AN ABSTRACT OF THE THESIS OF

Matthew D. Knudson for the degree of Master of Science in
Mechanical Engineering presented on February 11, 2008.

Title:

Applying Hierarchical and Adaptive Control to Coordinating Simple Robots

Abstract approved: _____

Kagan Tumer

Coordinating multiple robots to achieve a complex task requires solving two distinct control problems: the high-level control problem of ensuring that each robot aims to perform a useful task (e.g., coordination) and the low-level control problem of ensuring that each robot actually performs the correct actions to achieve its task (e.g., navigation and locomotion). Though addressing both problems simultaneously with one algorithm is appealing, this is often difficult to impossible in domains requiring a combination of complex actions (goal selection, navigation, obstacle avoidance). This thesis establishes a hierarchical control structure, presents an adaptive navigation method, compares it to reactive navigation, and applies established adaptive coordination techniques under severe restrictions.

The development and experimentation process produced results showing the following:

1. Hierarchical control structure proves effective and useful for use on resource limited robotic platforms allowing the subsequent navigation and coordination analyses to be addressed individually.

2. Adaptive navigation is an effective approach for dense environments with limited and noisy sensing, providing improvement over reactive navigation by up to 75%.

3. The application of an abstract difference objective function to training for coordination remains effective under limited information and physical robot motion restrictions, outperforming traditional system or local objectives by up to 50%.

Specifically, this work establishes that neuro-evolutionary methods are applicable and beneficial both for the discovery of successful navigation techniques, as well as for the generation of coordination behavior in realistic multi-robot teams where individuals are strongly limited in sensing, communication, and computational ability. Possible extensions include increased levels of communication among individuals as well as configuring individual sensing abilities for heterogeneous teams.

Applying Hierarchical and Adaptive Control to Coordinating Simple Robots

by

Matthew D. Knudson

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented February 11, 2008
Commencement June 2008

Master of Science thesis of Matthew D. Knudson presented on February 11, 2008.

APPROVED:

Major Professor, representing Mechanical Engineering

Head of the School of Mechanical, Industrial, and Manufacturing Engineering

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Matthew D. Knudson, Author

ACKNOWLEDGEMENTS

I would like to extend my thanks Dr. Kagan Tumer for all the time he has given toward my education and for his seemingly ceaseless availability. Also to Dr. Belinda Batten for her skill in keeping my eye on the future and for being my biggest fan in the department. I greatly appreciate all of the contributions of my graduate committee, especially for their enthusiasm toward my work and willingness to provide whatever support possible in furtherance of my career.
I would also like to personally thank Mr. Matthew MacClary for all the inspiration he gave to me during our time working together. His approach to simple and effective engineering is the basis for this work and for all the work I do.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Chapter 1 – Introduction

In many applications, exploration of an unknown environment is a prerequisite for the completion of complex robotic exploration tasks. For example, planetary and lunar surface exploration, as well as search and rescue operations require that a generally unknown environment is explored with the intent of gathering information or locating a specific target. This environment may be dangerous, uninhabitable to humans all together, or at such great distances from a central location, that directed exploration is difficult or impossible. Utilizing robots for exploration in these environments not only provides the benefit of safety, but cost-effectiveness in most situations as well.

In general these environments are cluttered with obstacles or contain difficult terrain making successful navigation a complex problem. Robots intended for use at great distances from Earth, or in dangerous search operations, must have the capability to articulate in challenging terrain, avoid time consuming or potentially damaging collisions with obstacles, as well as maintain successful progress toward their original intended purpose.

Specifically, in addition to locomotion, robotic exploration has two main requirements: decision making for low level tasks such as navigation, and for high level tasks such as destination selection. Challenges in addressing these requirements range from command flow to maximization of information gathering [39, 48].

Organizing robots into teams provides benefits to information gathering, as well as to robustness to failures, but in general adds complications to command flow. Therefore, addressing how the robot moves, how it selects potential paths in response to emergent environmental information, and its coordination with other team members to maximize the amount of information gathered presents a complex control problem.

From a design perspective, incrementally separating the above requirements into categorical groups during the design process is beneficial to reduce the tasks and the command complexity involved, as well as to provide the system designer with discrete levels to be addressed. Specific to large mobile robot teams, a hierarchical control structure on each robot allows for heterogeneous teams to be designed without the need for significantly different individual control procedures.

Additionally, it is beneficial to maintain simple robotic platforms with limited resources to further reduce the team complexity and decrease the overall system impact of individual robot failures. A simple, effective robotic platform is chosen and sensors are provided for each level of a hierarchical control structure pertinent to the coordination and information gathering domain.

Mobile robot navigation in unknown environments has been widely addressed with a variety of techniques. Specific to the domain of large teams of mobile robots, it is desirable for the robots to examine the environment and make subsequent decisions as simply as possible to reduce computational needs while isolating the requirements of navigation from goal selection mechanisms. A neuro-evolutionary adaptive algorithm is presented and compared against a predefined, probabilistic

reactive algorithm under a similar state/action space structure. This will address navigation control requirements, and is shown to be successful and outperform reactive navigation in cluttered environments.

The goal selection mechanism when operating in multiple robot teams is vital in producing effective coordination. For example, an individual robot must not re-examine areas of the environment already discovered and examined by other members of the team, unless it is directly beneficial to do so. Utilizing learning techniques in this domain have proven successful in other work (Chapter 2) for establishing good coordination behavior, and effective reward structures for training the team have also been established. These techniques and reward structures are applied under restrictions of physical robot motion in addition to control group isolation. Specifically, applying a difference objective function to coordination and information gathering with a neuro-evolutionary learning technique is shown to be successful when operating under these restrictions.

Chapter 2 addresses previous work done in the areas of robot navigation and multi-robot teams as it pertains to support of the work done in this thesis. It covers both model-based and model-free approaches to navigation as well as static and adaptive approaches to building and controlling multi-robot teams for application in domains where coordination among members is necessary.

Chapter 3 first presents a hierarchical control structure for use in the design of the individuals and teams within the exploration domain. It then presents a resource limited robotic platform and sensor structure is chosen and discussed as the model on which the simulations are based. Finally, the simulation technology

itself is presented as it pertains to the experimental structure for this work.

In Chapter 4, two general navigation techniques are presented: reactive and adaptive. Specific algorithms are chosen for each and their function is described in detail. In addition, a navigation problem structure is presented to provide for progressive analysis of algorithm behavior as it relates to the requirements of complex environment robotic navigation. Then, the experimental results and specific conclusions are provided throughout, along with a summary and discussion suggesting the next steps.

Chapter 5 presents the details of the robot team coordination problem chosen for this work, including objectives and the properties of the functions used for coordination training. Experiments are addressed to evaluate the functionality of the objective structure under the constraints of physical robot motion as well as hierarchical command flow. As above, the experimental results and conclusions are provided, followed by a summary and discussion.

Finally, Chapter 6 provides a summary of the intentions of this work, the contributions of the results obtained and conclusions drawn, and discusses the potential avenues for further research in this area.

Chapter 2 – Background

Two significant applications of techniques developed for robotic exploration of unknown domains are the areas of planetary or lunar surfaces and robotic search and rescue missions. Using robots for these tasks offers both safe and cost-effective solutions to challenging problems. For robots to accomplish such tasks with success however, two major challenges must be addressed: low level control (i.e., localization and navigation) and high level control (i.e., planning and decision making). Both are critical to the control of individual robots as well as teams of robots operating in complex environments with the objective of maximizing information gain safely.

2.1 Low Level Control

Establishing successful low level control policies (e.g., navigation) for continuous and complex environments is an important challenge. This can be done using physical models of the robot and the environment in which it operates to determine the limits of functionality and interaction [7]. In general, planning techniques have also proven successful for navigation in unknown environments [23], including a heuristic algorithm Dynamic A* [32], D* heuristic planning [31], and a modified D* replanning algorithm [22], which all utilize heuristic mechanisms for path plan-

ning. The utilization of Markov models and adaptive techniques for use in sensor interpretation have been shown to improve navigation performance as well [16, 52].

However, generating good models of an environment, either offline or online, can prove quite difficult. Therefore, low level control policies must effectively retain the ability to react to inaccurately generated or nonexistent environment models [47, 46]. Specifically, adaptive techniques for tasks including rocket and helicopter control, walking robots, pole balancing, and robot navigation have produced good results [6, 14, 17, 33, 41]. Model-free learning algorithms such as reinforcement learning can also be used for path planning applications [10, 45]. Though good solutions exist for decision making for single robots, the control problem becomes more complicated in multi-robot settings [15, 26].

2.2 High Level Control

Extending adaptive techniques to the high level control requirements in a multi-robot domain suggests significant benefits to exploration applications when the complexity and scope of many current domains exceed the capabilities of a single robot [25, 49]. The root design goal in such a complex domain is to create a cooperative team of multiple robots where the behavior of each single robot furthers a system-wide objective. When the cooperative team is small or the space of actions is limited, both navigation and goal selection (i.e. low and high level control) can be achieved with a single learning algorithm [49]. By contrast, problems requiring the robots to each perform complex, multiple time step actions as a consequence

of the tasking at hand present large amounts of sensor information in furtherance of the system objective and therefore make it an intractable control problem for a single adaptive algorithm. For example, robotic exploration domains would require the algorithm to not only determine where to go to help the "team" but also select safe and efficient paths to avoid obstacles and reach that destination quickly. Reinforcement learning has been applied for action selection [5, 45], as has neuro-control [49, 52] and behavior based policies [16, 27]. However, breaking the requirements into categorical tasks reduces the computational burden for both types of application.

The majority of non-adaptive approaches to coordination in multi-robot teams are based on planning, swarms, auctions, and domain specific algorithms based on detailed models. Role allocation and plan instantiation have proven successful for large teams of robots [38] and nondeterministic planning has been applied to adjust for uncontrollable robots with goals differing from the majority of the team [8]. The application of swarm algorithms have been successful for the coordination problem [12, 28], and utilizing auction techniques in the domain of robot coordination for exploration [29] and generating bidding rules for multi-robot routing [24] has shown to provide coordination as well. Several successful applications of robot coordination include search and rescue [51, 54], robotic soccer [28, 43], mobile sensor networks [19], and mine collection [16].

Applying adaptive algorithms, over such non-adaptive approaches, seeks to ensure that the coordination achieved within the team of robots is robust in continuous, dynamic, and stochastic environments. Specifically, methods that augment

traditional methods with learning features are ideal for addressing complex coordination problems. For example, using Markov Decision Processes for online mechanism design [35], or developing new reinforcement learning based algorithms [5, 44] have produced successful coordinated systems.

Finally, evolutionary algorithms have been explored where individual robots are selected based on their behavior contributions in promotion of particular system behavior [3, 49]. Additionally this work provides quantitative measures (i.e., *factoredness* and *learnability*) by which the objective functions used for the search can be evaluated. The majority of the coordination work in this thesis is based on these coordination problems and behavior analysis.

Chapter 3 – Simulation

The process of exploring an unknown environment includes identifying important features of the environment to facilitate navigation, in addition to determining what features indicate potential targets for investigation. These requirements can be applied directly to robotic exploration. Assuming a robot has the ability to efficiently move through its environment, a robot must have the ability to localize and determine a safe and effective path. Second, to gather information, a robot must have a method of identifying and prioritizing targets for exploration.

If these abilities are present, they must be exploited in a manner to minimize the cost of the exploration, while maximizing the amount of information gathered. Vital for an accurate robotic exploration solution is the effective control of robot locomotion to quickly and safely navigate, while seeking and exploring targets of interest.

One effective approach to providing these abilities is to segment the requirements into a hierarchical control structure. In this way, each requirement can be designed and evaluated independently of the others. In this work, the entire robot control environment is segmented in a manner directly related to the above requirement structure.

As shown in Figure 3.1, the three basic robot requirements are categorically segmented. At the base level the control tasks for robot locomotion are placed,

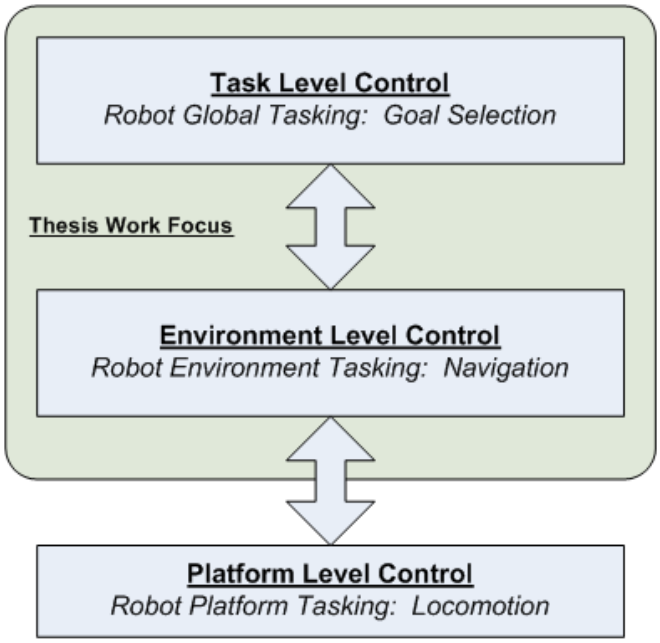

Figure 3.1: Control Grouping: Abstract representation of robot control group task segmentation.

including motor control for translational and rotational movement. The mid-level contains the control tasks for navigation through an environment, including path selection and obstacle avoidance. Finally, at the top level, the control tasks for global robot purpose are placed, including target acquisition and goal selection. This thesis presents the details of each from the platform level to the task level.

3.1 Robot Architecture

Beginning with the platform control level, a robotic platform was selected to base the simulation environment upon. The target of such a choice included (but was not

limited to) articulation, simplicity, power consumption, and resource availability. The robot platform chosen to fit these needs was the TekBot™robot platform developed at Oregon State University.

