THESIS

A METHOD FOR BIOMIMETIC DESIGN OF A COOPERATIVE MOBILE ROBOT SYSTEM TO ACCOMPLISH A FORAGING TASK

Submitted by

Jason Gerson Fleischer

Department of Mechanical Engineering

In partial fulfillment of the requirements

for the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 1999

COLORADO STATE UNIVERSITY

May 13, 1999

WE HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER OUR SUPERVISION BY JASON GERSON FLEISCHER ENTITLED, “A METHOD FOR BIOMIMETIC DESIGN OF A COOPERATIVE MOBILE ROBOT SYSTEM TO ACCOMPLISH A FORAGING TASK”, BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE.

Committee on Graduate Work

Advisor

Department Head

ABSTRACT OF THESIS

A METHOD FOR BIOMIMETIC DESIGN OF A COOPERATIVE MOBILE ROBOT SYSTEM TO ACCOMPLISH A FORAGING TASK

Traditional robot design methodologies often have problems dealing with dynamic and uncertain environments. Behavior-based robotic architectures have gained acceptance in the last ten years as a viable approach to this problem, but there are few formal design tools in the field. Designers often rely on their own experience or examples of other similar systems to create a new robotic system. In many cases, the design of a behavior-based robot is said to be inspired by how a natural species accomplishes a similar task. These designers are trying to capture the dynamics of the natural system's task accomplishment, and re-create a qualitatively similar set of dynamics in the artificial system. Unfortunately, this process is ad hoc, not well understood, and still relies mostly on the experience of the designer.

This thesis develops a methodology for aiding the designer of behavior-based robotic systems by mimicking biology, called biomimetic design. It examines the issues involved in designing a robotic system with biomimetic methods, and presents an abstraction of the translation between the two worlds. This abstraction is used to better understand the process and pitfalls of re-creating the dynamics of the natural system in an artificial one, and to suggest a methodology for creating such systems.

The methodology is applied to the design of a cooperative system of six small mobile robots to find and retrieve ten target objects, of unknown location, in an 8m x 8m environment. The robots are designed to mimic both the foraging division of labor and the path integration navigation methods of desert ants (genus *Cataglyphis*). Testing shows that the navigation system is able to navigate the robots successfully back to home base at rates of up to ten robots every ten minutes. Results also indicate that their task performance is likely to be further improved by the addition of landmark recognition abilities, as demonstrated by experienced *Cataglyphis* foragers.

Finally, the biomimetic design model is discussed, and directions for future work are suggested for both the implemented robotic system and the biomimetic design model.

Jason Gerson Fleischer
Department of Mechanical Engineering
Colorado State University
Fort Collins, CO 80523
Summer, 1999

ACKNOWLEDGEMENTS

This work would not have been possible without the help and support of many people. I wish to thank my parents because without their immovable support I never would have made it to this point. I also wish to thank my advisor, Dr. Wade Troxell for his guidance and Vicki Jensen, without whose organization skills I would not have graduated.

All of my friends played a large part in keeping me sane. I especially want to recognize Dennis Clark for his electronics expertise, and Laurie Huckaby who offered me hours of advice drawn from her own postgraduate work. Also deserving thanks are Tony Scott who is a valuable resource of C programming knowledge, and Robin Callahan for his heroic proof-reading. Finally, I'd like to thank the entire Legio Borealis for being there when I needed it.

This work was partially supported by a grant from the Center for Engineering Infrastructure and Science in Space at Colorado State University, which is funded through the Colorado Space Grant Consortium and the NASA College Space Grant Program.

DEDICATION

For my son, Jacob.

TABLE OF CONTENTS

Chapter 1 - Introduction

Even though intelligent robotics has produced some exceptional results in recent years, the design of such systems is still difficult, poorly understood, and unreliable. To correct this, reliable design tools and a science of robot design must be developed. Additionally, most research has concentrated on *single* robots working individually on a given task. This approach may not be enough to realize the full potential of robotic systems. *Groups* of robots cooperating together may be more appropriate in some situations. This thesis demonstrates how appropriate use of paradigms from biology can aid in the design of such a group. It will present a design methodology based on mimicking biology, which may help to create more reliable robotic systems of all types. This method is then applied to design a group of mobile robots that cooperate to accomplish a foraging task.

1.1 Why use multiple robots?

Isn't a single robot complicated enough to deal with? Shouldn't we concentrate on improving single robot systems, rather then wasting time on the added complexities of getting multiple robots working together coherently? There are, after all, several problems that exist only in the multiple robot domain: physical interference between robots, complex communication schemes, computationally expensive control schemes if control is centralized, and knowledge sharing between robots if control is distributed (Arkin, 1998). Certainly multiple robots create a complex situation for the designer. Yet there are potential benefits to this approach.

Several researchers have examined reasons to use multiple robots (Arkin, 1998; Cao, Fukanaga, & Kahng, 1997; Dudek, et. al., 1996), and have come up with very similar lists.

