NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AUTONOMOUS COORDINATION AND ONLINE MOTION MODELING FOR MOBILE ROBOTS

by

Eric Sjoberg

September 2007

Co-Advisors: Kevin Squire
Craig Martell

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2007	Master's Thesis

4. TITLE AND SUBTITLE Autonomous Coordination and Online Motion Modeling for Mobile Robots	5. FUNDING NUMBERS
6. AUTHOR Eric J Sjoberg	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	

13. ABSTRACT (maximum 200 words)

Robots are rapidly becoming more involved in everyday military operations. As robots become more capable, their tasks will increase to include such roles as exploring enemy controlled buildings and caves. The goal of this thesis is to explore methodologies that allow robots to operate more autonomously. The first goal is to develop an algorithm that allows groups of robots to construct controlled formations with only local information. Experiments investigate the ability of this algorithm to handle obstacles, dynamic conditions, and varying number of robots. The second goal of this work is to demonstrate a method by which a robot can automatically determine how it is moving. Experiments demonstrate the ability of the algorithm to learn new models given models from other surfaces and robots. This work facilitates further research into creating complex formations using only local information and in fully automating current Simultaneous Localization And Mapping (SLAM) applications.

14. SUBJECT TERMS Simultaneous Localization And Mapping, Motion Model, Coordination, Online Parameter Estimation, Sony AIBO	15. NUMBER OF PAGES 87
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

AUTONOMOUS COORDINATION AND ONLINE MOTION MODELING FOR MOBILE ROBOTS

Eric J. Sjoberg
Captain, United States Marine Corps
B.S., Iowa State University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Eric J. Sjoberg

Approved by: Kevin M. Squire
Co-Advisor

Craig Martell
Co-Advisor

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Robots are rapidly becoming more involved in everyday military operations. As robots become more capable, their tasks will increase to include such roles as exploring enemy controlled buildings and caves. The goal of this thesis is to explore methodologies that allow robots to operate more autonomously. The first goal is to develop an algorithm that allows groups of robots to construct controlled formations with only local information. Experiments investigate the ability of this algorithm to handle obstacles, dynamic conditions, and varying number of robots. The second goal of this work is to demonstrate a method by which a robot can automatically determine how it is moving. Experiments demonstrate the ability of the algorithm to learn new models given models from other surfaces and robots. This work facilitates further research into creating complex formations using only local information and in fully automating current Simultaneous Localization And Mapping (SLAM) applications.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I would like to thank my loving wife, Chrissy. Your selfless and enduring support made this work possible.

To my family, especially my parents, who always encouraged me to further my education and whose example has taught me to always dig deeper and find the absolutes in life.

To my class and lab-mates, whose constant dialogue has kept me focused and reminded me of the usefulness of our collective work.

To Greg Ball, your ability to create code in your sleep and read other people's code astounds me. Thanks for all the help in getting much of the code used here in working order.

Last, but certainly not least, I would like to thank my advisors. Your knowledge and guidance has kept me on track and pointed me in the right direction too many times to count. Without your steerage, I am positive this work would have been worth half of what it is now and taken twice as long.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Since 1962 when General Motors installed the first industrial robot on its production line, robots have continued to take an ever more prevalent role in our everyday lives. In the last thirty years, advances in robotics have allowed robots to do everything from manufacture cars to explore neighboring planets to detonate ordnance on the streets of Iraq. The continued miniaturization of computer hardware has ushered in a wave of ever more capable robots. These robots have allowed researchers to continue to push the envelope of human-robot interaction and coordination, in a race to seamlessly integrate human and robotic life.

Figure 1. EOD robot working with US Forces in Iraq. (From: DOD PAO)

As robots continue to become more and more capable, we will find them taking ever more present roles in our

society. The US Department of Defense (DOD) is one of the major forces pushing the advances of robotics. For evidence of this, one only has to look at its budget. The rate of funding for Unmanned Vehicles (UVs) since the late 1980s has jumped from around $200 million to $1.9 billion in 2007, with projections out to 2009 jumping to nearly $3.0 billion [21]. The DOD's stated goal of this increase in funding is to allow "U.S. forces to pursue terrorists without putting our troops in harm's way."

One of the many projects funded by the DOD's push for autonomous systems is the DARPA Grand Challenge and the DARPA Urban Challenge [6]. The DOD desires to have vehicles of all types operate autonomously in order to reduce the risk to humans working many of the riskiest jobs in the DOD. These jobs currently consist of Explosive Ordnance Disposal (EOD), surveillance, and reconnaissance of named areas of interest. Automating these jobs not only decreases the risk to human operators, but in many cases increases the DOD's capability. The Global Hawk is a perfect example of this. The removal of the human operator from this platform allows it to sustain missions well beyond the endurance of any single human operator and eliminates the fear of losing an invaluably trained pilot in the case of a mishap.

Despite the big push for UVs, current robotic systems tend to be less than fully autonomous. That is most robots today have a human operator overseeing every action to ensure that the robot does not perform some undesirable action. As robots continue to become more robust, they will also become more autonomous and reliant on other robots to help complete tasks of ever-increasing difficulty.

A part of this effort is a desire to use groups of coordinating, mobile robots to provide services such as surveillance, mapping, or reconnaissance. Essential to every coordinating group of robots is the issue of physical arrangement. This issue can and often is resolved by designating one entity that tracks every element, determines where it should be situated, and then issues appropriate commands to each, as was done in [17]. This is not an optimal design case. For instance, what happens when natural or unnatural causes (such as a malicious attack) interrupt communication with the controller? Clearly, a central controller is a critical asset to the formation, without which, the group will be left in disarray until a new controller is designated. Being a critical asset to the formation, the controller is also a critical vulnerability. The enemy only has to disable it to render the entire formation ineffective.

A more optimal design would allow each robot in the formation to determine its appropriate place and move to that location without the oversight of a central controller. This would remove the vulnerability poised by having such a central controller and make the group better suited to operate in dynamic environments.

Formation control presupposes several conditions though, not the least of which is that each robot can locate itself in its environment and report its location to its neighbors. Obtaining this essential bit of information is much more difficult than it sounds. Although the Global Positioning System (GPS) can serve as a solution to this issue, there are many places where GPS data is not available, for instance, the insides of buildings and

caves. Certainly, there are also times when this information will not be available due to jamming, inclement weather, or other environmental conditions. It is therefore important that autonomous robots be capable of inferring their location without the use of GPS data. This requires each robot to conduct localization.

B. THE ISSUE OF LOCALIZATION

Every formation control algorithm comes with a price. A distributed algorithm requires that each entity be capable enough to locate itself in its environment and communicate with its peers. Until recently, robots of this capacity were hard to come by. Even so, localization is still a significant issue. Without a collective map, each entity needs to conduct both mapping and localization, a process known as Simultaneous Localization And Mapping (SLAM) which is harder than either problem alone. If SLAM is not an option due to hardware limitations, then at minimum the robots must be able accurately, and reliably locate their neighbors, which is another difficult task.

Either way, the robot must have some basic knowledge of itself. It must be able to ascertain how it moves in the environment given a certain command. Without this information, the robot cannot automatically make decisions about how to get from one location to another. This model, that describes how a robot moves given a specific command, is the robot's motion model.

Ideally, one could precisely define the motion model for a robot. For instance, if a robot were given the command to turn left 90°, it would greatly simplify the task of locating itself after the turn if the only information

that changed was the heading of the robot by exactly 90°. In reality, the odds are that the robot did not turn exactly 90° and that the x and y position of the robot's centroid also changed. Thus, it is most common to describe a robot's motion model as a probability distribution.

Obtaining the parameters of this distribution is a subject of much human involvement. Typically, a human researcher determines the motion model of a robot only after observing the motion of the robot after the vehicle has performed numerous commands of each type. This is a time-intensive process that typically must be repeated every time a different robot is used or the surface under the robot changes. Automatically determining the robot's motion model could prove to save a significant amount time and allow the robot to operate in a much more autonomous mode.

C. ROBOTIC APPLICATIONS

As stated previously, robots continue to find themselves more and more a part of our everyday life. Mobile robots with appropriate sensors can generate maps, conduct reconnaissance and surveillance, mine-sweep, and explore other hazardous environments without fear of injury. Mobile robot teams equipped with network antennae can form a mesh of antennae to provide network services for an area of operation. The advantage of using a decentralized formation control algorithm in any of these environments is that the group of robots can automatically heal itself after a change in environment or the degradation of a node is detected. If only local rules and

communication are used, the need for long haul communication and a robust controller also disappear.

As for the uses of automatically learning a motion model, this provides several benefits. First, it eliminates the time required to calibrate a new robot or to recalibrate an old robot to a new surface. This time saving can be considerable when one considers that it may take several days to collect the data for a single robot on a single surface. If a user wanted to use four robots of similar class to quickly map out the inside of a building, due to variances in each robot, that user may have to calibrate each robot individually taking days to prime the entire group. A much more efficient way would be to have each robot learn its own motion model autonomously or have one robot learn a base motion model and pass along that information to the other robots that could then adjust the base model as necessary.

Another advantage of continuously learning and adapting a robot's motion model is that this allows the robot to notice changes in its behavior. If the robot, somehow begins to malfunction, sustains damage, or experiences significant wear during its operation, its motion model may change considerably. For a continuously learning robot, this is not a significant issue. The robot will modify its model and be able to determine its ability to complete missions based on the model. The same is not true for an unlearning robot that will continue to believe it can achieve goals of which it is physically incapable.

One final benefit of continuous learning is that the robot can cross between thresholds of surfaces and automatically generate a new motion model for the new

surface. This very accurately duplicates how an intelligent being would behave in a similar situation.

D. OUTLINE OF THIS WORK

This thesis has two main objectives. First is to describe a generic, scaleable algorithm for the formation control of mobile robots. Virtual simulations and experiments test the validity of the algorithm and tests on real mobile robots are conducted to ensure the algorithm succeeds in the ever noisy, real world. Due to limited physical resources, simulation alone will test the scalability of the algorithm.

The secondary goal of this thesis is to introduce a method for automatically learning a robot's motion model. The algorithm will be employed on real mobile robots so as to ensure the method can learn an accurate model. Experiments are conducted to ensure that the learned motion model is a precise enough approximation of the robot's true behavior for every day use in robotic mapping applications.

The remainder of this document is structured as follows:

- Chapter II discusses other work related to this work. This includes a discussion on formation control, robotic mapping methods, and motion models.
- Chapter III details how an autonomous coordination algorithm was developed and tested in simulation and on real robots.
- Chapter IV describes a method for automatically learning motion models for robots and the results from experiments designed to test the algorithm.

- Chapter V highlights the conclusions drawn from the experiments ran in support of Chapters III and IV and provides recommendations for future research.

II. RELATED WORK

Understanding the significance of this work requires a remedial understanding of several topics in robotics. These include the use of groups and formations of robots to perform tasks, the importance of motion models, and the difficulties of performing SLAM.

A. FORMATION CONTROL

Formation control has been an area of interest in robotics for quite some time. Many formation control algorithms tend to be biologically inspired. The motivation here is to learn from the many species of animals naturally forming interesting and useful groups. For instance, many types of birds form up and manipulate each other's wake so as to reduce the collective effort required to make long, annual migrations. Many species of small fish tend school together to give larger, predatory fish the impression that they are one fish that is too large to be devoured.

This theme, that the "whole is greater than the sum of the parts," is the inspiration behind most biologically inspired formations. Rather than invent a completely new approach from scratch, many researchers have attempted to characterize and mimic naturally occurring formations.

As far back as the early 1950s, researchers have tried to characterize these natural formations mathematically. C. Breder described the interaction of schooling fish as a force system [4]. His work was largely devoted to characterizing the influence between individual fish and the collective group. He described the interaction as a set of forces, one attractive and one repulsive. At great

distances, the attractive force overrides the repulsive force thus causing the fish to come together. As the fish move closer, the repulsive force begins to take over to keep the fish from colliding. As one could expect, the fish naturally tend to come to rest at the equilibrium point where the repulsive and attractive forces equal.

In 1987, Craig Reynolds introduced his Bird-OID (BOID) objects designed to mimic a flock of birds in flight [22]. Reynolds focus was to simulate a flock of birds for a computer graphics simulation. After a detailed study of many flocks of birds, he designed a simple, three-rule system that at least visually appeared to model the flight of birds. His model included a cohesion rule that pulled the BOIDs together, a separation rule that prevented collisions, and an alignment rule that matched the heading of local flock-mates.