The TekBot™is a simple, scalable robotic platform. It utilizes two wheels for translational and lateral actuation with a ball drag bearing to provide the third point of ground contact. This robot [40]:

Has Small Footprint: The robot is approximately 20x20cm.

Is Lightweight: The entire package weighs less than 2kg.

Uses Low Power: The robot contains six NiCd 700mAh AA batteries and under normal operating conditions consumes less than 10 Watts.

Has Long Range: At maximum speed, 25 cm/s, and full processing load, the robot can operate for up to 2 hours providing a range of approximately 1.8 km.

Is Inexpensive: The platform with microcontroller costs less than three hundred dollars.

The TekBot™(shown in Figure 3.2) is an ideal robot for the multi-robot coordination domain because it is small, readily available, and relatively inexpensive to build. Furthermore, it lends itself well to the development of limited resource algorithms due to its simple but powerful Xilinx Spartan3 FPGA microprocessor. For investigating autonomous multi-robot coordination and navigation, multiple

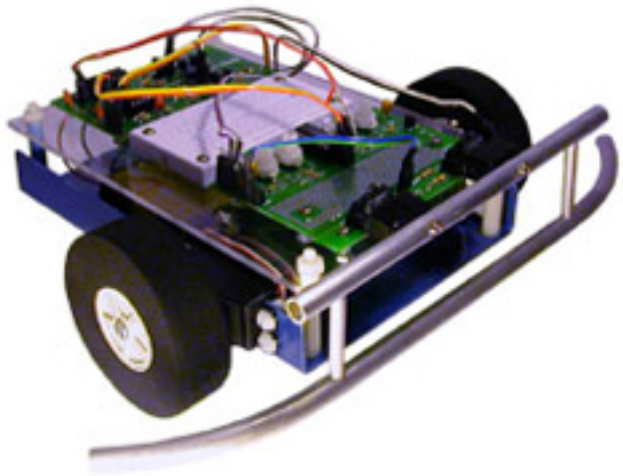

Figure 3.2: Robot Platform: The robotic platform chosen for the basis of physical motion within the simulator.

robots were constructed and therefore the platform allows for safe and efficient algorithm development and evaluation.

The robot chosen does not have an active steering system. Instead, the robot utilizes two DC motors, one for each wheel, and the difference in motor outputs produces steering capability. This technique allows for maximum robot articulation with minimal power. The robot has the ability to travel directly forward or backward, alter heading while traveling, or rotate in place. To move directly forward or backward, the motors are driven at the same speed, in the same direction. To alter heading while traveling, one motor is driven to a lower speed than the other and in the same direction, altering heading towards the slower motor. To rotate in place, the motors are driven at the same speed in opposing directions.

Speed control for the robot is done utilizing feedback provided by inertial sens-

ing, discussed in Section 3.2, as well as motor voltage detection. The speed of the robot is scaled from maximum based on the distance to the current waypoint or goal and the path quality provided by the navigation algorithm. The output of this speed control is then sent to the motor controller.

Motor output is also determined utilizing feedback provided by an inertial measurement sensor. Basic locomotion control for determining the speed of the wheels relative to each other is done with a standard model-based proportional-derivative controller [11]. This controller determines the signals to send to the motor-driven wheels based on the desired speed and heading correction provided by the path selection algorithm, presented in Chapter 4.

3.2 Sensing

Sensing requirements were separated into three sensor categories based on their application to the respective control levels within the hierarchy. The robot is provided with sensors for navigation, obstacle detection, and goal identification. Detail of the sensors chosen for these applications follow.

3.2.1 Localization and Navigation

The navigation sensor for this robot is an inertial navigation sensor. This type of sensor is low power and easy to use. The rudimentary technique for localization is dead-reckoning, where the robot position is determined based on known robot

dynamics. It provides accurate localization within a given time window and general distance [48]. An advanced technique is GPS, but these devices have much larger power requirements and cannot be used indoors. The sensor chosen contains two accelerometers and two gyroscopes for x, y movement detection as well as yaw and pitch deflections. The sensor also contains a magnetometer for determining Earth relative magnetic heading. It can be actively and simply normalized to provide noise reduction.

3.2.2 Obstacle and Robot Identification

Obstacles are identified using two simple stereo ultrasonic sensors positioned on a rotating servo to provide information in eight regions surrounding the robot. These sensors provide basic information about the robot environment with very little power. Other alternatives, including infrared laser diodes, consume a great deal more power. With this sensor, the information is provided in a distance in centimeters under 20 degrees Celsius and sea level conditions. A rolling average is kept for all readings performed in each of the eight regions to filter erroneous data.

Other robots within the environment also return sonar information which is then paired with simple RF signal strength techniques to identify other robots within the environment. This technique identifies the number of other robots within range and returns rudimentary (and approximate) distances to each. This makes assumptions based on distance ranges. For example, if the RF signal received from another robot indicates it is a much greater distance than that of the

sonar signal in that direction, the sonar signal is assumed to be a static obstacle, not the other robot.

3.2.3 Goal Identification

The robot is equipped with two thermal sensors mounted on the same rotating servo to provide thermal information in four regions. These sensors have very low power requirements and allow target acquisition to proceed independently than obstacle detection. These thermal sensors contain eight pixels each, therefore segmenting each region into eight sub-sections. The information is scaled to integers inclusively between 0 and 9 representing the degree of thermal information above the actively measured ambient temperature.

3.2.4 Combined Sensor Operation

Figure 3.3 shows a logical function and command flow for robot sensing and navigation. Information from Point Of Interest (POI) thermal sensors, navigation sensors, and obstacle sensors is split categorically and processed individually by unique interpreters, then combined progressively to produce actuation commands. This is done first by identifying targets (POIs) and selecting one (goal selection), determining a safe and efficient path to the selected goal (navigation), then executing that path (locomotion).

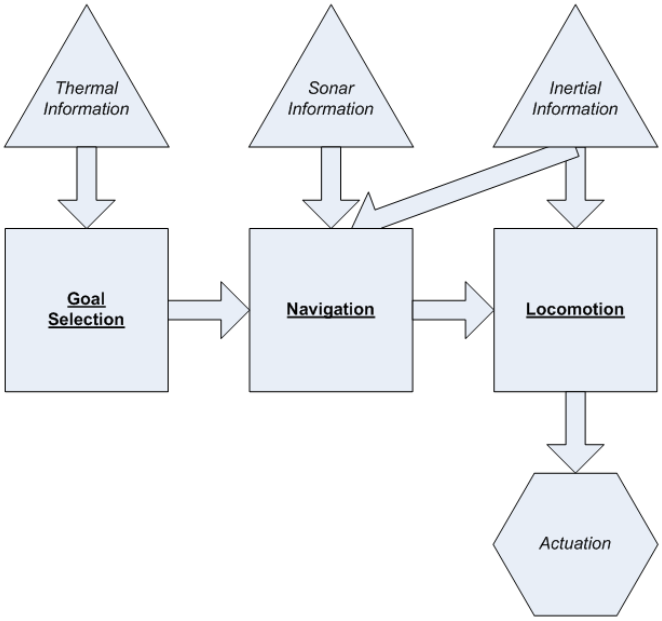

Figure 3.3: Logical Command Flow: Logical function and command flow for robot tasking. Thermal information is provided to the goal selector, which chooses a destination to give to the navigator. The navigator uses destination, sonar, and inertial information to choose a safe path and provides speed and heading to the platform control which then determines actuator settings based on inertial information.

3.3 Simulator Domain

In general, it is difficult to directly test and evaluate higher-order functionality on a limited resource microcontroller. The lack of debugging information and the compilation and iteration time consumption make it difficult to collect and store the relevant information. To overcome this difficulty, a modular simulation environment consisting of an environment simulator, a sensor simulator and a robot simulator was created. A summary of the simulation environment follows:

1. The environment simulator provides random or targeted environment generation (e.g., placement and geometry of goals and obstacles). Using basic two dimensional geometry equations it computes both the robot position and orientation, and provides realistic data to the sensor simulator.

2. The sensor simulator "scans" the environment by requesting information from the environment simulator in a specific format for the three sensor types - navigation, sonar, and thermal:

   (i) Navigation sensor simulation is provided by using numerical integration on the robot equations of motion, with the wheel rotation speeds as input, which are provided by the robot simulator (see below).
   (ii) Sonar sensor simulation is provided by creating two dimensional line segments emanating from the robot at specific vehicle relative angles and labeling those segments as requesting "echo" data in return. This is done continuously using triangle-circle intersection geometry to prevent discretization errors.
   (iii) Thermal sensor simulation is provided in similar fashion, but by requesting "heat intensity" data rather than "echo" data. This data is determined based on the distance of the sensor from the object being sensed.

3. The robot simulator models the robot actuators and provides the control systems with the ability to transparently modify individual wheel direction and speed. The simulator also tracks "power usage" by the simulated motors and processor over time and modifies the actuator behavior as a result of

diminishing battery power.

In modularizing the simulation package, code for the operating system on the robot presented in Section 3.1 is generated during the development process precisely as if it were already in operation on the physical robot. The other simulators become transparent to the control systems being designed and evaluated, and therefore a transition to the physical robot environment is as seamless as possible.

Chapter 4 – Navigation

Under the hierarchical control structure presented in the previous chapter, the specifics of robot navigation through unknown, complex environments need to be addressed. The robot must have the ability to choose safe and efficient paths through an environment to reach a specific destination. This includes the ability to avoid obstacles and maximize robot speed, while maintaining a level of robustness to inaccuracies (noise) in sensor and actuator signals.

In addressing these requirements, it is important to determine the benefits and drawbacks of a reactive algorithm (i.e., does not modify behavior) as it compares to an algorithm with the ability to learn and change behavior as it operates within an environment. This adaptive ability is important when the exact solution to effective navigation is unknown to the system designer as well as for situations where the algorithm can not be easily shut down and redesigned.

For this work, navigation through the environment was achieved using three algorithms for controlling the robot based on environment information obtained from the sonar and inertial sensors. A random technique is used to produce a baseline, where the action taken at each time step is random. The three algorithms investigated for the control requirements in the mid level of the hierarchical control grouping are:

1. **Reactive Navigation:** A probabilistic navigation algorithm begins by ex-

ecuting a predefined waypoint list, or randomly generating a list. The algorithm assigns a path quality to each possible robot heading, based on a predefined probabilistic interpretation of environment detection information, then creates a path at the heading of greatest path quality.

2. **Adaptive Navigation I:** This is a Multi Layer Perceptron (MLP) [1, 9, 36, 13] based controller designed under the same premise as the probabilistic driver in that it interprets environment detection information and assigns a path quality to each potential path. However, it deviates in the interpretation method where the sensor information for that potential path is fed directly to an MLP and the output of the MLP is used as the path quality.

3. **Adaptive Navigation II:** This is a Reinforcement Learning path selector that chooses the right actions based on a Markov Decision Process (MDP) [21, 45, 36]. The robot associates a "value" to each action based on its current state and updates that value based on the final outcome of a trial run.

These unique algorithms are evaluated with the goal of determining which provides the best individual control of the robot, the most flexibility in terms of fitting within a hierarchical control flow, and which provides a computationally realistic procedure for navigation technique on a resource limited platform.

4.1 Reactive Navigation

Reactive navigation is achieved through a probabilistic path selection algorithm. It is termed "reactive" because the algorithm does not contain any mechanism for modifying behavior based on knowledge gained during operation. Given a "desired location" (e.g., through the goal selection algorithm or waypoint list) the path selector needs to compute the speed and heading corrections to reach that location.

In this approach, each path is assigned a probability of leading to the desired location based on the distance to that location and current heading of the robot. Then, each path is assigned a probability of safety, which is based on the presence of obstacles along a given path. The product of these two probabilities provides a prediction of success for a given path and is labeled the *path quality*. The path with greatest quality (greatest "probability of success") is then chosen as the, potentially new, desired robot heading.

In the absence of prior information, this is a likelihood based approach, where each path is evaluated solely based on collected data. However, including prior information and updating the path quality posterior probabilities would provide a true Bayesian navigation method. Figure 4.1 shows the algorithm itself and Figure 4.2 shows sample probabilities of certain paths as a function of distance to nearest obstacle and angle to desired location. The heading with the highest path quality is selected, and the speed is linearly scaled based on that quality (e.g., 90% or greater path quality equates to maximum speed, whereas a 50% path quality

Gather desired location, current state

For $\theta_i \leq 360$ Loop:

1. Calculate P_{direct} given α_v and α_{des}
2. Calculate P_{safe} given d_{θ_i}
3. $Q(\theta_i) \Leftarrow P_{direct} \times P_{safe}$

$\alpha_u \Leftarrow argmax Q(\theta_i)$

$V_u \Leftarrow F(Q(\alpha_u))$

Figure 4.1: Probabilistic Navigation Algorithm: Algorithm to determine the quality of potential robot paths. P_{direct} and P_{safe} are the probabilities of the path being direct to the desired location and safe respectively, θ_i is the potential path, α_v, α_{des}, and α_u are the current, desired, and commanded robot headings respectively, d_{θ_i} is the obstacle distance reported at the potential path, $Q(\theta_i)$ is the quality assignment (probability of success) to the potential path, and finally V_u is the commanded robot speed (given by a linear function $F(Q)$).

results in half speed). This scaling is one component of the speed control discussed in Section 3.1.

