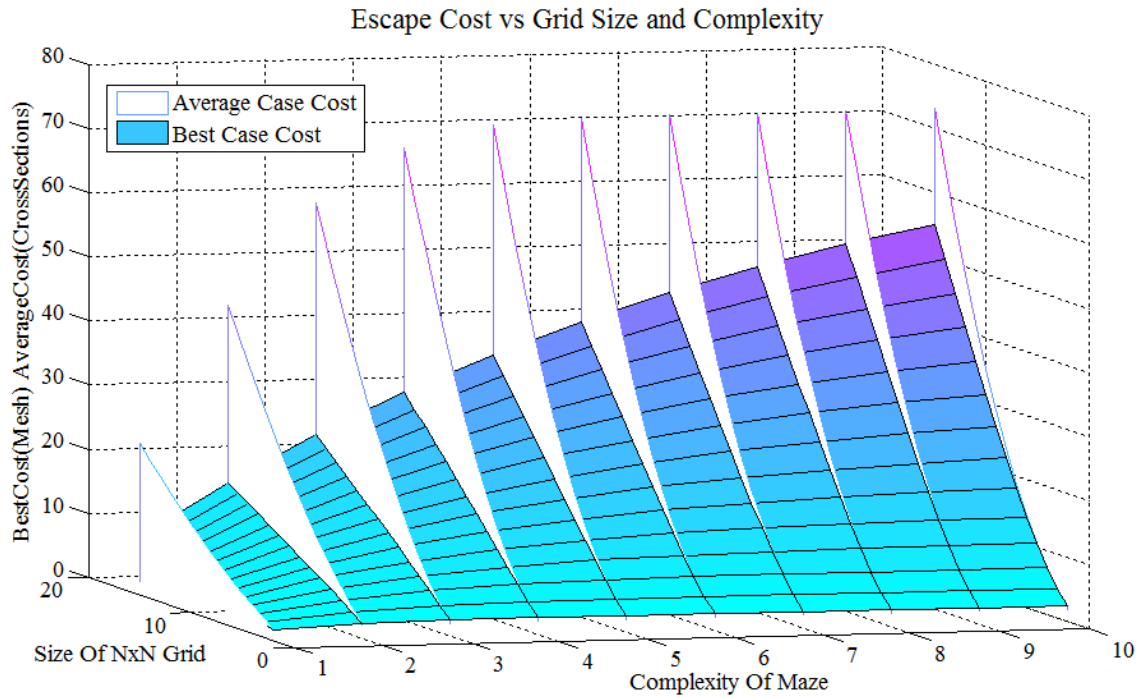
required to explore an unknown block given  $i$  currently known blocks? The answer is maze complexity, how much is physical movement impeded by the maze structure. It is desired to know, given a start and destination location pair, the minimum travel distance between the two. Given any maze structure, its complexity can be quantified by equation 21 where  $B$  is the set of blocks within the maze. At each step, the coordinate system distance between the start and destination blocks is multiplied by  $\varphi$ . This effectively inflates travel costs to represent the resistance made by maze obstacles. Phi's minimum is 1 indicating that no obstacles are present in the maze while its maximum is unknown.

$$\text{Number of Block Pairs} = \sum_{b_i=b_0}^B \sum_{b_j=b_{i+1}}^{B-(b_0 \rightarrow b_i)} 1 = \sum_{k=1}^{N^2-1} k = \frac{(N^2-1)N^2}{2} = P(N) \quad (20)$$

$$\text{MazeComplexity} = \frac{1}{P(N)} \sum_{b_i=b_0}^B \sum_{b_j=b_{i+1}}^{B-(b_0 \rightarrow b_i)} \frac{\text{MinTravelDistance}(b_i, b_j)}{\text{CoordinateSystemDistance}(b_i, b_j)} = \varphi \quad (21)$$

The exact start and destination block locations are still unknown at each step; consequently, phi will simply be used to vary the average case. The average case being at least  $\sqrt{n}$  moves must be made to explore at step  $X_n$ . Figure 27 shows the distribution of average and best case costs over a range of maze sizes when different phi values are applied. As the complexity of the maze increases, both best and average cases converge to the worst case cost. This puts the realistic range of phi values to lie between 1 and  $M(N)$ , where  $M(N)$  is the minimum value of phi that causes the average case to nearly reach the worst case. A better definition would be the first value of phi that causes the cost increase to fall beneath a threshold, as increasing phi while holding grid size

constant results in a monotonically increasing sequence. This information will be used to determine optimal system configurations for mazes of varying complexity.



