COORDINATION IN

MULTI-AGENT ROBOTIC SYSTEMS

By

Omar Ahmad

Submitted to the Department of Computer Engineering
in Partial Fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Thesis Advisor

Dr. Mahmood Anwar Khan

College of Electrical and Mechanical Engineering

National University of Sciences and Technology, Pakistan

2007

In the name of Allah, the most Merciful and the most Beneficent

TO MY PARENTS AND TEACHERS

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Mahmood Anwar Khan, whose sheer interest and helping hand inculcated in me, the incentive and the dedication for this project.

I would specially like to thank Dr. Javed Iqbal, my ex-supervisor (he went to Australia for his post-doc), for inspiring me and guiding me along the path and helping me out in with any problem I faced during my research. Even from so far away and busy with his own research he found time to correct my research paper for presentation in an IEEE conference. I am truly indebted to him.

I would like to thank my friend Ahmad Salman and Ali Hassan for giving me valuable advice regarding my research and specially report writing.

Finally I am thankful to my parents and my wife for their support, prayers and patience during the research work.

Abstract

During last few decades, major research efforts were focused on improving the performance of an individual mobile robot by using advanced sensors, actuators, and intelligent control algorithms. This was mainly driven by the need to perform increasingly complex tasks required by real world applications. As a result, individual mobile robot has become very sophisticated.

The current trend in robotic research, both from engineering and behavioral viewpoints, has been to move away from the design and deployment of few, rather complex, usually expensive, application-specific robots. In fact, in the last decade the interest has shifted towards the design and use of a large number of "generic" robots which are very simple, with very limited capabilities and, thus, relatively inexpensive, but capable, together, of performing rather complex tasks. In a system consisting of a set of totally distributed agents the goal is generally to exploit the multiplicity of the elements in the system so that the execution of a certain predetermined task occurs in a coordinated and distributed way.

The advantages of this approach are clear and many, including: reduced costs (due to simpler engineering and construction costs, faster computation, development and deployment time, etc); ease of system expandability (just add a few more robots) which in turns allows for incremental and on-demand deployment (use only as few robots as you need and when you need them); simple and affordable fault-tolerance capabilities (replace just the faulty robots); re-usability of the robots in different applications (reprogram the system to perform a different task). Moreover, tasks that could not be performed at all by a single agent become manageable when many simple units are used instead.

I intend to design and implement a distributed multi-robot system using some multi-robot simulation tool. Coordinating robots in a dynamic environment is a difficult task. They must be able to carry out their contributions to the overall goal of the system efficiently and effectively while not impeding each other. The focus of multi-robot coordination should therefore be twofold: each robot should consider the objectives of the

team while maintaining its own functional integrity. As such, the team plan should exist at a level where it provides strategies for each robot to contribute to the team's success. Each robot must consider the team strategy and execute it as best it can without compromising its ability to maintain functional operation.

There are two ways of making the robots coordinate. First is to use explicit communication between the robots so that they may easily coordinate. But in actual scenarios this explicit communication is usually not possible or feasible (e.g. in a hostile environment). The other choice is to make the robots coordinate by observing their environment or the behaviors of the other robots. This latter approach is a new concept in recent research. I wish to study and implement this approach during the course of my thesis work.

Chapter 1

Introduction

Chapter 1: Introduction

1.1 General

The field of distributed and cooperative robotics has its origins in the late 1980s, when several researchers began investigating issues in multiple mobile robot systems [5,7]. Prior to this time, research had concentrated on either single-robot systems or distributed problem-solving systems that did not involve robotic components [3]. Since then, the field has grown dramatically, with a much wider variety of topics addressed.

An autonomous agent is a computational system that acquires and analyzes sensory data or external stimulus and executes behaviors that produce effects in the environment. It decides for itself how to relate sensory data to its behaviors in its efforts to attain certain goals. Such a system is able to deal with unpredictable problems, dynamically changing situations, poorly modeled environments, or conflicting constraints. The motivation behind the research and development in multi-agent robotic systems comes from the fact that the decentralized multi-robot approach has a number of advantages over traditional single complex robotic systems approaches.

Distributed robots can readily exhibit the characteristics of structural flexibility, reliability through redundancy, simple hardware, adaptability, re-configurability, and maintainability. The robots can interact with their local environments in the course of collective problem-solving. Responding to different local constraints received from their task environments, they may select and exhibit different behavior patterns, such as avoidance, following, aggregation, dispersion, homing, and wandering. These behaviors are precisely controlled through an array of parameters (such as motion direction, timing, lifespan, age, etc.), which may be carefully predefined or dynamically acquired by the robots based on certain computational mechanisms.

In order to successfully develop multi-agent robotic systems, the key methodological issues must be carefully examined. At the same time, the underlying computational models and techniques for multi-agent systems engineering must be thoroughly understood.

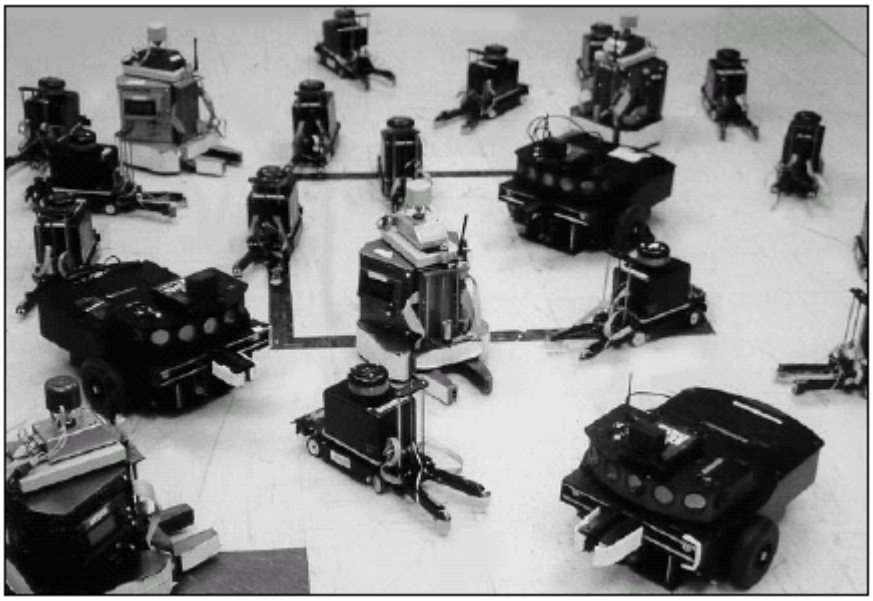


Figure1.1. The Nerd Herd: A multi-robot test bed developed at the MIT

1.2 Tasks of the Project

The sequence of the project tasks is going to be in the following way.

- Review the current research in Multi-Robot coordination.
- Review coordination techniques which minimize communication.
- The design and simulation of a communication-less coordination scheme.
- Test this new scheme for simple and complex group-level tasks.
- Verification and validation of this new scheme by comparing results.
- Identifying areas where future research can be directed to develop on this technique.

1.3 Overview of Multi-Robot Systems

In this section I discuss the advantages and additional issues involved in the control of multi-robot systems (MRS) as compared to the single-robot systems (SRS) discussed in the previous section. An MRS is a system composed of multiple, interacting robots. The study of MRS has received increased attention in recent years. This is not surprising, as continually improving robustness, availability, and cost-effectiveness of robotics technology has made the deployment of MRS consisting of increasingly larger numbers of robots possible. With the growing interest in MRS comes the expectation that, at least in some important respects, multiple robots will be superior to a single robot in achieving a given task. In this section I outline the benefits of a MRS over a SRS and introduce issues involved in MRS control and how they are similar and different to those of SRS control.

1.3.1 Advantages of Multi-Robot Systems

Potential advantages of MRS over SRS include a reduction in total system cost by utilizing multiple simple and cheap robots as opposed to a single complex and expensive robot. Also, multiple robots can increase system flexibility and robustness by taking advantage of inherent parallelism and redundancy. Furthermore, the inherent complexity of some task environments may require the use of multiple robots, as the necessary capabilities or resource requirements are too substantial to be met by a single robot. To summarize following are the main advantages of using multiple robots;

- a larger range of task domains
- greater efficiency
- improved system performance
- fault tolerance
- robustness
- lower economic cost
- ease of development
- distributed sensing and action
- inherent parallelism

- insight into social and life sciences

1.3.2 Challenges in Design of Multi-Robot Systems

The utilization of MRS poses potential disadvantages and additional challenges that must be addressed if MRS are to present a viable and effective alternative to SRS. A poorly designed MRS, with individual robots working toward opposing goals, can be less effective than a carefully designed SRS. A paramount challenge in the design of effective MRS is managing the complexity introduced by multiple, interacting robots[12]. As such, in most cases just taking a suitable SRS solution and scaling it up to multiple robots is not adequate.

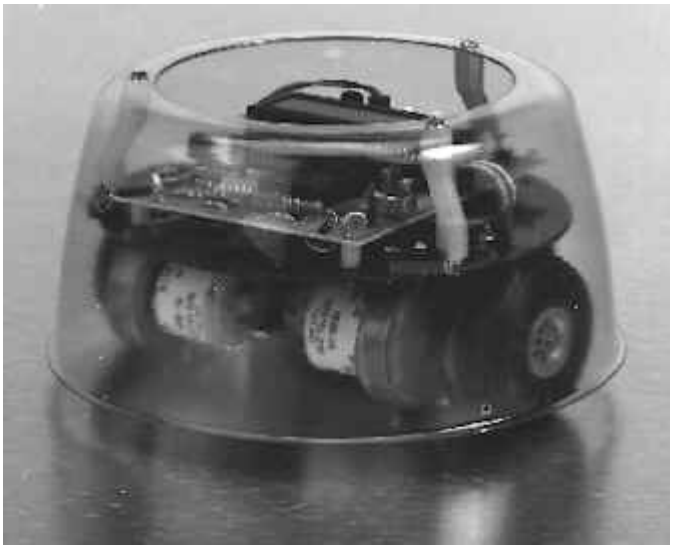


Figure1.2. Rug-Warrior: A robot designed for Multi-Robot Systems research.

1.4 Coordination in Multi-Robot Systems

In order to maximize the effectiveness of a MRS, the robots' actions must be spatiotemporally coordinated and directed towards the achievement of a given system-level task or goal. Just having robots interact is not sufficient in itself to produce interesting or practical system-level coordinated behavior. The design of MRS can be quite challenging because unexpected system-level behaviors may emerge due to unanticipated ramifications of the robots' local interactions. In order for the interacting robots to produce coherent task-directed behavior, there must be some overarching

coordination mechanism that spatio-temporally organizes the interactions in a manner appropriate for the task. The design of such coordination mechanisms can be difficult; nonetheless, many elegant handcrafted distributed MRS have been demonstrated, both in simulation and on physical robots [19,20,21]. The methods by which these systems have achieved task-directed coordination are diverse and the possibilities are seemingly limited only by the ingenuity of the designer. From a few robots performing a manipulation task [22,23], to tens of robots exploring a large indoor area [24,25], to potentially thousands of ecosystem monitoring nano-robots [26,27], as the number of robots in the system increases, so does the necessity and importance of coordination. The next section examines mechanisms by which system-level coordination can be successfully achieved in a MRS.

1.4.1 Classification of group level tasks

The main advantage of using multiple robots is to perform tasks more efficiently than a single robot, or to perform tasks that a single robot is incapable of doing. Imagine a single robot wandering in a mine field looking for mines and disabling them. It may take it hours yet there will be no guarantee that the field has been cleared. In contrast imagine the same field crowded with small robots all looking for mines and avoiding other robots in due course. This approach should lead to reduced time and a performance increase of manifolds. Thus each robot will be running two separate sets of tasks. One is the group of local tasks that a robot needs to avoid obstacles and other robots, and the second is the group level tasks of disabling the mines in an efficient manner.

Now group level tasks can be further divided on the basis of task complexity. This is discussed in the next section.

1.4.2 Simple Tasks

Simple tasks in the context of multi-robot systems are those tasks that can be completed by using comparatively less intelligent robots. Even robots that can neither remember nor anticipate can complete these tasks effectively. Foraging is an example of such a task. In nature we see ants foraging very efficiently yet they don't have a brain intelligent enough to remember any thing of the past[24]. Whatever is hard wired in their

brains remains so for the duration of their life yet still they show intelligent behavior unmatchable be the best of robots. Thus I propose that simple tasks can be completed using very limited capabilities in the robots.

1.4.3 Complex Tasks

Complex tasks are those group-level tasks which require greater intelligence, computational power and memory in the participating robots. Examples of these tasks are robots playing football (Figure 1.3), a group of robots collectively carrying a heavy object to some target location etc. These tasks are more demanding as far as designing constraints are concerned.

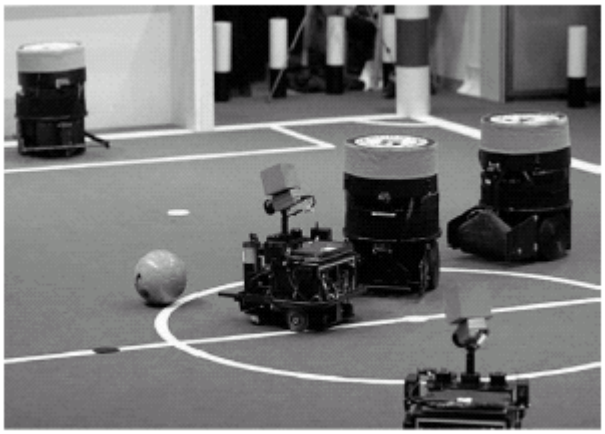


Figure1.3. Robot football: An example of complex multi-robot task

1.5 Interaction in Multi-Robot Systems

Given the importance of coordination in a MRS, I now address the issue of how to organize the robots' local interactions in a coherent manner in order to achieve system level coordination. There are many mechanisms by which the interactions can be organized. I classify them into three broad and often overlapping classes: interaction through the environment, interaction through sensing, and interaction through communication. These classes are not mutually exclusive because MRS can, and often do, simultaneously utilize mechanisms from any or all of these classes to achieve system level coordinated behavior. In the following sections I describe each of these interaction

classes in detail. Through the discussion of empirical case studies I demonstrate how each type of interaction can be used to achieve system-level coordination in a MRS.

1.5.1 Interaction through communication

The first mechanism for interaction among robots is through explicit communication. Such robot-directed communication can be used to request information or action from other robots or to respond to received requests.

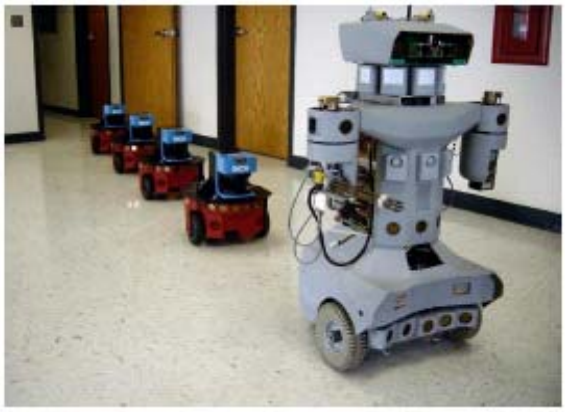


Figure1.4. A 'Leader' robot coordinates by communicating with the follower robots

Communication in physical robotics is not free or reliable and can be constrained by limited bandwidth and range, and unpredictable interference [7,8]. When utilizing it, one must consider how and toward what end it is used. In some domains, such as the Internet, communication is reliable and of unlimited range; however, in physical robot systems, communication range and reliability are important factors in system design [2,36]. There are many types of communication. Communication could be direct from one robot to another, direct from one robot to a class of other robots, or broadcast from one robot to all others. Furthermore, the communication protocol can range from simple protocol-less schemes to a complex negotiation-based and communication-intensive schemes The information encoded in a communication may be state information contained by the communicating robot, a command to one or more other robots, or a request for additional information from other robots, etc. Communications may be task-related rather than robot-directed, in which case it is made available to all (or a subset) of the robots in the MRS. A common task-related communication scheme is

publish/subscribe messaging. In publish/subscribe messaging, subscribing robots request to receive certain categories of messages, and publishing robots supply messages to all appropriate subscribers. In the next subsection, I describe a case study of the effective use of interaction through communication.

