UNIVERSITY OF CALIFORNIA
RIVERSIDE

Multi Robot Cooperation Based on Auction Theory With a Value Map

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Angello Patricio Pozo

December 2011

Thesis Committee:

Dr. Bir Bhanu , Chairperson
Dr. Christian Shelton
Dr. Gerardo Beni

The Thesis of Angello Patricio Pozo is approved:

_____

_____

_____

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to Thank Dr. Bir Bhanu for his support on this work. Additionally, I thank Dhananjay Ipparithi for his help throughout the project. Vincent and Amirali, for their coding help. Also Ben Anderson for letting me ask random code questions throughout these months.

To my parents and family for all their support.

ABSTRACT OF THE THESIS

Multi Robot Cooperation Based on Auction Theory With a Value Map

by

Angello Patricio Pozo

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, December 2011
Dr. Bir Bhanu , Chairperson

The goal is to create a robotic search party that autonomously coordinates tasks to efficiently find the target. The induced coordination minimizes the communication, computation, and distance traveled while maximizing area covered. Each robot in our system has a unique combination of sensors and abilities. These include movement speed, localization accuracy, and computational abilities. The Market based algorithm allows the robot to achieve the above goals effectively. Allowing each robot to exchange tasks in an effort to maximize their unique utilities. With the addition of a value map as a representation of what areas have been observed, the robots have a driving force to move to unseen locations due to the increased value. Each robot maintains its own version of the map until it encounters another robot and share value maps. The combination allows each robot to understand what areas have been observed, hence limiting possible locations to move to. When robots meet, the algorithm pushes them away with the second path method. The robots divert themselves in an effort to maximize the area searched. Our system is distributed with no central agent with full knowledge or control of any system. Reducing the effect of robot failure to the system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Utilizing multiple robots as a search party has caught the attention of many researchers. Some have used formal strategies, where the robots maintain a formation as they explore the area. Others have utilized behavior methods [1] where specific reactions based on the messages created from each robot in the system. Others like to mimic nature and apply an Ant colony mentality into the robots. Where each robot leaves a path that other robots may follow. There also exist evolutionary [3] methodologies where exploration, movement, localization, parameters are altered based on the experience of the robot system.

The use of multiple robots has become ever present in our real world. The mars rovers could have been given a general task of exploring an area, rather than the current move wait system employed today. Harnessing the capabilities of the robots, to accomplish a goal with little to no human interaction is important.

Another driver for multiple robot utilization, is surveillance. Instead of having a security guard wondering around, a robot system can be employed to observe an area. Or it can be assigned

to intercept and observe a specific spot on a map. For example, if the robot is doing its normal job, rather than have the operation tell which robot to move somewhere, the general task is given to each robot. The task reverberates through the Market system and a winning robot is assigned the task.

This study encapsulates the use of a distributed robot system to find, track, and search for a target. The results are compared with Particle swarm Optimization (PSO) and Modified Particle Swarm Optimization (MPSO). The study is begins with chapter 2 where older search algorithms are described. Their respective benefits and faults are addressed. Chapter 3 describes the Market algorithm in detail. Chapter 4 describes the test setup and results that demonstrate the capabilities of the Market algorithm. Lastly, chapter 5 includes the Future improvements that can be implanted by another person.

Chapter 2

Related Work

Working with multiple robots, the ultimate goal is to explore and complete complex tasks effectively. Their use can be applied to soccer bots [5] [14] to coordinate actions between the team. They can be used to increase localization accuracy for a system of robots [7] [15]. Similarly if a system of robots with cameras can improve visual Simultaneous localization and mapping (SLAM) [6]. By utilizing multiple robots in any system, it is expected to perform better than single robots. There are many algorithms that are used to improve the usefulness of multiple robots. These include Game theory, Swarm Optimization, Evolutionary algorithms, and other Market based strategies.

2.1 Game Theory

Game theory allows for cooperation with limited communication requirements. The information is limited to informing each robot that stye are in a game, requirements, and updated rules and specifications. Once done, each robot may report their decision and take actions based

on the result. The cooperation stems from the desire of each agent wanting to get the best 'deal' for itself. This is achieved by having each player compute its best course of action, based on the actions of the other player. This is best characterized by the prisoners dilemma, where two suspects are separated, and asked to either confess their crimes or stay silent as shown in table 2.1. Given

Table 2.1: Prisoners Dilemma

	B stays silent	B confesses
A stays silent	A wins - B wins	A wins - B loses
A confesses	A loses - B wins	A loses - B loses

that prisoner A and B know the rules of the game, they each can cooperate and win. But if one decides to confess, he punishes the other prisoner. For the best course of action to be taken, both players must consider the actions of the other player and play the best hand.

In the above example, no communication was necessary between prisoners. But each prisoner understood that they were in a game scenario, and took the best course of action based on their previous knowledge. For robots to enter a game scenario direct communication between the players is important. In a multiple robot system, each robot must know what game scenario they are in and out of. Work has been done to show that direct communication is not required to enter a game scenario [10].

Game Theory has been used to optimize the scheduling system for a manufacturing facility [12], and to simulate a flock of robots moving through an environment [19]. Since game theory requires a robot to predict the actions of another, direct communication is not required but can be interpreted [10]. The limitation is the visual systems ability to understand the movements of

another robot. Similar to how the XBOX360 Kinect ™ camera interacts with the user.

2.2 Swarm Intelligence Algorithms

Particle Swarm Optimization (PSO) algorithms require the use of many particles/robots. Where each particle is a solution to the problem given a fitness evaluation. The fitness evaluation provides a method to identify which particles are most correct. For example if the objective is to find a single target in the image; particles are distributed around the image. Each particle evaluates a fitness for its position $\vec{x}_i$ where 'i' is the current time step. The best $\vec{x}_i$ position for a particle is $\vec{x}_s$. The $\vec{x}_i$ fitness is shared between all particles, and the best $\vec{x}_i$ position is equated as the global best position $\vec{x}_g$. Then each particle will choose a random value $\backsim \cup(0,1)$. Then utilizing user specific variables ω , ϕ_s , and ϕ_g to control behavior are specified. The resulting equations update the velocity 2.1 and position 2.2 of the particle for the next step (i+1).

$$\vec{v}_{i+1} = \omega\vec{v}_i + \phi_s r_s(\vec{x}_s - \vec{x}_i) + \phi_g r_g(\vec{x}_g - \vec{x}_i) \tag{2.1}$$

$$\vec{x}_{i+1} = \vec{x}_i + \vec{v}_i \tag{2.2}$$

Each particle/robot computes its next move $\vec{x}_{i+1}$ based on the self best $\vec{x}_s$ and global $\vec{x}_g$ positions. Those parameters effect the speed of the particle from $\vec{x}_i$ to $\vec{x}_{i+1}$. As the number of particles in the system increases, the amount of communication required grows exponentially. Therefore the particle swarm method is best applicable for a low number of robots. PSO algorithms are limit to simulations because of the communication handicap.

There exists a modified PSO (MPSO) algorithm that allows the robot to evaluate the direction of the next best PSO vector direction. Meaning, if the PSO algorithm has best vector

upwardly, if the robot came from that direction, the vector is reduced and the robot searches the immediate area for the target.

Ant Colony Optimization (ACO) is a method that mimics ant's use of pheromones as the solution to a shortest path to the target. As an ant moves through an area searching for the target a pheromone trail is left behind. As the path is used, it accumulates pheromones which means the path is desirable. One glaring issue is that the area covered is not maximized. More than one ant walk through a path, hence limiting the search capabilities. A major benefit is if the robot has multiple targets in an environment, then once the ACO algorithm has run, then a short path between all targets can be found [4].

To remedy the overlapped paths issue in the ACO algorithm, it was combined with the PSO algorithm [11]. Where the ACO aspect is used to determine what target to move to and the PSO controls the movement. The amount of pheromones determines how likely one ant will be willing to move to a new target over its current target. Allowing the robots to move around the environment while still finding a short path to the target. One major drawback is that the ACO algorithm is not dynamic. If the target location is moved, it will take time for the system to correct itself and find a new solution.

2.3 Genetic Algorithms

A Genetic Algorithm (GA), similarly to a PSO or ACO, requires a population of particles/solutions/robots. The best outcome of a GA algorithm, is a Pareto efficient solution. Those solutions are when any one parameter cannot be increased without decreasing the value of another

parameter. Each parameter is represent variables that we want to maximize. For example if we want to be as close as possible to the target, we cannot increase x or y without moving away from the target. The output is not a single solution. Rather there may be many x and y combinations that have equally good fitness values. To limit the number solutions, multiple fitness evaluations are needed. Or a solution can be manually selected from the list. But it excels in finding solutions to Multi-Objective Optimization problems. The GA algorithm permeates through possible solutions and the output is the resulting solution. Generally, the longer a GA algorithm is run the better a solution is.

A GA algorithm emulates natural selection in an effort to remove unsatisfactory solutions and keep good solutions. The definition of satisfactory and unsatisfactory is controlled by the fitness evaluation equations. A GA algorithm has two main components it uses to emulate natural selection. The first is deciding what set of possible solutions will be allowed to combine together. Some methods allow for strictly greedy methods and allow only the best solutions are allowed to combine. Other may use random or a unique heuristic method to choose who mates. If only the best are allowed to combine, the population may not survive a catastrophic change in environment. Because the parameters to survive that environment will be completely removed from the system.

The second step is to decide how the parameters will be combined. One algorithm may use the best parameter between the parts for the child solution. Others may take an average or a unique heuristic method to combine the parent parameters. Additionally, all of these methods need to some randomness as to allow diversity in the system. The mutation rate is important here because if it is low the solution may settle in a local-minima. But if the mutation is too high, then possible good solutions may be removed.

To obtain a resulting solution can be achieved after a strict set of generations, reaching a predetermined fitness, or stopping after the change between generations becomes negligible. Lastly, the program may be manually halted. If a fitness requirement is placed, then the algorithm may never stop or take a long time to active a solution.

Researchers have attempted to utilize an evolutionary algorithm in a Market Strategy methodology. Where the evolutionary algorithm is used to deal with path planning while the market strategy handles the task assignment between robots [9].

2.4 Other Market Based Methodologies

The basic model of a Market strategy requires tasks, bidders, and auctioneers, where bidders place bids on tasks at auctions. Contract net protocol outlines how to handle the communication and bidding requests to creation and evaluation of tasks [16]. This was the beginning of using a market or creating of task that are evaluated individually and shared amongst a group.

It was later expanded to allow for complex tasks that require direct coordination between robots in the M+ algorithm [2]. Tasks can be "pick up from robot" or "put down on robot" require direct communication and cooperation to complete the task. This method employs a self assigning to tasks. Every robot is informed of a task, they evaluate the task. If the task is favorable, an announcement is made to all robots in the system. If no other robot says they want it, it becomes the winner. If a secondary robot is interested, they compare values and assign a winner. This is repeated until all tasks are completed.

Similarly another method called Hoplites also addressing coupled tasks [8]. Coupled

tasks are task that require extra coordination between robots. For example, moving a bucket of water. If two robots need to push a heavy bucket of water, they need to agree on their orientation, direction, and time of pushing. So Hoplites introduced an active coordination component that allowed a robot to enact a risky move to solve the problem. It does this by creating a team plan that would increase its own 'profitability'. It then informs the teammates who do further evaluations and choose actions to take. Once all the robots are above their minimum profitability, enacts the action for all the robots.

A similar method to the Hoplites algorithm is the "Move Value Estimation for Robot Teams" (MVERT) [17]. MVERT tries to address multiple goals and task by creating a value function for exploration, target sampling and communication. This method employs a similar characterization of evaluating and comparing possible actions, but one major difference is the inclusion of a prediction of what another robot can do. In Hoplites the robots inform one another on their moves and reevaluate their move when more information is present. But in MVERT, the robots predict what the other robots do, receive those predictions and then reevaluate again. This algorithm also assumes each robot will move synchronously, hence the robots all decide on where to go, move, and then the algorithm starts again. While my Market algorithm allows for the system to work throughout the experiment regardless of the state of the robot.

2.5 Contributions

One major issue with a Market algorithm is the difficulty of addressing coupled tasks, because coupled tasks generally require constant communication between agents. A strategy like

hoplites addresses the issue but creates a informal formation of robots [8]. The method adopted here allows each robot to explore independently but provides a protocol when encountering a robot. It is triggered when the two agents create locations that are within proximity of one another. This forces them into the second path plan function which adds a level of coordination between the robots. The two robots exchange value maps, decide who owns the contention location between them, and what set of second-step-paths to consider. This allows the robot to predict and compute two steps instead of one. The robots will have different target locations that force them to move away from one another. The purpose of computing a second target location is to allow agents to accept the low utility now for a higher second bid later.

Another addition is the value map augmentation. The value map augmentation is when the robot manipulates the value of area on the map based on its need and observations. This means that if the robot perceives a possible location for the target, an estimation of the location is done and the area is increased in value. Similarly if the robot is lost, it can create an increased value around itself based on the covariance of its location. Allowing the robots to use and manipulate the map is powerful as it allows for a dynamic understanding of the world, hence when tasks are evaluated, there is more information being crunched.

Table 2.2: Comparison

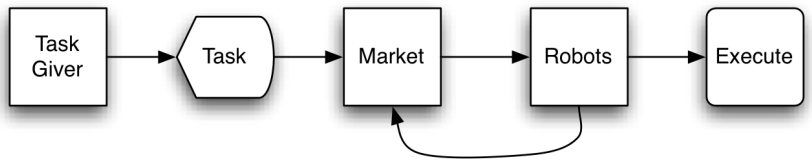
	ValueMapMarket	APF	MVERT	Hoplites	ACO	Genetic	PSO
Multi-Robot Coordination	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Computation	Low	Med	Low	Med	Low	High	Low
Path Planning	Yes	No	No	Yes	Yes/No	Yes	No
Distributed	Yes	No	No	Yes	No	Yes/No	Yes
Learning	No	No	No	No	Yes	Yes	No
Known Map	Yes/No	Yes	Yes	Yes	No	No	Yes/No
Minimize Overlap	Yes	No	Yes	Yes	maybe	Yes	No
Learning	No	No	No	No	Yes	Yes	No
Dynamic-Environment	Yes	No	Yes	No	Yes	Yes	Yes
Job Sharing	No	No	No	Yes	No	No	No
Heterogeneous Team	Yes	No	No	No	No	No	No
Physical Robots	Yes	No	No/Yes	Yes	No	No	No
Predict Team Action	No	No	Yes	No	No	No	No

Chapter 3

Technical Details

The way overall overview of how a market algorithm works is shown in Figure 3.1. The

Figure 3.1: Market Algorithm



first part requires a task giver. Tasks can be anything from find target, get object, find object, obtain stereoscopic image, localize a robot etc. These are outside task given to the market where the robots exist. The market informs every robot in the system. Each robot then does its own evaluation and creates subtasks. These subtasks are generally move command tasks. As each robot shares these subtasks, the robots move closer to completing the original task. This works because if a new task is given to the robots to obtain a stereoscopic image, then only the robots with a stereoscopic camera

can complete the task. The market informs all robots, and each individual robot will evaluate it. Some robots will evaluate to zero because they have no camera, but the few that do have the required sensor will share amongst themselves move to location subtasks.