**Figure 27: Relationship between maze size and complexity with respect to average and best case expected exit costs/times**

## 5.5 Multiple Robot Complications

Since the method used to calculate expected exit cost is derived from the viewpoint of the individual, only the cost function will be changed to account for the effects of communication and coordination. As described in earlier, spatial queries substitute communication cost for movement cost when exploring a new unknown cell. A query's success depends on the collective knowledge of the system. If the system does not know about the desired cell, then any cost accrued from attempting to query the data was spent

without benefit. There is a probability that a query will succeed and we must determine what this is at a given state. The area that the known system has in common with a robot's interest region versus the size of the interest region is directly related to the probability of query success, as stated by equation 6 from section 5.2.1. However, since the information required to evaluate equation 6 is not available. Therefore, a method of query success probability estimation must be developed.

### **5.5.1 Query Success Probability**

The probability that a randomly chosen cell is contained in the system's collective known region is the size of the union of each robot's known regions over the size of the maze. The size of this region is dependent on two things, the amount of redundant exploration and the total number of physical explorations by the system.

Now that unknown cell exploration can be performed by communication query, the total number of known cells in a robot's state does not accurately represent the amount of unique work a robot has performed. Hence, an additional state variable  $m$  is required to record the number of known blocks that were explored by physical movement. This state variable can be used to make an inference about the number of unique blocks known to the system. A crude estimate is  $m * R$  where  $R$  is the number of robots in the system. Redundant exploratory work reduces the total number of known cells, therefore, to improve this estimate, it must be taken into account.

Distributed optimal path selection used in the system's solution ensures that two robots will never explore the same unknown edge unless it is the only option. Consequently, redundant exploration only results from a condensed starting distribution. A condensed starting distribution occurs when multiple robots start in the same location.



If there are less unique routes than robots, redundant physical exploratory work is unavoidable. The total number of unique starting locations over the number of robots provides a ratio that describes the chance of unavoidable redundant exploration.

To produce the probability of query success, the size of known regions that are disjoint from the current robot's known regions must be estimated. Both the amount of redundant work and the number of successful queries contribute to this estimate. The number of explorations performed through query indicates a reduction in the amount of unique information contained in the exterior system without a contribution to the amount of physical work performed by the system. Simply, the more a robot completes queries successfully, the less likely subsequent queries will succeed without physical effort. If the latter were not true, every robot in the solution would move briefly, and then wait for everyone else to find the exit. State variable  $q$  is added to record the number of successful queries.

Equation 22 puts both new state variables together to produce an estimate of query success probability. The expression represents the probability that a randomly chosen block not included in the current robot's known region lies in the system's set of known blocks. A logically sound method for modeling the characteristic of unavoidable redundant exploration is not currently known and thus an assumption of evenly distributed starting locations is used.

$$Pr[QuerySuccess] = \frac{(R - 1)m - q}{N^2 - m - q} \quad (22)$$

### 5.5.2 Query Attempt Expected Cost

If any region bordering the current robot's known region is known by the system, then one of the possible queries will succeed. When it is decided to attempt a communication based exploration, the behavior of a robot is to query regions until success is met or every bordering region has been attempted, constituting failure. Using this logic and the calculated probability of a single query success, the expected cost of a communication based exploration attempt is reached. The cost to send and process a single query locally is given by the variable  $C_q$ . However, it must be inferred that if we are making a query,  $R$  other robots are making queries. No assumption is made about the timing of these externally originating queries, although the cost accrued locally to process them is allocated here. Equation 23 denoting the cost of a single query effectively takes into account an even piece of the total work required to execute queries over the distributed system. The expected communication exploration cost calculation must use the number of bordering regions; equations 24 and 25 utilize the expected number of edges accordingly.

$$Cost(Query) = C_q R = C(Q) \quad (23)$$

$$E_{N,r}[ExhaustiveQueryCost] = \sum_{i=0}^{E_{N,i}[number\ edges]} C(Q)(1 - P_r[QuerySuccess])^i \quad (24)$$

$$E_{N,n}[number\ edges] = Edges_N(n) \quad x = 1 - P_r[QuerySuccess]$$

$$E_{N,n}[ExhaustiveQueryCost] = C(Q) \frac{1 - x^{Edges_N(n)}}{1 - x} = E_{N,n}[C_q] \quad (25)$$

If every query fails, a new block must still be explored, so a physical move is made. The probability that the communication exploration is unsuccessful and the cost of a

physical move must be included in the expected cost calculation. This probability is shown in equation 26. Equation 28 provides the full communication exploration attempt cost definition.

$$P_r[CF] = (1 - P_r[QuerySuccess])^{E_{N,n}[number\ edges]} \quad (26)$$

$$P_r[CS] = 1 - P_r[CommFail] \quad (27)$$

$$E_{N,n}[CommAttemptCost] = P_r[CS] * E_{N,n}[C_q] + P_r[CF] * (E_{N,n}[C_q] + M(n)) \quad (28)$$