The path quality determination for each potential path (Figure 4.2) is engineered offline based on known robotic articulation capabilities (e.g., speed and turning ability) and on experimental observations made during the initial design. As shown, the potential path quality falls away from 1.0 as either the potential path is further away from the desired location angle or the sonar distance reduces to "dangerous" levels. By assigning a quality to each potential path and choosing the best (Figure 4.1), a smooth transition is made between discrete path choices as the

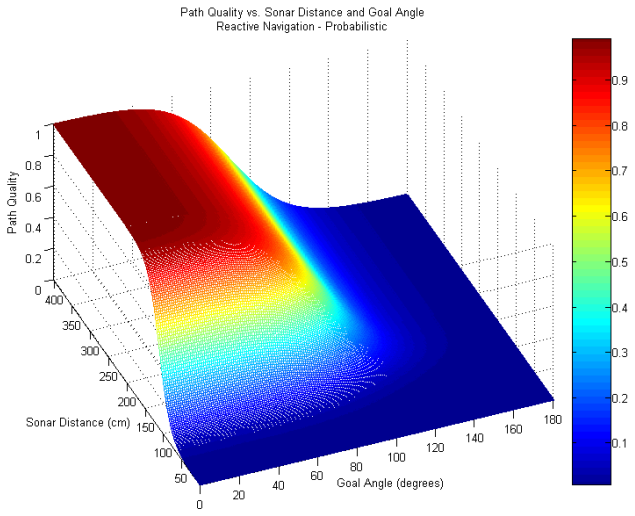

Figure 4.2: Probabilistic Output: A demonstration of the path quality output for varying angular distance of the potential path from the desired location and nearest obstacle distance.

robot navigates through the environment, even for dynamically changing desired locations. For example, the goal selection control level may deliver a dramatically different desired location at the next time step, however since the navigation algorithm is executed (and a simple path recalculated) at every time step, rather than complex multiple time step paths being designed for new destinations, the navigation control level is minimally affected by the sudden change in destination.

4.2 Adaptive Navigation

A significant drawback of reactive navigation is the engineered quality of its decision process. The entire state to action mapping is created by the system designer

either prior to operation entirely, or based on observations made during testing. However, rarely is that testing process capable of exploiting the entire range of states and the responsive actions a robot may experience in a continuous multi-robot navigation domain. In addition, interpretation of the sensor information available to the robot to maximize the performance may prove overly difficult or impossible in some situations.

The environment itself in which the robot is to navigate may also present a complex set of issues. For example, when the robot is operating in an environment where a system designer may observe and make corrections to the navigation technique quickly, it can be trivial to identify and repair a problem. On the other hand, if the robot is operating in an environment where direct communication with the system designed can take hours or days (e.g., extra-planetary exploration), it is impractical or impossible to make major adjustments to robot behavior after it has been tasked.

Adaptive navigation techniques present a potential solution to these problems. By creating a structure whereby the robot may modify its own interpretation of sensor information and act on its own based on environments it encounters after it has been tasked, the robot may adapt to situations it has never before encountered, and even adapt online to sensor or actuator failures. In this work, two adaptive navigation methods are examined in a contrasting context to the reactive navigation method presented above. These methods include:

Neuro-Evolution: A multi-layer perceptron (MLP) is used to interpret sensor information and determine what quality to assign to potential paths. An

evolutionary search technique is used to locate skilled MLPs (hereafter referred to as *neural networks*) in a population.

Reinforcement Learning: Several general RL techniques are implemented to evaluate robot state information and produce appropriate actions. This is done with state-action pairing either with direct estimation of value or with function approximation.

There are two distinct steps in implementing an adaptive control technique for a specific domain: constructing the learner (algorithm) and training the learner. Construction of the learner involves: a) determining an effective way of representing sensor information as the state, b) the way in which the algorithm interprets the state and stores knowledge, and c) interpreting the hypothesis for the optimal action the algorithm makes. Training the learner generally involves determining the best way to classify the actions taken by the algorithm as "good" or "bad" reflected by the next state the algorithm achieves. This classification is generally domain specific and does not include the method of presentation to the algorithm, which also must be addressed, and is discussed for this work in Section 4.3.

4.2.1 Neuro-Evolution

The first of the two steps as it applies to the neuro-evolutionary method of control involves determining the state and action space available to the baseline neural network. The state/action structure of the reactive navigation control presented in

Section 4.1 contains a beneficial approach to path selection. It is simple, which reduces computational complexity as well as the number of potentially unpredictable behaviors, and applies well to a hierarchical control structure.

To capture those benefits while injecting the benefits of adaptability into navigation control, the state and action spaces are maintained. Specifically, one state variable represents the angular distance of a potential path from the desired heading, and one represents the distance to the nearest impassable object in that path. The action space then contains a single state variable representing the quality of the path given that information. Therefore, for this work, the baseline network structure created is a two layer, sigmoid activated, artificial neural network, shown in Figure 4.3.

This network is run at each time step, for each potential path, generating a path quality function in a similar fashion to that presented in Figure 4.1. The difference is in the replacement of static predefined probability distributions by an adaptive artificial neural network.

The second step in implementing an adaptive control technique as it applies to this work involves application of an evolutionary search algorithm for ranking and subsequently locating successful networks within a population [34, 30, 49]. The algorithm maintains a population of ten networks, utilizes mutation to modify individuals, and ranks them based on a performance metric specific to the domain. The search algorithm used is shown in Figure 4.4 which displays the ranking and mutation steps.

In this domain, mutation (Step 2) involves adding a randomly generated num-

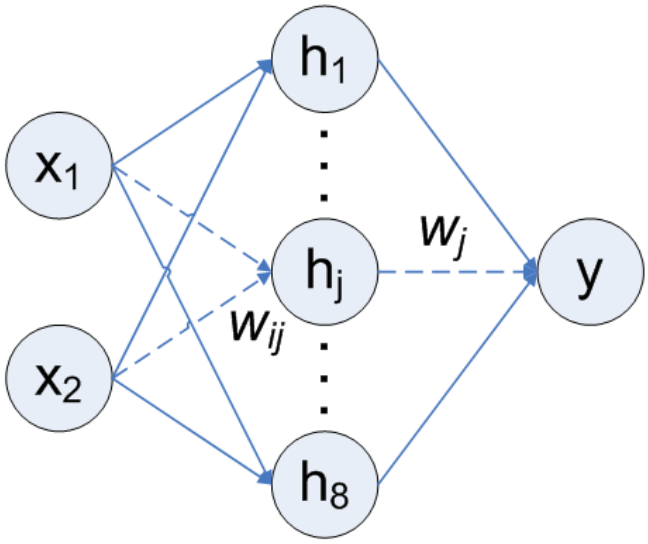

Figure 4.3: Baseline Network Structure: The baseline network structure for the neuro-evolutionary adaptive navigation technique. Two inputs describe the angular distance from the desired heading and the nearest obstacle distance, it contains a single layer of eight hidden units, and a single output represents the path quality.

ber to every weight within the network. This can be done in a large variety of ways, however it is done here by sampling from a random Cauchy distribution [4] where the samples are limited to the continuous range [-10.0,10.0]. Ranking of the network performance (Step 4) is done using a domain specific objective function, and is discussed in detail in Section 4.3.

4.2.2 Reinforcement Learning

The reinforcement learning technique of adaptive control is a structure of algorithm that must be formed such that it can be "rewarded" directly based on a predefined

Initialize N networks at T=0

For $T < T_{max}$ Loop:

1. Pick a random network N_i from population
   With probability ϵ: $N_{current} \Leftarrow N_i$
   With probability $1 - \epsilon$: $N_{current} \Leftarrow N_{best}$
2. Mutate $N_{current}$ to produce N'
3. Control robot with N' for next episode
4. Rank N' based on performance (objective function)
5. Replace N_{worst} with N'

Figure 4.4: Evolutionary Algorithm: An ϵ-greedy evolutionary algorithm to determine the weights of the neural networks. N_{best} and N_{worst} are the best and worst networks in the population, T_{max} is the number of episodes, $N_{current}$ is the network that is chosen at step T, N' is the modified $N_{current}$ that controls the robot at step T and ϵ is the probability of exploration.

objective function of the next state of the robot or the next state and action taken. In this work, the algorithm is rewarded based on a set of actions taken during a training episode. There are a great many reinforcement learning algorithm structures, however two general and commonly used structures were chosen for this work:

Direct Q Learning: This involves maintaining a table representing a discretized Q function that is based on state-action pairs, $Q(s, a)$. The intention is to produce a global maximum value for each specific state such that an appropriate action to take as a result of that state can be determined [36].

Function Approximation Q Learning: This method involves utilizing a function approximator rather than a table for representation of the $Q(s, a)$ function. There are several choices for function approximation including generalized linear structures or artificial neural networks. For this work, a neural network was chosen [42, 36].

The state and action spaces here are very important as well. In order to maintain comparability, the spaces are identical to that used in the probabilistic and neuro-evolutionary techniques utilized above and described in Sections 4.1 and 4.2.1. This structure, where path quality is assigned to potential paths, lends itself to reinforcement learning, as the maximum value of the action-value function corresponds proportionally to the maximum path quality and therefore automatically "chooses" the best path (action) based on current knowledge.

For the second step in implementing an adaptive navigation technique, updating the algorithm, two techniques were utilized [36, 20]:

Q-Learning: The temporal difference Q-learning approach requires no model, therefore does not require a known probability distribution from current to next states based on action taken. The action-value function $Q(s, a)$ is updated using the following equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

where $R(s)$ is the reward for the current state s, with s' and a' as the next

state and action respectively. The learning parameters α and γ dictate the learning rate and current knowledge discounting respectively.

SARSA: This learning method is quite similar to the temporal difference Q-learning method. It is different in that it is less aggressive by not maximizing over possible actions, resulting in the following update equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s) + \gamma Q(s',a') - Q(s,a)\right)$$

where all parameters are described as above.

Both learning techniques were paired with the two algorithm structures resulting in four unique adaptive navigation control techniques. For function approximation the action-value $Q(s,a)$ function cannot be directly updated, however the update is similar in structure to the update equations given above. The parameters of the function approximator in this case are the weights of the neural network. Referring to a general network structure such as that shown in Figure 4.3, the weight updates are done using the following equation [36]:

$$w_{ij} = w_{ij} + \alpha \left(R(s) + \gamma \max_{a'} \hat{Q}_w(s',a') - \hat{Q}_w(s,a)\right) \frac{\partial \hat{Q}_w(s,a)}{\partial w_{ij}}$$

where w_{ij} is an individual weight, $\hat{Q}_w(s,a)$ is the output of the function approximator given state s and action a, and s' and a' are the next state and action respectively. The above is for the temporal difference Q learning update; the

SARSA parameter update equation is similar, removing the next action maximization function.

It is important here as well to employ an explore/exploit technique to provide additional states and actions that may not have been observed and therefore are not included in current knowledge. An ϵ-greedy method similar to that used in the neuro-evolutionary algorithm is used in this application, as shown in Figure 4.5.

Initialize $Q(s, a)$

For $T < T_{max}$ Loop:

1. Capture current state s'
2. $r' \Leftarrow R(s)$
3. Update action-value function
   (or FA parameters)

   With probability ϵ: $a' \Leftarrow a_{random}$

   With probability $1 - \epsilon$: $a' \Leftarrow a_{function}$
4. $s, a, r \Leftarrow s', a', r'$
5. Execute a'

Figure 4.5: Reinforcement Learning Algorithm: An (ϵ-greedy) reinforcement learning algorithm. s, a, and r are the current state, action, and reward respectively. Next state, action, and reward are denoted with $'$. a_{random} is a randomly selected action where $a_{function}$ is the action given by the current $Q(s, a)$ and $R(s)$ is the reward assignment for being in state s.

4.2.3 Reinforcement Learning Difficulties in this Domain

Unfortunately, the reinforcement learning techniques described above did not provide desirable results when compared to the neuro-evolution technique. Though all of the above discussed methods learned and began to produce coherent behavior, the learning rate was approximately 150 times slower than that of the neuro-evolutionary algorithm and by the complex nature of the simulations it was intractable to further evaluate performance. This is likely a result of the continuous and stochastic nature of the domain for which there are reinforcement learning modifications that can be made to improve learning performance, which are discussed briefly in Chapter 6.

4.3 Problem Definition

For the simulations of the robot as described in Chapter 3, an arena of 5 meters square was created with a varying number of obstacles specific to the experiment and a specific destination was chosen in order to ensure that the desired behavior is the same for all evaluated algorithms. Various environments were generated to evaluate algorithm performance and are discussed further in the next section.

The training method is episodic in that the robot is allowed to operate for a fixed maximum amount of time (60 seconds in this work), is evaluated, then the environment is reset. When the robot reached the destination, the episode was considered complete and terminated. To prevent the adaptive algorithm from memorizing specific situations, the starting position is selected at random at the

beginning of an episode as is the initial robot heading. The range of positions from which the starting locations are selected varies for each experiment and is presented with the experiment results. The training is executed for 2000 episodes and each experiment is repeated 20 times for each algorithm. The results are then averaged for analysis.

The objective function for behavior ranking was designed to capture three important aspects of mobile robot navigation in unknown environments; 1) total path length the robot uses to reach the destination, 2) time the robot consumes reaching the destination, and 3) time the robot consumes recovering from a collision with an obstacle. These incorporate choosing the shortest path, executing it with greatest speed, and doing so in a safe manner. In order to convert the above to maximization rather than minimization, and support the constantly shifting initial conditions, the best possible behavior is incorporated, generating the following objective function:

$$G(s) = \alpha\left(d_{best} - d_{actual}\right) + \beta\left(t_{best} - t_{actual}\right) - \gamma\tau_{collision} \tag{4.1}$$

where d is the path length (best possible and episode actual), t is the time consumed, and $\tau_{collision}$ is the total amount of time spent recovering from collisions. α, β and γ are constants used to increase or decrease the respective terms' contribution to the overall function. For all subsequent navigation experiments these constants were found to give good behavior when set to 1.0, 10.0, and 10.0 respectively. The best path length and time vary by experiment and episode, there-

fore are calculated at the beginning of each episode using a Manhattan Distance concept [37] whereby the best length is a straight path from starting position to destination and the best time is that path executed at maximum robot speed. The best possible $\tau_{collision}$ is of course 0, resulting in the negative sign of that term. As a result, $G(s)$ is then always less than, or in very limited situations equal to, zero.