To accomplish coordination, we are using a Market Based algorithm. The algorithm mimics a free economy where agents (businesses) must compare costs between different decisions (tasks) and choose the best decision. The trick is to create a procedure that allows the robot to distinguish between good and bad decisions. A real world business balances decisions based on cost and value.

Other market based algorithms use cost minimization and choose the cheapest task. The method proposed here tries to maximize the amount of value returned for exploring an area. Value is computed by creating a value map that represents the history of locations traversed by the robot. The Value map is periodically increased at a known frequency. A winner is distinguished not solely based on cost but on an expected utility.

The value map associates value to a location that increases with time. The value is reduced when an agent moves through the area, reducing the value of that location. The value map is a history of visited locations. The Value map is used to limit possible move locations later on because if the value is low, that means it has been observed. Therefore increasing the area observed and reduces completion time, computations, and communication costs. The Full algorithm is shown below..

Overall Market Algorithm:

1. Create Tasks

2. Broadcast/receive task

3. Filter Out Tasks

4. Evaluate cost and utility for remaining tasks

5. Place bids on task with highest marginal utility (feedback)

6. Receive/Send confirmation to winning agent

7. Attempt completion of assigned task

Each agent begins by creating a set of tasks (Task List). Tasks can be a set of move locations, map error correction requests with a location , or triangulation requests with a location from another agent. The most often created task is the move to target location task. After each agent creates a Task List, it is broadcasted to the remaining robots. The next step is to filter out tasks by evaluating requirements of each task, including the task time out and specific sensor requirements. This reduces the number of possible locations that the robot will spend computations on. While filtering, a check to see if locations are close to one another is done. If two locations are close, then the Second Path function is called. Here the robots delegate who owns the task and what to do after the contended location. At the same time, the robots share their value maps so to understand the state of the system better. From there, the robot computes realistic costs and value possibilities. Then sends this information to the bid block, which sends its bid to either itself or an external robot. Upon receiving tasks, the auction house delegates who has done a task based on the highest bidder. Because the system is a blind auction, only the auction house is aware of the bids that are received.

Consider robot X creates Task list A, and robot Y create Task List B. After computing a

bid, robot X wants to place a bid on a subset task in Task List B. Robot A will send its bid to robot Y, where it compares bids and assigns winners to robots.

In the next section 3.1 we provide an overall explanation of the algorithm. Then we provide a structure for the Value map in section 3.2. I then explain task creation in section 3.3. Followed by how filtering tasks is done in section 3.4. Continuing to the computational background in section 3.5. Then into how the robots compute their respective bids in section 3.6. Then onto how the robots address bids for tasks in section 3.7. Followed by the Second path plan portion of the algorithm in section 3.8. Localization of the robots model is in section **??**. Then I go into the communication protocol in section 3.9.

3.1 Market Strategy Overview

Figure 3.2: Market Block Diagram

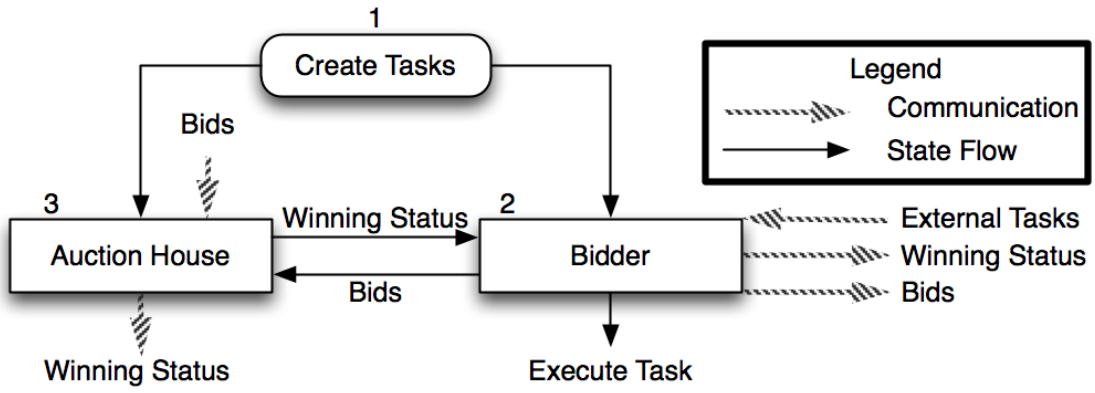


Figure 3.2 shows the Market Block diagram from the perspective of one robot. Initially

the robot creates a set of tasks. These tasks are sent to the bidding block (block 2), then forwarded to other robots. Similarly, this robot receives tasks from other robots. The bidding block then does further evaluations and will bid to an external auction and/or an internal auction. Tasks that are sent directly to the auction are owned by the auction. Meaning, it is the responsibility of the auction house to delegate who wins a task. Since each robot maintains its own auction, robots sometimes bid on external robot auction houses. Once the block has determined a winner, it sends a flag to the winning robot for the task. This can either be itself or another robot at a different location.

Figure 3.3: Bidder Block Details

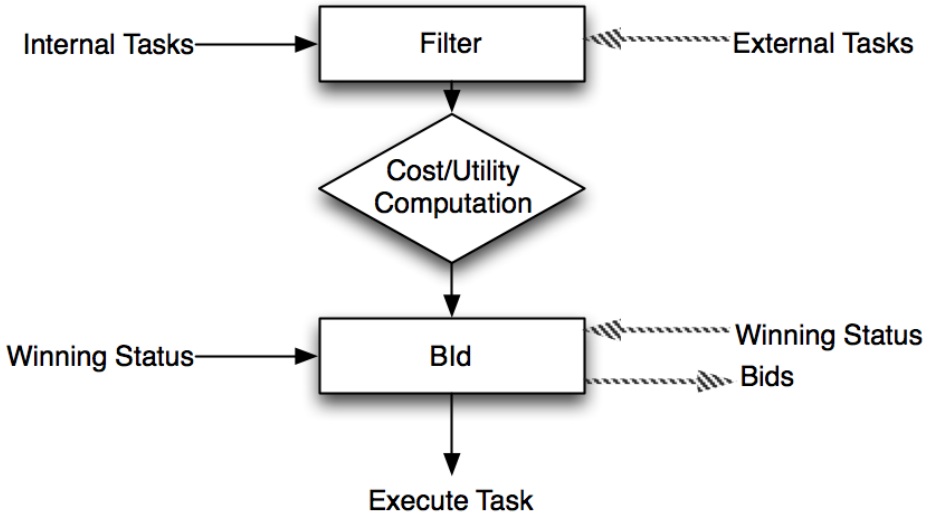


Figure 3.3 gives a detailed representation of the bidder block in Figure 3.2. In an effort to minimize computational costs, tasks are filtered to reduce the number of evaluated tasks. With a smaller subset of completable tasks, the robots compute a cost and utility. Then compute a bid and communicate with the task owners auction house. Additionally, this block also receives information

abut tasks that have been won either through the robot's auction, or external (other robots') auctions. Either from itself or from an external auction. Once it receives a winning status on a task, the robot moves to complete it.

Figure 3.4: Filter Block Details

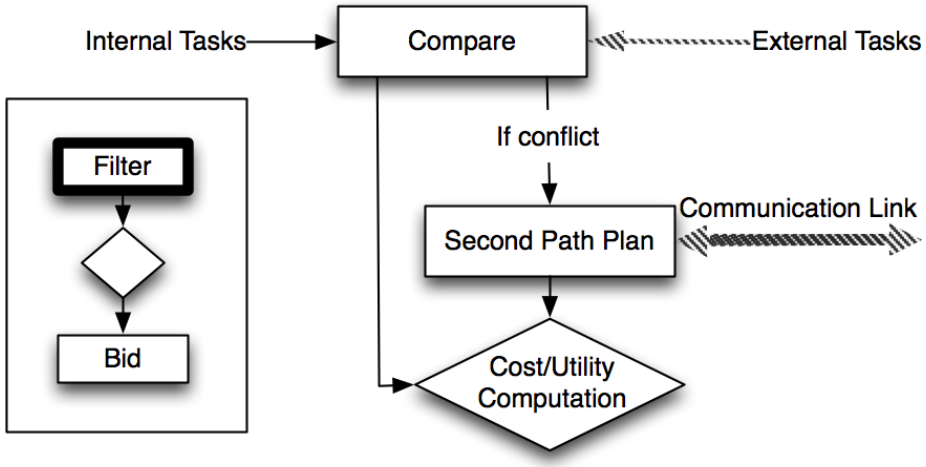


Figure 3.4 shows the procedure for the filter. The first block evaluates requirements for the task. This includes current battery power, sensor requirements, and distance to the task. If the task has a location conflict, it is passed through the second path plan block where further processing is done with another robot. The robot closest to the conflict location is declared the owner. The winning robot then computes a new set of locations from the conflict location. The last step is to pass the information into the cost and utility computation block to do the computations.

We move back into Figure 3.3 and enter the bid block resulting in Figure 3.5. Here the robot converts the cost parameters and outputs bids. It then waits for a confirmation that it won a task. This means that it is responsible for executing all tasks that it wins. The decide block has a

Figure 3.5: Bid Block Details

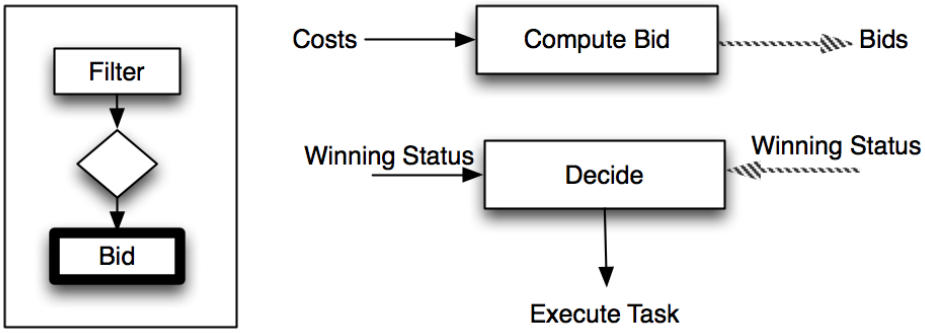


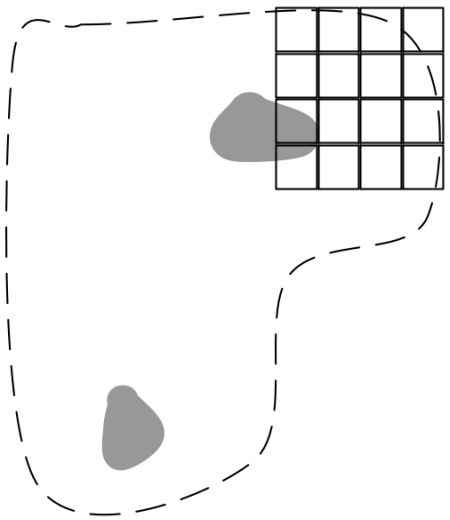
table of currently available tasks. It will choose the task with the highest amount of utility and start to complete the task. This allows the robot to have a pick of possible locations to move to.

3.2 Value Map

The value map is identical to the map of the known area except an occupancy grid is placed over the entire map (Figure 3.6). Each grid is 1x1 ft to allow a robot to completely occupy it. The Value Map is a structure with the following parameters for every grid: [X_c, Y_c, Obstacle, Time Update, Value, Visibility]. Where X_c, Y_c is the center of a specific occupancy grid in terms of the global frame. This is required because the resolution of our map is greater than that of our occupancy grid. Furthermore, if a task resides within an occupancy grid, all costs and evaluations are made to the center (X_c, Y_c) of the occupancy grid. The small difference in distance adds a little randomness to the system. The obstacle parameter is the percent of the grid that is an obstacle. Looking at Figure 3.6 parts of some of the squares contain an obstacle. Therefore its obstacle parameter will be less than 1 but greater than 0 while other locations will have the obstacle

parameter equal to 0. The Time Update parameter is the time of the last update. This becomes important later when robots exchange value maps when two values need to be compared.

Figure 3.6: Map Occupancy Grid



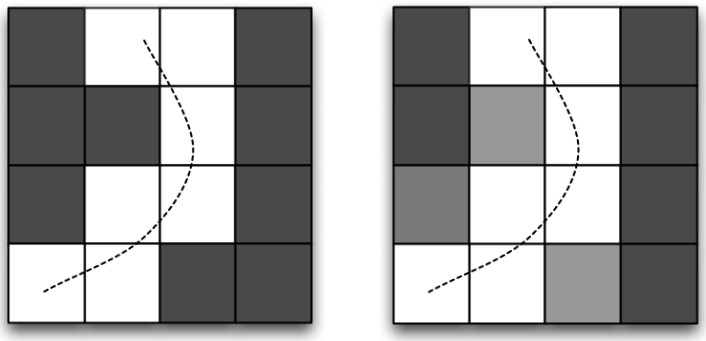
Next is the value parameter where each grid in the value map must be updated every t seconds via the value equation 3.1. The equation represents the update for any occupancy grid at (x,y).

$$Value_{i+1}(x,y) = Value_i(x,y)(1-b) + k * t \tag{3.1}$$

Where k is the slop of the gain, t is the time update interval, and b is the visibility parameter. For our implementation the t variable is updated every 10 seconds, and the b parameter is grabbed from the field of view of the camera model. Therefore for a specific grid (x,y) where a robot has no visibility, the value grows. And any location where the robot has a high visibility parameter, the value is nearly removed. The Value of any location is not allowed to be a negative number.

