Modeling, Simulation, and Characterization of Distributed Multi-agent Systems

by

Reed F. Young

Department of Mechanical Engineering and Materials Science
Duke University

Date:_____

Approved:

_____
Devendra P. Garg, Supervisor

_____
Silvia Ferrari

_____
Brian Mann

_____
Lawrence Virgin

_____
Gary Ybarra

Dissertation submitted in partial fulfillment of
the requirements for the degree of Doctor of Philosophy in the Department of
Mechanical Engineering and Materials Science in the Graduate School
of Duke University

2011

ABSTRACT

Modeling, Simulation, and Characterization of Distributed Multi-agent Systems

by

Reed F. Young

Department of Mechanical Engineering and Materials Science
Duke University

Date:_____

Approved:

_____
Devendra P. Garg, Supervisor

_____
Silvia Ferrari

_____
Brian Mann

_____
Lawrence Virgin

_____
Gary Ybarra

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Mechanical Engineering and Materials Science in the Graduate School of Duke University

2011

Abstract

The goal of this novel research effort was to address the impact that variations in a pre-selectable composition of a multi-agent system (MAS) can have on its resulting capabilities and performance while utilizing a composition-tunable potential-function control scheme. This was accomplished by first defining the litany of capabilities and performance characteristics of individual agents, and the intra-agent relationships between these. Next, various incarnations of the MAS were hypothesized where distinctions existed through variables such as agent types, payload types and capabilities, relative quantities, motion behaviors, and most notably, potential function control tuning parameters. These were extensively represented and tested both in simulation and in hardware experimentation resulting in rich data sets. Then, trade-off analyses were conducted to identify those factors demonstrating significant importance. These analyses provided insight to the control system and hardware design such that the core elements of the MAS itself could be orchestrated to provide the best efficiencies against comprehensive mission accomplishment. Finally, the analyses provided insight to mission planners such that they would be able to tailor the composition of a task-specific MAS in terms of cost and performance.

At its infancy, the study of mobile robotic technologies centered on simple wheeled and legged vehicles with elementary sensors and articulation to accomplish tasks such as movement through an area strewn with obstacles to find a goal location. Initial thrusts centered on providing these individual mobile platforms better sensing capabilities

including high-resolution imagers and LIDARs. These platforms offered increasing capabilities; however, quickly became very complex and costly. Recently, the research community is exploring MAS whereby the entity accomplishing tasks is composed of a number of heterogeneous or homogeneous mobile robotic platforms (agents) that behave as a system through coordinated, centralized or distributed control theory. These systems offer significant potential and advantages in that each agent can be a simpler, and therefore more robust and inexpensive, robot while the coordinated system can still accomplish complex missions. Additionally, the MAS construct offers inherent benefits including redundancy, spatial efficiency, and graceful degradation.

For this research, the critical enabling thrust explored was control via potential functions leveraging state space representation. A novel implementation of potential function control is developed that utilizes tuning parameters generated from state space relativity. For example, a tuning parameter consisting of the relative charge ratios of the individual agents and state space occupancy map cells is manipulated to optimize agent dispersion, path planning efficiency, and mission completion times.

The results of this research show how the census of agents, including quantity and composition, ultimately impacts the overall performance of the MAS. For example, increases in quantity result in a non-linearly fading increase in performance measured by time to complete a pre-defined mission. This non-linearity means that more is not always better, especially in consideration of logistic burdens that increase with additional agent quantities. Also, the potential-function control architecture proved particularly robust

and capable when used in conjunction with state space representation. Manipulation of the relative charges associated with MAS components and occupancy mapping resulted in a 25 percent improvement in task completion criteria. Finally, the concept of "sensing opportunity" is postulated where the combined capability of a MAS as a function of its agents' sensing systems is very much impacted by each sensor's opportunity potential to collect information of value to the desired MAS behavior, and not necessarily the sum total raw capability of the combined sensor payloads.

Dedication

To the men and women of our Armed Services. We have served in war and in peace together to defend America and its way of life. May you have God's speed and blessing.

Contents

List of Tables

List of Figures

List of Nomenclature and Abbreviations

ς	single-agent state space
G	global state space
N_A	number of agents/robots
$\dot{x}_\iota, \dot{z}_\iota, \dot{\phi}_\iota$	absolute velocity and angular velocity
v_i	linear velocity
ω_i	rotational velocity
x_i, z_i, ϕ_i	absolute position and orientation
$u_i, \dot{u}_i$	position and velocity of the i^{th} agent
u_{MAS}	vector representing the positions of the MAS agents
r_i	i^{th} robot
v_E, ω_E	systematic velocity and angular velocity odometry errors
$\delta_R, \delta_L, \delta_B$	odometry errors
B	perpendicular distance between the wheels on a robot
x_p, z_p	coordinates of the object's incident point
β	offset distance that correlates a specified sensor return value with its instrumented distance
θ_p	offset angle of the specified proximity sensor
$\emptyset_i$	robot's heading angle
τ	applied torque
s	number of proximity sensors utilized
βr_i	Braitenberg Matrix calculated based on physical parameters of the robot utilized and desired reactive control behavior
ϑ_i	collected sensor value
E	energy consumed
Δ	distance traveled
$Uatt$	attractive potential function
ξ	potential function scaling factor

m	potential function shape factor
F_{ATT}	attractive force
∇	gradient function
U_{REP}	repulsion potential function
ρ_0	distance of influence
F_{REP}	repulsive force
σ	velocity scaling parameter
$\hat{F}_i$	force vector witnessed by the i^{th} agent
Q_i	inherent charge of the i^{th} agent
Q_j	inherent charge of the j^{th} agent
q_j	inherent charge of the i^{th} cell
rad_j	radius vector from the i^{th} agent to the j^{th} agent or j^{th} cell
N_C	number of cells in the occupancy map
P	calculated force contribution from specified payloads
μ_x	tuning coefficients
q_i	charge of the i^{th} cell
P_O	cell occupancy probability
q_{max}	maximum possible cell charge
V	number of visits to the ith cell
n	number of times that a percentage of the cells have been visited
FR_{ij}	agent-to-agent repulsive force
γ	statistical skew
N_d	number of data points
$\bar{x}$	statistical mean
sd	statistical standard deviation

CV	Coefficient of variation
DARPA	Defense Advanced Research Projects Agency
DLL	Dynamic linked library
EPFL	Ecole Polytechnique Federale de Lausanne
GPS	Global positioning system
MAS	Multi-agent system
MAST CTA	Micro Autonomous Systems and Technologies Collaborative Technology Alliance
MEX	MATLAB executable
MOS	Measure of success
R&D	Research and development
RAMA	Robotics and Manufacturing Automation
SLAM	Simultaneous localization and mapping
SWIR	Shortwave infrared
TCP/IP	Transmission control protocol/internet protocol
TT	Tracking Tools
UDP	User datagram protocol
US	Unmanned system
VRPN	Virtual Reality Peripheral Network

Acknowledgements

Completing a graduate education while simultaneously working full-time as a program manager for the Army Research Office has indeed been a challenging task that was enabled only through the generous and gracious support of both my academic and military leaders and managers. First and foremost, I must thank my advisor and great friend, Professor Devendra P. Garg. You are a remarkable person indeed who I strive to emulate on many levels. Your tutelage, guidance, motivation, passion, and spirit are unmatched. I have been fortunate indeed to have studied under your charge. Additionally, I must thank Dr. David Skatrud, Director of the U.S. Army Research Office (ARO). I have tremendous admiration and respect for you personally and professionally, and especially for your love of fundamental research. I greatly appreciate your undying support to my career and education. You serve our military with great distinction and have been a tremendous mentor and friend. Finally, special thanks to ARO's Dr. Randy Zachery. You are a great friend who always went the extra mile. Your guidance and assistance were invaluable on so many levels.

I would also like to give special mention to others who have been critical in my endeavor. Many thanks to the other faculty who served on my Ph.D. qualification, preliminary exam, and final defense committees for their time and effort: Dr. Silvia Ferrari, Dr. Brian Mann, Dr. Dejan Milutinovic, Dr. Loren Nolte, Dr. Lawrence Virgin, and Dr. Gary Ybarra. Also, special thanks go to my Robotics and Manufacturing Automation laboratory colleague and good friend Greg Fricke who, with an uncanny

breadth of experience and depth of knowledge, was invariably willing to help and support. Finally, many thanks and much appreciation to Marianne Hassan, Jerry Kirk, Patrick McGuire, Kathy Parrish, Katie Rogers, and Michele Thompson for their invaluable and superb administrative and technical support.

In the true spirit of military esprit de corps and teamwork, many on the Army side were key enablers: LTG Ross Thompson, LTG Bill Phillips, and MG Nick Justice for supporting me tremendously both personally and professionally, allowing me to remain in graduate school and in North Carolina at a time when requirements for colonels peaked; and Mr. John Miller, Dr. Chris Arney, Dr. Cliff Wang, Dr. Tom Doligalski, Dr. Ralph Anthenien; Dr. Brian Ashford, and the rest of the ARO team for easing the work load and affording me time for independent research.

Finally, to my family whom I dearly love, cherish, and respect. My wife Norma and son Brian march on while I'm away working and researching, affording me the time and support I needed to progress. But moreover, you mean so much to me and fill so much of my life with happiness and love. There is no way I could have accomplished this without you. To my parents, Alice and Stanley Young; sister, Stacey Young-McCaughan; her husband Patrick; brother Steven Young; his wife Galane; and in-laws Manual and Lucila Esparza. You each have impacted me in so many different and important ways, be it your ambition, accomplishment, moral character, caring, or drive. I am a lucky man indeed to be able to call you mine.

Chapter 1. Introduction

The concept of organizing groups of robots or agents into "multi-agent systems" (MAS) has existed for several years. Research in the field has centered on understanding the root mechanisms and control features that can define and drive the behavior and capability of MAS architectures in terms such as task allocation, flocking, or foraging [1]. The study of analogous biological systems has also provided valuable insight. The Army's Micro Autonomous Systems and Technologies Collaborative Technology Alliance (MAST CTA) [2] is exploring bio-inspiration in conjunction with significant reductions in agent size and weight in pursuit of man-portable capability against surveillance and detection missions (Figure 1).

Figure 1: Robots in a Multi-agent System Cooperating in an Ordnance Disposal Mission
(Image by MAST CTA and BAE Systems, mast-cta.org, April 5, 2010)

However, to date, there has been little work done with respect to understanding or optimizing the capability of the MAS as a function of its composition in terms of quantity or capability, given the option of pre-mission selection of the performance and capability of individual agents. As an example, Figure 2 illustrates a mission to maneuver agents so as to maximize target-object imaging coverage [3]. A representative independent variable set here might be the optical range and resolution of the agents' sensor suite. Three agents are placed in the operational environment. The coverage percentage is simply defined as the percentage of the target's surface area that can be observed by the MAS' sensors. Then, sensitivity analyses explore the impact that changes in relative and absolute positions (illustrated here by travelling along the red arrow) have on coverage percentage.

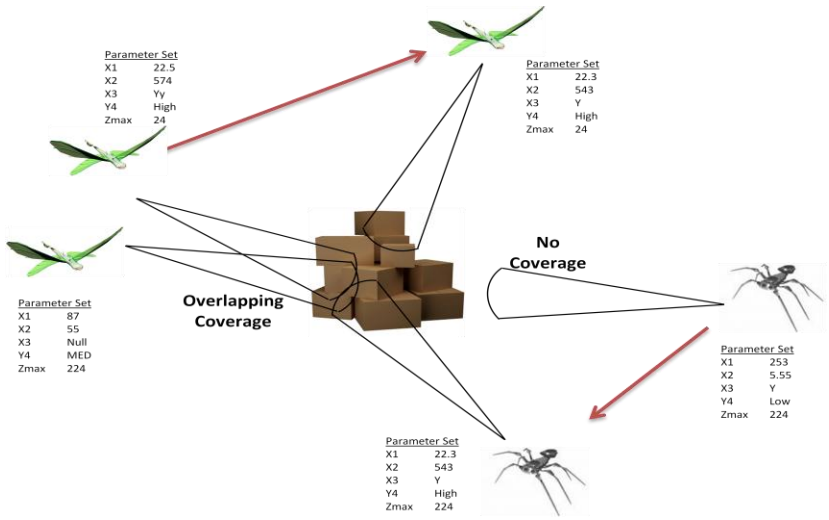


Figure 2: Maximizing Target-object Optical-coverage Percentage
(Agent images only by MAST CTA and BAE Systems, mast-cta.org, April 5, 2010)

One can see that the motion planned is very much a function of the capability of the sensor. Range to the target certainly will be calculated based on the resolution and magnification capabilities of the imager. However, this is a reactive calculation. How can planners become pro-active, meaning how can planners select payloads or inherent characteristics that afford better opportunities to successfully accomplish the surveillance mission?

1.1 Motivation

The ability of a MAS to perform complex missions beyond the capability of today's existing unmanned systems will principally be a function of three advancements. First is the reduction of the electrical and mechanical components of the physical configuration while increasing or maintaining desirable performance characteristics. There are several design options to be considered (e.g., legs versus wheels versus tracks) and the associated cost-performance tradeoffs in terms of power consumption, velocity, maneuverability, and motion characteristics. The second is the characterization of the behavioral control mechanisms on the MAS, such that complex tasks can be performed with significant mission accomplishment in a dynamic set of boundary conditions, core capabilities, and constraints. Third, and the principal motivation for this research, was increasing the capability of the MAS itself as a function of composition. Due to the inherent flexibility and variety that robotic systems typically offer, it is merely intuitive that the strength of a MAS' capability will be in its diversity. The main challenge was to

refine the tremendously large number of variations of MAS compositions and identify the most significant mission planning design considerations. Motivation to explore this was high due to the potential value added in terms of a better ability to define the MAS architecture components as a function of component cost, capability, and mission robustness. Further, motivation was also highly based on the forecasted ability to significantly influence important design considerations in terms of defining the relative advantages and detriments of components and agents; defining valuable, low-burden information collection techniques; and providing opportunities for injection of overarching architecture composition and intelligence schemes.

1.2 Problem Statement

Unmanned systems (US), both airborne and ground-borne, have provided a tremendous capability to accomplish tasks with the obvious advantage of removing operators from proximity to what could very well be a hazardous or dangerous environment. However, the vast majority of today's commercial systems thus far have been measured in tens if not hundreds of pounds in weight, and feet to tens of feet in dimension. While invaluable in their capability, these systems also exhibit a number of functional shortcomings due to their sizes. These include limited maneuverability in confined or complex urban environments, significant logistic burden, and very unique materiel construct. In order to continue US' mission-value growth, research and development (R&D) is exploring miniaturization technologies that will enable mission-capable US in the one pound class, or even lighter. However, this miniaturization will

likely come with some detriment to the inherent capability of the US itself. Therefore, other accommodations will be required to maintain overarching capabilities.

One potential solution is combining these US into a coordinated multi-agent system (MAS). A number of research efforts have considered or are currently considering various aspects and enabling technologies of MAS including cooperative motions (swarming behaviors) [4], locomotion, power, sensing (both for mission and for self), processing, and communications [5-12]. This research has proposed developing a better understanding of MAS mission performance via development of a systematic and comprehensive formal approach towards modeling and control of a large number of autonomous mobile agents working in coordination in dynamic and uncertain environments with a goal to meet specified task objectives. Representative tasks included collaboratively mapping an infrastructure, sensing obstacles, and locating targets. Germane to this capability was to understand the inherent performance limitations of the MAS. For example, the physical sizes caused limitations in available processing power, sensor capability, communications range and throughput, and mobility range/speed/duration.

The primary thrust of this research project was to explore novel methodologies to characterize the composition of MAS architectures assuming that the mission planners have latitude to define the components comprising the architecture. The composition was evaluated in terms of the ability to accomplish representative missions and scenarios. The composition itself was defined as a function of 1) performance and capability characteristics of individual agents within the MAS; and 2) the census of the agent

quantity and type. The analysis also explored the definition and impact of control forcing functions that can and should influence the overarching MAS control scheme to have a positive impact on its behavior.

The agent performance and capability was defined in a large number of classic terms such as velocity, relative or absolute position, orientation, mission life, and maneuverability. Therefore, the composition of the MAS had an extremely large number of combinations replicating a situation where mission planners could mix and match from select components.

1.3 Approach for Solution

The solution methodology centered on a characterization improvement of the MAS construct via a sensitivity and trade-off analysis conducted in the inter- and intra-agent parameter space. In order to accomplish this, it was of paramount importance that the representation of the parameter space be ubiquitous such that it could accommodate a wide variety of MAS configurations among highly dynamic scenarios and conditions.

Each MAS was described by a comprehensive variety of possible characterizations. That set included such parameters as relative or absolute position, sensor image resolution, maximum velocity, maneuverability, mission endurance, mobility, or communications throughput. These parameters span from highly qualitative, through quantitative, to perhaps even only binary values recognizing that there must be an intuitive description of the MAS so as to facilitate mission-user interface. As such, the actual values that these parameters attained included broad characterizations (e.g., high or

low), continuous or discrete scale (e.g., 3 feet or 10.75 ft/sec), probabilities (e.g., 95 percent detection probability), and binary (e.g., passive infrared sensor trigger state). However, the qualitative parameters had to be translated into representative quantitative terms for use in the control mechanisms. To be exhaustive and comprehensive, each MAS was described with the same overarching state vector where parameters that do not apply were simply set as null. Along with the nodal state parameters, there were also descriptions of other states of inter-nodal activity (e.g., relative position, location, and velocities).

The next step was to define the entirety of the MAS architecture composed of the variety of agents along with any other contributing capabilities. There exist numerous variations of that construct given that each agent parameter could be an independent variable, and that parameters such as the total number of agents, agent configuration, and agent groupings were also independent variables. In light of this, constraints and assumptions were applied based on current and projected pragmatic capabilities in line with the scale and mission set of the representative scenarios. For example, dimensions in the order of microns, and agent quantities in the hundreds were suppressed. However, viable architecture characteristics included, for example, ground-borne agents configured in MAS groups with distinct heterogeneous or homogeneous sensor capabilities dispersed among the agents.

The sensitivity analyses consisted of varying independent or small combinations of independent variables while holding the other variables constant. In order to understand the impact of the variations, evaluation criteria and metrics were constructed

that assessed the overarching performance of the architecture against pre-defined missions sets and scenarios. Initially, this was limited to small samples.

Iterations of the sensitivity analysis explored the broad sample spaces among the more intuitive characterization improvements to both provide early validation to the experimental models and to enable identification of the more widely-described conclusions. Armed with that understanding, and the awareness of emerging results, more complex behaviors and combinations of agents, parameters, and architectures were explored.

The sensitivity analysis itself was conducted analytically relying upon mathematical modeling of the parametric relationships and definitions. In order to develop a better understanding of the complex relationships and interactions involved, a comprehensive graphical simulation represents the architectural and environmental space while illustrating the MAS' behavior comparing and contrasting optimal, near-optimal, and sub-optimal solutions. Finally, a number of parametric characterization improvements representing the most novel of those witnessed were chosen and implemented experimentally using facilities available in Duke University's Robotics and Manufacturing Automation (RAMA) laboratory.

1.4 Research Contributions

The principal contribution of this research was gaining an understanding of characterization improvements of MAS architectures assuming that the mission planners have latitude to define the components comprising the architecture.

Given this trade space, the control system enhancements achieved by this research foretell significant capability enhancements while accommodating the MAS' materiel shortcomings. In view of the tremendous array of system configurations, this research enlightened the tradeoff process and characterization improvements so as to allow a system's maximum capability while minimizing burdens in terms of communication bandwidth requirements, cost, and/or processing power required.

Additionally, a novel potential-function control scheme linked with state space representation resulted in a robust and complete guidance mechanism with which to provide distributed control of the agents comprising the MAS.

1.5 Summary of Results

A series of individual vignettes and excursions explored varying MAS parameters such as the number of imaging payloads, number of agents comprising the MAS, and relative agent and cell charge values. These were evaluated against mission performance criteria such as time to explore a selected percentage of the identified space or the number of positive detections collected by proximity sensing or imaging. With these parameter and metric sets, methodical experiments were conducted sequentially to result in datasets that were reduced and interpreted for qualitative and quantitative characterization of the relative and absolute performance of the various MAS configurations. Results provided valuable insight to the behavior of the various MAS configurations (i.e., tradeoff between MAS capability and consumption) such that a priori mission planning could be optimized as a function of prioritized and weighted criteria

and planning factors. The most interesting conclusion drawn was that the relative performance of a MAS as a function of the number of agents comprising the MAS continually increased, but faded non-linearly. This means that more is not always better and that there will certainly be a point at which increased performance does not justify the associated logistic burden borne. Enhancements to classic potential-function control theory in a state mapping construct through the use of tuning coefficients proved very successful. Mission completion times were reduced 25 percent across the variable space.

1.6 Dissertation Organization

The next chapter provides general background on mobile robots, robot control theory, the concepts of multi-agent and swarming robotic systems, kinematics, local collision avoidance, and system measures of success. Chapter 3 introduces the concept of state space representation and its value for databasing knowledge of the MAS and the environment. In Chapter 4, potential functions are described whereby surrogate charges influence the path planning of agents. Tuning factors are introduced that provide a methodology to increase system performance as a function of sensing. Chapter 5 summarizes the test excursions accomplished in simulation while Chapter 6 summarizes those accomplished in hardware experimentation. Chapter 7 summarizes the conclusions drawn by the research en toto and offers suggestions for future work. Several appendices are included that document data graphing and software coding.

Chapter 2. Background

2.1 Mobile Robots

The term "robot" comprises a tremendously wide variety of technologies extending from the multiple degree-of-freedom arms and manipulators used for manufacturing tasks, through intelligence and autonomy that provide logic to control motion and processes, and into fully self-contained, self-directed, and self-controlled entities that, as the years go by, get closer and closer to mirroring the capabilities of human beings. One subset of the greater robotic spectrum is "mobile robots" (Figure 3) typically (although not exclusively) characterized by some sort of ambulatory vehicle, power source, data storage and processing, and sensing. In broad terms, they can be grouped as ground, air, water-surface or underwater vehicles (or perhaps combinations thereof). They also may be wheeled, winged (rotary or fixed), tracked, or legged.

Control runs the gamut from tele-operation (where a person manually and remotely controls the robot motion) to full autonomy (where the robot itself uses on-board logic to sense the environment, process decisions, and execute motion). There are also several degrees of semi-autonomy where operators inject decisions or command actions at various points in the high- and low-level control loops. Within these characteristics, there are almost infinite combinations to witness in academia and retail industry as researchers explore the inherent strengths and weaknesses of the components in an effort to provide materiel solutions to accomplish real-world tasks. Likely the earliest examples appeared during World War II with the Ruhrstahl X-4 wire-guided air-

to-air missile (never operationally deployed) and the V1/V2 series of rockets featuring crude autopilot mechanisms. In the late 1940s, Elsie (Figure 3a) and Elmer [13] appear as autonomous robots that explored by using a light sensor to move toward light sources while avoiding obstacles. One of the more famous commercial robotic products is the Roomba (Figure 3b) developed by iRobot [14]. Roomba is an autonomous mobile robot used to vacuum floors that has also spawned the Scooba to wash floors, the Verro to clean pools, the Looj to clean gutters, and the Create to serve as a research platform.

(a) **(b)**

Figure 3: Elsie and Roomba Mobile Robots
(Images by W. Grey Walter, extremenxt.com/walter, and iRobot Corporation, iRobot.com, respectively, February 2, 2011)

Arguably one of the most impressive robotic achievements is the BigDog legged robot [15] (Figure 4) developed by Boston Dynamics under support from the Defense Advanced Research Projects Agency (DARPA) that demonstrates phenomenal capability to walk on a variety of terrain and balance when physically perturbed.

Figure 4: BigDog Legged Mobile Robot
(Image by Boston Dynamics, bostondynamics.com/robot_bigdog, February 2, 2011)

2.2 Robot Control Theory

The logical control of a robotic entity provides an intelligence and decision-making capability that underpins the robot's ability to accomplish tasks of interest. Much like the variety of categories that comprise robotics itself, control has numerous sub-categories that describe varying levels of complexity and capability. Closed-loop control, classically implemented with the well-known proportional+integral+differential (PID) controller, uses feedback mechanisms in a return loop to inform the control processor for the next iteration of action determination. Within the realm of control, topics of interest include determining the stability of the process (pioneered by Alexander Lyapunov in the

1890s), controllability, observability, optimality, and robustness with their obvious benefits in terms of providing efficient, effective, and comprehensive capability to achieve the desired actions.

One control characterization important for this research is centralized versus decentralized control. This characterization is regularly assigned to multi-agent systems. Centralized control is where some single entity provides the entirety of the decision-making capability to the other entities comprising the system. For example, consider a warehouse where robotic carts are tasked with moving inventory among reception, storage, and delivery (Figure 5). In this case, the cart itself has low-level control functionality that can steer the vehicle, sense localization beacons, and locally avoid obstacles. However, the overarching motion plan and associated timing for the cart to follow is generated from a central computer that has awareness of the parts management process and status, entire set of carts' locations and inventories, and desired performance parameters (e.g., throughput). Benefits of this scheme include reducing the cost of the remoted agents (i.e., less processing capability required) and excellent central awareness across the system. However, there are significant detriments including high communications throughput requirements (to transfer the relatively large amount of data from central processor to the remote agents) and agent failure where limited on-board processing reduces a capability to auto-recover.

Figure 5: Robotic Warehousing System
(Image by IEEE Spectrum Magazine, spectrum.ieee.org/robotics/robotics-software/three-engineers-hundreds-of-robots-one-warehouse, July 1, 2008)

In contrast, decentralized control is characterized by agents in a multi-agent system that are completely autonomous [16]. In this case, there is no central command authority. Rather, each agent has a variety of sensor or communications means to transmit and receive information about the environment, situation, and peer agents [17]. With that information, the agent uses on-board processing capabilities to make decisions on motion and actions, executing accordingly. Clearly, this scheme provides numerous benefits including robust MAS functionality, graceful degradation (a single agent failure does not necessarily mean system failure), recoverability (significant on-board processing enables the overcoming of failure modes), and reduced communications bandwidth

requirements. However, the significant detriment is that the cost and complexity of each individual agent increases due to the increased capability required.

2.3 Multi-agent and Swarming Robotic Systems

From the inception of the development of robotic technologies, researchers have generally concluded that, with many if not most applications, there are significant advantages to be gained through the coordination and orchestration of multiple robotic entities [18-26], or multi-agent systems. This is not difficult at all to imagine by simply observing Darwinian development in biological systems whereby two or four legs proved superior to one for locomotion; or separate arms each with a hand, fingers, and an opposing thumb proved superior for grasping and cooperatively manipulating objects. Further, it similarly is understandable that a small group of separate robotic entities (e.g., a small number of cooperating or coordinating unmanned ground vehicles) could also exhibit significantly desirable characteristics in certain circumstances [27]. Five significant motivations for using multi-agent systems include: task complexity; task distribution; resource distribution; parallel processing; and robustness through redundancy [28]. Some researchers have explored the advantages gained for specific tasks such as exploration where now the task can be accomplished by numbers of agents surveying the maneuver space versus a single agent tasked with exploring the entirety by itself [29-36]. Indeed, these become valuable against overarching task accomplishment; however, not without burden. Multi-agent systems add communications overhead and

workload with the requirement for inter-agent cooperation and knowledge. They also add complexity to path planning and collision avoidance schemes since now each agent's cooperating peer also exists as an obstacle to avoid [37]. Multi-agent systems, say on the order of 10 agents, increase this complexity. Beyond that number, perhaps tens or hundreds of agents comprise "swarms" that typically result in orders of magnitude complexity increase [38-41].

In an effort to better understand and overcome these complexities, researchers looked to nature for clues [42-44]. Ants, searching in cooperation for food, utilize complex exploration, trail marking, and mobility functions all with relatively low "processing power" on board [45]. They also exhibit remarkable teaming skills combining their physical strength to accomplish tasks such as transporting foodstuff (Figure 6). Studying behaviors such as flocking, foraging, and forming provides insight to the animal's decision making process and also conveniently defines motions with associated advantages and disadvantages [1]. However, research generally has been limited to these generic understandings en route to exploring more complex behaviors. Further, for swarm-sized systems, the focus has been on the swarm as an entity on to itself where the control and resultant behavior of the swarm might be very dynamic, but the composition therein is somewhat constant.

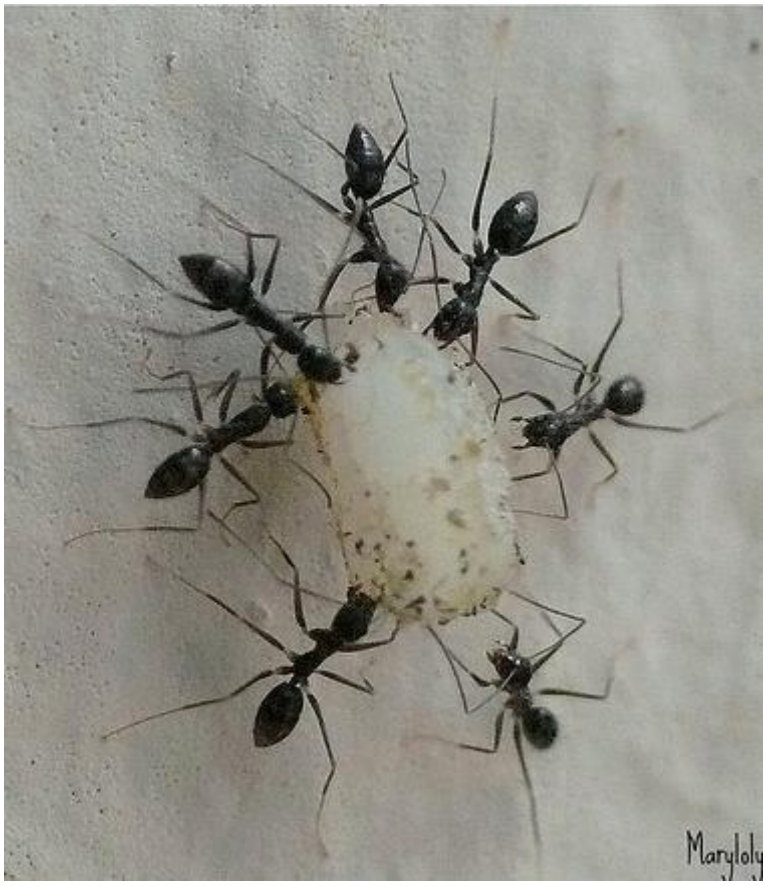


Figure 6: Ants Cooperating to Transport Food
(Image by Maryloly Guerrero, flickr.com/photos/maryloly/2436907185/in/set-72157608367566669, January 5, 2011)

Various attempts at defining MAS [46-50] composition explored heterogeneous versus homogeneous systems, highlighting concepts such as hierarchic social entropy, diversity, and cooperative localization. Here, the researchers begin to explore the value of acknowledging the inherent differences even among robots of similar design and function. However, they also extend the theories to cooperating agents of differing types and quantities, and also to configurations where sensor capability is spread among various agents and then becomes the concatenation of the series [51]. Component health

has also been researched as a consideration [52]. Heterogeneous MAS have been used to explore unknown environments for the purpose of mapping and exploration [53-59]. Here, each MAS featured varying types of single-mode sensor payloads, including for example, an infrared camera or sonar, coordinated in an occupancy-grid Bayesian mapping algorithm. But again, the composition of the architecture was held constant.

The underlying control architecture plays an absolutely critical role in the efficiencies observed in the multi-agent versus single-robot architectures [60-63]. Clearly, beyond the inherent capability of the electro-mechanical configuration as well as the sensor payloads, the single greatest contributor to mission efficiencies and task capabilities will exist in the capability and comprehensiveness of the control system. A wide variety of algorithms and schemes have been explored including feedback laws, leader follower, task-oriented mission planning, task reallocation and reordering, and deliberative and reactive behavior modeling.

Scalability in the MAS has proven to be a significant challenge in the construction of control architectures [64]. A single-agent controller has associated a state space represented by ς that in this case also comprises the global state space *G*. However, as the number of agents N_A in the multi-agent system increases, and assuming that each individual robot's state space is of equal size, the size of the global state space increases geometrically, or:

$$G = \varsigma^{N_A} \tag{2-1}$$

As such, centralized control of multi-agent systems will not typically scale well. Alternately, distributed control, or as in this research, distributed control with shared state space information (e.g., occupancy mapping) provides a viable methodology to control the multi-agent system in real time.

2.4 Agent Kinematics

Differential-wheeled robots comprise a large segment of the experimental robot type used in research today [65]. The e-puck robot [66-67] (Figure 7) used both in simulation and experiments to support this research is a classic example of a differential-wheeled robot. In this case, there are two drive wheels situated parallel and opposite to each other that can be independently controlled via applied torque. The robot is balanced by a rotating idler wheel or friction point located in the rear of the robot that simply provides the third ground contact point for balance and pitch. Forward motion is achieved by applying the same torque (in amount and direction) to both wheels. Applying different torques in the same rotational direction will result in left or right turns. A significant advantage to the differential-wheeled robot is in its ability to execute a zero-radius turn meaning that it can rotate about its central vertical axis with no translation by applying the same torque to each wheel, but in the opposite rotational direction. This configuration of robot is also described as being "nonholonomic" meaning that it cannot immediately translate in any direction. Rather, it must orient prior to moving in a particular direction.

Figure 7: Simulated e-puck Robot
(Image by Cyberbotics, Cyberbotics.com, July 15, 2009)

The kinematic equations governing differential-wheeled robots (Figure 8) are:

$$\dot{x}_i = v_i \sin(\phi_i) \tag{2-2}$$

$$\dot{z}_i = v_i \cos(\phi_i) \tag{2-3}$$

$$\dot{\phi}_i = \omega_i \tag{2-4}$$

where,

$\dot{x}_\iota, \dot{z}_\iota, \dot{\phi}_\iota$ are the absolute velocity and angular velocity,

v_i is the linear velocity,

ω_i is the rotational velocity, and

x_i, z_i, ϕ_i are the absolute position and orientation, respectively.

Further, the position of the i[th] robot u_i can be represented by the vector:

$$u_i = [x_i, z_i]^T, u_i \in \boldsymbol{R}^2 \tag{2-5}$$

The vectors representing N_A robots comprising the multi-agent system (MAS) is:

$$u_{MAS} = \left[u_1^T, u_2^T, \ldots, u_{N_A}^T\right]^T, u_{MAS} \in \boldsymbol{R}^2 \tag{2-6}$$

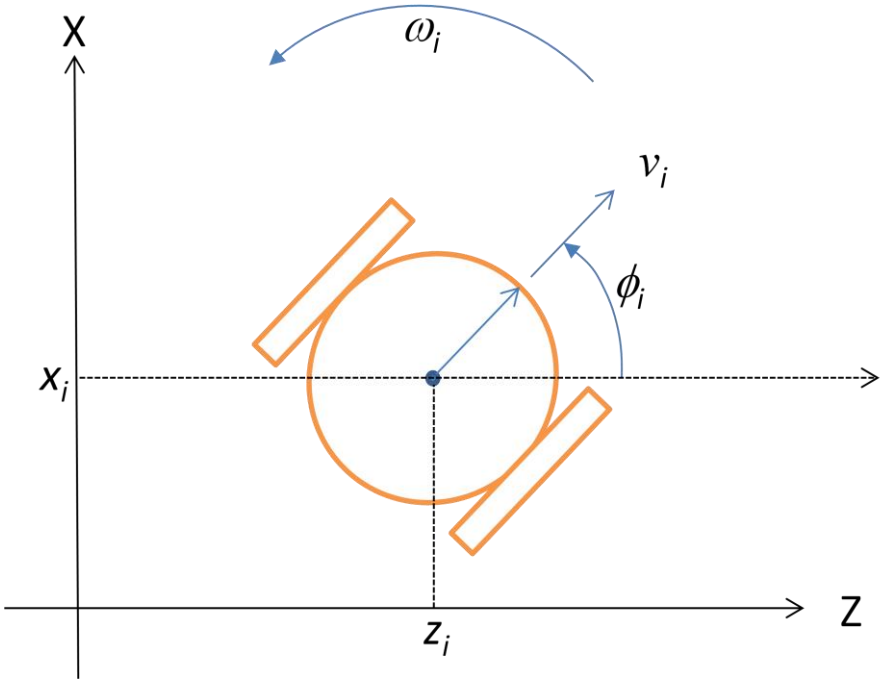


Figure 8: Differential-wheeled Robot Kinematics

Finally, literature offers a characterization of the systematic velocity v_E and angular velocity ω_E odometry errors that exist due to inherent inaccuracies in the mechanical system, principally due to manufacturing imperfections and wheel slippage [68]. These can be represented as:

$$v_E = \frac{\delta_R\, v_R + \delta_L\, v_L}{2} \tag{2-7}$$

and

$$\omega_E = \frac{\delta_R\, v_R - \delta_L\, v_L}{\delta_B B} \tag{2-8}$$

where:

δ_R, δ_L, and δ_B are the odometry errors, and

B is the distance between the wheels.

2.5 Local Collision Avoidance

There are a number of options to implement local collision avoidance mechanisms [69]. Many experimental robotic systems utilize proximity sensors in order to facilitate local collision avoidance. The e-puck robot is fitted with eight infrared proximity sensors mounted on the perimeter of the robot's body and oriented radially outward [70]. These sensors are located at +/- 16, 45, 90, and 150 degrees offset respectively from the robot's forward orientation. The functioning of the sensor itself is to emit a pulse of light and then quantify the reflected energy. Therefore, a relationship can be established between the reflected energy received and the distance to the object [71]. A stronger return indicates a closer object. Since each individual proximity sensor is not perfectly manufactured, each of the robot's proximity sensors must be calibrated to determine its exact characterization. That calibration then is incorporated into the control code so that the correct distance is correlated with the actual reflected energy quantified and reported by the sensor. The Webots simulation offers a very controllable

characterization of the proximity sensor whereby individual sensors distance/return relationships can be uniquely represented by complex curves. The default curve established by Webots is a straight-line relationship between energy received and distance to the object. Reported values are then distributed normally around the curve by a nominal 10 percent to represent environmental impacts such as reflectivity or ambient lighting.

With this sensor characterization, incident surfaces of objects (obstacles or targets) can now be localized (Figure 9). The logic here is to select a fixed sensor return value. Knowing the calibration curve, one can then calculate the distance from the robot's perimeter to the incident surface of the object. Further, knowing the robot's position and orientation, one can subsequently calculate the location in global coordinates of the incident surface given by:

$$x_p = x_i + \beta \cos(\theta_p + \emptyset_i) \tag{2-9}$$

$$z_p = z_i + \beta \sin(\theta_p + \emptyset_i) \tag{2-10}$$

where,

x_p and z_p are the coordinates of the object's incident point,

x_i and z_i are the robot coordinates,

β is the offset distance that correlates a specified sensor return value with its instrumented distance,

θ_p is the offset angle of the specified proximity sensor, and

$\emptyset_i$ is the robot's heading angle.