With the intentions of minimizing total exit cost, an exhaustive communication attempt or a sole physical exploration must be attempted at each step. The decision is based on which possibility has the lower expected cost. The expected communication cost is dependent on the state variables  $m$ ,  $q$ , and  $n$  while movement cost is dependent on  $n$ . If movement is selected, state variable  $m$  is incremented by one and its cost and time is added to calculation counters. Otherwise, state  $m$  is incremented by the product of one and the probability that all queries failed,  $q$  is incremented by the product of one and the probability that at least one query succeeded, and the expected cost and time are added to state counters. This procedure is described with pseudo-code in figure 28.

```

if(queryAttemptCost < stepExplorationCost)
    q = q + 1 * pAnyQuerySucceeded;
    x = x + 1 * pEveryQueryFailed;
    cost_Xn = cost_Xn + queryAttemptCost;
    time_Xn = time_Xn + pEveryQueryFailed * stepExplorationCost;
else
    x = x + 1;
    cost_Xn = cost_Xn + stepExplorationCost;
    time_Xn = time_Xn + stepExplorationCost;
end

```

**Figure 28: Expected cost of exploration options dictate which option is chosen. Logic contributes to expected cost/time calculation**

### 5.5.3 Physical Exit Cost

When an exit is found through a query, extra effort is required for the robot to physically reach that exit. To account for this extra time and cost, the product of exit probability, step probability, and movement cost is added to the total exit cost and time. This cost does not affect the cost of reaching later steps as it is only added when an exit is found. Equation 29 provides a definition for exit cost.

$$ExitCost(X_i) = P[X_i] * P[Esc(X_i)] * M(i) \quad (29)$$

## 5.6 Results

Now that other complications have been accounted for, the following calculations will include all variables considered in this research. These variables are listed below.

- $N$  – size of grid width and height
- $R$  – number of robots
- $C_m$  – cost of physically moving the robot one block
- $C_q$  – cost of a single spatial data query

The goal is to find the optimal number of robots given a maze size and cost configuration. The optimal number of robots occurs when the maze exit time becomes minimal. Maze exit time is derived from the state variable  $m$ , reflecting the time spent on physical movement. The time spent on communication is thought to be negligible in reference to movement time and is therefore excluded from the exit time summation.

Equations 30 through 32 define the final calculation method for expected exit cost and time.

$$E[Work] = \sum_{i=1}^{N^2} \left[ P[X_i] * \frac{E_{N,i}[\text{number exit edges}]}{E_{N,i}[\text{number edges}]} * Cost(X_i) + ExitCost(X_i) \right] \quad (30)$$

$$Cost(X_i) = Cost(X_{i-1}) + Cost(i) \quad (31)$$

$$Cost(i) = Min(E_{N,i}[\text{CommAttemptCost}], M(i)) \quad (32)$$

### 5.6.1 Maze Size vs. Number of Robots

First, the number of robots with respect to grid sizes will be calculated while holding both movement and communication costs constant. Figure 29 is a three-dimensional graph of all exit costs and times given different grid sizes and robots each iterated over a value range.