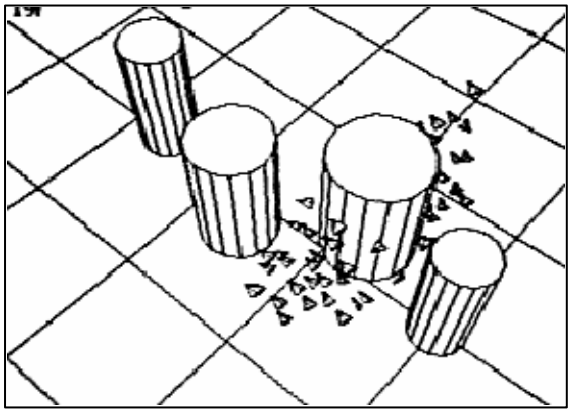

Figure 2. Reynolds' BOIDs in flight. (From: [22])

More recently, Sugawara and company demonstrated a simple force based system that formed tight formations of robots ([24] and [25]). Their method appears to model many types of group motion quite well, everything from marching, oscillating, wandering, to simple swarming. Their approach

is very much an engineering approach to formation control though. They described the motion of each agent by the interaction of four different forces: 1) a drag force, 2) a propulsive force, 3) a short-range interactive force, and 4) an attractive force towards the notional center of the group of agents. The first two forces are self-explanatory. The short-range interactive force is more complicated and is the heart of the algorithm. It defines a force that similar to Breder's force equation, causes agents to want to cluster together but not too closely. The final force ultimately causes all the local clusters of agents to form one large group. Admittedly, Reynolds' and Sugawara's algorithms can be quite expensive computationally as each agent must at minimum make a decision about whether or not to disregard every other agent in the environment. So for large numbers of agents, the load on each agent or the controlling entity can be quite demanding ($O(n^2)$).

The US DOD continues to explore the use of groups of autonomous systems for military purposes. In [17], Ludwig proposes using small swimmer vehicles to conduct robotic minesweeping. His control algorithm relies on a supervisor vehicle to maintain communication with each of the swimmers and provide tasking to each vehicle. His demonstration, although effective, is highly dependent on the supervisor vehicle to task and direct each robot.

The significant issue with all of these models is that they scale poorly and/or require some piece of global knowledge on the part of one or all agents. A generic, scaleable algorithm that provides for complex formations remains as of yet undeveloped. Such an algorithm could form topologies consisting of massive numbers of robots or

mobile sensors that can provide everything from area surveillance to mobile reconnaissance to combat attack formations.

B. MOTION MODEL ESTIMATION

Before any group of robots can autonomously achieve a formation, each robot must have some knowledge of how it moves in the real world. The motion model of a robot is a model of how the robot moves in the real world given an environment and a command. This is to say that the motion model of a robot allows the robot to determine its new position given its old position and a specified command. Represented as a probability distribution this is

$$P(\text{current\_pose} \mid \text{previous\_pose}, \text{command}, \text{environment}) .$$

Frequently this is represented by a Gaussian distribution,

$$P(\text{current\_pose} \mid \text{previous\_pose}, \text{command}, \text{environment}) \sim \text{N}(\mu, \sigma) ,$$

where μ and σ are the mean and variance of the distribution, respectively.

Typically, a human user provides this information to the robot. The user must observe and characterize the robot's motion over the set of useful commands. This process tends to be time consuming since each type of surface encountered and each robot employed requires calibration separately.

Automatically learning the motion model of a mobile robot has been a subject of interest for some time. Previous work in this area has largely focused on estimating the systematic error in wheeled robots. In [3], Borenstein and Feng introduced a systematic method for measuring and correcting the error in a robot's reported

odometry. In [18], Martinelli and company used a Kalman filter to automatically determine the error reported by wheel encoders on a differential drive robot. In [5], Caltabiano and company used an Extended Kalman Filter (EKF) to calculate the absolute position of and to estimate the odometry parameters for a six-wheeled, volcano-exploring robot.

In [23], Roy and Thrun demonstrated a statistical method for calculating the systematic and nonsystematic error in the reported odometry of a wheeled robot. Roy and Thrun's method is an online algorithm that uses calibrated sensors to determine the odometry error in the robot. Their method involves calculating the odometry error parameters that maximize the likelihood of the current observations given the immediately previous set of observations.

In [9], Elizar and Parr used an expectation maximization approach to calculate the values for a probabilistic model of odometry error. Their method is similar to the method used in [23] in that it determines the parameters that maximize the probability of the current set of observations. The difference from Roy and Thrun’s work is that Elizar and Parr's algorithm uses the sum of the observations made during the duration of the robot's operation to calculate the probability of a set of observations instead of just the last observations. Elizar and Parr also discuss running their algorithm over the entire data set so as to provide the best possible parameters over an entire data run.

As Table 1. highlights, a key assumption that the authors of [3], [18], [9], [23], and [5] all made was that the robot is wheeled and has the ability to report on its

odometry. These are characteristics that many high-quality, research-grade robots possess; however, not all robots maintain these properties and it is absurd to imagine that every robot should be. Therefore a more robust and flexible algorithm is necessary to deal with the more general case robot. Such an algorithm would allow the robot's motion model to change over the duration of the run and should not require that the robot report on its odometry. As of yet, very little work has been done in this area and demonstrating a method that can handle changes in surface in real time has also yet to be accomplished.

Author	Methodology	Assumptions/ equipment	Online?
Borenstein and Feng	Hand measured using a set of tests	Wheeled robot, odometry sensors	No
Martinelli	Augmented Kalman Filter	Wheeled robot, odometry sensros	Yes
Caltabiano	EKF sensor fusion	Wheeled robot, odometry sensors, DGPS	Yes
Roy/Thrun	Maximum Liklihood	Wheeled robot, odometry sensors	Yes
Elizar/Parr	Expectation Maximization	Wheeled robot, odometry sensors available	Partially

Table 1. Summary of Motion Model Estimation Methodologies

Additionally, no currently published work details a method robust enough to automatically estimate the motion model of a legged robot. Legged robots provide a much richer and potentially more complex motion model than wheeled ones. Learning such a model can prove troublesome

as each type of command must be modeled separately and interactions between commands can cause inconsistencies in the model.

Ideally, a learning algorithm should be robust enough to allow for the incorporation of an un-modeled robot. Such an algorithm could start with a rough approximation of the model, or even a guess, and observe the robot's motion over time. Given no configuration, health, or operating surface changes, the algorithm should converge to the robot's actual model. It may also be the case though, that the robot's motion model does change during the course of a run due to wear and tear, operating surface change, or battery condition. The algorithm should also track these changes.

C. SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

As previously mentioned, SLAM is a process by which a robot concurrently generates a map of its environment and locates itself in that map. Several techniques currently exist for doing this. These techniques vary largely in how they represent the environment and the robot's location. Kalman Filter based implementations tend to represent the environment as features typically acquired from visual sensors. Other implementations use occupancy grids to represent the floor plan of the environment the robot is exploring. A detailed explanation of each implementation of SLAM is not necessary here, but a cursory explanation of the SLAM algorithm used in this work is beneficial. Interested readers can reference [26] for a description of other methods.

The SLAM implementation utilized in this work is Distributed Particle SLAM (DP-SLAM). For a complete

discussion on DP-SLAM see References [7], [8], and [10]. DP-SLAM maintains a joint probability distribution over robot poses and maps using particle filters [26]. This allows the algorithm to maintain multiple estimates of where it believes the robot is situated and what the environment looks like.

Particularly key to understanding some of the latter work is how DP-SLAM represents where it believes the robot is located. DP-SLAM does this by using a predetermined number of particles, each one of which represents a guess as to the robot's current pose and how, if the robot were located at that position, its observations would fit into the current map. After the robot executes each command, DP-SLAM estimates the robot's motion by scattering the current set of particles using the given robot's motion model. Particles in the current generation that are more likely will generate more children particles than the less likely particles will. This helps prevent the algorithm from diverging by using more likely particles to create better estimates of the robot's new pose.

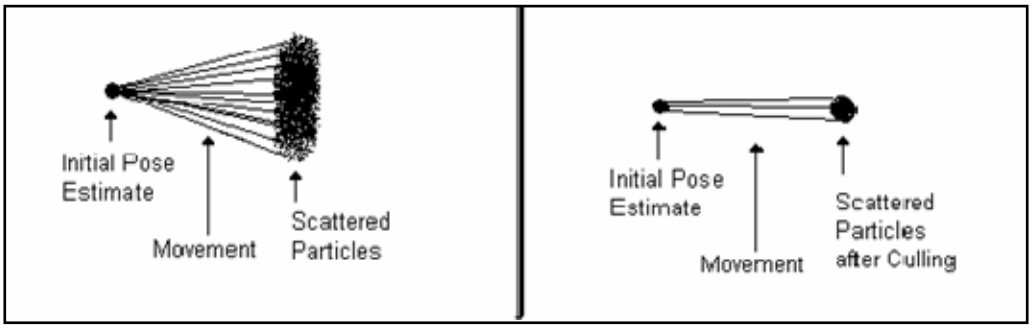

Figure 3. Particle Filtering Demonstration.

As the algorithm progresses, each of the particles is assigned an evaluation on how well its current observations fit in what is already known of the map. So if all of a

particle's observations fit well into what is known of the map, that particle is evaluated as more likely than a particle whose observations only partially fit well into the map.

If a particle does not meet some predetermined minimum threshold for “goodness", it is discarded or culled. At any given moment, the estimate of the robot's current position can be determined by viewing the weighted average of all the particles' location (Figure 3). The current map is similarly viewed, by examining the sum of the observations made into the occupancy grid. This ends up appearing as a two-dimensional map where each grid square maintains a likelihood that it is occupied. These likelihoods can be scaled and viewed as an image (Figure 19).

By now, the reader should have a cursory understanding of the key work in coordination, SLAM, and motion model estimation that has led up to this thesis. This thesis aims to use this work as a stepping-stone to create more robust methods.

THIS PAGE INTENTIONALLY LEFT BLANK

III. AUTONOMOUS COORDINATION

A. MOTIVATION: A SIMPLE SYSTEM

The goal of this portion of the work was to create a control algorithm that can realize complex formations while explicitly denying the use of global information. The inspiration for this algorithm, like many of the current methods used in robotics comes from an underlying principle in the natural world, opposing forces. At the atomic level, the strong and the weak force interact to create a balanced system. In astronomy, gravitation and a centripetal force balance each other to create a sort of harmony. In pack animals, we also see what appears to be a two-force system. The animals draw each other together but at distances that encroach on each other's personal territory an opposing force establishes equilibrium. This is precisely the interaction that this algorithm should model. It seems in nature that molecules, planets, and animals are all capable of creating well-ordered systems without the continuous intervening of a controlling entity to restore order.

To implement such an algorithm requires only that each entity somehow determine the appropriate level of interaction from its nearest neighbors. If the neighbors are too close, the agent should move away. If they are too far away, the separation should close at an appropriate speed. The essential task is determining how close a robot's neighbors are. This information can be based on where the robot, itself, perceives its neighbors are in relation to itself or from the robot's belief of where it and where its neighbors believe they are in a shared coordinate system. The former stipulation requires each

robot to accurately identify each of its neighboring robots and determine their distance from itself. The latter stipulation requires that the team of robots share a common map or at minimum, a common reference system. Although both may, at first, appear to be simple enough tasks, neither is trivial.

If each robot has a functioning Global Positioning System (GPS), localization is essentially free. However, functioning GPS is not always a valid assumption. For instance, consider a military unit tasked with gaining control of a building. The unit most certainly could use a current floor plan and information concerning the occupants and contents. A team of robots designed to survey the area, using an algorithm to maintain proper spacing, may run into several problems if they were to rely on GPS sensors alone. First, the GPS sensors may be jammed or unavailable due to inclement weather. Second, the satellite cluster’s signal may not penetrate the walls and ceiling of the building. A final problem is that the resolution of even military grade GPS coordinates may not be fine enough to ensure the robots are properly spaced. Since obtaining a shared map and a universally agreed upon reference system is still nontrivial, further work in this section is done under the assumption that at a minimum the robot's absolute position is known. A detailed discussion later should make it clear to the reader that this information is not essential but merely simplifies the task of validating the algorithm.

B. COORDINATION ALGORITHM IN DETAIL

As described above, a two-force system is the heart of the coordination algorithm. Table 2. outlines the algorithm as executed by each agent.

Step Number	Pseudocode
0	At regular time interval T DO:
1	Establish a list of N nearest neighbors.
2	Calculate the net force from the sum of the individual interactions for the N neighbors.
3	Move in the direction of the net force.