1.5.2 Interaction through sensing

The second mechanism for interaction among robots is through sensing, described in [9], interaction through sensing 'refers to local interactions that occur between robots as a result of sensing one another, but without explicit communication.' As with interaction through the environment, interaction through sensing is also indirect as there is no explicit communication between robots; however, it requires each robot to be able to distinguish other robots from miscellaneous objects in the environment. In some instances, each robot may be required to uniquely identify all other robots, or classes of other robots. In other instances, it may only be necessary to simply distinguish robots from other objects in the environment. Interaction through sensing can be used by a robot to model the behavior of other robots or to determine what another robot is doing in order to make decisions and respond appropriately. For example, flocking birds use sensing to monitor the actions of other birds in their vicinity to make local corrections to their own motion. It has been shown that effective flocking results from quite simple local rules followed by each bird responding to the direction and speed of the local neighbors [32].

In the follow subsection I describe a case study in a formation marching domain in which interaction through sensing is used to achieve coordinated group behavior. Other domains in which interaction through sensing has been utilized in MRS include flocking [33], in which each robot adjusts it motions according to the motions of locally observed robots. Through this process, the robots can be made to move as a coherent flock through an obstacle-laden and dynamic environment. Interaction through sensing has also been demonstrated in an adaptive division of labor domain [34]. In that domain, each robot dynamically changes the task it is executing based on the observed actions of other robots and the observed availability of tasks in the environment. Through this process, the group of robots coherently divides the labor of the robots appropriately across a set of available tasks.

1.5.3 Interaction via Stigmergy

The third mechanism for interaction is through the robots' shared environment. This form of interaction is indirect in that it consists of no explicit communication or physical interaction between robots. Instead, the environment itself is used as a medium of indirect communication. This is a powerful approach that can be utilized by very simple robots with no capability for complex reasoning or direct communication. An example of interaction through the environment is demonstrated in stigmergy, a form of interaction employed by a variety of insect societies. Originally introduced in the biological sciences to explain some aspects of social insect nest-building behavior, stigmergy is defined as the process by which the coordination of tasks and the regulation of construction do not depend directly on the workers, but on the constructions themselves [28]. This concept was first used to describe the nest-building behavior of termites and ants [29]. It was shown that coordination of building activity in a termite colony was not inherent in the termites themselves. Instead, the coordination mechanisms were found to be regulated by the task environment, in that case the growing nest structure. A location on the growing nest stimulates a termite's building behavior, thereby transforming the local nest structure, which in turn stimulates additional building behavior of the same or another termite. Through the careful design of robot sensing, actuation, and control features, it is possible to utilize the concept of stigmergy in task-directed MRS.

This powerful mechanism of coordination is attractive as it typically requires minimal capabilities of the individual robots. The robots do not require direct communication, unique recognition of other robots or even distinguishing other robots from miscellaneous objects in the environment, or the performance of computationally intensive reasoning or planning. Stigmergy, and more generally interaction through the environment, has been successfully demonstrated as a mechanism to coordinate robot actions in a number of MRS. It has been demonstrated in an object manipulation domain [30] in which a large box was transported to a goal location through the coordinated pushing actions of a group of robots. There was no globally agreed upon plan as to how or over what trajectory the box should be moved; however, each robot could indirectly

sense the pushing actions of other robots through the motions of the box itself. Through simple rules, each robot decided whether to push the box or move to another location based on the motions of the box itself. As a large enough number of robots pushed in compatible directions, the box moved, which in turn encouraged other robots to push in the same direction. Other examples of the use of stigmergy in MRS include distributed construction in which a given structure was built in a specified construction sequence [31]. The individual robots were not capable of explicit communication and executed simple rule-based controllers in which local sensory information was directly mapped to construction actions. The construction actions of one robot altered the environment, and therefore the subsequent sensory information available for it and all other robots. This new sensory information then activated future construction actions. In the following subsection I discuss in detail how the concept of stigmergy was utilized in a MRS object clustering task domain [28].

Chapter 2

Coordination without Communication

Chapter 2: Coordination without Communication

2.1 General

Coordination between multiple agents is an essential requirement for successful completion of many large-scale tasks. Many tasks require more resources than a single robot can possibly provide. By enabling many simple robots to cooperate together on a larger task, it becomes possible to solve a problem that would be infeasible using a single large-scale machine.

A major concern, however, in many attempts to coordinate multiple-agent behavior is the perceived requirement that a hierarchy must be present to effectively carry out a task. A master/slave relationship, although often allowing for a more efficient conduct of particular tasks, introduces brittleness into the system. This brittleness is characterized by dependence on a single source (or a few) for guidance of the other cooperating agents. Three major drawbacks can be seen with this approach. The first involves the communication bottleneck when a master is trying to coordinate the behavior of a swarm of slave agents. This can be a severely limiting factor for large collection of simple robots.

2.2 Is communication necessary for Coordination?

Now a question arises that is communication necessary at all for carrying out group level tasks? There has been much debate on this issue ever since. We have already seen the advantages of a communication-less system there for in the following section we seek answers to the above said questions that whether we require communication, if so to what extent and if not are there exceptions?

2.3 The Hensel twins

The story of the Hensel twins [16] sowed the seeds of doubt. Abigail and Brittany, kindergartners, share a body. Their separate heads rise from a single pair of shoulders topping a body with the usual number of arms and legs. Inside, things are more complex, with two hearts pumping a single circulatory system, three lungs, two stomachs, and separate spines jointed at the pelvis. Their nervous systems are disjoint. A touch on the

right side is felt only by Abby. Each controls one arm and one leg. And therein lies the rub. At fifteen months, they learned to walk. Now they swim, bike, and tie shoe laces, all requiring considerable coordination of actions.

This coordination is apparently achieved without communication. "But," I've been told, "they do communicate through the environment. Each is aware of the other's actions via vision, proprioception, etc. That's communication." Is it? True, all communications occur when one agent acts on the environment and the other senses the results of that action. But would all such acting and sensing comprise communications? I think not. It's far fetched to call my following the tracks of a bobcat in the snow communication between the bobcat and me. Communication, in the sense of the word I intend, requires the sending and receiving of signals. Is the bobcat signaling when he leaves tracks in the snow? I think not. This situation is quite analogous to me following a river downstream while canoeing

2.4 Lybrinth game

Is such coordination without communication possible only under the most unusual circumstances of the Hensel twins? Stan Franklin [4] reports and I quote, “Let me give you another example that should have alerted me to the possibility of coordination without communication years ago. My older children during their teenage years were quite fond of a maze game called Labyrinthspiel. Picture a roughly one foot square wooden box some four inches tall. Mounted in the upper interior of the box were two nested trays. The inner tray was a maze containing some sixty holes strategically placed as pitfalls for the unwary traveler. The traveler was a small steel ball, just able to fall through one of the holes into the bottom of the box. The outer tray, mostly rim, was mounted at its center on a metal rod running north south. A knob, attached to the rod, could be rotated, tilting both trays in the east west direction. The inner tray was independently mounted on a second metal rod perpendicular to the first. Rotating its knob tilted only the inner tray in the north south direction. With the ball in the starting position, the player attempted to guide the ball through the maze, avoiding all the holes. This required well coordinated manipulation of the two knobs using the principle that balls roll down hill. It wasn't easy.

Over time, two of the children, Phil and Mimi, completely mastered Labyrinthspiel, being able to reliably guide the ball back and forth through the maze, missing all the pitfalls. But such mastery led to boredom. So, they invented two-person Labyrinthspiel, Mimi controlling one knob and Phil the other. To my amazement, they were soon able to coordinate their movements so as to successfully negotiate the entire maze."

It seems to me that this sort of coordination is impossible to achieve in real time via any sort of signaling. It can only result from each agent frequently sampling the environment, in this case watching the ball and the attitude of the maze, and reacting to it with its goal in mind. Reacting, in this case, should be taken as reacting not only to the ball's location, but to its velocity (including direction)1 as well. Coordination emerges incidentally. I also suspect that there are no internal signals from the right hand's controller to the left's when a single human is playing. Probably sampling and reacting suffices.

2.5 Feasibility of coordination without communication

With all the information I have just gathered it seems quite plausible that coordination can be achieved without the involvement of any explicit communication. Many researchers have come across this reality that removing explicit communication in multi-robot scenario results is much faster task completion but at the cost of reduced reliability.

Mataric [18] compares the effects of explicit communication, goal communication and state communication for foraging task. The results suggest that through state communication results comparable to other approaches can be achieved.

Now question arises that under what conditions is coordination without communication is feasible. I suggest that under following circumstances coordination without communication is feasible.

- In hostile environments where eavesdropping is a possibility.
- In noisy environment where reliability of communication cannot be guaranteed.

- In hi speed applications where communication overhead will slow down the execution speed.
- In cases where human life is not at stake and inaccuracy can be tolerated.
- In cases where the number of robots in the swarm is great and the cost of communication is very high.

2.6 Requirements of the task

To achieve coordination without communication robots must possess some qualities that would enable them to interact with each other in meaningful way and work together to achieve some group-level goal. In the following section I explore these necessary qualities in detail

2.6.1 Intelligence

Intelligence is a quality of mind whether human, animal or computer There have been meny definitions of intelligence throughout the AI history but a definition which is widely accepted and signed by 52 AI researchers given in 'wikipedia' is:

a very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. It is not merely book learning, a narrow academic skill, or test-taking smarts. Rather, it reflects a broader and deeper capability for comprehending our surroundings—"catching on", "making sense" of things, or "figuring out" what to do.

In our case the robots must be able to exhibit intelligent behavior and carry out tasks without failing to be labeled as intelligent

2.6.2 Memory

A simple definition of memory is "an organism's ability to store, retain, and subsequently recall information". In the case of robots a robot must keep track of its whereabouts and recall certain experiences (good or bad) and be able to use them promptly when needed. This would require some adaptability, some physical memory device and some clever algorithms to achieve this task.

2.6.3 Anticipation

As mentioned above, an anticipatory robot should assess the current situation, predict the future consequence of the situation, and execute an action to have desired outcome based on the assessment and the prediction. The concept of an anticipatory robot may be best represented by Rosen's diagram (Figure 1). Rosen [12] proposed the notion of anticipatory systems in order to analyze how adaptive living organisms work. The labels S, M, and E in the figure stand for object system, model, and effectors, respectively. More specifically, S represents some dynamical system that interacts with the environment. For example, the system could be a microorganism, animal, or even an economy of some country. M is a model of S. Given a current state of S and an environment, M foretells what state S is likely to reach in the future. E is the effector that can interact with S or the environment in order to influence the future state of S. According to Rosen [12], the function of the anticipatory system is to: (a) Do nothing if M expects that S is likely to stay in a "desirable" state; or (b) activate E to correct the "trajectory" of S if M forecasts that an unwanted outcome is imminent. One of the important properties of the anticipatory system is that, unlike a reactive system that executes actions simply as a response to a current state (stimuli), the system reacts to a state that is expected to happen in the future.

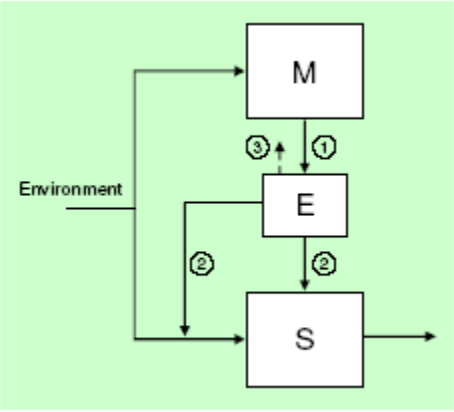


Figure2.3. Rosen's Diagram: An anticipatory system

Chapter 3

Intelligence in Robots

Chapter 3: Intelligence in Robots

3.1 General

The science of making machines act intelligently is usually referred to as Artificial Intelligence, or AI for short. Artificial intelligence has no commonly accepted definitions. One of the first text books on AI defined it as “the study of ideas that enable computers to be intelligent,” which seemed to beg the question. A later textbook was more specific, “AI is the attempt to get the computer to do things that, for now, people are better at.”[19]. This definition implies that certain AI problems can be solved, which is generally not true because there can always be a better (more intelligent) ways of doing things.

3.2 Classical AI approach

The term Artificial Intelligence (AI) was first used by John McCarthy who used it to mean "the science and engineering of making intelligent machines".[1] It can also refer to intelligence as exhibited by an artificial (man-made, non-natural, manufactured) entity. While AI is the generally accepted term others, including both Computational Intelligence and Synthetic Intelligence have been proposed as potentially being "more accurate."[2] The terms strong and weak AI can be used to narrow the definition for classifying such systems. AI is studied in overlapping fields of computer science, robotics, psychology, philosophy, neuroscience, and engineering, dealing with intelligent behavior, learning, and adaptation and usually developed using customized machines or computers.

Research in AI is concerned with producing machines to automate tasks requiring intelligent behavior. Examples include control, planning and scheduling, the ability to answer diagnostic and consumer questions, handwriting, natural language, speech, and facial recognition. As such, the study of AI has also become an engineering discipline, focused on providing solutions to real life problems, knowledge mining, software applications, strategy games like computer chess and other video games. One of the biggest difficulties with AI is that of comprehension. Many devices have been created

that can do amazing things, but critics of AI claim that no actual comprehension by the AI machine has taken place.

The symbol system hypothesis, [30], states that intelligence operates on a system of symbols. The implicit idea is that perception and motor interfaces are sets of symbols on which the central intelligence system operates. Thus, the central system, or reasoning engine, operates in a domain independent way on the symbols. Their meanings are unimportant to the reasoner, but the coherence of the complete process emerges when an observer of the system knows the groundings of the symbols within his or her own experience. Somewhat more implicitly in the work that the symbol system hypothesis has inspired, the symbols represent entities in the world. They may be individual objects, properties, concepts, desires, emotions, nations, colors, libraries, or molecules, but they are necessarily named entities. There are a number of effects which result from this commitment. Recall first, however, that an intelligent system, apart from those which are experiments in the laboratory, will be embedded in the world in some form or another.

3.2.1 The Interface between Perception and Symbols

The central intelligence system deals in symbols. It must be fed symbols by, the perception system. But what is the correct symbolic description of the world around the intelligence system? Surely that description must be task dependent. The default assumption has been that the perception system delivers a description of the world in terms of typed, named individuals and their relationships. For instance in the classic monkeys and bananas problem, the world, description is in terms of boxes, bananas, and above ness. But for another task (e.g., deciding whether the bananas are rotten) quite a different representation might be important. Psychophysical evidence [32] certainly points to perception being an active and task dependent operation. The effect of the symbol system hypothesis has been to encourage vision researchers to quest after the goal of a general purpose vision system which delivers complete descriptions of the world in a symbolic form (e.g. [5]). Only recently has there been a movement towards active vision [4] which is much more task dependent, or task driven [1].

3.2.2 Inadequacy of Simple Symbols

Symbol systems in their purest forms assume a knowable objective truth. It is only with much complexity that modal logics, or non-monotonic logics, can be built which better enable a system to have, beliefs gleaned from partial views of a chaotic world. As these enhancements are made, the realization of computations based on these formal systems becomes more and more biologically implausible. But once the commitment to symbol systems has been made it is imperative to push on through more and more complex and cumbersome systems in pursuit of objectivity. This same pursuit leads to the well known frame problem (e.g., [27]), where it is impossible to assume anything that is not explicitly stated. Technical deviations around this problem have been suggested but they are by no means without their own problems.