Lastly the visibility parameter is the percentage of the grid viewed by the robot. This

is computed from the field of view model of the camera. The larger a visibility parameter, the more value is taken away from the map. Therefore if a location on the value map grid is only partially observed, the value is not fully reduced. Figure 3.7 contrasts the difference between a static reduction (Figure 4.4a) and dynamic reduction (Figure 4.4b). The darker a grid location is, the more value it has. A static reduction is a boolean operation where if a grid is observed the



(a) Static Reduction (b) Dynamic Reduction

Figure 3.7: Static and Dynamic Value Reduction

value is equated to zero. A dynamic method is used because the robots cannot fully know that the occupancy grid has been fully observed. Hence built into the model, the robot only removes what it possibly has seen. For example if the robot only managed to get a 50% view, then it will only receive 50% of that grids value.

3.3 Creating Location/Tasks List

As robots explore the environment, they create tasks. Creating these tasks takes several sub steps shown in the "Location Task Creation Procedure" list. The procedure allows the robot to reduce the number of possible move locations to a smaller manageable subset. Subsequently, the robot needs to seed locations in the environment in an effort to discover unobserved areas.

Location Task Creation Procedure

1. Create N Task Locations within search space
2. Remove Obstacle locations from List
3. Evaluate initial cost and compute utility
4. Export N locations and location tasks.

The search space is a list of possible grids within a circle around the robot. Grabbing information associated within the grid, the robot creates a list called Location/Task list. Locations classified as obstacles within the search space are removed from the Location/Task list. Reducing the number of possible locations, computational costs, and increases the battery life of the robot. Mathematically, the search space (H_i) is defined as:

$$H_i(x,y) = r_{bound} | \{ r_{bound}^2 = X_{Ri}^2 + Y_{Ri}^2 \} \tag{3.2}$$

where i is the robot, and r_{bound} is the radius of the search space. X_{Ri} and Y_{Ri} are the center location of the robot in terms of the global frame. Every location created lies within the H_i as:

$$L_i = \{ B_i(x,y) < z | z = x^2 + y^2 = z^2 \} \tag{3.3}$$

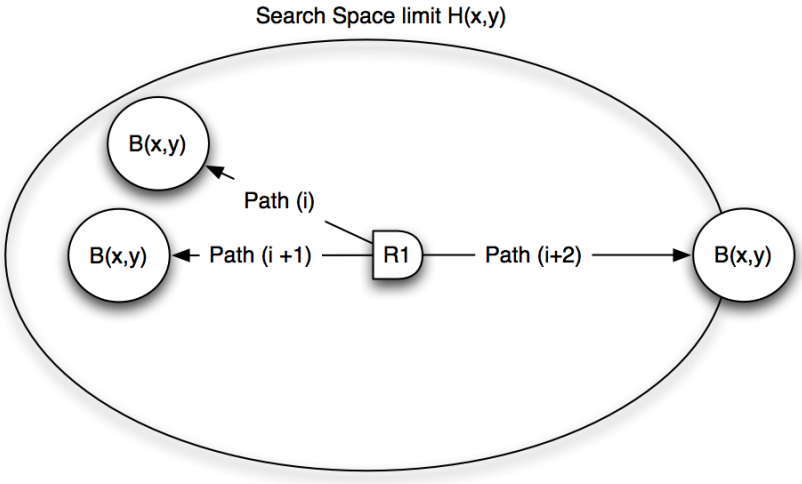
the end result is a variable L_i that represents the desired target locations B(x,y) shown in Figure 3.8 where $\{i = 1, 2...n\}$. The function L_i is a circle around (X_{Ri}, Y_{Ri}). The bound function can be altered to any shape. It is a circle for convenience. The resulting lists of L_i equations are the task locations that are broadcasted from the robot.

Several methods can be used to determine where the L_i exist can be done in several manners listed below.

Methods

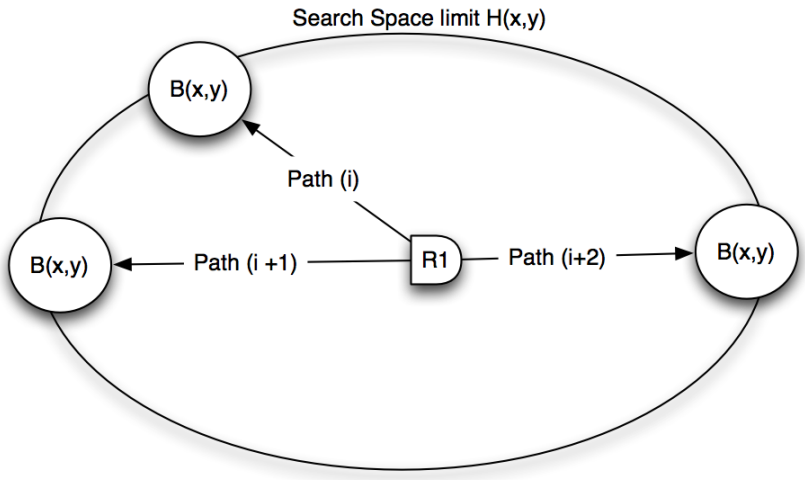
1. Greed function (Gradient Follow)
2. Random
3. Search along bound with constraints

Figure 3.8: Follow Gradient on H(x,y) Bound



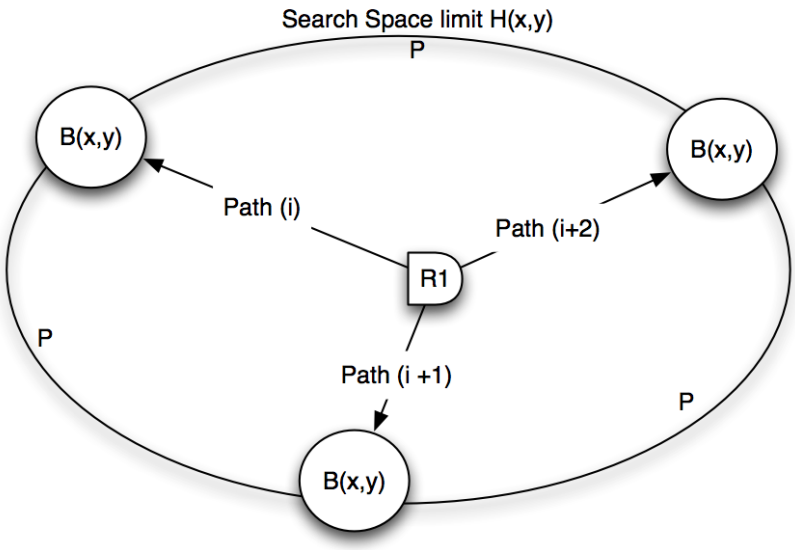
The Greedy function chooses locations along the H search space with the highest utility value (Figure 3.8). Since this method searches the entire space, it is computationally expensive. Meaning it is possible to have multiple locations close to one another that have a high value making computations redundant. This does not allow for cooperation between robots. Because if a second robot is close by, the second path plan (section 3.8 will not be initiated. Forcing the robot to work alone and miss possible cooperation instances. Missing those opportunities means the robots may explore areas already seen from one another. Thus reducing the amount of area explored and wasting precious time.

Figure 3.9: Random on H(x,y) Bound



The second method randomly chooses N location in the H search space (Figure 3.9). This method is not optimal because it may choose locations with low values, or it may create all the locations close to one another. Randomly choosing locations is not a good choice methodology given the lack of control of where the locations may be.

Figure 3.10: Constraint Bound H(x,y) Bound



The last method chooses locations with a large value and are P degrees away from one another (Figure 3.10). The value of P is the minimum allowable degree. This method allows for a spacing of locations along the search bound.

3.4 Filtering

After creating a task list and receiving tasks from other robots, they are filtered by enforcing constraints. The filter removes tasks that are costly and/or cannot be completed by the robot. Filtering takes part in three stages. The first stage removes tasks that are outside of the environment and/or within known obstacles. The second removes locations that are far away and/or have low pre-estimate utility. Lastly, a check on the battery status deters if any task is worth the effort.

The second filter requires the computation of a $LineUtility$, which is the rate of return of

completing a task. To compute $LineUtility$, the robot calculates the distance to the target location via equation 3.4. This acts as a basic initial cost estimate of moving to the location.

$$LineCost(x_L, y_L) = \sqrt{(x_R - x_L)^2 + (y_R - y_L)^2} \tag{3.4}$$

Where (x_R, y_R) is the center point of the robot and (x_L, y_L) is the target location. Each occupancy grid the line touchers is summed into $LineValue$ as an estimate of the elected value returned for completing a task.

$$LineValue = \sum_{i=1}^{n} Value(LineCost(x_L, y_L)) \tag{3.5}$$

Where n is the number of occupancy grids the line touches. Depending on the location of the robot and the state of the value map, the $LineValue$ will not be the same. Because the line may pass through an area where the value has not been reduced. Resulting in an increase in value for that $LineValue$. The computation of the $LineValue$ is the combinations of equations 3.4 and 3.5, resulting in 3.6.

$$LineUtility = LineValue / LineCost \tag{3.6}$$

The utility represents the expected rate of return if a task is to be completed. If the cost is greater than the value, the utility will be small and not be work the effort. While if the value is grater than the cost, the location has expected returns. The $LineUtility$ will generally be lower than reality because the robot will later use its field of view to grab value. Where the pre-estimate model is a line rather than a cone. And as the robot moves towards its target location, the value of the map will increase. Therefore after traversing any path, the robot should obtain more value than expected. But on occasion the robot may return a lower value. This can occur if the robot does not complete

its task. Or possibly if the robot tries to complete its task and moves through a low value area. This estimation is akin to an investor pre valuing a startup. This evaluation only allows the robot to place a possible value on a location. But because the robot invests in multiple locations, every so often, the returns will be much higher than expected: high enough such that any losses from previous investments are recouped.

The last filter takes into account the battery life left on the robot. The purpose is to allow the robots to know when to return home. Because if the robots explore an area, its best if it can return home to be fixed and recharged. The robots needs a minimum of 11 volts to operate. But during movement voltage can vary by 0.2 volts. Therefore we use 11.2 volts as a lower limit in voltage for the robot. The full battery life filter equation is in equation 3.7.

$$BatLifeFilter(\beta) = \frac{1}{1 + e^{-\alpha(\beta - \tau)}} \tag{3.7}$$

$$u(x) = \begin{cases} 1 & \text{if } BatLifeFilter \geq \eta \\ 0 & \text{if } BatLifeFilter < \eta \end{cases} \tag{3.8}$$

Where β is the current voltage of the robot. Equation 3.7 is an S curve function controlled by α and τ that alter slope and center of the function respectively. The $u(x)$ function in equation 3.8 is an evaluation that means if the $BatLifeFilter$ returns a value less than η, the robot has a low likely hood of completing the task. Therefore the best course of action would be to return home if possible. The η value depends on the robot range of voltage drop range and tolerance.

3.5 Cost/Utility Computation

After locations are filtered, the robot computes a realistic cost and value. The parameters taken into account are distance, communication, and battery life. The robot computes a distance from itself to the end location. Then the amount of communication cost remains mostly static throughout the experiment. Lastly, the battery life cost is computed with the remaining voltage of the robot. The following equations are refined versions of the filtering equations.

The cost computation is similar to the filtering equation but instead of a line distance, a path is computed to the target location. The *PathCost* is the summation of the distance between the calculated points on the path as shown in equation 3.9.

$$PathCost_i(x_L, y_L) = \sum_{j=0}^{n-1} \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2} \tag{3.9}$$

Where n is the number of locations between between the robot position and target location. The cost in 3.9 will be larger than that of 3.4 because the path will generally include a small change in angle. Requiring the robot to turn or traverse around known obstacles. Similarly the *PathValue* will be larger because of the added distance more values can in summed. The equation is shown in equation 3.10.

$$PathValue_i = \sum_{j=1}^{n} Value(PathCost_i(x_L, y_L)) \tag{3.10}$$

Where n is the number of occupancy grids the path touches. Similar to *PathCost*, *PathValue* will be larger than *LineValue* simply because the path taken is longer.

The communication equation 3.11 is the summation of the data sent for communication for a task 'i'. his includes sending the position, time, and owner as part of the task.

$$Comm = Datapacket + ExchangeMap() \tag{3.11}$$

If the ExchangeMap function is called, the communication cost is increased a static amount required to share the map. As the robot moves around the communication costs will largely be static, except when the exchange map function is called.

The last equation allows fro the robot to know when to return home as oppose to continuing to the next location. The purpose is to have the cost of moving towards a location increase as the battery voltage goes down.

$$Battery_i = \frac{1}{(BatVolts - 11.2)^2} \tag{3.12}$$

As the battery life gets closer the to the 11.2 limit, the cost increases. This means that the cost will remove more utility and therefore make the task undesired.

Once the previous equations are computed for all tasks in the Task List, the total cost and utility values are computed. The cost is the addition of all sub costs of distance, communication, and battery life show in equation 3.13. It represents the amount of resources required for a task.

$$Cost_i(x,y) = Distance_i + Communication_i + Battery_i \tag{3.13}$$

The next computation is the utility function in equation 3.14. This value represents the rate of gain. If the rate is large, the task is worth doing because of the expected amount of returns.

$$PathUtility_i = PathValue_i / Cost_i \tag{3.14}$$

Conversely, if the cost is high value is low, the location is not working going to. The purpose of the utility function is to evaluate what task is worth the effort to complete. The next step is to compute the marginal utility defined in equation 3.15.

$$MarginalUtility = (\frac{PathValue}{PathCost} - \frac{LineValue}{LineCost}) \tag{3.15}$$

Marginal utility is the difference between two utilities. Hence we compare our pre-estimate value with our new post-estimate computations. A location with a negative marginal utility means that original estimation was too large. This means the cost of moving to a location was too large or the *PathValue* was too small. Physically this means the robot has to move through a low value are for a longer amount of time. Ultimately meaning the locations aret worth moving to.

3.6 Bids

The bid represents the robots desire to complete a task. The best choice is chosen after the marginal utility is computed. The marginal utility is the rate of gain between the filter equation 3.6 computations and the PathUtility in equation 3.14.