4.4 Experiments

Several experiments were designed to evaluate the navigation algorithms for a specific set of behaviors discussed in the problem definition. These progressively increased in difficulty and scope from basic navigation to a destination, through advanced navigation in cluttered environments with other robots. Following is a list of navigation experiments with their intentions:

Situational Environment: The robot must navigate to a destination with progressively more difficult, predefined, obstacle placement.

Dense Environment: The robot must navigate to a destination through an environment dense with obstacles.

Multiple Robot Environment: Multiple robots must navigate simultaneously to similar destinations within an environment dense with obstacles.

In addition to the above, in some situations sensor and actuator noise was injected to the simulation by randomly modifying readings or commands with a given scale percentage. The noise added is discussed in detail with the experimental

results. These experiments evaluate not only the navigation algorithm's ability to seek a destination, but safely and intricately navigate around obstacles in an unknown environment even in the face of dynamically moving obstacles (other robots) and sensor/actuator inaccuracies.

4.4.1 Situational Environment

In these experiments the algorithms are evaluated for simple to intricate navigation techniques. The first of these determines ability to relocate to a destination and the initial conditions are shown in Figure 4.6. The structure of this figure is utilized throughout the results presentation. It shows the to-scale experiment arena with sample robot (green square) and destination (blue circle). The yellow area shows the possible starting locations for each episode, the number of which is listed at the base of the arena.

Figure 4.7 shows the training results for the Total Clearance situation. The objective function shown in Equation 4.1 is plotted over training episodes as an average over the 20 iterations for each algorithm. The random driver, probabilistic driver, and the neuro-evolutionary algorithm are provided with error bars indicating the standard deviation of the data. As shown, the random driver sets a good baseline by taking random actions. The probabilistic driver does very well as all starting positions produce direct paths without the need for obstacle avoidance. Finally, the neuro-evolutionary algorithm is shown to learn quickly, performing statistically similar to the probabilistic driver within approximately 600 training

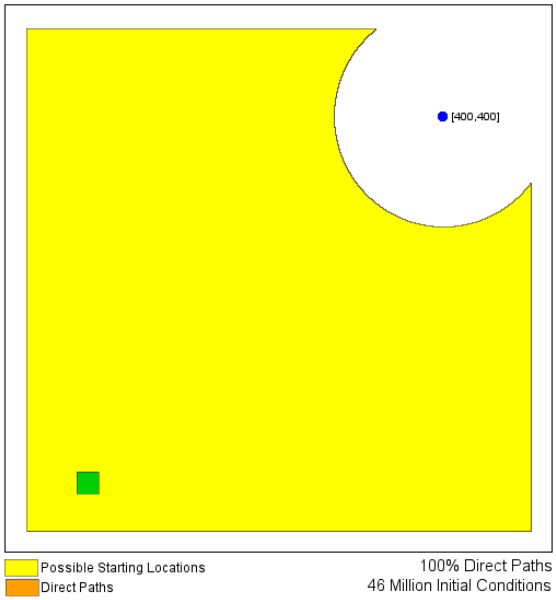

Figure 4.6: Total Clearance Situation: The experimental arena for the Total Clearance situation. The figure is to scale and shows a sample robot (green square), the destination (blue circle) as well as the possible starting locations for an episode (yellow area). The number of possible initial conditions as well as the approximate percentage of those being direct paths are listed at the base of the figure.

episodes. Because the adaptive navigation technique inherently contains some exploration, the mean is below reactive navigation in very simple situations. For example, the probabilistic navigator deterministically drives directly to the destination, where the neuro-evolutionary technique chooses random policies with ϵ probability.

Shown in Figure 4.8 is a sample control surface produced by the neural network after training, presented in a similar fashion to the probabilistic algorithm in Figure 4.2. What is interesting here is the different shape and structure of the surface as it compares to the probabilistic algorithm, while still producing successful

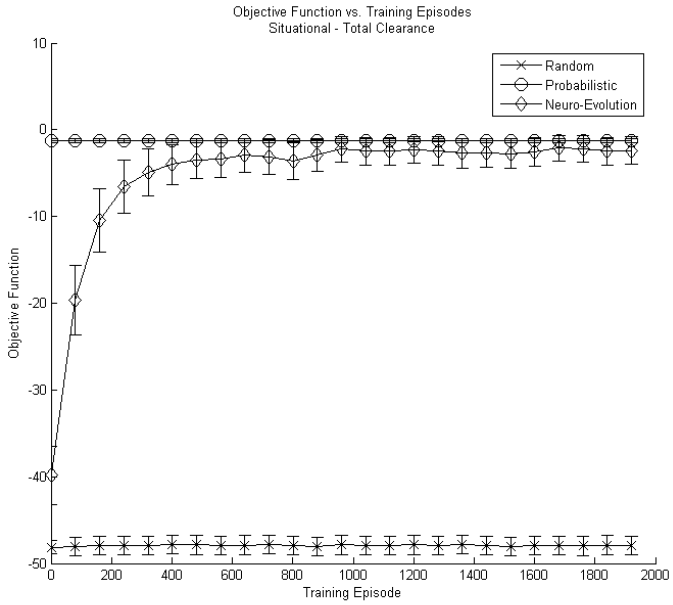

Figure 4.7: Total Clearance Training: The results of the training for the Total Clearance situation. The objective function is plotted for the random, probabilistic, and neuro-evolutionary algorithms as an average over 20 iterations.

navigation technique. The path quality remains high regardless of sonar distance when the potential path is direct to the goal, resulting from the lack of need to avoid obstacles in transit. The quality falls away quickly for rising angular distance with lower sonar readings however which forces the robot to be more selective with potential paths and slows the robot significantly for more careful navigation.

The next step was to install obstacles into the environment to reduce the number of direct path initial conditions and force the robot to avoid obstacles in transit to the destination. Figure 4.9 shows the experiment configuration presented in a similar fashion to Figure 4.6. The number of direct paths is reduced to approx-

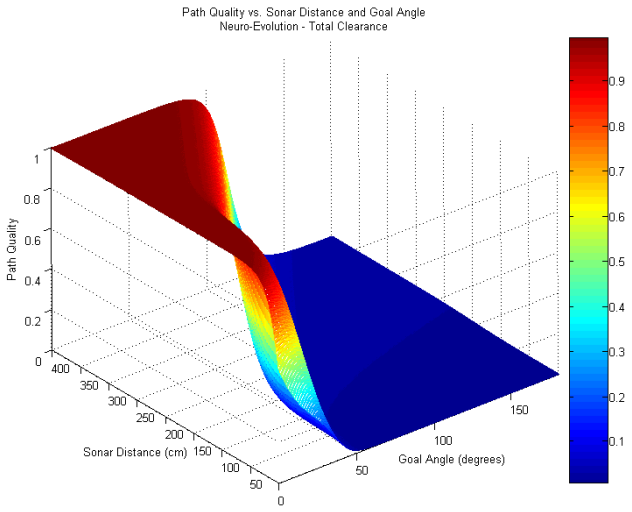

Figure 4.8: Total Clearance Output: A sample output of the neural network after training for the Total Clearance situation. The path quality is plotted over varying angular distance from the desired location and sonar distance.

imately 50% while the number of potential initial conditions remains the same. Therefore the robot still may begin an episode in a position to navigate directly to the goal, but has a 50% chance of being presented with an obstacle before reaching the destination. As shown in the results of training (Figure 4.10), the probabilistic algorithm still performs well and the neuro-evolutionary algorithm proves to learn quickly and produce effective behavior statistically similar to reactive navigation.

Finally, a difficult situation was configured where there were few potential direct paths to the destination and the robot is required to navigate carefully between obstacles in order to reach the destination. Figure 4.11 shows the layout of the obstacles as they relate to potential robot starting positions.

The results of training, shown in Figure 4.12, are interesting in that the neuro-

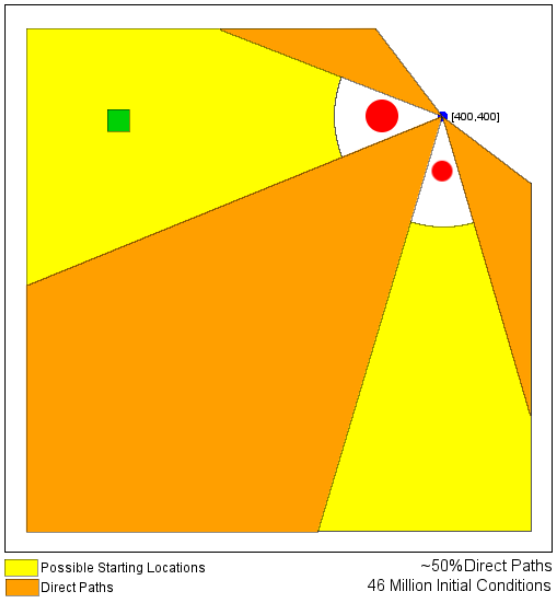

Figure 4.9: High Clearance Situation: The experimental arena for the High Clearance situation. The figure is to scale and shows a sample robot (green square), the destination (blue circle) as well as the possible starting locations for an episode (yellow area). The number of possible initial conditions as well as the approximate percentage of those being direct paths are listed at the base of the figure.

evolutionary algorithm appears to begin exceeding the performance of reactive navigation. In addition, the learning appears to take place in two phases: initially learning how to reach the destination then adjusting to avoid the obstacles it encounters. A sample output of the neural network is shown in Figure 4.13 and again is quite different from the probabilistic algorithm as well as the resulting surface from the Total Clearance situation. Again the path quality falls away very quickly as the robot points away from the desired location, in a more strict fashion than that of the Total Clearance situation, and a "shelf" exists for sonar distances greater than approximately 100cm.

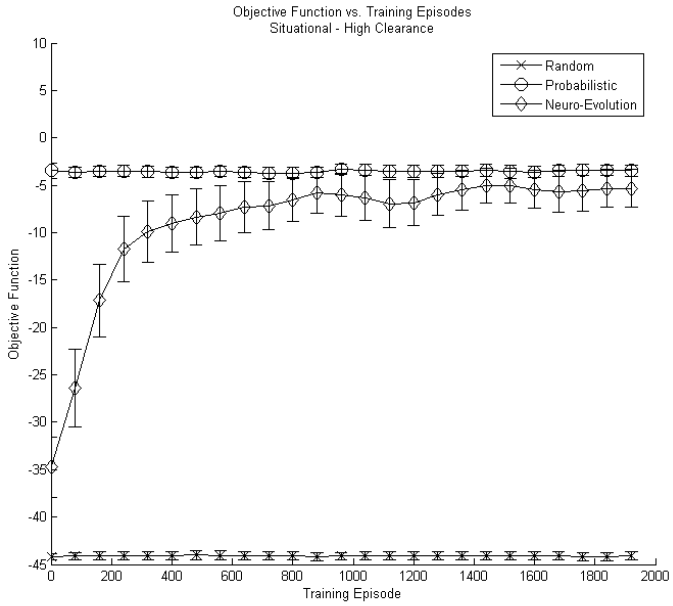

Figure 4.10: High Clearance Training: The results of the training for the High Clearance situation. The objective function is plotted for the random, probabilistic, and neuro-evolutionary algorithms as an average over 20 iterations.

These experiments progressively determined the capability of the neuro-evolutionary algorithm to learn how to navigate the robot in general, as well as avoid obstacles to reach a specified destination. The adaptive method of navigation proves to perform equivalently to predefined reactive navigation, generating different solutions for different environments.

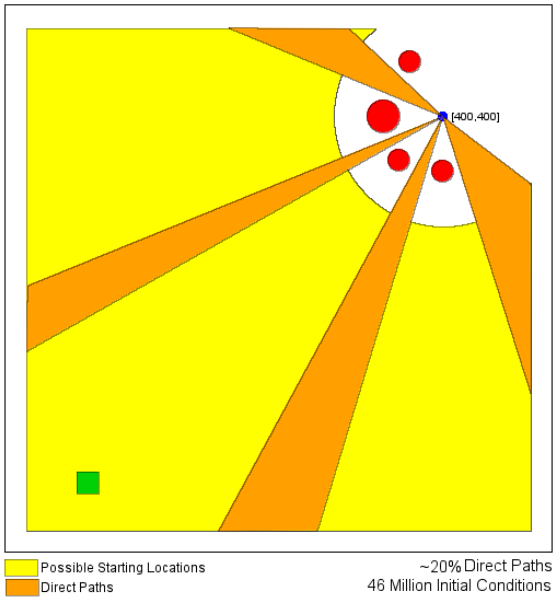

Figure 4.11: Low Clearance Situation: The experimental arena for the Low Clearance situation. The figure is to scale and shows a sample robot (green square), the destination (blue circle) as well as the possible starting locations for an episode (yellow area). The number of possible initial conditions as well as the approximate percentage of those being direct paths are listed at the base of the figure.

4.4.2 Dense Environment

The next step in the progression of analysis was to determine how the algorithms perform in a complex environment dense with obstacles. It is desirable to force the robot to navigate through the majority of the environment and reduce or eliminate the number of potential direct paths. To provide this, the range of starting locations was limited to areas on the opposing side of the experimental arena and 15 obstacles were placed arbitrarily throughout the environment. Figure 4.14 shows the environment configuration for these experiments. As shown, the starting po-

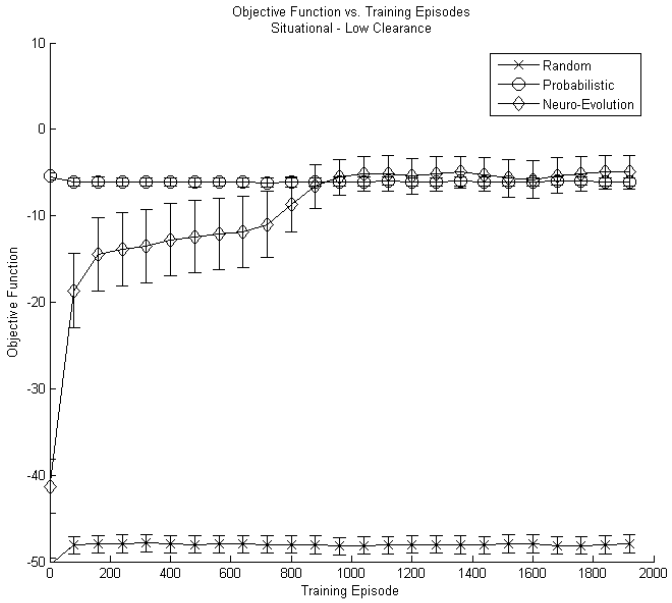

Figure 4.12: Low Clearance Training: The results of the training for the Low Clearance situation. The objective function is plotted for the random, probabilistic, and neuro-evolutionary algorithms as an average over 20 iterations.

sitions are narrowed to the outer edges of the arena, but the robot initial heading was still randomized at the beginning of each episode.