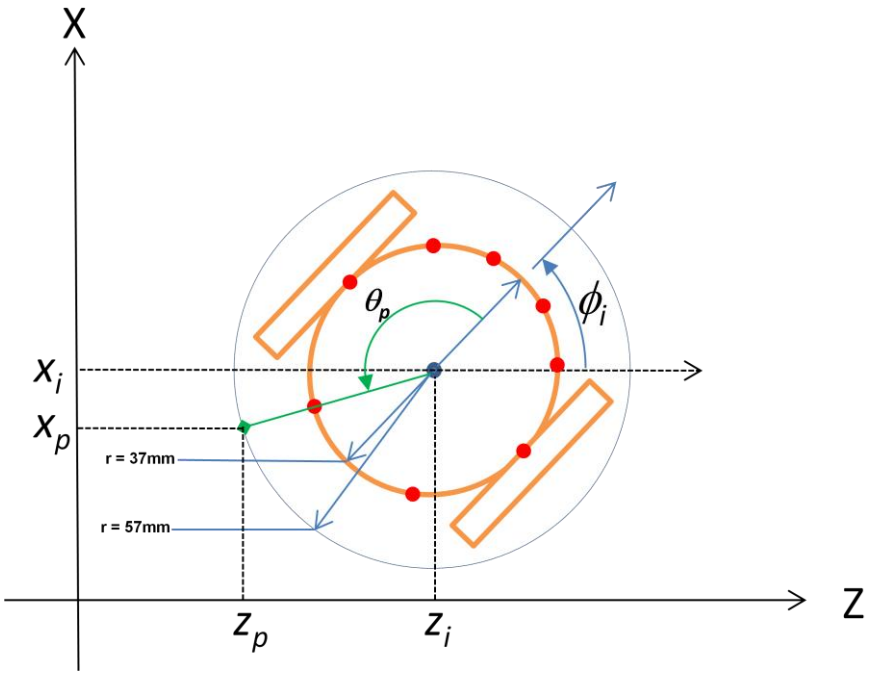


Figure 9: Proximity Sensor Geometry

Another significant advantage of the proximity sensors is utilization of a Braitenberg algorithm to execute local collision avoidance as described in the analytics paragraphs above. The basic concept is to calculate the individual wheel torques as a function of weighted proximity sensor values. The proximity sensors on the left side of the robot have a higher weighting for the left wheel forward torque calculation knowing that will result in a right-hand turn, i.e., a turn away from the obstacle. One experiment complexity is that typically, the individual proximity sensors are not perfectly uniform and therefore must be calibrated to generate a sensor value vs. obstacle distance relationship on a by-sensor basis. In execution, the eight proximity sensors are sampled on their respective absolute scales (as opposed to being thresholded). That vector of

eight values then is inputted to the Braitenberg matrix in order to result in specific wheel velocities as follows:

$$\tau = \sum_{i=1}^{s} \beta r_i \vartheta_i \tag{2-11}$$

where,

τ is the torque applied to the left or right wheel,

s is the number of proximity sensors utilized,

βr_i is the Braitenberg matrix calculated based on physical parameters of the robot utilized and desired reactive control behavior, and

ϑ_i is the collected sensor value.

An example Braitenberg matrix used in this research for the e-puck robot is:

Table 1: e-puck Braitenberg Matrix

	Prox 1	Prox 2	Prox 3	Prox 4	Prox 5	Prox 6	Prox 7	Prox 8
Left wheel	-1	-1	-.05	0.5	-0.5	0.5	1	1
Right wheel	1	1	0.5	-0.5	0.5	-0.5	-1	-1

Research has also explored extensions to the Braitenberg algorithm using a variety of techniques to refine the parameter tuning in a dynamic environment. One example leverages fuzzy logic to generate a tuning scheme based on distribution directives of the proximity sensors [72].

2.6 Measures of Success

The greater MAS architecture is comprised of very large numbers of parameters and variables that have the potential to provide viable measures with which to evaluate the performance of the MAS [73]. The architecture may have a number of grouped agents. Each MAS may consist of a number of agents characterized individually as ground borne, airborne, immobile, single sensor, multi-sensor, and line of sight or non-line-of-sight-communications capable. Further, each characteristic has associated performance values in terms of range, power, lifespan, velocity, and resolution. Finally, many of these characteristics and parameters can change as a function of time and under a completely dynamic and largely unknown mission environment.

The measures of success (MOS) evaluated in this research are defined both in terms of overarching system and sub-system ability and accomplishments. These MOSs emerge as quantitative values at the sub-system level, concatenating to qualitative terms at the system level. Hence, for example, sub-system performance is described with values such as total power consumption, distance traveled, communications availability, data throughput, and data collection rates. System-level performance is evaluated in terms such as target-object understanding, maneuver space understanding, surveillance-coverage efficiency, or time to mission success. The key to understanding the resultant accomplishment is minimizing and controlling the number of independent variables such that the influences of varying control parameters can be comprehensively understood.

Functions that represent these MOSs include summing the energy consumed E by the multi-agent system, or the cumulative distance traveled Δ from some starting

configuration through to the point where mission success is achieved. These values can be calculated as:

$$\sum_{t_{initial}}^{t_{final}} \sum_{i=1}^{N_A} E_i(t) \tag{2-12}$$

or,

$$\sum_{t_{initial}}^{t_{final}} \sum_{i=1}^{N_A} \Delta_i(t) \tag{2-13}$$

Chapter 3. State Space Representation

The environment and its contents that comprise the world that we live and operate in today are vast indeed. One can only imagine the size of the database that would be required to catalog each and every object description including its size, shape, position, orientation, physical characteristics, color, or even more obscure parameters. Nonetheless, this information is of paramount importance for use as input data for robotic applications. One clever databasing methodology is through the use of state space representation [74-79]. In this case, the physical space of interest is broken up into a discrete spatial/temporal two- or three-dimensional lattice described as cells. Each cell can then have its own state vector that can include sets of parameters, e.g., stochastic estimates of the occupancy probabilities.

3.1 Occupancy Mapping

For the purposes of this research, the simulation and experimental arenas were both set at 1.2 meters by 2.4 meters and limited to 2 dimensions. That area was then sub-divided spatially into 120 by 240 discrete 1 cm^2 cells. Within each cell's state vector, there is an occupancy probability element that represents the probability the cell is occupied by some physical entity (obstacle or target). The probability values range from 0.0 to 1.0. A value of 0.5 implies that the probability that the cell is occupied equals the probability it is unoccupied, i.e., that there is a complete lack of understanding. A value

of 1.0 implies that there is a 100 percent probability that the cell is occupied. A value of 0.0 implies that there is a 100 percent probability that the cell is unoccupied.

Depending on the initial boundary conditions of the experiment or the state of understanding at initiation, the occupancy probability is set. For this research, the agents begin with no understanding of the operating arena; therefore, all occupancy map probabilities are initialized to 0.5 at t=0 for each experimental run.

Just as in the real world, the occupancy map can be updated based on any new information that the agents glean from the environment through the use of any of their sensing capabilities. For example, if the agent has benefit of a GPS sensor, it then knows its own position and orientation. That, plus an understanding of its own inherent dimensions would allow it to update the cells co-located with its own presence concluding that if it exists at those cells' locations, then there are no objects present. GPS plus proximity sensing can extend that conclusion outward to the threshold perimeter of the proximity sensing. Alternately, if an object is detected by a single proximity sensor, the cell located at that boundary can be updated with the presence of the object. Note that unless a sensor can penetrate objects, this sensing scheme limits occupancy map updating to the outer perimeter of the object. Finally, if the agent utilizes an imaging sensor and image processing, the resultant data might include information on the presence or absence of objects that can be used to update occupancy mapping.

One favorite methodology to update cell information is through the application of Bayes' theorem [80-81]. This approach states that the conditional probability (also referred to as the posterior probability) of a state matrix (the independent variable of

interest), can be estimated using the current observation of the state in combination with the previous understanding of the state. This relationship can exist in both discrete and continuous probability distributions; however, given the nature of iterative control theory, discrete representation provides the more valuable implementation whereby the occupancy probabilities at step N+1 can be calculated based on the existing probability (i.e., at step N) in conjunction with the sensor readings at that step. For the discrete case, the theorem states that:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\bar{A})P(\bar{A})} \quad (3\text{-}1)$$

where,

P(A|B), or "posterior probability," is the conditional probability of A given B,

P(B|A), or "likelihood," is the conditional probability of B given A,

P(A) is the prior or unconditional probability of A,

$P(B|\bar{A})$ is the probability of making a false observation, and

$P(\bar{A})$ is the probabiltiy of "not" A.

For application to probabilistic occupancy mapping, those parameters are defined as follows:

- *A* is the event whereby an obstacle exists (i.e., a cell is definitively occupied).
- *B* is the event whereby an obstacle is detected by a sensor.
- *P(A|B)*, the desired value, is the newly-calculated probability that an obstacle exists (i.e., cell is occupied) given an obstacle was detected by an agent's sensor.

- $P(B|A)$ is the probability that an obstacle is observed when it actually exists. This is the detection probability of the sensor calculated as a function of the physical capability of the sensor itself.
- *P(A)* is the probability that an obstacle exists on the grid. In practice, this is the existing probability of occupancy at the cell.
- $P(B|\bar{A})$ is the false positive, i.e., the probability that an obstacle is observed when one does not exist. This too is a function of the inherent characteristics of the sensor.
- $P(\bar{A})$ is the probability that an obstacle does not exist on the grid.

3.2 Cell Charge Contribution to Potential Functions

Given the mission of exploration of an unknown environment, occupancy probabilities offer valuable input information for the path planning calculations knowing that the ultimate goal is to search the space and database its state. The methodology proffered in this research to accomplish this is to assign an attractive charge to each cell as a function of its occupancy probability (Figure 10). The highest attractive charge would be assigned to cells with occupancy probabilities of 0.5 meaning that exploring agents should have the highest relative attraction toward the complete unknown. The lowest attractive charge would be assigned to cells with occupancy probabilities of 0.0 or 1.0 knowing that there is no desire to explore cells whose state awareness is completely

known. Between these extremes, charge values can be distributed linearly as seen in the left curve of Figure 10 as utilized in this research.

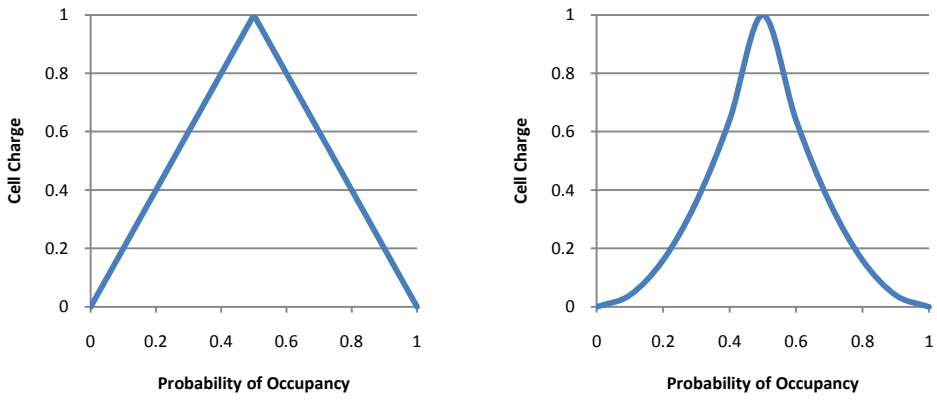


Figure 10: Cell Charge versus Probability of Occupancy

Knowing that each agent has a point charge associated with it, the attraction can be calculated to draw the agent with greater priority, in a global sense, to those cells that heretofore are unexplored. Note that there is an intuitive efficiency gained in this scheme. A single unknown cell isolated in a field of known cells has a minimal attractive influence to the agent's path planning. In contrast, a group of 10 or 20 unknown cells will concatenate their collective charges to generate a higher attractive force with which to draw in the roaming agent. The efficiency exists in that the agent is proximate to the group of cells and can explore their entirety without having to wander back and forth over already-explored space.

A subject of follow-on research might be to explore the impact of shaping the cell charge curve. For example, the right side of Figure 10 shows a second-order polynomial

cell charge curve that places proportional emphasis on the unknown cells effectively magnifying their influence in the overarching path planning calculations. This might very well have a positive effect on MAS exploration times and efficiencies.

3.3 Time Variation of Occupancy Mapping

The nature of occupancy mapping and its updating allows it to be very robust in view of time-varying conditions (such as transient moving obstacles) and erroneous data. As an example of the former, assume an agent's sensor has correctly detected the presence of a peer agent through its proximity sensing. That data element is updated in the occupancy map accordingly by increasing the occupancy probability by some calculated value. However, since the agents are moving, that cell likely becomes unoccupied in the very next timestep as the peer agent moves. Until such time that another agent senses that cell, the occupancy probability will remain unchanged and in fact lean toward occupation because of that single transient detection. Nonetheless, the advantage of probabilities is that they are merely indicators versus absolutes. So, when another agent samples that now-unoccupied cell, it can decrement the occupancy probability accordingly bringing the state of space more close to truth at that point in time. Similarly, erroneous data (e.g., a sensor's false detection) might have some local skewing effects temporally; however, these too are quickly remedied by subsequent correct sampling that brings the occupancy probability back in line with truth.

3.4 Characterization of Objects in the Occupancy Map

A significant challenge posed in the occupancy map scheme is the handling and characterization of objects (targets or obstacles). There has been significant research on path planning based on the interpretation of the occupancy map assuming a priori knowledge [82]. The basis of occupancy map updating hinges on the ability of the agents' collective sensors to determine what, if anything, occupies a specific cell location in the occupancy map. For many of the cells and their respective contents, this is a relatively easily accomplished task. If the agent's body occupies the cell, then one can conclude for that timestep, the cell is not occupied by objects. Therefore, the agent's physical presence precludes occupancy by any other tangible object. If the agent's proximity sensor detects the wall of an object, then by knowing the instrumented calibration of the sensor (i.e., object range versus sensor signal return) and position/orientation of the agent, one can calculate the location of the object wall and conclude that the cell at that location is at least partially occupied. However, these sensors can typically only sense the outer perimeter of the object. Most do not have a capability to sense the interior of the object. In this case, the best conclusion that can be reached is that the full perimeter of the object is sensed with subsequent update to the occupancy map. But, the interior of the object remains unsensed resulting in the occupancy probability for those cells remaining un-impacted and unknown.

In the potential-function path planning scheme, this proves very problematic. The simulation phase of this research witnessed a significant shortcoming in mission accomplishment based on this dilemma. The interior cells of the objects would never be

sensed. Figure 11 is the resulting probability of occupancy mapping for a four-agent simulation run. This mapping clearly shows the four objects (in this case obstacles) existing in the arena that remain unexplored. Over time, this meant that these interior cells would overwhelm the potential-force calculations resulting in paths directed toward these interiors. Of course, the robots are unable to search the interior having been blocked by the object perimeters. The behavior witnessed was that the robots would wander the object perimeters with ever present desire to search the interior only to be thwarted by the object's boundary. This challenge was overcome in the experimentation phase. The fundamental requirement was to determine appropriate handling of the object interiors without benefit of direct physical sensing in the occupancy cell mapping architecture.

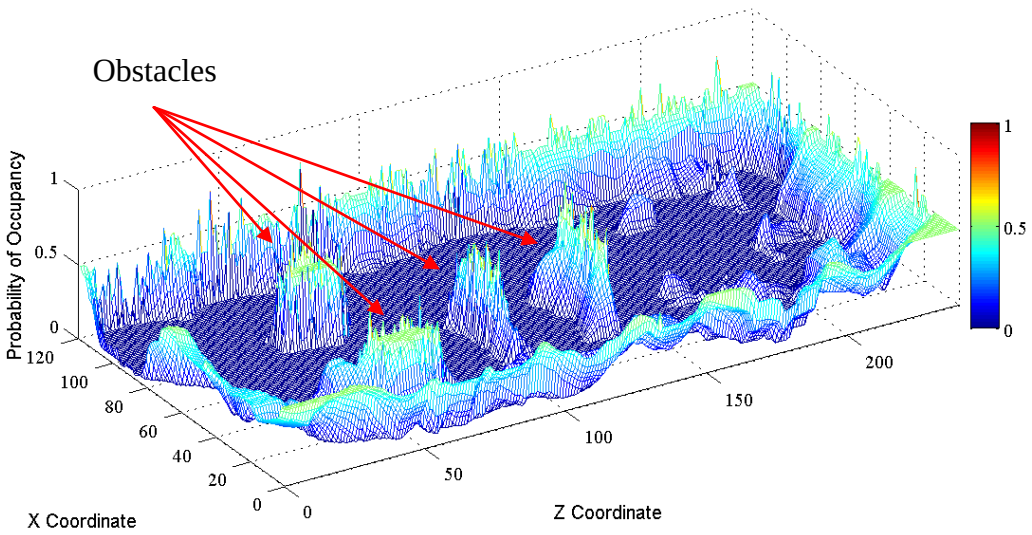


Figure 11: Unexplored Obstacles in Simulation

A novel solution was devised by treating the occupancy map as a pseudo-image and then applying powerful image processing algorithms to extrapolate information on areas unobtainable through classic physical sensing. There are strong analogies between two-dimensional occupancy map representations and visual images. Both are represented mathematically as two-dimensional matrices, be it the ordinate and abscissa of the physical mapping of the occupancy map, or the width and height of the planar image. Both populate the values within the matrix itself with physical data. The occupancy map puts occupancy probabilities or potential charges in the mapping. The image puts values ranging from zero to one in a single channel for grayscale images, or in three channels (red, green, and blue) for color images. Finally, and perhaps most importantly to validate the strength of the underlying analogy, both the occupancy mapping and images ultimately represent parameters of real physical existence which allows one to conclude that image processing techniques such as edge detection and object fill can be appropriately applied to occupancy mapping.

Given image processing as a validated technique, one can explore the underlying logic for the handling of objects in the occupancy mapping architecture. At the extreme where there is comprehensive and accurate information available, conclusions can be drawn quite easily. For example, let's assume that the proximity sensors of a robot have exhaustively explored the perimeter of a specific object and updated the respective cells of the occupancy map to a very high occupancy probability. Image processing then could be applied to determine that there is a closed object in the occupancy map "scene" defined as a locus of fully-connected cells comprising a closed boundary. Then, that

object can be filled meaning that all of the occupancy probability within the cells comprising the object's interior would be increased accordingly. However, it is likely more common that there will be something less than perfectly accurate and comprehensive information available about the occupancy map. For example, an agent's proximity sensor may only have accomplished sensing of 70 or 80 percent of the object's perimeter. Further, it may or may not be prudent to conclude that the artificial closure of the perimeter and boundary would accurately define the object. There could be many situations where this would not be valid, e.g., a hollow object with a single breech in the perimeter (Figure 12). In this case, the breech itself may only represent a very small percentage of the distance of the perimeter; however, allow full access for the robots to maneuver to the interior.

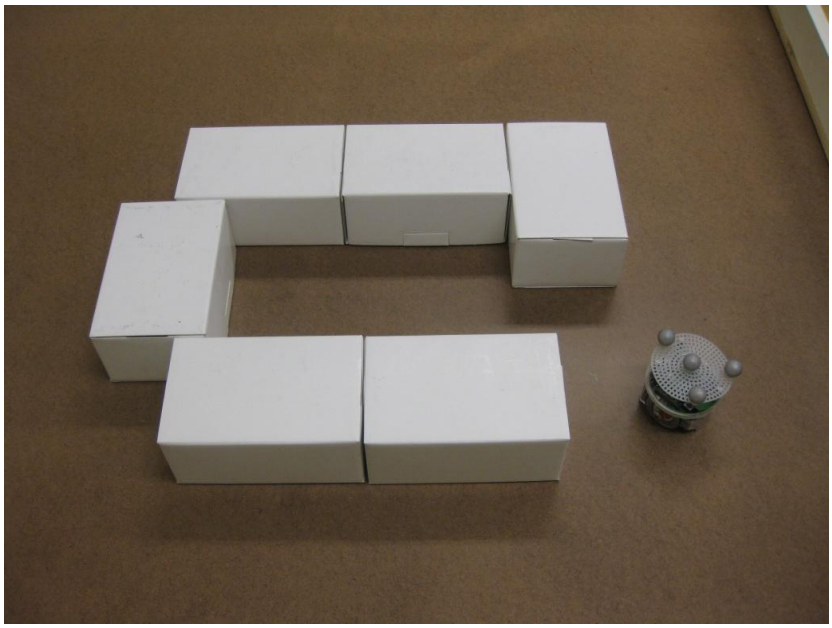


Figure 12: "Hollow" Object

The strength of using probabilities in the occupancy mapping architecture affords an elegant solution to this dilemma in that it allows one to assume that perfect knowledge is not required. Rather, the image processing techniques can be applied to impact the probability calculations with some associated confidence less than absolute. Therefore, even if errors are witnessed, subsequent sensing can still have viable corrective impact on the occupancy mapping without preclusion based on any image processing conclusions.

In application of image processing techniques, there is a trade-off between a priori knowledge and understanding of the environment, and the required complexity of the techniques. In application, it becomes extremely important to understand these trade-offs in order to streamline the efficiency of the processing algorithms and not overburden (in terms of time or power) the control scheme, or under-enable resultant capability. Extending the previous example of the filled versus hollow object, if one knows a priori that each object sensed consists of a filled convex polygon, then there would likely be greater inclination to accept perimeter gaps in the image processing computations.

For the research experimentation, MATLAB was selected as the software utilized due in part to its powerful Image Processing Toolbox and its comprehensive function set capable to manipulate matrices. The experiment scenario was defined such that the agents had no a priori information about the arena. However, they did have a fundamental understanding about the obstacles and targets that comprised the objects located in the arena. For example, obstacles were known to consist of convex polygons that had a pre-defined-maximum orthogonal cross-sectional area. Given these various characterizations, the object-fill algorithm utilized progressed as follows:

Table 2: Object-fill Algorithm Using Image Processing Techniques

Step	Action
1	*Initialize the working data set with the most current occupancy map.*
2	*Threshold data set at 0.49 to isolate regions of interest (i.e., image processing objects) that are either unknown or occupied.* The result is a binary mapping of the data set where '1' represents cells unknown/occupied and '0' represents probabilities of free space.
3	*Remove regions that are larger than the pre-defined object areas* including accommodation for margins of error. Knowing any object feature characterization, in this case area, allows the algorithm to reject image-processed objects that exist outside acceptable feature definitions. Other characterizations might include color, shape, height, temperature, etc.
4	*Identify regions that have perimeter occupancy map cell values greater than a pre-determined confidence threshold.* In this case, very high information surety is provided by proximity sensing. Therefore, fusing high occupancy probabilities at the object's boundary and the region identification determined previously significantly increases the overarching confidence that the object segregated is an identified obstacle or target.
5	*Conclude that those regions are objects and fill the region interiors* (i.e., increase the respective occupancy map cell values to the confidence threshold).

To illustrate the progression of this image processing algorithm, the following several figures show corresponding step sequences. Figure 13 shows ground truth consisting of the arena with several obstacles (white boxes) and targets (yellow and black cylinders).

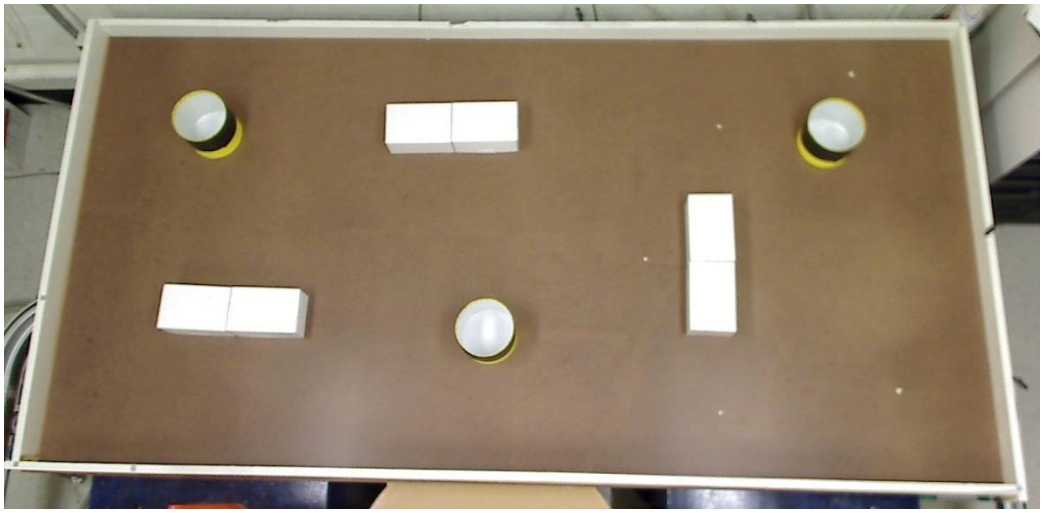


Figure 13: MAS Arena - Ground Truth

Figure 14 shows a representative occupancy mapping. In this case, the agents comprising the MAS have explored the region for several hundred timesteps during which time the cells' probabilities have been continually updated. The black regions indicate space that has been sensed and determined to have low probabilities of occupancy (i.e., free space). The gray areas represent unknown spaces. The white cells indicate the presence of objects determined either by proximity sensing or imaging. This comprises the input occupancy map in Step 1 above.

Finally, the algorithm as described is applied to this mapping to result in the image-processed occupancy map (Figure 15). In this case, regions have been identified to be obstacles or targets and filled accordingly as seen by the new white areas.

Figure 14: Occupancy Map Prior to Image Processing

Figure 15: Image-processed Occupancy Map with Objects Filled

Clearly, there are inaccuracies accommodated in this algorithm. The newly-filled objects do not exhibit fine resolution at their perimeters. Additionally, there are false

detections as indicated above by the red box where by mere coincidence, a region has exhibited characteristics within the definition of objects. Carefully refining the existing definitions, or adding other characteristics can reduce these inaccuracies. Nonetheless, therein lies the strength of the scheme. The processing time required is less than a millisecond. Further, the updated information removes the influence, albeit of poor resolution, of the heretofore hollow object from the path planning algorithm near instantaneously while not precluding continued sampling of the locations proximate to the perimeter of the object. Over time then, the resolution of the object will be continually refined. Given more complex scenes or less a priori knowledge about the obstacles and targets that comprise the space, a more complex and comprehensive algorithm may certainly be required to afford appropriate levels of accuracy in the representation and update of the occupancy map. Nonetheless, this baseline afforded a viable solution from which the experiment could accurately evaluate the multi-agent system's capability to explore based on census and payload capability.

Chapter 4. Multi-agent System Control via Potential Functions

Control via potential-field theory has proven to be a valuable basis upon which more complex control feedback influence can be implemented. In particular, potential-field theory affords a comprehensive and simple method to add factors that can be scaled either as independent input functions or dependently upon selected system features [83-89].

The basis of the potential field is to represent system control influences as independent or dependent forces [90]. Each force is calculated based on physical parameters (such as real distances) [91] or on representative values (such as arbitrary repulsion to facilitate obstacle avoidance or attraction to a desired target location). In conjunction with the potential field, it is convenient to represent the operational space as an occupancy map, i.e., a two- or three-dimensional grid that sub-divides the space into identifiable cells. Each cell would then have characteristics such as a potential related to its occupancy status or status as a target destination. Since potential is a function of relative proximity, the relative location of the cells would also be cataloged.

4.1 Classic Potential Functions for Path Planning

Applied specifically to a path planning methodology, potential functions offer a robust tool whereby the robot's operating space is filled with an artificial potential field representing attraction toward the goal position and orientation, and repulsion away from objects [91-97]. In the case of multi-agent systems, that repulsion can also include peer

agents. There are some inherent limitations to this scheme. Notably, potential functions tend not to accommodate local minima and narrow passages. However, there are a number of tailorings that can overcome these shortfalls including such examples as a strategic insertion of false point repulsion forces to reshape local minima.

In general, a common representation [98] of an attractive potential function $Uatt$ takes the form:

$$Uatt(r) = \frac{1}{2}\xi\left\|r_{goal} - r\right\|^{m} \tag{4-1}$$

where,

ξ is a positive scaling factor,

$\left\|r_{goal} - r\right\|$ is the distance between robot r and its desired goal, and

m is a factor that drives the shape of the potential function, e.g., conical for m=1 or parabolic for m=2.

Then, the calculated attractive force F_{ATT} is formed by the negative gradient ∇ of that attractive potential:

$$F_{ATT}(r) = -\nabla U_{ATT} = \xi\left\|r_{goal} - r\right\| \tag{4-2}$$

Similarly, a common repulsion potential function U_{REP} is represented as:

$$U_{REP} = \begin{cases} \xi\left(\frac{1}{\|r_{obst} - r\|} - \frac{1}{\rho_0}\right)^2 & if\ \|r_{obst} - r\| \leq \rho_0 \\ 0 & if\ \|r_{obst} - r\| \geq \rho_0 \end{cases} \tag{4-3}$$

where,

ρ_0 represents the distance of influence of the object.

The corresponding repulsive force F_{REP} is given by:

$$F_{REP}(r) = -\nabla U_{REP}(r) \tag{4-4}$$

$$= \begin{cases} \xi\left(\frac{1}{\|r_{obst} - r\|} - \frac{1}{\rho_0}\right)\left(\frac{1}{\|r_{obst} - r\|^2}\right)(\nabla\|r_{obst} - r\|) & if\ \|r_{obst} - r\| \leq \rho_0 \\ 0 & if\ \|r_{obst} - r\| \geq \rho_0 \end{cases} \tag{4-5}$$

And therefore, the total force witnessed by the robot is:

$$F_{TOT} = F_{ATT} + F_{REP} \tag{4-6}$$

4.2 Potential Functions for Exploration and Mapping

A second application of potential function methodologies, and the basis for the research conducted herein, is in exploration and mapping. In this case, instead of having an attractive force exist at the goal location, calculated attractive forces exist at each and every cell in the occupancy map representing the operating space. The magnitude of the force existent at each cell can be defined by some metric such as the relative desire to search that space (e.g., occupancy probabilities as described previously).

For the MAS, potential field theory also conveniently provides a basis with which to relate inter-nodal relationships and capabilities as well as those among the agents comprising the greater architecture [99-101]. Further, complex tuning can be accomplished by introducing coefficients among the force potential terms where each

coefficient can be a static boundary condition, dynamic feedback update, or a scaling function.

The dynamics of individual agents in the MAS can be described as:

$$u_i = (x_i, z_i) \tag{4-7}$$

$$\dot{u}_i = -\sigma \frac{\frac{\partial \hat{F}_i}{\partial u_i}}{\left|\frac{\partial \hat{F}_i}{\partial u_i}\right|} \tag{4-8}$$

where,

$\dot{u}_i$ is the velocity vector of the i[th] agent,

$\hat{F}_i$ is the force vector witnessed by the i[th] agent,

u_i is the position vector of the i[th] agent in the [x, z] coordinate system, and

σ is a velocity scaling parameter.

In this equation, classic potential function theory holds that the term $\frac{\partial \hat{F}_i}{\partial u_i}$ represents the gradient of the force $\hat{F}_i$with respect to the i[th] agent. σ is a velocity parameter that scales the gradient to a desired agent motion. Given a differential-wheeled agent configuration, this parameter effectively defines the angular velocity ratio between the two agent wheels.

The goal of the MAS' control function will be to calculate the agent-unique potential force, $\hat{F}_i$, witnessed by each of the agents comprising the MAS. Force will define the resultant heading and speed that the individual agent would follow. The potential force exigent on the i[th] agent is given by:

$$\hat{F}_i = Q_i \left(\sum_{j=1}^{N_C} \frac{\mu_1 q_j}{\left\| rad_j \right\|^2} \widehat{rad_j} + \sum_{j=1}^{N_A - 1} \frac{\mu_2 Q_j}{\left\| rad_j \right\|^2} \widehat{rad_j} \right) + \sum_{j=1}^{L} \mu_3 P_j \tag{4-9}$$

where,

Q_i is the inherent charge of the i[th] agent,

Q_j is the inherent charge of the j[th] agent,

q_j is the inherent charge of the j[th] cell,

rad_j is the radius vector from the i[th] agent to the j[th] agent or j[th] cell,

N_C is the number of cells in the occupancy map,

N_A is the total number of agents,

P is the calculated force contribution from specified payloads, and

μ_x are the tuning coefficients.

Then, the resultant heading angle θ_i of the i[th] agent is given by:

$$\theta_i = \tan^{-1}\left(\frac{F_{Z\,i}}{F_{X\,i}}\right) \tag{4-10}$$

The calculation of P will vary with payload type as the value and weighting of each payload type can and should have varying degrees of influence on the resultant potential force calculation. As noted earlier, payload types span a wide variety including imaging, laser range finders, infra-red proximity sensors, ultrasonic range finders, or simple contact switches. Even within these modalities, there exist a multitude of options. For example, a simple camera might offer still or video modes, various resolutions, black and white versus color, zoom, and orientation.

Tuning coefficients (μ_x) provide a mechanism whereby the relative contribution of each of the potential forces can be directly tuned either as an independent variable, or dynamically tuned based on a feedback mechanism. This feature is particularly important with multi-agent systems due to scalability. Since the potential forces contributed by elements within the system are cumulative, it becomes very important to gain an awareness of the proportional contributions and subsequently be able to control them proactively. To illustrate, let us assume an occupancy grid of 120 by 240 cells or a total of 28,800 total cells. Each cell might offer a contributing charge value ranging from 0 to 1 meaning a span of 28,800 charge units. If other contributors merely summed to a maximum of 1,000 charge units, the cells' contribution could easily, and likely inappropriately dominate any potential force calculation. These relationships can be observed in Figure 16. Here there are five agents comprising the MAS. The agent associated with this force calculation is located in cell (14,110). The force values represented proximate to this agent are the cell forces. The four other agents are annotated. Their force contributions are the spikes at the agents' respective locations (remembering that agent-to-agent forces are represented as point forces). Therefore, the total force simply sums the area under these curves. Once can clearly see that the area under the cell curve is the significant contributor in this case.

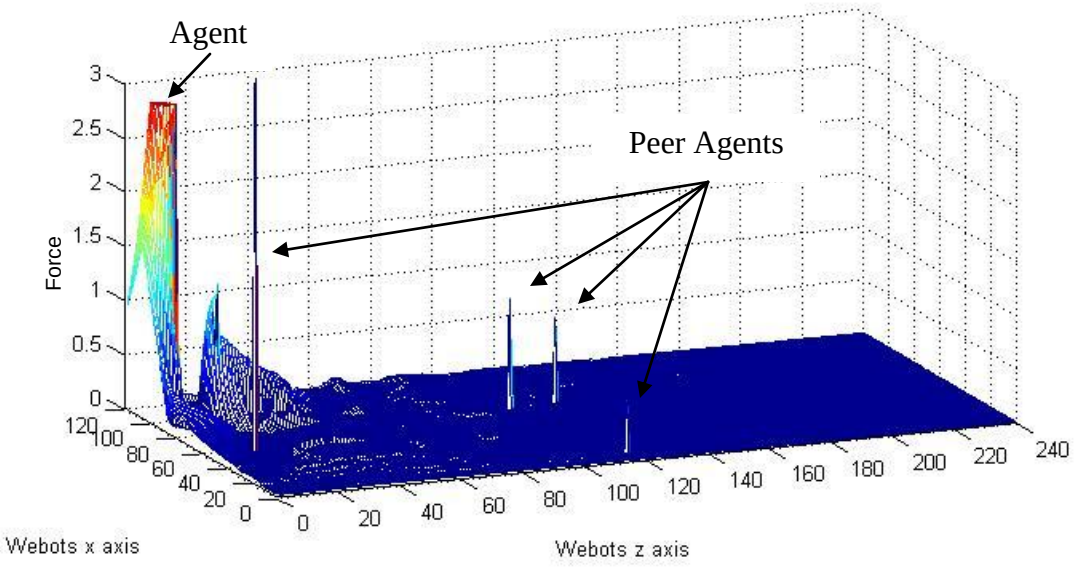


Figure 16: Total Potential Force Mapping

The occupancy map cell characteristic vector represents the probability that the cell is occupied. A value of 1.0 indicates that there is a 100 percent chance that the cell is occupied. A value of 0.0 indicates that there is a 100 percent chance the cell is unoccupied. A value of 0.5 would indicate an equal probability that the cell is either occupied or unoccupied. Separate variables that provide valuable influence in the force calculation are the number of times that a cell has been sampled by a sensor and the number of times that a percentage of the entire occupancy map has been visited. The usefulness here is to accommodate the dynamic nature of agents moving on the arena as well as the potential for moving objects in the field of regard. As a result, the charge associated with each cell q_i is given by:

$$q_i = \begin{cases} P_O q_{max} & \text{for } 0 < V < n \\ 0 & \text{for } V > n \end{cases} \tag{4-11}$$

where,

P_O is the cell occupancy probability,

q_{max} is the maximum possible cell charge,

V is the number of visits to the ith cell, and

n is the number of times that a percentage of the cells have been visited.

4.3 Hybrid Control Systems

The nature of the potential functions for exploration and mapping is to apply inputted contributions and probabilities from the peer agents and the occupancy map at each timestep without regard to imminent collisions, either agent to object or agent to agent. Further, additional influences and desired path planning (e.g., inclusion of image processing calculations) can be included in the potential calculations, but more likely will occur outside of that routine. As such, one way to accommodate these realities is to leverage a hybrid control methodology whereby elements of the governing control occur either in parallel or series and concatenate to generate the resultant guidance commands.

Specifically, for this research, a hybrid control architecture was utilized that sequenced serially through collision avoidance, path planning, and image processing for occupancy map populating. In the first step, the agent collects local information from its proximity sensors. If an object is detected, then a Braitenberg algorithm is utilized to maneuver the agent away from the object. Once clear of the object, the control scheme progresses to the second in its control sequence of potential-function path planning.

Output at this step is the calculated heading to which the agent strives. Finally, the agent executes the third in its control sequence whereby an image is collected and processed to gather and post information important for the update of the occupancy map.