3.2.3 Symbol Systems Rely on Emergent Properties

In general the reasoning process becomes trivial in an NP-complete space (e.g., There have been large efforts to overcome these problems by choosing simple arithmetically computed evaluation functions or polynomials to guide the search. Charmingly, it has been hoped that intelligence will somehow emerge from these simple numeric computations carried out in the sea of symbols. [28] was one of the earliest examples of this hope, which later turned out to be only partially correct (his learned polynomials later turned out to be dominated by piece count), but in fact almost all instances of search in classical AI have relied on such judiciously chosen polynomials to keep the search space manageable.

3.3 Modern Approaches in AI

Nouvelle AI is based on the physical grounding hypothesis. This hypothesis states that to build a system that is intelligent it is necessary to have its representations grounded in the physical world. Our experience with this approach is that once this commitment is made, the need for traditional symbolic representations soon fades entirely. The key observation is that the world is its own best model. It is always exactly up to date. It always contains every detail there is to be known. The trick is to sense it appropriately and often enough. To build a system based on the physical grounding

hypothesis it is necessary to connect it to the world via a set of sensors and actuators. Typed input and output are no longer of interest. They are not physically grounded. Accepting the physical grounding hypothesis as a basis for research entails building systems in a bottom up manner. High level abstractions have to be made concrete. The constructed system eventually has to express all its goals and desires as physical action, and must extract all its knowledge from physical sensors. Thus the designer of the system is forced to make everything explicit. Every short-cut taken has a direct impact upon system competence, as there is no slack in the input/output representations. The forms of the low-level interfaces have consequences which ripple through the entire system.

3.3.1 Evolution

I already have an existence proof of the possibility of intelligent entities — human beings. Additionally many animals are intelligent to some degree. (This is a subject of intense debate, much of which really centers around a definition of intelligence.) They have evolved over the 4.6 billion year history of the earth. It is instructive to reflect on the way in which earth-based biological evolution spent its time. Single cell entities arose out of the primordial soup roughly 3.5 billion years ago. A billion years passed before photosynthetic plants appeared. After almost another billion and a half years, around 550 million years ago, the first fish and vertebrates arrived, and then insects 450 million years ago. Then things started moving fast. Reptiles arrived 370 million years ago, followed by dinosaurs at 330 and mammals at 250 million years ago. The first primates appeared 120 million years ago and the immediate predecessors to the great apes a mere 18 million years ago. Man arrived in roughly his present form 2.5 million years ago. He invented agriculture a mere 19000 years ago, writing less than 5000 years ago and "expert" knowledge only over the last few hundred years. This suggests that problem solving behavior, language, expert knowledge and application, and reason, are all rather simple once the essence of being and reacting are available. That essence is the ability to move around in a dynamic environment, sensing the surroundings to a degree sufficient to achieve the necessary maintenance of life and reproduction. This part of intelligence is where evolution has concentrated its time—it is much harder. This is the physically grounded part of animal systems. An alternative argument to the preceding is that in fact

once evolution had symbols and representations things started moving rather quickly. Thus symbols are the key invention and AI workers can sidestep the early morass and start working directly with symbols. But I think this misses a critical point, as is shown by the relatively weaker performance of symbol based mobile robots as opposed to physically grounded robots. Without a carefully built physical grounding any symbolic representation will be mismatched to its sensors and actuators. These groundings provide the constraints on symbols necessary for them to be truly useful. [26] has argued rather eloquently that mobility, acute vision and the ability to carry out survival related tasks in a dynamic environment provide a necessary basis for the development of true intelligence.

3.3.2 Reactive control

In artificial intelligence, reactive planning denotes a group of techniques for action selection by autonomous agents. These techniques differ from classical planning in two aspects. First, they operate in a timely fashion and hence can cope with highly dynamic and unpredictable environments. Second, they compute just one next action in every instant, based on the current context. Reactive planners often (but not always) exploit reactive plans, which are stored structures describing the agent's priorities and behavior.

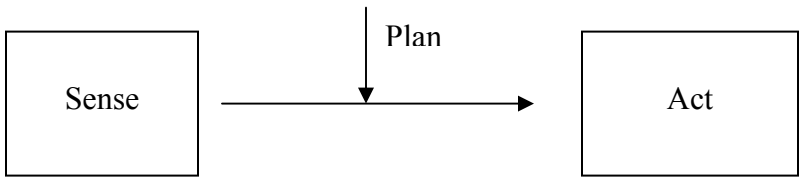


Figure3.3 The direct coupling of action to sensing is the defining characteristic of reactive control.

A condition action rule, or if-then rule, is a rule in the form: if condition then action. These rules are called productions. The meaning of the rule is as follows: if the condition holds, perform the action. The action can be either external (e.g., pick something up and move it), or internal (e.g., write a fact into the internal memory, or evaluate a new set of rules). Conditions are normally Boolean and the action either can be performed, or not.

Production rules may be organized in relatively flat structures, but more often are organized into a hierarchy of some kind. For example, subsumption architecture consists of layers of interconnected behaviors, each actually a finite state machine which acts in response to an appropriate input. These layers are then organized into a simple stack, with higher layers subsuming the goals of the lower ones. Other systems may use trees, or may include special mechanisms for changing which goal / rule subset is currently most important. Flat structures are relatively easy to build, but allow only for description of simple behavior, or require immensely complicated conditions to compensate for the lacking structure.

3.3.3 Behavior based AI

Behavior Based Artificial Intelligence (BBAI) is a methodology for developing AI based on a modular decomposition of intelligence. It was made famous by Rodney Brooks and his subsumption architecture was one of the earliest attempts to describe a mechanism for developing BBAI. It is extremely popular in robotics and to a lesser extent intelligent virtual agents because it allows the successful creation of real-time dynamic systems that can run in complex environments. For example, it underlies the intelligence of the Sony Aibo and many RoboCup robot teams.

The most important attribute of a behavior based system is that the intelligence is controlled by a set of independent semi-autonomous modules. In the original systems, each module was actually a separate device or was at least conceived of as running on its own processing thread. Generally though the modules are just abstractions. BBAI may be seen as a software engineering approach to AI, perhaps akin to object oriented design.

BBAI is often associated with reactive planning, but the two are not synonymous. Brooks advocated an extreme version of cognitive minimalism which required initially that the behavior modules were finite state machines and thus contained no conventional memory or learning. This is associated with reactive AI because reactive AI requires reacting to the current state of the world, not to an agent's memory or preconception of that world. However, learning is obviously key to realistic strong AI, so this constraint has been relaxed, though not entirely abandoned.

Mataric [18] states that behavior-based robotics designs controllers for omarwing robots with intelligent behavior, based on "a biologically inspired philosophy that favors parallel, decentralized architectures." It draws on the idea of providing the robots with a range of basic behaviors and letting the environment determine which behavior is more suitable as a response to a certain stimulus. Sukhatme and Mataric [21] define behaviors as "real-time processes that take inputs from sensors and/or other behaviors and send outputs to actuators and/or other behaviors." In behavior-based robotics, basic behaviors are fundamental units for control, reasoning, and learning. The environment plays a central role in activating a certain basic behavior at any given time. The behavior modules and the coordination mechanisms are usually designed through a trialand- error process in which a designer progressively changes them and tests the resulting behavior in the environment.

Extending the reactive and behavior-based approaches to a multi-agent domain will lead to completely distributed systems with no centralized controller. Behavior-based robotics has been an active and popular approach to robot control in the multi-robot domain, allowing multi-robot systems to adapt to realworld environments.

Behavior-based systems are praised for their robustness and simplicity of construction. Based on Brooks' behavior based subsumption architecture [6], for example, Parker developed the ALLIANCE architecture [4] for controlling groups of heterogeneous robots and demonstrated it on a group of four physical robots performing puck manipulation and box-pushing. He divides tasks into subtasks, with groups of behaviors addressing each subtask. At the highest level, "mutually inhibitory motivational behaviors are designed to direct the overall behavior of a robot, which in turn activates lower-level behaviors to perform a subtask" [4]. Along with the typical sensor-based conditions that might trigger motivational behaviors, Parker adds impatience and acquiescence. Impatience increases if no other robot is tempting to solve the subtask associated with a motivational behavior, while acquiescence inhibits the behavior if the robot is not successful in the subtask. The combination of the ordinary conditions of impatience and acquiescence in a group enables the robots to cooperate in striving to achieve an overall task.

Balch [7] takes motor schemas as an example of behavior-based robot control. Motor schemas are the reactive components in an Autonomous Robot Architecture (AuRA) [7]. AuRA's design integrates "deliberative planning at the top level with behavior-based motor control at the bottom." The lower levels are concerned with executing reactive behaviors. Individual motor schemas, or primitive behaviors, express separate goals or constraints for a task. For example, the schemas for a navigational task may involve avoiding obstacles and moving to a goal. Since schemas are independent, they can run concurrently, providing parallelism and efficiency. Motor schemas may be grouped to form more complex, emergent behaviors.

Chapter 4

Design and Implementation of the Proposed Coordination Scheme

Chapter 4: Design and Implementation of the Proposed Coordination Scheme

4.1 Finite State Automation (FSA)

Finite state machine (FSM) is model of behavior of a system. FSMs are used widely in computer science. Modeling behavior of agents is only one of their possible applications. A typical FSM, when used for describing behavior of an agent, consists of a set of states and transitions between these states. The transitions are actually condition action rules. In every instant, just one state of the FSM is active, and its transitions are evaluated. If a transition is taken it activates another state. That means, in general transitions are the rules in the following form: if condition then activate-new-state. But transitions can also connect to the 'self' state in some systems, to allow execution of transition actions without actually changing the state.

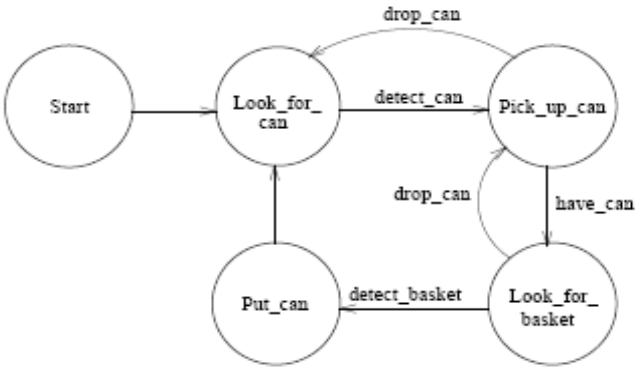


Figure4.1. A state diagram for a trash collecting robot

There are two ways of how to produce behavior by a FSM. They depend on what is associated with the states by a designer --- they can be either 'acts', or scripts. An 'act' is an atomic action that should be performed by the agent if its FSM is the given state. This action is performed in every time step then. However, more often is the latter case. Here, every state is associated with a script, which describes a sequence of actions that the

agent has to perform if its FSM is in a given state. If a transition activates a new state, the former script is simply interrupted, and the new one is started. If a script is more complicated, it can be broken down to several scripts and a hierarchical FSM can be exploited. In such an automaton, every state can contain substrates. Only the states at the atomic level are associated with a script (which is not complicated) or an atomic action.

Computationally, hierarchical FSMs are equivalent to FSMs. That means that each hierarchical FSM can be converted to a classical FSM. However, hierarchical approaches facilitate designs better. See the paper of Mataric [25] for an example of ASM of computer game bots, which uses hierarchical FSMs.

4.1.1 Design of the mine-clearing robot using FSA technique

Design of the mine-clearing robot required that I write schemas for ‘mine detection’, ‘obstacle detection’, ‘robot detection’, ‘avoid obstacles’ etc so that they could then be strung up in a flow chart to form an FSA. Moreover the FSA should be that such that no planning or lengthy data processing should be done by the robot. The algorithm should be ‘constant time’ ‘Sense-Act’ mechanism that is the basis of all the behavioral systems. Keeping these constraints in mind I designed the basic FSAs for simple tasks and then combined them to form a complete control algorithm for different tasks that were assigned to the robots. Keep in mind that a very big constraint in the robot coordination was that no communication should be involved between the robots. This made the task quite difficult.

In the following pages in Figures 4.2 to 4.5 I have listed the algorithms for these basic tasks using flow charts.