The results of training in this environment are very interesting. The neuro-evolutionary algorithm learns more quickly than in the situational experiments and exceeds the reduced performance of reactive navigation. This is a result of the algorithm being presented with more information during the training process (through richer sonar data), the reduced number of starting positions, and the discovery of more accurate obstacle avoidance technique over reactive navigation. The results of the training process are presented in Figure 4.15.

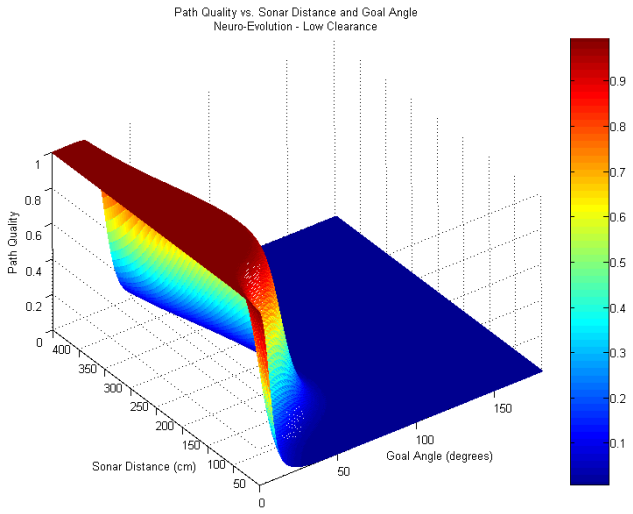

Figure 4.13: Low Clearance Output: A sample output of the neural network after training for the Low Clearance situation. The path quality is plotted over varying angular distance from the desired location and sonar distance.

4.4.3 Dense Environment with Noise

Previously the sensors and actuators produced ideal data and robot motion. This is not a realistic situation for physical robots, as all sensors contain stochastic differences in readings of the environment, and actuators may not produce exactly the intended robot motion. Therefore, random noise was injected into the sonar and inertial sensor data as well as the output of the navigation algorithm to the actuators. Specifically, 5% random noise was present from the beginning of the training and to simulate potential failures, the noise level was phased up to 10% over 200 episodes surrounding the 1000th episode (e.g., from 900 to 1100). The results of the training are presented in Figure 4.16.

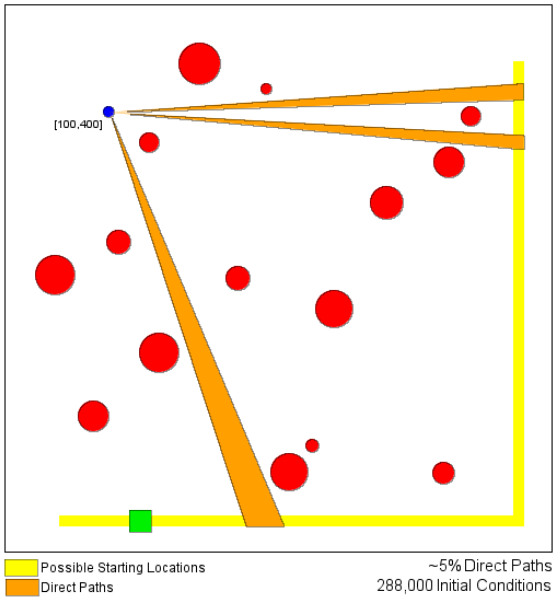

Figure 4.14: Dense Environment: The experimental arena for the dense environment. The figure is to scale and shows a sample robot (green square), the destination (blue circle) as well as the possible starting locations for an episode (yellow area). The number of possible initial conditions as well as the approximate percentage of those being direct paths are listed at the base of the figure.

It can be seen that the learning process is dampened as a result of the noise present at the beginning of the training process. It is important to note that as the increased noise is phased in, there is little or no effect in the performance of the neuro-evolutionary algorithm. Learning continues unimpeded to significantly outperform the probabilistic navigation algorithm which is strongly affected by the increased noise level in sensing and actuation. This is a result of the learning occurring while noise is present in the system such that good behavior is learned in spite of the noise and therefore an increase in the noise level during operation (once successful behavior has been learned) does not affect the neuro-evolutionary

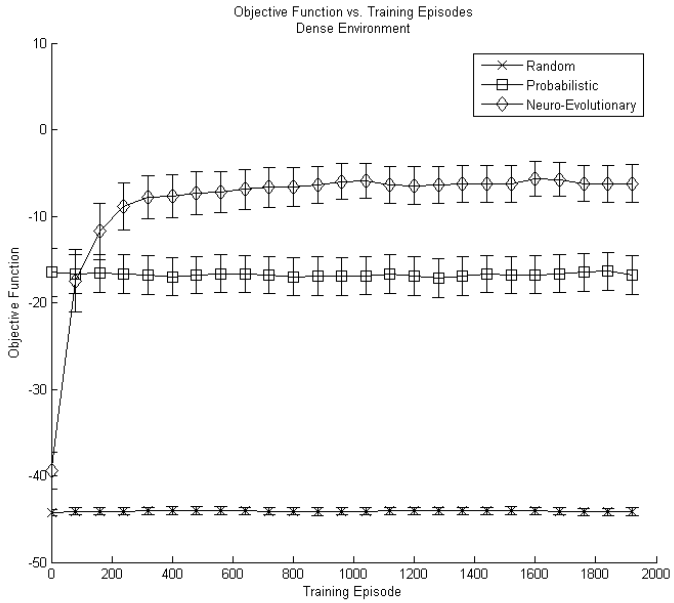

Figure 4.15: Dense Training: The result of the training in a dense environment. The objective function is plotted for the random, probabilistic, and neuro-evolutionary algorithms as an average over 20 iterations.

algorithm performance.

A sample of the control surface learned during the dense environment with noise experiment is shown in Figure 4.17. This surface is interesting in its aggressiveness. For example, the path quality remains large for a much greater range of paths facing away from the desired location, though a slight slope remains to ensure the tendency to point towards the destination. As the sonar distance approaches dangerously low values, the path quality falls away quickly regardless of where the robot is pointed, which demonstrates the algorithm's ability to avoid collisions while still driving as quickly as possible. The fact that the path quality falls away

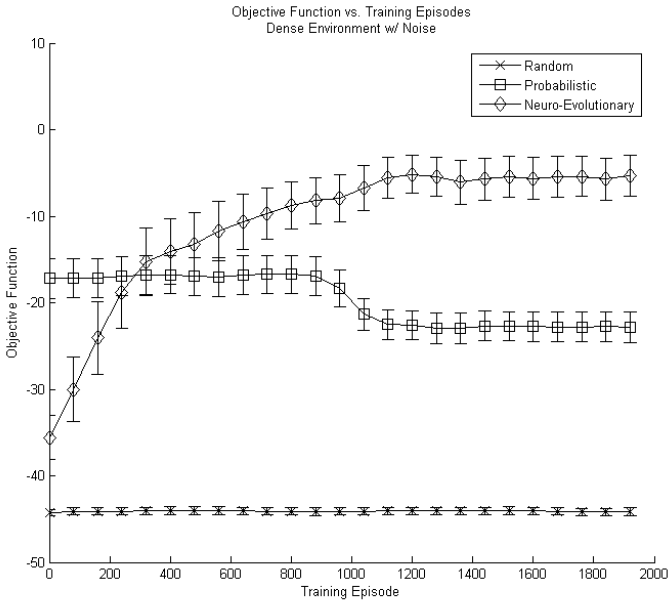

Figure 4.16: Noisy Dense Training: The result of the training in a dense environment while sensor and actuator noise was present. The objective function is plotted for the random, probabilistic, and neuro-evolutionary algorithms as an average over 20 iterations.

at large sonar distances (which should be safer) is explained by the lack of exposure to that portion of the state space. For example, in this dense environment the sonar was almost always reporting distances to obstacles and walls under 300cm.

These experiments demonstrate the ability of the neuro-evolutionary algorithm to outperform the probabilistic navigation algorithm in environments densely populated with obstacles and to learn effective navigation techniques when the sensors and actuators are not operating at ideal levels.

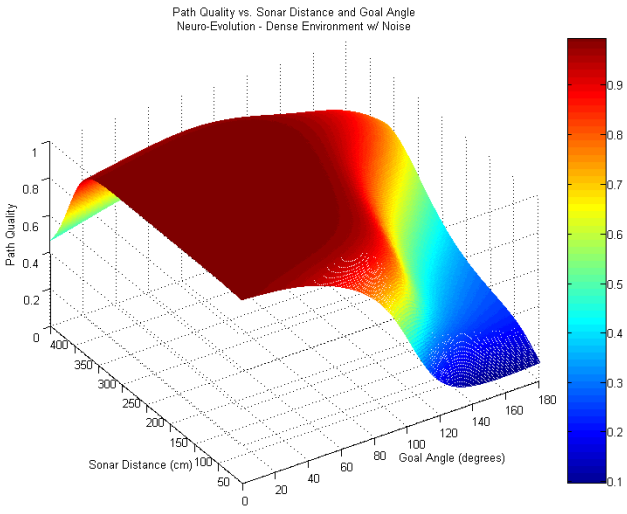

Figure 4.17: Noisy Dense Output: A sample output of the neural network after training for the dense environment with noise experiment. The path quality is plotted over varying angular distance from the desired location and sonar distance.

4.4.4 Multi-Robot Environment

The intention of the hierarchical control structure evaluation in this work was for application to multi-robot domains. Therefore it follows that a determination of the performance of the adaptive navigation technique in multi-robot settings was required. For this experiment, 5 robots were placed in the same dense environment presented in the last section. The range of initial conditions is similar, changing only in that robots are not allowed to be placed atop one another at the beginning of the episode. All the robots in the domain must proceed as previously discussed, each learning how to navigate to a destination (near to but not the same as the others) while avoiding obstacles.

When the neuro-evolutionary algorithms operating on the robots were started from scratch (random weights) at the beginning of the experiment, only one of the robots was able to learn any coherent behavior above taking random actions, and even then had a difficult time learning to avoid obstacles (and other robots). This problem is addressed further in Chapter 5. Alternative to all robots learning at once, one robot was run utilizing a neuro-evolutionary algorithm while the others ran the reactive navigation technique to determine whether the problem was with the learning algorithm itself, or with the dynamics of a multi-robot environment. The results were similar to those presented in Section 4.4.2. However, this method is not conducive to the multi-robot intentions of the work, where all robots are adaptive, learning agents.

Therefore a second alternative was chosen where a neural network is "bootstrapped" from a known functional algorithm, in this case the probabilistic algorithm, and distributed across the robots to determine if an improvement can be made. Bootstrapping was done using standard neural network error backpropegation [36, 13] where the error is calculated as the mean-squared difference between current network output and the probabilistic algorithm output. The process is run until the error drops below 20%, as the intention is not for the network to duplicate the probabilistic algorithm behavior exactly.

The process is run five times, one for each robot, then the evolutionary search algorithm is replaced and the experiment begins. Figure 4.18 presents the results of the training under these conditions. Though each robot was ranked based only on its own performance (with Equation 4.1), the sum of all performances was used for

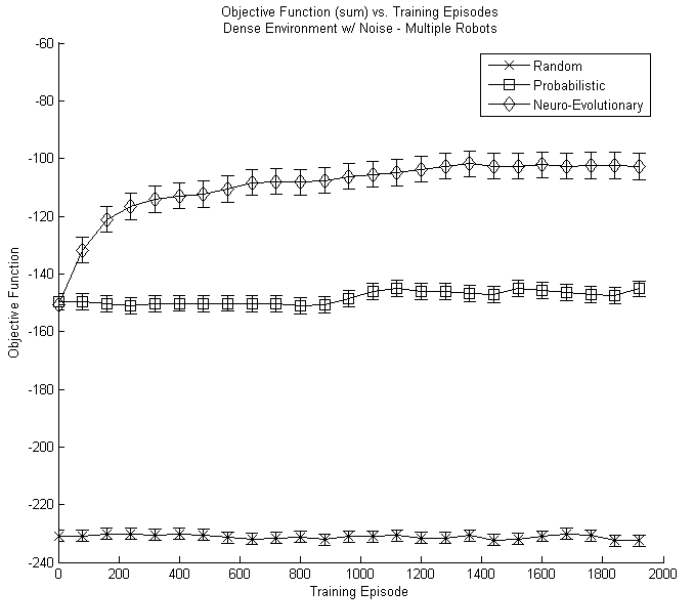

Figure 4.18: Multi-Robot Training: The result of the training in a dense environment while 5% sensor and actuator noise is present and 5 robots are active. The sum of the individual objective functions is plotted for the random, probabilistic, and neuro-evolutionary algorithms as an average over 20 iterations.

presentation purposes. It is shown that the neuro-evolutionary algorithm begins close to the performance of the probabilistic algorithm, however quickly learns to exceed the performance to a significant degree. Sensor and actuator noise (5%) was present in this experiment as well, but was not increased during the training process.