Table 2. Coordination Algorithm Pseudocode

Step one requires each agent to either properly identify its neighbors with its own sensors or locate itself in a globally shared reference system and communicate its position to it neighbors. In many situations it is feasible for an overseeing agent to use its own sensors to locate each of the entities in the environment and distribute the list of locations to the mass. In this case, though, the algorithm is not decentralized and would suffer the woes of the previously mentioned methods that used global knowledge.

In step two, the force interaction between each agent is determined and summed to provide a net force. For the experiments done in support of this algorithm, the interaction force between agents was determined using:

$$force = k\left(d^2 - \frac{(\text{desired\_spread})^4}{d^2}\right),$$

where d is the distance between two agents, desired_spread is the desired separation between agents, and k is a predetermined coefficient. As can be inferred from the equation, positive forces are pulling forces and negative forces are pushing ones. This force equation defines a pulling force that increases as the square distance between two agents increases from the desired spread. As the distance between two agents drops below the desired spread, the force equation creates a rapidly increasing pushing force. Figure 4, below, shows the magnitude of force generated between two agents when the desired spread is 50 units.

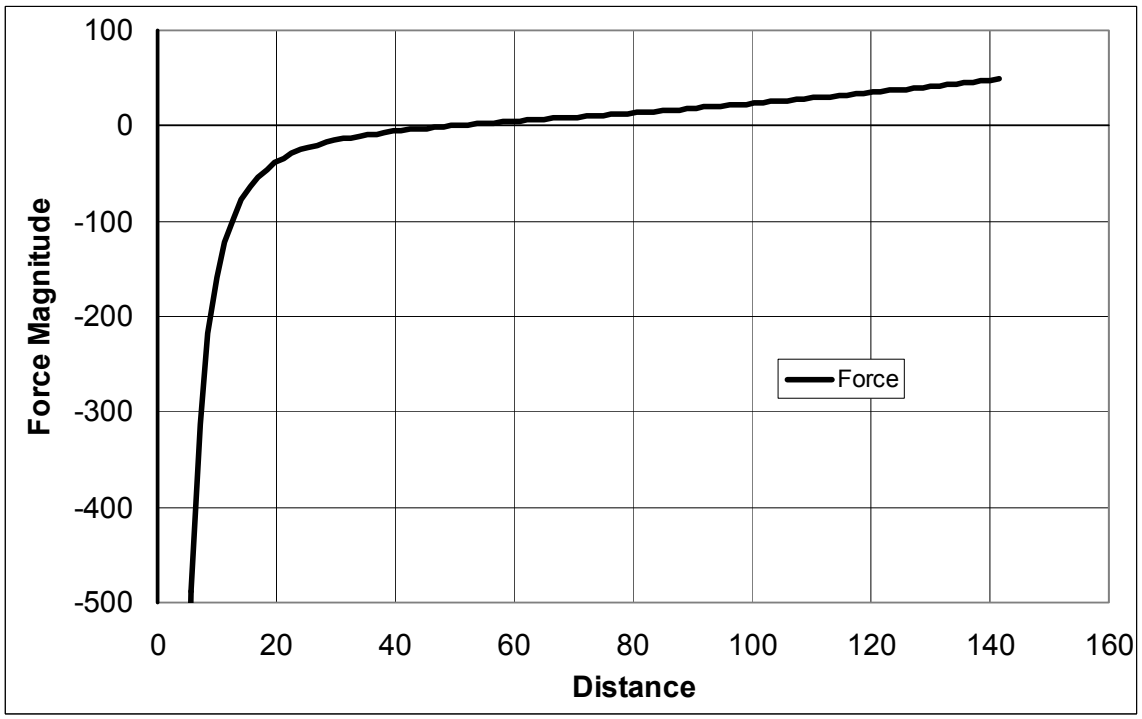

Figure 4. Graph showing interaction force between two agents.

The salient point for the force equation utilized in this work is that the pushing force rapidly and asymptotically approaches infinity as the distance between agents approaches zero. This is a desired trait, as it will keep the agents from colliding even if several other agents are pushing or pulling a single agent into one of its neighbors. It is also important to note that the use of the equation highlighted above is not required. Instead, the implementer can use any system that balances the forces between agents at the desired distance. One such example is the interaction force used in [24],

$$force = -c\left[\left(\frac{|d|}{\text{desired\_distance}}\right)^{-3} - \left(\frac{|d|}{\text{desired\_distance}}\right)^{-2}\right]\frac{d}{\text{desired\_distance}}.$$

In step three, the robot takes measures to move in the direction of the net force exerted on it by its neighbors. In all realms where this algorithm was tested, it was useful to ignore sufficiently small forces. Allowing the agent to ignore small forces effectively creates a range of distances where the agent is close enough to the desired separation. Without this property, the formation would often exhibit excessive jittering as the formation of agents congealed and agents tried to position themselves with precise separation.

The final item of discussion is how often to update the force felt by each agent. Clearly one would like to keep this force updated continuously; however, this is not practical. As a lower limit, the updates should occur regularly enough so that agents do not collide. A better constraint is to perform updates such that each agent update the force on it at least once while it is in the

area where the net force is considered negligible. This will keep agents from transiting through the ideal area and subsequently bouncing back and forth across.

C. SOFTWARE IMPLEMENTATION

As an initial proof of concept, an agent that implemented the coordination algorithm was developed in the virtual world of Robocode [14]. Robocode is a virtual world where user programmed tanks battle to the death. This implementation proved that the algorithm did create desirable formations of agents. Since Robocup's environment and agents are not open to easy modification, the author developed a new, Java-based simulation environment inspired by Robocode's design, AgentSim.

In AgentSim, agents and any desired obstacles inhabit a simple rectangular environment. The environment provides several vital functions. In order to simplify localization and visualization, it keeps track of all the agents and their absolute positions. Each agent merely queries the environment to determine its current location. The environment also receives communication messages from each agent and passes messages onto the appropriate agents. The final function that the environment provides is notifying agents when they have physically encountered either an environmental boundary or an obstacle.

In AgentSim, each agent is a separate object with its own thread for execution. Each agent is wholly contained and only receives precepts directly from the environment to simulate how a real autonomous agent would act.

AgentSim allows the user to configure many properties at start-up. The number of agents in the environment, the desired spread of each agent, the number of neighbors each agent tracks, the size of the environment, the presence of obstacles, and the size of the force to be considered negligible are all configurable during start up. This allows many different parameter settings to be rapidly tested. A final feature that AgentSim offers is the ability to dynamically remove and restore agents. This allows the user to test the algorithm in a dynamic environment where agents can be disabled and restored at the user's discretion.

Tests examined how all of the different parameter configurations effected the formation. Formations containing as few as three agents and up to as many as 200 were formed. Obstacles restricted the agent's movement in an effort to see if restrictions to motion would effect the execution of the algorithm. Results from the simulation runs proved highly successful.

D. ROBOTIC IMPLEMENTATION

After testing the algorithm in a virtual environment, it was time to test it on a real, robotic system, the Sony AIBO ERS-7M2. Due to limited assets, the algorithm could only be tested on four robots simultaneously. Since GPS was not an option on these robots and SLAM out of the research group's grasp at the time, some simplifications were applied in order to facilitate testing.

Figure 5. One of the Sony AIBO ERS-7M2s used in testing.

The primary issue encountered while implementing the algorithm on live robots was localization. To simplify the matter, each robot was provided a map of the operating environment. Visually distinct beacons occupied known locations in the environment. This greatly reduced the problem of localization since each robot now just had to visually locate only two of the beacons and then use simple triangulation to locate itself. Since each beacon was visually distinct, accurately and reliably locating each was achievable using simple color segmentation.

As a further test, the algorithm was modified slightly to allow the user to specify a goal for the robots to converge toward. This provided the formation with a function other than simply forming a cluster. The objective here was to create a regular formation centered on a specified location that the robots would survey. Thus, the robots would form a perimeter around the goal. The goal location was implemented as a virtual robot that only

exerted a pulling force on every agent. The force increased with the distance between each agent and the goal such that the furthest agent felt the most pull towards the goal.

The final simplification made was to perform all visual processing on an off-board system. Since AIBO has limited memory and processing power, it proved easier to conduct the visual processing and triangulation pieces on a desktop machine where memory and CPU usage are not as restrictive.

E. ALGORITHM EXTENSIONS FOR COMPLEX FORMATIONS

The pseudocode for the formation control algorithm is very simple and straightforward (Table 2.). Its function is similarly minimal; it merely spaces agents out in a controlled manner. To add further functionality is uncomplicated. For instance, the goal swarming behavior described above is simply achieved by inserting one more step in between steps two and three of the pseudocode. In this new step (Table 3.), each agent calculates the pulling force from the goal and adds that force to the net force acting on the agent.

Implementing other modifications are as simple as adding an additional step in the same location that modifies the net force on the robot in the desired way. Initially, this work focused on creating such implementations and demonstrating how complex formations such as columns, lines, and wedges could be formed using only local knowledge. Difficulties in demonstrating these capabilities on real robots due to localization issues led to work in the realm of automatically generating motion models (Chapter IV).

Step Number	Pseudocode
0	At regular time interval T DO:
1	Establish a list of N nearest neighbors.
2	Calculate the net force from the sum of the individual interactions for the N neighbors.
3	Calculate pulling force from goal and add to net force.
4	Move in the direction of the net force.

Table 3. Modified pseudocode for goal swarming.

F. RESULTS

1. Number of Neighbors

This first parameter explored in the formation control work was the number of neighbors to track. As Figure 6 shows, tracking too few numbers results in incomplete formations. Varying the number of neighbors tracked from one to ten showed that tracking fewer than four neighbors tended to lead to undesirable results. Agents tended to cluster in small, isolated groups oblivious to other formations, or in the case of tracking three neighbors, form groups with obvious gaps.

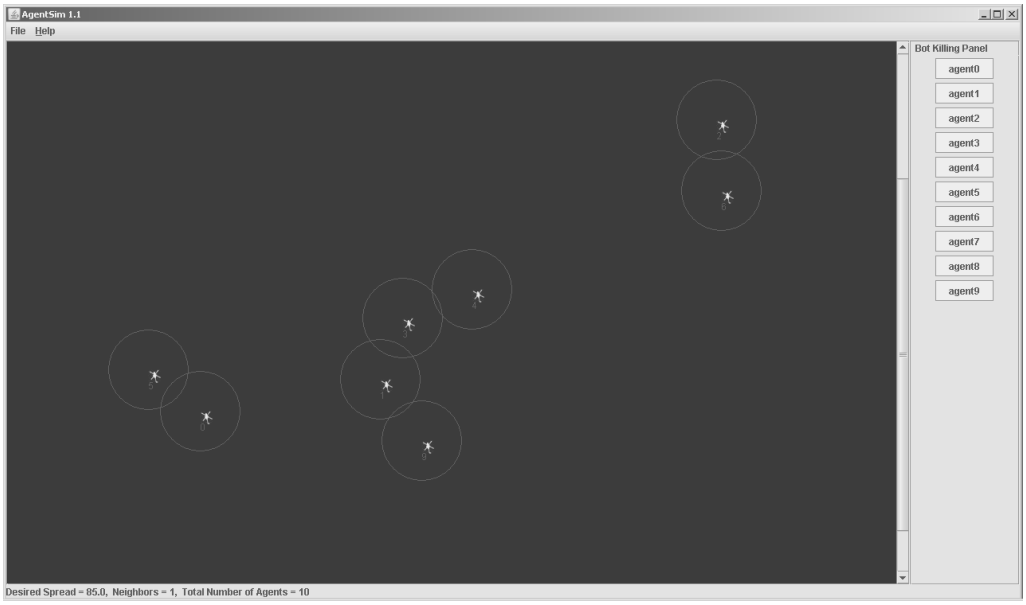

Figure 6. Formations created when too few neighbors are tracked.

When four or more neighbors were tracked, optimal formations regularly occurred. Figure 7 shows one such optimal formation using ten agents each tracking four of its nearest neighbors. This formation is ideal since the agents, each starting in a random location, have formed a lattice like structure on their own and can provide uniform sensor coverage of the area around the group.