4.4 Stability Analysis via Lyapunov Functions

Given the construct of the MAS's dynamics and utilization of potential function theory to define the path planning and control of the MAS, Lyapunov functions become an exceptionally powerful means to analyze the stability of the multi-agent system to prove that the system is universally stable throughout the comprised motions [102]. The concept of Lyapunov stability states that if a dynamical system initiates near a point of stability and subsequently remains near that stability forever, then it is "Lyapunov stable." In practice, the challenge is to define an accurate and proper Lyapunov function representing the witnessed system and then prove through derivation that the calculated function is positive and decreasing across all elements and time. Lyapunov stability is mathematically defined as:

$$\hat{F}(u) > 0 \; \forall \; u, t \tag{4-12}$$

$$\dot{\hat{F}}(u) < 0 \; \forall \; u, t \tag{4-13}$$

A MAS-specific Lyapunov function $\hat{F}(u)$ can be defined relative to the potential force represented in Equation 4-9 as:

$$\hat{F}(u) = \sum_{i=1}^{N_A} \hat{F}_i\,(u) - \sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \frac{\mu Q_i Q_j}{\|rad_i\|^2} \tag{4-14}$$

$$\hat{F}(u) = \sum_{i=1}^{N_A} Q_M \left(\sum_{i=1}^{N_A} \frac{\mu_1 q_i}{\|rad_i\|^2} \hat{r}_i \right) + \sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \frac{\mu Q_i Q_j}{\|rad_i\|^2} \tag{4-15}$$

The first Lyapunov stability criterion is analyzed by evaluating the quantities comprising the equation above. Each of the individual terms is non-zero thereby insuring that $\hat{F}(u) > 0 \; \forall \; u, t$.

The derivation proving achievement of the second Lyapunov stability criterion starts with the first derivative of Equation (4-15) above. For ease, the agent-to-agent repulsive force term is simplified to FR_{ij}:

$$\dot{\hat{F}}(u) = \sum_{i=1}^{N_A} \frac{\partial \hat{F}_i\,(u)}{\partial u_i} \dot{u}_i + \sum_{i=1}^{N_A} \sum_{i \neq j} \frac{\partial \hat{F}_i\,(u)}{\partial u_j} \dot{u}_j - \sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \left(\frac{\partial FR_{ij}}{\partial u_i} \dot{u}_i + \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j \right) \tag{4-16}$$

Focusing on the second term, note that the only contribution to $\partial \hat{F}_i\,(u)$ is through the agent-to-agent repulsive force. Therefore, that quantity can be replaced with $\partial \mathrm{FR}_{ij}$ resulting in:

$$\dot{\hat{F}}(u) = \sum_{i=1}^{N_A} \frac{\partial \hat{F}_i\,(u)}{\partial u_i} \dot{u}_i + \sum_{i=1}^{N_A} \sum_{i \neq j} \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j - \sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \left(\frac{\partial FR_{ij}}{\partial u_i} \dot{u}_i + \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j \right) \tag{4-17}$$

The next three steps manipulate the bracketed term with a goal of equating it to the last term:

$$\dot{\hat{F}}(u) = \sum_{i=1}^{N_A} \frac{\partial \hat{F}_i(u)}{\partial u_i} \dot{u}_i + \left[\sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j + \sum_{i=1}^{N_A} \sum_{j=1}^{i-1} \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j \right] - \sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \left(\frac{\partial FR_{ij}}{\partial u_i} \dot{u}_i + \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j \right) \tag{4-18}$$

$$\dot{\hat{F}}(u) = \sum_{i=1}^{N_A} \frac{\partial \hat{F}_i(u)}{\partial u_i} \dot{u}_i + \left[\sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j + \sum_{k=1}^{N_A} \sum_{l=k+1}^{N_A} \frac{\partial FR_{kl}}{\partial u_k} \dot{u}_k \right] - \sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \left(\frac{\partial FR_{ij}}{\partial u_i} \dot{u}_i + \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j \right) \tag{4-19}$$

$$\dot{\hat{F}}(u) = \sum_{i=1}^{N_A} \frac{\partial \hat{F}_i(u)}{\partial u_i} \dot{u}_i + \left[\sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \left(\frac{\partial FR_{ij}}{\partial u_i} \dot{u}_i + \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j \right) \right] - \sum_{i=1}^{N_A} \sum_{j=i+1}^{N_A} \left(\frac{\partial FR_{ij}}{\partial u_i} \dot{u}_i + \frac{\partial FR_{ij}}{\partial u_j} \dot{u}_j \right) \tag{4-20}$$

Here, the bracketed and last terms are equal and therefore drop away leaving only the first term:

$$\dot{\hat{F}}(u) = \sum_{i=1}^{N_A} \frac{\partial \hat{F}_i(u)}{\partial u_i} \dot{u}_i \tag{4-21}$$

Now, substituting for $\dot{u}_i$:

$$\dot{\hat{F}}(u) = \sum_{i=1}^{N_A} \frac{\partial \hat{F}_i(u)}{\partial u_i} \left[-\sigma \frac{\frac{\partial \hat{F}_i}{\partial u_i}}{\left| \frac{\partial \hat{F}_i}{\partial u_i} \right|} \right] \tag{4-22}$$

$$\dot{\hat{F}}(u) = -\sigma \sum_{i=1}^{N_A} \left| \frac{\partial \hat{F}_i(u)}{\partial u_i} \right| \tag{4-23}$$

And therefore, the second Lyapunov criterion, $\dot{\hat{F}}(u) < 0 \; \forall \; u, t$, is also satisfied.

Chapter 5. Test Excursions in Simulation

5.1 Simulation Environment

The robot agents utilized are the e-puck robot (Figure 17) originally developed by the Ecole Polytechnique Federale de Lausanne (EPFL) in Lausanne, Switzerland. The e-puck is principally an educational robot that offers significant research capability both through its own autonomous processing and sensing capability, as well as its close linkage with Cyberbotics' Webots robot simulation package [103-104]. An e-puck is approximately 7 cm in diameter. It features a Microchip dsPIC processor running at 60MHz or about 15 MIPS. Mobility is provided by two stepper motors in a differential-wheel drive configuration. The robot has 8 infrared sensors operating in the 950 nm wavelength that are positioned around the robot's perimeter to provide both proximity sensing and light sensing. There is a 640 by 480 pixel camera on-board that can provide video and imagery in either color or black and white modes. Bandwidth limitations throttle collection nominally to 52 by 39, 40 by 40, or 640 by 1 pixel images at up to three frames per second. There are three omni-directional microphones for sound localization, a three-axis accelerometer, and on-board speaker. Communication is provided by Bluetooth. Programming for cross compilation and download to the e-puck is exclusive to the C programming language.

Figure 17: e-puck Robot
(Image by École Polytechnique Fédérale de Lausanne,
www.e-puck.org, August 17, 2009)

Cyberbotics' Webots simulation package (Figure 18) provides an extremely robust capability and interface with very high resolution characterization of the robots in a physics-enabled, three-dimension, virtual world. Webots can control the robot in two modes:

- Remote control. In this mode, the control program resides on the Webots' host computer and drives the robot behaviors via low-level commands sent to the e-puck via Bluetooth in real time. This mode affords use of at least four programming-language options: C, C++, Java, and MATLAB.

- Cross compilation. In this mode, a C program can be developed within Webots and fully tested in the simulation environment. That same code can then be cross compiled and downloaded to the e-puck for autonomous operation. Realistically though, there are severe limitation to this mode due to

the minimal processing and memory capability on-board the e-puck, and the inability to use Bluetooth for agent-to-agent communications.

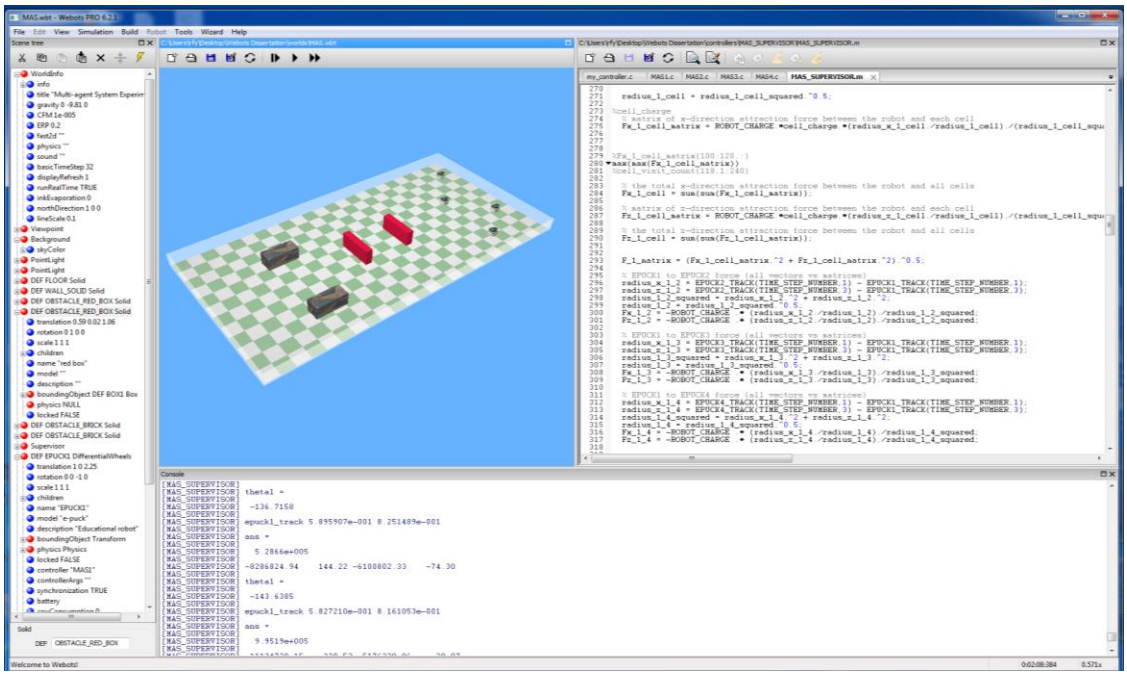


Figure 18: Webots Robot Simulation Screenshot

The representative simulation and experimental agent arena for this research consists of a 120 cm by 240 cm arena bordered by a low wall. The arena is sub-divided into a representative occupancy mapping grid consisting of 1 cm by 1 cm cells each having an associated vector representing dynamic characteristics such as the probability of occupancy or potential charge. The green and white checkerboard appearing in the simulation's arena is physically 10 cm x 10 cm and exists merely to allow users to identify rough approximation of localization. The arena also hosts obstacles of varying shapes and sizes as well as identified agent starting points. In simulation, the localization

of each agent is provided by a surrogate global positioning system function afforded through the Webots simulation software and in stark contrast to the real e-puck's lack of such capability. In keeping with the representation and standard established in the Webots simulation package, the world coordinate system throughout both the simulation and hardware experiment was established as illustrated in Figure 19.

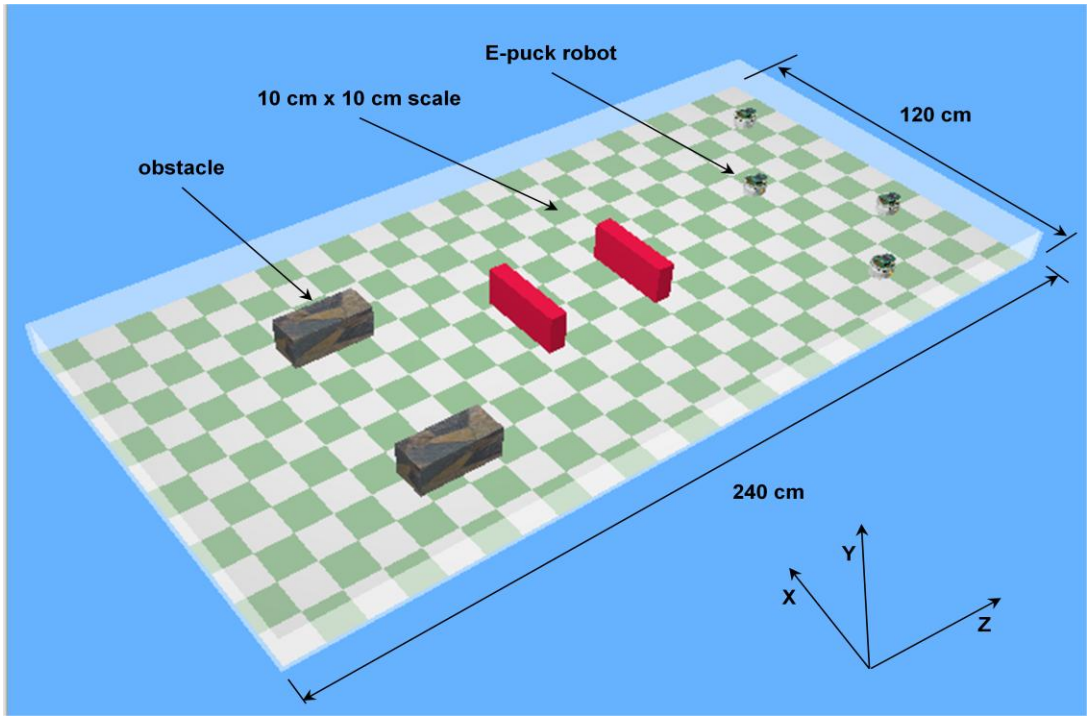


Figure 19: e-puck Arena in Webots

The e-puck robot offers three payload options for information collection. First are eight proximity sensors vectored parallel to the ground and outwardly around the perimeter of the agent. These sensors have a limited range effectiveness, typically less than 8 cm from the agent's center of mass. The proximity sensors are used to detect

objects near the agent by thresholding the sensor return. This means that if a sensor returns a reading above a pre-determined threshold, then the object is calculated to be located at the intersection of the sensor ray and the distance associated with the threshold. The simulation sets the distance/sensor return relationship as a linear function. Therefore, knowing the location of the agent, geometry of the sensor orientation and distance to the object, the object's location can be calculated and subsequently used to update the respective occupancy map probabilities as shown in Figure 9 (Chapter 2.5).

The second sensor available on the e-puck is a 640 by 480 pixel camera. The camera can be utilized in either a streaming or imaging mode in full color (256 bit range in three color channels) or grayscale. Unfortunately though, the real e-puck's Bluetooth communication capacity imposes a severe and real limitation on the camera mode selection such that the maximum collection mode available is on the order of 9 kilobits per second, roughly equivalent to a color 52x39 image delivered at 2 Hertz [105].

Control in the Webots environment has been successfully implemented in a variety of research projects [106]. For this research, control is implemented in two levels - high and low (Figure 20). High-level control includes the logic for path planning, peer dispersion, and state space representation. The high-level control algorithm and image processing is accomplished using MATLAB coding. The distinct advantage is its availability of a comprehensive function set as well as inherent ability to manipulate matrix-organized data. A low-level controller is utilized to provide basic control functionality to the robot including sensor data extraction, actuation (wheel torque commands), and local collision avoidance. It is written in the C programming language.

Webots offers a surrogate emitter/receiver function whereby agents can communicate directly with each other. This function was utilized to effectively pass information from the e-puck itself and its low-level controller. Like the GPS functionality, this emitter/receiver function exists in stark contrast to the real e-pucks lacking inter-agent communications capability.

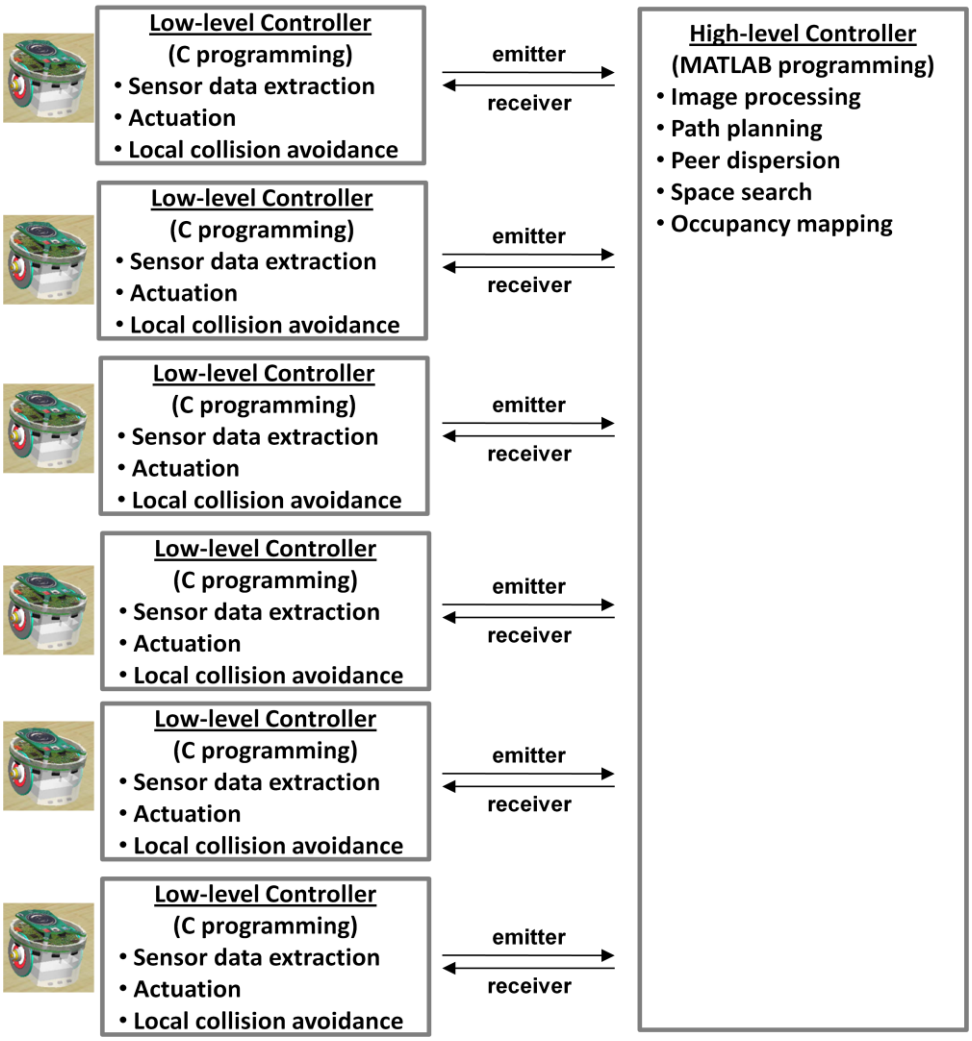


Figure 20: Simulation Software Architecture and Functionality

In order to visualize and track the progress of a simulation run and its representative evaluation metrics, a MATLAB figure is generated every 100 time steps that offers information on the e-pucks' track history, cell charge, occupancy mapping and cell visit count (Figure 21). A comprehensive set of state and evaluation metrics are collected at every control time step that provide an opportunity to observe the progression of the exploration task. Agent trajectories provide insight to the energy consumed by each individual agent in terms of the distance traveled. Trajectories also illustrate the qualitative uniformity and dispersion with which the team of agents uniquely maneuver in and canvas the arena. The occupancy map data vector at a specific time step is displayed to show the occupancy probability, cell charge, and cell visit count. Observed individually and sequentially over time, these vectors provide insight to the progression of understanding of the heretofore unknown arena and resolution with which obstacles and targets appear. One can also conclude trends such as the extent of exploration and behaviors proximate to obstacles. Finally, mean characteristic values graphed over time quantify the performance of the multi-agent system and illustrate its ability to progress toward threshold goals.

The average occupancy map value shows the time progression of the average across each cell's probability of occupancy. Since there is no a priori knowledge afforded, the initial value is 0.5 meaning all cells have equal probability of being occupied or unoccupied. Over time, as the agents explore the arena and determine free and occupied space, the curve progresses as expected decreasing over time with the free space determined and step increasing periodically with the sensor-enabled detection of

obstacles, i.e., occupied space. The curve is non-linear due to the proximity of unexplored space diminishing over time resulting in the necessity to travel longer distances (and consuming more time) to sample those unexplored spaces.

The average cell charge plots the average of all the cells' calculated charge values over time. This curve initially decreases due to the fact that as cells are explored, their charge decreases so as to have less impact in subsequent path planning loops. However, in deference to the dynamic state of the arena, the average cell charge begins to increase in correlation with the map coverage count due to the forcing relationship in Equation (4-11). The average cell visit count tracks the average number of times that any agent samples individual cells over time. This curve has a constant slope as expected knowing that the agents' speed varies very little perhaps only to slow for short period of times while maneuvering around obstacles. Finally, the map coverage count, *n* in Equation (4-11), is an evolving parameter that tracks the number of times that a pre-defined percentage of the arena has been sampled at least the count's number of times. For example, the count starts at zero. As soon as 75 percent of the arena has been sampled once, the count increments to 1. As soon as 75 percent of the arena has been sampled twice, the count increments to 2. This parameter is used within the control loop to influence the charge calculation in order to accommodate the dynamic nature of the state representation. The shape of this curve provides qualitative clues as to the efficiency of the search.

The transition from the simulation environment to the experiment environment is greatly facilitated by the capabilities of the e-puck/Webots combination. In fact, the

Webots' environment was designed to allow development of controlling code initially using the simulation as the target output for low-level commands. With success there, a software "switch" can alternately direct the actuation command output to the e-puck hardware itself via Bluetooth.

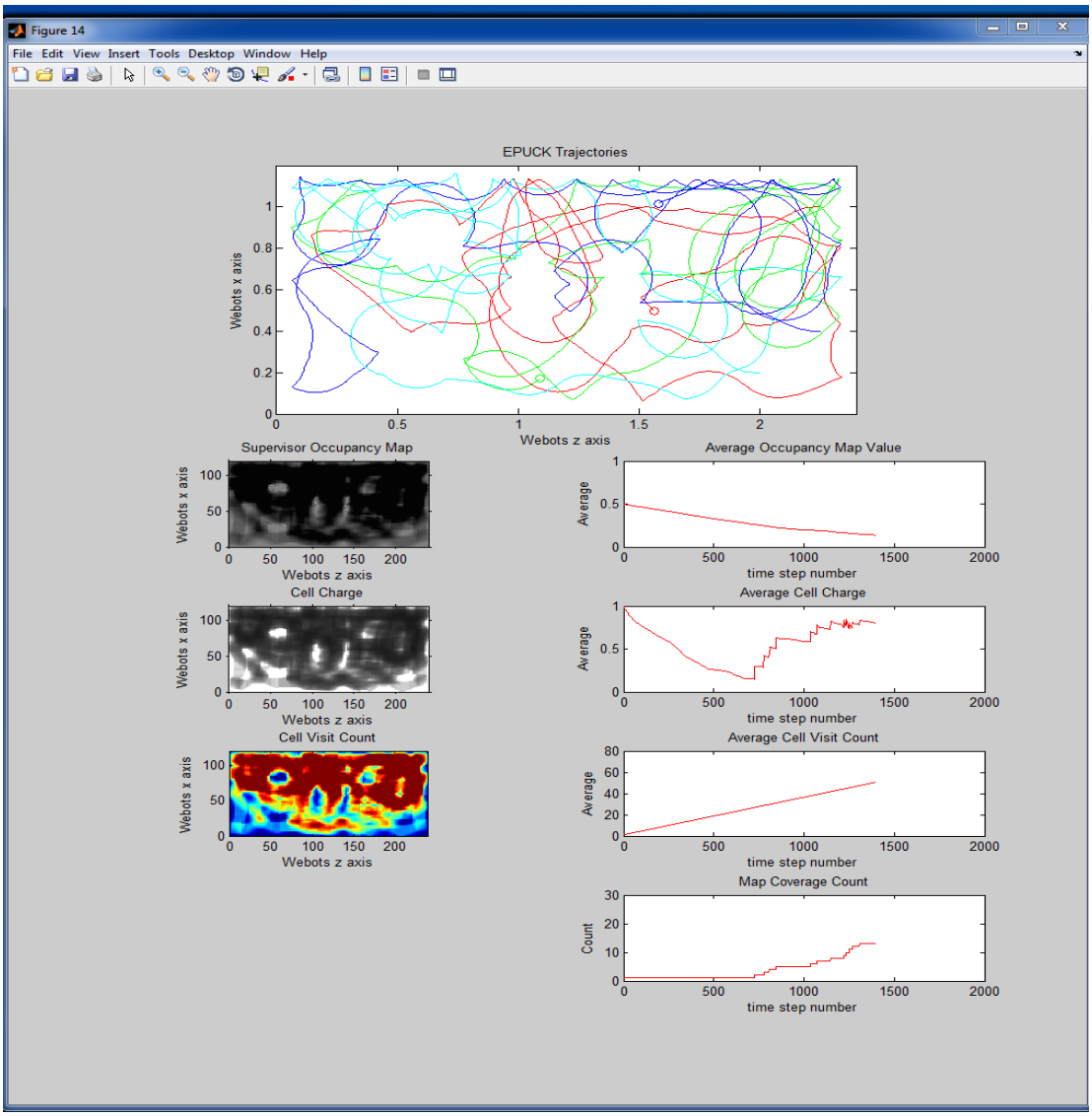


Figure 21: Simulation Metrics Output Screen

5.2 Assumptions

Two assumptions were made in this research without inadvertent or unnecessary distraction to the principal goal of exploring the impact of payload selection on mission accomplishment. The purpose principally is to insure isolation of the core influences on system performance and to minimize undesirable dependencies.

1. *The state-space and occupancy mapping is limited to two dimensions*. The Webots-hosted simulation and experiment itself will be implemented in full three dimensions. In the simulation, that means that the arena, e-puck robots, obstacles, and targets will have full dimensionality and functionality within. Sensing too will occur in the full three dimensions. The proximity sensors and camera will all operate throughout the space. The limiting assumption is restricted to the state representation. The arena itself is completely flat and aligned to the x and z axes (as per the Webots' global reference frame) parallel to the floor. The obstacles and targets will be volumes, but with vertical sides orthogonal to the arena. The motion of the e-puck robots will be limited to the x-z axes and not move vertically in the y axis. With these restrictions, the occupancy map need only to exist as a two-dimensional matrix consisting of 120 by 240 cells each 1 cm^2 in the x-z plane. The significant savings here is that the map can be represented by 28,800 elements versus, for example, more than 4.3 million if the map were to include 150 cells in the y (vertical) direction. Given the already-intensive MATLAB-based series of matrix calculations occurring in the path planning and collision avoidance schemes, this simplification enables a more real-time processing frame.

2. *Targets are characterized with surface drawn simple graphic symbols without requiring complex image processing and recognition*. The goal of this research is not to evaluate the performance of image recognition or image processing algorithms. Rather, it is to evaluate the impact of planning payload capabilities on the total mission performance. In this case, the image processing is a fixed quantity by design to insure that the mission performance metric is isolated to the independent variable of payload planning.

5.3 Discussion of Simulation Results

With this research, the desired and expected result was to define and isolate individual factors and associated design and planning mechanisms that have the largest positive impact on the overarching MAS to accomplish its mission. In terms of cost, this includes expecting that the simplest of sensor modalities dispersed among the agents that are used in orchestration can contribute significantly to MAS behavior definition and mission accomplishment.

The accomplished research consisted of development and implementation of the potential function control logic in simulation. This included both the high-level controller written in MATLAB M-code programming language that was used to host the overarching MAS potential function, occupancy mapping and sharing, and e-puck communications; and the low-level controller written in C programming language that hosted basic robot functionality of articulation, motion, sensing, localization, and communication with the high-level controller. Similar architectures can be found in

literature [107-108]. The payload capabilities were restricted to the proximity sensors. The simulated arena consisted of a bounded 120 cm by 240 cm flat stage with an unknown number of objects of unknown size, shape, and color. The scenario used was to have the MAS search the bounded arena and populate the occupancy map with information collected by the proximity sensing. The scenario was run for a constant 2,000 time steps per simulation experiment. The independent variable considered was the number of agents (e-pucks), from one to four, comprising the multi-agent system. The dependent information collected, reduced, and evaluated (Appendix A) were the resultant 1) occupancy maps; 2) cell charge mapping; 3) cell visit count; and map coverage count (Figure 22). The data itself was represented in three-dimensional graphs with dimensional scaling on the x and z axes (the horizontal plane in the graph), and values vertically. The range on the value axis was the same for all four charts to facilitate qualitative analysis, i.e., the charts are all proportionally similar.

There were several very interesting observations and characterizations derived from the reduced and analyzed data.

1. The map coverage count increases non-linearly with additional robots in the multi-agent system. Figure 22 shows the map coverage count witnessed from the simulation runs of the various sized MASs. This was an expected result given the time/space efficiencies offered by MAS in consideration of inter-robot repulsive forces included in the path planning algorithm. This repulsive force tended to spread the agents such that the distance traveled to get from current location to desired location to low occupancy probability locations tended to be lower. The trade-off however is the logic

processing power required, which similarly is increasing exponentially as a function of the number of agents in the MAS.

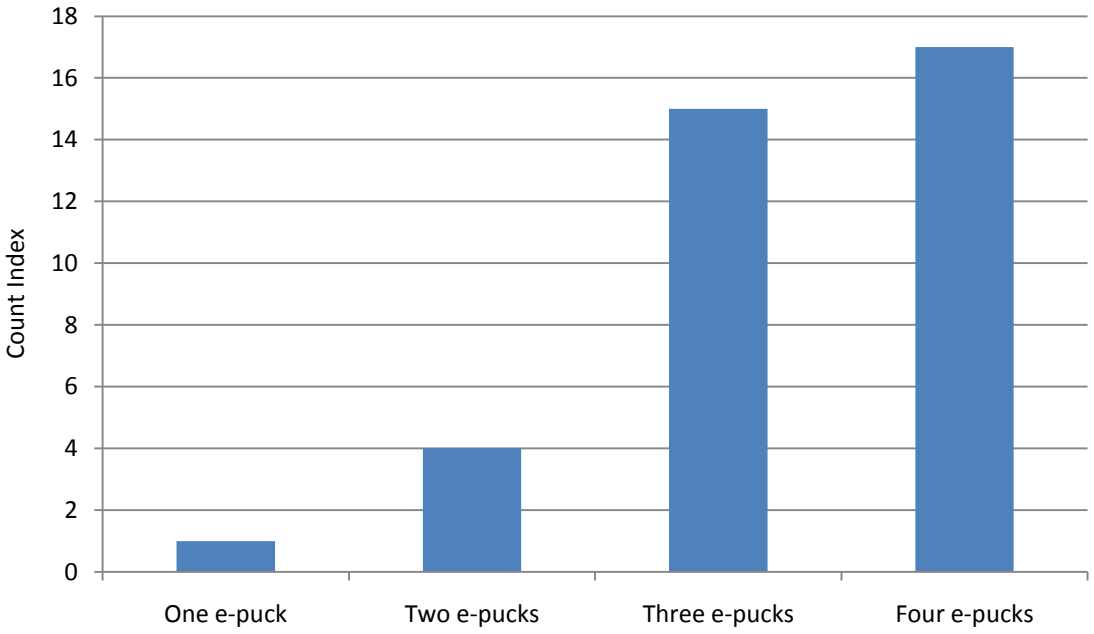


Figure 22: Map Coverage Count

2. *The map coverage count curve of a fixed size MAS approaches an asymptote.*

Appendix A displays the map coverage count vs. time curves for the four MAS variants. This was an unexpected result as one would expect the map coverage count to continue increasing as the agents continued to search the space. One observation however was that this is a local asymptote whereby continuing the search would show another exponential growth of the map coverage count to a second asymptote. Alternately, this could very well be an indication that the space has been exhaustively searched such that a confidence could be declared.

3. The potential energy function offers efficient and uniform coverage capability in a search mode. This observation was drawn from the cell visit count graphs as well as the e-puck trajectories in Appendix A. The data represented proves to be uniform over the arena space and comprehensively covers the arena itself. This was quite expected given the influence that occupancy mapping has and the related magnification of unexplored space therein. The one exception noted is discussed in the next item.

4. Objects, more specifically the charge mapping of occupancy cells comprising the object space, skew the path planning. This was a shortcoming noted in published research [68] and reconfirmed herein. This was a result of the fact that the agent can never sense inside the perimeter boundary of the objects such that the cell charges within are decremented. The solution implemented during the experimentation phase was quite simple whereby a routine was added that identified the objects as a function of the occupancy mapping, and then artificially set the cell charges accordingly to remove their influence.

5. The relative proportions of the elements comprising the force calculation in the path planning routine can vary significantly over time. This too was a subject ripe for further investigation in continuing research. As an example, the force associated with a mature occupancy mapping (i.e., a well known environment) was possibly only 1-5 percent of that associated with an unknown environment. In addition, the collective repulsive force from agent to agent was a function of distance between the agents, but was consistent over time since the average distance remained somewhat constant. The result of these two facts was that the influence of agent repulsion over time effectively

increases. In this limited case, it actually becomes a desirable feature as one would endeavor to spend less energy searching known space. However, experimentation closely considered this phenomenon especially as it applied to the addition of imaging and video payloads and the dynamic nature of their information contribution.

6. The underlying computational requirements of Webots plus an instantiation of MATLAB for each robot within the MAS was not scalable. The computer used to host the simulation offered significant capability in terms of processing power and available memory. Specifically, it consisted of two quad-core processors and 48 GB of RAM. However, the four-robot configuration consumed nearly 100 percent of the CPU availability and additionally throttled the simulation time factor to further accommodate the processing requirements. This was not unexpected as correspondence with Webots User Groups and the corporate helpdesk have reported similar burdens. This observation led to a significant decision for the experimental phase whereby a separate computer was dedicated one per e-puck robot. This was not unrealistic as one would expect that a robot would offer its own computing capability to conduct activity.

7. The force values dropped precipitously as a function of distance. The physics of potential energy functions dictate that the force is inversely proportionate to the square of the distance. In this research, the energy physics were replicated as such. However, Figure 23 shows that this tends to greatly magnify the influence of proximate cells and greatly diminish distance cells. The conclusion drawn here was that in an effort to normalize cell charges, and gain better responsiveness over the entirety of the arena versus proximate cells, experimentation explored varying the distance's exponent.

Clearly, this deviates from classic potential theory and defies the laws of physics, but might nonetheless offered some interesting advantages in path planning efficiencies.

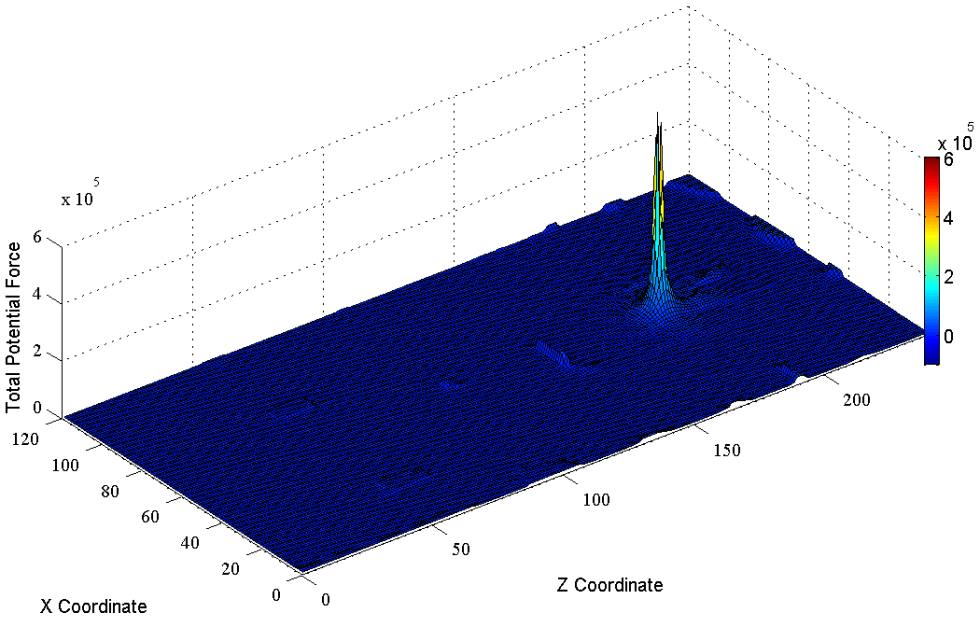


Figure 23: Total Potential Force

8. Transition from global path planning control to local collision avoidance control exhibited undesirable oscillations. Global path planning was generated using the potential function while local collision avoidance, triggered by obstacle detection, used classic Braitenberg theory. Simulation runs illustrated that as an individual robot transitions between these two control states, an undesirable oscillation appears at the boundary (refer to the upper edge of the arena in Figure 24). This phenomenon resulted from two realities. First, the cell charges tended to be higher at the perimeter boundaries due in fact to this oscillation. This was confirmed by referring to the cell charge mapping in Appendix A. Therefore, the robots strived toward that boundary. Secondly, the

existing Braitenberg algorithm was tuned to simply motion reflect off of the boundary as opposed to perimeter follow.

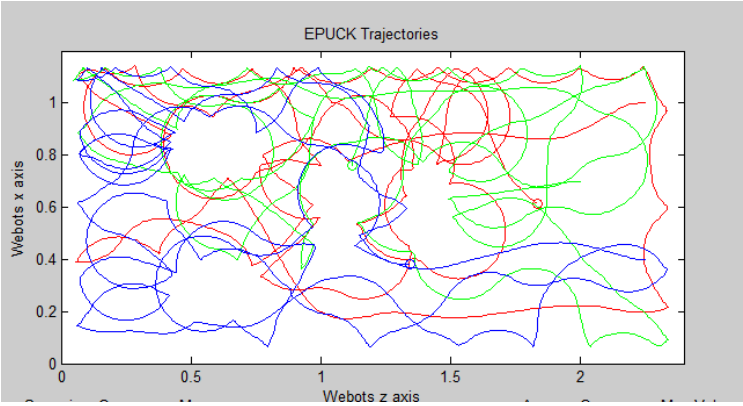


Figure 24: e-puck Trajectories

Chapter 6. Experimental Validation

6.1 Hardware and Software Architectures

The infrastructure supporting experimentation provided a critical capability in support of the research. Both corporate research institutions and a wide variety of academia describe particularly novel and capable configurations implemented [109-113].

Figure 25 summarizes the hardware architecture utilized for this experiment. Fundamentally, each agent (e-puck) was controlled remotely via Bluetooth by a dedicated Dell Optiplex 780 desktop computer. The state space representation was managed by a separate desktop computer (Dell Precision T7500). Communications among these desktops was enabled via a gigabit Ethernet backbone. These are described in more detail following.