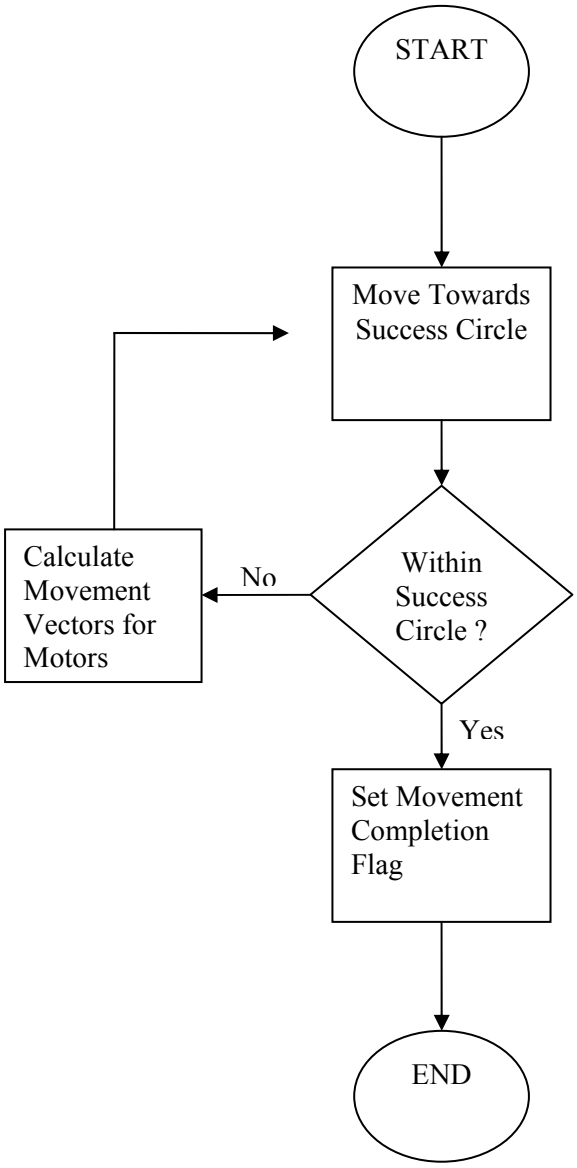


Figure4.2. Flow Chart for 'Move to Goal' Behavior

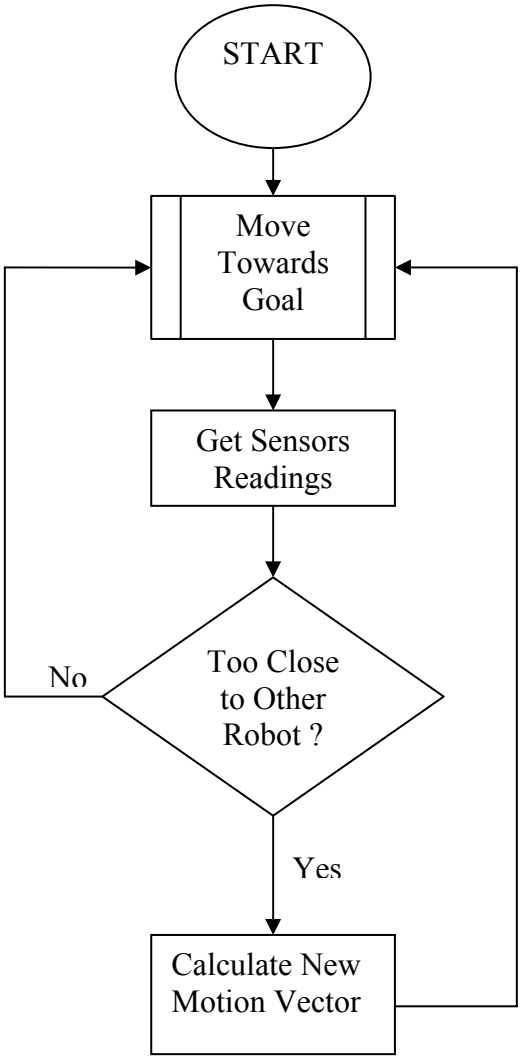


Figure4.3. Flow Chart for 'Avoid Robot' Behavior

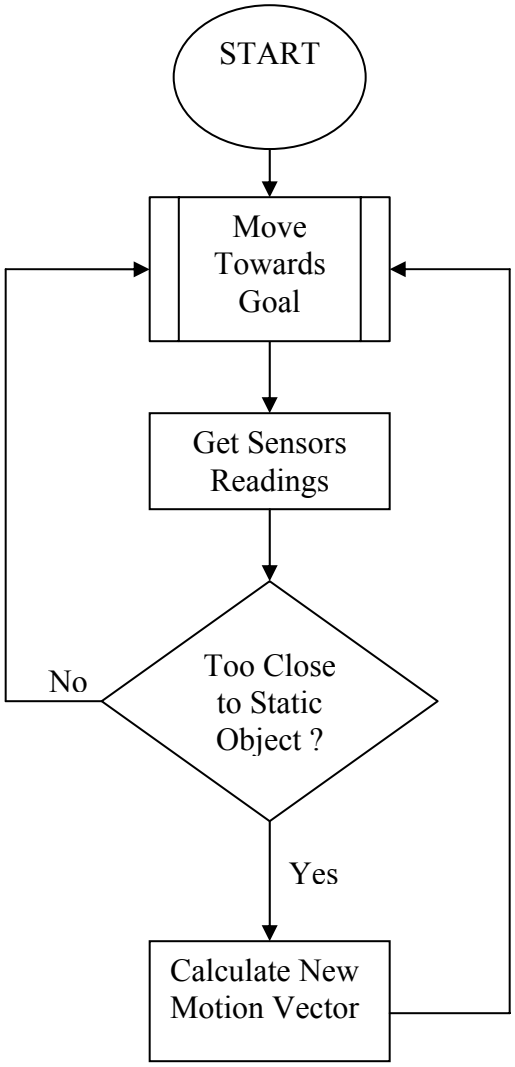


Figure4.4. Flow Chart for 'Avoid Static Object' Behavior

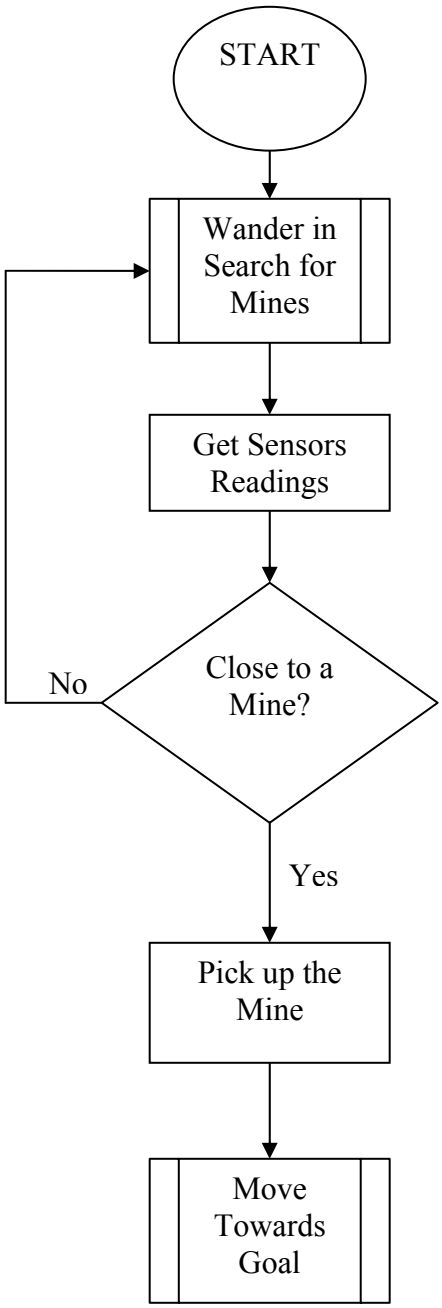


Figure4.5. Flow Chart for 'Search for Mine' Behavior

4.2 Reinforcement learning

In computer science, reinforcement learning is a small area of machine learning that deals with how a robot should take actions in an environment so as to maximize a long-term reward. Reinforcement learning algorithms try to find a policy that maps states of the world to the actions the robot should take in those states.

The environment is typically formed as a finite-state Markov decision process (MDP), and reinforcement learning algorithms for this context are highly related to dynamic programming techniques. State transition probabilities and reward probabilities in the MDP are typically stochastic but stationary over the course of the problem.

Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Further, there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). The exploration vs. exploitation tradeoff in reinforcement learning has been mostly studied through the multi-armed bandit problem.

Formally, the basic reinforcement learning model consists of:

- a set of environment states S;
- a set of actions A; and
- a set of scalar "rewards" in $\mathbb{R}$.

At each time t, the agent perceives its state $s_t \in S$ and the set of possible actions A(st). It chooses an action $a \in A(s_t)$ and receives from the environment the new state st+1 and a reward rt+1. Based on these interactions, the reinforcement learning agent must develop a policy $\pi: S \rightarrow A$ which maximizes the quantity R=r0+r1+...+rn for MDPs which have a terminal state, or the quantity $R=\Sigma_t \gamma^t r_t$ for MDPs without terminal states (where γ is some "future reward" discounting factor between 0.0 and 1.0).

Thus, reinforcement learning is particularly well suited to problems which include a long-term versus short-term reward tradeoff. It has been applied successfully to various

problems, including robot control, elevator scheduling, telecommunications, backgammon and chess

After I have defined an appropriate return function to be maximized, I need to specify the algorithm that will be used to find the policy with the maximum return. There are two main approaches, the value function approach and the direct approach.

The direct approach entails the following two steps: a) For each possible policy, sample returns while following it. b) Choose the policy with the largest expected return. One problem with this is that the number of policies can be extremely large, or even infinite. Another is that returns might be stochastic, in which case a large number of samples will be required to accurately estimate the return of each policy. The direct approach is the basis for the algorithms used in Evolutionary robotics.

The problems with the direct approach might be ameliorated if I assume some structure in the problem and somehow allow samples generated from one policy to influence the estimates made for another. Value function approaches do this by only maintaining a set of estimates of expected returns for one policy π (usually either the current or the optimal one). In such approaches one attempts to estimate either the expected return starting from state s and following π thereafter,

$$V(s) = E[R|s,\pi],$$

or the expected return when taking action a in state s and following π thereafter,

$$Q(s,a) = E[R|s,\pi],$$

If someone gives us Q for the optimal policy, I can always choose optimal actions by simply choosing the action with the highest value at each state. In order to do this using V, I must either have a model of the environment, in the form of probabilities P(s'|s,a), which allow us to calculate Q simply through

$$Q(s,a) = \sum V(s')P(s' \mid s,a),$$

or I can employ so-called Actor-Critic methods, in which the model is split into two parts: the critic, which maintains the state value estimate V, and the actor, which is responsible for choosing the appropriate actions at each state.

Given a fixed policy π, Estimating $E[R|.]$ for $\gamma=0$ is trivial, as one only has to average the immediate rewards. The most obvious way to do this for $\gamma>0$ is to average the total return after each state. However this type of Monte Carlo sampling requires the MDP to terminate.

Thus carrying out this estimation for $\gamma > 0$ in the general does not seem obvious. In fact, it is quite simple once one realizes that the expectation of R forms a recursive Bellman equation: $E[R \mid st] = rt + \gamma E[R \mid st + 1]$

By replacing those expectations with our estimates, V, and performing gradient descent with a squared error cost function, I obtain the temporal difference learning algorithm TD(0). In the simplest case, the set of states and actions are both discrete and I maintain tabular estimates for each state. Similar state-action pair methods are Adaptive Heuristic Critic(AHC), SARSA and Q-Learning. All methods feature extensions whereby some approximating architecture is used, though in some cases convergence is not guaranteed. The estimates are usually updated with some form of gradient descent, though there have been recent developments with least squares methods for the linear approximation case.

The above methods not only all converge to the correct estimates for a fixed policy, but can also be used to find the optimal policy. This is usually done by following a policy π that is somehow derived from the current value estimates, i.e. by choosing the action with the highest evaluation most of the time, while still occasionally taking random actions in order to explore the space. Proofs for convergence to the optimal policy also exist for the algorithms mentioned above, under certain conditions. However, all those proofs only demonstrate asymptotic convergence and little is known theoretically about the behavior of RL algorithms in the small-sample case, apart from within very restricted settings.

An alternative method to find the optimal policy is to search directly in policy space. Policy space methods define the policy as a parameterized function $\pi(s,\theta)$ with

parameters θ. Commonly, a gradient method is employed to adjust the parameters. However, the application of gradient methods is not trivial, since no gradient information is assumed. Rather, the gradient itself must be estimated from noisy samples of the return. Since this greatly increases the computational cost, it can be advantageous to use a more powerful gradient method than steepest gradient descent. Policy space gradient methods have received a lot of attention in the last 5 years and have now reached a relatively mature stage, but they remain an active field. There are many other approaches, such as simulated annealing, that can be taken to explore the policy space. Work on these other techniques is less well developed

4.2.1 Coordination based on learning

I believed that coordination could be further improved if the robots could remember and learn with experience so that they could anticipate and improvise more. For this purpose I used the method of Q-learning to program robots to learn from their experiences. As I will show in the next sections, the technique works fine with simple tasks but with complex tasks further capabilities like fast episodic memory recall is needed to achieve better results. These I have left for future work during my PhD.

4.2.2 Learning in simple tasks

Q-learning is a reinforcement learning technique that works by learning an action-value function that gives the expected utility of taking a given action in a given state and following a fixed policy thereafter. A strength with Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment. A recent variation called delayed-Q learning has shown substantial improvements, bringing PAC bounds to Markov Decision Processes.

The core of the algorithm is a simple value iteration update. For each state, s, from the state set S, and for each action, a, from the action set A, I can calculate an update to its expected discounted reward with the following expression:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \phi max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where r is an observed real reward, α is the learning rate such that $0 < \alpha < 1$, and φ is the discount rate such that $0 < \varphi < 1$

4.3 Implementation of the design in hardware

4.3.1 Robot Platform

Differential Drive with Encoder Feedback

For wheeled and tracked robots, *differential steering* is the most common method for getting the machine to go in a different direction. The technique is exactly the same as steering a military tank: one side of wheels or treads stops or reverses direction while the other side keeps going. The result is that the robot turns in the direction of the stopped or reversed wheel or tread. Because of friction effects, differential steering is most practical with two-wheel-drive systems. Additional sets of wheels, as well as rubber treads, can increase friction during steering.

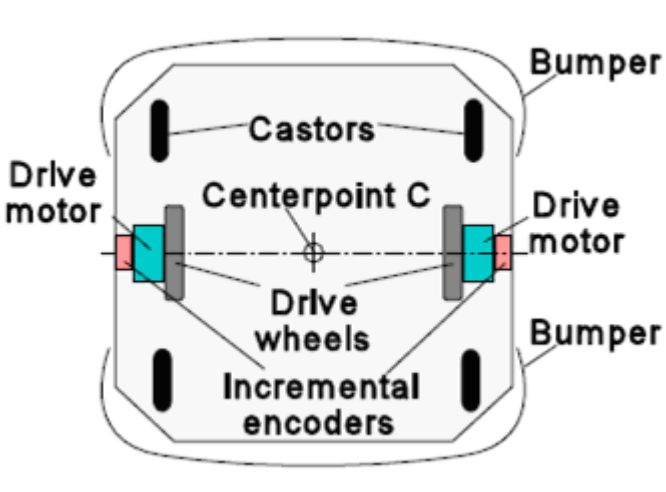


Figure4.6. A Typical Differential Drive Robot

Ackerman Drive

Pivoting the wheels in the front is yet another method for steering a robot. Robots with car-type steering are not as maneuverable as differentially steered robots, but they are better suited for outdoor uses, especially over rough terrain. You can obtain somewhat better traction and steering accuracy if the wheel on the inside of the turn pivots more

than the wheel on the outside. This technique is called Ackerman steering and is found on most cars but not on as many robots.

Differential Drive

One of the greatest drawbacks of the differentially steered robot is that the robot will veer off course if one motor is even a wee bit slow. You can compensate for this by monitoring the speed of both motors and ensuring that they operate at the same rpm. This typically requires a control computer, as well as added electronics and mechanical parts for sensing the speed of the wheels. Car-type steering, described in the last section, is one method for avoiding the problem of "crabbing" as a result of differences in motor speed simply because the robot is driven by just one motor. But car-type steering makes for fairly cumbersome indoor mobile robots. A better approach is to use a single drive motor powering two rear wheels and a single steering wheel in the front. This arrangement is just like a child's tricycle. The robot can be steered in a circle just slightly larger than the width of the machine. Be careful about the wheelbase of the robot (distance from the back wheels to the front steering wheel). A short base will cause instability in turns, and the robot will tip over opposite the direction of the turn. Tricycle-steered robots must have a very accurate steering motor in the front. The motor must be able to position the front wheel with sub-degree accuracy. Otherwise, there is no guarantee the robot will be able to travel a straight line. Most often, the steering wheel is controlled by a servo motor. Servo motors use a "closed-loop feedback" system that provides a high degree of positional accuracy.

Laser Range finder

The *LRS90-3 Laser Radar Scanner* is an adaptation of the basic LD90-3 electronics, fiber optically coupled to a remote *scanner unit*. The scanner package contains no internal electronics and is thus very robust under demanding operating conditions typical of industrial or robotics scenarios. The motorized scanning head pans the beam back and forth in the horizontal plane at a 10-Hz rate, resulting in 20 data-gathering sweeps per second. Beam divergence is 0.3° (5 milli radians), with the option of expanding in the vertical direction if desired up to 2°.

4.4 Implementation of the design in software

For implementation of my design in software I needed a multi-robot simulator. The commercial ones available were very expensive and were not meant for research. I first tried 'Player-Stage' but it had some errors. Then I found MissionLab® from Georgia Institute of Technology. It is a wonderful software. Its only limitation is that it is Linux based and its full documentation is not available, but one can experiment around and use it to accurately simulate multi robot scenarios. I have given detail about this software in Appendix B.