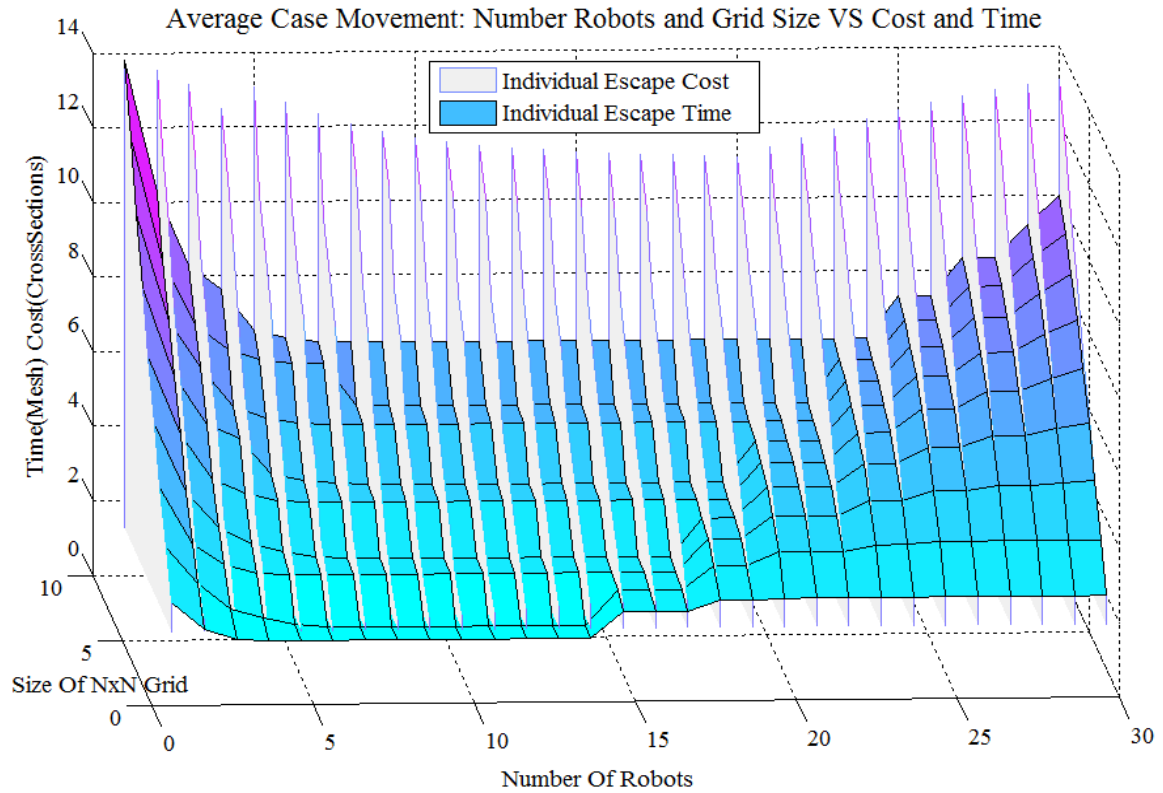


Figure 29: Time dependence on the number of robots used to solve a maze. Varies with grid size.

The initial trend with respect to robot number over all grid sizes is an exponential decay of exit time while individual exit cost decreases in a gradual linear pattern. As the number of robots continues to increase, a point is reached where exit time begins to increase towards the limit of the single robot exit time. Formula 33 represents a fitting pattern for this exit time trend, where  $t_1$  is the exit time of a lone robot,  $t_m$  is the minimum exit time of any robot configuration, and  $R_0$  is the first number of robots that triggers the second trend segment. Both  $\mu_1$  and  $\mu_2$  determine the rate of exponential change and are dependent on grid size. It is observed that there are ranges of robots for each grid size that result in minimum exit times. Table 8 provides these exact ranges for the cost configuration  $C_m = 1$  and  $C_q = .1$ . The staggered cost increases and minimum robot ranges is due to the varied accuracy of the edge estimates used over the grid size range.

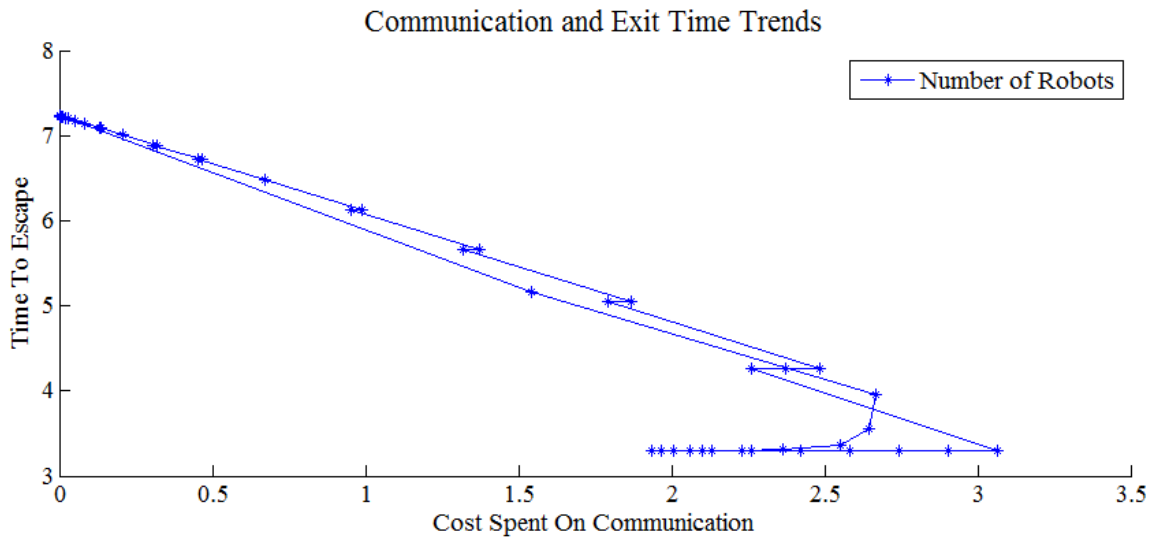
$$ExitTime_N(R) = \max\left(t_1 e^{\frac{R}{\mu_1}}, t_m\right) + \max\left(0, (t_1 - t_m) \left(1 - e^{\frac{-R+R_0}{\mu_2}}\right)\right) \quad (33)$$

**Table 8: Minimum exit times and the ranges of robots that are able to achieve those times**

Grid Size ( $N$ )	Min Exit Time	Range of Robots
2	1.76	(4,11)
3	2.13	(8,14)
4	3.09	(7,17)
5	3.41	(10,17)
6	4.50	(9,19)
7	4.81	(12,19)
8	6.02	(10,22)
9	6.34	(13,22)
10	7.64	(12,24)