Shown in Figure 4.19 is a sample control surface from one of the robots after training. All five surfaces were different, one is presented here for brevity. It is clear that the roots of the surface reside with the probabilistic algorithm (Figure 4.2), but it has been significantly modified into one similar to that produced in the Low

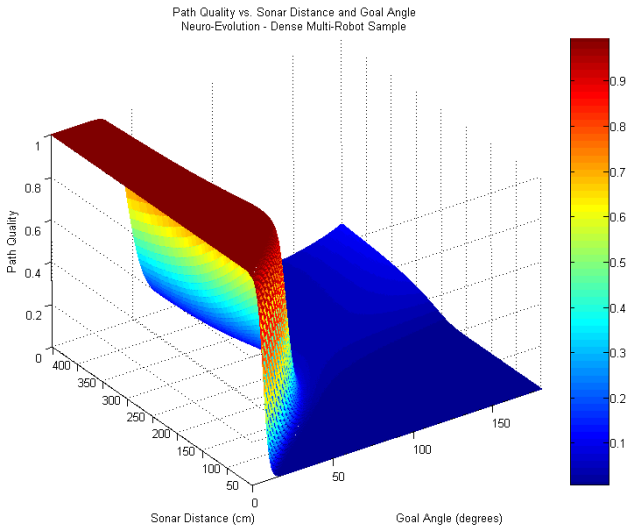

Figure 4.19: Multi-Robot Output: A sample output of the neural network from one robot after training in a dense environment. The path quality is plotted over varying angular distance from the desired location and sonar distance.

Clearance situational experiment (Figure 4.13).

This experiment shows that the neuro-evolutionary adaptive navigation technique is successful in environments where multiple robots are in operation, improving performance over that of probabilistic path selection. However, the experiment also reports that the learning process is complicated to the degree of disruption if all robots begin the learning process from scratch. Bootstrapping the learning algorithm before the learning process is begun was shown to successfully overcome this limitation.

4.5 Navigation Discussion

In this chapter algorithms were presented and evaluated to achieve the control tasks required in the mid level of the hierarchical control structure. The random driver algorithm established a good baseline by assigning random values to path quality. The probabilistic driver used pre-engineered probability distributions to determine the direction and safety of potential paths of travel to a destination. Several reinforcement learning techniques were employed under the same state and action space structures however were intractable in their learning speed.

The neuro-evolutionary adaptive navigation technique was shown to *successfully learn to perform the tasks required including basic navigation and advanced obstacle avoidance.* In defined situations, the algorithm performed statistically similar to that of the probabilistic algorithm and *in dense environments was shown to quickly exceed the performance.* Included in the dense environment experiment was the ability of the algorithm to overcome the affects of sensor and actuator noise, where *the neuro-evolutionary algorithm was shown to be robust.*

The algorithm also demonstrated the ability to improve upon current knowledge when placed in a noisy, dense environment where multiple robots were in simultaneous operation, if prepared via bootstrapping. This lends itself well to the coordination domain, however as shown in the forthcoming chapter, the less stochastic probabilistic navigation method must be used while evaluating coordination behavior (to isolate the hierarchical control levels). This is the case only in development, once all levels have been developed and trained, adaptive techniques

may be used in both the mid and high levels.

Chapter 5 – Coordination

Often it is beneficial to examine the potential application of multi-robot teams in complex exploration applications. A single robot may be able to effectively manage the task at hand, however it may consume a great deal of time or resources in order to accomplish the goal. In designing teams of robots to coordinate with one another, exploration of an unknown environment may be accomplished in a much shorter time, and the resources required of individuals may be dramatically decreased.

As an example, a single robot exploring the surface of a planet requires a great deal of time to navigate to a destination and a great deal of resources to examine that destination. Sequentially visiting each location, potentially large distances apart, and spending time examining each with a myriad of sensors consumes both travel time and power. If the task were placed to a large team of robots, each individual would need to travel much smaller distances to contribute to the exploration, and potentially each robot could be given different sensor types to further reduce power and examination time requirements. Finally, it generally poses a greatly reduced impact on the exploration mission if an individual of a team fails compared to that of a single explorer.

5.1 Problem Definition

The multi-robot information gathering problem used for this work consists of a set of robots that must observe a set of points of interest (POIs) within a given time window [49]. These requirements are gathered in the top level of the hierarchical control grouping, isolated from the navigation requirements. The POIs have different importance to the system, and each observation of a POI yields a value inversely related to the distance the robot is from the POI. Though multiple observations of a POI may in some instances be valuable (e.g., different spectra, different angles), the case where the robots are homogeneous and hence only the observation of the closest robot contributes to the system objective function was investigated. First introduced in [49], the work in this thesis expands on the problem by introducing physical constraints on robot motion (i.e., robots consume space and time while in operation and have limited articulation) in addition to a separate navigation structure (i.e., previously all control was performed by a single algorithm) and limited sensing (i.e., information degradation over distance).

Given the problem specification above, the system objective function is defined as [49]:

$$G(s) = \sum_{j} \frac{V_j}{\min \delta(L_j, L_i)} \tag{5.1}$$

where V_j and L_j are the value and location of POI j respectively, L_i is the location of robot i, and the distance metric δ is the squared Euclidean norm, bounded by a minimum observation distance, $\delta_{\min}$:

$$\delta(x, y) = \max(\|x - y\|^2, \delta_{\min}^2) \tag{5.2}$$

where $\delta_{\min}$ represents the robot "bumping" into the POI or another robot.

It was discovered in Section 4.4.4 that without careful initialization, the neuro-evolutionary adaptive navigation technique can produce highly stochastic behavior in multiple robot domains. To isolate robot behavior, in this case down to goal selection and coordination only, the more deterministic probabilistic navigation method is employed throughout. In this problem then, robot coordination is achieved through the goal selection mechanism only, for each robot. Because the system objective is based on having close observations of the POIs, only one of many robots that aim for the same POI contributes to the system objective. The goal selection mechanism is then critical for success in this domain.

5.2 Goal Selection for Coordination

The goal selection algorithm is responsible for determining the destination. The selection process involves mapping the sensor inputs to an x, y translation relative to the current position of the robot. Each robot utilizes a two layer sigmoid activated artificial neural network to perform this mapping [18].

The inputs to this function approximator are the four POI sensors (Equation 5.3) and the four robot sensors (Equation 5.4), where x_q^{POI} and x_q^{ROBOT} provide the POI and robot "richness" of each quadrant q, respectively, V_j and L_j are the value and location of POI j respectively, L_i is the location of the current robot

i and $\theta_{j,q}$ is the separation in radians between the POI and the center of the sensor quadrant.

$$x_{i,q}^{POI} = \sum_{j} \frac{V_j}{\delta(L_j, L_i)} \left(1 - \frac{|\theta_{j,q}|}{(\pi/4)}\right) \tag{5.3}$$

$$x_{i,q}^{ROBOT} = \sum_{k,k\neq i} \frac{1}{\delta(L_k, L_i)} \left(1 - \frac{|\theta_{k,q}|}{(\pi/4)}\right) \tag{5.4}$$

These sensors are idealized versions of the thermal and sonar sensors discussed in Section 3.2. The structure of the information provided to the function approximator is essentially the way proposed in [49], however the way in which the data is gathered is more strictly dependent on the realities of the sensors chosen. For example, sonar information becomes more vague (the "echo" spreads out) and thermal information declines (the "intensity" spreads out) as the distances increase from the sensor. These restrictions are placed on the above data before structuring for input to the function approximator.

Two outputs from the function approximator indicate the distance to travel on the axes parallel and perpendicular to the current robot heading. The distances are translated into a "desired location" which is then given to the navigator for execution. At that point the neural network (similar to Figure 4.3) is no longer queried until the navigator reports that the location has been reached within a radius equal to $\delta_{\min}$ used in Equation 5.2. This ensures that the neural network is queried again if it chooses a desired location outside the boundaries of the environment, or "inside" a POI or other robot.

The weights of the neural network are adjusted through an evolutionary algorithm [2, 3, 30]. The algorithm is identical to that used for the neuro-evolutionary navigation technique in Figure 4.4, however describing instead the algorithm for the robots' goal selection training. One of the keys in the success of such an algorithm is in determining the objective function that provides the selection mechanism (step 4).

In these experiments, three different objective functions [49] to rate the performance of the goal selection algorithms were evaluated: the system objective function which rates the performance of the full system; a local objective function that rates the performance of a "selfish" robot; and a difference objective function that aims to capture the impact of a robot in the multi-robot team. More precisely, these three functions are:

- The system objective given in Equation 5.1. This objective reflects the performance of the full team. Though robots optimizing this objective guarantees that the robots all work toward the same purpose, robots have a difficult time discerning their impact on this function, particularly as the number of robots in the system increases.

- The local objective given by:

$$P_i(s) = \sum_j \frac{V_j}{\delta(L_j, L_i)} \tag{5.5}$$

  This objective reflects the performance of the robot operating alone in the environment. Each robot is rewarded for the sum of the POIs it alone ob-

served. If the robots operate independently, optimizing this objective would lead to good system behavior. However, if the robots interact frequently, then each robot aiming to optimize its own local function may lead to competitive rather than cooperative behavior.

- The difference objective given by:

$$D_i(s) = \sum_j I(j,i) \left(\frac{V_j}{\delta(L_j, L_i)} - \frac{V_j}{\delta(L_j, L_{i'})} \right) \tag{5.6}$$

where $I(j,i)$ is an indicator function equal to 1 when robot i is the closest observer of POI j and $L_{i'}$ is the location of the next closest observer of POI j. This objective reflects the impact a robot has on the full system [2, 49]. By removing the value of the system objective where robot i is inactive, the difference objective computes the value added by the observations of robot i alone. Because only POIs to which robot i were closest need this difference computed, this objective is "locally" computable in most instances.

5.2.1 Objective Function Properties

In multi-robot teams, the properties of the objective functions selected for the individual robot training process are vital to generate effective local behaviors as well as coordination within the team. Each objective function must therefore be evaluated with quantitative measures to determine their respective effectiveness. Two measures used to perform this evaluation in the coordination experiments

were *factoredness* and *learnability* [50, 53].

The concept of factoredness is derived from the requirement of the objective function used locally to maintain alignment with the system level objective. This indicates that an action taken by a robot within the team to improve its own objective also improves the system objective. Therefore an objective function that has high-factoredness means that if a set of robot behaviors is evolved to maximize its local objective function, it will also maximize the system level objective.

In addition, the concept of learnability is derived from the complexity, and local availability, of information. This indicates that it is desirable for an objective function to provide information directly pertinent to the observations available to a robot after an action that it takes. As a result, an objective function that has high-learnability means that it contains "clean" signals regarding the change of state as a result of an action taken.

These concepts can be described intuitively by referring to the objective functions described above. For example, if individual robots are evolved using the system level function directly (Equation 5.1), they will be by definition highly-factored. However, the objective function contains information dense with the results of actions taken by the entire team, and therefore contains a highly noisy signal regarding the individual robot's effect on the system. Consequently, using the system level objective for evolution provides low learnability.

Conversely, using the local objective function for training (Equation 5.5) has high learnability. It indicates to robot i only the result of its own actions taken within the system, none of the actions of the other members of the team. Though

the signal is related to the system level objective, and provides a clean signal, it has low factoredness as it does not indicate how the robot effected the team as a whole. This objective function therefore only provides for "greedy" behavior evolution.

Both properties have beneficial contributions to the learning process. Therefore it is desirable to develop an objective function that has both high factoredness and high learnability. This is done as shown in Equation 5.6 by beginning with the original system level objective and removing the portion of the system level objective where robot i was inactive. This provides a signal indicating how robot i affected the system as it compares to other team members (high factoredness), and indicating the direct result of the action taken by robot i (high learnability). Using the difference objective therefore evolves behavior in individual robots to maximize the system level objective while coordinating within a team.

5.3 Experiments

For all simulations, the robots start in the middle of the arena and POIs are distributed randomly throughout the remainder of the environment with varying value. In placing POIs for an experiment, as well as moving them for the dynamic environment, POIs were allowed to be adjacent to one another, but not located within the same location. In this way, discrete POI locations and values were maintained.

As with the navigation experiments discussed in Section 4.4, in order to prevent

the robots from memorizing specific actions and (x, y) locations in the arena, each robots' initial heading was randomized before the start of the episode. The robots then operate for the length of the episode (15 seconds in the reported experiments), as described in Section 3.1. For these experiments, the minimum observation distance δ_{min} is set to half the length of the robot, or 10cm.

At the end of the episode, all objective functions are calculated and the evolutionary search algorithm is executed on each robot. The objective functions are used to rate the performance of the neural network used for goal selection, which is then added to the population of networks from which the robot selects its next goal selector for an episode. For the analysis presented in following sections, all the POIs were fixed (except for the "dynamic environment" presented in Section 5.3.4), and with the same value, for the entire training process. The learning converged in approximately 1000 episodes in all cases. Each run (set of 1000 episodes) was repeated 20 times, and the average results over those 20 runs along with the differences in the mean are reported.

In this work, the behavior of the robots in a variety of different environmental conditions is explored. In particular focusing on the following four scenarios:

1. POI Poor Environment: There are as many POIs as there are robots, meaning that each robot could concentrate on a single POI (Section 5.3.1).

2. POI Rich Environment: The POIs outnumber the robots, which forces the robots to select paths to observe more than one POI, and concentrate on different regions to maximize total POI observations (Section 5.3.2).

3. POI Density: The number of POIs vary from 5 to 40 for 10 robots (Section 5.3.3).

4. Dynamic Environment: At each episode, three POIs have a 10% probability of moving to a random location in the environment. This produces a new environment on average every 100 training episodes. Two situations are addressed (Section 5.3.4):

    a) A POI Poor environment (20 robots and 20 POIs), where the number of POIs matches the number of robots.

    b) A POI Rich environment (10 robots and 30 POIs), where the number of POIs largely exceeds the number of robots.