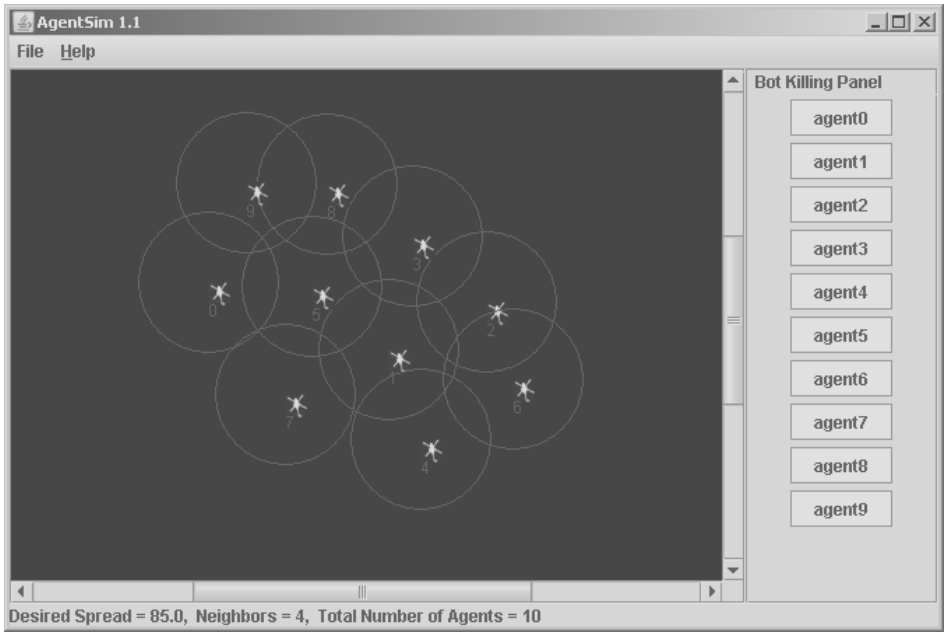

Figure 7. Optimal formation created from tracking the four nearest neighbors.

As the number of neighbors tracked increased toward the high extreme, the total number of agents in the simulation environment, the following phenomenon was observed. Whereas formations that tracked four agents maintained proper spacing, formations that tracked more than four tended to collapse into tighter than desired formations. Figure 8 shows the result of changing the number of neighbors tracked from three to nine. The resultant collapse is not altogether unexpected. If one views the system from the standpoint of a single agent on one of the sides of the formation it becomes clear what is occurring. This agent is treating all the other agents as neighbors and although its two or three nearest neighbors may be pushing against it, the rest of the formation is pulling this agent towards the center of the group. Consider that the same thing is happening on the other side of the formation and it becomes obvious that the net

pulling force from the formation is overcoming the pushing force from the few closest neighbors causing the inter-agent distance to become compressed.

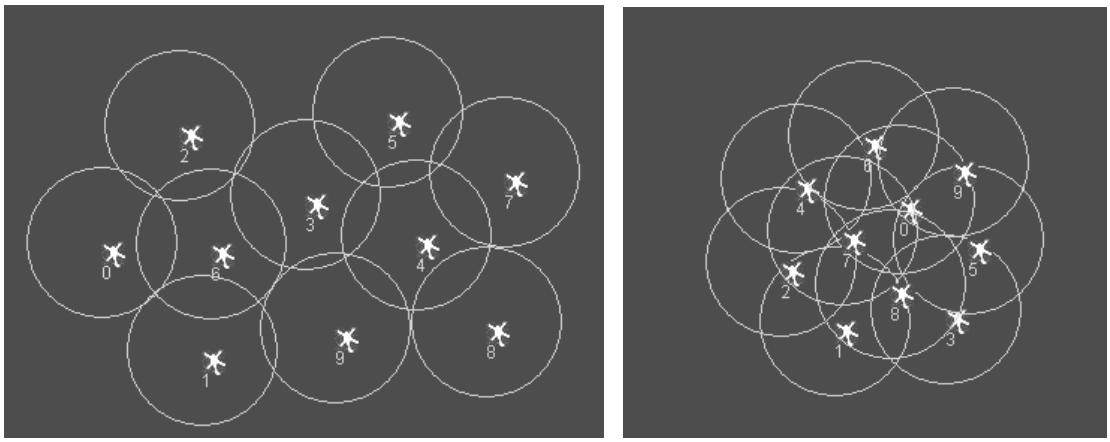

Figure 8. Formation size difference as a result of changing the number of neighbors tracked from three (left) to nine (right).

2. Creating Complete Formations

As mentioned above, an issue discovered when conducting experiments into the ideal number of neighbors tracked was the creation of incomplete formations. Figure 6 shows this occurring. Since the agents are only tracking a single neighbor, it is very easy for them to form isolated groups. This phenomenon is not isolated to agents tracking one neighbor. Groups of agents tracking three or four neighbors and started in large but distant clusters tend to form isolated groups instead of one large group. The issue here is an issue of balance.

If the agents are naively tracking the three closest neighbors and a given agent has three close neighbors to its right side, the agent will, of course, pick the three neighbors to its right and not the distant one to its left. This causes isolated groups. To resolve this issue a simple

fix was applied. The algorithm was modified such that the agents divide the space surrounding them into N sectors. So if N=2, the agent would create two semi-circles and if N=4, it would create four quadrants around itself. The agent then picks the nearest agent in each sector. If there are no observable agents in a sector, as is the case for every agent on the outskirt of the formation, the agent picks the rest of its neighbors from other sectors. This simple solution led to agents that congealed into one large group with no gaps in sensor coverage.

3. Obstacles

The next step investigated in this line of work was the ability of agents to form up in the presence of obstacles to their motion. For this, simple obstacles were placed in the simulation environment at random locations. The agents were able to from one, cohesive group in the presence of a stifling number of obstacles. However, since no path planning was implemented in these agents, situations did arise when agents could not form one complete group. This occurred when an agent was prevented from moving directly towards its nearest neighbors by 'L' or 'V' shaped groups of obstacles. In this case, some path planning is necessary to get a trapped agent out of its currently confined stated.

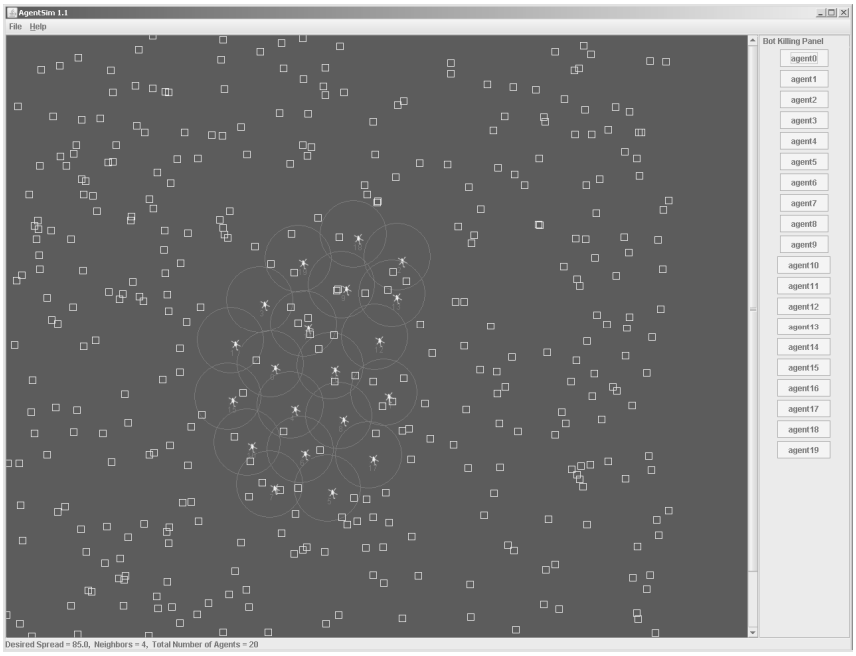

Figure 9. A group of 20 agents forming up in the presence of 400 randomly placed obstacles.

4. Removing and Restoring Agents

Another key aspect of a good coordination algorithm is the ability to seamlessly handle the removal and addition of agents. As would be expected, this algorithm handles these cases quite well. When an agent is removed, the only immediately affected entities are its nearest neighbors. Each of these agents merely picks the next nearest agent as one of its nearest neighbors. The formation quickly closes any gaps that created by the loss of the agent.

When agents are restored or added to the environment, similar results are seen. The formation quickly and seamlessly assimilates these agents into the group, regardless of whether they initially appear on periphery or in the heart of the group.

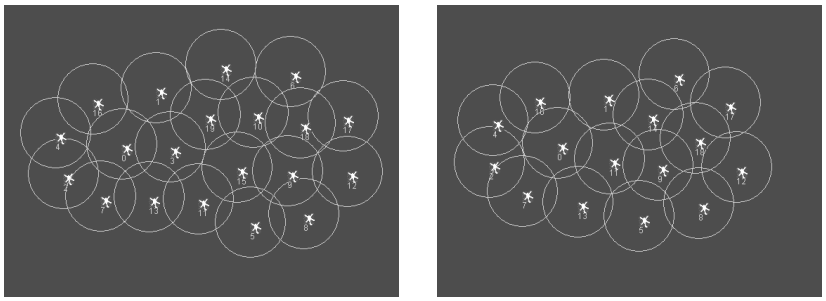

Figure 10. Formation of 20 agents before (left) and after (right) the four agents in the center are rapidly removed.

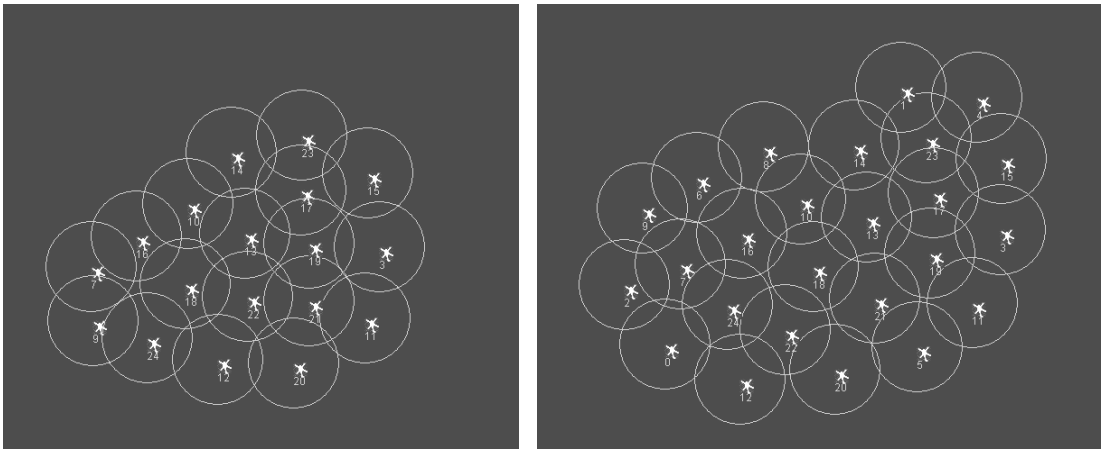

Figure 11. Formation of 19 agents before (left) and after (right) six agents are added to the environment on the periphery of the formation.

5. Robotic Implementation

Since initial tests in a simulated environment proved successful, experiments on real robots were conducted in order to determine if agents operating in the real world could achieve the same results. Initially tested was a very crude implementation that consisted of four Sony AIBOs on a floor with a grid. Each AIBO started in a random position in the grid. A human observer would then provide each agent with its position in the globally shared reference system and each AIBO would then determine its neighbors, calculate the resultant force, and move in the direction of the

force. The four AIBOs consistently formed up as the agents in the simulation environment indicated they would. This success led to the test of the goal swarming behavior mentioned earlier in this chapter. To test this behavior, the AIBOs were provided the coordinates of a notional goal location that each agent treated as a neighbor. The result was that the formation of AIBOs materialized, centered on the goal location.

There are two major issues with using a human observer to provide locations. First, it requires a considerable amount of time for an operator to accurately determine the robot's location at each time step. Second, the robots are in no way autonomous. To resolve this issue a shared map was constructed and provided to each AIBO. The map had several, visually-distinct landmarks, and using some rudimentary computer vision techniques, each AIBO could locate itself by determining its position relative to at least two of the landmarks. This, indeed, sped up the robots' ability to form up; however, being provided such detailed information about the map is also undesirable since again, each robot is not truly autonomous in that it still relies on a human to provide it with a complete map.

To increase the level of autonomy the agents exhibited, the issue of multi-agent SLAM broached. In a multi-agent SLAM environment, each robot must generate a map of an unknown environment, locate itself in the map, and then assemble a combined map from the maps generated by itself and the other agents. The idea was that this map could be utilized as a reference system in which the AIBOs could demonstrate the capabilities of the control algorithm. Faced with the prospect of multiple surfaces and

extremely noisy motion models, the research into online learning was undertaken so that the AIBOs could effectively and autonomously operate in the control environment.