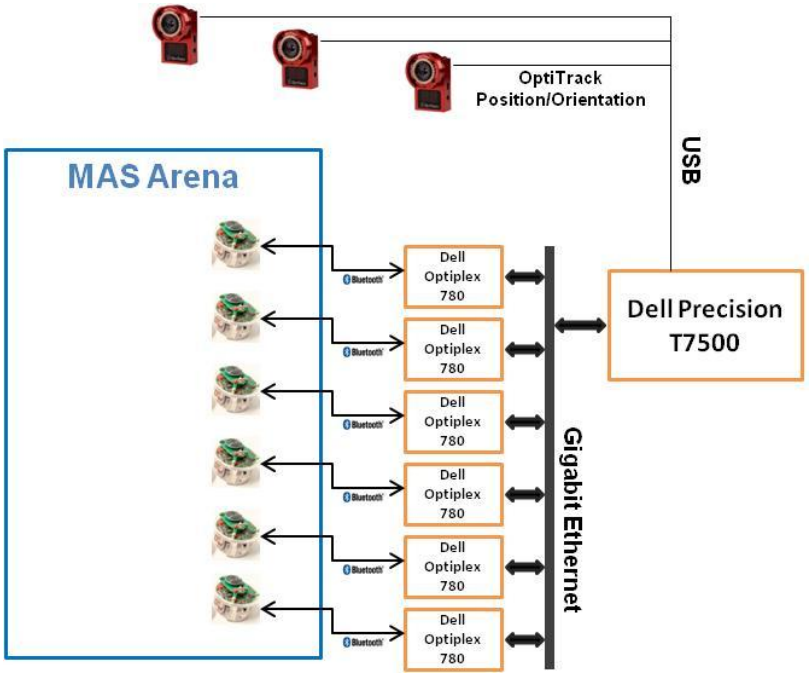


Figure 25: Experiment Hardware Configuration

The e-puck robot, while quite capable and user friendly in its own right, has some significant materiel limitations in terms of on-board processing and storage. Specifically, the e-puck utilizes a dsPIC 30F6014 processor running at 60 Mhz designed to support C programming. It has a 16-bit microcontroller, 8 KB of RAM and 144 KB of FLASH memory. With this configuration, the e-puck is quite limited in its total processing capability. In particular, large-dimension matrix operations, such as those necessary for image processing and manipulation are not readily enabled. Fortunately, two operating modes are supported – full autonomy (i.e., all processing and control signals are generated on board) and remote control. In remote control mode, the robot is controlled remotely via Bluetooth communications from an external computer running the controller script or software code.

High level control can be implemented in any of a number of programming languages (C, C++, MATLAB, Java). Interface is provided to the e-puck hardware via Bluetooth to pass data collected (camera images or IR proximity sensor values) and actuation commands (drive wheel values or LED illumination). Capability here proved wanting as bandwidth limitations significantly throttled data transmission. For example, a 40-by-40 pixel color image (approximately 5 KB) required nearly 0.5 seconds for transmission. Communication ranges experienced were also limited being 2.5 meters versus the Bluetooth-specified range of over 10 meters. Low level commands are executed onboard the robot. For this experiment, the computer was a Dell OptiPlex 780 with a Intel® Core™ 2 Quad Q9400 with VT (2.66GHz, 6M, 1333MHz FSB) and 4 GB of RAM. The obvious benefit was the ability to process the high level control algorithms

and image processing via MATLAB in a real time manner. Since the focus of this research was dependent upon a comprehensive ability to process payload information, it merely became pragmatic and necessary to operate exclusively in the remote control mode.

6.1.1 State Manager and Agent Controller Software

The MATLAB program and associated scripting language were utilized to code the software required for state space management, databasing, and distribution; and for the individual agent controller software. Sample code is included in Appendices B and C, respectively. The specific benefit afforded by MATLAB is its ability to quickly manipulate matrices. For both mapping and image processing, this proved to be tremendously valuable in terms of significantly reducing the control-loop cycle time. EPFL provides a MATLAB function set named ePic that provides low-level interface to an e-puck's basic actuation [67].

6.1.2 Surrogate GPS

In contrast to single mobile robots with their broad, comprehensive capabilities (vision, LIDAR, differential global positioning, ultrasonic proximity sensing, etc), the agents comprising a MAS tend to have somewhat limited organic capability due to a desired affordability or physical size. An example of one such capability is localization that could provide accurate agent position and orientation in a global reference akin to the

Global Positioning System (GPS). While GPS receivers have become small and inexpensive, they typically cannot be used indoors where many MAS experiments are conducted.

As such, previous research effort has been focused on the simultaneous localization and mapping (SLAM) problem [108, 114-115]. SLAM acknowledges that externally-provided localization information is not typically available, and also acknowledges a scenario of exploration of heretofore unknown space. Therefore, clever schemes have been explored that use available modalities (e.g., inter-agent communications) to determine relative localization among the agents. That, combined with robot-sensor-enabled, continually-updated mapping databases, generates a comprehensive representation of the spatial and spectral environment. Many times though, inherent SLAM functionality is not germane to the research at hand. In this case, an extra-MAS localization capability is most desirable to minimize the experiment's independent variables and uninteresting complexities that might have a detrimental impact on the core research.

There are several materiel possibilities for such an extra-agent capability. Fricke et al utilized a combination of a Cognex camera and Sick LIDARs to calculate localization [116]. The Vicon motion capture system is a popular solution that uses connected high-speed digital cameras viewing the MAS experiment workspace with redundant coverage to provide localization of tracked objects [117].

The NaturalPoint OptiTrack system utilizing its Tracking Tools (TT) software application provides a similar capability [118]. The OptiTrack cameras are installed and

calibrated for the desired experiment workspace (Figure 26). Each robot is given a unique spatial configuration of reflective markers (Figure 27) that enable individual robot identification during the tracking process (Figure 28).

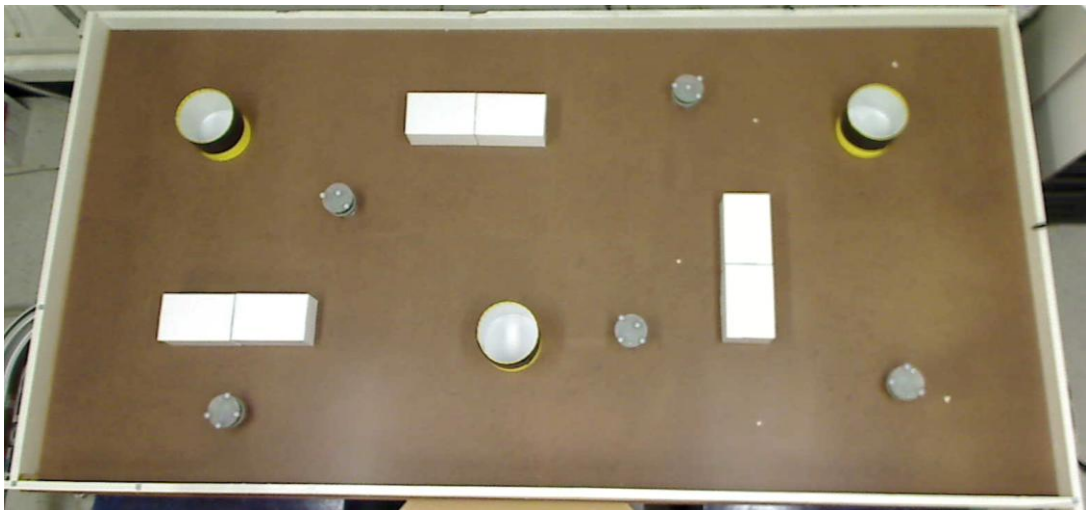


Figure 26: MAS Experiment Workspace

Figure 27: Optical Tracking Reflectors on e-puck Robots

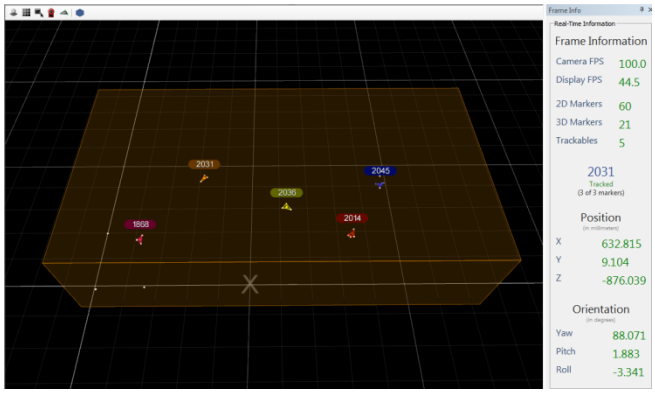


Figure 28: Screen shot of NaturalPoint Tracking Tools

A critical functionality important to supported experimentation that TT offers is the ability to multicast tracked-object localization real time through a designated network port on the computer hosting the TT application. If connected to a local-area network, any other computer (e.g., one hosting an agent controller) could then too have near-instantaneous access to the real time streaming position information of itself, and of any other agent in the workspace with update rates approaching 100 Hertz. This is particularly desirable as this streaming broadcast scheme significantly reduces the inter-agent communications requirement. TT offers three formats for broadcast: industry-standard Virtual Reality Peripheral Network (VRPN) and Trackd®, and NaturalPoint's own NatNet format. VRPN, provided by the University of North Carolina's Department of Computer Science, offers several advantages including a comprehensive C/C++ dynamic-linked library (DLL) and public domain license [119-120]. For illustration, NaturalPoint provides sample C++ code titled "VRPN-Listener" [121] that provides a

simple capability to sample to the TT port and output the received localization information to the screen.

Control can be coded in a wide variety of software languages, each offering unique benefits and shortcomings. MATLAB offers significant advantages in terms of comprehensive image processing routines and robust matrix manipulation routines to support important functionality such as volume mapping.

In implementation with the VRPN construct though, a significant challenge arose in that there were no MATLAB native commands or functions available to interpret the VRPN broadcast as the DLL is C/C++ based. This research describes one solution and its experimental implementation realized by creating a MATLAB executable (MEX) function [122] that allowed the agents' MATLAB-based control algorithms to access and call the C++ DLL native to VRPN. This enabled a comprehensive surrogate localization function.

The capability of the created MEX-function was twofold. First, given identification of the specific agent of interest, the MEX-function goes to the VRPN broadcast port and obtains the most current localization information for that agent. Secondly, the MEX-function converts the provided localization information into a desired coordinate system and units. For the cited experiment, this consisted of converting the VRPN-provided orientation in the three-dimensional quaternion rotation space in radians to roll-pitch-yaw parameters in degrees.

MEX-functions are an organic capability offered by MATLAB that allow a programmer to compile software routines written in C/C++ and linked to C/C++ DLLs

resulting in a function callable from within MATLAB programs just as one would call a MATLAB built-in function. The specific C++ code used is a task-specific modification of the NaturalPoint-provided "VRPN-Listener" code written in C++. The code is included in Appendix D.

The implementation proved quite successful. In a control scheme where cycle times on the order of 200 milliseconds is desirable, the surrogate localization function created only minimal processing time burden consuming only low-single-digit milliseconds. Moreover, this function proved a robust capability to provide the variety of network-connected controllers immediate access to its own controlled agent, but also to the peer agents operating on the arena, all in near real time.

6.1.3 UDP Communications for Data Sharing

Multi-agent systems can be characterized organically as having either distributed or centralized control with the distinction being the level to which each individual agent in the system governs its own motion. Fully distributed control is characterized by a complete absence of data or control-command exchange among the agents whereby each agent relies upon its own sensors to understand the environment, and its own processing capability to determine controlled actions. On the other end of the spectrum, centralized control is characterized by a single processor calculating the collective motion of the system and disseminating commands to each agent for the orchestration movement. As a result, distributed control architectures require robust on-agent processing capabilities,

but minimal communications. In contrast, centralized control architectures are accomplished with a single robust processor (the central controller), relatively minimal agent processing; however, a robust inter-system communications capability to facilitate distribution of necessary control commands, sensor information, and other data such as state and localization information. Control can also exist on a spectrum between the extremes of centralized and decentralized. One can imagine that there are varying levels of processing and communications required among the various control schemes ranging from fully distributed to fully autonomous [111,123].

For the research conducted under this effort, the agents comprising the system were constructed and defined as being fully distributed with each agent and its associated controlling computer fully responsible for calculating motion commands. However, there was a slight variation in that a separate "state" computer existed to act as a surrogate global positioning system (via the OptiTrack Tracking Tools system and software); and to collect, maintain, update, and distribute state-space information (position and orientation of each agent as well as representation of the operating space). Because of this, a comprehensive inter-computer communications infrastructure and protocol was required and created to support to passing of data among the state computer and controller computers.

From a hardware perspective, the seven computers (state computer plus the six controller computers) achieved full gigabit Ethernet inter-computer communications throughput capability by connection to a single DLink DGS-2208 eight-port gigabit Ethernet switch. That switch in turn, is connected to a Netgear Prosafe FVS336G gigabit

Ethernet router set up to statically assign internet-protocol addresses to each of the experiment's computers as follows:

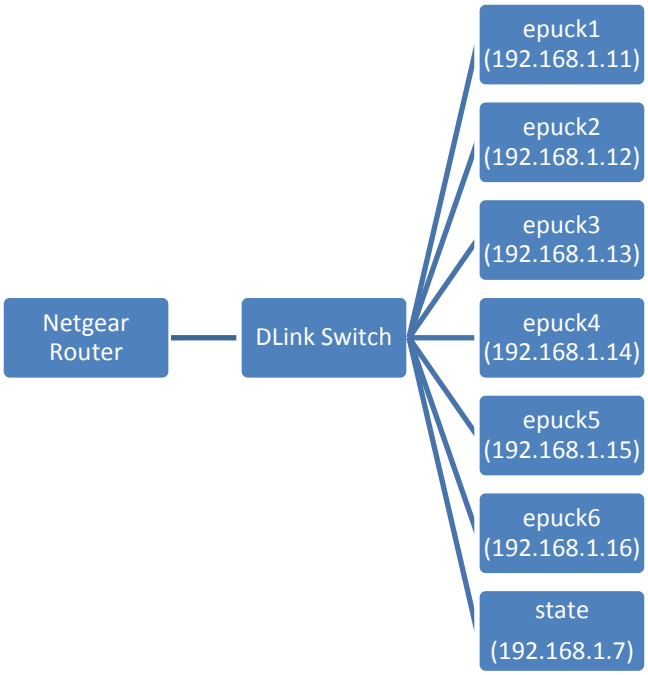


Figure 29: Network Communications Architecture

With this architecture, each computer was uniquely identified by its statically-assigned IP address facilitating unique computer programming.

The second aspect of providing a robust information exchange infrastructure was to select the underlying data protocol used. There were several protocols available that could be used in MATLAB and C programming including notably transmission control protocol/internet protocol (TCP/IP) and user datagram protocol (UDP). In fact, MATLAB's Instrument Control Toolbox provides a very comprehensive set of TCP/IP and UDP command sets. Each offers advantages and disadvantages. For example,

TCP/IP offers a strong recovery feature that consists of error-checking and resending data packets if they are not accurately received. UDP offers greater throughput potential due to its use of minimal data fields for error-checking overhead. Ultimately, UDP was utilized based on the construct of the experiment. In fact, MATLAB's TCP/IP protocol cannot be used to communicate between disparate MATLAB instantiations, i.e., communications between MATLAB running on computer "X" and a separate instantiation of MATLAB running on computer "Y." The concern here however is that while it offers the greatest throughput, UDP does not offer any organic error checking. This turned out not to be an issue as error checking in the authored code itself was able to accommodate any transmission errors. Further, monitoring of the UDP errors throughout the conduct of experimentation witnessed only very minimal problems.

In order to get MATLAB sessions communicating with each other, each required creation and definition of a socket infrastructure and associated data infrastructure. The socket infrastructure defined specific electronic 'locations' in each respective computer where the controller code directed data for transmission to and from specific other computers. For example, the MATLAB controller code for agent "epuck1" would initialize the construction of a socket via the command set:

```
u = udp('192.168.1.7', 7011, 'LocalPort', 7011);
u.timeout=100;
u.inputbuffersize =5000000;
u.outputbuffersize = 5000000;
u.inputdatagrampacketsize = 60000;
u.outputdatagrampacketsize = 60000;
u.ReadAsyncMode = 'continuous';
fopen(u);
```

The first line of codes established "u" as the variable representing the communications path throughout the software code. "u" then was defined as the IP address and port of the distant computer with which to establish connectivity (i.e., 192.168.1.7:7011) and the identified port on the agent's computer (i.e., LocalPort:7011). Note that while the same port number was used on each of the two computers, they each refer to separate ports.

The next lines of code defined various parameters associated with the socket. It will timeout in 100 seconds meaning that calls for data will expire in exactly 100 seconds. The input buffer and output buffers were both 5 megabytes meaning that was the maximum the buffer can hold at any instance in time. Further, the input and output datagram packet sizes were 60,000 meaning the maximum number of packets that can be buffered at any one time. Finally, the async mode was continuous meaning the application continually queries the socket to determine if data is available to be read.

Now that the sockets are established, the controller's software code can pass data back and forth via MATLAB's "fread" and "fwrite" commands. For example:

```
if (u.bytesavailable >= 230400)
  for i = 1:120
      occ_map(i,:) = fread(u, 240, 'double');
  end
end
```

In this example, the desire was to receive a dataset representing one instance of the 120 by 240 data element state matrix. Each data element was represented by one bit. Therefore, that total dataset is 28,800 bits large (i.e., 120x240), or 230,400 bytes large

(i.e., one bit = 8 bytes). So, the code started by sampling the "u" buffer to see if a complete dataset is available for reading. If so, then the program read each of the 120 columns of data in the matrix one at a time and populated the "occ_map" variable correspondingly with the fread function.

Similarly, the computer requested data merely by writing a trigger value to the appropriate socket such as:

```
fwrite(v, 1, 'uint8');
```

In this case, the computer was writing the unsigned 8 bit integer "1" to socket "v" as a trigger to tell the state computer to send the state matrix.

Witnessed results from a plethora of agent experimentation has resulted in the conclusion that given a dedicated and isolated gigabit Ethernet backbone and UDP-based communications protocol, the agent and state computers comprising the multi-agent system robustly and reliably passed appropriate and complete datasets.

6.1.4 Image Processing

"A picture is worth a thousand words," or so goes the saying. In fact, today's robotic systems typically center their information collection capability significantly on the image and video collection and subsequently apply a wide variety of tremendously complex processing algorithms to extract valuable information for use in the control algorithms [124-129]. This information can feed local path planning, target detection and identification, beacon recognition, physical state determination (e.g., door open/closed),

and situational awareness. Recently, higher levels of intelligence are being implemented that provide clues to the more arcane such as intent.

For this research experimentation, image collection and processing was implemented that provided a capability whereby the agent could collect low-rate video, conduct image processing, and achieve target detection and identification. The goal was to characterize this capability in the MAS architecture and draw conclusions on the proportional contribution to the MAS' mission.

The e-puck robot included a 640 by 480 pixel camera that was capable of collecting both single images and streaming video. However, the relatively small amount of on-board memory significantly limited the capability of the robot to actually use the camera's full capability. On-board memory was limited to 8 KB of RAM and 144 KB of flash memory to accommodate the entirety of low-level mechanical and electrical system control and management, logic code hosting and execution, memory swapping, and storage. With the ePic bootloader installed on the e-puck robot, this effectively limited the single-image storage and forwarding capability of the e-puck to approximately 5 KB. This translated into a single-image processing capability of a 40 by 40 color image. An additional limitation is that the e-puck's Bluetooth communications throughput was bandwidth limited. This resulted in a throttling of image passing back to the governing computer of approximately 400 ms per image, or 2.5 Hertz sustained. Examples of collected imagery are included in Figure 30. Both are 40 by 40 pixel color images. The left shows a target and the right is a peer e-puck robot.

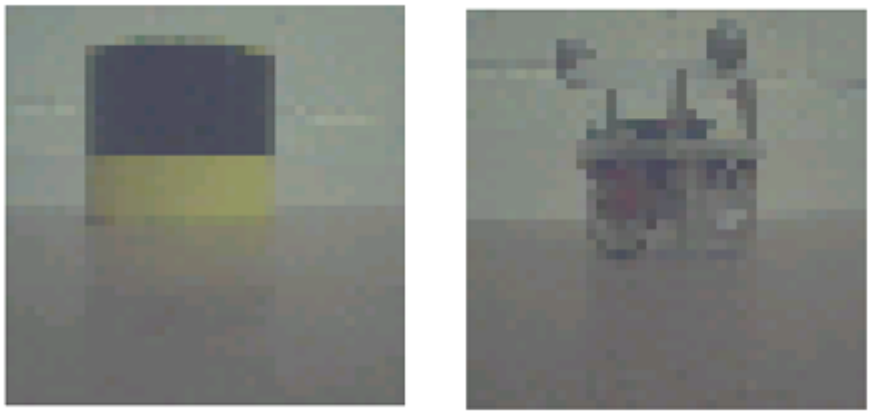


Figure 30: e-puck Image Collection

Visually, these images are clearly pixelated with relatively poor resolution. Nonetheless, they are data rich indeed and enable viable image processing. As the focus of this research was the proportional contribution of payloads (e.g. imagers) and not the image processing itself, a fundamental algorithm was utilized that would compare the collected image to a target library to uniquely identify the object as a target. Since each target was a fixed diameter vertical cylinder, a calibration routine was enabled that correlated the orthogonal projection of the target diameter to range to target. Knowing the range to target, agent position/location, and target diameter then allowed the calculation of the target's projected position and corresponding update of the probabilistic occupancy map [130].

6.2 Discussion of Experimentation Results

6.2.1 Experimentation Infrastructure

The novel surrogate-localization and inter-agent data-communications solutions developed for experimentation infrastructure proved to be robust and highly conducive to support control algorithm cycle rates. Based on the data collection capability and mobility of the individual agents, governing control schemes strive for cycle times less than 200 milliseconds to insure exhibition of responsive behaviors. The surrogate localization function created only minimal processing time burden consuming only low-single-digit milliseconds and with near-real-time latencies. This proved extremely supportive of the control scheme with the agents being able to accurately represent the locations of themselves, other agents, targets, and obstacles - certainly well within the 1 cm^2 resolution of the state map. A significant enabler to this was an ability to calibrate the accuracy of the OptiTrack system itself to sub-millimeter levels. Additionally, this function proved a robust capability to provide the variety of network-connected controllers immediate access to its own controlled agent, but also to the peer agents operating in the arena, all in near real time, and without having to burden the inter-agent communications architecture.

The UDP-based data-communications function also proved to provide a reliable capability to pass large datasets of state information among the networked computers. A single state map consists of 28,800 cells (120 by 240) each with a 64-bit data element. Therefore, the single full mapping approaches 1.85 megabits of data. Over the course of an individual experiment, hundreds of state maps were transmitted to the agent

controllers along with thousands of state map updates from the agent controllers. This means tens of thousands of packets navigating the network among the state and controller computers. Notwithstanding UDP's lack of an error correction capability, the data-communication solution herein resulted in zero packet loss realized across more than 300 individual experiments. Certainly, this was enabled by isolating the network on a single 8-port network switch; however, also enabled by careful consideration and definition of ports and buffers.

6.2.2 Parametric Analysis and MAS Characterization

The resulting experiment infrastructure enabled a comprehensive series of individual vignettes and excursions that explored varying MAS parameters such as the number of imaging payloads, number of agents comprising the MAS, and relative agent and cell charge values. These were evaluated against mission performance criteria such as time to explore a selected percentage of the identified space or the number of positive detections collected by proximity sensing or imaging. With these parameter and metric sets, methodical experiments were conducted sequentially to result in datasets that can be reduced and interpreted for qualitative and quantitative characterization of the relative and absolute performance of the various MAS configurations.

6.2.2.1 Data Collection Validation

Given the dynamic and probabilistic nature of the control theory and environment, it is important to validate the sets of data to insure that the information collected provides a verifiable basis from which to draw quantitative and qualitative conclusions about the MAS' behavior. There are a wide variety and not-insignificant quantity of factors that concatenate to have an impact on the realized system actions and performance. For example, the e-pucks' proximity sensors exhibited variation in terms of the curve representing distance versus signal. This variation certainly exists in comparing different proximity sensors, but was minimized through a baseline calibration process. However, the variation also somewhat exists temporally within a single proximity sensor with readings varying by percentage points and spiking infrequently likely due to ambient lighting. All of these disparate variations combine to add a degree of randomness in the actual computed motion of the agent. In fact, no experimental run can be completely replicated in terms of exact path taken or information collected. Nonetheless, by conducting series of runs with the same boundary conditions and parameter sets, datasets can be generated and analyzed en toto to inform the research process. The requirement then becomes to validate the datasets to insure that the variation is reasonable and that the set means truly conclude representative behaviors of the MAS.

The first set of experiments explored behaviors as a function of varying the number of agents, and among those, selectively operating with or without imaging sensors. For these series, the independent parameters were fixed and ten runs per set were conducted. Dependent variables (i.e., the evaluation criteria) were collected such as

time to complete the exploration, total distance traveled, and number of image or proximity target/obstacle detections. For the purposes of data validation, the time-to-complete metric was evaluated statistically. This excursion consisted of tens runs each of twenty MAS configurations. The distinction among the configurations was to vary the number of agents, and of those, vary the number of which had use of its imaging sensor. There is no necessary correlation between the sets of ten runs as each individual agent is unique as a function of its materiel construct and of course unique by condition of use of its imaging sensor. However, the ten runs within a configuration set should correlate. The statistical parameters utilized were the minimum value, maximum value, mean, standard deviation, coefficient of variance, 95 percent confidence level, and skew. During the data collection, datasets were rejected if there were obvious mechanical issues such as an agent's battery running low or processor fault causing erratic behavior. In addition, the minimum and maximum values would be compared to the mean. A run was rejected if either the minimum or maximum of any single collection was greater than 50 percent away from the mean. This action accommodates the case where there are not-readily-observable mechanical or electrical failures. In practice though, the vast majority of failures were catastrophic and immediately obvious.

The standard deviation provided a measure of variability to describe the level of dispersion away from the mean value. Lower values indicate better correlation among the runs and indicate consistency. However, the greater importance of this parameter is its contribution to the coefficient of variation (CV) which is the ratio of the standard deviation to the mean. The CV is a central focus of the validation process as it represents

both dispersion of the data points within a set, and scale of that data to the mean. Additionally, as a ratio, CV analysis allows the comparison of disparate information, i.e., experiment runs of differing configurations. Certainly, lower CV values are deemed better. Beyond that, CV evaluation is very much a function of the required data precision. If data is expected to be extremely precise, for example, repeatability of a manufacturing robot, then CVs less than 0.1 percent are expected. However, in the case of the e-puck robots comprising a MAS and the inherent variability of the system, CVs less than even 50 percent would be acceptable. The data collected shows CVs between 2.7 percent and 20.3 percent with an average across the configurations of 11.3 percent indicating excellent data validity.

The statistical parameter skew characterizes the level of symmetry of a distribution around its mean. Positive skew is a result of an abundance of values above the mean, while negative skew has that abundance below the mean. The equation utilized for skew γ is:

$$\gamma = \frac{N_d}{(N_d - 1)(N_d - 2)} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{sd} \right)^3 \tag{6-1}$$

where,

N_d is the number of data points,

$\bar{x}$ is the mean of the data set, and

sd is the standard deviation of the data set.

A perfect normal distribution has a skew of zero. Beyond that, if one strives toward that normal distribution, then a skew value, positive or negative, nearer zero is

better. For the data collected, the skew values ranged from -0.87 to 2.10 with an average value of 0.76. As a visualization of skew, Figure 31 plots data points calculated as the series mean minus the rank-ordered absolute value of the distance from a data point to the mean. This results in a curve shape mimicking a bell curve. Note the qualitative consistency of the curve. Therefore, with benefit of both the calculated skew and qualitative analysis of the data plot, again validation is confirmed by the excellent characteristics and behavior of the data sets.

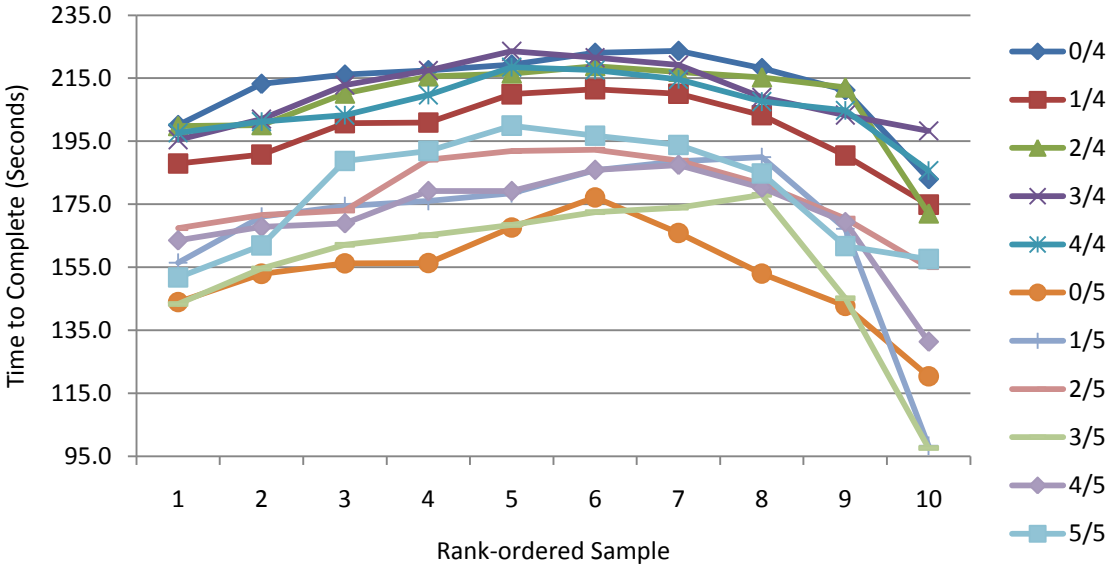


Figure 31: Time to Complete versus Rank-ordered Sample

The final consideration in the validation exercise is to contemplate if there were enough runs per dataset. All considered, more is better. However, pragmatics typically dictate the time and resources available. For these series of experiments, typically 10 and sometimes 5 runs comprised a set for a series. Methodical analysis of the runs as

illustrated with the treatment above consistently conclude that the CV and skew both confirm that run averages adequately represent the behavior and performance of the MAS.

Table 3: Statistical Analysis of a Data Sample

Number of Cameras per Agent/ Number of Agents	Experiment Number 1	2	3	4	5	6	7	8	9	10	Min	Max	Mean	Std Dev	Coef of Var	95% Conf Level	Skew
0/1	780.1	824.0	803.3	889.7	829.9	933.6	802.0	1065.9	910.9	940.4	780.1	1065.9	878.0	88.0	10.0%	54.5	1.03
1/1	851.7	825.2	768.0	867.1	755.9	714.4	821.6	880.6	837.3	851.1	714.4	880.6	817.3	53.8	6.6%	33.4	-0.87
0/2	394.2	373.6	450.4	431.6	384.5	475.6	455.7	435.7	398.3	455.0	373.6	475.6	425.5	35.2	8.3%	21.8	-0.20
1/2	485.8	427.0	414.3	440.7	396.5	389.8	465.8	524.5	473.3	413.3	389.8	524.5	443.1	43.3	9.8%	26.8	0.60
2/2	484.9	430.0	428.3	386.3	482.1	508.7	397.3	425.3	440.0	415.5	386.3	508.7	439.8	39.8	9.1%	24.7	0.54
0/3	358.9	292.2	292.2	226.3	261.7	252.2	258.0	276.8	262.0	229.7	226.3	358.9	271.0	38.1	14.0%	23.6	1.33
1/3	302.3	310.3	248.7	251.8	396.0	291.6	305.0	261.9	241.3	267.0	241.3	396.0	287.6	45.7	15.9%	28.3	1.55
2/3	260.4	271.1	274.2	268.4	271.4	275.1	263.0	271.1	287.9	274.4	260.4	287.9	271.7	7.5	2.7%	4.6	0.73
3/3	254.7	251.8	289.1	271.5	278.9	257.6	275.2	288.5	304.9	256.9	251.8	304.9	272.9	17.8	6.5%	11.0	0.44
0/4	223.0	270.3	219.3	213.2	242.1	217.5	229.7	235.1	216.1	200.1	200.1	270.3	226.6	19.3	8.5%	12.0	1.23
1/4	232.7	261.1	190.8	226.0	245.5	187.9	200.7	201.0	224.5	209.9	187.9	261.1	218.0	24.1	11.1%	15.0	0.46
2/4	272.4	210.1	200.1	215.6	199.8	218.9	227.5	216.5	229.3	232.5	199.8	272.4	222.3	20.9	9.4%	12.9	1.57
3/4	251.2	223.5	246.2	240.6	217.4	228.0	212.8	202.1	195.5	230.3	195.5	251.2	224.8	18.3	8.2%	11.4	-0.14
4/4	203.3	201.2	238.9	258.0	236.1	229.0	218.5	226.1	197.7	209.7	197.7	258.0	221.8	19.4	8.7%	12.0	0.46
0/5	153.0	156.4	167.7	156.2	206.5	144.0	177.2	251.9	229.6	219.4	144.0	251.9	186.2	37.7	20.3%	23.4	0.62
1/5	188.7	178.5	185.8	197.7	171.0	289.3	220.4	176.1	156.4	174.5	156.4	289.3	193.8	37.7	19.4%	23.3	2.10
2/5	171.6	191.9	173.0	167.5	219.1	192.4	200.7	208.1	234.5	189.1	167.5	234.5	194.8	21.6	11.1%	13.4	0.50
3/5	162.2	154.7	165.2	213.3	143.4	260.8	174.0	172.5	178.0	168.5	143.4	260.8	179.2	34.0	18.9%	21.0	1.81
4/5	169.0	179.2	206.7	179.2	167.8	188.6	185.9	163.5	244.6	195.8	163.5	244.6	188.0	23.9	12.7%	14.8	1.59
5/5	242.7	151.8	188.8	206.5	162.0	238.6	200.4	191.9	203.5	215.5	151.8	242.7	200.2	29.0	14.5%	18.0	-0.19
														AVE:	11.3%		0.8

Time to Complete

6.2.2.2 MAS Performance versus Number of Agents

Perhaps the most interesting conclusion drawn was that the relative performance of a MAS as a function of the number of agents comprising the MAS increased, but faded non-linearly (Figure 32). In this series of experiments, mission success was defined as exploring 75 percent of the arena's cells. Further, each agent was capable to use its on-board camera and associated image processing capability to detect and localize targets in the arena. Each agent could also use its proximity sensing for local collision avoidance and for object detection and localization. Excursions were run whereby the number of agents comprising the MAS was varied from one to five with ten experiment runs per excursion. Mean values were taken across the ten runs to conclude the excursion results. In an attempt to curve fit the data, both polynomial and power types were evaluated. A third-order polynomial curve (red line) provided a better fit (R^2=1) within the bounds of one to five agents; however, did not extrapolate as values above six agents drove the time to completion to negative numbers. A power curve fit (blue line) also resulted in an excellent fit (R^2=0.9898) and additionally provided better extrapolation beyond five robots with 6- through 8-agent MAS resulting in 158, 137, and 122 seconds, respectively. Additionally, this curve fit has an asymptote at 0 seconds as one would expect. Logically, this is the case whereby the entirety of the arena is covered by the concatenation of agents' sensors such that that entirety is exhaustively sensed in the very first control cycle’s data collection.

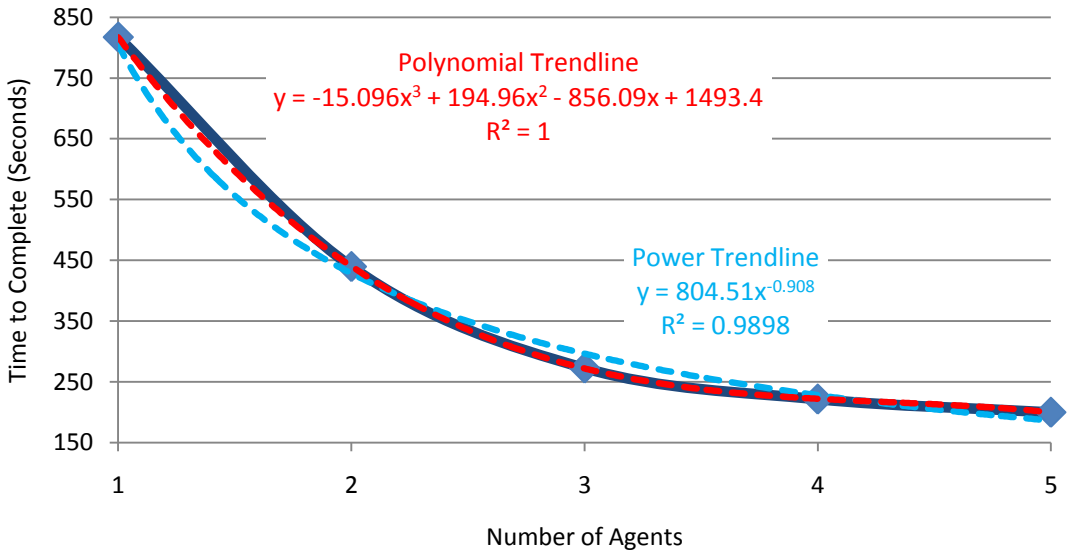


Figure 32: Time to Complete versus Number of Agents

One might be tempted then to simply continue to add agents to the MAS to achieve the best performance possible. However, the fallacy in this strategy is that each additional agent also brings additional costs in terms of procurement, maintenance, and energy consumption. Energy consumption (i.e., total cumulative distance traveled) versus number of agents from the excursion above generated Figure 33. This curve reflects the fact that consumption (here represented as distance traveled) increases linearly per unit time with the number of agents. However, the total time to successfully accomplish the mission is decreasing non-linearly. The combination of these two phenomenon produces a consumption curve akin to a 3rd degree polynomial. Note that curve fitting here resulted in relatively poor R2 values due to an insufficient quantity of data points. Comparing the two curves does confirm that the knee in the performance curve (at three agents) corresponds with the inflection point in the consumption curve.

The conclusion to draw is that increasing the number of agents comprising the MAS does continue to improve the overall performance; however, that there is a knee in the curve (very much dependent upon the materiel configuration of the MAS) above which adding agents results in ever-decreasing incremental improvements in performance while suffering linear increases in burden such as power consumption. At that point, agent census (agent quantity and individual capability) selection becomes an exercise in prioritizing and weighting the various metrics (e.g., performance is twice as important as energy consumption) and subsequently selecting census accordingly.

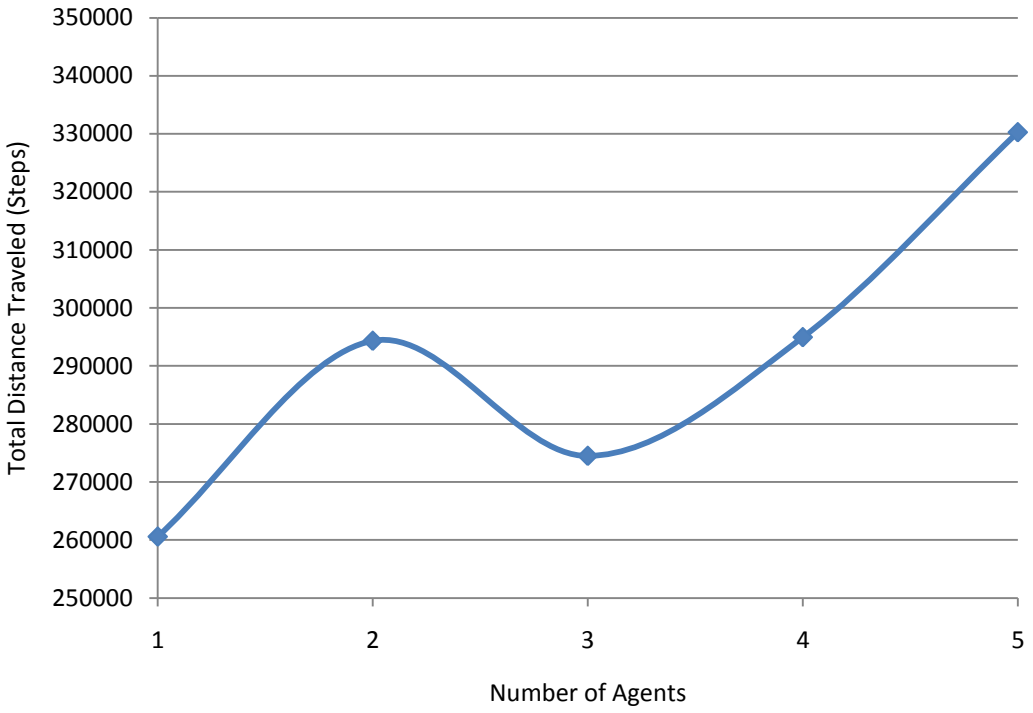


Figure 33: Total Distance Traveled versus Number of Agents

Another consideration for future research is to explore an upper quantity of agents comprising a MAS whereby the agents begin to impinge on each other's maneuverability. One can imagine a point where the discrete arena is so populated with agents that they effectively spend all their time executing local collision avoidance (of each other) as opposed to conducting their mission.