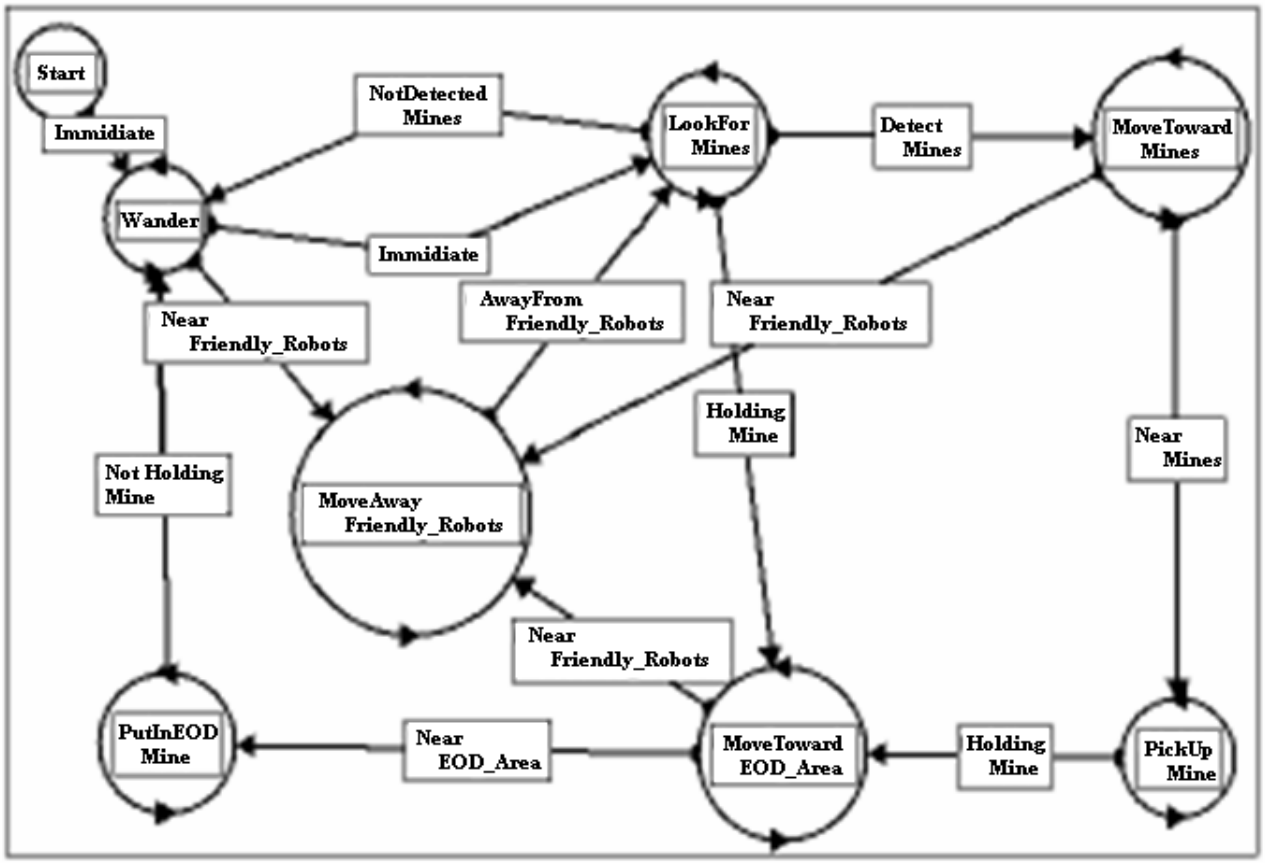


Figure4.7. A Finite State Automation (FSA) diagram for mine clearing robots

4.4.1 Simulation setup for FSA for mine-clearing robot

An area of 100 x 100 m was chosen as the test field for simulated robots. The automation diagram for the robots for this task is shown in Fig4.7. The experiment was carried out using varying number of robots, mines and obstacles. The results are presented in the next chapter.

4.4.2 Simulation setup for mine-clearing robots using Q-Learning

MissionLab provides built-in functions for Q-Learning. In Fig:. the experimental setup for this task is shown. Again varying numbers of Robots were used for this experiment and the results are presented in the next chapter.

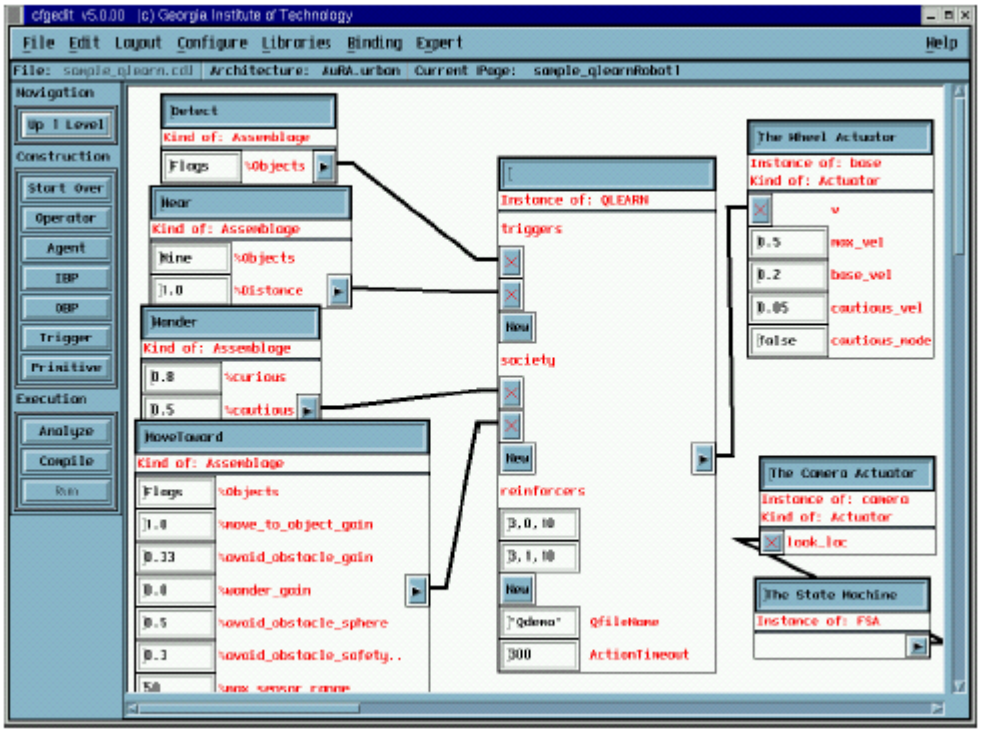


Figure4.8. Implementing Q-learning in Mission Lab for foraging task

4.5 Experimental setup: Sut-Pala game

I chose a playground competitive game to demonstrate the coordination skills of the robots. The game 'sut-pala' is shown in Fig.4.9. The game is played between two teams consisting of equal number of players (robots in our case).

The task of first team (shown in blue) is to start at 'side A', cross the playing field (drawn on the ground) to side B, and then come back to side A. Meanwhile they have to avoid the players of the other team which try to catch them. Second team (red) is restricted in its movements by the lines drawn on the ground. The captain can move in the first horizontal area as well as the central vertical area. The rest of the players of this team can only move in their respective horizontal areas.

Both the teams consisted of intelligent robots and as the game progressed different patterns emerged both in predators and the prey. An important thing to note is that there was no real intelligence (memory based) involved, only the state transitions sufficed.

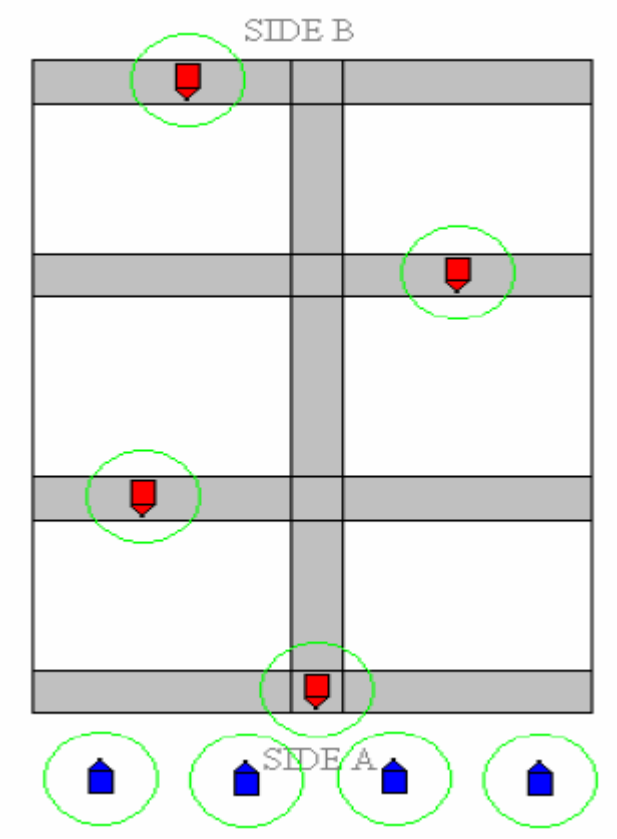


Figure4.9. Sut-Pala a competitive game of two teams

Chapter 5

Simulation Results and Discussion

Chapter 5: Simulation Results and Discussion

5.1 Simulation Results for Mine Clearing robots using FSA Technique

The experiments started with deploying simple robots with no communication and anticipatory powers and the results were used in comparative studies of other experiments. The simulation environment was filled with many movable target objects (mines). The robots once localized repelled each other and were attracted by the goal objects (mines). Once a robot had picked up a mine it went to drop it at an EOD area. If a wandering robot saw another robot moving the object it did not interfere and left to find some other target object.

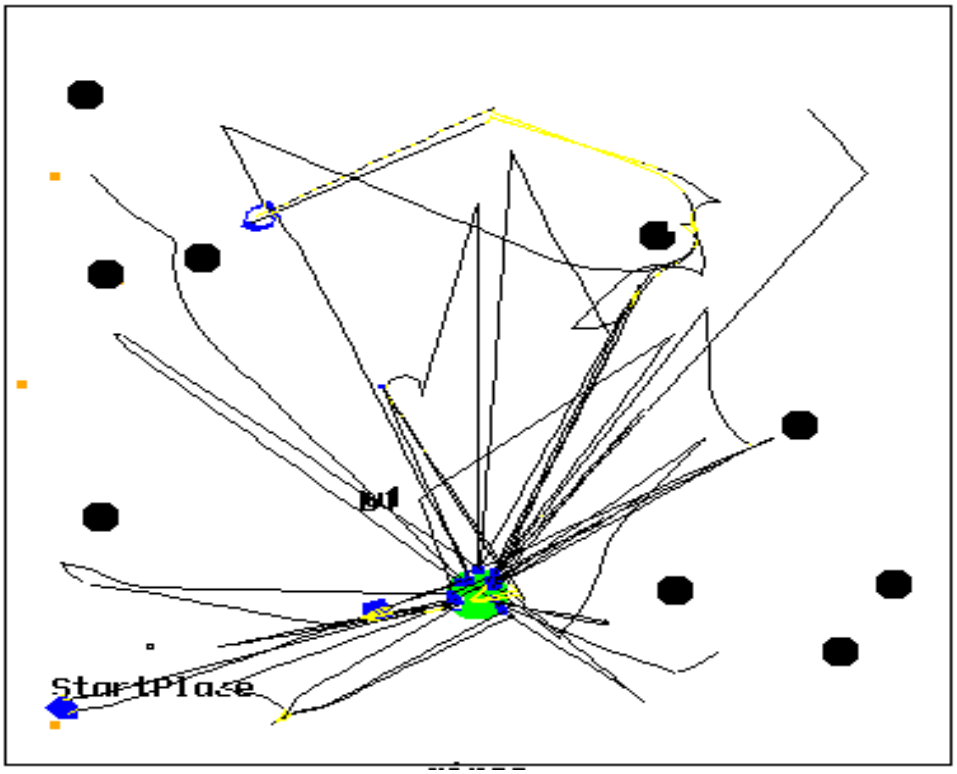


Figure5.1. Four robots clearing a minefield while avoiding each other and obstacles

From Table5.1 we see that the robots expertly handled the task. With the increase in number of robots up to an optimal value the time of finishing the task reduces

significantly. If the number of robots is increased beyond this optimal value then there is a decline in efficiency of the robots as the time to finish the job increases. The robots now take longer to finish the task because now most of the time is wasted in recognizing and avoiding the other robots.

No. of Robots	Time to Completion	% of Task Completed
3	20 Min	95
6	9 Min	96
10	2 Min	99
15	2 Min	89
20	8 Min	90

Table5.1. Mine Searching Task with an area of 100 x 100 m and no obstacles.

In the second phase of experiments where we added 10 obstacles in the arena, we see a similar result, though many of the mines are also left out. This behavior is due to the fact that many of the mines are hidden behind the obstacles and are avoided along with the obstacle. A better obstacle avoidance algorithm may rectify this problem.

No. of Robots	Time to Completion	% of Task Completed
3	25 Min	85
6	17 Min	88
10	2 6 Min	90

Table5.2. Mine Searching Task with an area of 100 x 100 m and 10 obstacles.

5.2 Simulation Results for Mine Clearing robots using Q-Learning

In this phase of experiments robots which used Q-Learning method to learn to collect the mines and bring back to the EOD area were used. The robots started learning

by trying to maximize the rewards. The rewards were offered for moving, finding a mine, picking up a mine and bringing it back to the EOD area. The rewards would decrease if the robot remained stationary, collided with and obstacle or did not find a mine. The behaviors or schemas for these actions like moving, finding a mine, picking up a mine etc were already programmed but there was no FSA this time round. The robots learned in real time by maximizing rewards. Table5.3 shows the results for these experiments. As expected the robots need more time to learn but the quickly adapt to the environment.

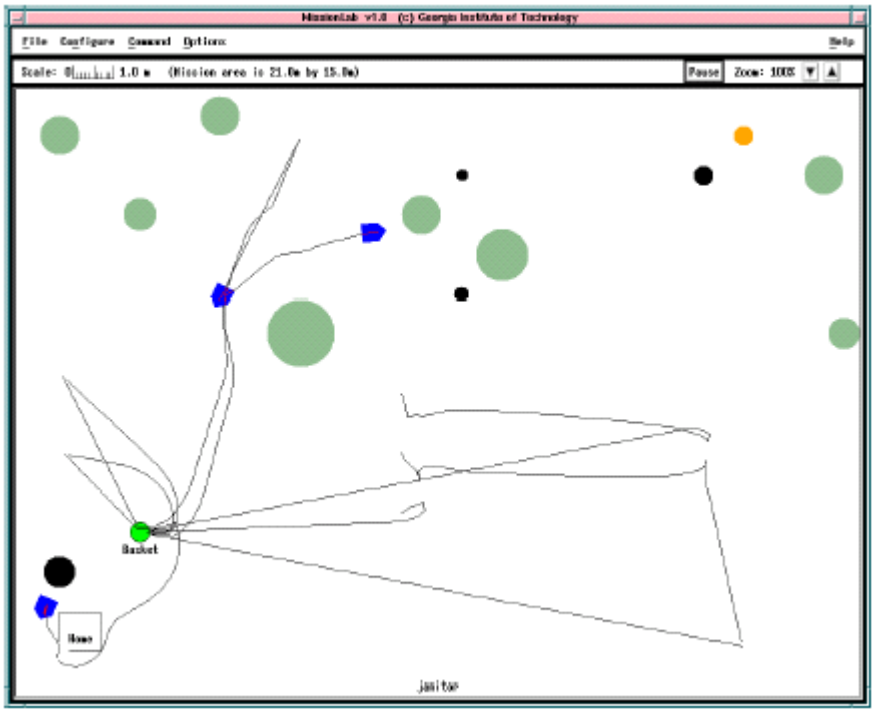


Figure5.2. Four robots clearing the mines and avoiding obstacles while learning

No. of Robots	Time to Completion	% of Task Completed
3	30 Min	95
6	20Min	96
10	32 Min	97
15	40Min	78

Table5.3. Mine Searching Task using Q- Learning Approach.

5.3 Simulation Results for Sut-Pala Game

These experiments were carried out with teams consisting of 2, 3, 4 and 5 simulated robots. The results showed that nearly 61% of the times the robots exhibited intelligent emergent behavior and successfully dogged or hunted down the robots of the opposing team. Another important thing is that no communication was involved in both the teams.