Looking into the cause of the aforementioned exit time trends, the cost spent on communication is compared against time of exit. Figure 30 illustrates the relation between the two, the number of robots changing communication cost and query success probability. The figure shows that at the beginning, adding more robots will increase communication cost expenditures while significantly reducing escape time. There is a point where adding more robots starts to decrease communication cost while continuing to decrease escape time. This is due to the fact that many more attempted queries succeed; thereby reducing the total communication cost even though individual query attempts are more expensive. A later moment in the trend shows that eventually the increase of single communication cost outweighs the incurred benefits to query success probability. When this happens, robots find it more cost effective to physically explore

their surroundings verses attempting queries, producing an increase in maze exit time. These exact points of pattern change are dependent on the base communication cost and the size of the maze. Both correlate with an accelerated moment placement with respect to the number of robots added to the system.

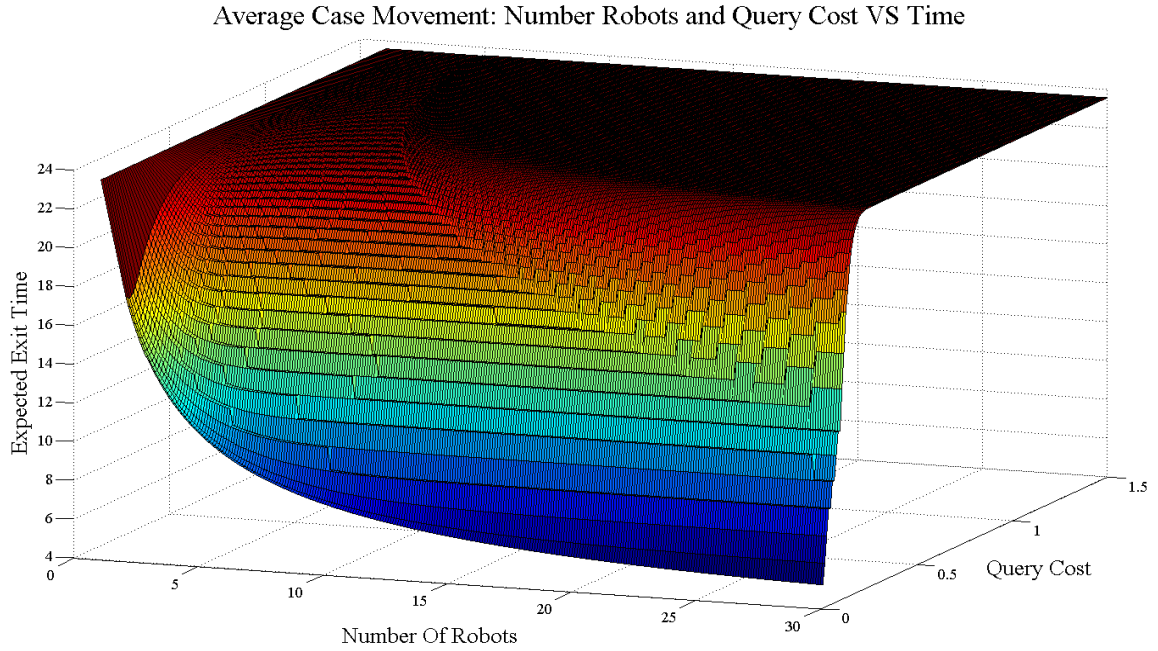


**Figure 30: Exit time trend changes as more robots are added to the solution**

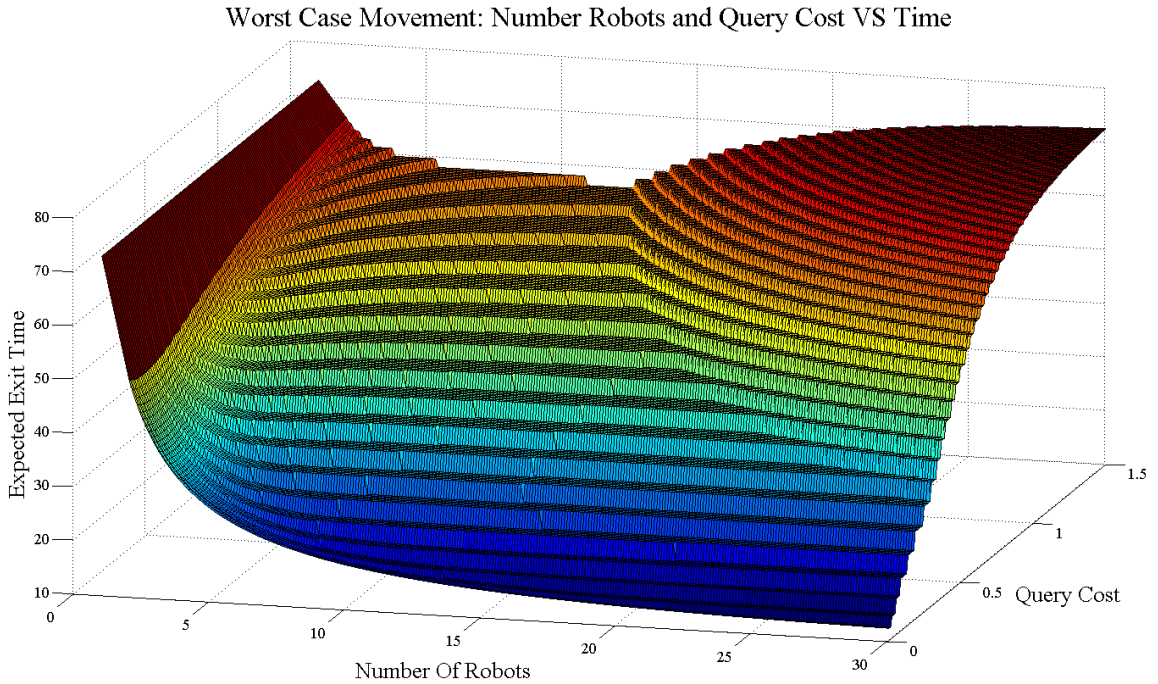
### 5.6.2 Communication Cost vs. Number of Robots