IV. MOTION MODEL ESTIMATION

As mentioned in Chapter III, at the outset of this work, the focus was to generate complex formations using only local information. In order to do this, efforts were made to perform multi-agent SLAM such that a universally shared map was generated. Each robot would utilize this map to localize itself and thereby obtain the necessary information for the robots to construct complex formations. A fundamental component in performing SLAM is generating motion models that accurately describe how the robot moves in its environment. Obtaining such models for the robot of choice proved particularly challenging.

A. HAND MEASURED MODELS

The first step in measuring a motion model is determining the coordinate system. One of the most convenient ways to measure a motion model is to use a coordinate system centered on the robot. This way changes in the robot's position can be measured directly and updating the robot's location in the global coordinate system just requires some simple geometry. For this work, the robot's position in the global coordinate system is represented by $\phi = (x, y, \theta)$ where x and y are the Cartesian coordinates of the robot and θ is the heading of the robot. Each command the robot executes moves the robot in the direction of the current heading (D) and in the direction perpendicular to the robot's heading (C) and modifies the heading of the robot (T). Figure 12 shows how D, C, and T related to changes in the robot's position.

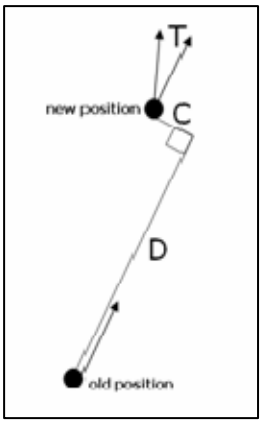

Figure 12. Motion model parameterization.

Given a robot's pose at time t, the robot's pose at time $t+1$ is

$$\begin{aligned} x_{t+1} &= x_t + D\cos(\theta_t) + C\sin(\theta_t) \\ y_{t+1} &= y_t + D\sin(\theta_t) + C\cos(\theta_t) \\ \theta_{t+1} &= \theta_t + T \bmod 2\pi \end{aligned} .$$

1. A Simple Solution

Initial efforts to capture AIBO's motion model were very crude. The robot executed less than 20 commands of every type. The change in the robot's location was then hand-measured using a reference leg to determine D and C. T was estimated by lining up a yard stick to the centerline of the robot and determining the change from the original heading. For each command, the robot was returned to a starting position and reset for the next run.

Several issues arose from this method. First, accurately measuring the parameter T relied too much on the human eye. As a result, the recorded behavior of this parameter contained more variance than was actually true. Second, resetting the robot after each command introduced a significant source of noise. It turns out that the process of resetting causes the position of the leg joints to

change. This change unnecessarily modifies the behavior of the robot's next command and does not accurately capture how the robot behaves when given successive commands of the same type. Lastly, but certainly not the least of which, the number of commands given was far too few to accurately capture the robot's behavior.

After capturing all of this data, a simple Gaussian idealized the robot's motion model. The decision to use a Gaussian was based on convenience and a lack of data suggesting a different distribution was necessary. Despite all the issues discussed above, using this crude motion model in DP-SLAM still provided decent maps. In order to get these maps though, several thousand particles had to be used to represent the robot's position. In addition, the variance in each parameter had to be increased ten-fold from what the hand-recorded values were. In most cases, this allowed the poor model to approximate what the robot was actually doing.

More often than desirable though, the poor model would not accurately capture the robot's true motion, and the robot would become lost and begin creating poor, inaccurate maps. In some cases, the DP-SLAM algorithm could not create accurate estimates of the robot's pose at all and would subsequently misplace an entire set of observations, creating ghost walls that did not exist. In other cases, DP-SLAM would select poses that made what should have been straight walls appear to be curved (Figure 13).

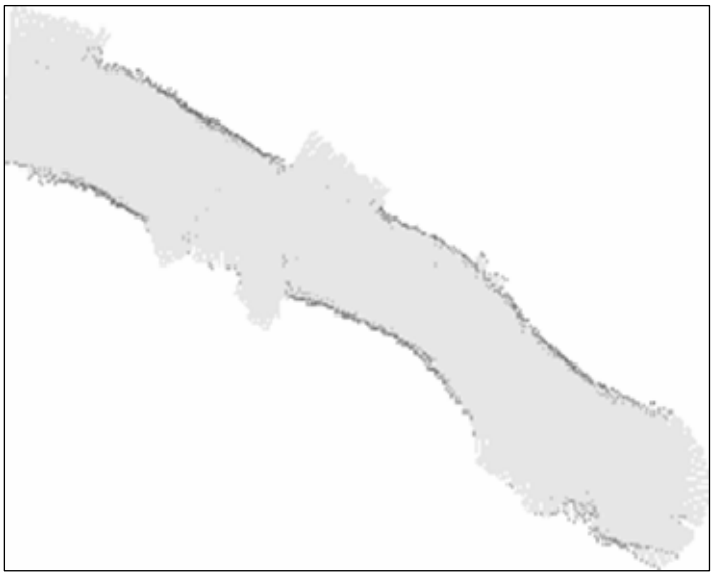

Figure 13. A crooked hallway, the result of a bad motion model.

2. More Detailed Efforts

In order to use fewer particles, estimate variances that are more accurate, and eliminate curved and ghost walls, it became obvious that a more representative motion model was necessary. In order to rectify the issues that occurred in the initial motion model capture, several corrections were needed. First, the robot had to execute more iteration of each command type. For the motion model recapture, 50 iterations of each run were made. This provided more data to analyze and characterize the robot's actual behavior. Second, AIBO performed commands in succession. Instead of resetting the robot each time, the robot executed one command right after another. Finally, to better capture the robot's position, a small Allen wrench was suspended from the robot's belly (Figure 14). This provided a convenient and stable method for measuring the change in robot's position and heading. To standardize the measurement process, after each command a small piece of

tape recorded the location and orientation of the Allen wrench (see Figure 15 and Figure 16). Great care was taken to ensure the Allen wrench did not affect the robot's motion by dragging on the ground.

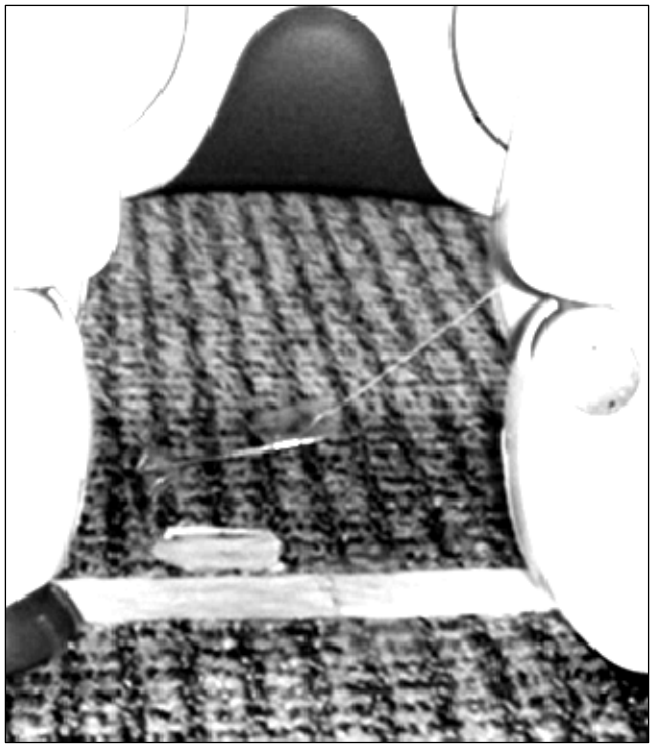

Figure 14. Close up of hanging Allen wrench.

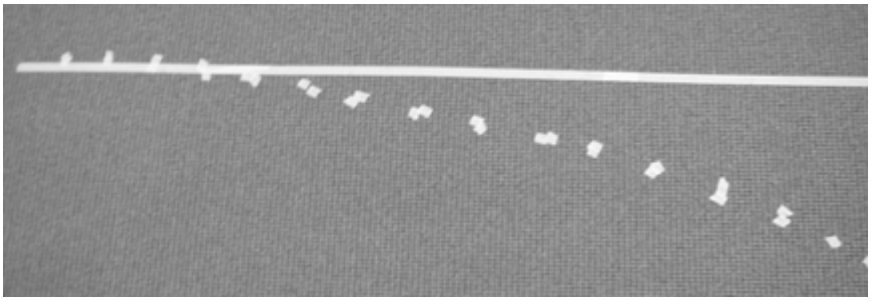

Figure 15. Tape markings used to record series of commands executed by AIBO.

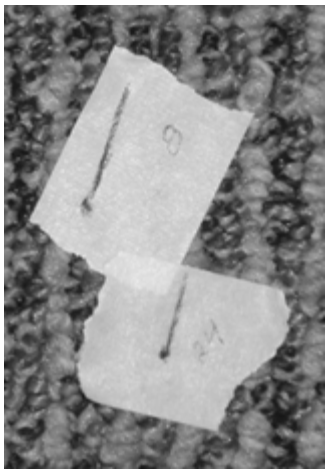

Figure 16. Close up of tape marking used to capture AIBO's motion.

To further improve the model, it was no longer assumed that AIBO's motion model could be approximated by a simple Gaussian. In fact, a histogram of several parameters exhibited bimodal behavior (Figure 17) while others appeared skewed (Figure 18).

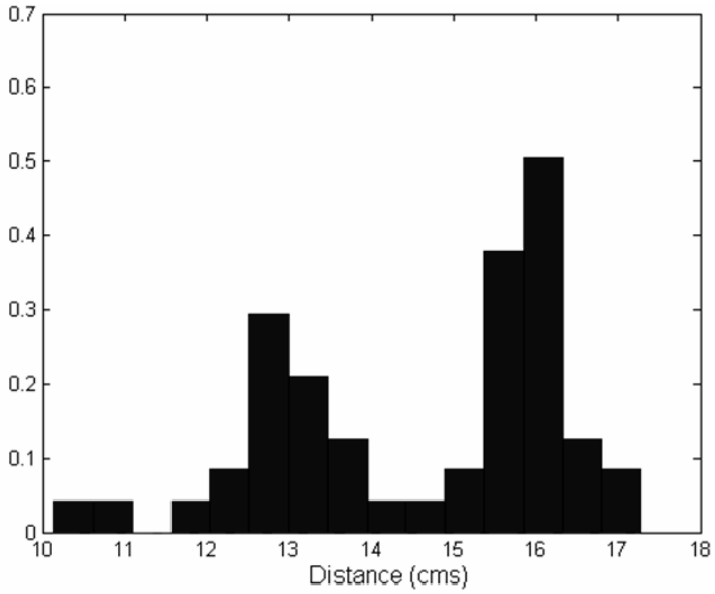

Figure 17. Histogram of the D parameter for the swalk(2) command on a carpeted surface showing a bimodal distribution.

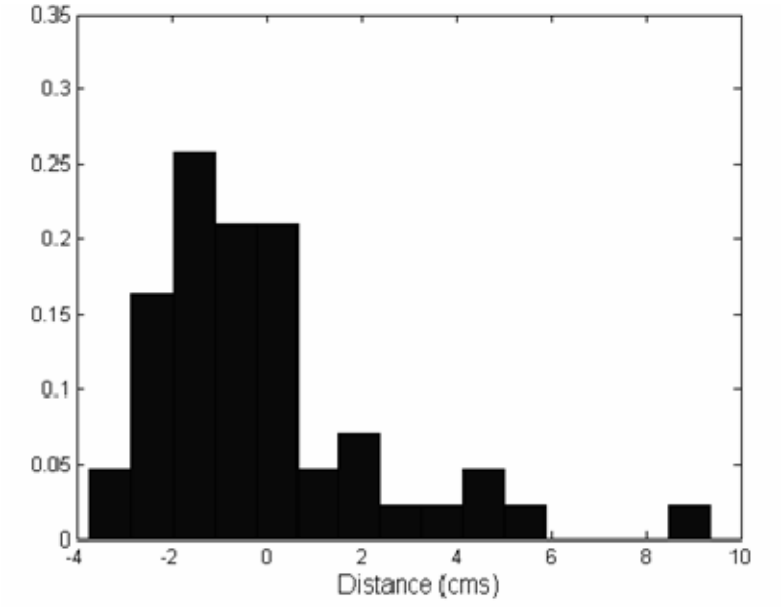

Figure 18. Histogram of the C parameter for the swalk(3) command on a carpeted surface showing skew.