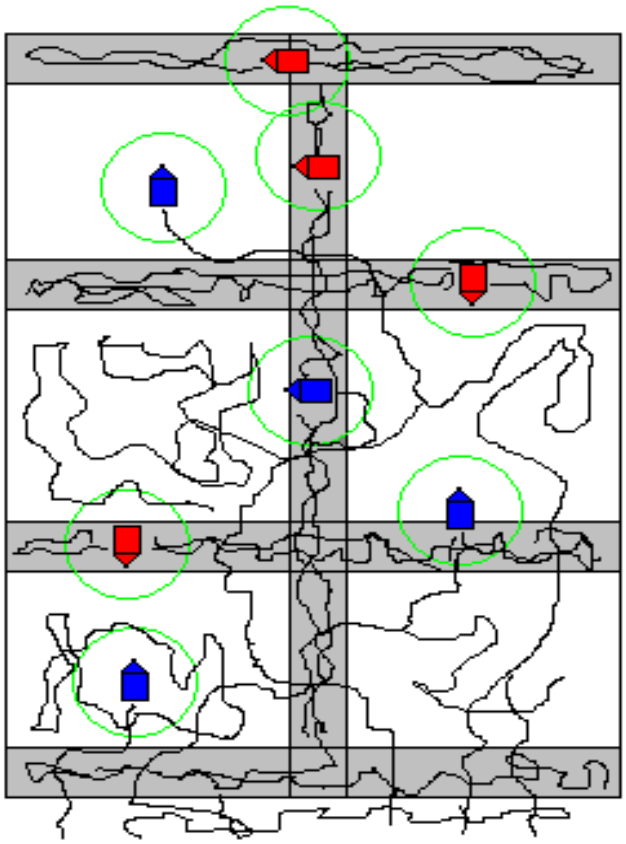


Figure5.2. Robots making trails as the game progresses

Chapter 6

Conclusions and Continuing Work

Chapter 6: Conclusions and Further Work

6.1 Achievements

Lets recap on what have we achieved up till now and what remains to be done. At the start I listed some goals that needed to be achieved, to be of any contributable value to the ultimate task of multi-robot coordination. I shall now list the achievements in a similar manner so that a comparison of goals and achievements can be made.

- I was able to achieve communication-less coordination in simple tasks such as foraging (mine clearing robot in our case)
- I was able to achieve (a degree less efficient though) communication-less coordination in complex tasks such as the competitive game Sut-Pala
- Learning capability was successfully incorporated in robots doing simple group-level tasks which shows great promise in this field.
- I could not properly use Q-learning for complex tasks but as Newton said “I did not fail, I just found another way that doesn’t work”.

6.2 Future targets and open issues

When we explore things that are unknown to us previously we end up getting the knowledge of a few of them but finding many other things that need to be explored. Similar was the case with this project. I found ways to achieved coordination without communication in some cases but this led to so many more questions that it would require another research project to uncover those.

Future research should focus on making the robots anticipate and improvise to truly capture the essence of intelligence that humans possess. A well coordinated team of football players rarely uses verbal calls or signals to coordinate their attack; rather they have prior plan settings and coordinate by observing others and anticipating their next moves.

It remains an open issue, whether such improvisation would be possible in behavior based systems or will it slow down the system performance to a level which is not entirely acceptable.

6.3 Coordination in complex tasks

Further research is needed to find methodologies that allow for designing, and reliably predicting, swarm behavior, given only features of the individual swarm members. Here, stigmergy is an essential tool for systematically studying swarm-behavior, even though other tracking methods are available. Recently Bristol robotics laboratory has developed an ultrasonic position tracking system for swarm research purposes.

Complex problems can be solved using multiple robots but coordinating these robots carry out these tasks is a more difficult problem. New ways have to be explored to effectively control a swarm of robots and make the swarm operate reliably to solve problems.

6.4 Anticipation and improvisation

In artificial intelligence, anticipation is the concept of an agent making decisions based on predictions, expectations, or beliefs about the future. It is widely considered that anticipation is a vital component of complex natural cognitive systems. As a branch of AI, anticipatory systems are a specialization still echoing the debates from the 1980s about the necessity for AI for an internal model.

The anticipation of future states is also a major evolutionary and cognitive advance. Anticipatory agents belonging to Rosen's definition are closer to humans capabilities of taking decisions at a certain time T taking into account the effects of their own actions at different future timescales T + k. Machine learning methods started to integrate these capabilities in an implicit form as in reinforcement learning systems [49] where they learn to anticipate future rewards and punishments caused by current actions (Sutton & Barto, [49]. Moreover anticipation enhanced performance of machine learning

techniques to face with complex environments where agents have to guide their attention to collect important information to act.

6.5 On the horizon

Swarm Intelligence-based techniques can be used in a number of applications. The U.S. military is investigating swarm techniques for controlling unmanned vehicles. ESA is thinking about orbital swarm for self assembly and interferometry. NASA is investigating the use of swarm technology for planetary mapping. A 1992 paper by M. Anthony Lewis and George A. Bekey discusses the possibility of using swarm intelligence to control nano-bots within the body for the purpose of killing cancer tumors. Artists are using swarm technology as a means of creating complex interactive environments. Disney's The Lion King was the first movie to make use of swarm technology (the stampede of the wildebeasts scene). The Lord of the Rings film trilogy also made use of similar technology, known as Massive, during battle scenes. Swarm technology is particularly attractive because it is cheap, robust, and simple.

6.6 FPGA based controller for reactive control of a mobile robot

The recent trend in robot control is to have a microcontroller based robot brain to control the robot motion as well as do the sensing and planning part. As discussed earlier in behavior based robot control the planning part is done at the design level and the robot control is left to sense and act cycle. It is important to note here that there is a ‘Cycle’ of sensing and acting in robot control algorithms, whereas in insects and low intelligence animals there is no sign of a ‘Cycle’ in sensing and acting. Instead the sensing and acting is carried out in parallel. This is due to the fact that robots run their algorithms sequentially or at most using time division as in case of multitasking but still true parallelism is lacking. Also we have seen that the level of intelligence presented at a group level by the social insect like bees and ants is much higher than that of the state of the art robots of the present day. Perhaps this large gap it is due to this intrinsic parallelism found in insect’s brain. So in future work I would like to design an FPGA

based controller for the behaviors in a robot. The advantages of this approach would include 1. Close resemblance in control mechanism to there real life counterparts. 2. Parallelism in computation will hopefully bring a revolutionary change in the growth of a robots brain. A rough first draft of the design is presented in Figure6.1.

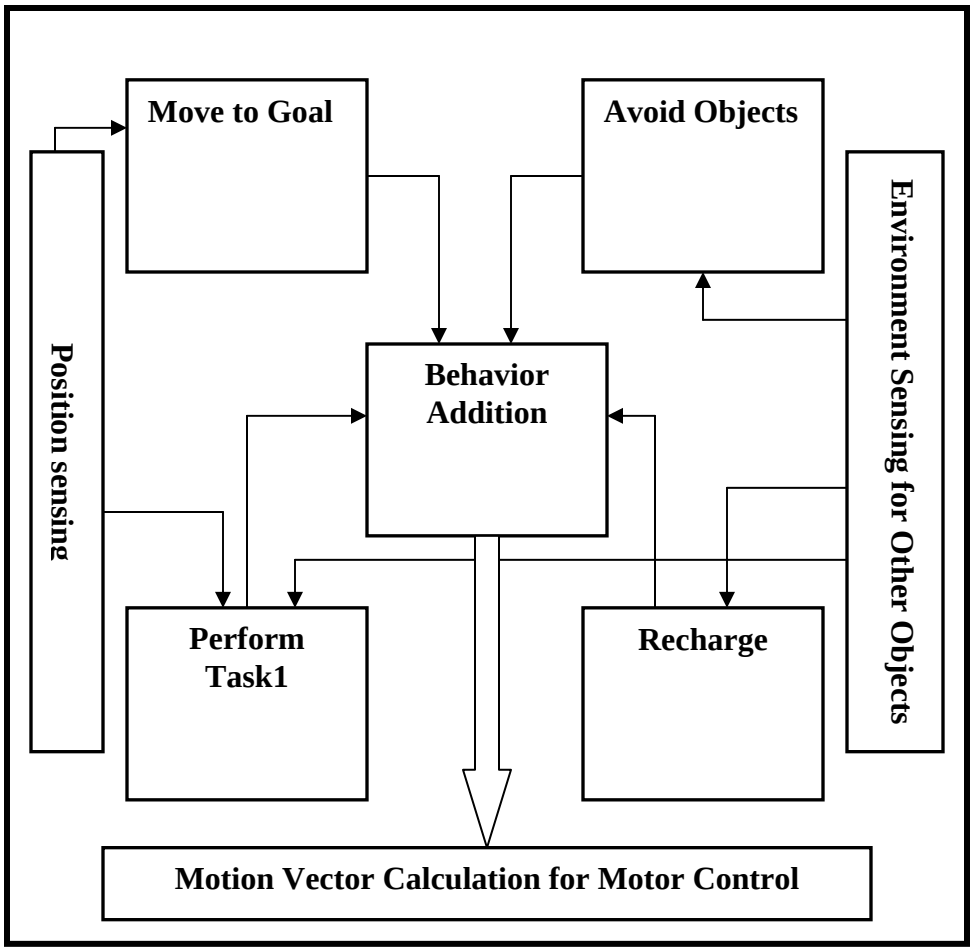


Figure6.1. An FPGA implementation of the behavior based controller

References

[1] Balch, T., Arkin, R.C. **"**Behavior-based formation control for multirobot teams" 1998 *IEEE Transactions on Robotics and Automation* 14 (6), pp. 926-939

[2] Balch, Tucker, Arkin, Ronald C. **"**Communication in reactive multiagent robotic systems" 1994 *Autonomous Robots* 1 (1), pp. 27-52

[3] Arkin, Ronald C "Cooperation without Communcation: Multiagent Schema based Robot navigation" 1992 Journal of robotic Systems 9(3) pp 351-364

[4] Stan Franklin "Coordination without Communication"**,**unpublished. http://www.cs.memphis.edu/~franklin/coord.html

[5] Balch, Tucker, Arkin, Ronald C. "AuRA: Principles and Practice in Review" Journal of Experimantal and Theoretical Artificial Intelligence 9:2-3 pp. 175-189 April-September 1997.

[6] Brooks, Rodney A. **"**Elephants don't play chess" 1990 *Robotics Amsterdam* 6 (1-2), pp. 3-15

[7] Kaiser, M., Dillmann, R., Friedrich, H., Lin, I., Wallner, F., Weckesser, P. **"**Learning coordination skills in multi-agent systems" 1996 *IEEE International Conference on Intelligent Robots and Systems* 3, pp. 1488-1495

[8] Paul E. Rybski, Amy Larson, Harini Veeraraghavan, Monica LaPoint, and Maria Gini, "Communication strategies in Multi-Robot Search and Retrieval: Experiences with MinDART", DARS 2004, Toulouse, France, June 2004.

[9] Arkin, Ronald C **"**A Neural Schema Architechture for mobile robots"

[10] Brooks, R.A. **"**New approaches to robotics" 1991 *Science* 253 (5025), pp. 1227-1232

[11] Brooks, R.A. **"**Intelligence without representation" *Artificial Intelligence* 47 (1-3), pp. 139-159

[12] Holland, O., Melhuish, C. **"**Stigmergy, self-organization, and sorting in collective robotics" 1999 *Artificial Life* 5 (2), pp. 173-202

[13] MacKenzie, Douglas C., Arkin, Ronald C. **"**Evaluating the usability of robot programming toolsets" 1998 *International Journal of Robotics Research* 17 (4), pp. 381-401.

[14] Di Caro, Gianni, Dorigo, Marco **"**Mobile agents for adaptive routing" 1998 *Proceedings of the Hawaii International Conference on System Sciences* 7, pp. 74-83

[15] Endo, Y. **"**Anticipatory and Improvisa-tional robot via recollection and exploitation of episodic memories". 2005 *AAAI Fall Symposium - Technical Report* FS-05-05, pp. 57-64

[16] Life, April 1996, p. 44; Time, March 25, 1996, p. 60

[17] Robin Murphy. *"Introduction to AI Robotics". MIT Press* 2000.

[18] Matarić, M. (1997). *Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior*. Hexmoor, H., Horswill, I., and Kortenkamp, D., eds., Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents 9(2-3):323-336.

[19] Brooks, R. (1986). *A Robust Layered Control System for a Mobile Robot*. IEEE Journal of Robotics and Automation 2(1):14-23.

[20] Maes, P. (1989). *The Dynamics of Action Selection*. In Proceedings of the International Joint Conference on Artificial Intelligent (IJCAI-89). Detroit, MI, pp. 991- 997.

[21] Pirjanian, P. (2000). *Multiple Objective Behavior-based Control.* Robotics and Autonomous Systems 31(1-2):53-60.

[22] Payton, D., Keirsey, D., Kimble, D., Krozel, J., Rosenblatt, J. (1992). *Do Whatever Works: A Robust Approach to Fault-tolerant Autonomous Control*. Applied Intelligence

[23] Maes, P.,Brooks, R. (1990). *Learning to Coordinate Behaviors*. In Proceedings of the American Association of Artificial Intelligence (AAAI-91), Boston, MA, pp. 796- 802.

[24] Pirjanian, P. (1999). *Behavior Coordination Mechanisms – State-of-the-Art.*

University of Southern California Institute of Robotics and Intelligent IRIS Technical Report IRIS-99-375.

[25] Matarić, M. (1992). *Integration of Representation Into Goal-Driven Behavior-Based Robots.* In IEEE Transactions on Robotics and Automation 8(3):304-312.

[26] Nicolescu, M., Matarić, M. (2001*). Experience-based Representation Construction: Learning from Human and Robot Teachers.* In Proceedings of IEEE/RSJ International Conference on Robots and Systems (IROS-01). Maui, Hawaii, pp. 740-745.

[27] Nicolescu, M., Matarić, M. (2000*). Extending Behavior-Based Systems Capabilities Using An Abstract Behavior Representation*. University of Southern California Institute for Robotics and Intelligent Systems IRIS Technical Report IRIS-00-389.

[28] Cao, Y., Fukunaga, A., Kahng, A. (1997). *Cooperative Mobile Robotics: Antecedents and Directions*. Autonomous Robots 4:7-27.

[29] Dudek, G., Jenkin, M., Milios, E. (2002*). A Taxonomy of Multirobot Systems. In Balch, T. and Parker, L., eds. Robot Teams:* From Diversity to Polymorphism, Natick, Massachusetts: A.K. Peters, pp. 3-22.

[30] Balch, T., Parker, L., eds. (2002). *Robot Teams: From Diversity to Polymorphism. A. K. Peters.*

[31] Bohringer, K., Brown, R., Donald, B., Jennings, J. (1997*). Distributed Robotic Manipulation: Experiments in Minimalism*. In O. Khatib ed. Experimental Robotics

IV, Lecture Notes in Control and Information Sciences 223, pp. 11-25. Berlin: Springer Verlag.

[32] Gerkey, B., Matarić, M. (2004). *A Formal Analysis and Taxonomy of Task Allocation in Multi-robot Systems.* International Journal of Robotics Research 23(9):939- 954.

[33] Howard, A., Parker, L., Sukhatme, G. (2004). *The SDR Experience: Experiments with a Large-scale Heterogenous Mobile Robot Team*. In 9th International Symposium on Experimental Robotics. Singapore. June 18-20.

[34] Konolige, K., Fox, D., Ortiz, C., Agno, A., Eriksen, M., Limketkai, B., Ko, J., Morisset, B., Schulz, D., Stewart, B., Vincent, R. (2004). *Centibots: Very Large Scale Distributed Robotic Teams*. In 9th International Symposium on Experimental Robotics. Singapore. June 18-20.