Now that the pattern of exit time with respect to grid size and number of robots is known, the effect of communication cost variation is desired. The following calculations result from holding grid size constant while the values of communication cost and number of robots are iterating over a specified range. Figure 31 and 32 display expected exit time values give a 15 by 15 grid. Figure 32 assumes the worst case step travel distance while figure 31 assumes an average travel distance. Where the previous calculation held communication query cost constant at ten percent of a single block movement, the current calculations iterate over a range of one to one hundred and fifty percent.





**Figure 31: Dependence of expected exit time on a varied query cost with respect to the number of robots. Average movement exploration costs are assumed.**

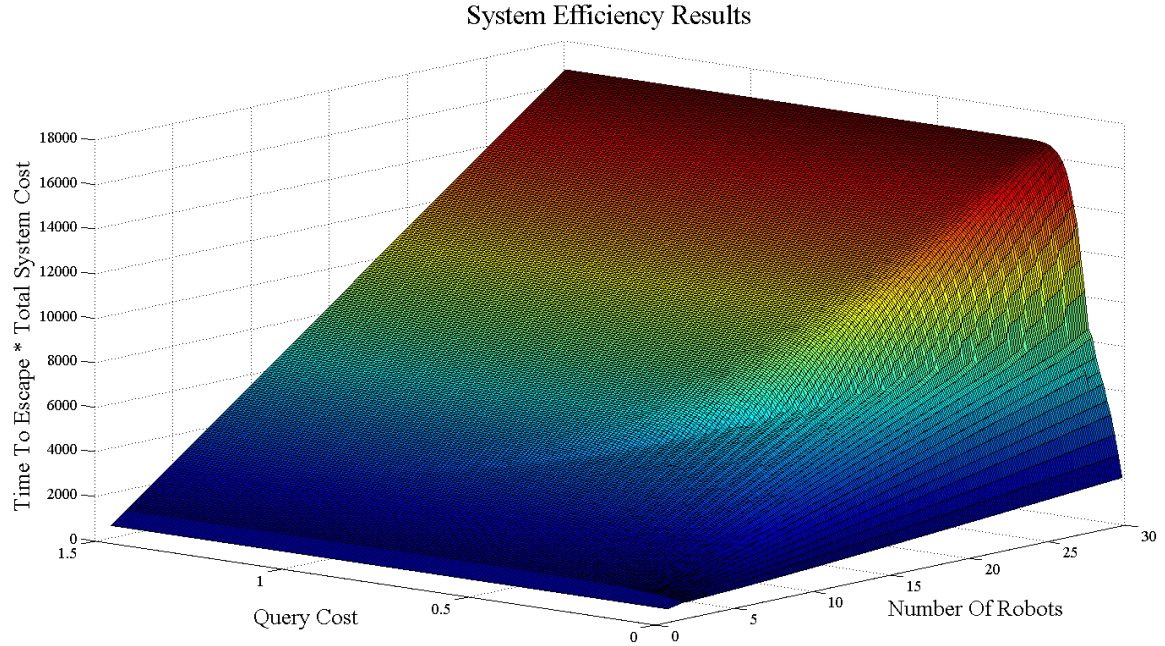


**Figure 32: Dependence of expected exit time on a varied query cost with respect to the number of robots. Worst case movement exploration costs are assumed.**

It becomes clear that query cost and the maximum number of robots able to benefit an exit solution are negatively correlated. Meaning, as query cost is increased, the maximum number of beneficial robots decreases. As mentioned previously, when too many robots are added to the system, it becomes more cost efficient to physically explore than to query. Decreasing the base query cost that is scaled by the number of the robots in the system, allows for more robots to be effectively utilized before optimal cost and therefore exit times are adversely impacted.