After completing the computations for all tasks in the Task List, the robot chooses the best n tasks using equation 3.16 where $n|\{n \leq i\}$.

$$Choice_n = max(PathUtility_i) \tag{3.16}$$

The result is a Choice vector of tasks that the robot will compute a bid for. Grabbing the locations of tasks in Choice, a bid is computed from equation 3.17 where $i \in \{(x,y) \in Choice\}$

$$bid_i = (PathValue_i - Cost_i) \tag{3.17}$$

This bid value is passed to the Bid block in Figure 3.3 where it is processed and sent to the correct auction house. The subtraction of the value and cost will be positive if *PathValue* is larger than the cost. Meaning the robot expects returns for completing the task.

3.7 Auction Process

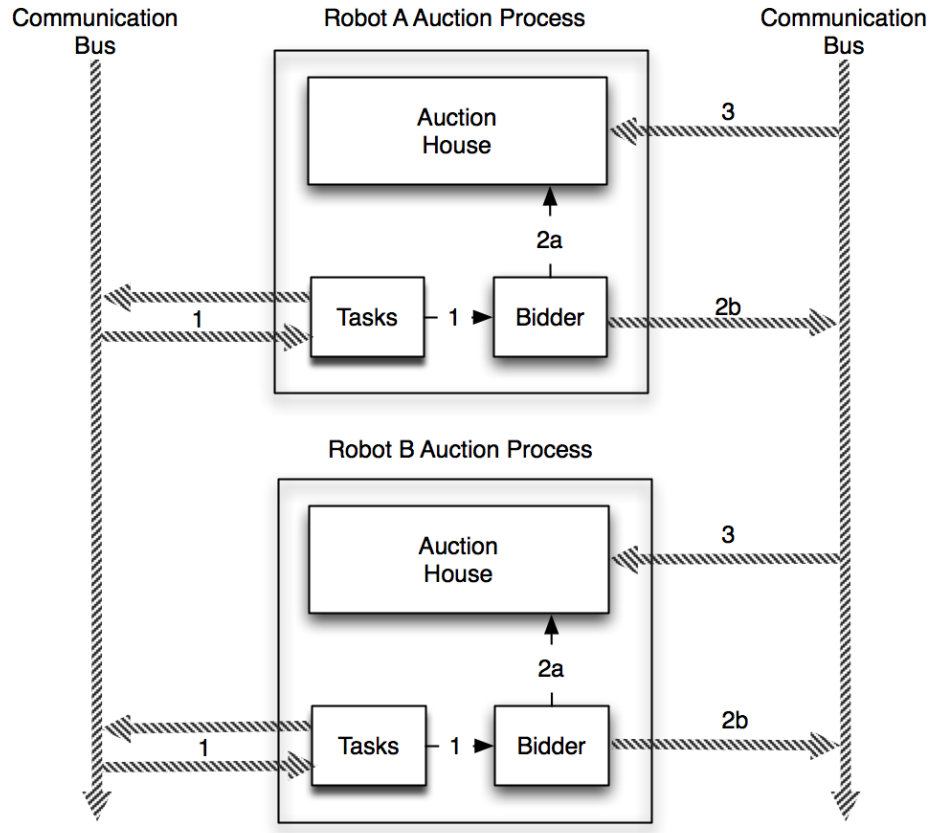
Auctions require an auction house, items (tasks), and bids to function. Every robot is an auction house and a bidder. The auction process represents the relationship between the auctioneer and the bidder. Looking at Figure 3.12 the first step is to create tasks. Once created, tasks are sent to itself and any external robot through step 1. The bidder block then does further processing and then decides to place bids on tasks. Each robot who creates a task becomes the owner of the task. This information allows each robot to know whether to place a bit to itself shown as step 2a or an external auction shown in step 2b. Step 3 represents bids being received by the auction house from external robots. The last step, not shown, is to inform the winning robot that it has one a task.

To limit the amount of communication packets sent, a blind auction is implemented. A blind auction means only the auction house is aware of the bids that are received. An open auction is where every robot knows the bid sent from another robots. This requires an increase in communication because every robot must be updated about the state of the rest of the system. Therefore to reduce communication requirements, a blind auction is used.

Along with the bid being placed, the creation time is shared. This allows the auction house to know when to send a confirm to a robot for a location it has created. Once bids are received the largest bid for any location is the winner. if a robot receives not confirm from any robot, including itself, the location with the largest $MarginalUtility$ is used.

Formalizing Figure 3.11 into an example results in Figure 3.12. Within Robot A, once a task is created it is sent to an internal (solid line) bidder block and an external (hash arrow) bidder lock (step 1). The bidder processes tasks, and places bids on on external or internal auction houses

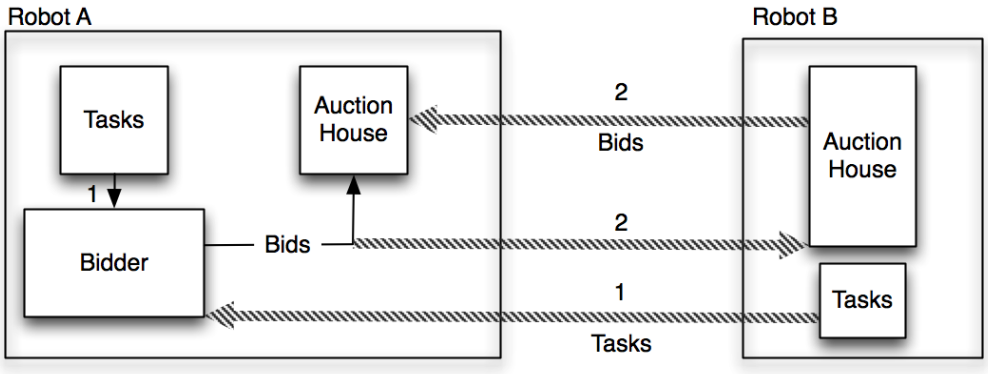
Figure 3.11: Auction Structure



(step 2). Robot B similarly sends bids to robot A's Auction house (step 2). The last step is when the auction house decides who wins a task. A confirmation win is sent to the winning robot.

The process in Figure 3.12 is repeated asynchronously, meaning not all robots will receive all tasks. Similarly, each robot does not bid on every task. Each task created has a time out stamp, which bars it from being bid on once it expires. Even if bid on, the task will be rejected by the host robot. The asynchronous task creation and bidding arrises from the difference in the task completion by a robot. It may take robot A 1 minute to complete its first task and robot B 2

Figure 3.12: Auction Overview Example



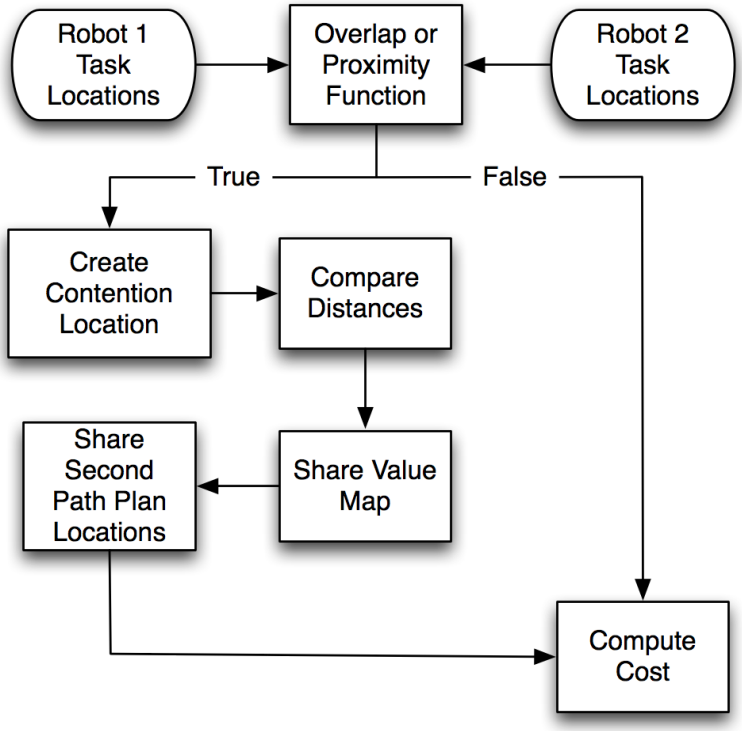
minutes. Hence at any one time, not all robots will be creating and bidding on new tasks.

3.8 Second Path Plan

The purpose of the second path plan is to allow the robots to detect when a collision is possible, exchange information, and utilize shared new information to determine what is the best course of action is. Once a Task List is created, it is shares it with other robots. If two locations in different Task Lists are identical or close, the system enters the second path plan function shown in Figure 3.13. The first step is to create a center target location that is the average between the two locations. Removing the two contention locations from consideration. Then each robot computes a distance from itself to the target location, and whichever is closes becomes the task owner.

After a contended location is identified, the robots share their value maps. This allows each robot to understand the others value map state. Therefore when robot A has explored an area, robot B will learn about it and devalue those locations. Later when robot A evaluates its Task list,

Figure 3.13: Second Path Plan Structure



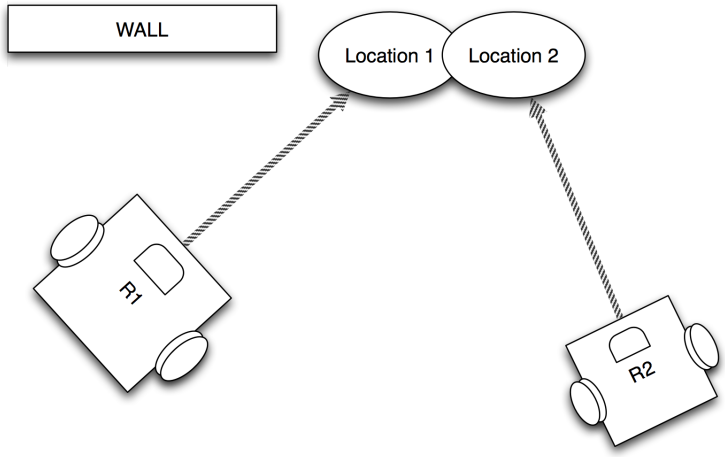
those locations will have less value and there are removed as possible bid candidates.

The Task owner then creates new secondary Task Locations centered around the contended location. This allows the robot to plan to the locations further away. Allowing each robot to move towards a location that is away from one other.

3.8.1 Guided Example

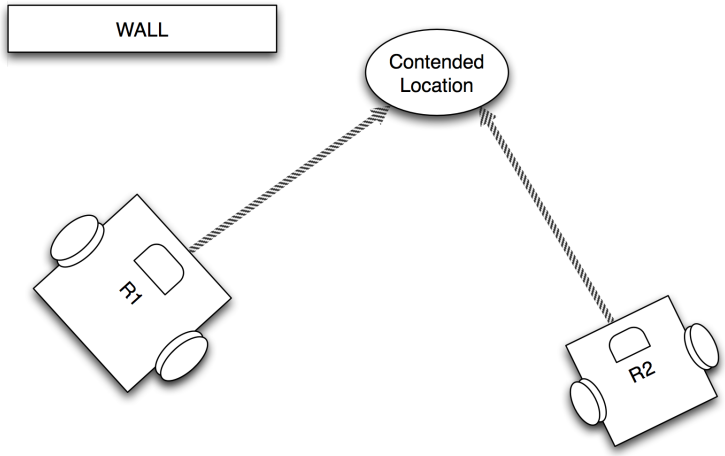
Starting at Figure 3.14. Robot 1 and Robot 2 created locations 1 and 2 respectively. These locations are too close, initiating the second path plan algorithm.

Figure 3.14: Task Location Overlap Conflict



The result, shown in Figure 3.15, is a new task location that is the average of the two previous locations. The original locations are deleted.

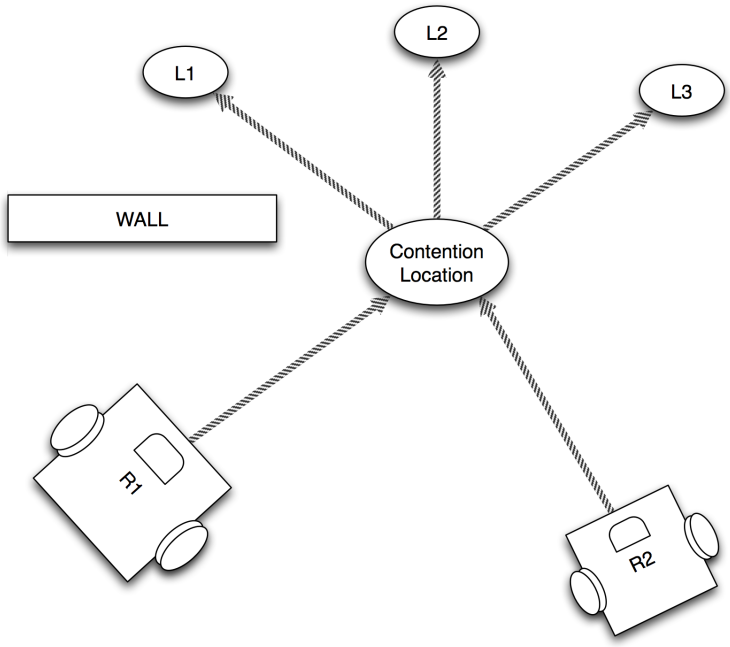
Figure 3.15: Combine Task Location



They each compute their distance to the contended location and share their respective distances. The closest robot is assigned the conceded location owner. The owner then creates a new

set of locations and shares them with the opposing robot. The desired result from the example is

Figure 3.16: Second Path Result



to have robot 1 move to L1 and robot 2 move to L3. Moving to those locations allows the robot to search the greatest amount of area possible. The robot may choose new locations after the step.

3.9 Communication

Our experiments use a small number of robots, nevertheless reducing communication between all robots is important. The low amount of required communication allows the robots to be robust in low communication environments. Allowing the least amount of communication further reduces the amount of computational resources used by the robots. The instances the robots do communicate are shown below.

1. Send/Receive Tasks
2. Send/Receive Bids
3. Send/Receive Task Confirm
4. Send/Receive Contention Location Distance
5. Send/Receive Value Maps

The ability to share tasks is vital to the algorithm. without this the robot only knows its own tasks. Tasks are sent with the location, time, and bidding robot name. WIth these three pieces of information the receiving robot can proceed with its own computations.