In the case of parameters that showed bimodal behavior, it is particularly unsatisfactory to idealize these by a Gaussian. Figure 10 shows an example of this issue. The mean of this distribution is approximately 14.6cm. Modeling this parameter by a Gaussian means that most of the particles generated by DP-SLAM will have values of the D parameter near 14.6. However, this is precisely where the trough of the measured distribution lies, and DP-SLAM should generate few particles here.

In order to keep the DP-SLAM algorithm as simple as possible, it was decided to model all of the different observed distributions by a single type of distribution. To determine the best general distribution, we used a maximum likelihood approach to compare the following distributions: Gaussian, a mixture of two Gaussians, uniform, Johnson Su, Johnson Sb, and the Generalized Lambda Distribution (GLD). The latter three distributions were chosen because they

have the ability to model many different types of distributions using just four parameters.

The first step in determining the best distribution was calculating the parameters for the tested distributions. For the uniform and Gaussian distributions, this process is well defined and straightforward ([13]). Matlab provides a built-in function that calculates the Johnson Su and Sb parameters. The parameters for the mixture of Gaussians required expectation maximization. Reference [19] describes a useful technique for calculating the parameters of a mixture of two Gaussians. Obtaining the parameters for the GLD requires numerical methods to calculate its four parameters based on the mean, variance, skew, and kurtosis of the data. There are known data sets that cannot be modeled by the GLD. As it turns out, this was the case with several sets of data collected from the AIBO; therefore, the GLD was eliminated as a candidate for the best general distribution.

Once the parameters for each distribution type were determined, each distribution was rated by calculating:

$$\prod_{i=1}^{n} P(data_i \mid distribution),$$

where n is the total number of data sets modeled. The best distribution is the one that maximizes the likelihood of the data:

$$best\_distribution = \arg\max_{distribution} \sum_{i=1}^{n} \log(P(data_i \mid distribution)).$$

This measure is not a standard method of rating a distribution's goodness of fit. A more commonly used measure is the Chi-Squared Goodness of Fit, defined as:

$$goodness\_of\_fit = \sum_{i=1}^{k} \frac{\text{observed}_i - \text{expected}_i}{\text{expected}_i},$$

where observed_i is the count associated with bin i in the histogram of the observed data, and expected_i is the count associated with bin i for the idealized distribution, and k is the total number of bins of data in a given data set. Conveniently, the Chi-Squared measure and the maximum likelihood method provided the same determination as to the best distribution that fit the data.

As indicated earlier, the Gaussian was not the clear winner. In many cases, the Gaussian was actually worse than a uniform distribution. Although no single distribution worked best for all the different distributions, the mixture of Gaussians was better than all other distributions for over 90% of the datasets. The mixture was able to model parameter distributions that were bimodal, nearly uniform, and unimodal with skew.

As a side note, during the process of collecting data on the Aibo's motion model, it was noticed that Aibo's motion seems highly dependant on the current position of its legs. Its legs' current configuration is a direct result of the previous command given. Even giving the robot a StandUp() command after each movement was not enough to correct this issue. Tolerances in Aibo's servos prevent the joints from reaching exact positions given even a series of repeated commands. The high number of joints and loose tolerances in Aibo make it particularly difficult to estimate its motion model.

After determining that a mixture of Gaussians was the best idealized distribution, DP-SLAM was modified as such

to use AIBO's hand measured motion model. As expected, this better model captured AIBO's true motion and generated accurate robots poses and representative maps for the desired surface. The issue now was that this model took over 40 person-hours to collect and was only valid for the specific AIBO and surface from which the data was collected. In order to perform multi-robot mapping with the remaining three AIBOs on the two different surfaces present in the laboratory's building would require an additional 280 person-hours just to collect the motion models.

B. ONLINE LEARNING AND TRACKING

To eliminate the prerequisite of hand measuring complete motion models for the set of robots on all of the surfaces of interest, a learning algorithm was sought. Ideally, each robot could learn its own motion model automatically with no human input. However, this is not a realistic expectation since learning requires a source of valid feedback. Without feedback, the robot may learn a completely incorrect model, the exact situation trying to be avoided. To simplify the issue of learning a complete model from scratch, it was deemed an appropriate “cheat” to allow the robot to start with the hand-recorded model and modify the hand-measured model as the robot moved. This further begged the issue of obtaining valid feedback.

In the DP-SLAM algorithm, the quality of an estimate of the robot's pose can be determined by observing how well the robot's current observations fit into what is known of the map. This is an essential step in the DP-SLAM algorithm. Particles that do not have a high quality are poor representatives of the robot's current pose and are

typically culled. If the sum over all of the particles' quality is used to normalize the quality of each particle, the result can be viewed as a probability distribution over particles. This is so since

$$\sum_{i=1}^{N} \frac{q(x_i)}{\left(\sum_{i=1}^{N} q(x_i)\right)} = 1, \text{ and}$$

$$\forall j \left(0 \le \frac{q(x_j)}{\left(\sum_{i=1}^{N} q(x_i)\right)} \le 1 \right),$$

where $q(x_i)$ is the quality of particle x_i and N is the total number of particles. If the generation of quality values for each particle is in fact valid, the distribution of qualities for a set of particles generated during a DP-SLAM iteration will probabilistically describe how the robot moved during the previous time step. This information provides an appropriate method for the robot to understand how it is moving provided one stipulation, the initial motion model generates particles that at least partially represent the robot's motion. If not, the DP-SLAM algorithm may get hopelessly lost, build an invalid map, and use the invalid map to improperly score particles.

To allow the robot to learn its own motion model, the following algorithm was developed to update each Gaussian density for each of the motion model parameters: D, C, and T. First, the overall quality of the set of generated particles is calculated. This quality determines how much the current set should update the model, compared with previous and future sets of particles. The quality measure is invaluable when the robot moves into an area of sparse

sensor readings, as the total quality will be low and therefore should not account for much of the model update. In data rich environments, the total quality of particles will be much higher and will have a more pronounced effect on the model update. The measure of quality we calculate is the sum of all the measures of quality for the N particles generated at time t,

$$g_t = \sum_{i=1}^{N} q(x_i) \ .$$

Next, it is necessary to track the total weight used to update the distribution,

$$G_{t+1} = G_t + g_t \ ,$$

in order to determine how much the current particles set should update the model.

The parameters that describe the motion of the particles can be calculated directly from the particles using

$$\mu_{new} = \frac{1}{g_t} \sum_{i=1}^{N} q(x_i) m_i$$
$$\sigma^2_{new} = \frac{1}{g_t} \sum_{i=1}^{N} \left[q(x_i)(\mu_{new} - m_i)^2 \right]$$

where $m_i \in \{C, D, T\}$ that particle x_i move with at time t. Finally, the motion model updated by an exponential estimator:

$$\mu_{t+1} = \frac{G_t}{G_{t+1}} \mu_t + \frac{g_t}{G_{t+1}} \mu_{new}$$
$$\sigma^2_{t+1} = \frac{G_t}{G_{t+1}} \sigma^2_t + \frac{g_t}{G_{t+1}} \sigma^2_{new} \ .$$

This method is useful in learning a stationary model for a single Gaussian. Since the operating environment for

this research includes several surfaces and the actual motion model is a mixture of Gaussians, this method required modification. Since the model is expected to not be stationary, it is necessary to update G so that it does not include an infinite history. If G continues to grow unbounded, successive batches of particles will account for less and less of the update. To resolve this issue, a sliding window can be used to update the current value of G for only the last k iterations of SLAM. Experiments conducted for this research indicate that a present value of k with $5 \leq k \leq 10$ shows acceptable parameter changes and good tracking results.

To modify the above algorithm to update a mixture of two Gaussians, the portion of weight to assign to each Gaussian in the mixture is determined as

$$g_{j,t} = \sum_{i=1}^{N} \frac{p_j(x_i)}{p_1(x_i) + p_2(x_i)} q(x_i) ,$$

where $p_j(x_i)$ is the value of the probability density function of particle i for mixture component j for the set of particles. Therefore, $g_{j,t}$ is the sum total of weight assigned to the jth Gaussian in the mixture at time t. Using this measure allows each Gaussian in the mixture to be updated separately without penalizing Gaussians with few high-quality samples.

The actual update involves calculating the total weight to assign to the jth Gaussian in the mixture at time t via

$$G_{j,t+1} = G_{j,t} + g_{j,t} .$$

From these weights it is easy to calculate the percentage of total particles, λ, that came from each of the j components of the mixture, along with the mean, μ, and variance, σ^2, using:

$$\lambda_{j,new} = \frac{g_{j,t}}{g_{1,t} + g_{2,t}}$$

$$\mu_{j,new} = \frac{1}{g_{j,t}} \sum_{i=1}^{N} \frac{p_j(x_i)}{p_1(x_i) + p_2(x_i)} q(x_i) m_i \quad .$$

$$\sigma^2_{j,new} = \frac{1}{g_{j,t}} \sum_{i=1}^{N} \frac{p_j(x_i)}{p_1(x_i) + p_2(x_i)} q(x_i)(\mu_{j,new} - m_i)^2$$

The overall weight for each Gaussian in the mixture is calculated using

$$\lambda_{j,t+1} = \frac{G_{j,t}}{G_{j,t+1}} \lambda_{j,t} + \frac{g_{j,t}}{G_{j,t+1}} \lambda_{j,new} .$$

Updating the mean and variance of each Gaussian is accomplished using the same exponential estimator described above.

Important to note for this algorithm is that it can be implemented in one pass over the N particles used. Thus, the total processor overhead required for updating the motion model is $O(N)$. However, since each particle must be scored by the SLAM algorithm to determine how well it models the robot's current pose, this entire algorithm can actually be implemented during the regular execution of SLAM adding only a constant to the SLAM implementation's current overhead. The only long-term storage requirement is for the last k values of g_t and given the use of j Gaussian to model n parameters, this is only $O(j*k*n)$ memory locations.

Mentioned several times throughout the above discussion are the reasons a windowed update on a mixture of Gaussian was employed. Here is a more detailed summary of these reasons. First, a windowed approach allows the robot to track changes in the motion model that may arise due to surface changes or robot health issues. Second, calculating the total weight of particles each step allows sets of particles that contain a lot of information about a given parameter to update the model of that parameter much more than a set with little information. This means that when the robot enters an area of sparse data, the low weight particles will not unnecessarily modify the motion model. Next, updating each Gaussian separately allows the mixture to still model a variety of possible underlying distributions. This happens since each mixture component is modified only when it generates particles that actually provide feedback to that component. Finally, this method does not require a great deal of overhead in either processor time or memory usage, so the total overhead of SLAM is not significantly increased.

This methodology does not come free of cost. An initial estimate of the motion model is required as input, however, as will be shown later, this estimate can be quite rough. Another issue is that using a mixture of Gaussians adds a layer of complexity for parameters that are Gaussian. Despite this, these costs are relatively small considering the advantages.

C. RESULTS

The results from experiments designed to test the robustness of motion-model learning algorithm proved far

more successful than originally thought. The online learning proved to be able to learn a valid model given anything from a close approximation to almost an entirely incorrect model. It also did an excellent job at tracking how the model changes when a new surface is encountered and was even able to learn the model of a new robot given the model of an other robot of the same class.

Before testing the capabilities of the algorithm, the validity of the hand-measured model had to be established. To ensure that it correctly described the robot's motion, the non-learning DP-SLAM application was run with the hand-measured model. This run created an excellent map (Figure 19) and the robot did not get lost throughout the entire run. This suggests that the manually measured model sufficiently captures Aibo's true motion.

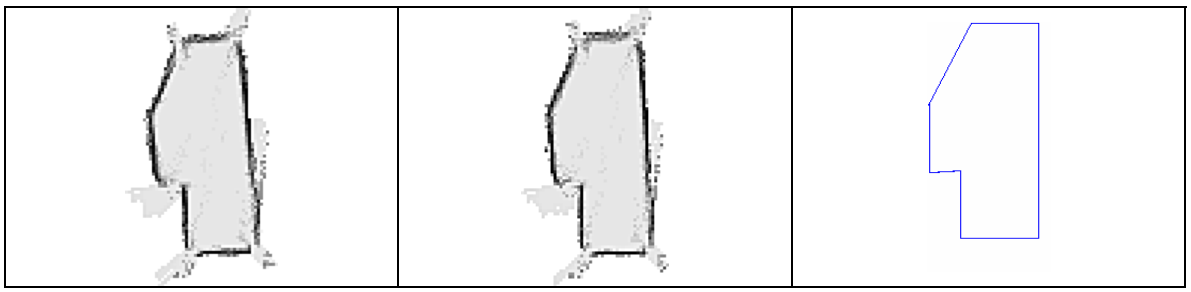

Figure 19. Maps generated by non-learning SLAM (left) and learning SLAM (center) applications when input was the measured motion model. Each pixel in the map represents approximately a 4x4 cm segment of floor space with the entire room measuring 133x305 cm. Note that maps are nearly identical and accurately represent the room's floor plan (right).