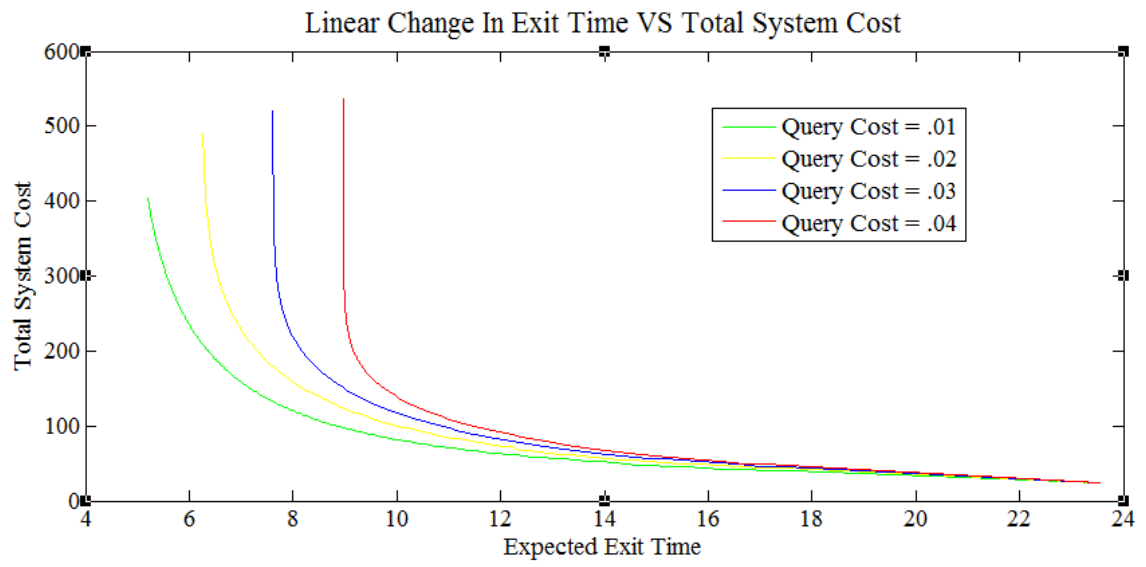
There is an interesting difference present in the crowding effect distributions of the average verses the worse movement case. The number of robots able to benefit the solution seems to be greater in the worst case assumed calculation. This is counter intuitive since querying is much more beneficial to the worst case solution, hence it may be thought that variance in its cost would have a greater adverse impact on the solution. However, the maximum estimated exit time is far greater in the worst case, and so it takes more crowding for the increased cost to become greater than any movement possibility. The information to extract from the average and worst case distribution difference is, the less total travel distance required, the more impact communication cost has on solution scaling.

It might be desired to find the most cost efficient number of robots to collaboratively escape a maze. Adding a robot will most likely increase the total cost of the system but may also reduce exit time. The most efficient number of robots for a given configuration will have the minimum product of total system escape cost and escape time. Figure 33 provides the cost benefit distribution described.



**Figure 33: Reflection of the rate of change in efficiency with respect to query cost and the number of robots in a solution configuration**

Unfortunately, the best performance with respect to total system cost is achieved with a single robot over all valid query costs. As a result, the points of most efficient performance have the slowest maze exit times. When analyzing total system cost with respect to a linear change in exit time, it is observed that exponential system cost increase is required for linear exit time decrease. This correlation is shown by figure 34.



**Figure 34: Exponential increase in cost with respect to a linear change in time scale.**

## CHAPTER 6

### CONCLUSIONS AND RECOMMENDATIONS

With respect to the distributed query technology, the message passing policy of optimistic replication through broadcast provides an optimal method of graph traversal over a mesh network. It was found that without filtering batches of messages, the total number produced per query at a given node is of exponential order. Fortunately, filtering and merging of like query messages at the node level decreased the order of messages generated to a constant, transforming optimistic replication into a practical policy for distributed spatial query systems. The filtering and merging methods used to obtain the change in performance rely on the information gained by the query completion polygon. Therefore, distributed applications that do not explicitly benefit from the maintenance of coverage polygons may gain implicit benefit through its use. Simulation based utilization and performance testing of the query system discovered it to be a feasible and scalable method of spatial communication. Additional testing is required to produce accurate performance characteristics in terms of required bandwidth per query. Implementation of the designed system using physical routers with mesh network firmware is greatly recommended.