Next the robot must place bids on tasks. The bid includes the bidder name, and bid value. The auction house then receives these bids and does further evaluations to decided who wins what task location. Once the auction house chooses a winner, the confirmation is sent from the auction house to the winning robot. If the robot receives no confirmation from the auction house, it simply performs one of its self created tasks.

Sometimes task locations between two robots are created. This initiates the second path plan function that requires each robot to send its own distance to the contended location. The robot with the shorter distance becomes the owner of that task. Additionally the value map between the robots is sent. Depending on the size of the map and grid, this may require a lot or little communication.

Chapter 4

Experimental Setup

Experimentation is done to examine how quickly exhaustive, PSO, MPSO, and Market algorithms perform. Experiments are done by increasing the area and increasing the number of robots. Further experiments are done in two different areas to observe how well they react to different initial conditions. The amount of time and total area covered is compared between an exhaustive search and the Market algorithm.

4.1 Experiment Outline

The Market algorithm will be compared to three other search algorithms. These include an exhaustive search, PSO, and MPSO. The three experiments compare the of time to find the target, and area explored by the robots. All experiments are done with increasing number of robots. The full list of Experiments are described below.

Increasing Search Space

Test 1: Single Robot Time

Test 2: Double Robot Time

Test 3: Triple Robot Time and Area

Random Initial Conditions

Test 4: Single Robot Time and Area

Test 5: Double Robot Time and Area

Test 6: Triple Robot Time and Area

Courtyard Random Initial Conditions

Test 7: Single Robot Time and Area

Test 8: Double Robot Time and Area

Test 9: Triple Robot Time and Area

As described above, the time and area are recorded at the end of each experiment. When Handling multiple robots, the sum of the area is area is recored, and the robot that found the target is recorded as the completed time. The time to find the target should decrease from the single robot experiment to the triple robot experiment. Similarly, the amount of area covered to find the target should decrease with the addition of robots. The tests are ultimately compared to an exhaustive search of the total area.

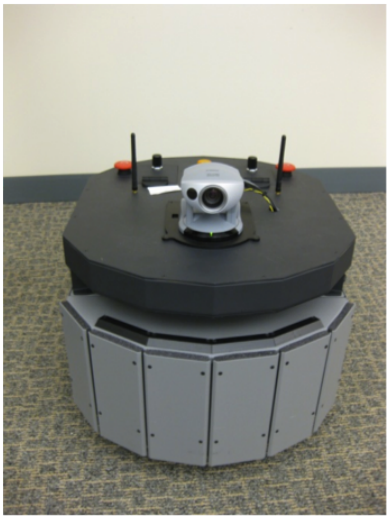
4.1.1 Robots

All robots has the same 1.8 Ghz Intel Pentium M processor with 512 GB of ram and 120 GB hard drive space. Each robot has a unique combination of sensors that allow it to perform

certain tasks better than one another. Thus some tasks can be evaluated differently based on the type of available sensors.

PatrolBot shown in Figure 4.1 has sonar sensors in the back sensors to help it detect walls, and other objects. From and back bump sensors to help it detect collisions. And a Canon VC-C50i PTZ video camera used to detect the target objects. A Canon VC-C50i Pan/Tilt/Zoom (PTZ) video camera. And lastly a SICK LMS 2000 LASER range finder, used mainly for localization. The laser

Figure 4.1: PatrolBot



scanner allows PatrolBot to have the highest localization score amongst the whole team. This means it is unlikely to lose itself while exploring the environment. This addition comes at a computational cost and lower batter life compared to the other robots.

P3at 8 front and back sonar sensors used to localize itself in the map. The onboard encoders attached to four wheel drive allow for localization. low localization score causes problems later on, but is allowed in the system to account for a broken robot. Similar to the patrol bot, P3at

has front and back bump sensors. The robot also has the same Canon VC-C50i PTZ video camera as PatrolBot.

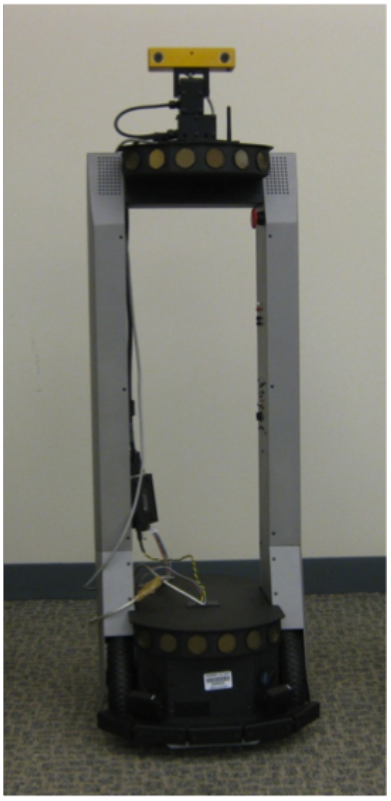
Figure 4.2: P3atBot

Lastly PeopleBot in Figure 4.3 has front and back sonar sensors. An integrated gyroscope allows it to detect and correct heading angles. Including a BumbleBee 2 CCD video camera. The camera is mounted on a Directed Perception PTU 46-17.5, which is a pan tilt unit attached to the onboard computer.

4.1.2 Camera Model

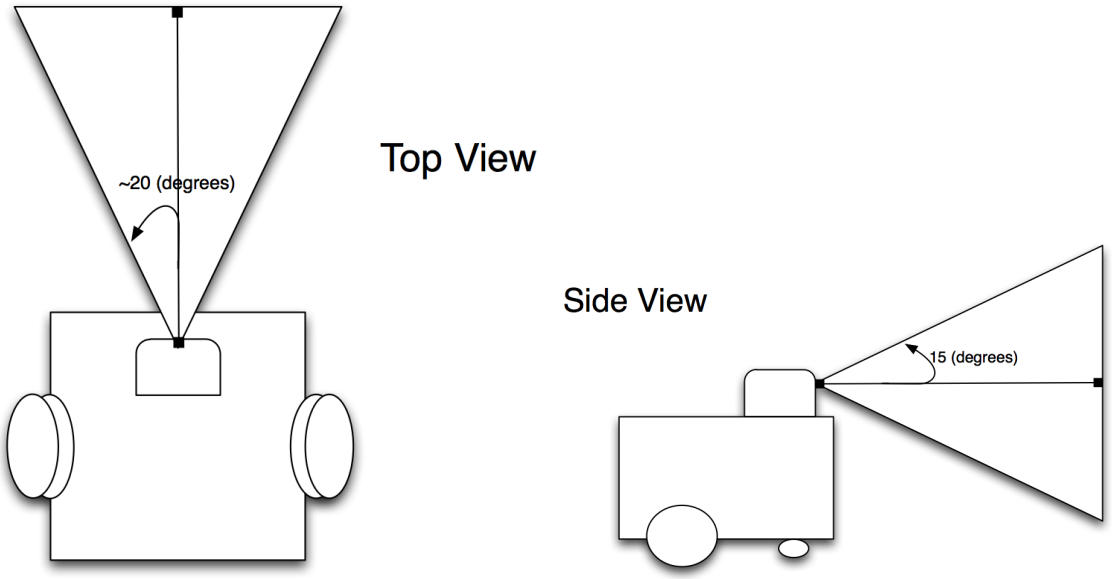
Modeling the field of view of the camera is important for the algorithm or it allows for proper reduction of occupancy grids. As the robot moves around the environment, the value of any grid observed from the camera Field of View is reduced. The robot has a horizontal FOV of twenty degrees and a vertical FOV of thirty degrees show in Figure 4.4.

Figure 4.3: PeopleBot


z

4.2 Experiment 1: Increasing Search Area

This test is designed to observe how each algorithm responds to the increase in search space. The environment is divided into three areas shown in Figure 4.5. For the experiment, the three search areas are described in table 4.1. The area increased doubles from the smallest to largest area. The label for each case is (1/2S, 2/3S, S), where'S' is the Search Space. Each case is repeated by increasing the number of robots in the system and observing how the change of area and number of robots effects each algorithm.



(a) Top View

(b) Side View

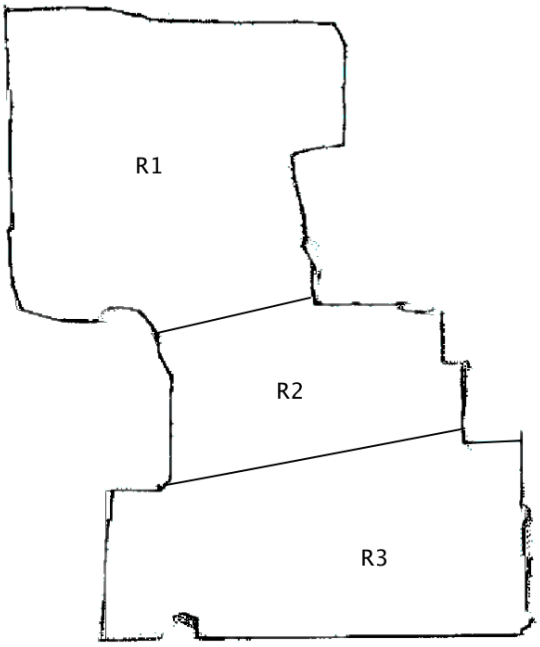
Figure 4.4: Camera Model

4.2.1 Single Robot Test Case

This experiment shows the baseline to compare multi-robot systems. Since this method does not allow for any cooperation, these tests show how well an algorithm works by having a major component removed. The target locations for each increasing area are shown in red on image 4.6.

The location of the Target 1/2 S, is the last location the exhaustive method will move to. This means that the time it takes the exhaustive method to find the target is the longest. Target 2/3 S is not as far away as it can be, but rather 60% to the furthest location. Lastly, the location of target S is the furthest possible point from the initial beginning. The results between the robots is shown

Figure 4.5: Increasing Map Size



in Figure 4.7.

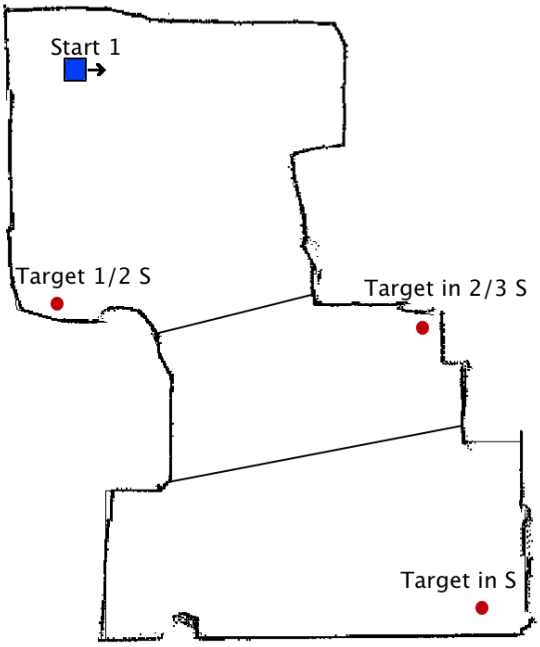
For the 1/2 S test, PSO and MPSO methods work better than the Market and exhaustive algorithms. This runs true for all cases as the search area increases. The difference in time between the PSO and MPSO method stays within 1 minute. The difference between the Market and exhaustive methods on average is 53 seconds. The Market algorithm, despite the large standard deviation of up to a 1 minute it is still better than the fastest exhaustive worst case. The single robot case does not allow the Market algorithm to take advantage of any cooperation, hence the poor time results.

Despite little difference in time to completion, the Market algorithm find the target observing a smaller area compared to the exhaustive method. This can be seen for every case in Figure 4.14 Increasing the size of the search space forces the robot to explore more area. But the rate of

Table 4.1: Size of search region in experiments

Search Space	Area ft^2	Region
1/2S	252.5	R1
2/3S	336	R1+R2
S	505	R1+R2+R3

Figure 4.6: Singel Robot Map Setup



change of area explored dos not grow linearly. The change of area explored from the 1/2S case to the 2/3 S case is 70 ft^2, while the 2/3 S and S case is a difference of 23 ft^2. This is different to the exhaustive method where the area explored increases linearly with the size of the map.

Figure 4.7: Single Robot Search Time

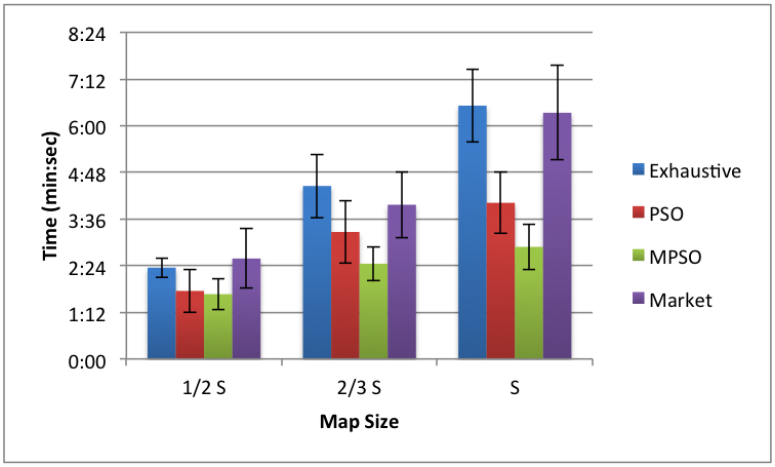
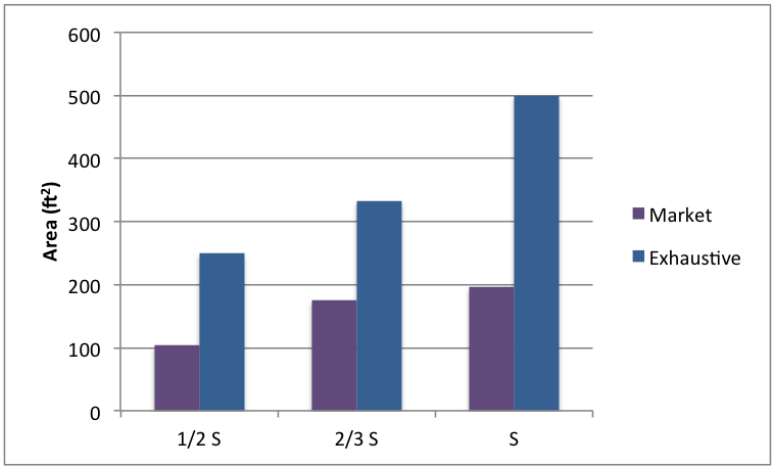


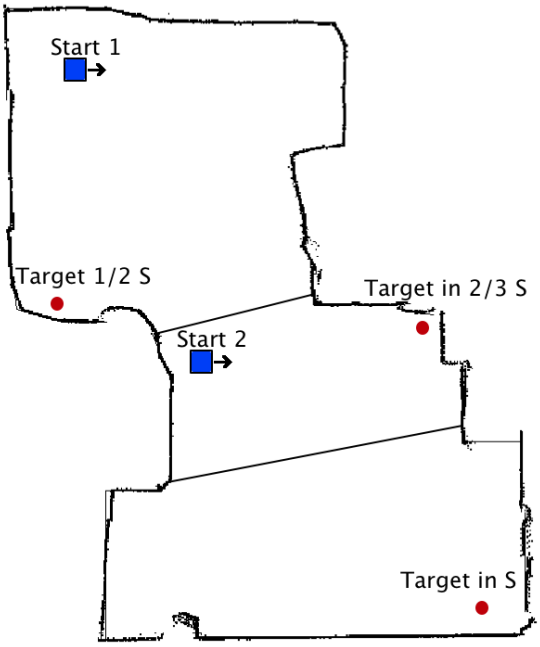
Figure 4.8: Single Robot Exploration Area



4.2.2 Double Robot Test Case

With the addition of a robot to the system, each robot is capable of moving around and completing tasks quicker. As expected, the robots were able to find the target faster than the single robot case. Figure 4.9 shows the init a positions of the robots.