Next, the learning algorithm needed to be tested to ensure that it was valid and would not cause unnecessary changes to a valid model. To this end, the same set of sensor data was run through the learning SLAM application.

Examination of the output map and robot locations proved promising. Both were nearly identical to the non-learning version. Minor changes in the estimate of the robot's position were noted; however, these changes are expected since the learning algorithm is allowed to pare the variance in the motion model parameters down to their true values. This means that more particles are generated in valid locations and should, in theory, provide better estimates of the robot's actual position. These results suggest that the learning algorithm did not change the motion model unnecessarily when the model was indeed accurate to start with.

1. Learning the Model

The second method for testing the learning algorithm consisted of seeing whether the algorithm, if intentionally started with a poor motion model, would converge to something like the hand-crafted model. To test this hypothesis, data was collected from the robot while exploring the laboratory. This data was run through the learning SLAM application multiple times. Each time, the algorithm started with a slightly different initial model based on the measured model. For these runs, all of the models converged to nearly the same final model, and the maps (Figure 20) and locations generated by each run were very similar.

Figure 20. Maps generated using a good motion model (left) versus an initially skewed model with learning (right). The close similarity demonstrates how the robot has learned the correct motion model.

Figure 21 through Figure 24 show the major parameters for the walk (D) and turn (T) commands converging given the 19 different starting points. In all runs, the input models converge to nearly the same model, but note that this model is different than the hand-measured motion model. The major difference is the decrease in variance of the parameters. The convergence of the parameters is not unexpected. Since each particle is scored according to how well the observations fit into what is known of the world at that particular time step, good particles cause the models to converge to values that actually describe the robot's motion. As long as the observations provide some useful information about the environment and the current motion model at least partially captures how the robot is behaving, the learning algorithm will continue to adjust the model parameters towards the particles that best approximate how the robot actually moved.

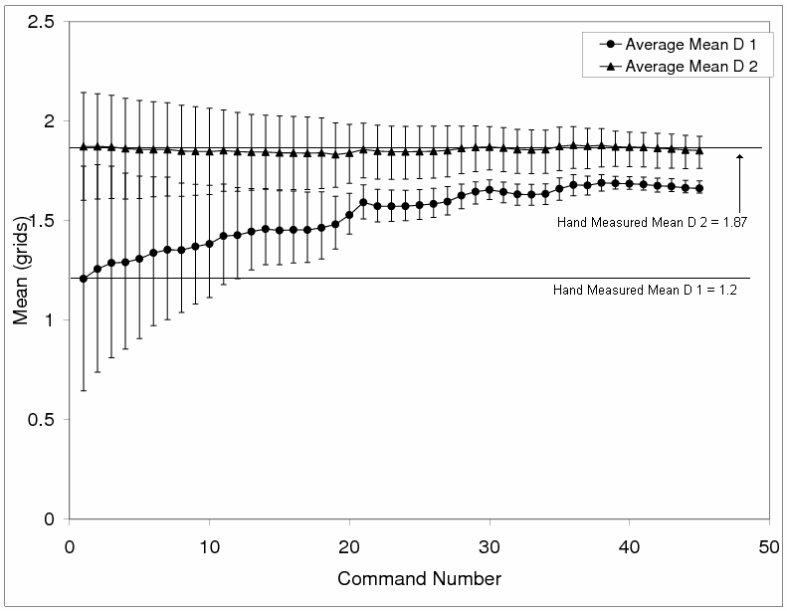

Figure 21. The means of the D parameter for the swalk(1) command converging given 19 different starting points. Solid lines show the average value and error bars indicate minimum and maximum values.

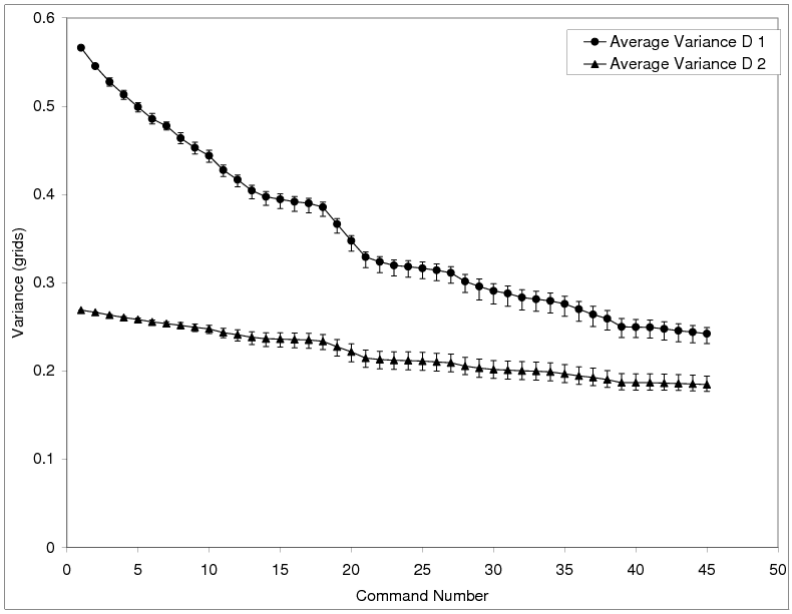

Figure 22. The variance of the D parameter for the swalk(1) command converging given 19 different starting points. Solid lines show the average value and error bars indicate minimum and maximum values. Note the starting point for the variance was not modified from the measured model.

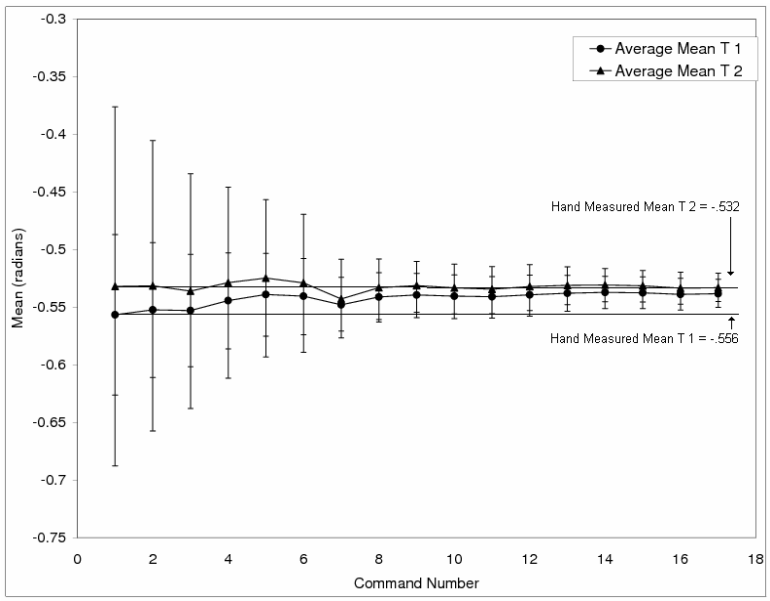

Figure 23. The means of the T parameter for the sturn(1) command converging given 19 different starting points. Solid lines show the average value and error bars indicate minimum and maximum values.

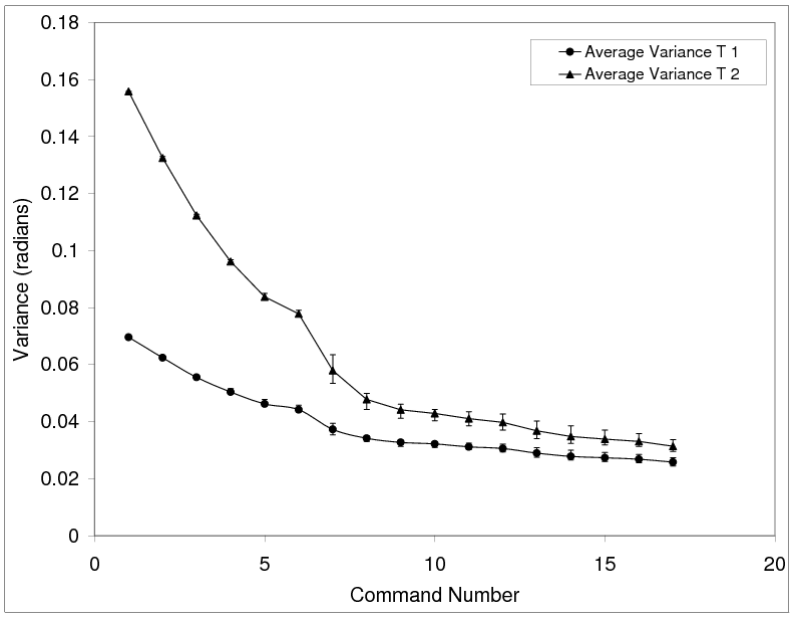

Figure 24. The variance of the T parameter for the sturn(1) command converging given 19 different starting points. Solid lines show the average value and error bars indicate minimum and maximum values. Note the starting point for the variance was not modified from the measured model.

As final test of the robustness of the algorithm to correct the motion model, DP-SLAM was initialized such that the sturn(1) command utilized the values of the sturn(2) command. This meant that when the robot actually turned approximately 30°, DP-SLAM would believe the robot had turned 60°. Even in this drastic situation, the learning algorithm corrected the model appropriately. It took longer for the model to be corrected, but nevertheless was adjusted appropriately.

2. Tracking the Model

To test the robustness of the learning in a changing environment, Aibo collected data starting on a surface with a known motion model and then transitioned onto a different surface for which the motion model was not valid. As could be expected, the non-learning version did not detect the change in surface and subsequently generated a poor map (Figure 25). The learning algorithm was able detect the change in surface and recalibrate its motion model, online, to generate a much more accurate map (Figure 26).

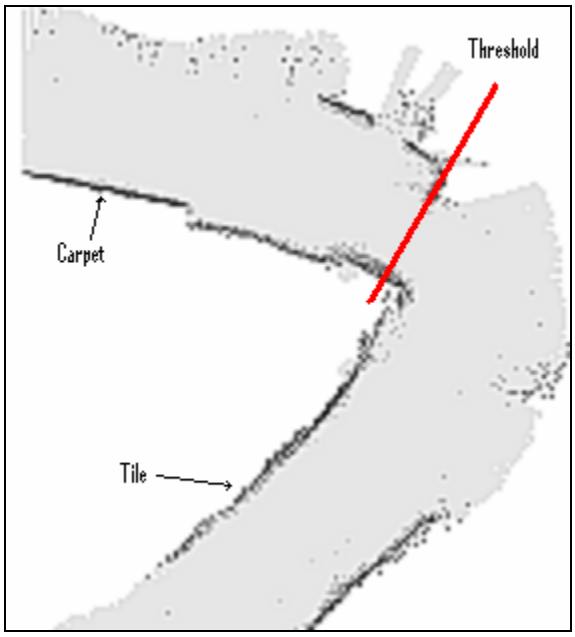

Figure 25. Map generated by Aibo during a run in which the robot changes surface. No learning occurred during the generation of this map leaving the map bent and some of the walls curved.

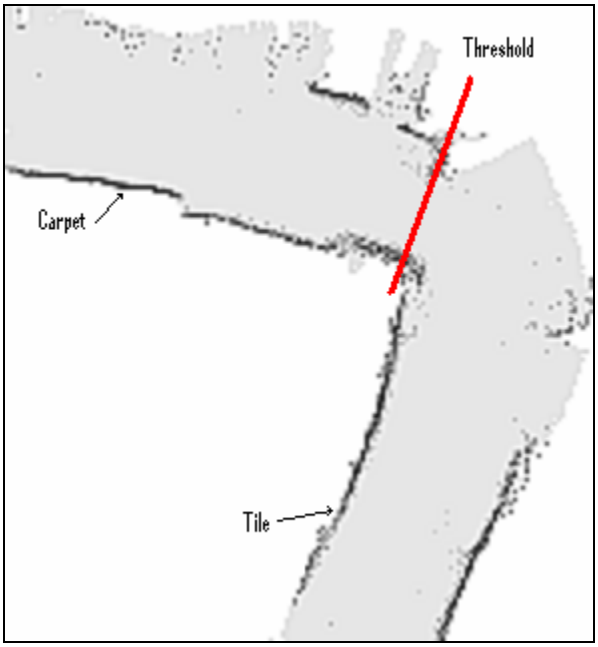

Figure 26. Map generated by Aibo during a run in which the robot changes surface. The robot was able to learn the model of the new surface and thus not get lost and generate a poor map.