6.2.2.3 Potential-function Tuning Parameters

In the potential-function control strategy, arbitrary charges are assigned to elements of the complete system that are required to exhibit attractive or repulsive forces. For example, each agent is itself given a positive charge. One intentional desired result is that similar to like-charged magnets, the agents will repulse each other within the path planning process. The first significant benefit of this behavior in an exploration and mapping mission is that the agents tend to spread out more consistently avoiding inter-agent interference and bottlenecks. The second benefit is that this repulsion effectively supplements the collision avoidance algorithm such that higher proportions of time can be allocated in the control loop to global path planning (with a connotation of mission productivity) versus collision avoidance (with a connotation of failure avoidance).

The use of potential functions is also particularly well suited in conjunction with probabilistic occupancy mapping in that charge values can be calculated for each cell that comprises the state space representation as a function of any parameter of interest. For example, if a mission planner is exclusively interested in mapping, then the cell charge

can be calculated as a function of the cells' probabilities of occupancy with maximum charges assigned to those cells having the least information sensed. In a dynamic environment (i.e., moving targets and obstacles), one may increase cell charge as a function of the time passed since the last physical state sampling thereby weighing path planning back toward these cells.

One final concept to consider is the relative proportions of charges assigned in the architecture. As in the physical world, each entity in the arena (agents, objects, cells) has a charge assigned. In the real world, nature decides what charge entities have. However, in MAS experimentation, the researcher has an opportunity to assign charges that may or may not necessarily follow the rules of nature. If each agent in the system is absolutely identical, logic would dictate that each would have exactly the same charge. However, if an agent has a unique capability or assigned importance, then the mission planner very well could methodically assign a greater relative charge to that agent so as to influence its motion behavior. It is also important to understand that charge values are not necessarily absolute, but rather are relative to some arbitrary scale. For example, if the inherent charge of every entity in the arena was increased by an order of magnitude, the resultant behavior would remain unchanged. Changes only come when the ratio of charges between entities changes.

The scenario for this series of experiments in the conducted research was to focus on the ratio between the charge assigned to the cells and the charge assigned to the agents. This ratio effectively defines one potential tuning parameter previously discussed. All cells have the same assigned charge, as do all agents. Against an

exploration and mapping mission, these clearly are the two significant contributors to performance. As described earlier, the ratio between these two sets of charges is most important. This was accomplished by holding the agent charge constant (at 100 units) while varying the cell charge through values of 0.01, 0.1, 0.5, 1.0, 5.0, and 10. Five runs per cell charge excursion were conducted. The averaged results of the performance metrics are captured in Figures 34 and 35.

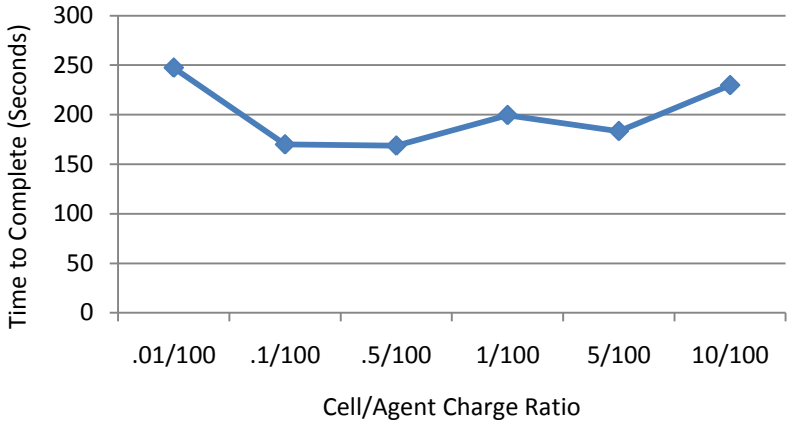


Figure 34: Time to Complete versus Cell/Agent Charge Ratio

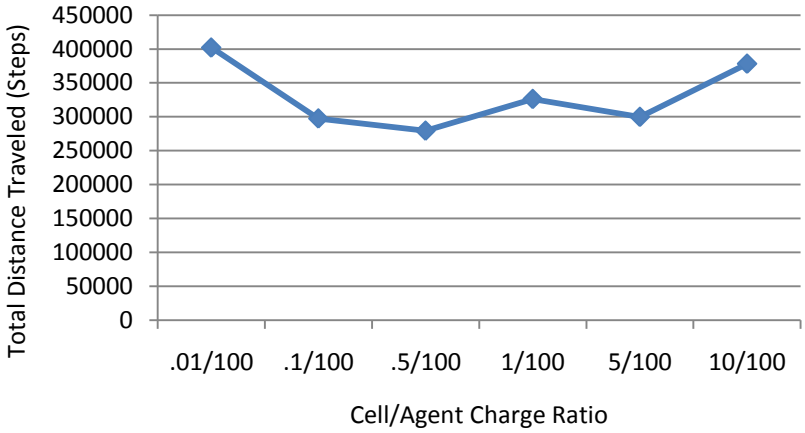


Figure 35: Total Distance Traveled versus Cell/Agent Charge Ratio

The first performance metric to be considered was time to complete the mission. Figure 34 clearly shows poorer performance at the extreme cell charge values with a 30 percent decrease in time to complete at both 0.1 and 0.5 charge units. The respective cell charge ratio maps (Figure 36) highlight a qualitative assessment that agent behavior appears more erratic with the extreme cell charge values. Additionally, observing the agents' behavior during these excursion runs concluded that the path planning behavior was clearly being over-influenced at these extremes. With the cell charge at relatively low values, the agents' behaviors were much more impacted by the inter-agent repulsive charge such that any attraction toward unexplored cells was overwhelmed. Behavior was also more erratic with relatively high cell charges assigned. Here, the path planning algorithm lost all benefit of inter-agent repulsive charge and the associated benefit realized by having the agents distribute across the arena. Therefore, extraordinary mission time was consumed as agents attempted to explore the same space, many times having to maneuver to avoid collisions as opposed to maneuvering to explore thereby inducing inefficiencies. The erratic behaviors are also confirmed by observing the total distance traveled (Figure 35). Clearly, this figure shows that the summation of distance traveled by the agents to accomplish the mission increases significantly at cell charges of 0.01 and 10 units confirming the inefficiencies realized to explore the arena.

Cell charge values of 0.1 and 0.5 units resulted in nearly identical times to complete while a charge value of 5 units showed a slight time increase of 10 percent. Interestingly, there was an increase of time to complete of some 18 percent moving from the 0.5 to 1.0 unit cell charge. Initial indications are that these variations are merely

within the probability of error of the sampling. With observed CVs among the excursion runs upwards of 20 percent in some cases, curve variance at 10 percent is not unexpected and likely does not indicate a phenomenon.

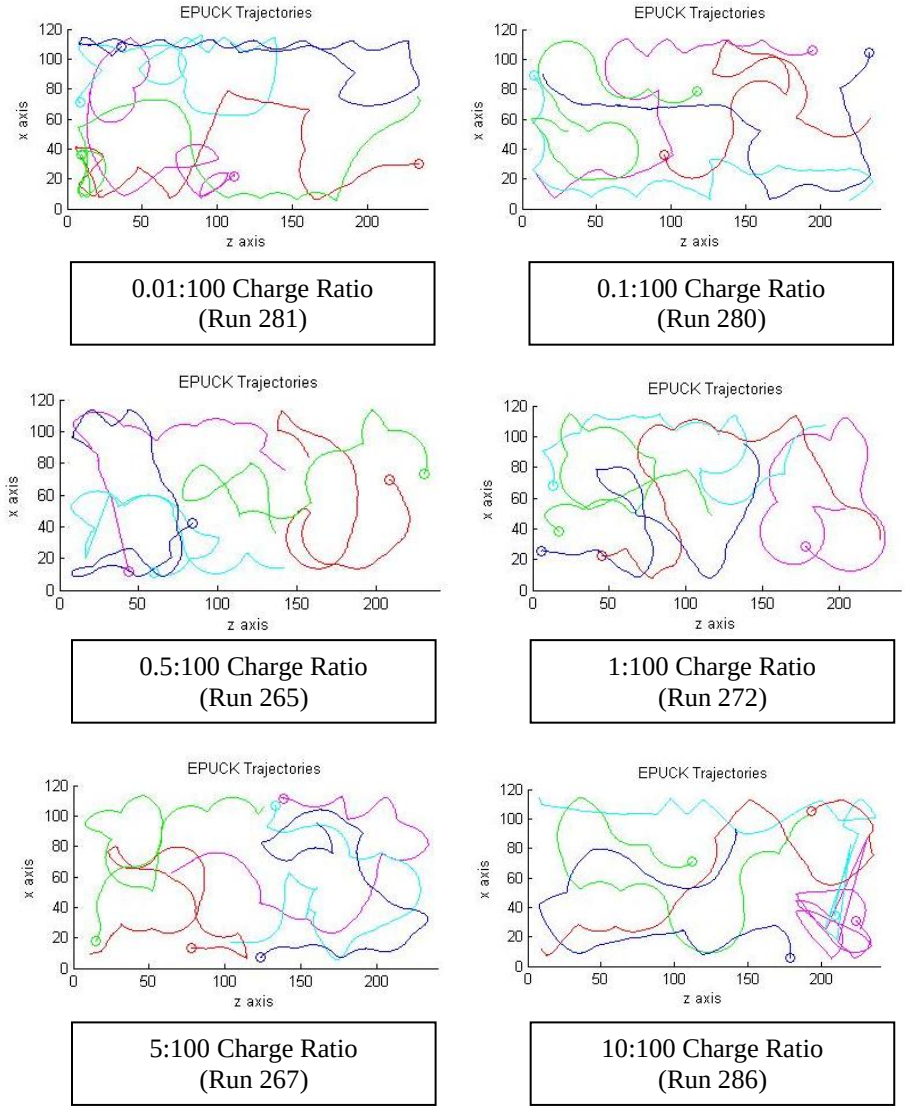


Figure 36: Charge Ratio (75 percent of Arena Searched)

Evaluating the temporal cell charge mapping (Figure 37) for the runs also illuminated a number of MAS behaviors. This mapping allowed a comprehensive overview of how individual cells influenced the total potential calculation and subsequent agent motion. During the simulation phase, this mapping highlighted the issue with not conducting object fill routines and the significant influence that unfilled objects had on the agents' motions. The mapping also highlighted the contribution of the arena's perimeter cells. Figure 37 shows that these cells, even late in the run, remain virtually unexplored and therefore continue to have force influence in the potential-function calculations. However, due to the symmetry of the arena, the cumulative force tends to cancel except proximate to the agent itself.

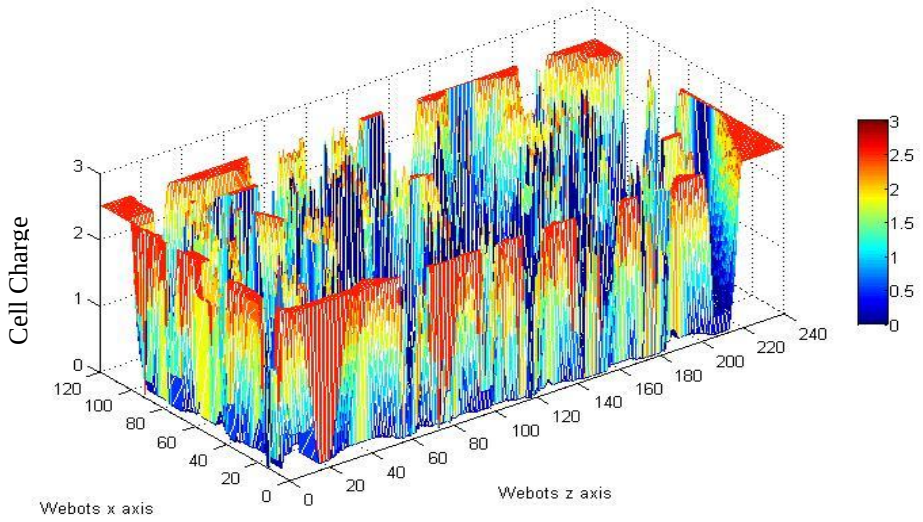


Figure 37: Calculated Cell Charge Mapping (Run 300)

The final performance metric to evaluate in this excursion is the number of detection events recorded by both the imaging sensor and the proximity sensor (Figure

38). As expected, these curves have similar shapes as the time-to-complete curve knowing that the likelihood that a sensor detects an obstacle or target increases linearly with the mission time. Specifically, there appears to be no relation between the cell charge and number of detections. This is merely intuitive as the cause of detections has more to do with the local proximate location of the agent as opposed to the global path planning calculation.

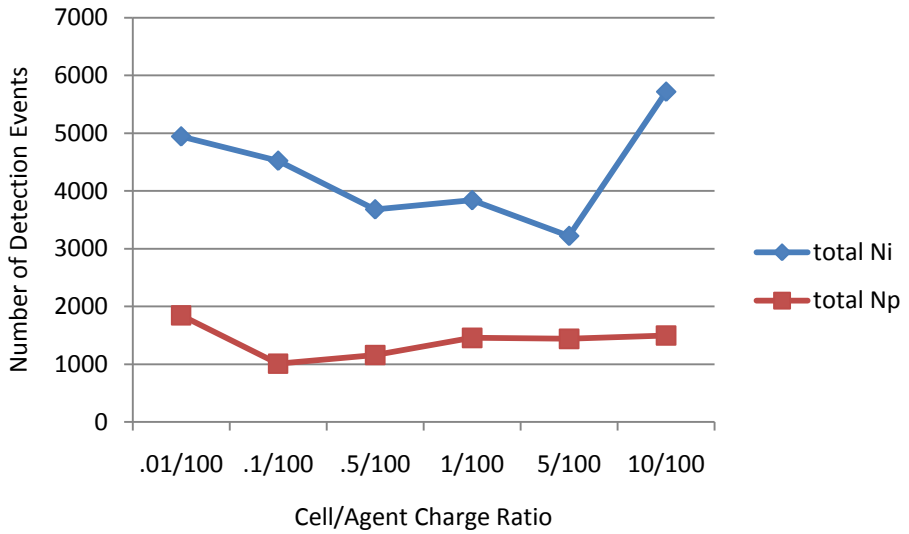


Figure 38: Number of Detection Events versus Cell/Agent Charge Ratio

6.2.2.4 Contribution of the Imaging Sensors

Experimentation conclusively demonstrated that imaging sensors provided a significant number of target detection and identification updates that subsequently updated the probabilistic occupancy mapping. Figure 39 summarizes a series of excursions whereby the MAS was sequentially comprised of one to five agents. Within each of these, the number of agents able to utilize its respective imaging sensor was

varied. For example, the green triangle curve shows a three-agent MAS where zero, one, two, and finally three of the agents were imager enabled. These families of curves clearly show that as the number of imagers increases, so increases the total number of image detections provided as one may very well expect.

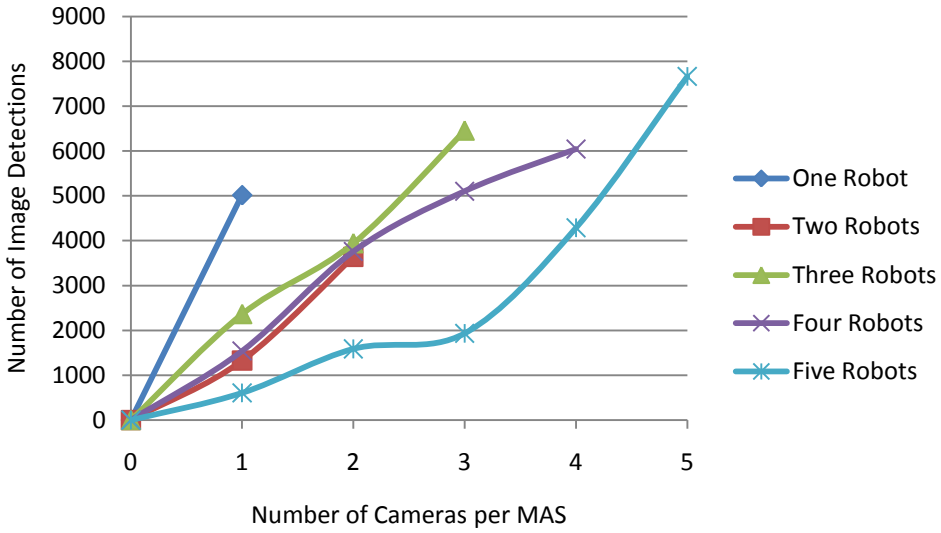


Figure 39: Number of Image Detections versus Number of Cameras per MAS

This may seem to be an obvious conclusion. However, this data must be taken in context. Figure 40 shows these same excursions but against the time to complete. Witnessed here is the fact that time to complete is not a function of the number of imager-enabled agents. The explanation of this phenomenon enlightens a significant observation for this research. The reason that time to complete is not decreasing is that the potential or opportunity that imaging has to contribute to the overarching MAS mission is relatively small when compared to the other two data collection capabilities - proximity sensing and agent localization.

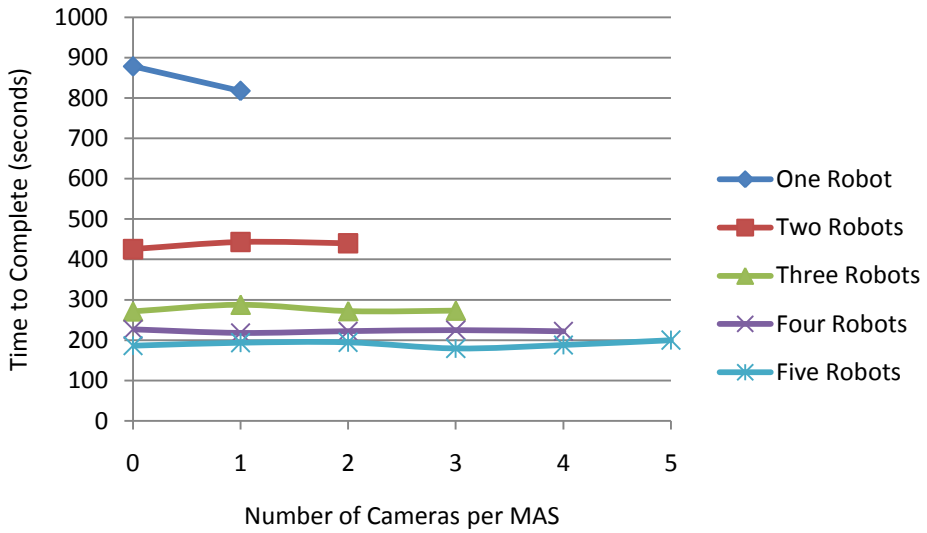


Figure 40: Time to Complete versus Number of Camera per MAS

In this scenario, the mission is effectively divided into three separable tasks: explore unoccupied space, localize obstacles, and localize targets. The first task is exclusively enabled by the agents' collective localization knowing that if an agent occupies space, that space cannot be occupied by an object at that point in time. The second task is exclusively enabled by the proximity sensors used in conjunction with the occupancy-map image-processing routines. The third task is exclusively enabled by the imaging sensor and image-processing-enabled target identification routines. Therefore, the contribution that any mission payload may have toward mission accomplishment can largely be a factor of the potential the payload has to accomplish mission tasks.

Strictly as a function of area, Table 4 summarizes the relative proportions of unoccupied space, obstacles, and targets in the experiment scenario. As typically witnessed in the real world, unoccupied space dominates that proportion with over 95

percent of the total available. Therefore, against an exploration task, the potential to sample space and characterize the probability of occupancy is greatest with sensors capable to explore unoccupied space.

Table 4: Area Contribution of Obstacles and Targets in the Arena

	Quantity	Area per Object (cm^2)	Total Area (cm^2)	Percent of Total
Total Arena			28,800	
Unoccupied Cells		1	27,546	95.6
Obstacles	6	198	1,188	4.2
Targets	3	33	66	0.2

Table 5 summarizes the total number of detections from each of the three information-collecting mechanisms in a five-agent excursion where each agent has full payload capability. From that, the total area sampled by each sensor is calculated. Unlike the ratios from Table 4, the imaging sensor indeed accomplished over 33 percent of the total area sampled. However, this doesn't accurately represent the impact of imaging because it does not reflect the redundancy with which the imaging sensors revisit the same three targets in the arena. True, there are many target identification events, but after a small number, the added benefit of continuing identifications becomes merely redundant therefore confirming the fact that increasing imaging does not have furthering impact on time to complete the mission.

Table 5: Area Contribution of Agent Sensors

	Number of Detections	Area Sampled per Detection (cm^2)	Total Area Sampled (cm^2)	Percent of Total
Localization	2,850	169	481,650	65.9
Proximity Sensor	1,500	1	1,500	0.2
Imaging Sensor	7,500	33	247,500	33.9

6.2.2.5 Contribution of the Proximity Sensors

Proximity sensing was a critical contributor to the overall capability of the individual agent and the consolidated MAS. Principally, the proximity sensors were used to detect objects in the vicinity of the maneuvering agent and, via Braitenberg calculations, maneuver the agent in a reactive away. Given that the agent knew its own position and orientation, and orientation and range of the individual proximity sensors, the agent could calculate the intercept point of the sensor and object and determine the specific cell. That information enabled update of the occupancy map. Additionally, the proximity sensors were used to confirm free space. As the agent explored the arena, an absence of proximity detections allowed the agent to conclude that the space explored was free out to the effective range of the sensors, approximately 4 cm radially outward from the perimeter of the agent. This mechanism resulted in significant data contribution to the exploration task, typically more than 60 percent of the total as shown in Table 5.

The metric utilized relative to proximity sensing was the number of proximity sensor detections. A single detection is defined as the action of a single proximity sensor

in a timestep sensing a specific cell in pursuit of object detection. In the case of no objects, there would be a total of eight detections (i.e., detecting the absence of an object) in a timestep per agent as each of the agent's eight sensors would be sampled once. If an object was present, local collision avoidance would cycle within the timestep meaning there could be tens or hundreds of samplings in a single timestep.

The first relationship to explore is the number of proximity sensor detections as a function of number of agents and therein, the number of cameras per MAS (Figure 41), both in the context of time to complete the exploration mission. As concluded previously, increasing the number of agents does decrease the time required to complete the mission in a non-linear fashion (Figure 41, blue curve). Similarly, the average number of proximity sensor detections increases non-linearly (Figure 41, red curve) in a rough mirrored approximation to the time curve. Recall though that the total contribution of proximity sensing in terms of area explored is quite small (Table 5), in fact, only approximately 0.2 percent of the total. However, proximity sensing has a magnified impact in that a single detection is a critical enabler to the object fill image processing algorithm. Therefore, although a small number of proximity sensor detections result themselves in small areas explored, they can impact up to the total combined area of obstacles and targets.

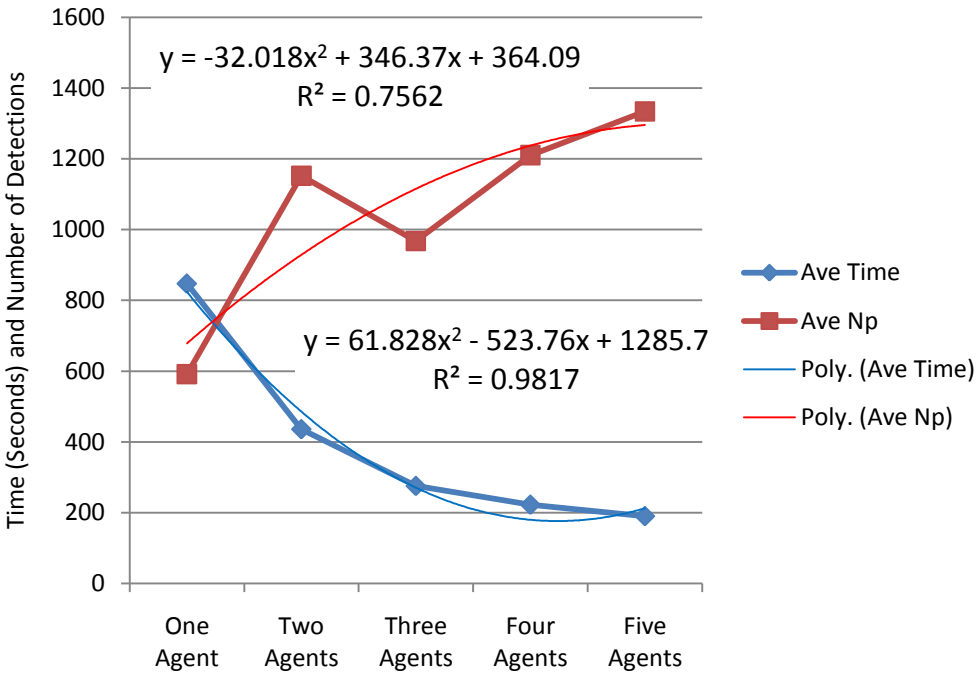


Figure 41: Average Time and Number of Detections versus Number of Agents

One of the original hypotheses was that as the number of cameras per MAS increased, the number of proximity sensor detections would decrease. Intuitively, this might be suspected thinking that the capability afforded by the imaging sensor would supplant the necessity for proximity sensing. Even though both are occurring simultaneously, it is the combination of the sensing capabilities that ultimately enable mission success. However, Figure 42 shows this not to be the case. In fact, the number of proximity sensor detections remains relatively constant, or in the isolated case of two agents goes up some 50 percent, as a function of number of cameras per agent. The explanation here harkens back to the concept of sensing opportunity. For the proximity sensors, opportunity is a function of area sensing and object sensing. The former is

somewhat limited as stated previously. For object sensing, Table 4 shows that a mere 4.4 percent of the total area consists of obstacles plus targets. Therefore, the sensing opportunity for the proximity sensors is similarly limited. With this disproportion highlighted, it becomes understandable that the number of proximity sensor detections is not well correlated to cameras per MAS as it is such a minor contributor to the mission. This conclusion is also confirmed by the fact that experimentation showed that the average total number of images collected in a five-agent excursion, with and without benefit of proximity sensing, were remarkably close at 1047.28 and 1046.24, respectively.

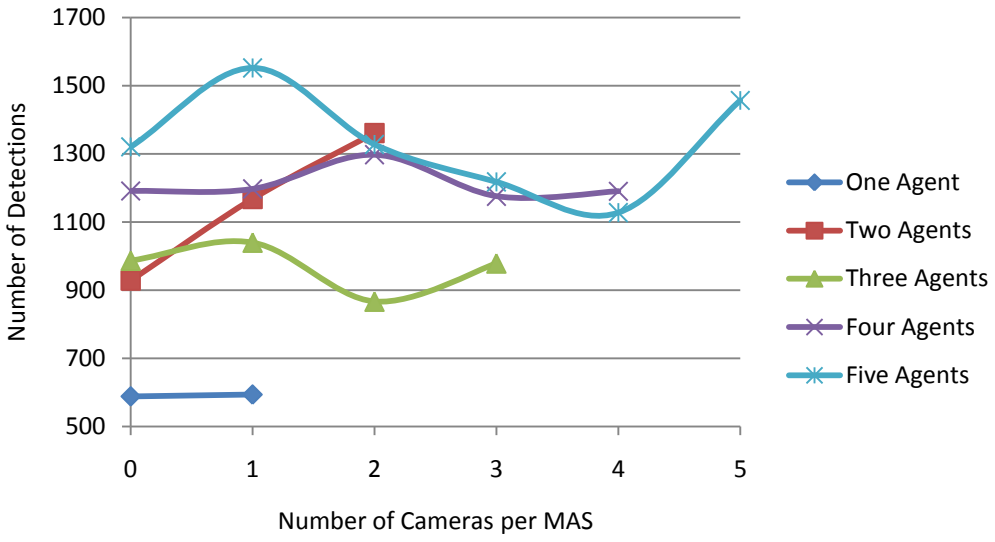


Figure 42: Number of Detections versus Number of Cameras per MAS

The next series of excursions explored the impact, in a five-agent MAS, of varying the number of agents within that had proximity sensing. The proximity sensors would still be used for local collision avoidance, but would not provide detections based

on object localization to the occupancy map. Each had benefit of the imaging sensor, except one excursion removed all imagers for comparison. In stark contrast to the number of cameras per MAS (Figure 42), the time to complete a mission shows a decreasing non-linear relationship to the number of proximity sensors per MAS (Figure 43).

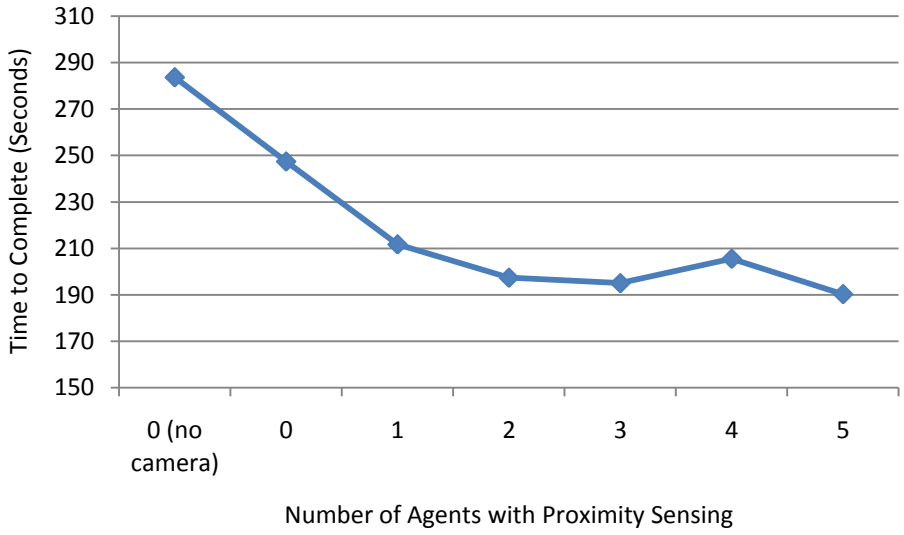


Figure 43: Time to Complete versus Number of Agents with Proximity Sensing

The reason for this is directly related to the object-fill capability within the occupancy map. Recall that the algorithm for object fill requires a proximity sensor confirmation. Therefore, as the number of proximity-sensor-enabled agents decreases in the MAS, to the opportunity for object fill decreases. This is illustrated in Figures 44 and 45. Both show the progression of the number of explored cells as a function of time, the difference being that the former does not have benefit of any proximity sensing. Examining Figure 45, one can clearly see the step increase in number of cells explored in

the 800 timestep region where the object fill algorithm successfully increases the number of cells explored.

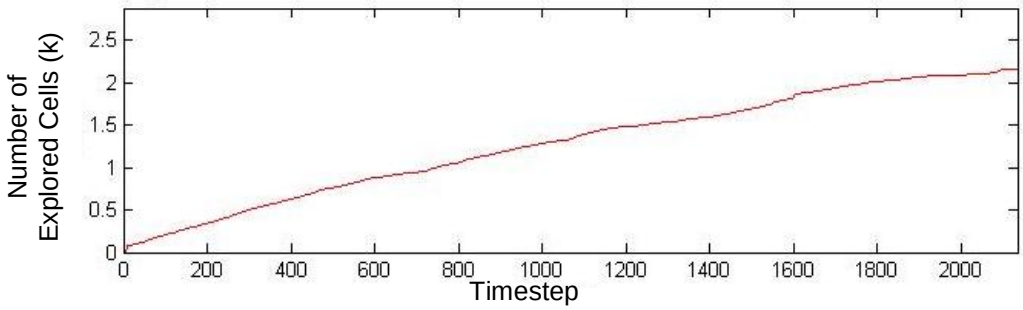


Figure 44: Number of Explored Cells versus Timestep, No Proximity Sensing (Run 241)

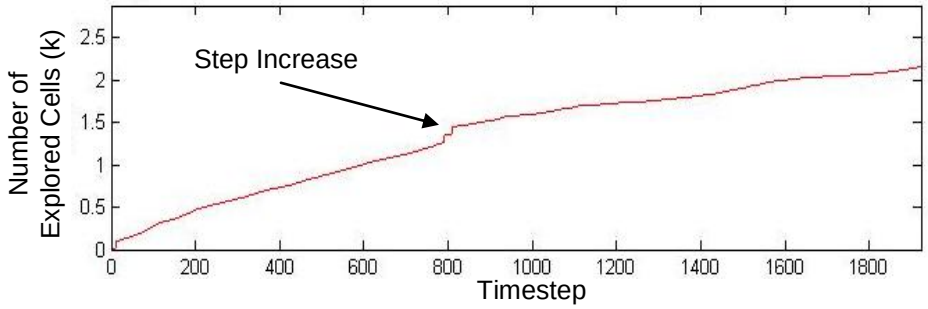


Figure 45: Number of Explored Cells versus Timestep, All Proximity Sensing (Run 144)

This is also confirmed by Figure 46 which shows the total number of proximity detections as a function of the number of agents (red curve). As agents are added to the MAS, so increases the total number of detections thereby increasing the probability that object fill will be successful. A separate observation from Figure 46 shows that the contribution from a specific agent of the MAS (agent number 2045) maintains approximately the same number of proximity detections regardless of the census of the

MAS. This supports the hypothesis as the opportunity for any individual agent to detect objects via proximity sensing is more a function of the object density.

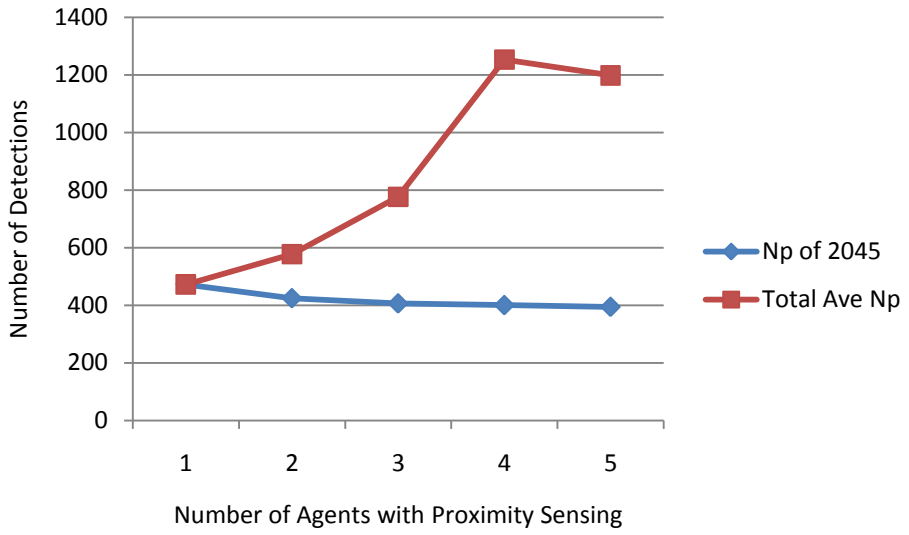


Figure 46: Number of Detections versus Number of Agents with Proximity Sensing

6.2.2.6 Completion Percentage

For both the simulation and hardware experimentation phases of this research, a test run was defined as complete when a pre-defined percentage of the arena's cells had been explored at least once. The arena consisted of a 120 by 240 matrix of cells or 28,800 individual cells. If the completion criterion was 85 percent, then completion was achieved when 24,480 of the cells had been explored. Prior to implementing the object-fill algorithm in the occupancy mapping software code, this criterion was a requirement because without that capability, the maximum arena area possible to explore was the total (28,800) minus the summation of object area. This was due to the physical reality that the agent's sensor suite could not explore the interior of objects. For example, Table 4

documents an experiment configuration where the total navigatable area sums to 96.4 percent of the total space available.

As a vignette to the core experiment and to verify the expected behavior of this requirement, an excursion explored the various evaluation criteria versus the completion percentage. Starting with a representative individual run (Figure 47), one observes that the slope of the exploration curve decays slightly over time. The step increases in the curve highlight where the object-fill algorithm is successful. This decay is due to the fact that the availability and proximity of unexplored cells decreases over time. This is an inefficiency created in the motion where an individual agent must transit increasingly larger areas of already-explored space in order to arrive proximate to the unexplored space. One can imagine that at the onset of a run, every cell is unexplored, so any motion results in accomplishment. However, as time goes on, the individual agents are increasingly forced to transit more explored space during which, no exploration progress is made.

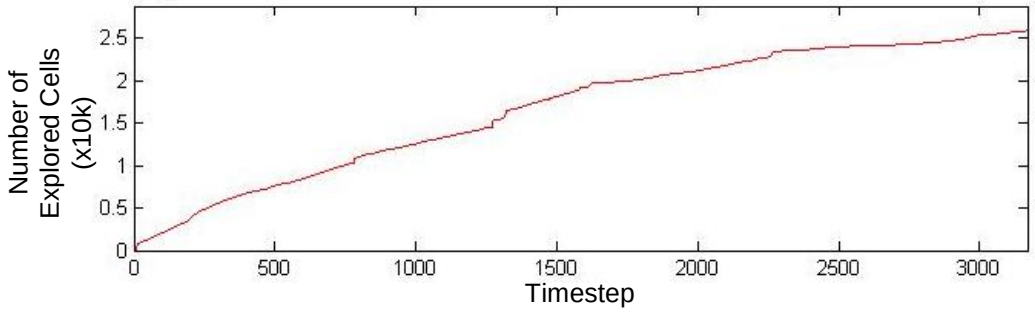


Figure 47: Cells Explored (x10k) versus Timestep (Run 297)

In order to amplify the later periods of the runs, Figures 48 and 49 represent averages over several runs comparing the time to completion and distance traveled versus completion percentage. The completion percentage is increased from 75 to 90 percent in increments of 5 percent. This also effectively increases the total number of cells to explore linearly. Sample agent trajectories for each of these excursions are included in Figure 50. Interestingly, magnifying this portion of the curves show a near linear increase in both metrics. This simply confirms the fact that the change in slope over time is indeed slight and that a confidence exists whereby curtailing the experiment at lower values of completion percent explored is valid. Speculation holds that adding numbers of agents beyond five would also confirm this confidence as the additional agents would offer even better coverage of the arena and shorter travel distances to proximate unexplored areas. Future research could extend completion percentages out to 95 and 100 percent.