Figure 4.9: Two Robot Map Setup



For the 1/2 S case, the Market algorithm is able to beat the PSO method. This does not continue for the larger 2/3 S and S case. PSO remains slightly faster than the Market algorithm by finding the target 5 to 7 second faster. But looking at the standard deviation between the PSO and Market 2/3 S case times, they overlap. Meaning they are statistically similar. Compared to the MPSO, the Market algorithm does poorly. Though the standard deviation of the MPSO and Market algorithms do overlap for the 1/2 S and S cases.

Similar to the previous single robot experiment, the robot is able to find the target far faster the exhaustive method. One striking difference is that the S case has a reduction of area explored for both the Market and Exhaustive methods. This was due to the paths taken by the Market algorithm cooperation between one another. Plus the time to complete was shortened so

Figure 4.10: Double Robot Search Time

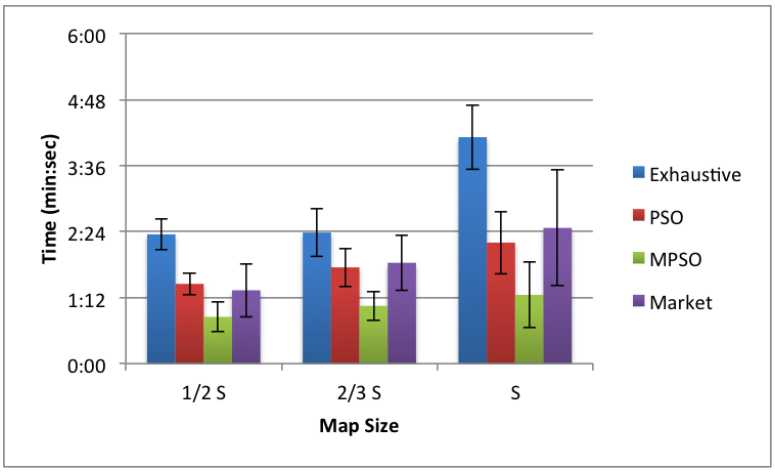
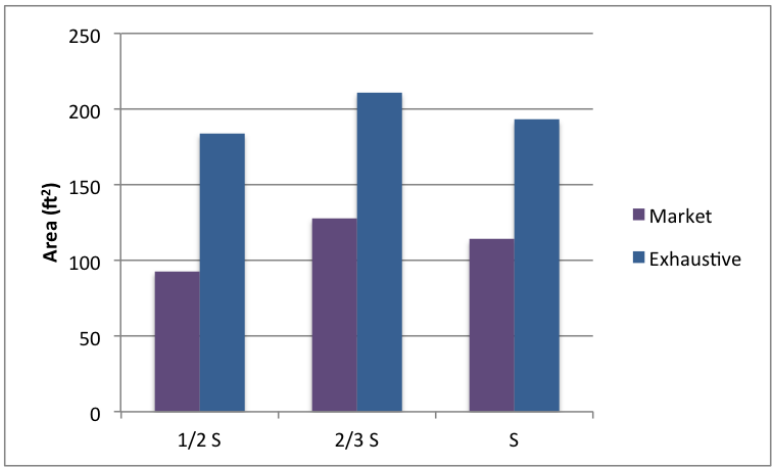


Figure 4.11: Double Robot Exploration Area

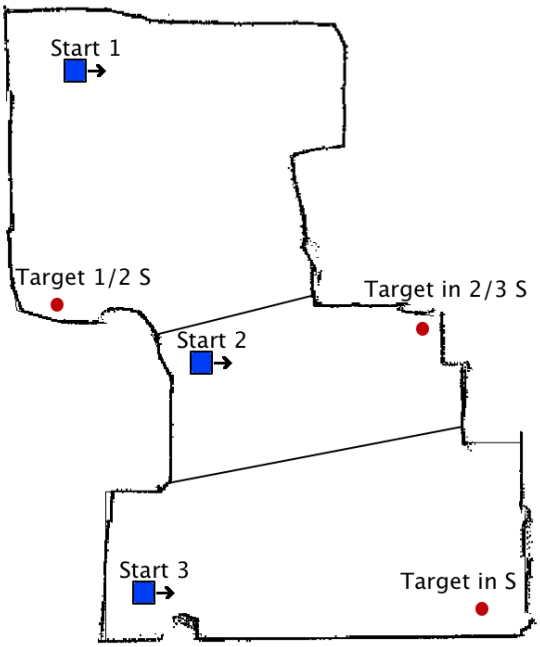


that the amount seen by the robots was reduced. On average the Market algorithm takes 111 ft^2 to find the target, while the exhaustive method requires 199 ft^2.

4.2.3 Triple Robot Test Case

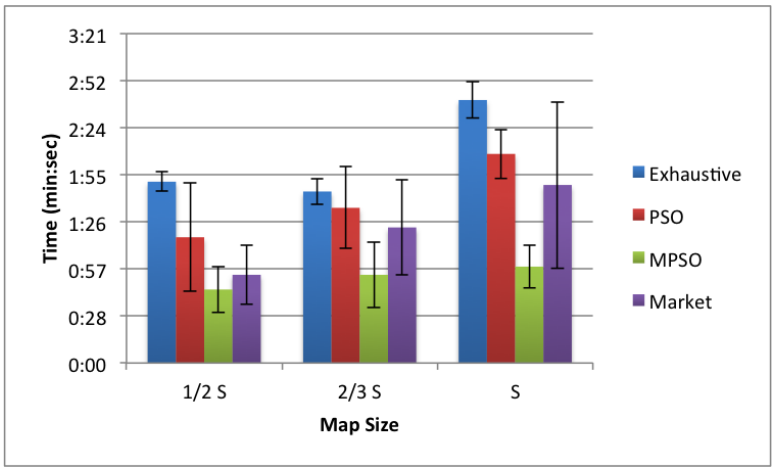
The last set of these tests is with all three robots working together. The start positions and target locations is shown in Figure 4.12.

Figure 4.12: Three Robots Map Setup



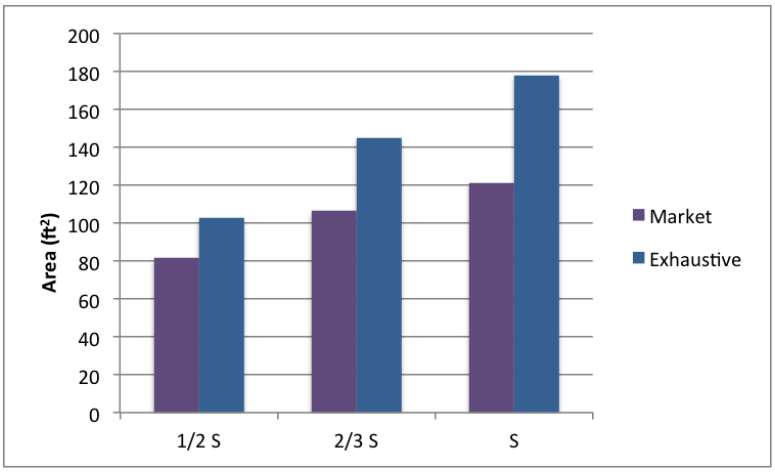
As expected the completion times continue to reduce, and now hover over a minute and a half. In this test, the Market algorithm was able to find the target faster than the PSO method for all cases and on average 18%. Similarly to the previous experiment, the Market algorithm worst standard deviation is quicker than the exhaustive method. The standard deviation of the Market algorithm for the full map becomes large. This is due to some overhead in planning between robots.

Figure 4.13: Triple Robot Search Time



Using three robots the amount of area explored by the third robot does not increase the amount of are observed. Rather the amount of area required to be observed stays around 103 ft^2. The reduce area observed to find the target is a direct result of having the robots in a small confide

Figure 4.14: Triple Robot Exploration Area

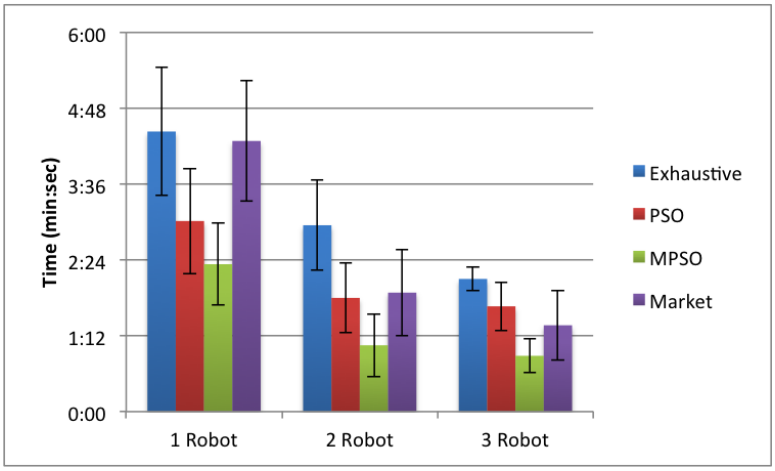


area. Their paths were different and hence able to find the target quicker and observed less area.

4.2.4 Experiment 1 Summary

The previous sections decried how the robots adapted to finding the target as the area increased. This section describes how each algorithm responds when a robot is added to the system. The average time it took each algorithm to complete the previous tests cases is shown in Figure

Figure 4.15: Experiment 1 Time Performance Number of Robots



4.15.Initially the Market algorithm performs no better than the exhaustive search. Adding a second robot allows the Market algorithm to perform similar to the PSO algorithm. Once the third robot is introduced, the Market algorithm is able beat the PSO completion times.

Defining performance increase as shown in equation 4.1.

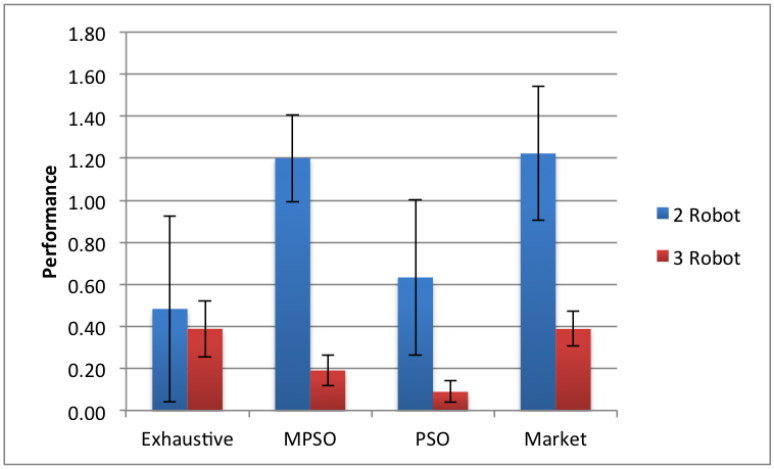
$$P = \frac{R_i - R_{i+1}}{R_{i+1}} \tag{4.1}$$

Where i is the number of robots. R is the algorithm experimental results. The performance equation is the percentage increase as the number of robots are introduced.

In terms of time, the performance equations yields the percent reduction in the time nec-

essary to find the target. The results are shown in Figure 4.16 as the performance of each algorithm as the number of robots increase.

Figure 4.16: Experiment 1 Time Performance



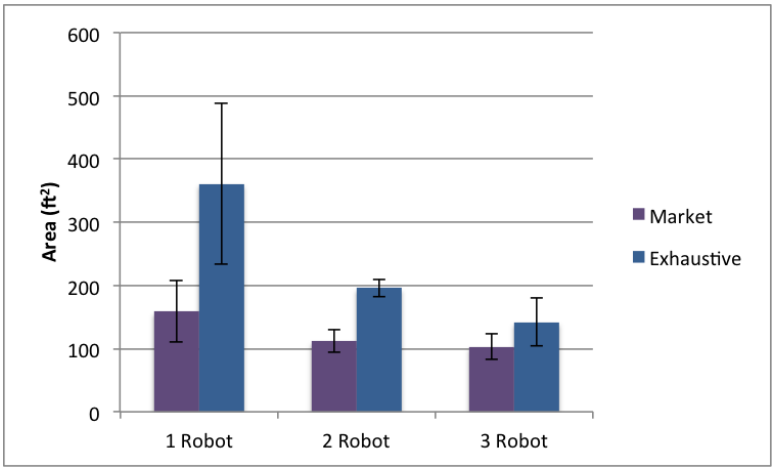
Adding a secondary robot to the exhaustive algorithm allowed it to find the target 44% quicker than the single robot case. This is the shortest increase in performance of any of the other algorithms. The Market algorithms increase in performance is best with a 122% increase over its single robot case. Second is the MPSO algorithm with a 120% increase in performance, and in third place is the PSO algorithm with a 64% increase.