5.3.1 POI Poor

Figure 5.1 shows the starting situation for 20 robots and 20 POIs. The robots are clustered in the center all facing different headings. The POIs are shown as labeled circles with a number indicating their value to the system objective. The black circle surrounding the POI represents the $\delta_{\min}$ minimum observation distance. Figure 5.2 displays the results for 20 robots and 20 POIs. These results are typical of POI poor environments and are qualitatively the same from 10 to as many as 30 robots and POI combinations.

When the number of robots matches or exceeds the number of POIs, the robots can simply move for the closest, highest value goal, and since there are more (or

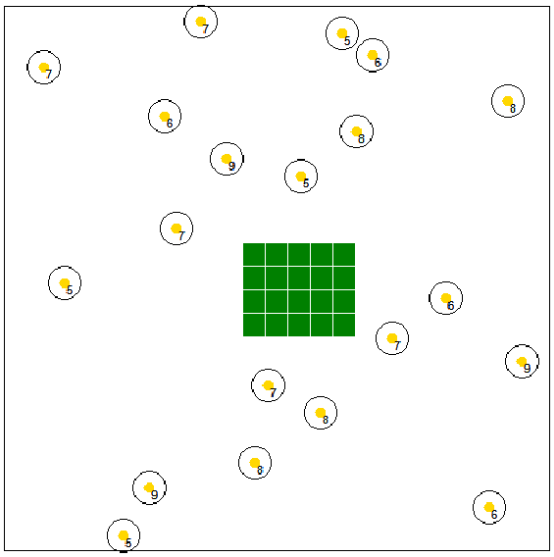

Figure 5.1: POI Poor Situation: Starting situation for 20 robots and 20 POIs. All robots are together at the middle of the arena, POIs are represented by solid circles. The number by each is the value of the POI, and the circle surrounding the POI represents the δ_{min} minimum observation distance. All objects are to scale for the physical environment being simulated.

an equal number of) robots than POIs, all POIs are likely to be visited. Therefore, locally greedy behavior results in satisfactory system behavior, as there is no "congestion" in this setting. Robots using selfish objective functions therefore perform slightly better than robots using the difference objective function because factoredness is not an issue, and though both objective functions have high learnability, the selfish objective has slightly higher learnability. Robots using the system objective function however perform poorly because they have low learnability (i.e., even with as few as 10 robots, the signal is masked by the noise of the actions of the other robots in the system).

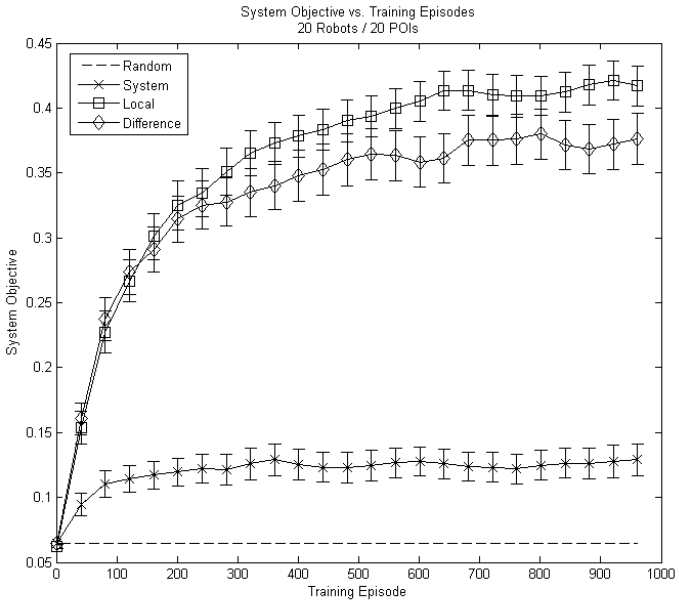

Figure 5.2: POI Poor Training: The number of robots matches the number of POIs. The system objective is plotted over the number of episodes for each of the objective functions used in training. Using the local objective produces greedy behavior which performs well with limited goal choices.

5.3.2 POI Rich

For 10 robots and 40 POIs, the performance of robots using the selfish objective deteriorates when compared to the previous setting. Figure 5.3 displays the results when the number of POIs exceeds the number of robots in the environment. Here the local objective leads the robots to act selfishly and pursue the highest value goals. However, unlike the previous case in a POI poor environment, this behavior does not produce good system level behavior. This is because robots acting competitively instead of cooperatively causes "clustering" around high value goals,

which in turn causes large portions of the available exploration space to be ignored.

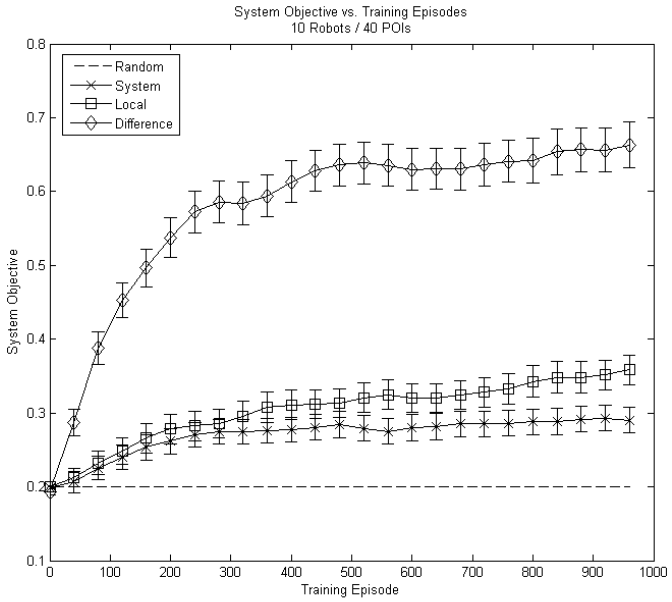

Figure 5.3: POI Rich Training: The number of POIs exceeds the number of robots. The system objective is plotted over the number of episodes for each of the objective functions used in training. Using the difference objective for training produces significantly better behavior.

The difference objective, on the other hand, performs well in this setting. In terms of the system properties discussed in Section 5.2.1, the system evaluation function has poor learnability and the selfish objective function has poor factoredness. Only the difference objective has both high learnability and high factoredness, which allows robots using this objective function to successfully select goals that contribute to the system.

5.3.3 POI Density

The results of the previous section demonstrate that the ratio of POIs to robots is a critical parameter in determining which objective functions promote coordinated system behavior. In this section, this issue was directly investigated by varying the number of POIs from 5 to 40 for a 10 robot system. Figure 5.4 shows the performance of all three objective functions at the end of 1000 episodes.

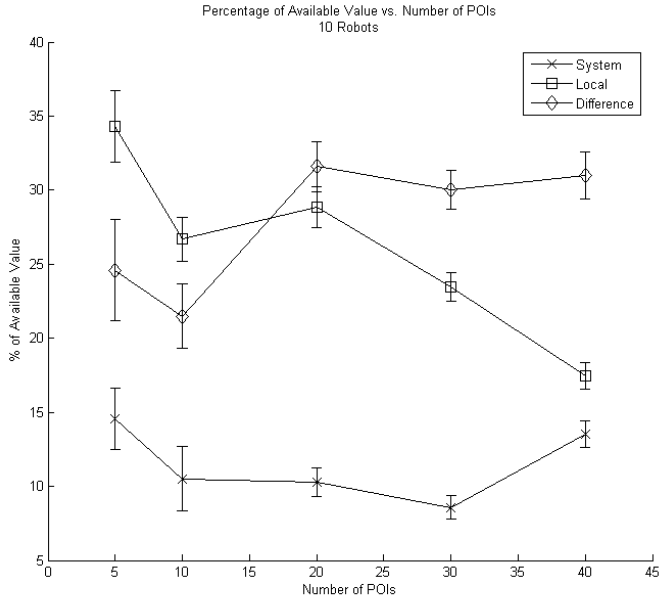

Figure 5.4: POI Density Response: The percentage of available system objective value gathered is plotted over number of POIs for 10 robots.

Robots using the system objective yielded poor performance across the board, where only a few robots made good decisions in any one setting. Robots using the local objective performed well for simple problems where robot interactions

are less important (with 10 robots and 5 POIs, selfishly aiming for the high valued POIs is satisfactory behavior), but their performance declined when the system required coordination.

In contrast, robots using the difference objective produces behavior similar to the local objective in situations of lower difficulty, but the relative performance improves to overtake the local objective when the difficulty of the problem increases. This is because the behavior promoted by the difference objective balances goal selection and goal "crowding" preventing the robots from seeking the same POIs. Figure 5.4 demonstrates this clearly, by showing the percentage of system value available gathered during each episode as the number of POIs increased.

5.3.4 Dynamic Environment

In addition to the static environments analyzed in the previous three experiments, the performance of the robots in a dynamic environment was explored. In this setting, there was a 10% probability that three of the POIs were relocated to a new random spot within the test arena at each episode. In doing so, the environment was almost entirely rearranged on average every 100 episodes. To ensure that the overall system value available remained the same, preventing the need for normalization, the POI values were not changed from the initial random generation. The robots still began an episode in the middle of the arena, their starting headings were still randomized at every episode, and each episode consisted of 15 seconds.

5.3.4.1 POI Rich

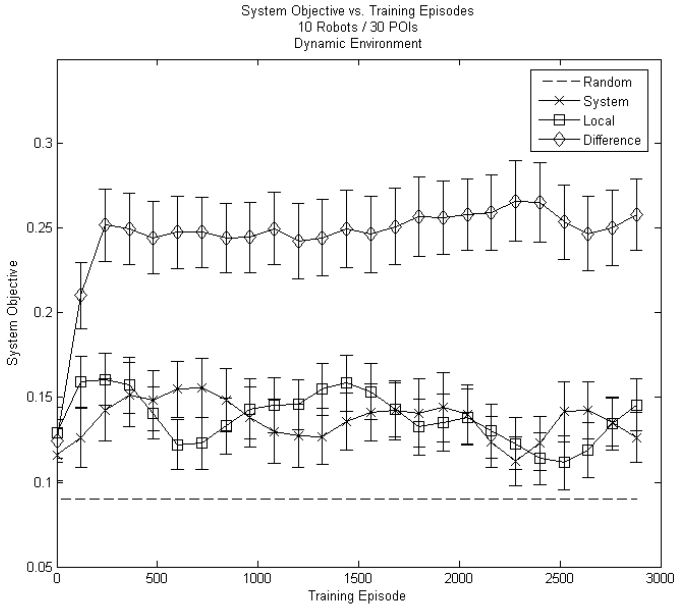

Figure 5.5: Dynamic Environment, POI-Rich: At each episode, three of the POIs had a 10% probability of moving to a random location within the environment. The system objective for each of the training sessions is plotted against the number of training episodes. The results are an average over 20 sessions for each training objective. Note here that the simulation was run for 3000 training episodes instead of 1000 in order to facilitate many environment changes and emphasize the difference objective consistency.

Figure 5.5 shows the results for the dynamic environment with 10 robots and 30 POIs. The increased variability in the mean is a direct result of the dynamic nature of the environment. For example, the environment is randomized, therefore an environment potentially differing to a great extent is encountered at corresponding training episodes for different runs.

Robots using either the system or the local objective functions performed poorly for different reasons. Indeed, they performed only slightly better than using random goal selection, increasing and decreasing as the environment changes between benefitting learnability (local objective peaks) and factoredness (system objective peaks). The drop in performance for the local objective in particular is interesting in this case: *Unlike in the static environment, robots do not learn to settle for low valued POIs, and pursue the high valued POIs more stubbornly, resulting in low system performance.* After repeated failures, robots using the local objective in static environments generally aim for low valued POIs, but such a trade-off is not discernible in the dynamic environment.

5.3.4.2 POI Poor

Figure 5.6 shows a dynamic environment situation where the number of POIs is equal to the number of robots. This represents a dynamic version of the situations presented in Section 5.3.1, or a POI-poor environment. Here it is shown that while the local objective performs well in simple, static POI poor situations, it breaks down in a similar though dynamically changing environment. Therefore, greedy behavior no longer benefits the overall system because not all POIs are likely to be visited, indicating that a dynamic environment reproduces the "clustering" effect that the local evaluation had produced in the static POI rich environment.

Robots using the difference objective however performed well and remain consistent throughout the changing environment. Small fluctuations in the difference

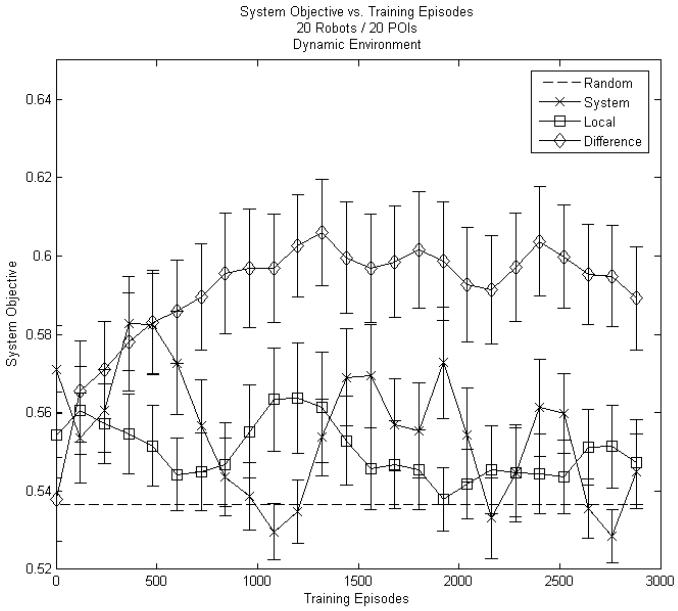

Figure 5.6: Dynamic Environment, POI-Poor: At each episode, two of the POIs had a 10% probability of moving to a random location within the environment. The system objective for each of the training sessions is plotted against the number of training episodes. The results are an average over 20 sessions for each training objective.

objective are explained by the environment changing significantly to a more difficult situation (e.g., high value POIs very near the edges of the arena). This demonstrates a key feature of using the difference objective: balancing factoredness and learnability allows the robot to maintain overall good behavior even though it must deal with the constraints of robot motion and navigation as well as a dynamically changing environment.