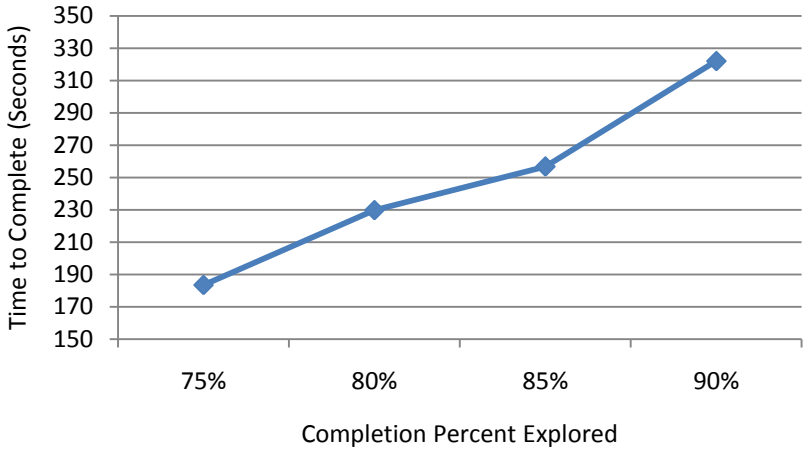


Figure 48: Time to Complete versus Completion Percentage

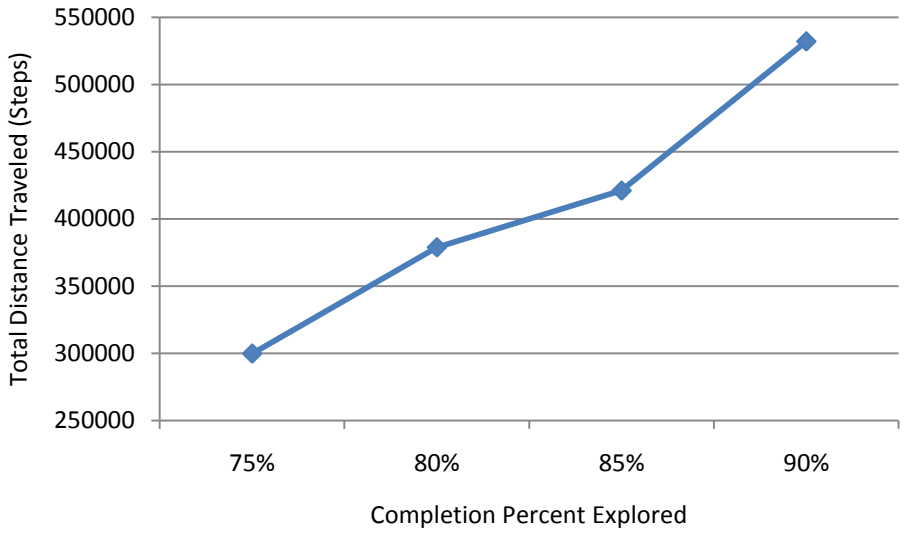


Figure 49: Total Distance Traveled versus Completion Percentage

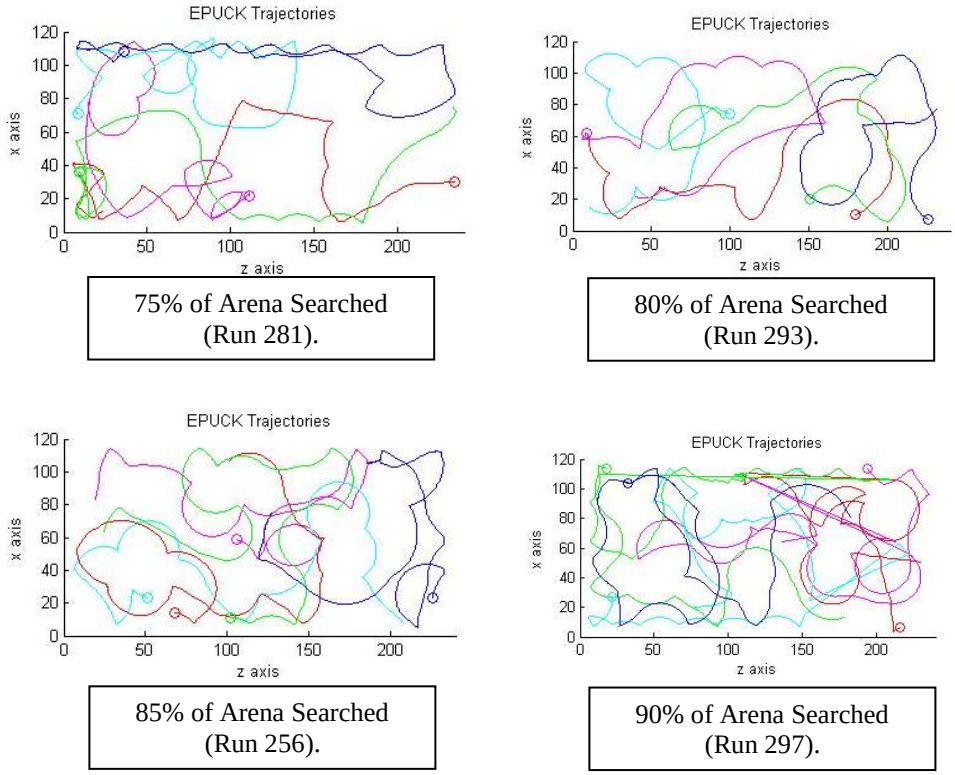


Figure 50: Agent Trajectories versus Completion Percentage

Figure 51 shows the resultant occupancy mapping after a representative test run of the 90 percent completion criteria. It is interesting to observe that the majority of the remaining unexplored space exists at the outer perimeter of the arena. This phenomenon occurs for two reasons. First, as discussed previously, the agents must transit area periodically to arrive at unexplored space. Mathematically, this transit space will more probabilistically exist in the center of the arena as opposed to the perimeter because there is no potential associated with area outside the arena, i.e., on the other side of the perimeter. Secondly, the arena itself is bounded by a low wall. Because of this, the agents' proximity sensing and local collision avoidance tend to dominate control action at the perimeter of the arena supplanting the potential function path planning. This is clearly observed in the rounded boundaries at the perimeter where the agents 'bounce' off the wall as they exercise the Braitenberg algorithm.

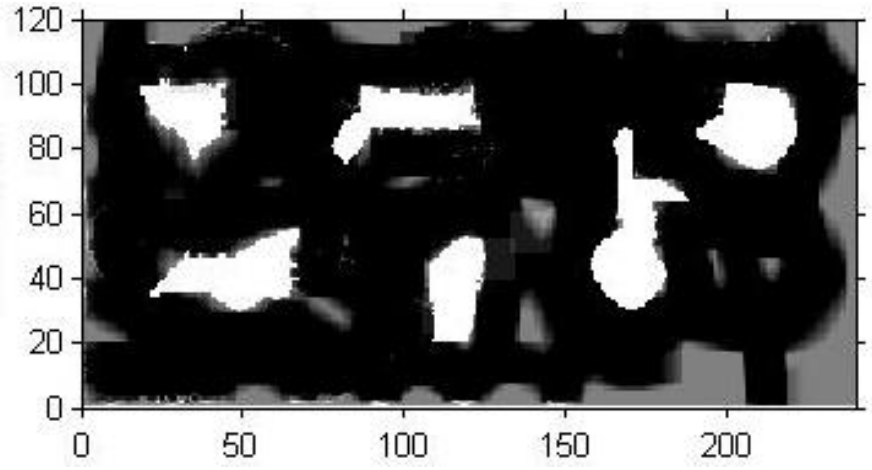


Figure 51: Occupancy Cell Mapping

The overarching conclusion to draw from this analysis is that curtailing an experiment run at some value less than 100 percent does not have a negative impact on the accuracy of any other observation. The progression behavior at this stage in an experiment is significantly linear and therefore allows accurate extrapolation.

6.2.2.7 Direction Bias

Symmetry is a consistent characteristic of the agents and the control algorithms. The robots themselves are symmetric about the center axis in terms of position and orientation of the proximity sensors, position of the drive wheels, and orientation of the camera. The control algorithm too strives toward symmetry most notably with the Braitenberg matrix utilized that reflects around the center axis. Finally, even disparities among proximity sensors is negated through calibration that purportedly baselines each sensor to result in commensurate readings. Data collected during each experimental run included the distance traveled by each agent's left and right wheels. This information was particularly valuable to determine agent energy consumption acknowledging the correlation between distance traveled and energy. An additional interesting vignette was to plot the ratio of left wheel to right wheel distance traveled over the number of runs (Figure 52). This ratio effectively shows the turning direction bias of the system whereby a value greater than 1.0 indicates that on average, the left wheel traveled farther than the right wheel resulting in a right-turn bias. A value less than 1.0 indicates the opposite where the left wheel traveled less than the right wheel resulting in a left-turn bias. As

Figure 52 shows, there are experiment samples that exist both above and below the 1.0 threshold. However, averaging the bias calculation over the litany of experiments results in a value of 1.03 or a 3 percent bias toward left turns. This is qualitatively confirmed to some degree by referring to the chart. There are a number of explanations for this perturbation. Certainly, inaccuracies in the proximity sensor calibration, non-linearities in the mechanical system, or manufacturing variances (e.g., the drive motors themselves) would contribute to motion biases. The observation here though is to understand that MAS control in a global sense must accommodate these biases and be able to overcome if not leverage their existence. A 3 percent left-turning bias perhaps does not impress; however, in considering total MAS energy consumption over time, this 3 percent equates to waste that must be compensated. This is a detriment indeed.

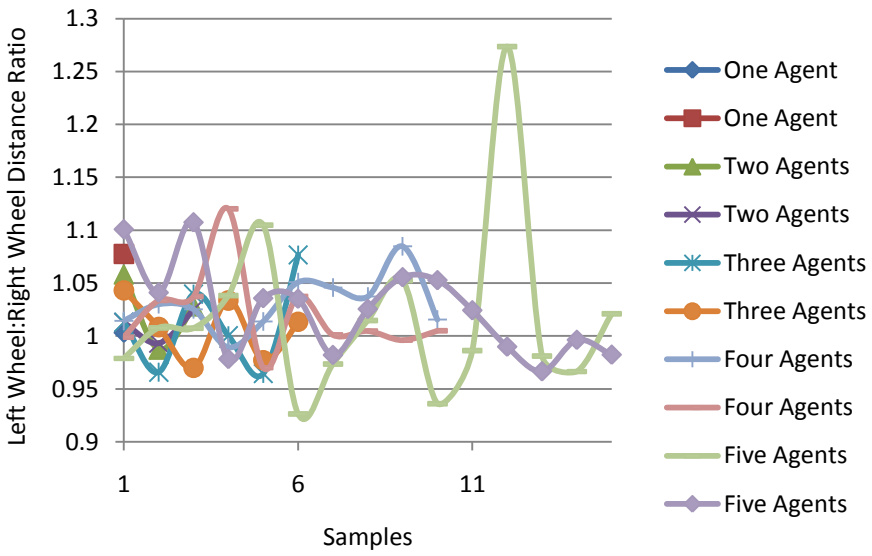


Figure 52: Wheel Distance Ratio

Chapter 7. Conclusions

7.1 Summary

Today's mobile robotic platforms offer tremendous capability in terms of exploration, reconnaissance, manipulation, and sensing with scales that range from micro-sized "insects" to larger wheeled and tracked vehicles. Imagers, LIDARS, proximity sensors, and ultra-sonic sensors comprise only a few of the information collection capabilities available. Recent research has centered on grouping these individual robots into multi-agent systems (MAS) with the belief that the combined effort of multiple coordinated robots will result in capabilities beyond the sum total of the individuals. Toward that end, control theory has delved into the interaction among and overarching behavior of these coordinating agents both decentralized within the agent, and centralized within a MAS supervising entity. However, to date, there has been little work done with respect to understanding the capability of the MAS as a function of its composition in terms of quantity or capability, given the option of pre-mission selection of the performance and capability of individual agents.

The goal of this novel research effort was to address the impact that variations in a pre-selectable composition of a MAS can have on its resulting capabilities and performance while utilizing a composition-tunable potential-function control scheme. State space representation provided a critical and comprehensive architecture with which to capture and manipulate probability-based occupancy mapping and cell vector parameters in conjunction with potential-function tuning factors to result in desirable

agent and MAS motion. Numerous experiments were run, both in simulation and hardware experimentation, to test combinations of system parameters to garner a better understanding that those parameters have on the overarching performance of the MAS.

Research conclusions showed several relationships and novel behaviors important to an a priori mission planning exercise. Increases in agent quantity resulted in a non-linearly fading increase in performance measured by time to complete a pre-defined mission. This non-linearity means that more numbers of agents are not always better, especially in consideration of logistic burdens that increase with additional quantities. In fact, unique to each agent/MAS configuration, there is likely an inflection point that optimizes the cost/performance curve.

Potential-function tuning factors provided a tremendously powerful mechanism to optimize control efficiencies and mission performance. For example, manipulation of the relative charges associated with MAS components and occupancy mapping resulted in a 25 percent improvement in task completion evaluation criteria. Further, behavior foci could be emphasized effectively through an artificial increase or decrease of the charge associated with the desired behavior. For example, agent charges could be increased to result in a more spatial distributed exploration, i.e., the individual agents repel each other to a larger degree during path planning calculations.

The progress that a MAS achieves during an exploration task increased consistently during the mission; however, faded non-linearly. This was directly a result of the fact that over time, spatial efficiencies decreased as the individual agents must transit longer distances to reach heretofore unexplored area. Nonetheless, derivation and

application of Lyapunov functions based on the underlying potential-function control showed that the motion is stable.

Perhaps the most interesting, albeit qualitative conclusion drawn was that the ultimate performance and potential of a MAS must be considered in reference to the "sensing opportunity" of the system. During the course of this research, it became readily apparent that a sensor's contribution to the MAS was not a function exclusively of the sensor's inherent capability. Rather, it was that plus the real opportunity the sensor had to perform. Imagine a shortwave infrared (SWIR) hyperspectral sensor consisting of 50 discrete collection channels in the 3-5 μ band - a phenomenal capability indeed. However, if the target of interest emits principally in the 8-12 μ band, the opportunity for the SWIR hyperspectral to be useful is minimal, i.e., its sensor opportunity is low. Similarly, a MAS is composed of agents with similar or varying sensor payload capabilities. The net sum of these capabilities does not, in and of themselves, define the MAS capability. Rather, that evaluation must be made in reference to sensor opportunity to truly understand the resultant capability of the MAS.

This sensing opportunity phenomenon was validated by the fact that utilization of higher numbers of imaging sensors did not result in increased MAS performance. It has been long known that imagers provide a rich data set from which to draw information and guide control schemes. However, given the construct of this experiment, the sensing opportunity for the imagers was limited and somewhat redundant. Therefore, the realized

contribution too was limited. In fact, excursions in a five-agent MAS varying the camera census from zero to five showed no improvement in MAS performance.

In stark contrast, proximity sensing made significant contribution to the witnessed MAS performance in spite of the sensors' relatively limited range and accuracy. Here, the sensing opportunity is high because the proximity sensors enable unique verification of unoccupied space and also confirm object perimeters for the image-processing-enabled object-fill routine. As a result, performance as a function of proximity sensing increases significantly moving from zero to five agents being proximity sensor enabled in the MAS. However, this curve too fades non-linearly such that continuing to add agents to the MAS results in only incremental improvements to performance.

Mission planners will have a near-infinite combination of payload quantities and types available from which to construct MASs. As such, it is critically important to inform that planning process with guides for efficiencies to insure that these important missions can be accomplished in a capable yet cost effective manner. This research shows several important relationships valuable to that mission planning process. Most importantly though, it highlights the fact that careful consideration must be made in that construction process in terms of affording selected sensors appropriate opportunity to accomplish desired behaviors.

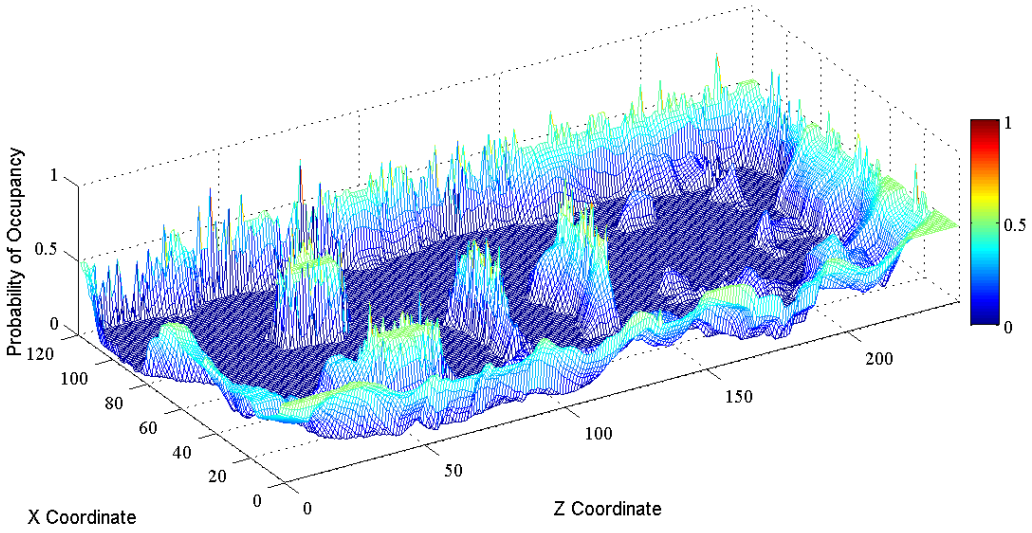
7.2 Suggestions for Future Work

There were several opportunities identified during the conduct of this research that pose related challenges, the solutions to which could further enlighten and enhance the core research thrusts. At a fundamental level within the potential-function construct, one might explore the cell charge versus occupancy probability curve shapes such as step functions or polynomials. The associated charges could be manipulated individually, or even adjusted temporally during a mission to pursue optimization of agent behaviors. Increasing MAS and scenario parameters, including agent quantity, agent capability, communications range, and mobility would each offer interesting complexities to explore resulting in separate observations and characterizations. Increasing the arena area and more valuably arena dimensionality poses significant processing and data management challenges, but certainly is required to approach real-world conditions. Increasing MAS heterogeneity, notably adding airborne agents or overwatching immobile sensors, foretells increases in capability.

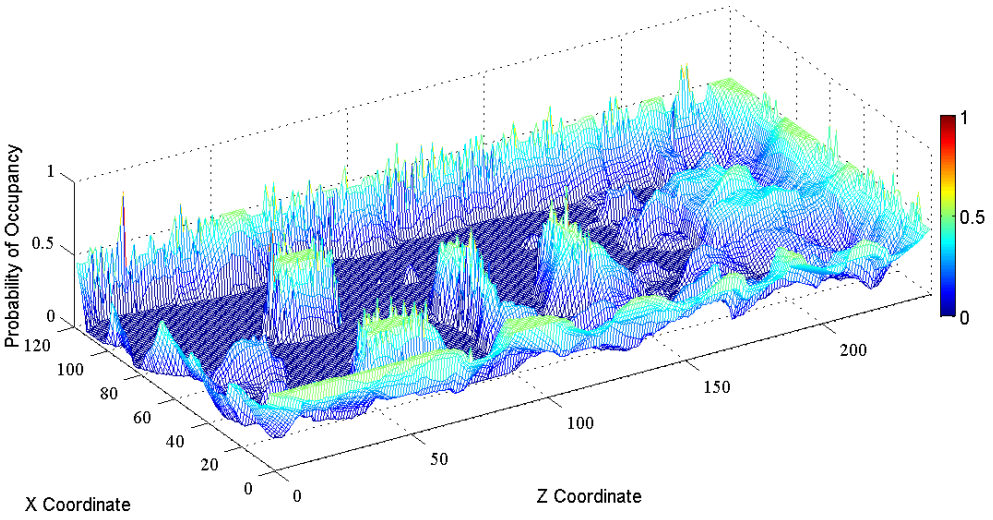
Ultimately though, the result of any continued research should strive toward and culminate with articulation fine enough to inform real-world applications consisting of mission planning and the associated implementation to result in optimization and success.

Appendix A. Simulation Results

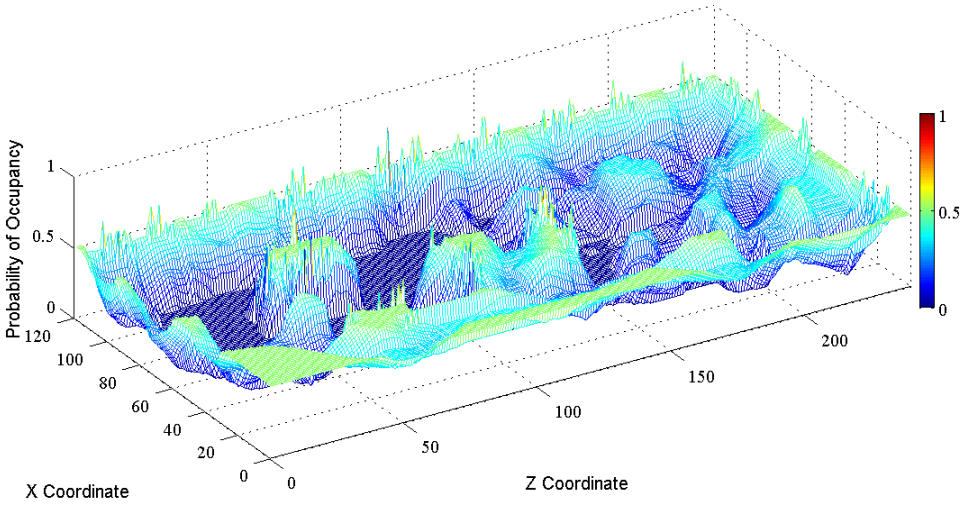
OCCUPANCY MAP RESULTS.