[35] Rahimi, M., Pon, R., Kaiser, W., Sukhatme, G., Estrin, D., Srivastava, M. (2004). *Adaptive Sampling for Environmental Robotics.* In Proceedings of the International Conference on Robotics and Automation, pp. 3537-3544. New Orleans, Louisiana.

[36] Zhang, B., Sukhatme. G., Requicha, A. (2004). *Adaptive Sampling for Marine Microorganism Monitoring.* To appear in Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Sendai, Japan. October 30-September 2.

[37] Holland, O., Melhuish, C. (1999) *Stigmergy, Self-organization, and Sorting in Collective Robotics.* Artificial Life 5(2):173-202.

[38] Franks, N., Deneubourg J-L (1997) *Self-organising Nest Construction in Ants: Individual Worker Behavior and the Nest's Dynamics.* Animal Behaviour 54:779-796.

[39] Kube, C., Zhang, H. (1996). *The Use of Perceptual Cues in Multi-robot Boxpushing.* In Proceedings IEEE International Conference on Robotics and Automation (ICRA-96), pp. 2085-2090, Minneapolis, Minnesota.

[40] Jones, C., Matarić, M. (2004). *Automatic Synthesis of Communication-Based Coordinated Multi-Robot Systems.* In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Sendai, Japan. October 30-September 2.

[41] Reynolds, C. (1987). Flocks, Herds, and Schools: *A Distributed Behavior Mode*l. Computer Graphics 21(4):25-34.

[42] Matarić, M. (1995). *Designing and Understanding Adaptive Group Behavior.* Adaptive Behavior 4(1):51-80.

[43] Jones, C., Matarić, M. (2003). *Adaptive Division of Labor in Large-Scale Minimalist Multi-Robot Systems*. In Proceedings of the IEEE/RSJ International Conference on Robotics and Intelligent Systems (IROS). Las Vegas, Nevada. pp. 1969- 1974.

[44] Fredslund, J., Matarić, M. (2002). *A General, Local Algorithm for Robot Formations. IEEE Transactions on Robotics and Automation*, Special Issue on Multi- Robot Systems 18(5):837-846.

[45] Gerkey, B., Matarić, M. (2001). *Principled Communication for Dynamic Multi-Robot Task Allocation.* In Proceedings International Symposium on Experimental Robotics 2000. Waikiki, Hawaii, pp. 341-352.

[46] Parker, L. (1997). *Behavior-Based Cooperative Robotics Applied to Multi-Target Observation*. In Bolles, R., Bunke, R., Noltemeier, H. eds. Intelligent robots: Sensing, modeling, and planning, R. Bolles, H. Bunke, and H. Noltemeier. World Scientific. pp. 356-373.

[47] Lerman, K., Galstyan, A. (2002). *Mathematical Model of Foraging in a Group of Robots: Effects of Interference.* Autonomous Robots 13(2):127-141.

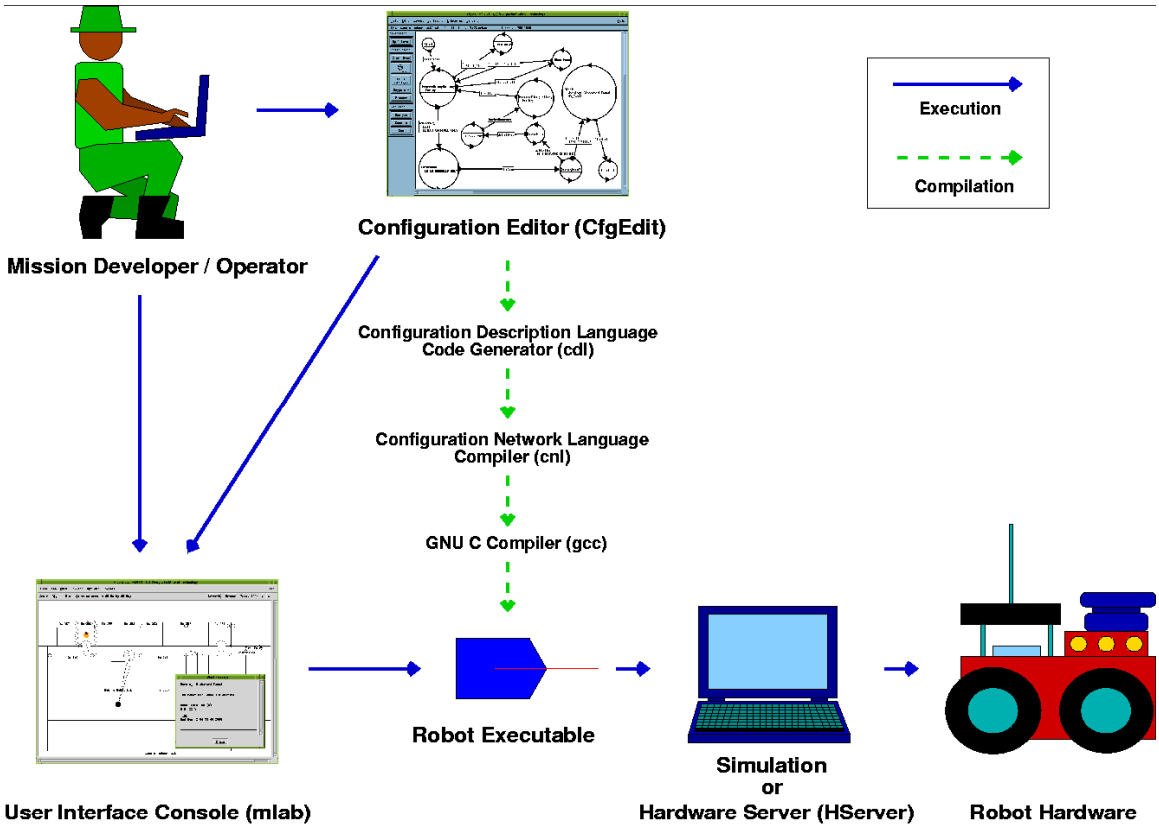
[48] Lerman, K., Galstyan, A., Martinoli, A., Ijspeert, A. (2001). *A Macroscopic Analytical Model of* Collaboration in Distributed Robotic Systems. Artificial Life 7(4):375-393.

[49] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An introduction.* MIT Press

What isMissionLab?

MissionLab Overview

MissionLab is a powerful set of software tools for developing and testing behaviors for single robots and a group of robots. Code generated by MissionLab can directly control commercial robots. ATRV-Jr / Urban Robot (iRobot), AmigoBot / Pioneer AT / Pioneer 2DX (ActivMedia, Inc.), and Nomad-150 / 200 (Nomadic Technologies, Inc.) are among those robots MissionLab has supported successfully. A primary strength of MissionLab is its support of both simulated and real robots. A developer can experiment with behaviors in simulation and then run those same configurations on mobile robots (Figure 1).



MissionLab has a distributed architecture. Thus, the main user's console can run on one computer while multiple robot control executables are distributed across a network, potentially on-board the actual robots they control. The core of the MissionLab tool-set is composed of six primary components:

• mlab: mlab is a console-like program from which a user monitors the progress of experimental runs of the robot executables. Locations of the robots and detected obstacles are examples of various data mlab can monitor. When mlab is used for simulation (as opposed to controlling mobile robots), it serves as a sensor and actuator simulator from the point of view of the robot executable. On mobile robots, the actual sensors are used instead. of mlab.

• CfgEdit: The Configuration Editor, or CfgEdit, is a graphical tool for building robot behaviors. The designer can build complex control structures with the point and click of a mouse. CfgEdit generates source code which, when compiled, can directly control a simulated or real robot.

• cdl: The cdl code generator translates the CDL (Configuration Description Language), which is generated by CfgEdit, into CNL (Configuration Network Language) code. In general, users will not need to be concerned with CNL. However, because programming in CNL is very similar to programming in the C language, advanced users may develop their own primitive behaviors and store them as a library and/or write their own control programs without using CfgEdit.

• cnl: The cnl compiler compiles CNL code generated by the cdl code generator, and produces C++ code. Once this C++ coded is compiled with the GNU C Compiler (gcc), the compiled program (or robot executable) may now directly control a robot. The cnl compiler is automatically invoked by CfgEdit when needed.

• HServer: HServer (Hardware Server) directly controls all the robot hardware, either via TCP/IP or a serial link, and provides a standard interface for all the robots and sensors. The CfgEdit generated code uses this standard interface to control the real

robots. HServer also provides direct control, configuration, and status of the robots and sensors.

• CBRServer: CBRServer (CaseBased Reasoning Server) generates a mission plan based on specs provided by the user by retrieving and assembling components of previously stored successful mission plans.. In addition to CDL and CNL described above, there are two more original languages that were specifically developed for the MissionLab system:

• CMDL: The Command Description Language (CMDL) may optionally be used for describing simple sequential robot missions. A CMDL file, containing both background and command information will run the code.