5.4 Coordination Discussion

By segmenting the tasks, goal selection is done independently of robot motion and navigation constraints, and therefore is more capable of complex high level tasking than if all control tasks were being performed by a single learner. In this approach, the high level controller is put to "sleep" after a goal selection decision has been made. Though this presents an injection of noise into the input signals (e.g., the goal selector possibly "wakes up" in a slightly different location than previously chosen), using an artificial neural network as an input/output function approximator proves robust to this noise. As a consequence, the high level controller is allowed to focus on learning behavior beneficial to the system, rather than confounding high-level decisions with detailed navigational instructions.

The results show that control task separation is successful in multi-robot teams operating in dense and POI rich environments, as well as in dynamic environments, where the richness of the POIs within the environment has less effect. In particular, the use of the difference objective provides good performance across a variety of environment settings. However only in the simplest case did the local objective provide good behavior (i.e., when the number of robots matches or exceeds the number of goals). When either the number of POIs was increased or the environment became dynamic, the performance of the local objective dropped rapidly, becoming counterproductive to producing coordinated behavior.

In contrast, the results show that utilizing a difference objective function that provides a learnable signal which maintains alignment with the system objective

produces up to 54% better behavior in static situations, as well as up to 49% better behavior in dynamic situations over using the traditional system objective or local objective.

Chapter 6 – Conclusion

The work presented in this thesis addressed three primary technological applications with the intention of utilization in multiple robot coordination domains, specific to resource limited robots. The coordination domain involved robots required to navigate a complex unknown environment with the goal of coordinating with others for information gathering, where resources (involving power, computation, and sensing capabilities) were limited. The control flow was grouped using a hierarchical control structure to isolate requirements for navigation and goal selection from each other such that each level could be designed and evaluated independently. Two primary investigations were performed under this hierarchical structure:

1. **Adaptive Control for Navigation:** Adaptive learning techniques were implemented for the navigation and goal selection control requirements in order to establish ability as it relates to placement within a hierarchical control structure, implementation on resource limited platforms, and within complex coordination tasks.

2. **Objective Function Evaluation for Robot Coordination:** Under the restrictions of simulated physical robot motion and hierarchical control structure, three previously established objective functions were implemented and evaluated as they apply to robot coordination in unknown environments.

6.1 Contributions

In segmenting the control requirements in a hierarchical structure and isolating information and command flow, it was shown that individual aspects of mobile robots may operate independently of one another, only coming together in very limited fashion. As was presented, the platform level of control only received a desired heading and speed from the navigation level and therefore was only responsible for articulating the specific platform to achieve that heading and speed, based on sensors relevant only to that task. The navigation level received only a desired location within the environment and therefore needed only to determine a safe and efficient method, independent of robot articulation capabilities, to reach that destination through sensors relevant only to that specific task. Finally, the robot purpose level was required only to be concerned with utilizing categorically specific sensors in order to achieve the system level goal, in this case exploration and coordination for information gathering.

The success of this control requirement segmentation was proven through the application of independent analysis of control techniques for the two higher levels while holding the others constant. For example, while holding the platform level constant with model-based control, adaptive navigation techniques were implemented and evaluated as they compared to a pre-engineered reactive navigation technique. The neuro-evolutionary unsupervised learning technique proved to not only perform statistically similar to reactive navigation in contrived situations, but outperform reactive navigation in dense unknown environments even in the face

of significant sensor and actuator signal noise. The algorithm also was shown to improve upon current knowledge when trained offline based on known functional algorithms.

The reinforcement learning versions of supervised learning implemented were reported to function is this domain, but the learning rate was discovered to be intractably slow when implemented in the complex simulation domain utilized in this work. This was a result of the continuous nature of the domain. There are several modifications that can be made to the techniques presented to assist in overcoming this restriction, including domain specific discretization of the state and action spaces as well as online objective functions. The former involves choosing functional ranges of the state and action spaces that still represent the spaces but reduce the amount of information the algorithm is required to learn. The latter involves generating more specific reward structures leading to a global maximum through the episode. For example, instead of giving zero reward until the episode is complete, the action-value function can be updated repeatedly through the episode based on time and distance from obstacles. These modifications may be implemented in the future to continue adaptive navigation analysis.

Finally, the platform and navigation levels were held constant at model-based and probabilistic control respectively while the robot purpose level was examined for effective coordination behavior. A similar neuro-evolutionary technique to the navigation analysis was employed, with the neural network function approximator instead performing the task of goal selection as it pertained to coordinating with other robots in the domain. The goal selection process required that the robots

work together to examine as many points of interest within the environment as possible in a short amount of time. Specific to this analysis was the investigation of three objective functions providing different qualitative properties to the system and robot. The difference objective was shown to significantly outperform the local and system objectives when the environment is rich in information and in dynamically changing environments.

The results presented in this thesis prove the following:

1. Under strong information restrictions and command isolation, segmentation of control requirements is a beneficial technique for resource limited robot design, as well as operation within a robot coordination domain.

2. Adaptive navigation techniques (e.g., evolutionary search over artificial neural networks), are a valid and beneficial method over pre-engineered reactive navigation, for both navigation and goal selection applications when applied to a robot coordination domain.

3. In noisy environments, adaptive navigation is able to overcome deficiencies in sensing and actuation to provide behaviors robust to realistically stochastic robotic platforms.

4. Utilizing a difference objective function combining the qualities of system objective and local objectives is a valid and beneficial training structure for learning robots within a team of coordinating robots, specific to the application of unknown environment exploration, while the restrictions of realistic robot motion and task isolation are in place.

5. In dynamic environments, the utilization of difference objectives for training provide increasing benefits over system and local objectives through the balance of system objective alignment and locally computable information.

In addition to the above, the results prove that adaptive techniques are successful in general for isolated control tasks under noisy, continuous domains.

6.2 Future Work

There are many avenues for further exploration in this domain. These include but are not limited to continuing investigations into the interaction between control levels as they pertain to multiple learning algorithms working together, behaviors resulting from further restrictions on sensor and actuator signals, and a detailed examination of the effects of obstacle avoidance on coordination behavior.

In addition, many aspects of the coordination technology may be investigated to enhance information gain. A sample of aspects for further investigation follow:

Communication: Among individuals within a team such as that presented in this thesis, communication may be added to examine the potential benefit to the amount of information gathered and the coordination behavior.

Teams of Robots: Multiple, smaller teams may be organized to coordinate with one-another. For example, instead of one robot visiting a goal, the system objective may require that three visit for the investigation to be valid, therefore three robots must work together within the larger team to maximize the

information gathered.

Heterogeneous Team(s): Robots within a team, or teams, may be constructed such that individuals have differing levels of sensing capabilities. A single team would provide singularly differing benefits to each sensor type, or as above, it may be configured that three different types of sensors must perform an investigation.

Coordination for robotic exploration is an intricate and challenging control problem. As this thesis suggests, there are successful methods for addressing the challenges as are there many areas for further research.

Bibliography

[1] Y. Abu-Mostafa. Learning from hints in neural networks. *Journal of Complexity*, 6(2):192–198, 1990.

[2] A. Agogino and K. Tumer. Efficient evaluation functions for multi-rover systems. In *The Genetic and Evolutionary Computation Conference*, pages 1–12, Seatle, WA, June 2004.

[3] A. Agogino and K. Tumer. Distributed evaluation functions for fault tolerant multi rover systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Seattle, WA, July 2006.

[4] A. Agogino, K. Tumer, and R. Miikulainen. Efficient credit assignment through evaluation function decomposition. In *The Genetic and Evolutionary Computation Conference*, Washington, DC, June 2005.

[5] Mazda Ahmadi and Peter Stone. A multi-robot system for continuous area sweeping tasks. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1724–1729, May 2006.

[6] J. Bagnell, S. Kakade, A. Ng, and J. Schneider. Policy search by dynamic programming. In *Neural Information Processing Systems*, volume 16. MIT Press, December 2003.

[7] Tucker Balch and Ronald C. Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.

[8] Michael H. Bowling, Rune M. Jensen, and Manuela M. Veloso. *Planning in Intelligent Systems: Aspects, Motivations and Methods*. John Wiley and Sons, Inc., 2005.

[9] J. Bridle. Probabilistic interpretation of feedforward classification ... *Neurocomputing: Algorithms, Architectures, and Applications*, pages 227–236, 1990.

[10] B. Clement and E. Durfee. Theory for coordinating concurrent hierarchical planning agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 495–502, 1999.

[11] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems*. Prentice-Hall, Inc., 2001.

[12] M. Dorigo and T. Stützle. *Ant Colony Optimization*. Bradford Books, 2004.

[13] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. John Wiley and Sons, 2006.

[14] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *Proc. of Conf. on Simulation of Adaptive Behavior*, 1994.

[15] Jakob Fredslund and Maja J. Mataric. A general, local algorithm for robot formations. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, 2002.

[16] Dani Goldberg and Maja J. Mataric. Maximizing reward in a non-stationary mobile robot environment. *Autonomous Agents and Multi-Agent Systems*, 3(6):281–316, 2003.

[17] F. Gomez and R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Chicago, Illinois, 2003.

[18] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.

[19] Andrew Howard, Maja J. Mataric, and Gaurav S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 13(2):113–126, 2002.

[20] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[21] M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 740–747, 1999.

[22] Sven Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *Transactions on Robotics*, 21(3):354–363, 2005.

[23] Sven Koenig, Craig Tovey, and Yuri Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*, 147(1-2):253–279, 2003.

[24] Michail G. Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton Kleywegt, Sven Koenig, Craig Tovey, Adam Meyerson, , and Sonal Jain. Auction-based multi-robot routing. In *Proceedings of the Conference on Robotics: Science and Systems (ROBOTICS)*, 2005.

[25] A Martinoli, A. J. Ijspeert, and F. Mondala. Understanding collective aggregation mechanisms: From probabilistic modelling to experiments with real robots. *Robotics and Autonomous Systems*, 29:51–63, 1999.

[26] M. J. Mataric. Coordination and learning in multi-robot systems. In *IEEE Intelligent Systems*, pages 6–8, March 1998.

[27] Maja J. Mataric. Learning in behavior-based multi-robot systems: Policies, models, and other agents. *Cognitive Systems Research*, 2(1):81–93, 2001.

[28] Colin P. McMillen, Paul E. Rybski, and Manuela M. Veloso. Levels of multi-robot coordination for dynamic environments. *Multi-Robot Systems: From Swarms to Intelligent Automata*, 3, 2005.

[29] Justin Melvin, Pinar Keskinocak, Sven Koenig, Craig Tovey, and Banu Yuksel Ozkaya. Multi-robot routing with rewards and disjoint time windows. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[30] David Moriarty and Risto Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5:373–399, 2002.

[31] A. Mudgal, C. Tovey, S. Greenberg, and S. Koenig. Bounds on the travel cost of a mars rover prototype search heuristic. *SIAM Journal on Discrete Mathematics*, 19(2):431–437, 2005.

[32] Apurva Mudgal, Craig Tovey, and Sven Koenig. Analysis of greedy robot-navigation methods. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (AMAI)*, 2004.

[33] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

[34] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In *Proc. of Artificial Life IV*, pages 190–197, 1994.

[35] D. Parkes and S. Singh. An MDP-based approach to online mechanism design. In *NIPS 16*, pages 791–798, 2004.

[36] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc., Upper Saddle River, New Jersey 07458, 2003,1995.

[37] S. Salzberg. Distance metrics for instance-based learning. *Lecture Notes in Computer Science*, 542:399–408, 1990.

[38] P. Scerri, J.A. Giampapa, and K. Sycara. Techniques and directions for building very large agent teams. In *2005 International Conference on Integration of Knowledge Intensive Multi-Agent Systems, KIMAS*, 2005.

[39] R. Siegwart and I. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.

[40] Adriaan Smit, Donald Heer, Roger Traylor, and Terri S. Fiez. A custom microcontroller system used as a platform for learning^{TM}in ece. In *ASEE Annual Conference Proceedings*, pages 2733–2740, 2004.

[41] K. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, CA, 2002.

[42] S.Thrun and A.Schwart. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 255–263, Hillsdale, NJ, 1993.

[43] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 2005.

[44] Daniel Stronger and Peter Stone. Towards autonomous sensor and actuator model induction on a mobile robot. *Connection Science*, 18(2):97–119, 2006. Special Issue on Developmental Robotics.

[45] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[46] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, MA, 2005.

[47] S. Thrun, D. Fox, and W. Burgard. Monte carlo localization with mixture proposal distribution. In *AAAI National Conference on Artificial Intelligence*, pages 859–865. AAAI Press / The MIT Press, 2000.

[48] S. Thrun and G. Sukhatme. *Robotics: Science and Systems I*. MIT Press, 2005.

[49] K. Tumer and A. Agogino. Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In *The Genetic and Evolutionary Computation Conference*, Washington, DC, June 2005.

[50] K. Tumer and D. Wolpert. A survey of collectives. In *Collectives and the Design of Complex Systems*, pages 1,42. Springer, 2004.

[51] J. Wang, M. Lewis, and P. Scerri. Cooperating robots for search and rescue. In *Agent Technology for Disaster Management Workshop at AAMAS'06*, 2006.

[52] Shimon Whiteson, Peter Stone, Kenneth O. Stanley, Risto Miikkulainen, and Nate Kohl. Automatic feature selection via neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, June 2005.

[53] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.

[54] X. Zheng, S. Jain, S. Koenig, and D. Kempe. Forest-based multirobot coverage. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2005.