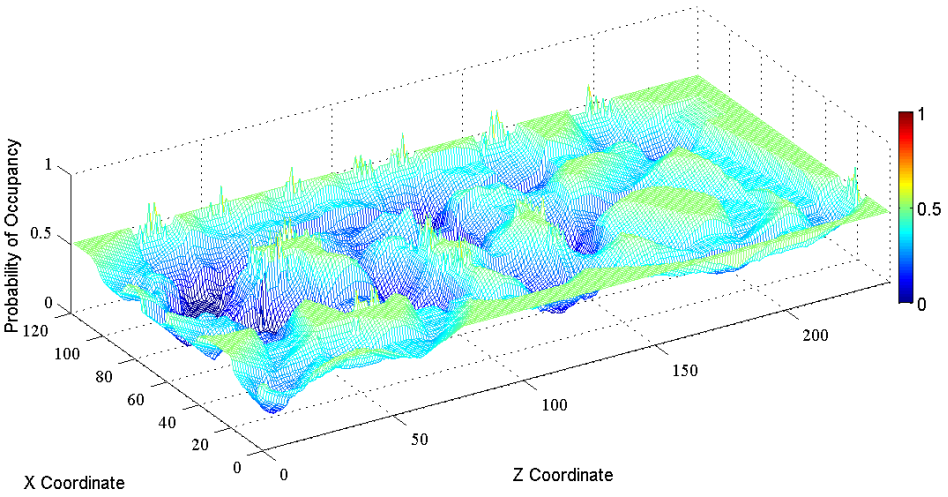
Four Robots



Three Robots

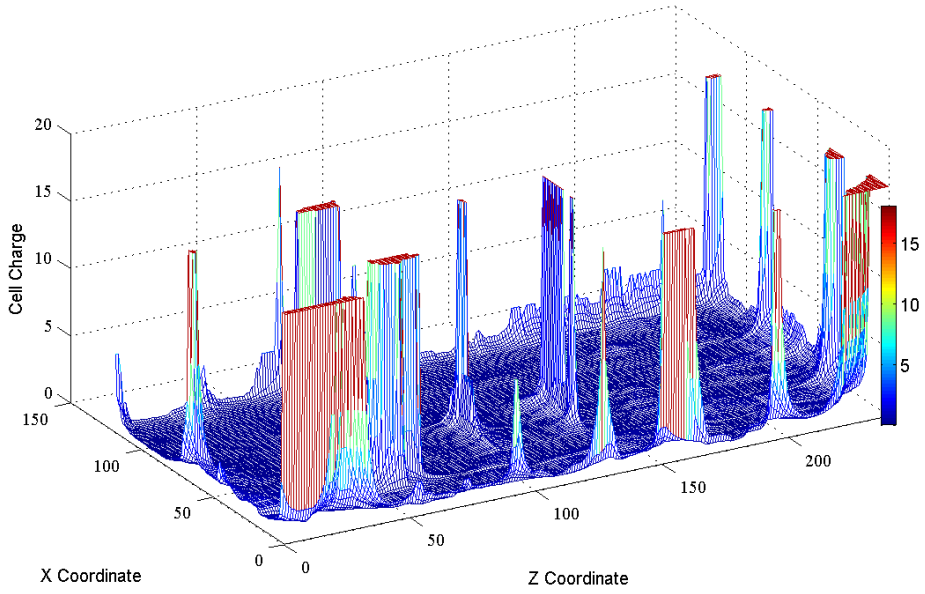


Two Robots

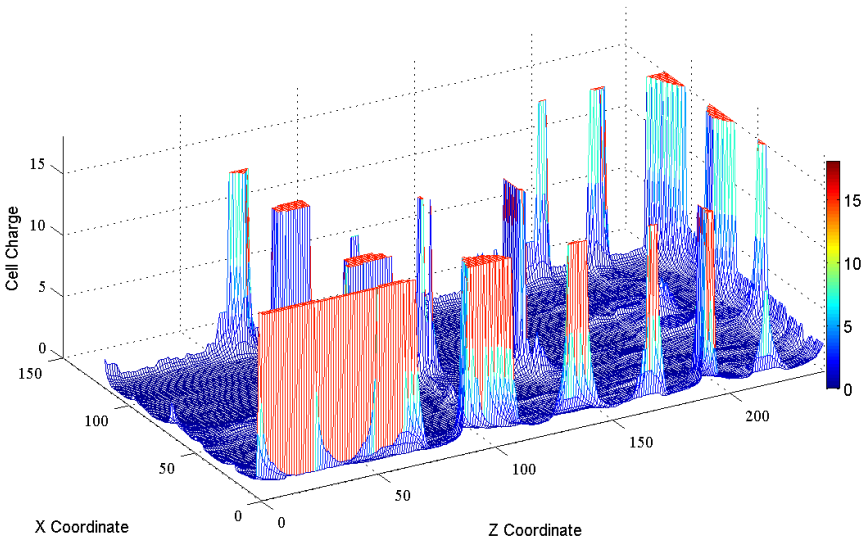


One Robot

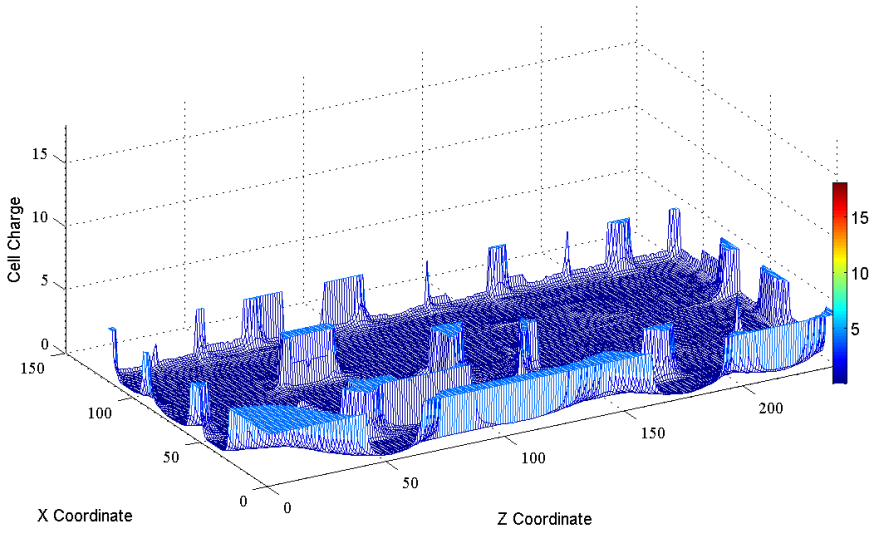
CELL CHARGE RESULTS



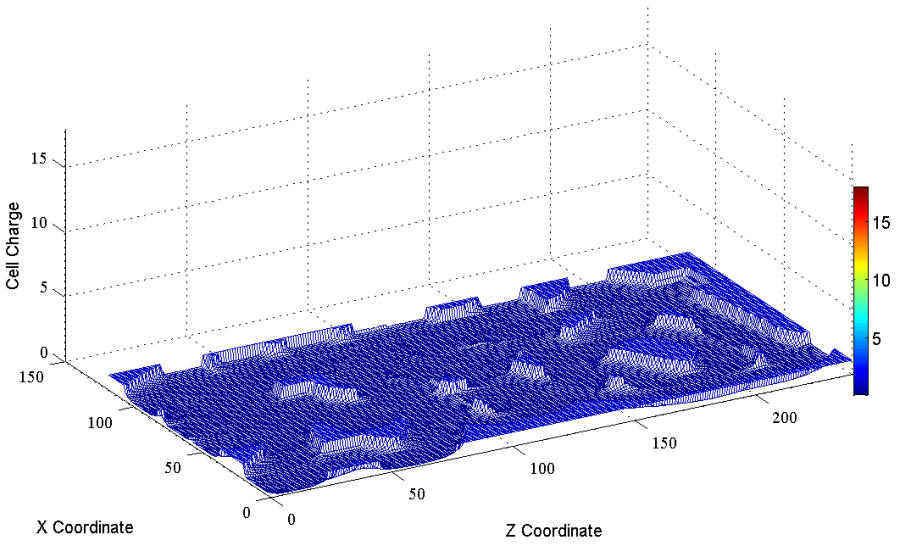
Four Robots



Three Robots

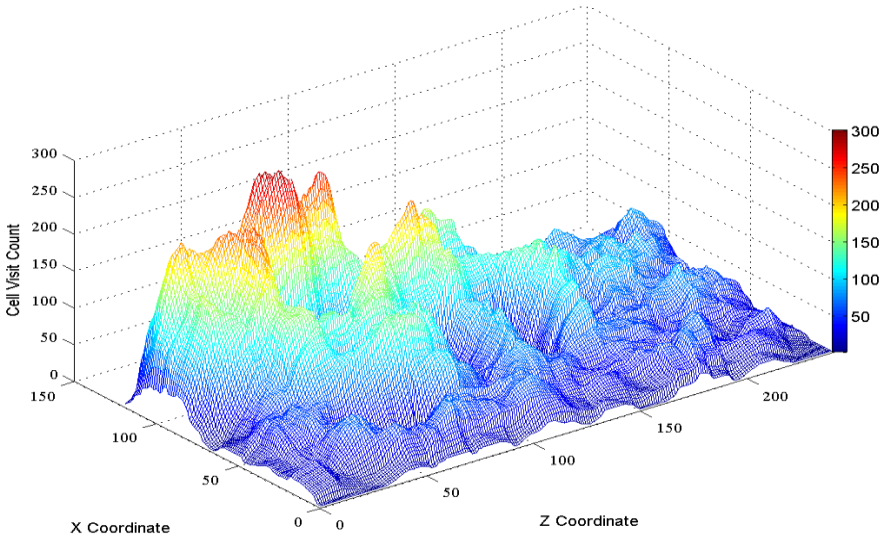


Two Robots

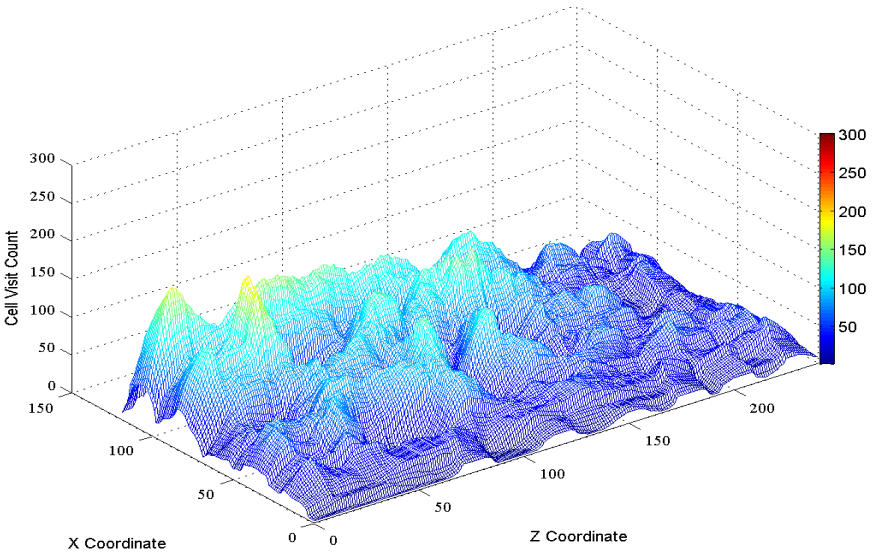


One Robot

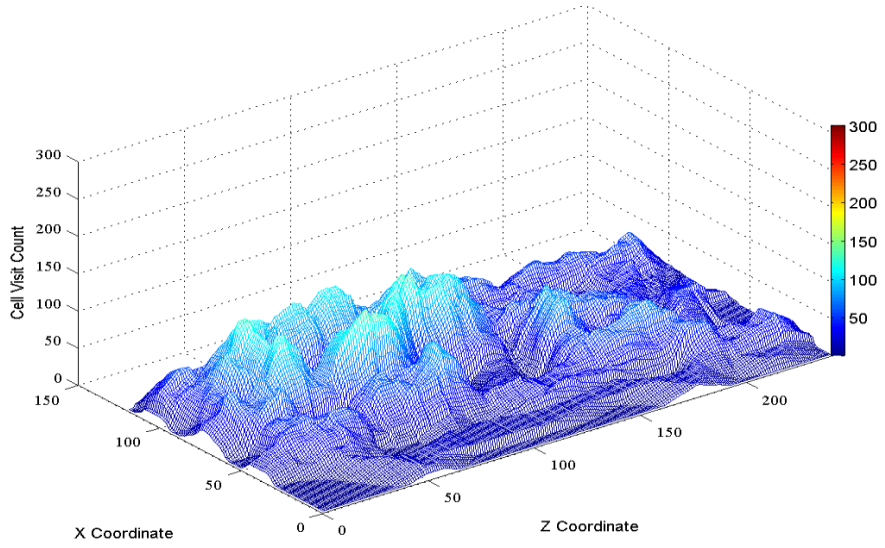
CELL VISIT COUNT RESULTS



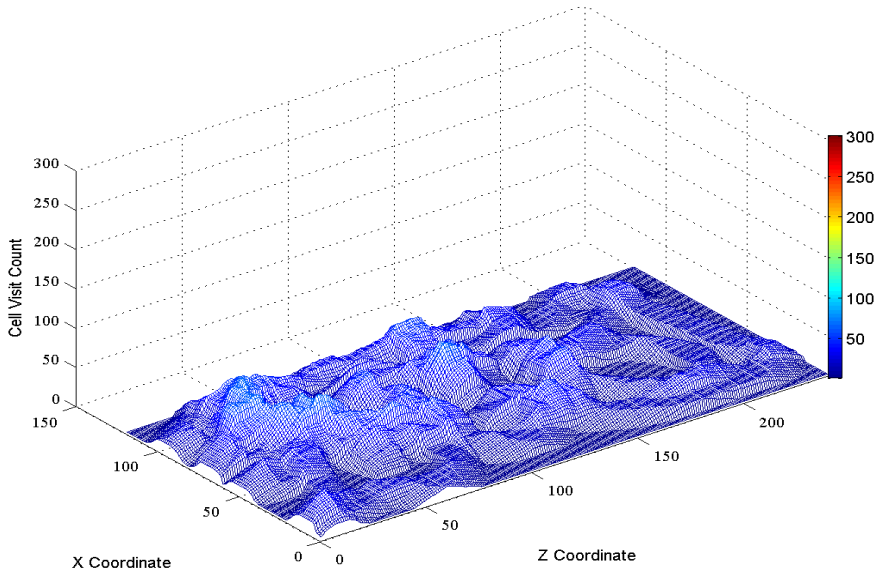
Four Robots



Three Robots



Two Robots



One Robot

Appendix B. State Space Software Code

```
% State Matrix and data manager for FOUR ROBOTS
% File:  state_five_bots.m
% Date:  29 Jan 11
% Description:  This routine manages the experiment state matrix and
%               associated data.
% Author:  Reed Young
% Modifications:

TIME_STEP = uint32(1);  % index for time, not absolute time.
MAX_TIME_STEP = uint32(4000); % maximum number of times steps in an
experiment
bot_loc = zeros(MAX_TIME_STEP,6,3);
    % bot_loc is a MAX_TIME_STEPX6X3 matrix that stores the history of
    % the six epucks' position/orientation.  The
    % first element is the time step (1 to MAX_TIME_STEP).  The second
    % element is the epuck ID number (1-6).  The
    % third element is the position/orientation as: x, z, heading.
Values
    % are single point precision.  This is a historic record only.
Values
    % collected here are not passed to individual controllers.
Distance in
    % cm; heading in radians.

bot_loc_cell = uint8(zeros(MAX_TIME_STEP,6,2));
    % bot_loc_cell is the x, z cell location of the bot, i.e.,
    % ceil(bot_loc).

occ_map_history = zeros(MAX_TIME_STEP,120,240);
    % occ_map is a MAX_TIME_STEPX120X240 matrix that stores the history
of
    % the occupancy map.  The first element is the time step.  The
second
    % and third elements refer to the 120X240 matrix that comprises the
    % occupancy map cell grid.  The cell is 1 cm^2, 120 elements in x
axis
    % (or vertical axis - keeping Webots' reference system) and 240
elements
    % in z axis or horizontal axis).  The values describe the
probability
    % that the cell is occupied.  Values vary from 0 to 1.  0
    % means that there is 100% probability the cell is unoccupied.  0.5
    % means there is complete uncertainty on occupancy.  255 means that
    % there is a 100% probability the cell is occupied.  The matrix is
    % initialized to 0.5 meaning complete uncertainty at inception.

occ_map = 0.5*ones(120,240);
```

```
occ_map_update = uint8(zeros(1,2));
TT = zeros(120,240);

%cell_charge = zeros(MAX_TIME_STEP,120,240);
    % cell_charge has same construct as occ_map.  The values  reflect
    % the strength of the negative or positive charge of the cell.

%cell_charge_update = zeros(120,240);

%cell_visit_count = uint32(ones(MAX_TIME_STEP,120,240));
    % cell_visit_count keeps track of the number of times a cell has
been
    % sensed.  Values range from  0 to 2^32 or >4B.

map_coverage_count = zeros(1,MAX_TIME_STEP);
    % map_coverage_count tallies the total number of cells that have a
    % occ_map value less than 0.4 or greater than 0.6.  These values
    % indicate if a cell is a known occupancy.
    % Values range from 0 to 2^16 or >65,000.

header = uint8(0);  % This is an integer that heads data transfer to
                    % specify how to handle the inbound data

% open up communications ports with inidividual controllers.  Note that
% "uX" here is outbound (vs being inbound on controller computers).
"vX"
% here is inbound (vs being outbound on controller computers).
Therefore,
% the u's talk, and the v's talk.

%     epuck     IP              outbound port (u)    inbound port (v)
% u1  1868      192.168.1.11    7011                 7021
% u2  2012      192.168.1.12    7012                 7022
% u3  2014      192.168.1.13    7013                 7023
% u4  2031      192.168.1.14    7014                 7024
% u5  2036      192.168.1.15    7015                 7025
% u6  2045      192.168.1.16    7016                 7026

if (~exist('u4','var')) % Create, define, and open sockets


    u1 = udp('192.168.1.11', 7011, 'LocalPort', 7011, 'Timeout', 100);
    % I.e., connect to 192.168.1.13:7013 through LocalPort:7013.  The
port
    % numbers are the same, but in fact refer to separate ports on the
    % respective computers.
%    u3.Terminator='';
    u1.inputbuffersize =5000000;
    u1.outputbuffersize = 5000000;
    u1.inputdatagrampacketsize = 60000;
```

```
    u1.outputdatagrampacketsize = 60000;
    u1.ReadAsyncMode = 'continuous';
    fopen(u1);

    v1 = udp('192.168.1.11', 7021, 'LocalPort', 7021, 'Timeout', 100);
%    v3.Terminator='';
    v1.outputbuffersize = 5000000;
    v1.inputbuffersize = 5000000;
    v1.inputdatagrampacketsize = 60000;
    v1.outputdatagrampacketsize = 60000;
    v1.ReadAsyncMode = 'continuous';
    fopen(v1);

    w1 = udp('192.168.1.11', 7031, 'LocalPort', 7031, 'Timeout', 100);
%    w3.Terminator='';
    w1.outputbuffersize = 5000000;
    w1.inputbuffersize = 5000000;
    w1.inputdatagrampacketsize = 60000;
    w1.outputdatagrampacketsize = 60000;
    w1.ReadAsyncMode = 'continuous';
    fopen(w1);

    x1 = udp('192.168.1.11', 7041, 'LocalPort', 7041, 'Timeout', 100);
%    w3.Terminator='';
    x1.outputbuffersize = 5000000;
    x1.inputbuffersize = 5000000;
    x1.inputdatagrampacketsize = 60000;
    x1.outputdatagrampacketsize = 60000;
    x1.ReadAsyncMode = 'continuous';
    fopen(x1);

    u3 = udp('192.168.1.13', 7013, 'LocalPort', 7013, 'Timeout', 100);
    % I.e., connect to 192.168.1.13:7013 through LocalPort:7013.  The
port
    % numbers are the same, but in fact refer to separate ports on the
    % respective computers.
%    u3.Terminator='';
    u3.inputbuffersize =5000000;
    u3.outputbuffersize = 5000000;
    u3.inputdatagrampacketsize = 60000;
    u3.outputdatagrampacketsize = 60000;
    u3.ReadAsyncMode = 'continuous';
    fopen(u3);

    v3 = udp('192.168.1.13', 7023, 'LocalPort', 7023, 'Timeout', 100);
%    v3.Terminator='';
    v3.outputbuffersize = 5000000;
    v3.inputbuffersize = 5000000;
    v3.inputdatagrampacketsize = 60000;
    v3.outputdatagrampacketsize = 60000;
    v3.ReadAsyncMode = 'continuous';
```

```
    fopen(v3);

    w3 = udp('192.168.1.13', 7033, 'LocalPort', 7033, 'Timeout', 100);
%    w3.Terminator='';
    w3.outputbuffersize = 5000000;
    w3.inputbuffersize = 5000000;
    w3.inputdatagrampacketsize = 60000;
    w3.outputdatagrampacketsize = 60000;
    w3.ReadAsyncMode = 'continuous';
    fopen(w3);

    x3 = udp('192.168.1.13', 7043, 'LocalPort', 7043, 'Timeout', 100);
%    w3.Terminator='';
    x3.outputbuffersize = 5000000;
    x3.inputbuffersize = 5000000;
    x3.inputdatagrampacketsize = 60000;
    x3.outputdatagrampacketsize = 60000;
    x3.ReadAsyncMode = 'continuous';
    fopen(x3);

    u4 = udp('192.168.1.14', 7014, 'LocalPort', 7014, 'Timeout', 100);
    % I.e., connect to 192.168.1.14:7014 through LocalPort:7014.  The
port
    % numbers are the same, but in fact refer to separate ports on the
    % respective computers.
%    u4.Terminator='';
    u4.inputbuffersize =5000000;
    u4.outputbuffersize = 5000000;
    u4.inputdatagrampacketsize = 60000;
    u4.outputdatagrampacketsize = 60000;
    u4.ReadAsyncMode = 'continuous';
    fopen(u4);

    v4 = udp('192.168.1.14', 7024, 'LocalPort', 7024, 'Timeout', 100);
%    v4.Terminator='';
    v4.outputbuffersize = 5000000;
    v4.inputbuffersize = 5000000;
    v4.inputdatagrampacketsize = 60000;
    v4.outputdatagrampacketsize = 60000;
    v4.ReadAsyncMode = 'continuous';
    fopen(v4);

    w4 = udp('192.168.1.14', 7034, 'LocalPort', 7034, 'Timeout', 100);
%    w4.Terminator='';
    w4.outputbuffersize = 5000000;
    w4.inputbuffersize = 5000000;
    w4.inputdatagrampacketsize = 60000;
    w4.outputdatagrampacketsize = 60000;
    w4.ReadAsyncMode = 'continuous';
    fopen(w4);
```

```
    x4 = udp('192.168.1.14', 7044, 'LocalPort', 7044, 'Timeout', 100);
%    w4.Terminator='';
    x4.outputbuffersize = 5000000;
    x4.inputbuffersize = 5000000;
    x4.inputdatagrampacketsize = 60000;
    x4.outputdatagrampacketsize = 60000;
    x4.ReadAsyncMode = 'continuous';
    fopen(x4);


    u5 = udp('192.168.1.15', 7015, 'LocalPort', 7015, 'Timeout', 100);
    % I.e., connect to 192.168.1.14:7014 through LocalPort:7014.  The
port
    % numbers are the same, but in fact refer to separate ports on the
    % respective computers.
%    u4.Terminator='';
    u5.inputbuffersize =5000000;
    u5.outputbuffersize = 5000000;
    u5.inputdatagrampacketsize = 60000;
    u5.outputdatagrampacketsize = 60000;
    u5.ReadAsyncMode = 'continuous';
    fopen(u5);

    v5 = udp('192.168.1.15', 7025, 'LocalPort', 7025, 'Timeout', 100);
%    v4.Terminator='';
    v5.outputbuffersize = 5000000;
    v5.inputbuffersize = 5000000;
    v5.inputdatagrampacketsize = 60000;
    v5.outputdatagrampacketsize = 60000;
    v5.ReadAsyncMode = 'continuous';
    fopen(v5);

    w5 = udp('192.168.1.15', 7035, 'LocalPort', 7035, 'Timeout', 100);
%    w4.Terminator='';
    w5.outputbuffersize = 5000000;
    w5.inputbuffersize = 5000000;
    w5.inputdatagrampacketsize = 60000;
    w5.outputdatagrampacketsize = 60000;
    w5.ReadAsyncMode = 'continuous';
    fopen(w5);

    x5 = udp('192.168.1.15', 7045, 'LocalPort', 7045, 'Timeout', 100);
%    w4.Terminator='';
    x5.outputbuffersize = 5000000;
    x5.inputbuffersize = 5000000;
    x5.inputdatagrampacketsize = 60000;
    x5.outputdatagrampacketsize = 60000;
    x5.ReadAsyncMode = 'continuous';
    fopen(x5);

    u6 = udp('192.168.1.16', 7016, 'LocalPort', 7016, 'Timeout', 100);
```

```
    % I.e., connect to 192.168.1.14:7014 through LocalPort:7014.  The
port
    % numbers are the same, but in fact refer to separate ports on the
    % respective computers.
%    u4.Terminator='';
    u6.inputbuffersize =5000000;
    u6.outputbuffersize = 5000000;
    u6.inputdatagrampacketsize = 60000;
    u6.outputdatagrampacketsize = 60000;
    u6.ReadAsyncMode = 'continuous';
    fopen(u6);

    v6 = udp('192.168.1.16', 7026, 'LocalPort', 7026, 'Timeout', 100);
%    v4.Terminator='';
    v6.outputbuffersize = 5000000;
    v6.inputbuffersize = 5000000;
    v6.inputdatagrampacketsize = 60000;
    v6.outputdatagrampacketsize = 60000;
    v6.ReadAsyncMode = 'continuous';
    fopen(v6);

    w6 = udp('192.168.1.16', 7036, 'LocalPort', 7036, 'Timeout', 100);
%    w4.Terminator='';
    w6.outputbuffersize = 5000000;
    w6.inputbuffersize = 5000000;
    w6.inputdatagrampacketsize = 60000;
    w6.outputdatagrampacketsize = 60000;
    w6.ReadAsyncMode = 'continuous';
    fopen(w6);

    x6 = udp('192.168.1.16', 7046, 'LocalPort', 7046, 'Timeout', 100);
%    w4.Terminator='';
    x6.outputbuffersize = 5000000;
    x6.inputbuffersize = 5000000;
    x6.inputdatagrampacketsize = 60000;
    x6.outputdatagrampacketsize = 60000;
    x6.ReadAsyncMode = 'continuous';
    fopen(x6);


else % Effectively clears buffers

    fclose(u1);
    fopen(u1);
    fclose(v1);
    fopen(v1);
    fclose(w1);
    fopen(w1);
    fclose(x1);
    fopen(x1);
```

```
    fclose(u3);
    fopen(u3);
    fclose(v3);
    fopen(v3);
    fclose(w3);
    fopen(w3);
    fclose(x3);
    fopen(x3);

    fclose(u4);
    fopen(u4);
    fclose(v4);
    fopen(v4);
    fclose(w4);
    fopen(w4);
    fclose(x4);
    fopen(x4);

    fclose(u5);
    fopen(u5);
    fclose(v5);
    fopen(v5);
    fclose(w5);
    fopen(w5);
    fclose(x5);
    fopen(x5);

    fclose(u6);
    fopen(u6);
    fclose(v6);
    fopen(v6);
    fclose(w6);
    fopen(w6);
    fclose(x6);
    fopen(x6);

end

%fprintf('TIME_STEP A %6i\n', TIME_STEP);

% clear zeros from VRPN buffer by calling MEX Bot_Tracker and pausing 1
sec
Bot_Tracker1868(0);
%Bot_Tracker2012(0);
Bot_Tracker2014(0);
Bot_Tracker2031(0);
Bot_Tracker2036(0);
Bot_Tracker2045(0);

while((bot_loc(TIME_STEP,1,1) == 0) ||...
      (bot_loc(TIME_STEP,3,1) == 0) ||...
```

```
      (bot_loc(TIME_STEP,4,1) == 0) ||...
      (bot_loc(TIME_STEP,5,1) == 0) ||...
      (bot_loc(TIME_STEP,6,1) == 0))
    bot_loc(TIME_STEP,1,:) = Bot_Tracker1868(1);
    bot_loc(TIME_STEP,3,:) = Bot_Tracker2014(1);
    bot_loc(TIME_STEP,4,:) = Bot_Tracker2031(1);
    bot_loc(TIME_STEP,5,:) = Bot_Tracker2036(1);
    bot_loc(TIME_STEP,6,:) = Bot_Tracker2045(1);
    pause(0.0001);
end

fprintf('START CONTROLLERS\n');

% Stall until the controller computers start
while((~v1.bytesavailable) ||...
      (~v3.bytesavailable) ||...
      (~v4.bytesavailable) ||...
      (~v5.bytesavailable) ||...
      (~v6.bytesavailable))
    pause(.001)
end
% MAIN LOOP
%================================================================

tic;

for TIME_STEP = 1:MAX_TIME_STEP

  pause(.05) % adjust to establish ~10hz timesteps

  % Call the surrogate localization function and database
  bot_loc(TIME_STEP,1,:) = Bot_Tracker1868(TIME_STEP);
  %bot_loc(TIME_STEP,2,:) = Bot_Tracker2012(TIME_STEP);
  bot_loc(TIME_STEP,3,:) = Bot_Tracker2014(TIME_STEP);
  bot_loc(TIME_STEP,4,:) = Bot_Tracker2031(TIME_STEP);
  bot_loc(TIME_STEP,5,:) = Bot_Tracker2036(TIME_STEP);
  bot_loc(TIME_STEP,6,:) = Bot_Tracker2045(TIME_STEP);

fprintf('TIME STEP %6i\n', TIME_STEP);

  % Set the bot_loc_cell based on bot_loc
  bot_loc_cell(TIME_STEP,1,:) = uint8(ceil(bot_loc(TIME_STEP,1,1:2)));
  %bot_loc_cell(TIME_STEP,2,:) = uint8(ceil(bot_loc(TIME_STEP,2,1:2)));
  bot_loc_cell(TIME_STEP,3,:) = uint8(ceil(bot_loc(TIME_STEP,3,1:2)));
  bot_loc_cell(TIME_STEP,4,:) = uint8(ceil(bot_loc(TIME_STEP,4,1:2)));
  bot_loc_cell(TIME_STEP,5,:) = uint8(ceil(bot_loc(TIME_STEP,5,1:2)));
  bot_loc_cell(TIME_STEP,6,:) = uint8(ceil(bot_loc(TIME_STEP,6,1:2)));

  % Decrease the occupation probability based on physical location of
  % robot, i.e., occ_map values decrease knowing that bot physically
```

```
  % occupies (including prox sensing range) the 16X16 set of cells
  % centered on the bot's centroid location.  Amount of decrease is an
  % arbitrary 0.05.

    % This series of if/else/end handles cases near border of stage.

  for i = [1,3,4,5,6]

      if (bot_loc_cell(TIME_STEP,i,1) <= 6) % case: lower x limits
          limit(1) = bot_loc_cell(TIME_STEP,i,1)-1;
          limit(2) = 6;
      elseif (bot_loc_cell(TIME_STEP,i,1) >= 114) % case: upper x
limits
          limit(1) = 6;
          limit(2) = 120-bot_loc_cell(TIME_STEP,i,1);
      else
          limit(1) = 6;
          limit(2) = 6;
      end

      if (bot_loc_cell(TIME_STEP,i,2) <= 6) % case: lower y limits
          limit(3) = bot_loc_cell(TIME_STEP,i,2)-1;
          limit(4) = 6;
      elseif (bot_loc_cell(TIME_STEP,i,2) >= 234) % case: upper y
limits
          limit(3) = 6;
          limit(4) = 240-bot_loc_cell(TIME_STEP,i,2);
      else
          limit(3) = 6;
          limit(4) = 6;
      end

      occ_map((bot_loc_cell(TIME_STEP,i,1)-limit(1)):...
                (bot_loc_cell(TIME_STEP,i,1)+limit(2))),...
              ((bot_loc_cell(TIME_STEP,i,2)-limit(3)):...
                (bot_loc_cell(TIME_STEP,i,2)+limit(4)))) = ...
        occ_map((bot_loc_cell(TIME_STEP,i,1)-limit(1)):...
                (bot_loc_cell(TIME_STEP,i,1)+limit(2))),...
              ((bot_loc_cell(TIME_STEP,i,2)-limit(3)):...
               (bot_loc_cell(TIME_STEP,i,2)+limit(4)))) - .01;

      % Set minimum values at zero.
      occ_map((bot_loc_cell(TIME_STEP,i,1)-limit(1)):...
              (bot_loc_cell(TIME_STEP,i,1)+limit(2))),...
            ((bot_loc_cell(TIME_STEP,i,2)-limit(3)):...
              (bot_loc_cell(TIME_STEP,i,2)+limit(4)))) = ...
        max(occ_map((bot_loc_cell(TIME_STEP,i,1)-limit(1)):...
                    (bot_loc_cell(TIME_STEP,i,1)+limit(2))),...
                    ((bot_loc_cell(TIME_STEP,i,2)-limit(3)):...
                    (bot_loc_cell(TIME_STEP,i,2)+limit(4)))),0);
```

```
end

% Test for data call from controller1868
if (v1.bytesavailable)
      header = fread(v1,1); % read header integer
      if (header == 1) % send current occ_map
          fprintf('SENDING OCC_MAP to 1868\n');
          for i = 1:120
              fwrite(u1, occ_map(i,:),'double');
          end
      end
end

% Test for data call from controller2014
if (v3.bytesavailable)
      header = fread(v3,1); % read header integer
      if (header == 1) % send current occ_map
          fprintf('SENDING OCC_MAP to 2014\n');
          for i = 1:120
              fwrite(u3, occ_map(i,:),'double');
          end
      end
end

% Test for data call from controller2031
if (v4.bytesavailable)
      header = fread(v4,1); % read header integer
      if (header == 1) % send current occ_map
          fprintf('SENDING OCC_MAP to 2031\n');
          for i = 1:120
              fwrite(u4, occ_map(i,:),'double');
          end
      end
end

% Test for data call from controller2036
if (v5.bytesavailable)
      header = fread(v5,1); % read header integer
      if (header == 1) % send current occ_map
          fprintf('SENDING OCC_MAP to 2036\n');
          for i = 1:120
              fwrite(u5, occ_map(i,:),'double');
          end
      end
end

% Test for data call from controller2045
if (v6.bytesavailable)
      header = fread(v6,1); % read header integer
      if (header == 1) % send current occ_map
          fprintf('SENDING OCC_MAP to 2045\n');
```

```
            for i = 1:120
                fwrite(u6, occ_map(i,:),'double');
            end
        end
    end

    % Read 1868 proximity sensor buffer and update occ_map
    while(w1.bytesavailable)

      fprintf('PROCESSING 1868 PROX DATA\n');
      occ_map_update = fread(w1, 2, 'uint8');

      % Set max value at 1.0
      if (occ_map(occ_map_update(1),occ_map_update(2)) <= 0.9)

          occ_map(occ_map_update(1),occ_map_update(2)) =...
              occ_map(occ_map_update(1),occ_map_update(2)) + 0.1;
%fprintf('occ_map %3.3f %3.3f\n\n',
occ_map(occ_map_update(1),occ_map_update(2)));
      end

    end

    % Read 2014 proximity sensor buffer and update occ_map
    while(w3.bytesavailable)

      fprintf('PROCESSING 2014 PROX DATA\n');
      occ_map_update = fread(w3, 2, 'uint8');

      % Set max value at 1.0
      if (occ_map(occ_map_update(1),occ_map_update(2)) <= 0.9)

          occ_map(occ_map_update(1),occ_map_update(2)) =...
              occ_map(occ_map_update(1),occ_map_update(2)) + 0.1;
%fprintf('occ_map %3.3f %3.3f\n\n',
occ_map(occ_map_update(1),occ_map_update(2)));
      end

    end

% Read 2031 proximity sensor buffer and update occ_map
    while(w4.bytesavailable)

      fprintf('PROCESSING 2031 PROX DATA\n');
      occ_map_update = fread(w4, 2, 'uint8');

      % Set max value at 1.0
      if (occ_map(occ_map_update(1),occ_map_update(2)) <= 0.9)
```

```
        occ_map(occ_map_update(1),occ_map_update(2)) =...
            occ_map(occ_map_update(1),occ_map_update(2)) + 0.1;
%fprintf('occ_map %3.3f %3.3f\n\n',
occ_map(occ_map_update(1),occ_map_update(2)));
    end

  end

  % Read 2036 proximity sensor buffer and update occ_map
  while(w5.bytesavailable)

    fprintf('PROCESSING 2036 PROX DATA\n');
    occ_map_update = fread(w5, 2, 'uint8');

    % Set max value at 1.0
    if (occ_map(occ_map_update(1),occ_map_update(2)) <= 0.9)

        occ_map(occ_map_update(1),occ_map_update(2)) =...
            occ_map(occ_map_update(1),occ_map_update(2)) + 0.1;
%fprintf('occ_map %3.3f %3.3f\n\n',
occ_map(occ_map_update(1),occ_map_update(2)));
    end

  end

  % Read 2045 proximity sensor buffer and update occ_map
  while(w6.bytesavailable)

    fprintf('PROCESSING 2045 PROX DATA\n');
    occ_map_update = fread(w6, 2, 'uint8');

    % Set max value at 1.0
    if (occ_map(occ_map_update(1),occ_map_update(2)) <= 0.9)

        occ_map(occ_map_update(1),occ_map_update(2)) =...
            occ_map(occ_map_update(1),occ_map_update(2)) + 0.1;
%fprintf('occ_map %3.3f %3.3f\n\n',
occ_map(occ_map_update(1),occ_map_update(2)));
    end

  end

  occ_map_history(TIME_STEP,:,:) = occ_map;

  % Test for and fill obstacles every 10 timesteps
  if ((double(TIME_STEP)/10) == TIME_STEP/10)

     prox_detect = im2bw(occ_map,0.7); % threshold image at 0.7, i.e.,
proximity sensor hits
```

```
%L1=bwlabel(prox_detect,8);

      BW=im2bw(occ_map,0.49); % threshold image at 0.49, i.e.,
unexplored plus occupied
      BW=bwareaopen(BW,125); % eliminate white blobs <125 cells, i.e.,
no obstacles are <125 cells large
      BW=imclearborder(BW,8); % eliminate blobs touching border;
assumes no obstacles at border

      [L,num]=bwlabel(BW,8);

%stat=regionprops(BW, 'BoundingBox', 'Centroid', 'Area');
%centroids=cat(1,stat.Centroid);


      for i=1:num
          count=0;
          [r,c]=find(L==i);
          rc=[r c];
          if (size(rc,1)<800) % exclude regions greater than 800 cells,
i.e., large unexplored space
              for j=1:size(rc,1)
                 if (prox_detect(rc(j,1),rc(j,2)))
                     count=count+1;
                 end
              end

              if (count>0) % i.e., if there is any proximity sensor hit
                  fprintf('Obstacle Filled....\n');
                  for j=1:size(rc,1)
                      occ_map(rc(j,1),rc(j,2))=1; % set occ_map to 1
                  end
              end
          end
      end
  end

  % Test to see if >75% of the occupancy map has been explored defined
as
  % occ_map>0.6 or occ_map<0.4.
  TT = (occ_map>0.6 | occ_map<0.4);
  map_coverage_count(TIME_STEP) = sum(sum(TT));
%  fprintf('map coverage count: %6.2f\n',
map_coverage_count(TIME_STEP));

  %if (map_coverage_count(TIME_STEP)>21600) % Or 75% of 120X240
  %if (map_coverage_count(TIME_STEP)>23040) % Or 80% of 120X240
  %if (map_coverage_count(TIME_STEP)>24480) % Or 85% of 120X240
  if (map_coverage_count(TIME_STEP)>25920) % Or 90% of 120X240
      % Note: each obstacle is 11X17 cm, so 8 obstacles = 1496 cm^2 or
```

```
      % 5.19% of 120X240cm stage
    total_time = toc;
    fprintf('Total time for run: %6.2f seconds\n', total_time);

    fwrite(x1, 1,'double'); % send end of experiment trigger to bot
    fwrite(x3, 1,'double'); % send end of experiment trigger to bot
    fwrite(x4, 1,'double'); % send end of experiment trigger to bot
    fwrite(x5, 1,'double'); % send end of experiment trigger to bot
    fwrite(x6, 1,'double'); % send end of experiment trigger to bot


    % set location on the screen where the figure will print
    figure('Position',[2000 10 750 750],'Units','inches')
    %figure('Position',[24 3 8 8],'Units','inches')

    subplot(3,1,1);
    hold on; % retains that track info on the trajectory map
    % plot entire bot track for each bot
    plot(bot_loc(1:TIME_STEP,1,2),bot_loc(1:TIME_STEP,1,1),'m');
    plot(bot_loc(1:TIME_STEP,3,2),bot_loc(1:TIME_STEP,3,1),'c');
    plot(bot_loc(1:TIME_STEP,4,2),bot_loc(1:TIME_STEP,4,1),'r');
    plot(bot_loc(1:TIME_STEP,5,2),bot_loc(1:TIME_STEP,5,1),'g');
    plot(bot_loc(1:TIME_STEP,6,2),bot_loc(1:TIME_STEP,6,1),'b');

    plot(bot_loc(TIME_STEP,1,2),bot_loc(TIME_STEP,1,1),'mo'); % plot
current bot position
    plot(bot_loc(TIME_STEP,3,2),bot_loc(TIME_STEP,3,1),'co');
    plot(bot_loc(TIME_STEP,4,2),bot_loc(TIME_STEP,4,1),'ro');
    plot(bot_loc(TIME_STEP,5,2),bot_loc(TIME_STEP,5,1),'go');
    plot(bot_loc(TIME_STEP,6,2),bot_loc(TIME_STEP,6,1),'bo');

    axis on;
    axis equal; % set the units on both axes to the same value
    axis([0 240 0 120]);  % note that x axis in Webots is y axis in
MATLAB, and z in Webots is x in MATLAB
    title('EPUCK Trajectories');
    xlabel('z axis');
    ylabel('x axis');
    hold off;

    subplot(3,1,2);
    imshow(occ_map);
    set(gca,'YDir','normal'); %reverse y axis so low values are at
origin
    axis on;
    axis equal; % set the units on both axes to the same value
    axis([0 240 0 120]);  % note that x axis in Webots is y axis in
MATLAB, and z in Webots is x in MATLAB
    title('Supervisor Occupancy Map');
    xlabel('Webots z axis');
    ylabel('Webots x axis');
```

```
    subplot(3,1,3);
    plot(map_coverage_count(1:TIME_STEP),'r');
    %axis on;
    %axis equal; % set the units on both axes to the same value
    axis([0 TIME_STEP 0 28800]);  % note that x axis in Webots is y
axis in MATLAB, and z in Webots is x in MATLAB
    title('Map Coverage Count');
    xlabel('z axis');
    ylabel('x axis');


    drawnow;

    break;
  end

  % Print a status screen every 1000 timesteps
  if ((double(TIME_STEP)/1000) == TIME_STEP/1000)

    % set location on the screen where the figure will print
    figure('Position',[2000 10 750 750],'Units','inches')
    %figure('Position',[24 3 8 8],'Units','inches')

    subplot(3,1,1);
    hold on; % retains that track info on the trajectory map
    % plot entire bot track for each bot
    plot(bot_loc(1:TIME_STEP,1,2),bot_loc(1:TIME_STEP,1,1),'m');
    plot(bot_loc(1:TIME_STEP,3,2),bot_loc(1:TIME_STEP,3,1),'c');
    plot(bot_loc(1:TIME_STEP,4,2),bot_loc(1:TIME_STEP,4,1),'r');
    plot(bot_loc(1:TIME_STEP,5,2),bot_loc(1:TIME_STEP,5,1),'g');
    plot(bot_loc(1:TIME_STEP,6,2),bot_loc(1:TIME_STEP,6,1),'b');

    plot(bot_loc(TIME_STEP,1,2),bot_loc(TIME_STEP,1,1),'mo'); % plot
current bot position
    plot(bot_loc(TIME_STEP,3,2),bot_loc(TIME_STEP,3,1),'co');
    plot(bot_loc(TIME_STEP,4,2),bot_loc(TIME_STEP,4,1),'ro');
    plot(bot_loc(TIME_STEP,5,2),bot_loc(TIME_STEP,5,1),'go');
    plot(bot_loc(TIME_STEP,6,2),bot_loc(TIME_STEP,6,1),'bo');

    axis on;
    axis equal; % set the units on both axes to the same value
    axis([0 240 0 120]);  % note that x axis in Webots is y axis in
MATLAB, and z in Webots is x in MATLAB
    title('EPUCK Trajectories');
    xlabel('z axis');
    ylabel('x axis');
    hold off;

    subplot(3,1,2);
```

```
    imshow(occ_map);
    set(gca,'YDir','normal'); %reverse y axis so low values are at
origin
    axis on;
    axis equal; % set the units on both axes to the same value
    axis([0 240 0 120]);  % note that x axis in Webots is y axis in
MATLAB, and z in Webots is x in MATLAB
    title('Supervisor Occupancy Map');
    xlabel('Webots z axis');
    ylabel('Webots x axis');

    subplot(3,1,3);
    plot(map_coverage_count(1:TIME_STEP),'r');
    %axis on;
    %axis equal; % set the units on both axes to the same value
    axis([0 TIME_STEP 0 28800]);  % note that x axis in Webots is y
axis in MATLAB, and z in Webots is x in MATLAB
    title('Map Coverage Count');
    xlabel('z axis');
    ylabel('x axis');


%     subplot(6,2,7);
%     imshow(cell_charge);
%     set(gca,'YDir','normal'); %reverse y axis so low values are at
origin
%     axis on;
%     axis equal; % set the units on both axes to the same value
%     axis([0 240 0 120]);  % note that x axis in Webots is y axis in
MATLAB, and z in Webots is x in MATLAB
%     title('Cell Charge');
%     xlabel('Webots z axis');
%     ylabel('Webots x axis');

    drawnow;

  end

end

fprintf('END OF RUN\n');
%save(ExperimentRun1, '-v7.3');
```

Appendix C. Agent Controller Software Code

```
%%%%%%%%%%%%

while(x.bytesavailable == 0) % x is trigger from state that run is
finished

  % Reset no_obstacle.
    no_obstacle = 0;

    % Update odometer.  epuck can only deliver a 16-bit number, so must
calc
    % at each timestep to exceed 32768.
    temp_odometer_new=abs(get(ePic,'pos')); %obtain current odometer
reading
    odometer(2,:)=odometer(2,:)+abs(temp_odometer_new-
temp_odometer_old);
    temp_odometer_old = temp_odometer_new;

  % Loop within local collision avoidance in the presence of an
obstacle.
    while(no_obstacle<8)

        no_obstacle = 0; % reset trigger

        % Reset the wheel speed
        wheel_speed = int16([200,200]);

        % Obtain the location/orientation of the bot using the MEX
function
        bot_loc_2031 = Bot_Tracker2031controller(1);
%fprintf('bot orientation: %6.2f\n', (bot_loc(3)*180/3.14159));
        % Calculate the cell location of the bot.  "ceil" takes
fraction to next
        % higher integer value.
        bot_loc_cell = uint8(ceil(bot_loc_2031(1:2)));

        % Get the proximity sensor values and put in 1x8 vector "prox"
        ePic=update(ePic);
        [prox,up]=get(ePic,'proxi');

        while(~all(prox<3000) || (max(prox)<=0)) % if any prox sensors
read > 2000, i.e., comms error
            fprintf('ERROR IN PROX READINGS\n');
            flush(ePic);
            ePic = updateDef(ePic,'proxi',1);
            ePic=set(ePic,'speed',[-50,50]);
            ePic=set(ePic,'camMode',1,'camSize',[40,40],'camZoom',8);
            ePic=update(ePic);
            [prox,up]=get(ePic,'proxi');  prox
```

```
            flush(ePic);
        end

        % Test for obstacle detection with proximity sensors and
        % perform local collision avoidance using Braitenberg if an
        % obstacle was detected.
        for i = 1:8
            if (prox(i) > THRESHOLD(i))
 fprintf('i PROX: %6i %6.2f\n', i, prox(i));

                x_obstacle = bot_loc_2031(1) + (OBSTACLE_OFFSET * ...
                    cos(bot_loc_2031(3) + ANGLE_OFFSET(i)));
                z_obstacle = bot_loc_2031(2) + (OBSTACLE_OFFSET * ...
                    sin(bot_loc_2031(3) + ANGLE_OFFSET(i)));
                if (x_obstacle>0) && (x_obstacle<=120) && ...
                        (z_obstacle>0) && (z_obstacle<=240)

                    occ_map_update(1) = uint8(ceil(x_obstacle));
                    occ_map_update(2) = uint8(ceil(z_obstacle));

                    % Create update and send to port
                    fwrite(w,occ_map_update,'uint8');
                    prox_count = prox_count + 1; % increment prox count
                else
                    fprintf('DROPPED OCC_MAP_UPDATE PACKET\n');
                end
                wheel_speed(1) = wheel_speed(1) +
int16(prox(i)*WEIGHTS(1,i));
                wheel_speed(2) = wheel_speed(2) +
int16(prox(i)*WEIGHTS(2,i));

            else
                no_obstacle = no_obstacle + 1;
            end
        end

        if (no_obstacle<8)
            wheel_speed = int16(min(wheel_speed, 200));
            % Limit wheel_speed to acceptable maximum of 200 each.
            wheel_speed = int16(max(wheel_speed, -200));
            % Limit wheel_speed to acceptable minimum of -200 each.

            % Send calcualted wheel speeds to epuck.
 fprintf('WHEEL SPEED1 %6i %6i\n', wheel_speed);
            ePic=set(ePic,'speed',wheel_speed);
            ePic=update(ePic);
        end

    end
```

```
    % Get an image and process for target recognition

fprintf('IMAGE COLLECTION:   ');
    flush(ePic); % Flush the MATLAB COM port
tic
    [ePic, mode, sizexy]=updateImage(ePic); % time consumer...
toc
    % updateImage turns off the LEDs for accurate imaging; therefore,
turn
    % back on here
%     ePic=set(ePic,'ledON',0);
%     ePic=set(ePic,'ledON',1);
%     ePic=set(ePic,'ledON',2);
%     ePic=set(ePic,'ledON',6);
%     ePic=set(ePic,'ledON',7);
%     ePic=update(ePic);

    [imagedata,up]=get(ePic,'image');
    flush(ePic);

    subplot(1,3,1);
    imshow(imrotate(imagedata,-90)); % Display the image on the screen

    I=rgb2gray(imagedata); % create grayscale image from RGB
    BW=im2bw(I,0.15); % create B&W image from grayscale, thresholded
                      % at 0.15 to exclude all but 'black' objects
    subplot(1,3,2);
    imshow(imrotate(BW,-90));
    BW=bwareaopen(BW,4); % eliminate white blobs < 4 pixels, i.e.,
                         % noise inside obstacles
    BW=~BW; % invert image so that white represents objects
    BW=bwareaopen(BW,100); % eliminate white blobs < 100 pixels, i.e.,
                           % too-distant obstacles or noise

    % eliminate structures at border of image since they are obscured
    BW=imclearborder(BW,8);

    % obtain the centroid coordinates and dimensions of the object(s)
    stats=regionprops(BW,'Centroid','BoundingBox');

    % Determine the number of objects in the scene, ie., the size of
the
    % first dimension of stats
    num_obs=size(stats,1);

    if(num_obs>0)
        subplot(1,3,3);
        imshow(imrotate(BW,-90)); % Disply the B&W image on screen
```

```
        % Obtain the location/orientation of the bot using the MEX
function
        bot_loc_2031 = Bot_Tracker2031controller(1);

        for i=1:num_obs

            obs_width(i)=stats(i).BoundingBox(4);
            if(obs_width(i)>10) % widths less than 11 yield inaccurate
distances

                % calc the obj position and pass back to state
                dist_to_obs(i)=0.1133*obs_width(i)^2-
7.9037*obs_width(i)+177.9;

                perp_dist(i)=(20-
stats(i).Centroid(2))*(0.0113*dist_to_obs(i)+0.0157);
                rel_dir_to_obs(i)=atan(perp_dist(i)/dist_to_obs(i));

                % add in bot and target radii to obtain centroid-to-
centroid distance
                dist_to_obs(i)=dist_to_obs(i)+6;

                x_obstacle = ceil(bot_loc_2031(1) + (dist_to_obs(i) *
...
                                  cos(bot_loc_2031(3) +
rel_dir_to_obs(i))));
                z_obstacle = ceil(bot_loc_2031(2) + (dist_to_obs(i) *
...
                                  sin(bot_loc_2031(3) +
rel_dir_to_obs(i))));

                for j=(x_obstacle-6):(x_obstacle+6)
                    for k=(z_obstacle-6):(z_obstacle+6)

                        if (j>0) && (j<=120) && ...
                            (k>0) && (k<=240)

                            occ_map_update(1) = uint8(ceil(j));
                            occ_map_update(2) = uint8(ceil(k));

                            % Create update and send to port
                            fwrite(w,occ_map_update,'uint8');
                            image_count = image_count + 1;
                        else
                            fprintf('EDGE - DROPPED OCC_MAP_UPDATE
PACKET\n');
                        end
                    end
                end
            end
```

```
        end
    end
    drawnow;

    % If a new state matrix is available at the port, retrieve into
'occ_map'
    % and calculate a new theta based on potential-based path planning.
    if (u.bytesavailable >= 230400)
 fprintf('RECEIVING OCC_MAP:   ');
        % Retrieve new state matrix.
        for i = 1:120
            occ_map(i,:) = fread(u, 240, 'double');
        end

        % Call for a new occ_map for next path planning loop.
        fwrite(v, 1, 'uint8');

        % Calculate cell_charge as a function of occ_map whereby
        % cell_charge is highest for unknown occupancies (occ_map =
        % 0.5) and lowest for either known occupancy (occupied or
        % unoccupied; occ_map = 1 or = 0, respectively)
        cell_charge = MAX_CELL_CHARGE * ...
          (((occ_map <= 0.5) .* occ_map) +...
            (occ_map > 0.5) .* (1.0 - occ_map));

        radius_x_cell = ((1:120)-bot_loc_2031(1)-0.5)'*ones(1,240);

        % This is the x distance from the bot_loc to each individual
        % cell in the occ_map.  "0.5" adjusts distance so that the
        % radius is measured to the center of the cell.

        radius_z_cell = ones(120,1)*(1:240)-bot_loc_2031(2)-0.5;

        % Similarly, this is the z distance.

        radius_cell_squared = max(radius_x_cell.^2 +...
                                  radius_z_cell.^2, 1.0);
        % Take the square of the sum.  Set min value to 0.001 to
        % prevent division by zero subsequently.

        radius_cell = radius_cell_squared.^0.5;

        Fx_cell_matrix = ROBOT_CHARGE.*cell_charge.*...
            radius_x_cell./radius_cell./radius_cell_squared;
            %(radius_x_cell./radius_cell)./radius_cell;
            %# set max value to 2 to truncate large force contributions
            %# at radius < 8
        Fx_cell_matrix = min(Fx_cell_matrix, 2);
        Fx_cell_matrix = max(Fx_cell_matrix, -2);
```

```
Fz_cell_matrix = ROBOT_CHARGE.*cell_charge.*...
    radius_z_cell./radius_cell./radius_cell_squared;
   % (radius_z_cell./radius_cell)./radius_cell;
   %# set max value to 2 to truncate large force contributions
   %# at radius < 8
Fz_cell_matrix = min(Fz_cell_matrix, 2);
Fz_cell_matrix = max(Fz_cell_matrix, -2);

Ftotal = (Fx_cell_matrix.^2 + Fz_cell_matrix.^2).^0.5;

Fx_cell = sum(sum(Fx_cell_matrix));
Fz_cell = sum(sum(Fz_cell_matrix));

% Call for position of other bots
bot_loc_1868 = Bot_Tracker1868controller(1);

% Bot-to-bot force calculation
radius_x_1868 = bot_loc_1868(1)-bot_loc_2031(1);
radius_z_1868 = bot_loc_1868(2)-bot_loc_2031(2);
radius_1868 = (radius_x_1868^2 + radius_z_1868^2)^0.5;

Fx_1868 = ROBOT_CHARGE^2*radius_x_1868/radius_1868^3;
Fz_1868 = ROBOT_CHARGE^2*radius_z_1868/radius_1868^3;

bot_loc_2014 = Bot_Tracker2014controller(1);

% Bot-to-bot force calculation
radius_x_2014 = bot_loc_2014(1)-bot_loc_2031(1);
radius_z_2014 = bot_loc_2014(2)-bot_loc_2031(2);
radius_2014 = (radius_x_2014^2 + radius_z_2014^2)^0.5;

Fx_2014 = ROBOT_CHARGE^2*radius_x_2014/radius_2014^3;
Fz_2014 = ROBOT_CHARGE^2*radius_z_2014/radius_2014^3;

bot_loc_2036 = Bot_Tracker2036controller(1);

% Bot-to-bot force calculation
radius_x_2036 = bot_loc_2036(1)-bot_loc_2031(1);
radius_z_2036 = bot_loc_2036(2)-bot_loc_2031(2);
radius_2036 = (radius_x_2036^2 + radius_z_2036^2)^0.5;

Fx_2036 = ROBOT_CHARGE^2*radius_x_2036/radius_2036^3;
Fz_2036 = ROBOT_CHARGE^2*radius_z_2036/radius_2036^3;

bot_loc_2045 = Bot_Tracker2045controller(1);

% Bot-to-bot force calculation
radius_x_2045 = bot_loc_2045(1)-bot_loc_2031(1);
radius_z_2045 = bot_loc_2045(2)-bot_loc_2031(2);
```

```
        radius_2045 = (radius_x_2045^2 + radius_z_2045^2)^0.5;

        Fx_2045 = ROBOT_CHARGE^2*radius_x_2045/radius_2045^3;
        Fz_2045 = ROBOT_CHARGE^2*radius_z_2045/radius_2045^3;

        Fx = Fx_cell - Fx_1868 - Fx_2014 - Fx_2036 - Fx_2045; %
Subtracting implements repulsive charge
        Fz = Fz_cell - Fz_1868 - Fz_2014 - Fz_2036 - Fx_2045; %
between bots

        theta = atan2(Fz, Fx);

        % Record Ftotal for movie making
        % Add bot-to-bot force into Ftotal
        index=index+1;

Ftotal(uint8(ceil(bot_loc_1868(1))),uint8(ceil(bot_loc_1868(2))))=...

Ftotal(uint8(ceil(bot_loc_1868(1))),uint8(ceil(bot_loc_1868(2))))...
            + (Fx_1868^2+Fz_1868^2)^0.5;

Ftotal(uint8(ceil(bot_loc_2014(1))),uint8(ceil(bot_loc_2014(2))))=...

Ftotal(uint8(ceil(bot_loc_2014(1))),uint8(ceil(bot_loc_2014(2))))...
            + (Fx_2014^2+Fz_2014^2)^0.5;

Ftotal(uint8(ceil(bot_loc_2036(1))),uint8(ceil(bot_loc_2036(2))))=...

Ftotal(uint8(ceil(bot_loc_2036(1))),uint8(ceil(bot_loc_2036(2))))...
            + (Fx_2036^2+Fz_2036^2)^0.5;

Ftotal(uint8(ceil(bot_loc_2045(1))),uint8(ceil(bot_loc_2045(2))))=...

Ftotal(uint8(ceil(bot_loc_2045(1))),uint8(ceil(bot_loc_2045(2))))...
            + (Fx_2045^2+Fz_2045^2)^0.5;

        % Replicate to indexed Ftotal
        Ftotal_index(index,:,:)=Ftotal(:,:);

%fprintf('THETA %6.2f\n\n', (theta*180/3.14159));
%fprintf('bot_loc %6.2f %6.2f\n\n', bot_loc(1), bot_loc(2));

    end

    % Maneuver and move toward theta be it newly calculated or the
existing.

% fprintf('THETA  bot_loc %6.2f %6.2f\n', (theta*180/3.14),
(bot_loc(3)*180/3.14));
    % Check to see if the desired heading (theta) is within 1
```

```
    % degrees or 0.175 radians from the current bot heading.
    if (abs(theta-bot_loc_2031(3))>0.0175)

        if ((abs(theta-bot_loc_2031(3))<3.14159))
            if (theta > bot_loc_2031(3))
                wheel_speed = int16([200,150]);
            elseif (theta < bot_loc_2031(3))
                wheel_speed = int16([150,200]);
            end
        else
            if (theta > bot_loc_2031(3))
                wheel_speed = int16([150,200]);
            elseif (theta < bot_loc_2031(3))
                wheel_speed = int16([200,150]);
            end
        end
    else
        wheel_speed = int16([200,200]);
    end

    % Send calcualted wheel speeds to epuck.
fprintf('WHEEL SPEED2 %6i %6i\n\n', wheel_speed);
    ePic=set(ePic,'speed',wheel_speed);
    ePic=update(ePic);

end

ePic=set(ePic,'speed',[0,0]); % stop the bot
ePic=update(ePic);


fprintf('DIST LEFT - FINAL, TRAVELLED:%6i %6i  \n', odometer(2,2),
odometer(2,2)-odometer(1,2));
fprintf('DIST RIGHT - FINAL, TRAVELLED:%6i %6i  \n', odometer(2,1),
odometer(2,1)-odometer(1,1));fprintf('AVE DISTANCE TRAVELLED: %6i\n',
((odometer(2,1)-odometer(1,1))+(odometer(2,2)-odometer(1,2)))/2);
fprintf('image_count: %6i\n', image_count);
fprintf('prox_count: %6i\n', prox_count);
ePic=disconnect(ePic); % disconnect from bot
```