Once the third robot is introduced, the benefit increase from MPSO and PSO algorithms drop dramatically. Therefore adding a third robot to the system yielded little benefit. Adding a third robot to the exhaustive and Market algorithm method yielded 39% increase. If a fourth robot were to be added to the system, the times are expected to be reduced.

Figure 4.17 is the full area results from the previous test cases. The amount of area required to find the target decreases as the number of robots increase. Because each robot is able

to observe unique areas, the amount required to find the target is reduced. If the experiments were

Figure 4.17: Experiment 1 Area Explored Summary



run for rate same duration, the amount of area observed would be larger as the number of robots increase. Because the experiments are terminated once the target is found, the area observed is reduced.

Using equation 4.1 to define performance, but using the area explored test restlts. The results are shown in Figure 4.18. Adding a second robot increased the performance of the Market algorithm by 42%. The performance of adding a second robot to the exhaustive algorithm increases by 82%. The dramatic increase is due to the low performance of the exhaustive algorithm. When the third robot is added, the performance increase of the exhaustive method is reduced to 38%. If the trend is extended to four robots, the performance of 4 exhaustive search robots is equivalent to 3 market based ones.

Lastly, Figure 4.19 is the rate of square feet observed by the robots as the number of

Figure 4.18: Experiment 1 Area Explored Performance

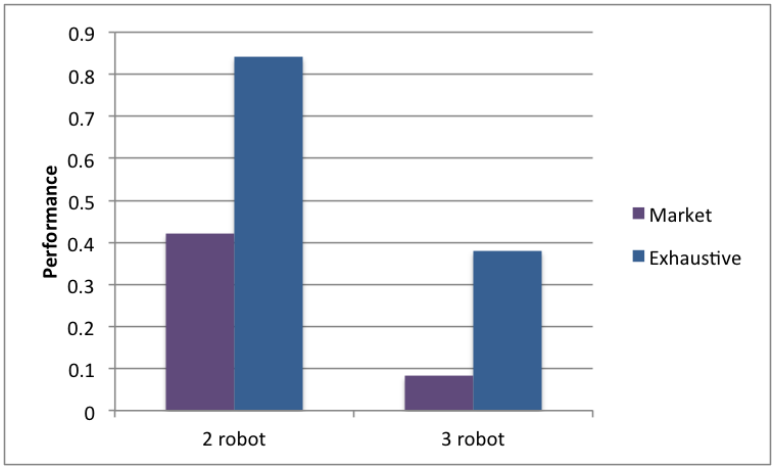
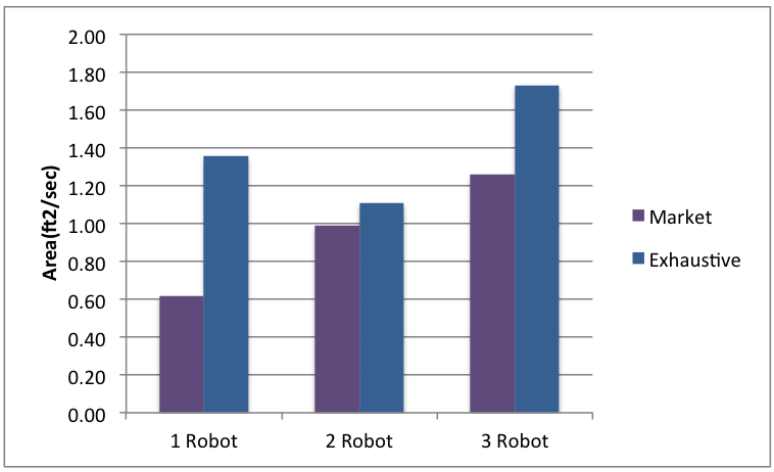


Figure 4.19: Experiment 1 Area Observed Rate

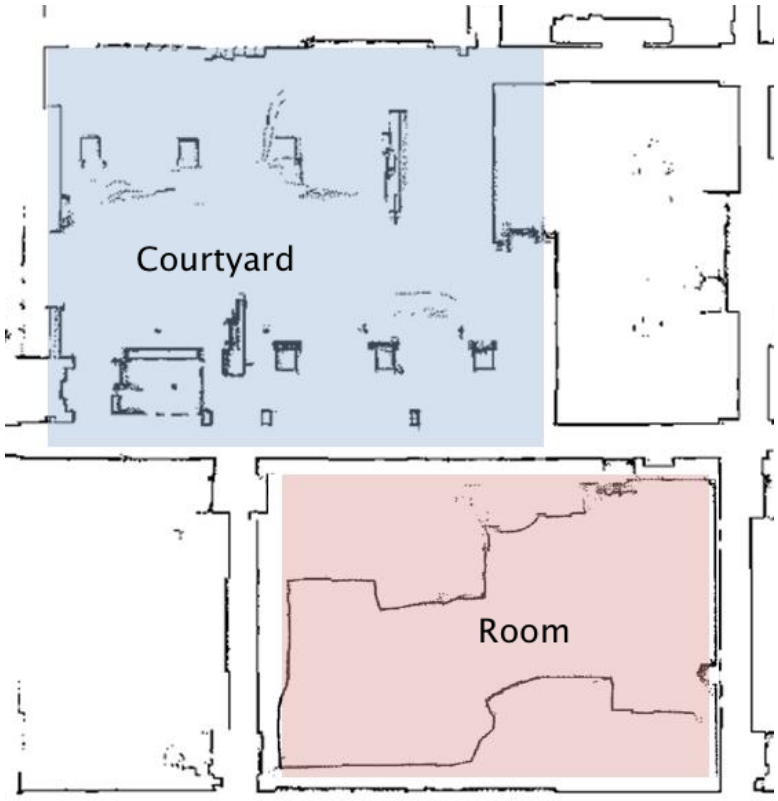


robots increases. The Market algorithm is unable to observes a slower rate of area per second for any number of robots. Though observing at a smaller rate, the Market algorithm is still able to find the target quicker.

4.3 Experiment 2: Random Initializations

There are two test cases for this experiment. They including having the robots initial positions, and target locations randomized in either the room search space (500 ft^2) or larger outside courtyard (8,000 ft^2). Both show in Figure 4.20. The purpose it to see how quickly the robots are

Figure 4.20: Room And CourtYard

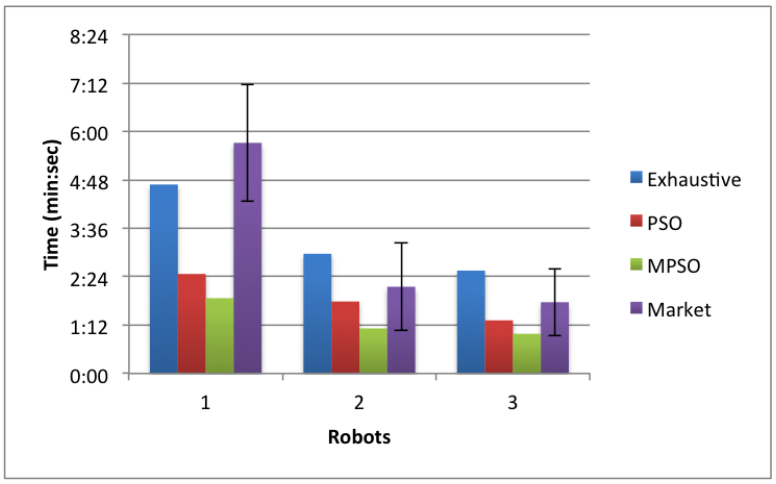


able to find the target given any initial set of conditions. Previously, we tested specific cases with different number of robots. Here the results demonstrate real world scenarios, where the robots will be at a certain position and then receive a request to find a target.

4.3.1 Room Search Space Test Case

The results are shown in Figure 4.21. The Market algorithm performs worse than the exhaustive method for the single robot case by 62 seconds. Adding a second robot, the Market algorithm beats the exhaustive method by 49 seconds, gain in performance of 37%. Adding the final robot increases the performance by 50% over the exhaustive method. Comparing the PSO and

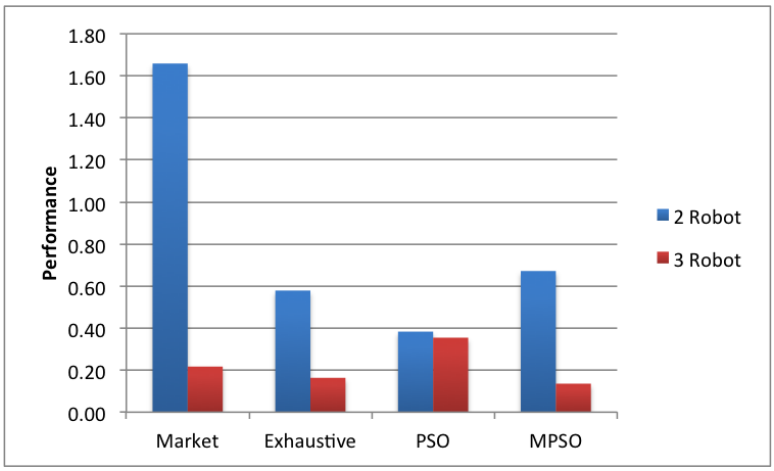
Figure 4.21: Random Initializations Room Completion Time



MPSO two robot methods, they beat out the Market algorithm by and average of 42 seconds. This gets reduced by 37 seconds when the third robot is introduced. For the two and 3 robot case, the standard deviation of the Market algorithm is within the average MPSO time.

The performance increase on their respective completion times is shown in Figure 4.22. The Market algorithm has the largest increase in performance increase of 166% when the second robot is introduced. This is larger than the summation of the other three algorithms combined! When the third robot is introduced, the gain for the Market algorithm is only 22%. The Market

Figure 4.22: Room Search Space Time Performance



algorithm has the second largest performance increase with the introduction of the third robot. The largest is the PSO algorithm with 38%. This is different from Experiment 1, where the area was increased.

The amount of area explored to find the target is shown in Figure 4.23. The Market algorithm is always able to find the target by observing a smaller area than the exhaustive method. The average area required for the market algorithm to find the target is 132 ft^2, compared to 250 ft^2 for the exhaustive algorithm. The amount of area observed for the three robot case are similar. The standard deviation of the Market algorithm is within bounds of the exhaustive average.

4.3.2 Large Outside Courtyard Test Case

The large courtyard has an area of 8000 ft^2, and increase of 16 compared to the room size. Increasing the size drastically, will demonstrate how the robot handles a large area. the map used is shown in Figure 4.24.

Figure 4.23: Random Initializations Room Exploration Area

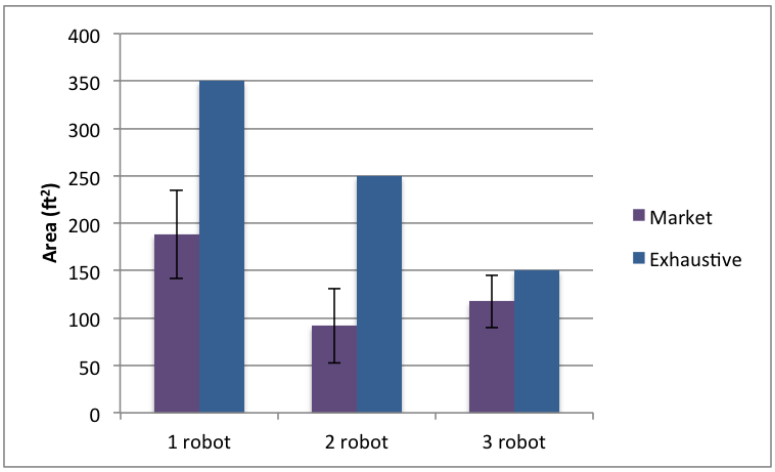


Figure 4.24: Courtyard Map

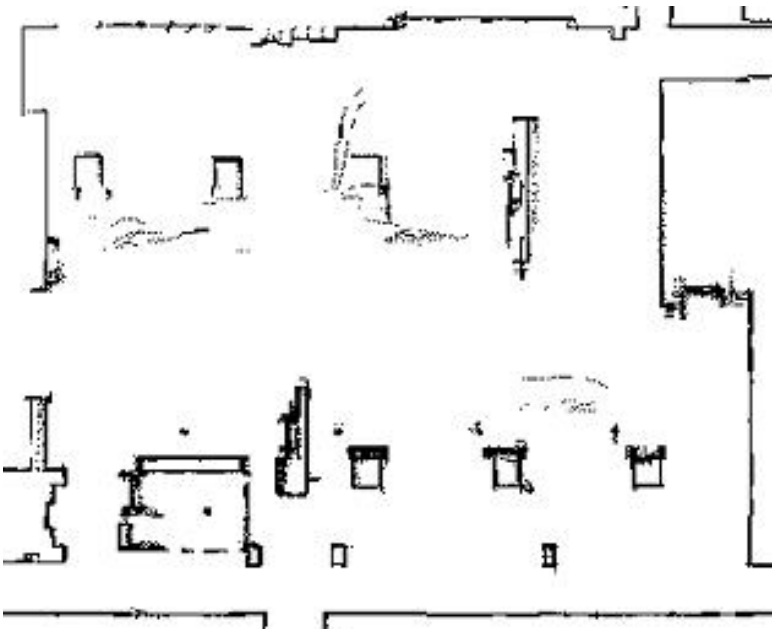
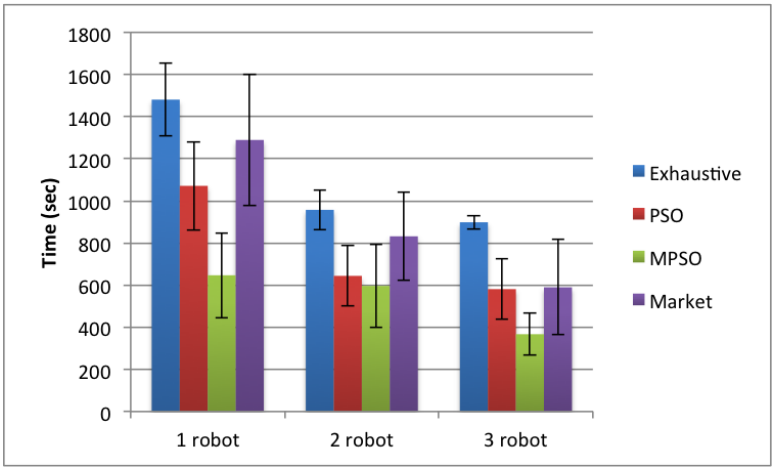


Figure 4.25 demonstrates the completion time results. The Market algorithm beats the exhaustive method every time. One major difference from the previous example is the increased standard deviation of the Market algorithm. Despite how large it is, the single robot case is still

better than the worse possible exhaustive standard deviation time. The best Market time is 3 minutes slower than the average PSO time. The MPSO for a single robot takes about 10 minutes to find the target.