The simulation program proved essential for the process of developing and testing all autonomous system algorithms and behaviors. It was verified that the method for minimal path calculation through modified graph traversal correctly created optimal exit paths for all known exits. Navigation point calculation provided waypoints that described key points of a minimal exit path. Exit gate stabilization eliminated all solution path selection looping over a stable map dataset. Regions selected for distributed spatial

query effectively described areas where additional data would impact solution path. It was found that a robot only queried for data that was essential to its own exit solution, thereby reducing communication cost to a minimal amount. Data returned from autonomous queries helped reduce redundant exploration and was a key part of decreasing exit time. Optimal path selection effectively distributed robots such that all exploration was done with as little redundancy as possible. The combination of intelligent query requests and distributed optimal path selection, resulted in a minimum amount of redundant exploration work. The control algorithm stabilized robots on their selected navigation plan while effectively avoiding obstacles. Any major control oscillation was due to toggling of selected paths. Future work could include a PID controller in order to eliminate minor control oscillations. The developed system effectively solved all mazes tested and utilized all participating robots while minimizing solution cost. Performance and correctness testing using existing map building and location estimation methods is desired for future work. The required changes for a system accounting for location and data error bounds are significant and may prove to be extremely difficult. However, once these changes are made, it is recommended that the system be implemented through physical devices.

The expected cost and time calculations showed that multiple robots cooperatively solving a maze can drastically decrease the total solving time and cost of escape per robot. It was found that given a maze size and communication cost ratio, there exists a range of participating robots that attain a minimum expected exit time. After too many robots are added, the exit time starts to converge back to the maximum exit time. This is due to the fact that the operating goal of the solving system is to exit with minimal cost.

Active robots attempt to utilize communication only when it could benefit the solution time. If this were not the case, adding robots to a system would only continue to converge the expected exit time to the minimum possible exit time. However, as robots are added, system cost per robot increases toward an unbounded limit.

Cost benefit analysis revealed that an exponential system cost increase was required for linear expected solution time decreases. There exists a range where linear speed-up increases only require linear solution cost increases with a slope coefficient dependent on base communication cost. Therefore, in energy cost limited solutions that have strict time requirements, efficient communication hardware is extremely beneficial to design a system that performs adequately. One major complication that was left out of the expected value calculations was shared exit conditions. Once a robot has found an exit, that exit location may be shared with all other robots. The reason that the complication was not included in the previous calculations is that what a robot can do with another's exit location is unknown. The path between that robot and the exit may be unknown and the expression for the expected cost of finding a connecting path is thus far undefined. This is something that could be addressed in future research.

## REFERENCES

- [1] Jorge Cortes and Francesco Bullo, "Coordination and Geometric Optimization via Distributed Dynamical Systems," 2006
- [2] Wolfram Burgard, Mark Moorsy, Cyrill Stachniss, and Frank Schneidery, "Coordinated Multi-Robot Exploration," 2005
- [3] Benjamin Tovar, Steven M. LaValle, and Rafael Murrieta, "Optimal Navigation and Object Finding without Geometric Maps or Localization"
- [4] Giorgio Cannata, and Antonio Sgorbissa, "A Minimalist Algorithm for Multirobot Continuous Coverage"
- [5] Benjamin Stewart, Jonathan Ko, Dieter Fox, and Kurt Konolige, "The Revisiting Problem in Mobile Robot Map Building: A Hierarchical Bayesian Approach"
- [6] Jared Napora, "Implementation, Evaluation, and Applications of Mobile Mesh Networks for Platforms in Motion," 2009
- [7] Roger Zimmermann, WeiShinnKu, and WeiCheng Chu, "Efficient Query Routing in Distributed Spatial Databases"
- [8] Nora Ayanian, and Vijay Kumar, "Decentralized Feedback Controllers for Multiagent Teams in Environments With Obstacles"
- [9] Steven M. LaValle, "Planning Algorithms," section 8.4.3 Optimal Navigation Functions, 2006
- [10] Vassilis Varveropoulos, "Robot Localization and Map Construction Using Sonar Data"
- [11] John Karvounis, "Theory, Design, and Implementation of Landmark Promotion Cooperative Simultaneous Localization and Mapping," 2011
- [12] Hugh Durrant-Whyte and Tim Bailey, "Simultaneous Localization and Mapping (SLAM): Part I The Essential Algorithms"
- [13] Patrick Beeson, Joseph Modayil, and Benjamin Kuipers, "Factoring the mapping problem: Mobile robot map-building in the Hybrid Spatial Semantic Hierarchy," 2010
- [14] Sebastian Thrun, "Robotic Mapping: A Survey," 2002
- [15] Sanjiv Kapoor, Sachindra N. Maheshwari, Joseph S. B. Mitchell, "An Efficient Algorithms for Euclidean Shortest Paths Among Polygonal Obstacles in the Plane"



- [16] Stephen R. Lindemann and Steven M. LaValle, “Smoothly Blending Vector Fields for Global Robot Navigation”
- [17] Ronnie Johansson, “Intelligent Motion Planning for a Multi-Robot System”
- [18] J.A. Rogge, D. Aeyels, “Multi-robot coverage to locate fixed targets using formation structures”