Appendix D. MATLAB MEX Software Code for Surrogate Global Positioning

```
//==
//== Bot_Tracker Function ==--
//==
//== This C++ code is used to generate a MEX file in MATLAB.  It is
//== used in conjunction with NaturalPoint's Tracking Tools
//== application.
//== The name of the tracked object (as established in the Tracking
//== Tools project) and VRPN port is identified
//== in the code below (e.g. "2036@localhost:3883" meaning object name
//== "2036" and port 3883 on the local computer).
//== It then obtains the position/orientation of "name" from Tracking
//== Tools via the VRPN broadcast.  Finally, it returns that data to
//== MATLAB for use in the M-code.
//==
//== You may also need to download the VRPN distribution files that
//== can be found at ftp://ftp.cs.unc.edu/pub/packages/GRIP/vrpn.
//==
//== Reference MEX function description at
//== http://www.mathworks.com/support/tech-notes/1600/1605.html
//== MATLAB MEX combile command:   mex Bot_Tracker2045.cpp -
//== IC:\vrpn_07_26_64bit\vrpn\ -
//== LC:\vrpn_07_26_64bit\vrpn\pc_win32\Debug -lvrpn

#include "C:\vrpn_07_26\vrpn\vrpn_Tracker.h"    //Path to these .h
  // files is dependent on your VRPN installation.
#include "mex.h"

// Global variable definition
double pos_ori[4];

void  VRPN_CALLBACK handle_pos (void *, const vrpn_TRACKERCB t)

{
  // Only values of interest are t.pos[0] or TT's x, t.pos[2] or TT's
z,
  // t.quat[1], and t.quat[3].  From those,
  // x, z, and yaw can be calculated and passed back to the main
routine.
      pos_ori[0] = t.pos[0];
      pos_ori[1] = t.pos[2];
      pos_ori[2] = t.quat[1];
      pos_ori[3] = t.quat[3];
}
```

```
//== mexFunction definition
void mexFunction(int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[])

{

      double *outMatrix;  //output matrix
      double *inMatrix;   //input matrix
  static vrpn_Tracker_Remote *tkr;
  inMatrix = mxGetPr(prhs[0]);
  if (inMatrix[0] == 0) // Initialize the tracker at first timestep
    {
        tkr = new vrpn_Tracker_Remote("1868@localhost:3883");
    }

  tkr->register_change_handler(NULL, handle_pos);
  tkr->mainloop();

  plhs[0] = mxCreateDoubleMatrix(1,3,mxREAL);  //allocate memory
                                    // and assign output pointer

  outMatrix = mxGetPr(plhs[0]);  //get a pointer to the real data
                                    // in the output matrix

  //populate the output matrix with data
      outMatrix[0] = -100*pos_ori[1];
      // This returns Webots x which is derived from TT's z.
      // *100 turns TT's meters into Webots' centimeters
      // Negative is to change TT's z axis (downward) to Webots' x
      //(upward)

  outMatrix[1] = 100*pos_ori[0];
      // This returns Webots' z which is derived from TT's x.

  //calculate the yaw angle in radians (from two quaternions)
  outMatrix[2]        =        -atan2(2*pos_ori[2]*pos_ori[3],        1-
2*pos_ori[2]*pos_ori[2]);

}
```

Given this C++ code including the appropriate formatting and callbacks, the MEX-function compile with library inclusion is achieved via the MATLAB command prompt as: "mex Bot_Tracker.cpp -IC:\vrpn_07_26\vrpn\ -LC:\vrpn_07_26\vrpn\pc_win32\Debug -lvrpn"

The resulting function that can be called from within MATLAB formatted simply as "Bot_Tracker."

References

[1] Olfati-Saber, Reze: Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory, *IEEE Transactions on Automatic Control*, Vol. 51, No. 3, March 2006 pp. 401-420.

[2] Mait, Joseph: Micro Autonomous Systems and Technology Collaborative Technology Alliance, *The Micro Autonomous Systems and Technologies Principal Investigators' Meeting*, College Station, Maryland, October 15, 2008.

[3] Anisi, David; Ogren, Petter; and Hu, Xiaoming: Cooperative Minimum Time Surveillance with Multiple Ground Vehicles, *IEEE Transactions on Automatic Control*, Vol. 55, No. 12, December 2010, pp. 2679-2691.

[4] Kumar, Manish; Garg, Devendra; and Kumar Vijay: Segregation of Heterogeneous Units in a Swarm of Robotic Agents, *IEEE Transactions on Automatic Control*, Vol. 55, Issue 3, February 8, 2010, pp. 743-748.

[5] Parker, Lynn: Multi-robot Team Design For Real-World Applications, *International Symposium On Distributed Autonomous Robotic Systems*, October 29, 1996, pp. 91-102.

[6] Şahin, Erol; Girgin, Sertan; Bayindir, Levent; Turgut, and Ali Emre: Swarm Robotics, *Swarm Intelligence*, Germany, Springer Berlin Heidelberg, 2008, pp. 87-100.

[7] Parker, Lynn: Task-oriented Multi-robot Learning in Behavior-based Systems, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 3, November 4, 1996, pp. 1478-1487.

[8] Vig, Lovekesh and Adams, Julie: Multi-robot Coalition Formation, *IEEE Transactions on Robotics*, Vol. 22, No. 4, August 2006, pp. 637-649.

[9] Garg, Devendra: Modeling, Analysis, and Control of Swarming Agents in a Probabilistic Framework, Proposal # 52488 funded by the Army Research Office, April 2008.

[10] Mihaylova, Lyudmila; Lefebvre, Tine; Bruyninckx, Herman; Gadeyne, Klaas; and DeSchutter, Joris: Active Sensing for Robotics – A Survey, *5th International Conference on Numerical Methods and Applications*, 2002, pp.316-324.

[11] Martinelli, Agostino: Using the Distribution Theory to Simultaneously Calibrate the Sensors of a Mobile Robot, *Research Report from the Institut National de Recherche en Informatique et en Automatique*, No. 6796, January 2009.

[12] Stone, Lawrence: The Process of Search Planning: Current Approaches and Continuing Problems, *Operations Research*, Vol. 31, Issue 2, 1983, pp. 207-233.

[13] Walter, William: An Electro-mechanical Animal, *Dialectica*, Vol. 4, 1950, pp. 42-49.

[14] iRobot Family of Products. Available: www.irobot.com.

[15] Railbert, Marc; Blankespoor, Kevin; Nelson, Gabriel; Playter, Rob; and the BigDog Team: BigDog, the Rough-terrain Quadruped Robot, *17th World Congress of the International Federation of Automatic Control*, July 6-11, 2008.

[16] Tanner, Herbert and Kumar, Amit: Formation Stabilization of Multiple Agents Using Decentralized Navigation Functions, *In Robotics: Science and Systems I*, June 2005.

[17] Singh, Amarjeet; Krause, Andreas; Guestrin, Carlos; and Kaiser, William: Efficient Informative Sensing using Multiple Robots, *Journal of Artificial Intelligence Research*, Vol. 34, 2009, pp. 707-755.

[18] Garg, Devendra, and Young, Reed: Coordinated Control of Cooperating Robotic Manipulators, *International Journal of Systems Science*, Vol. 21, Issue 11, November 1990, pp. 2161-2176.

[19] Young, Reed: The Coordination of Multiple Robotic Manipulators, Master's thesis, Duke University, 1987.

[20] Garg, Devendra and Young, Reed: Computer Simulation of coordinated Multiple Robots, *Twentieth Annual Pittsburg Conference on Modeling and Simulation*, Vol. 20, Part 5, May 5, 1989, pp. 2131-2135.

[21] Jadbabaie, Ali et al: Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbor Rules, *IEEE Transactions on Automatic Control*, Vol. 48, Issue 6, 2003, pp. 988.

[22] Cortes, Jorge; Martinez, Sonia; Karatas, Timur; and Bullo, Francesco: Coverage Control for Mobile Sensing Networks, *IEEE International Conference on Robotics and Automation*, Vol. 2, 2002, pp. 1327-1332.

[23] Shehory, Onn and Kraus, Sarit: Methods for Taks Allocation via Agent Coalition Formation, *Artificial Intelligence*, Vol. 101, Issues 1-2, May 1998, pp. 165-200.

[24] Montesano, Luis; Minguez, Javier; and Montano, Luis: Modeling Dynamic Scenarios for Local Sensor-based Motion Planning, *Autonomous Robots*, Vol. 25, Issue 3, 2008, pp. 231-251.

[25] Tang, Zhijun and Ozguer, Umit: Motion Planning for Multi-target Surveillance with Mobile Sensor Agents, *IEEE Transactions on Robotics*, Vol. 21, Issue 5, 2005, pp. 898-908.

[26] Chomchana, Trevai et al: Multiple Mobile Robot Surveillance in Unknown Environments, *Advance Robotics*, Vol. 21, Issue 7, 2007, pp. 729-749.

[27] Fricke, Gregory; Zhang, Guoxian; Caccavale, Adam; Li, Walter; and Garg, Devendra P., An Intelligent Sensing Network of Distributed Swarming Agents for Perimeter Detection and Surveillance, Paper No. DSCC 2010-4256, *2010 ASME Dynamic Systems and Control Conference*, September 12-15, 2010.

[28] Parker, Lynn: Multiple Mobile Robot Systems, *Springer Handbook of Robotics*, 2008, pp. 921-941.

[29] Burgard, Wolfram; Moors, Mark; Stachniss, Cyrill; and Schneider, Frank: Coordinated Multi-robot Exploration, *IEEE Transactions on Robotics*, Vol. 21, No. 3, June 3, 2005, pp. 376-386.

[30] Chung, Timothy; Gupta, Vijay; Burdick, Joel; and Murray, Richard: On a Decentralized Active Sensing Strategy using Mobile Sensor Platforms in a Network, *IEEE Conference on Decision and Control*, Vol. 2, December 14-17, 2004, pp. 1914-1919.

[31] Bopardikar, Shaunak; Bullo, Francesco; and Hespanha, Joao: Cooperative Pursuit with Sensing Limitations, *American Control Conference*, July 9-13, 2007, pp. 5394-5399.

[32] Capi, Genci: Multi-robot Collaborative Task Performance based on Evolution of Neural Controllers, *ICGST-ARAS Journal*, Vol. 9, Issue 1, July 2009, pp. 11-16.

[33] Butler, Zach and Rus, Daniela: Controlling Mobile Sensors for Monitoring Events with Coverage Constraints, *IEEE International Conference on Robotics and Automation*, April 2004, pp. 1568-1573.

[34] Gerkey, Brian and Mataric, Maja: A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems, *International Journal of Robotics Research*, Vol. 23, Issue 9, 2004, pp. 939-954.

[35] Mittal, Anurag and Davis, Larry: A General Method for Sensor Planning in Multi-sensor Systems: Extensions to Random Occlusion, *International Journal of Computer Vision*, Vol. 76, Issue 1, 2008, pp. 31-52.

[36] Baghaei, Khashayar and Agah, Arvin: Task Allocation Methodologies for Multi-Robot Systems, *Technical Report ITTC-FY2003-TR-20272-01*, November 2002.

[37] Voos, Holger: Agent-based Distributed Resource Allocation in Technical Dynamic Systems, *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, 2006, pp. 157-162.

[38] Kim, Dong Hun; Wang, Hua; and Shin, Seiichi: Decentralized Control of Autonomous Swarm Systems Using Artificial Potential Functions: Analytical Design Guidelines, *43rd IEEE Conference on Decision and Control*, Vol. 1, May 2, 2006, pp. 369-394.

[39] Fricke, Gregory; Milutinovic, Dejan; Garg, Devendra: Robotic Pose Estimation via an Adaptive Kalman Filter using State-varying Noise, *14th IASTED International Conference, Robotics and Applications*, November 2-4, 2009, pp. 289-295.

[40] Fricke, Gregory and Garg, Devendra: Discrimination and Tracking of Individual Agents in a Swarm of Robots, *American Control Conference*, Section ThA11, June 30-July 2, 2010.

[41] Falconi, Riccardo: Coordinated Control of Robotic Swarms in Unknown Environments, *Automatica e Ricerca Operative*, April 16, 2009.

[42] Kube, Ronald; and Zhang, Hong: Collective Robotics: From Social Insects to Robots, *Adaptive Behavior*, September 1993, pp. 189-218.

[43] Andrews, Burton; Passino, Kevin; and Waite, Thomas: Social Foraging Theory for Robust Multiagent System Design, *IEEE Transactions on Automation Science and Engineering*, Vol. 4, No. 1, January 2007, pp. 79-86.

[44] Fujisawa, Ryusuke; Imamura, Hikaru; Hashimoto, Takashi; and Matsuno, Fumitoshi: Communication Using Pheromone Field for Multiple Robots, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 22-26, 2008, pp. 1391-1396.

[45] Kube, Ronald and Zhang, Hong: Collective Robotics: From Social Insects to Robots, *Adaptive Behavior*, Vol. 2, 1993, pp. 189-218.

[46] Parker, Lynn: Guest Editorial, Special Issue on Heterogeneous Multi-robot Systems, *Autonomous Robots*, Vol. 8, Issue 3, June 1, 2000, pp. 207-208.

[47] Parker, Lynn: The Effect of Heterogeneity in Teams Of 100+ Mobile Robots, *Multi-Robot Systems: From Swarms to Intelligent Automata, 2003 International Workshop on Multi-robot Systems*, Vol. II, March 2003, pp. 205-218.

[48] Potter, Mitchell; Meeden, Lisa; and Schultz, Alan: Heterogeneity in the Coevolved Behaviors of Mobile Robots: The Emergence of Specialists, *International Joint Conference On Artificial Intelligence*, Vol. 17, Part 1, 2001, pp. 1337-1343.

[49] Schwager, Mac; Slotine, Jean-Jacques; and Rus, Daniela: "Consensus Learning for Distributed Coverage Control, *Proceedings of International Conference on Robotics and Automation*, Pasadena, CA, May 2008.

[50] Schwager, Mac; McLurkin, James; and Rus, Daniela: Distributed Coverage Control with Sensory Feedback for Networked Robots, *Robotics: Science and Systems II*, 2007.

[51] Cook, Diane; Gmytrasiewicz, Piotr; and Holder, Lawrence: Decision-theoretic Cooperative Sensor Planning, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 10, October 1996, pp. 1013-1023.

[52] Bieniawski, Stefan; Pigg, Paul; and Vian, John: Exploring Health-enabled Mission Concepts in the Vehicle Swarm Technology Laboratory, *AIAA Infotech@Aerospace Conference*, April 6-9, 2009.

[53] Grabowski, Robert; Navarro-Serment, Luis; Paredis, Christiaan; and Khosla, Pradeep: Heterogeneous Teams of Modular Robots for Mapping and Exploration, *Autonomous Robotics*, Vol. 8, Number 3, June 2000, pp. 293-308.

[54] Wagner, Israel; Lindenbaum, Michael; and Bruckstein, Alfred: MAC versus PC: Determinism and Randomness as Complementary Approaches to Robotic Exploration of Continuous Unknown Domains, *The International Journal of Robotics Research*, Vol. 19, Issue 1, January 1, 2000, pp. 12-31.

[55] Cai, Chenghui and Ferrari, Silvia: Information-driven Sensor Path Planning by Approximate Cell Decomposition, *IEEE Transactions on Systems, Man, and Cybernetics -- Part B: Cybernetics*, Vol. 39, No. 3, June 2009, pp. 672-689.

[56] Kristensen, Steen: Sensor Planning with Bayesian Decision Theory, *In Reasoning with Uncertainty in Robotics*, 1995, pp. 273-286.

[57] Qian, Ming and Ferrari, Silvia: Probabilistic Deployment for Multiple Sensor Systems, *Smart Structures and Materials 2005: Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems*, Vol. 5765, May 17, 2005, pp. 85-96.

[58] Zhang, Guoxian; Ferrari, Silvia; and Qian, M.: An Information Roadmap Method for Robotic Sensor Path Planning, *Journal of Intelligent and Robotic Systems*, Vol. 56, Issues 1-2, 2009, pp. 69-98.

[59] Atta, Debabrata; Subudhi, Bidyadhar; and Gupta, Madan: Cohesive Motion Control Algorithm for Formation of Multiple Autonomous Agents, *Journal of Robotics*, May 31, 2010, pp. 1-13.

[60] Desai, Jaydev; Ostrowski, Jim; and Kumar, Vijay: Controlling Formations of Multiple Mobile Robots, *IEEE International Conference on Robotics & Automation*, Vol. 4, May 1998, pp. 2864-2869.

[61] Alur, Rajeev; Das, Aveek; Esposito, Joel; Fierro, Rafael; Grudic, Gregory; Hur, Yerang; Kumar, Vijay; Ostrowski, James; Pappas, George; Southall, B., Spletzer, John; and Taylor, Camillo: A Framework and Architecture for Multirobot Coordination, *Experimental Robotics VII*, 2001, pp. 303-322.

[62] Kim, Jin; Shim, David; Rashid, Shahid; and Sastry, Shankar: Hierarchical System for Multiple-agent Scenarios, *DTIC Report A794314*, January 1, 2002.

[63] Dias, M. Bernardine; Browning, Brett; Veloso, Manuela; and Stentz, Anthony: Dynamic Heterogeneous Robot Teams Engaged in Adversarial Tasks, *Technical Report CMU-RI-TR-05-14*, Robotics Institute, Carnegie Mellon University, April 2005.

[64] Mataric, Maja: Issues and Approaches in the Design of Collective Autonomous Agents, *Robotics and Autonomous Systems 16*, 1995, pp. 321-331.

[65] Zhang, Fei; Yugeng, Xi; Lin, Zongli; and Chen, Weidong: Constrained Motion Model of Mobile Robots and Its Applications, *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, Vol. 39, Issue 3, 2009, pp. 773-787.

[66] Mondada, Francesco et al: The e-puck, A Robot Designed for Education in Engineering, *9th Conference on Autonomous Robot Systems and Competitions*, Vol. 1, No. 1, May 7, 2009, pp. 59-65.

[67] e-puck Educational Robot. Available: www. e-puck.org.

[68] Scott, Adele and Yu, Changbin: Cooperative Multi-agent Mapping and Exploration in Webots, *Proceedings of the 4th International Conference on Autonomous Robots and Agents,* February 2009, pp. 56-61.

[69] Williams, Robert; Wu, Jianhua: Dynamic Obstacle Avoidance for an Omnidirectional Mobile Robot, *Journal of Robotics*, September 14, 2010, pp. 1-14.

[70] Ecole Polytechnique Federale de Lausanne, ePic2 v2.1 Documentation Available: http://www.e-puck.org/index.php?option=com_content&view=article&id =29&Itemid=27.

[71] Young, Brian J.: Characterization of Proximity Sensing in Mobile Robotics, Technical Report, Duke University, 2011.

[72] Yang, Xianbin; Patel, Rajni; and Moallem, Mehrdad: A Fuzzy-Braitenberg Navigation Strategy for Differential Drive Mobile Robots, *Journal of Intelligent and Robotic Systems*, Vol. 47, Issue 2, October 1, 2006, pp 101-124.

[73] Parker, Lynne: Evaluating Success in Autonomous Multi-robot Teams: Experiences from ALLIANCE Architecture Implementations, *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 13, No. 2, 2001, pp. 95-98.

[74] Choset, Howie: Coverage for Robotics - A Survey of Recent Results, *Annals of Mathematics and Artificial Intelligence*, Vol. 31, No. 1-14, 2001, pp. 113-126.

[75] Lum, Christopher; Rysdyk, Rolf; and Pongpunwattana, Anawat: Occupancy Based Map Searching Using Heterogeneous Teams of Autonomous Vehicles, *AIAA Guidance, Navigation, and Control Conference*, August 21-24, 2006, pp 1-15.

[76] O'Callaghan, Simon; Ramos, Fabio; and Durrant-Whyte, Hugh: Contextual Occupancy Maps using Gaussian Processes, *2009 IEEE International Conference on Robotics and Automation*, May 12-17, 2009, pp. 1054-1060.

[77] Grabowski, Robert; Khosla, Pradeep; and Choset, Howie: An Enhanced Occupancy Map for Exploration via Pose Separation, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2003, pp 705-710.

[78] Elfes, Alberto: Using Occupancy Grids for Mobile Robot Perception and Navigation, *Computer*, Vol. 22, Issue 6, June, 1989, pp. 46-57.

[79] Lum, Christopher; Vagners, Juris; Jang, Jung Soon; and Vian, John: Partitioned Searching and Deconfliction: Analysis and Flight Tests, *American Control Conference*, June 30 - July 2, 1010, pp. 6409-6416.

[80] Kumar, Manish; Garg, Devendra P.: Intelligent Sensor Uncertainty Modelling Techniques and Data Fusion, *Control and Intelligent Systems*, Vol. 37, No. 2, November 2-4, 2009, pp 67-77.

[81] Zhou, Hongjun and Sakane, Shigeyuki: Sensor Planning for Mobile Robot Localization – A Hierarchical Approach using a Bayesian Network and a Particle Filter, *IEEE Transactions on Robotics*, Vol. 24, Issue 2, 2008, pp. 481-487.

[82] Pujol, Francisco; Garcia, Jose; Pujol, Mar; and Rizo, Ramon: A Morphological Proposal for Vision-based Path Planning, *Innovations in Applied Artificial Intelligence, Lecture Notes in Computer Science*, Vol. 3533/2005, 2005, pp. 62-64.

[83] Rimon, Elon and Koditschek, Daniel: Exact Robot Navigation Using Artificial Potential Functions, *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 5, October, 1992, pp 501-518.

[84] Koditschek, Daniel: Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations, *IEEE International Conference on Robotics and Automation*, Vol. 4, February 10, 1987, pp. 1-4.

[85] Hwang, Yong and Ahuja, Narendra: A Potential Field Approach to Path Planning, *IEEE Transactions on Robotics and Automation*, Vol. 8 No. 1, February, 1992, pp 23-32.

[86] Koren, Yoram and Broenstein, Johann: Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation, *IEEE International Conference on Robotics and Automation*, April 1991, pp. 1398-1404.

[87] Ren, Jing; McIsaac, Ken; Patel, Rajni; and Peters, Terry: A Potential Field Model Using Generalized Sigmoid Functions, *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, Vol. 37, Issue 2, 2007, pp. 477-484.

[88] Schwager, Max; Slotine, Jean; and Rus, Daniela: Consensus Learning for Distributed Coverage Control, *International Conference on Robotics and Automation*, May 2008.

[89] Zelinsky, Alex; Jarvis, Ray; Byrne, John; and Yuta, Shin'ichi: Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot, *International Conference on Advanced Robotics*, Vol. 13, 1993, pp. 533-538.

[90] Ren, Jing and McIsaac, Kenneth: A Hybrid-systems Approach to Potential Field Navigation for a Multi-robot Team, *IEEE International Conference on Robotics and Automation*, September 14-19, 2003, pp. 3875-3880.

[91] Cosio, Arambula and Castaneda, Padilla: Autonomous Robot Navigation using Adaptive Potential Fields, *Mathematical and Computer Modeling*, Vol. 40, Issues 9-10, November 2004, pp. 1141-1156.

[92] Balch, Tucker and Arkin, Ronald: Behavior-based Formation Control for Multirobot Teams, IEEE *Transactions on Robotics and Automation*, Vol. 14, Issue 6, 1998, pp. 926.

[93] Sandhlom, Tuomas and Lesser, Victor: Coalitions Among Computationally Bounded Agents, *Artificial Intelligence*, Vol. 94, Issues 1-2, July 1997, pp. 99-137.

[94] Ren, Jing and McIsaac, Kenneth: "A Hybrid-systems Approach to Potential Field Navigation for a Multi-robot Team, *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, Vol. 3, September 2003, pp. 3875-3880.

[95] Cosio, Arambula and Castaneda, Padilla: Autonomous Robot Navigation using Adaptive Potential Fields, *Mathematical and Computer Modeling*, Vol. 40, Issues 9-10, November 2004, pp. 1141-1156.

[96] Barnes, Laura: A Potential Field Based Formation Control Methodology for Robot Swarms, Ph.D. Dissertation, University of South Florida, March 3, 2008.

[97] Krogh, Bruce and Thorpe, Charles: Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles, *IEEE International Conference on Robotics and Automation*, Vol. 3, 1986, pp. 1664-1669.

[98] Ge, Shuzhi and Cui, Ya: New Potential Functions for Mobile Robot Path Planning, *IEEE Transactions on Robotics and Automation*, Vol. 16, Issue 5, 2000, pp. 615-620.

[99] Valavanis, Kimon; Hebert, Timothy; Kolluru, Ramesh; and Tsourveloudis, Nikos: Mobile Robot Navigation in 2-D Dynamic Environments using an Electrostatic Potential Field, *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, Vol. 30, Issue 2, 2000, pp. 187-196.

[100] Barraquand, Jerome; Langlois, Bruno; and Latombe, Jean-Claude: Numerical Potential Field Techniques for Robot Path Planning, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22, Issue 2, 1992, pp. 224-241.

[101] Pimenta, Luciano et al: Robot Navigation Based on Electrostatic Field Computation, *IEEE Transactions on Magnetics*, Vol. 42, Issue 4, 2006, pp. 1459-1462.

[102] Branicky, Michael: Multiple Lyapunov Functions and Other Analysis Tools for Switched and Hybrid Systems, *IEEE Transactions on Automatic Control*, Vol. 43, No. 4, April 1998, pp 475-482.

[103] Rohrer Fabien: Transfer a Webots Controller of the Rat's Life Contest from Simulation to Reality, *www.Cyberbotics.com*.

[104] Magyar, Balazs; Forhecz, Zoltan; and Korondi, Peter: Developing an Efficient Mobile Robot Control Algorithm in the Webots Simulation Environment, *IEEE International Conference on Industrial Technology*, Vol. 1, December 10-12, 2003, pp. 179-184.

[105] Michel, Olivier: Webots Professional Mobile Robot Simulation, *International Journal of Advanced Robotic Systems*, Vol. 1, No. 1, 2004, pp. 40-43.

[106] Magyar, Balazs; Forhecz, Zoltan; and Korondi, Peter: Developing an Efficient Mobile Robot Control Algorithm in the Webots Simulation Environment, *IEEE International Conference on Industrial Technology*, Vol. 1, December 10-12, 2003, pp. 179-184.

[107] Magnenat, Stephane; Retornaz, Philippe; Noris, Basilio; and Mondada, Francesco: Scripting the Swarm: Event-based Control of Microcontroller-based Robots, *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, November 3-7, 2008, pp. 1-14.

[108] Hou, Xiaolei; Yu, Changbin; and Summers, Tyler: A Virtual Framework of Robotic SWARM Testbed, *21st Annual International Conference on Chinese Control and Decision Conference*, June 17-19, 2009, pp. 930-935.

[109] Halaas, David; Bieniawski, Stefan; Pigg, Paul; and Vian, John: Control and Management of an Indoor, Health Enabled, Heterogeneous Fleet, *AIAA Infotech@Aerospace Conference*, April 6-9, 2009.

[110] Saad, Emad; Vian, John; Clark, Greg; and Bieniawski, Stefan: Vehicle Swarm Rapid Prototyping Testbed, *AIAA Infotech@Aerospace Conference*, April 6-9, 2009.

[111] Kumar, Vijay; Rus, Daniela; and Sukhatme, Gaurav: Networked Robots, *Springer Handbook of Robotics*, Chapter 41, June 27, 2008, pp. 943-958.

[112] Hou, Xiaolei and Yu, Changbin: On the Implementation of a Robotic SWARM Testbed, *4th International Conference on Autonomous Robots and Agents*, February 10-12, 2009, pp. 27-32.

[113] Samiloglu, Andac; Cayirpunar, Omer; Gazi, Veysel; and Koku, Bugra: An Experimental Set-up for Multi-robot Applications, *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, November 3-4, 2008, pp. 539-550.

[114] Scott, C. Yu: Cooperative Multi-agent Mapping and Exploration in Webots, *4th International Conference on Autonomous Robots and Agents*, February 10-12, 2009, pp. 56-61.

[115] Ahmad, Hamzah and Namerikawa, Toru: H∞ Filtering Convergence and Its Application to SLAM, *ICROS-SICE International Joint Conference*, August 18-21, 2009, pp. 2875-2880.

[116] Fricke, Gregory; Milutinovic, Dejan; Garg, Devendra: Sensing and Estimation on a Modular Testbed for Swarm Robotics, paper TuBT4.4., *2nd Annual Dynamic Systems and Control Conference*, October 12-14, 2009, Hollywood, CA,

[117] Vicon website. Available: http://www.vicon.com.

[118] NaturalPoint website. Available: http://www.naturalpoint.com.

[119] Virtual Reality Peripheral Network website. Available: http://www.cs.unc.edu/Research/vrpn/.

[120] Taylor, Russell; Hudson, Thomas; Seeger, Adam Weber, Hans; Juliano, Jeffrey; and Helser, Aron: VRPN: A Device-independent, Network-Transparent VR Peripheral System, *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, November 15-17, 2001, pp. 55-61.

[121] Natural Point's VRPN Streaming Sample Distribution Files. Available: http://media.naturalpoint.com/software/OptiTrack_files/VRPN-Tracking-v3.zip.

[122] MathWorks MATLAB "MEX-file Guide." Available: http://www.mathworks.com/support/tech-notes/1600/1605.html#example5.

[123] Balch, Tucker: Communication, Diversity, and Learning: Cornerstones of Swarm Behavior, *Swarm Robotics, Lecture Notes in Computer Science*, Vol. 3342/2005, January 28, 2005, pp. 21-30.

[124] Huang, Wesley; Ollis, Mark; Happold, Michael; and Stancil, Brian: Image-based Path Planning for Outdoor Mobile Robots, *Journal of Field Robotics*, Vol. 26, No. 2, 2009, pp. 196-211.

[125] Mezouar, Yocef and Chaumette, Francois: Path Planning for Robust Image-based Control, *IEEE Transactions on Robotics and Automation*, Vol. 18, No. 4 August, 2002, pp. 534-539.

[126] Tilbury, Dawn and Ulsoy, Galip: A New Breed of Robots that Drive Themselves, *Mechanical Engineering*, February 2011, pp. 28-33.

[127] Shojaeipour, Shahed; Haris, Sallehuddin; and Khairir, Muhammad: Vision-based Mobile Robot Navigation Using Image Processing and Cell Decomposition, *Visual Informatics: Bridging Research and Practice, Lecture Notes in Computer Science*, Vol. 5857/2009, 2009, pp. 90-96.

[128] Tedder, Maurice and Jin-Chung, Chan: Autonomous Robot Vision Software Design using MATLAB Toolboxes, *SPIE Intelligent Robots and Computer Vision XXII: Algorithms, Techniques, and Active Vision*, October 25, 2004, pp. 99-106.

[129] Spletzer, John and Taylor, Camillo: Sensor Planning and Control in a Dynamic Environment, *IEEE International Conference on Robotics and Automation*, May 2002, pp. 676-681.

[130] Nabbe, Bart; Hoiem, Derek; Efros, Alexei; and Hebert, Martial: Opportunistic Use of Vision to Push Back the Path-planning Horizon, *Robotics Institute*, Paper 282, January 1, 2006.

Biography

Reed F. Young was born on August 17, 1963 in Auburn, NY. He received a Bachelor of Science degree in Mechanical Engineering from the University of Nevada, Reno in 1986 attending on a Reserve Officer Training Corps 4-year scholarship. After commissioning, entry onto active duty, and selection for the Army's Technical Enrichment Program fully-funded graduate scholarship, Reed attended Duke University where he earned a Master of Science Degree in Mechanical Engineering in 1988 studying under the most-able tutelage of Professor Devendra P. Garg. The title of his Master's thesis was "Coordination of Multiple Robotic Manipulators" which spawned two articles: "Coordinated Control of Cooperating Robotic Manipulators" and "Computer Simulation of Coordinated Multiple Robots."

In 2008, Reed returned to Duke University to pursue a Ph.D. again under the supervision of Professor Garg and while serving as a program manager at the Army Research Office. His research interests center on the science, technology, and systems of robotics and autonomy that hold promise to take soldiers out of harms' way. The first article published from this work was titled "Modeling, Simulation, and Characterization of Distributed Multi-agent Systems." Several other submissions are under consideration or being drafted.

Reed is a member of the American Society of Mechanical Engineers and was also a senior member of the American Society of Logistics Engineers. He currently holds the rank of colonel in the U.S. Army and has served deployments to Iraq and Afghanistan.