Figure 4.25: Random Initializations Courtyard Completion Time



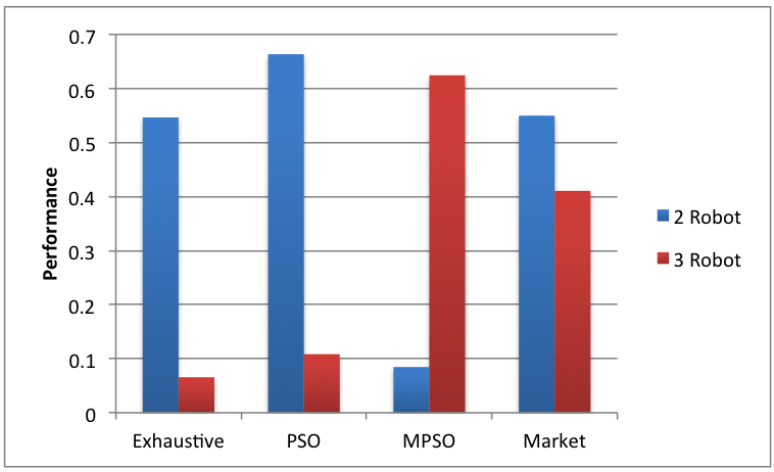
Moving onto the two robot case, the time completion are reduced by 10 minutes. The Market algorithm is able to find the target in 832 seconds (13 minutes 52 seconds). The PSO algorithm completion time is within bounds of the Market algorithm, but beyond the reach of the MPSO algorithm. The Market algorithm is 29% slower than PSO and and 39% slower than MPSO.

For the three robot case, the Market algorithm is able to find the target 52% faster than the exhaustive method. Interestingly, the Market algorithm is able to match the PSO algorithm. The standard deviation of the Market algorithm becomes somewhat of an issue because it gets close the the exhaustive time. The MPSO and Market standard deviations also overlap, meaning that the market algorithm is catching up with the MPSO algorithm.

Next up is the Performance measure as defined by equation 4.1. The introduction of a

second robot helps all algorithms except MPSO. The other algorithms found the object 50% faster than the 1 robot case. The addition of the second robot did little to improve the performance of the MPSO algorithm. This trend does not hold true for three robots. With the introduction of the

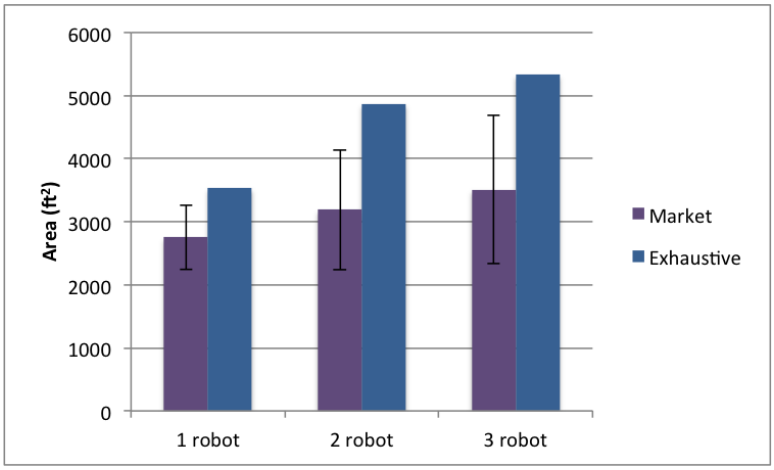
Figure 4.26: Random Initializations Courtyard Time Performance



third robot, MPSO obtains a 60% reduction in completion time. Next is the Market algorithm with a 41% increase. If a fourth robot were to be introduced, both MPSO and the Market algorithm are expected to receive some boost in performance.

Because the area of the courtyard is large, there is less overlap between the robot explored paths. This means that each robot is allowed to explore on its own. The results are shown in Figure 4.27. The robot area explored increased for every robot experiment. For the the three robot case, the increase in area was smaller. This is due to the robots finding the target faster. The standard deviation exploded with the three robots due to the robots finding the target quickly a few times. This skewed the results and created a larger standard deviation.

Figure 4.27: Random Initializations Courtyard Exploration Area



Chapter 5

Conclusions

The Results show that the Market algorithm can perform better than the exhaustive search. Adding a second and third robot to the system, makes the Market algorithm perform far better than the exhaustive search, and marginally better than PSO. The Market algorithm does not perform better than the MPSO algorithm. Though on experiments with 3 robots, the Market algorithm is within statistical bounds of the MPSO algorithm. Table reftable:SummaryTime shows the Market algorithm time performance against the Exhaustive algorithm. Table 5.2 similarly does the same for the area observed.

The first experiment demonstrates the effect of increasing the search area and increasing the number of robots. Table 5.1 shows that the market algorithm, is on average 40% faster than the exhaustive as the number of robots increases. Specifically, for the single robot case, the Market algorithm does not perform much better than the exhaustive search. Adding a second and third robot increases the performance drastically.

Table 5.1: Summary Of all Results For Time

Increasing Area	Exhaustive Base	Market Algorithm
1 Robot	0%	2%
2 Robot	0%	58%
3 Robot	0%	60%
Average	0%	40%
Random Initialization (small)	Exhaustive Base	Market Algorithm
1 Robot	0%	-18%
2 Robot	0%	38%
3 Robot	0%	44%
Average	0%	21%
Random Initialization (Large)	Exhaustive Base	Market Algorithm
1 Robot	0%	15%
2 Robot	0%	15%
3 Robot	0%	52%
Average	0%	27%
All Experiments	0%	29.5%

The second experiment zoomed in on the full map (S case from the first experiment) and observed how the robots found the object with random initializations. The only variable aside the the initial conditions, were the number of robots in the experiment. These results showed that for the single robot case, the market algorithm does not perform better than the exhaustive method. Including a second robot improves the performance drastically. The two robot market configuration allowed the robot to find the target 38% faster than the exhaustive search. Once the third robot is introduced, the Market algorithm outperforms the exhaustive algorithm by 44%. The end result is the Market algorithm beating the exhaustive search y 21% on average.

For the last set of tests, the robot was placed in the courtyard with random initializations. Here, for the single robot tests the Market algorithm was able to find the target 15% faster than the exhaustive method. This trend continues for the double robot case. When the third robot is

introduced, the performance over the exhaustive method increases to 52%.

When summing the performance of the Market algorithm over all experiments, the Market algorithm is 29% faster than the exhaustive method.

When comparing the area explored by the robot until the target is found, the market algorithm out performs the exhaustive search. From Table 5.2 the percentage of area observed for the first experiment is on average 37% less. For all experiments, the Market algorithm is able to find the target while searching a smaller area. These area observations are in line with the time

Table 5.2: Summary Of all Results For Area

Increasing Area	Exhaustive Base	Market Algorithm
1 Robot	0%	-40%
2 Robot	0%	-36%
3 Robot	0%	-36%
Average	0%	-37%
Random Initialization (small)	Exhaustive Base	Market Algorithm
1 Robot	0%	-89%
2 Robot	0%	-79%
3 Robot	0%	-28%
Average	0%	-65%
Random Initialization (Large)	Exhaustive Base	Market Algorithm
1 Robot	0%	-28%
2 Robot	0%	-52%
3 Robot	0%	-51%
Average	0%	-43 %
All Experiments	0%	-48%

required to explore the environment. Since the Market algorithm allowed for some cooperation between robots, they did not explore already explored areas. This allowed for each robot to search the most unobserved area compared to the exhaustive methods which did explore the same area.

There is a drastic reduction in area observation required for the robot to find the target.

In short, the market algorithm is able to find the target on average 19% quicker than an exhaustive search. Compared to the PSO algorithm, the market algorithm is able to match and outperform the PSO by adding additional robots to the system. At no point does the market algorithm perform better than MPSO. But it does start to reach the upper bound of the MPSO algorithm. Increasing code performances could yield the necessary results to push the Market algorithm towards MPSO time.

Comparing the Market and MPSO algorithm completion time, MPSO wins for every case. There exists recurring bug in the Market algorithm implementation where the robots are locked for at least six seconds for some tasks. If some of these locks are taken into account in the experimental results, the Market algorithm performs similarly to the MPSO algorithm. Because on some experiments the reduction in time can be between 39 seconds for multiple robot cases and unto 1 minute and 38 seconds for single and double robot cases.

MPSO has an inherently random process that is strictly dependent on the initial conditions. This means that algorithm can perform drastically different depending on the experiment. The market algorithm is structured, in an effort to remove randomness. There still exists randomness in the system, but it is a result of other aspects such as localization. Additionally, the market algorithm is able to handled multiple task at any one time. The MPSO algorithm is strictly an exploration algorithm, hence the Market algorithm is not expected to perform better. But the Market algorithm is able to perform similarly to MPSO and better than PSO on most experiments.

The Market algorithm also is actively trying to minimize the amount of overlapping area observed. The paths shown in Figure 5.1 represent how each robot moved through the environment

for one of the experiments The result is the Market algorithm able to search more unique area as

Figure 5.1: Path Comparison

oppose to overlapping areas that the MPSO algorithm produces. The Market algorithm is able to complete the task while minimizing the amount of overlapping area. Though still able to find the target faster than the Market algorithm , the lack of structure and control makes MPSO undesirable.

5.1 Future Improvements

5.1.1 Algorithmic improvements

The auction method specifications can be improved by allowing the robots to ask the auctioneer for further time. Currently if a task is evaluated and if the value is not high enough, it is

desecrated. An addition could be if the task is within a certain threshold, it queries the task creator for time or information. With the new information, the task is reevaluated, where the outcome is the task becomes favorable or unfavorable. This could catch results where the evaluation number was slightly below required, but can still be completed together with the robots current task.

Another addition could be a probabilistic history of the certainty of winning a task against another robot. For example, if robot X wins many robot Y tasks, robot X may alter a ”trust” variable. So when the evaluation of a robot Y task and robot Z task are equivalent, robot X should choose robot Y task because it is favorable. Another variable can be the ”completion trust” variable where if robot X does not complete robot Y tasks, then they both can have less ”completion trust” in one another. This means that when they evaluate each others tasks, they are reduced because they no longer trust one another.

A history of paths can be recorded and shared amongst the robot system. When the robots are introduced to a new environment, they have no knowledge as to what an optimal path may be. So if the robots share their path history, and alter the value of a grid box based on whether or not a secondary robot has moved through it. Hence initially, the robots will sub optimal paths, but through the sharing of path histories, they perform closer to a set of optimal paths.

The market algorithm can be tested against moving robots. With the addition of a target estimation function, the map can be augmented with new ”possible location” tasks. These possible locations can move the robot around towards these hot spots in an effort to see where the target may be.

5.1.2 Robot Improvements

Many issues exist in terms of the performance of the robot. An upgraded system will work better. A dedicated frame grabber card with its own object recognition code. Then interface the card with the Market algorithm code, freeing processes for the computer. Currently the frame grabber functions are done in the Market program, and in software as oppose to hardware. Another major change would be to not use MobileRobots path planning function because of its lack of performance. Too many issues were had in the use of that function.

Purchasing laser finders on all robots is highly suggested. Allowing for the robots to have good localization, which means the program can run longer without getting lost. These will intern allow for consistent results. The variability of the results above are a result of error the accumulated in the system.

Bibliography

[1] A Al-Jumaily and S Kozak. Behavior based multi robot cooperation by target/task negotiation, 2004.

[2] S. C. Botelho and R. Alami. M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement, 1999.

[3] D.J Cornforth. An investigation into dynamic problem solving in a hybrid evolutionary market-based multi-agent system. *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1732–1739, 2007.

[4] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.

[5] BP Gerkey. On role allocation in robocup. *RoboCup 2003: Robot Soccer World Cup VII*, 2004.

[6] Chen Haoyao, Sun Dong, and Yang Jie. Global localization of multirobot formations using ceiling vision slam strategy, 2007.

[7] A. Howard, G. S. Sukhatme, and M. J. Mataric. Multirobot simultaneous localization and mapping using manifold representations, 2006.

[8] N. Kalra, D. Ferguson, and Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams, 2005.

[9] Jong-Hwan Kim, Ye-Hoon Kim, Seung-Hwan Choi, and In-Won Park. Evolutionary multiobjective optimization in robot soccer system for education. *IEEE Computational Intelligence Magazine*, 4(1):31–41, 2009.

[10] S Kurihara, K Fukuda, and S Sato. Multi-agent coordination mechanism based on indirect interaction. *21'st Internation Conference on Advanced Information Networking and Applications Workshop*, 2007.

[11] Yan Meng, O Kazeem, and J.C Muller. A swarm intelligence based coordination algorithm for distributed multi-agent systems, 2007.

[12] D Messie and J.C Oh. Cooperative game theory within multi-agent systems for systems scheduling, 2004.

[13] N Michael, MM Zavlanos, and V Kumar. Distributed multi-robot task assignment and formation control. *Robotics and* , pages 128–133, 2008.

[14] E. Pagello, A. D;Angelo, and E. Menegatti. Cooperation issues and distributed sensing for multirobot systems. *Proceedings of the IEEE*, 94(7):1370–1383, 2006.

[15] S I Roumeliotis and G A Bekey. Distributed multirobot localization. *Robotics and Automation, IEEE Transactions on*, 18(5):781–795, 2002.

[16] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, C-29(12):1104–1113, 1980.

[17] AW Stroupe and R Ravichandran. Value-based action selection for exploration and dynamic target observation with robot teams. *Robotics and Automation*, 2004.

[18] Juhua Wu. Contract net protocol for coordination in multi-agent system. *Intelligent Information Technology Application, 2008. IITA ;08. Second International Symposium on*, 2:1052–1058, 2008.

[19] Cen Yuwan, Ye Ye, Xie Nenggang, Bao Jiahan, and Song Chongzhi. Flocking task research for multiple mobile robots based on game theory. In *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, pages 46–49.