Code Listing

```
/******************************************************************************
 **                                                                          **
 **                           MOVE_TO_GOAL.cnl                               **
 **                                                                          **
 **                                                                          **
 **                                                                          **
******************************************************************************/

/* $Id: MOVE_TO_GOAL.cnl,v 1.1.1.1 2006/07/12 13:37:59 omar Exp $ */

#include "cnl.inc"

/******************************************************************************
*/

procedure Vector MOVE_TO_GOAL with
   Vector   goal_rel_loc;
   double   success_radius;
header
body
   if( len_2d(goal_rel_loc) > success_radius )
   {
      /* generate a vector towards the goal */
      output = goal_rel_loc;
      unit_2d(output);
   }
   else
   {
      /* return a zero vector if within the success circle */
      VECTOR_CLEAR(output);
   }

   if( debug )
   {
      fprintf(stderr,"MOVE_TO_GOAL(%d) r=%.1f output=(%.1f %.1f)\n",
             robot_id, success_radius,output.x,output.y);
   }
pend


/******************************************************************************
# $Log: MOVE_TO_GOAL.cnl,v $
# Revision 1.1.1.1  2007/04/12 13:37:59  omar
# MissionLab 7.0
# Added RCS id and log strings.
#*****************************************************************************
/
```

```
/*-----------------------------------------------------------------------

      avoid_robot.cnl

      Avoid other robots.


      Written by:  Omar


-----------------------------------------------------------------------*/

/* $Id: AVOID_ROBOT.cnl,v 1.1.1.1 2007/04/13 13:37:58 omar Exp $ */


#include "cnl.inc"

procedure Vector AVOID_ROBOT with

double      sphere;
double      min_range;
Robots      readings;

header

Vector      sum;
Vector      t1,t2;
int   i;
double      len, mag;

body

VECTOR_CLEAR(output);
VECTOR_CLEAR(sum);

for(i=0; i<readings.cnt; i++)
      {

  if( debug )
   fprintf(stderr,"AVOID_ROBOT: robot detected at <%.1f %.1f>\n",
     readings.v[i].x, readings.v[i].y);

      len = len_2d(readings.v[i]);
      if (len < EPS_ZERO)
            {
            len = 0.0001;
            readings.v[i].y = 1.0;
            }
      mag = 0;
      if (len <= min_range)
            {
            mag = GT_INFINITY;
            }
      else if (len < sphere)
            {
                double denom = sphere - min_range;
                if( fabs(denom) < EPS_ZERO )
```

```
                {
               mag = GT_INFINITY;
                }
                else
                {
               mag = (sphere - len) / denom;
                }
            }
      if (fabs(mag) > EPS_ZERO)
            {
            t1.x = -readings.v[i].x;
            t1.y = -readings.v[i].y;
            unit_2d(t1);
            times_2d(t1, mag, t2);
            add_2d(t2, sum, sum);
            }
      }
output = sum;

if( debug )
   fprintf(stderr,"AVOID_ROBOT: <%.1f %.1f>\n",output.x, output.y);

pend


/*****************************************************************
# $Log: AVOID_ROBOT.cnl,v $
# Revision 1.1.1.1  2006/07/12 13:37:58  omar
# MissionLab 7.0
#
#****************************************************************/




/*-----------------------------------------------------------------

      AVOID_OBJECTS.cnl


      Written by:  omar


-----------------------------------------------------------------*/

/* $Id: AVOID_OBJECTS.cnl,v 1.1.1.1 2006/07/12 13:37:58 omar Exp $ */


#include "cnl.inc"

/*****************************************************************
*/

procedure Vector AVOID_OBJECTS with
   double   sphere;
```

```
   double   safety_margin;
   ObjectList   objlist;
header
   int          first_time=TRUE;
   double       old_safety_margin = 0;
body
   if( debug )
   {
      fprintf(stderr,"AVOID_OBJECTS(sphere = %.1f, safety_margin =
%.1f\n",
      sphere, safety_margin);
   }

   if( first_time || old_safety_margin != safety_margin )
   {
      /*
       * Report the safety margin I are using to the console so the
       * graphics can use it for showing virtual collisions
       */
      char buf[80];

      sprintf(buf, "%f",safety_margin);
      exec_put_console_state(SAFETY_MARGIN_MSG,buf);

      old_safety_margin = safety_margin;
      first_time = FALSE;
   }

   VECTOR_CLEAR(output);

   // For each object
   for(int i=0; i<objlist.count; i++)
   {
      if( debug )
         cerr << objlist.objects[i] << '\n';

      gt_Point contribution;

      // compute the closest point of the obstacle to the center of the
robot.
      gt_Point pt = objlist.objects[i].closest_point();
      double dist = len_2d(pt);

      /* Handle the case where within the max repulsion zone */
      if (dist < safety_margin )
      {
       /* generate an infinite (around 100000) vector away from the
obstacle*/

         if (dist < EPS_ZERO) // Epsilon Zero (~0.00001): Stop divide
by 0 error
         {
          /* Handle the case where an obstacle is EXACTLY centered
             where the robot is as a special case because I can't
             use the vector to determine a direction to move.
             So, arbitraily choose straight ahead direction for vector.
          */
```

```
          contribution.x = -GT_INFINITY;
          contribution.y = 0;
       }
       else
       {
          contribution = pt;
            unit_2d(contribution);
            mult_2d(contribution, GT_INFINITY);
       }
      }

      /* handle case where within the linear repulsion zone */
      else if (dist <= safety_margin + sphere )
      {
         /* set the magnitude of the repulsion vector (0...1) based on
how
          far I have intruded into the zone.
          Magnitude is 0 on outside edge (dist = sphere).
          Magnitude is 1 on inside edge (dist = safety_margin)

          sphere is the size of the zone.
          I are at distance (dist - safety_margin) beyond the inner
edge of
          the zone or (sphere - (dist - safety_margin)) from the outer
edge.

          So, take our distance from the outer edge and divide by the
size
          of the zone to get a ratio from 0 to 1 with zero at the
outside
          and 1 at the inside.
        */

         double mag = (sphere - (dist - safety_margin)) / sphere;

       contribution = pt;
         unit_2d(contribution);
         mult_2d(contribution, mag);
      }

      /* otherwise, outside obstacle's sphere of influence, so ignore
it */
      else
      {
         contribution.x = 0;
         contribution.y = 0;
      }

      if( debug )
         fprintf(stderr,"contribution = <%.1f, %.1f>\n",
          contribution.x, contribution.y);

      // Add it to the running sum */
      plus_2d(output, contribution);
   }

   // Note: Vector generated is pointing towards the obstacle,
```

```
   //       now flip it to be a movement vector away from the obstacles
   output.x = -output.x;
   output.y = -output.y;

   if( debug )
   {
      fprintf(stderr,"AVOID_OBJECTS: output vector <%.1f %.1f>\n",
       output.x, output.y);
   }
pend


/*****************************************************************
# $Log: AVOID_OBJECTS.cnl,v $
# Revision 1.1.1.1  2007/03/06 13:37:58  omar
# MissionLab 7.0
#
#
#
#*****************************************************************
/


/*****************************************************************
 **                                                             **
 **                          QLEARN.cc                          **
 **                                                             **
 **  Written By:  Omar                          **
 **                                                             **

*****************************************************************/

/* $Id: qlearn.cc,v 1.1.1.1 2006/07/12 13:38:00 omar Exp $ */

#include <string.h>
#include <time.h>
#include <iostream>

#include "qlearn.h"

using std::cout;
using std::endl;

/* Constructor for Qlearn
 * Parameters may be adjusted using accessor methods.
 *
 * numstates---the number of states the system could be in.
 * numactions--the number of actions or outputs to
 *             select from.
 * criteria----should be DISCOUNTED or AVERAGE.
 * seedin------the seed.
 */

double ** array2d(int x, int y)
{
  double **temp;
  int i, j;
```

```
  // OMAR - gcc 3.4
  //temp = new (double *)[x];
  temp = new double *[x];

  for (i = 0; i < x; i++)
    {
      temp[i] = new double[y];
    }

  for (i = 0; i < x; i++)
    {
      for (j = 0; j < y; j++)
        {
          temp[i][j] = 0;
        }
    }

  return (temp);
}

//make sure this doesn't leak
void delarray2d(double **array, int x)
{
  int i;

  if (array == NULL)
    return;

  for (i = 0; i < x; i++)
    {
      if (array[i] != NULL)
        delete[]array[i];
    }

  delete[]array;
}

Qlearn::Qlearn (int numstatesin, int numactionsin, char *filename,
                int robot_id, double alphaIn, double alphaDecay,
                double randomRate, double rScenarioDecay,
                int criteriain, int seedin)
{
  int i, j;
  FILE *file;
  int highest;

  dataName = new char[128];
  sprintf (dataName, "%d%c", robot_id, '\0');
  dataName = strcat (dataName, filename);
  dataName = strcat (dataName, ".dat");

  cout << "passed it to Q... " << dataName << endl;

  dataName = new char[128];
  sprintf (dataName, "%d%c", robot_id, '\0');
  dataName = strcat (dataName, filename);
```

```
  dataName = strcat (dataName, ".dat");

  tableName = new char[128];
  sprintf (tableName, "%d%c", robot_id, '\0');
  tableName = strcat (tableName, filename);
  tableName = strcat (tableName, ".tab");


  AVERAGE = 0;
  DISCOUNTED = 1;

  numstates = numstatesin;
  numactions = numactionsin;

  cout << "STATEs.." << numstates << "  Actions.." << numactions <<
endl;

  if ((criteriain != DISCOUNTED) && (criteriain != AVERAGE))
    {
      cout << "Error: INVALID CRITERIA.  Setting to DISCOUNTED" <<
endl;
      criteria = DISCOUNTED;
    }
  else
    criteria = criteriain;

  seed = seedin;
  srand ((unsigned) time (NULL));

  q = array2d (numstates, numactions);

  profile = array2d (numstates, numactions);

  p = array2d (numstates, numactions);

  last_policy = new int[numstates];
  for (i = 0; i < numstates; i++)
    last_policy[i] = 0;


  //NOTE---MAKE MY OWN RANDOM NUMBER GENERATOR
  for (i = 0; i < numstates; i++)
    {
      highest = 0;
      for (j = 0; j < numactions; j++)
        {
          //      q[i][j] = ((double) rand()/(double) RAND_MAX) - 1;
          q[i][j] = (rand () / (RAND_MAX + 1.0)) + 1;
          //q[i][j] = 0;
          p[i][j] = 0;
          profile[i][j] = 0;
          if (q[i][highest] > q[i][j])
            highest = j;
        }
      last_policy[i] = j;
    }
  xn = an = 0;
```

```
//variable initialization
  changes = 0;
  changesAll = 0;
  queries = 0;
  total_reward = 0;
  first_of_trial = 1;
  gamma = 0.8;
  alpha = alphaIn;
  randomrate = randomRate;
  randomratedecay = rScenarioDecay;
  seed = seedin;
  debug = FALSE;


  //  Read the Qtable...
  //   replace all variables just initialized with
  //   values from Qtable, if it exists

  if (dataName != NULL)
    {
      file = fopen (dataName, "r");
      if (file != NULL)
        {
          fclose (file);
          read ();
        }
    }

  alpha = alpha * alphaDecay;
  randomrate = randomrate * randomratedecay;
  first_of_trial = 1;

}

/*
 * Select an output based on the state and reward.
 *
 * curstate---the current state.
 * curreward--reward for the last output, positive
 *            numbers are "good."
 */

int Qlearn::query(int curstate, double curreward)
{
  int action;
  double randomnum;
  int highest;


  total_reward += curreward;
  queries++;

  //Check to see if the current state is out of range.
  if ((curstate > (this->numstates - 1)) || (curstate < 0))
    {
      return -1;
```

```
    }

  //Find approximate value of present state, and best action.
  //ie:  max q[yn][i] over all i, i is the best action.

  double Vn = -999999;          //very bad
  action = 0;
  for (int i = 0; i < numactions; i++)
    {
      if (q[curstate][i] > Vn)
        {
          Vn = q[curstate][i];
          action = i;
        }
    }
  if (first_of_trial != 1)
    {

      q[xn][an] = q[xn][an] + alpha * (curreward + gamma * Vn -
q[xn][an]);

      p[xn][an]++;              //count times in the last state/action
      profile[xn][an]++;        //count times for this trial
    }
  else
    first_of_trial = 0;

  for (int i = 0; i < numstates; i++)
    {
      highest = 0;
      for (int j = 1; j < numactions; j++)
        {
          if (q[i][highest] < q[i][j])
            highest = j;
        }

      if (highest != last_policy[i])
        {
          last_policy[i] = highest;
          changesAll++;

          if (i < numstates / 2)
            changes++;

        }
    }

  //Select random action, possibly
  //NOTE---MAKE MY OWN RANDOM NUMBER GENERATOR

  randomnum = rand () / (RAND_MAX + 1.0);
  if (randomnum <= randomrate)
    {
      action = ((int) (rand () / 10)) % numactions;
      if (action < 1)
        action = -1 * action;
    }
```

```
  printf ("%s LAST(STATE: %d, ACTION: %d) Action Chosen: %d reward:
%f\n",
          dataName, xn, an, action, curreward);

  //Remember for next time
  xn = curstate;
  an = action;

  //  if (logging) CheckForChanges();

  return action;
}


/**
 * Called when the current trial ends.
 *
 * @param Vn     double, the value of the absorbing state.
 * @param reward double, the reward for the last output.
 */
void
Qlearn::endTrial (double Vn, double rn)
{
  total_reward += rn;

  if (criteria == DISCOUNTED)
    {
      // Watkins update rule:
      q[xn][an] = (1 - alpha) * q[xn][an] + alpha * (rn + gamma * Vn);
    }
  else                          // criteria == AVERAGE
    {
      // average update rule
      q[xn][an] = (p[xn][an] * q[xn][an] + rn) / (p[xn][an] + 2);
      // see update above in query() for explanation
      // of this rule
    }

  p[xn][an] += 1;
  profile[xn][an] += 1;
}


/**
 * Called to initialize for a new trial.
 */
int
Qlearn::initTrial (int s)
{
  first_of_trial = 1;
  changes = 0;
  queries = 0;
  total_reward = 0;
  delarray2d (profile, numstates);
  profile = array2d (numstates, numactions);
  return (query (s, 0));
```

```
}

void
Qlearn::print ()
{
  int i, j;
  FILE *record;

  record = fopen (tableName, "a+");


  for (i = 0; i < numstates; i++)
    {
      for (j = 0; j < numactions; j++)
        {
          fprintf (record, "%g ", q[i][j]);
        }

    }
  fprintf (record, "\n");
  fclose (record);
}

//void Qlearn::save(char *filename)
void
Qlearn::save ()
{
  FILE *file;
  int i, j;

  file = fopen (dataName, "w");

  fwrite (&numstates, 1, sizeof (int), file);
  fwrite (&numactions, 1, sizeof (int), file);
  fwrite (&criteria, 1, sizeof (int), file);
//  fwrite(&changes, 1, sizeof(int), file);
  fwrite (&queries, 1, sizeof (int), file);
  fwrite (&first_of_trial, 1, sizeof (int), file);
  fwrite (&xn, 1, sizeof (int), file);
  fwrite (&an, 1, sizeof (int), file);
  fwrite (&debug, 1, sizeof (int), file);
  fwrite (&seed, 1, sizeof (long), file);

  fwrite (&total_reward, 1, sizeof (double), file);
  fwrite (&gamma, 1, sizeof (double), file);
  fwrite (&alpha, 1, sizeof (double), file);
  fwrite (&randomrate, 1, sizeof (double), file);
  fwrite (&randomratedecay, 1, sizeof (double), file);

  for (i = 0; i < numstates; i++)
    {
      for (j = 0; j < numactions; j++)
        {
          fwrite (&(q[i][j]), 1, sizeof (double), file);
          fwrite (&(p[i][j]), 1, sizeof (double), file);
          fwrite (&(profile[i][j]), 1, sizeof (double), file);
        }
```

```
      fwrite (&(last_policy[i]), 1, sizeof (int), file);
    }
//  readPolicy();

  fclose (file);

}

//void Qlearn::read(char *filename)
void
Qlearn::read ()
{
  FILE *file;
  int i, j;

  file = fopen (dataName, "r");

  fread (&numstates, 1, sizeof (int), file);
  fread (&numactions, 1, sizeof (int), file);
  fread (&criteria, 1, sizeof (int), file);
//  fread(&changes, 1, sizeof(int), file);
  fread (&queries, 1, sizeof (int), file);
  fread (&first_of_trial, 1, sizeof (int), file);
  fread (&xn, 1, sizeof (int), file);
  fread (&an, 1, sizeof (int), file);
  fread (&debug, 1, sizeof (int), file);
  fread (&seed, 1, sizeof (long), file);

  fread (&total_reward, 1, sizeof (double), file);
  fread (&gamma, 1, sizeof (double), file);
  fread (&alpha, 1, sizeof (double), file);
  fread (&randomrate, 1, sizeof (double), file);
  fread (&randomratedecay, 1, sizeof (double), file);

  for (i = 0; i < numstates; i++)
    {
      for (j = 0; j < numactions; j++)
        {
          fread (&(q[i][j]), 1, sizeof (double), file);
          fread (&(p[i][j]), 1, sizeof (double), file);
          fread (&(profile[i][j]), 1, sizeof (double), file);
        }
      fread (&(last_policy[i]), 1, sizeof (int), file);
    }

//  readPolicy();

  fclose (file);
}

void
Qlearn::readPolicy ()
{
  for (int i = 0; i < numstates; i++)
    {
      printf ("%d ", last_policy[i]);
    }
```

```
  printf ("\n");
}

/*****************************************************************
# $Log: qlearn.cc,v $
# Revision 1.1.1.1  2007/03/14 13:38:00  omar
# MissionLab 7.0
#
*****************************************************************/


/***********************************************
*
* This file DROP_OBJECT.cc was created with the command
* "/home/omar/MissionLab-7.0/bin/cnl -c -I../include -
I/home/omar/MissionLab-7.0/src/ipt/include DROP_OBJECT.cnl"
* using the CNL compiler, version 4.0.
*
***********************************************/


extern "C"
{
#include <cthread.h>
#include "../include/ipt/ipt.h"
}
#include <string.h>

#ifndef NULL
#define NULL 0
#endif
#define SRC_TYPE_LOCAL_NODE (0)
#define SRC_TYPE_REMOTE     (1)
#define SRC_TYPE_CONSTANT   (2)
#define require_input(name) parms->##name##require = true
#define NAME2NAME_REQUIRE(name) name##_require

extern int             _num_nodes;
extern int             _done_count;
extern mutex_t         _done_count_lock;
extern condition_t     _new_cycle;
extern char*           _prefix;
extern char*           _ipt_home;
extern IPCommunicator* _communicator;
extern IPConnection**  _ipt_modules;
extern int                 _ipt_is_active;
extern double          _zero_time;




/* Execution philosophy:
*
* Start of cycle
*      o Each node checks all of its users to see if its output will be
used
```

```
*        if no user wants the output, then the node is blocked.  Void
output
*        nodes never block.
*
*      o Nodes which are not blocked then execute, begining with the
sensors.
*
* End of cycle
*/

/*************************************************************************
***/
// Task Control Block class externs
template<class T> class _list {

    struct SLink
    {
        struct SLink *next;
        T             data;

        // constructor: create objects
        SLink(const T val) {

            data = val;
        }

        // destructor: needed so don't delete the data objects
        ~SLink() {

            // Nothing to do, here to keep from deleting the data
        }
    };
    SLink *last_ptr;


public:
    // constructor: create objects
    _list() {

        last_ptr = NULL;
    }

    // Is list empty?
    int isempty(void) const {

        return last_ptr == NULL;
    }

    // Append to end of list
    // Note: last_ptr->next is first record
    void append(const T data) {

        SLink *rec = new SLink(data);
        if( last_ptr )
        {
            rec->next = last_ptr->next;
            last_ptr->next = rec;
```

```
        }
        else
        {
            rec->next = rec;
        }
        last_ptr = rec;
    }

    // Return first element without removing it
    // Note: last_ptr->next is first record
    // Returns NULL if list is empty
    void *first(T *rec) const {

        if( this == NULL || last_ptr == NULL )
            return NULL;

        *rec = last_ptr->next->data;

        return (void *)last_ptr->next;
    }

    // Return next element without removing it
    // Note: must call first to setup the iterator
    // Returns NULL if list is empty
    void *next(T *rec, void *cur) const {

        if( (SLink *)cur == last_ptr )
            return NULL;

        *rec = ((SLink *)cur)->next->data;
        return (void *)((SLink *)cur)->next;
    }
};

enum _STAT_TYPE {STAT_UNKNOWN, STAT_BLOCKED, STAT_RUNNING};
class tcb {

public:
    enum _STAT_TYPE status_;
    _list<tcb *>    users_;
    const char     *task_name;
    bool            output_valid_;
    bool            done_;
    char           *activeFlag; // if not null, set to '1' if node ran
this cycle, '0' if blocked

    void update_status(void *port);
    virtual bool will_use_port(void *port) = 0;
    virtual void update_run_status() = 0;
    void reset_status();
    bool blocked();
    bool done();
    void block_me();
    void set_name(const char *name);
};
```

```
void reset_all_tasks();
void update_all_tasks();
extern _list<tcb *> list_of_tasks;

#line 1 "cnl.inc"
#line 9 "cnl.inc"

#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#include <string>

#include "gt_simulation.h"
#include "cnl.h"
#include "gt_std_types.h"
#include "gt_std.h"
#include "FSA_status.h"
#include "qlearn.h"

using std::cerr;
using std::string;

#line 25 "cnl.inc"
#line 180 "DROP_OBJECT.cc"

struct T_CMDLi : public tcb
{
    char* node_name;
    bool *recon_done;
    int recon_done_type;
    int recon_done_module;
    char* recon_done_src;
    tcb *recon_done_tcb;

    bool *moveto_done;
    int moveto_done_type;
    int moveto_done_module;
    char* moveto_done_src;
    tcb *moveto_done_tcb;

    Vector *recon;
    int recon_type;
    int recon_module;
    char* recon_src;
    tcb *recon_tcb;

    Vector *moveto;
    int moveto_type;
    int moveto_module;
    char* moveto_src;
    tcb *moveto_tcb;

    CNLString_t *cmdlFilename;
    int cmdlFilename_type;
    int cmdlFilename_module;
    char* cmdlFilename_src;
```

```
    tcb *cmdlFilename_tcb;

    CNLString_t *envFilename;
    int envFilename_type;
    int envFilename_module;
    char* envFilename_src;
    tcb *envFilename_tcb;
```

Table of Contents