The means of the D parameter (in Figure 27) clearly show the change in surface. At command 25, the robot begins to cross the threshold from one surface onto another. It takes four commands to cross the threshold, during which time the means of both Gaussians dip. After this crossing, the algorithm begins to learn that the robot has crossed onto another surface and the means of the distribution change at a rate previously only seen during the initial calibration at start-up. One issue noted here is that since over twenty commands had already been executed, the variance at this point is relatively small, approximately what is shown at a similar point in Figure 22. This small variance limits the rate at which the robot can learn the means of the Gaussians for the new surface. After the learning period though, the means settle back down and as one would expect, end at a place similar to the means shown in Figure 21. This is expected since both data runs were conducted by the same robot, ending on the same surface.

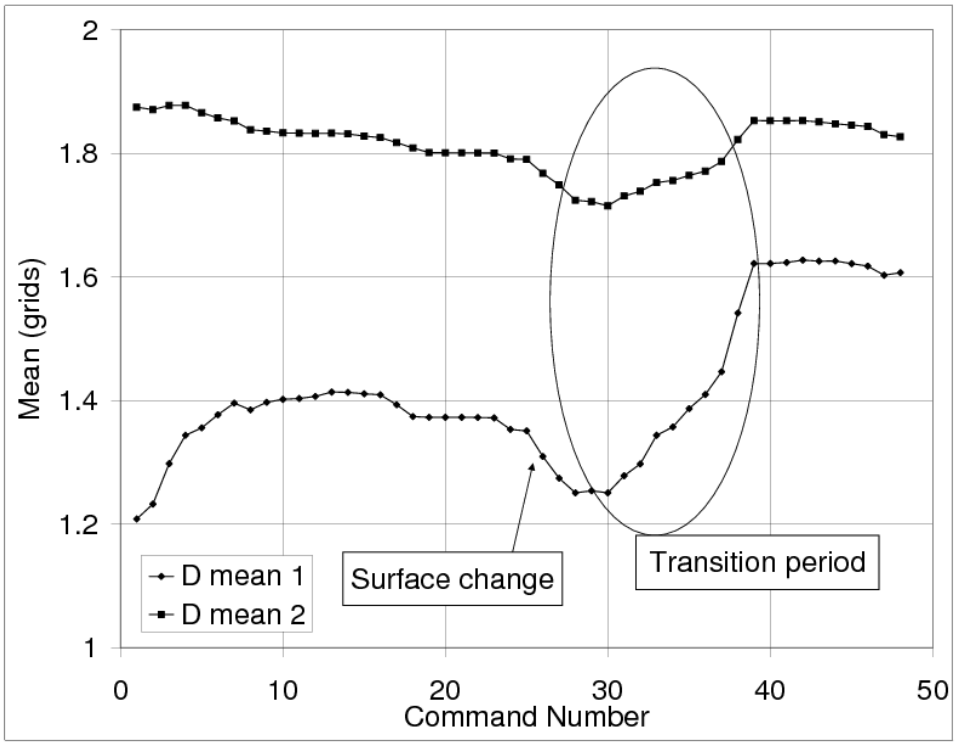

Figure 27. The change in threshold as shown by the means of the D parameter. Note the rapid change in means as the algorithm learns to adapt to the new surface.

As another test of learning in an uncertain environment, the robot started on a surface for which the input motion model was not valid. In Figure 28, the non-learning version (left) quickly gets lost and generates a very poor map. The learning version (Figure 28, right) is able to actively adjust the motion model and creates a much more accurate and consistent map. Note that the learning SLAM version does not detrimentally adjust the robot's motion model in areas of sparse readings. This is due to the low total quality of the particles at these steps. The ability of the learning to modify the model accurately in the presence of a new surface demonstrates how this methodology can be used to learn the model of a completely new surface without human intervention.

Figure 28. Maps generated by non-learning SLAM (left) and learning SLAM (right) given an incorrect motion model for the operating surface.

3. Learning a New Robot's Model

As a final test to the robustness of the learning, determining the motion model of a new robot of the same class was attempted. The learning SLAM program was started with the old robot's motion model in an effort to see if the algorithm would learn the motion model of the new robot.

During the run, the algorithm in fact learns the new motion model and does not get lost as the non-learning version does (Figure 29). This result is due largely to the fact that the old robot's motion model partially captures the new robot's behavior. This finding is significant in that it may allow for many robots of similar class to operate without the need to hand calibrate each robot's motion model independently. Note that the remaining skew in the learning map (Figure 29 right) is due to the inability of the algorithm to learn all the parameters for all of the commands issued due to a limited number of some commands

being executed. This issue can be resolved simply be allowing the robot to operate longer and perform more commands of the desired type.

Figure 29. Maps generated by non-learning SLAM (left) and learning SLAM (right) using the motion model for one robot as the input for a completely different robot.

V. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

The initial results from the autonomous coordination work proved extremely promising. Simple rules and local information were all that was necessary to generate simple clusters that could seamlessly handle obstacles, and the sudden loss or addition of agents. This basic algorithm seems capable of providing the base for more complex formations since it effectively maintains good spacing between all of the individual agents. For the more complex formations, methods are needed to restrict which agent is allowed to pick which other agent as its nearest neighbor.

The most stunning results came from the experiments to learn a motion model. Generating a motion model by hand is time-intensive and does not allow seamless transition to other robots or environments. The online learning algorithm presented here learns the parameters of a robot's motion model given only one, fairly weak, precondition: the initial model estimate generates at least some particles that approximate the robot's actual pose. The fact that the algorithm converges when started with an intentionally skewed model and when a different robot is used is essentially equivalent. The algorithm's convergence in one case necessitates that it will converge in the other since the model for one robot will most likely appear to be a skewed model to a different robot of the same class.

The maps generated by our learning algorithm are across the board better than the maps generated by the typical, non-learning SLAM. This improvement is due in

whole to better estimates of the robot's pose introduced by the learning algorithm to the SLAM application. These estimates keep the robot from becoming lost and allow successive observations to be placed correctly. The smaller variances in the resulting model allow for the use of fewer particles while the robot's pose is still effectively tracked. Preliminary findings show that up to 10 times fewer particles can be used and still create decent maps and robot poses. As a whole, the learning algorithm can greatly reduce the time to field new robots and allow robots to operate in a variety of environments even if they have not been tested in those environments.

B. RECOMMENDATIONS FOR FUTURE WORK

The coordination piece of this thesis requires several extensions before it can be successfully fielded in a combat setting. Some of the facets that need to be added are path planning to allow agents to better avoid obstacles and goal finding behaviors to move the formation in a desired way. The implementation of complex formations still needs to be fully investigated.

Additionally, several of the parameters used during the motion model learning trials were empirically set. The number of particles used and the value of k (the size of the sliding wind) are examples of such variables. Further work is needed to determine an effective lower limit for the number of particles. In addition, it is desirable to determine how to go about setting the value of k dynamically such that the learning of the motion model is optimized for any given situation. It seems intuitive that in large open areas, it may be better to increase the value

of k so that the robot will not change its motion model until it enters an area of non-sparse readings; however, this requires explicit testing.

Another learning issue that must be tackled is how to shorten the learning period when a surface is crossed. As mentioned earlier, a hypothesis is that small variances and limited particle numbers lengthen the learning period. Therefore, if the agent can actively monitor how the model is changing and use larger variances and increased particle numbers during learning periods, this should minimize the length of the learning period.

An AIBO specific item that requires some scrutiny is determining the dependency of one command type on future command types. For the implementation discussed here, the commands were assumed independent, that is, one command did not affect the motion of the next. To eliminate the issue of learning the interaction of commands in the place of learning the commands themselves, the learning algorithm was prevented from running immediately after command type were switched. This assumption prevented the model from determining the actual dependency of certain command types on other command times. Determining this interaction should further improve the motion model.

Finally, the mathematical convergence of the algorithm needs verification. Initial experiments indicate that even when the input motion model varies wildly from the measured model, the parameters converge toward the same point as the measured model. The speed of this convergence appears to be highly dependent on the number of particles used in the SLAM algorithm and the input variance to the distribution. Although these results are promising, accurately defining

the point and speed of convergence is quite important for determining how to most effectively learn and track a robot's motion model.

LIST OF REFERENCES

[1] J.-C. Baille, "URBI: towards a universal low-level programming language," IROS, 2005.

[2] R. Bergevin, "An Analysis of the Generalized Lambda Distribution," AFIT Master's Thesis, 1993.

[3] J. Borenstein and L. Feng, "Measurement and Correction of Systematic Odometry Errors in Mobile Robots," in IEEE Transactions on Robotics and Automation, Dec 1996.

[4] C. Breder, "Equations Descriptive of Fish Schools and Other Animal Aggregations," in Ecology, Vol 35, No. 3, Jul 1954.

[5] D. Caltabiano, G. Muscato, and F. Russo, "Localization and Self Calibration of a Robot for Volcano Exploration," in ICRA, 2004.

[6] DARPA, "The DARPA Grand Challenge," http://www.darpa.mil/grandchallenge/index.asp, Apr. 2007.

[7] A. Elizar and R. Parr, "Hierarchical Linear/Constant Time SLAM Using Particle Filters for Dense Maps," NIPS-05, 2005.

[8] A. Elizar and R. Parr, "DP-SLAM 2.0," Proceedings of the IEEE ICRA, 2004.

[9] A. Elizar and R. Parr, "Learning Probabilistic Motion Models for Mobile Robots," in International Conference on Machine Learning, 2004.

[10] A. Elizar and R. Parr, "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping without Predetermined Landmarks," in IJCAI-03, 2003.

[11] E. Foxlin, "Generalized Architecture for Simultaneous Localization, Auto-Calibration, and Map-building," in IEEE/RSJ Conf. on Intelligent Robots and Systems, Oct 2002.

[12] S. Hynes, "Multi-Agent Simulations for Assessing Massive Sensor Coverage and Deployment," Master's Thesis, Naval Postgraduate School, Monterey, CA, 2003.

[13] A. Law and W. Kelton, Simulation Modeling and Analysis. McGraw-Hill, 2001.

[14] Sing Li, "Rock 'em, sock 'em Robocode!" http://www-128.ibm.com/developerworks/java/library/j-robocode/, last date accessed, June/2007.

[15] J. Lin and A. Jadbabaie, "Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbors Rules," in IEEE Transactions on Automatic Control, June 2003.

[16] Z. Lin, M. Broucke, and B. Francis, "Local Control Strategies for Groups of Mobile Autonomous Agents," in IEEE Transactions on Automatic Control, Vol 49, No 4, Apr 2004.

[17] P. Ludwig, "Formation Control for Multi-Vehicle Robotic Minesweeping," Master's Thesis, Naval Postgraduate School, Monterey, CA, 2003.

[18] A. Martinelli, N. Tomatis, A. Tapus, and R. Siegwart, "Simultaneous Localization and Odometry Calibration for Mobile Robot," in Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, Oct 2003.

[19] T. Mitchell, Machine Learning. McGraw-Hill, 1997.

[20] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges," in IJCAI-03, 2003.

[21] Office of Management and Budget, "Budget of the United States Government, FY 2007," http://www.whitehouse.gov/omb/budget/fy2007/defense.html, last date accessed, April 2007.

[22] C. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," in Computer Graphics, July 1987.

[23] N. Roy and S. Thrun, "Online Self-Calibration For Mobile Robots," in Proceedings of the IEEE ICRA, 1999.

[24] K. Sugawara and T. Watanabe, "A Study on Biologically Inspired Flocking Robots," in Proceedings of the 2003 IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, Oct 2003.

[25] K. Sugawara, T. Watanabe, R. Arai, M. Sano, Y. Hayakawa, and T. Mizuguchi, "Collective Motion of Interacting Simple Robots," in The 27th Annual Conference of the IEEE Industrial Electronics Society, 2001.

[26] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics. MIT Press, 2005.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Director, Training and Education, MCCDC,
   Code C46
   Quantico, Virginia

5. Director, Marine Corps Research Center, MCCDC,
   Code C40RC
   Quantico, Virginia

6. Marine Corps Tactical Systems Support Activity
   (Attn:Operations Officer)
   Camp Pendleton, California