1) Multiple robots can perform tasks that are separated in space more efficiently then single robots by using a “divide and conquer” strategy.
2) Fault tolerance can be increased by having robots that are able to reorganize themselves “on the fly” to take on the task of a disabled co-worker.
3) Research into robotic cooperation could yield insights into problems in the social and life sciences. This approach is often studied in the field of Artificial Life (Lindgren & Nordahl, 1995).

4) Cost and design difficulty might be reduced when organizations of simpler robots are used instead of a single complex robot. However, this argument is sometimes used *against* multiple robot systems (Arkin, 1998), because some researchers believe that a poorly executed multiple robot solution can be more complex and costly, rather then less so.

The Far Side by Gary Larson

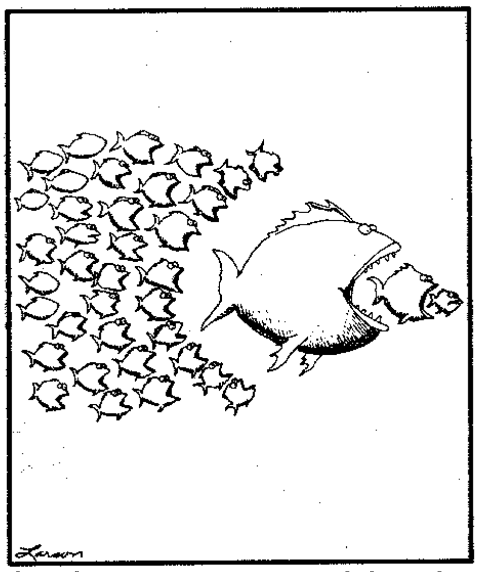


Figure 1.1. Multiple distributed agents versus a monolithic solution. THE FAR SIDE © 1981 FARWORKS, INC. Used by permission of UNIVERSAL PRESS SYNDICATE. All rights reserved.

But is there any case where a single robot just could not do the job? Dudek, et. al. (1996) postulate that there *are* tasks that computationally require multiple agents to complete. If a task is:

a) Geographically separated enough so that a single large robot would be impractical, and

b) requires sufficiently time synchronous operations at each location so that a single fast robot couldn't do it

then only multiple robots cooperating together can do the job. For example, when launching a missile two keys separated by a large distance need to be turned simultaneously. Note that such a task has the aspect of reason #1, being separated in space, but adds the complexity of time synchronous operation.

Another argument for multiple robot systems comes from the realms of biology. Many researchers have equated artificial systems to natural organisms to better understand one or the other, for instance (Simon, 1998; Braitenburg, 1984; Holland, 1975; Walter, 1953; Loeb, 1918). If this approach is valid, it generates an argument for multiple robot systems by existence proof. Most species in the world today exhibit some form of cooperative behavior between its members. In some species it is only the lowest level needed to ensure continued reproduction, but often it goes far beyond that. One of the most successful types of animals are the eusocial insects, such as bees and ants, who exhibit extreme cooperation among members of their hive or nest (Wilson, 1971). In fact most animals, including humans, live in some sort of social structure to maintain survivability and accomplish demanding tasks. Since so many species use cooperative behavior to ensure their viability, robotic systems might do well to try this method, too.

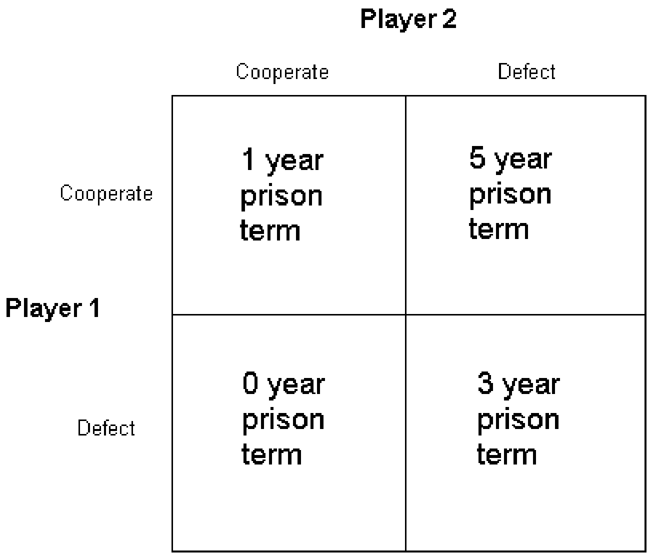
1.2 Cooperation

There are several different terms in use for multiple robots that cooperate with each other. These systems have been variously described as swarms, collectives, multi-agent, or are said to exhibit cooperative behavior (Dudek, et. al., 1996).

But for the sake of clarity and unity this thesis will need to select one of them to use exclusively. “Swarm” implies a very large number of robots, and therefore is not the generic case. “Collective” includes the idea of multiple manipulators controlled by a single centralized intelligence, which we wish to exclude. “Multi-agent” seems sufficiently generic, and is the favored terminology in artificial intelligence and artificial life research. However, this thesis will use the term *cooperative mobile robots* [CMR], after (Cao, et. al. 1997), because of its implications.

Cooperation is a vital part of applying multiple robots to a single task in an efficient manner. For instance, Parker (1995) showed that CMR that explicitly communicated with each other, a form of cooperation, performed a task more efficiently then those that did not. These results back up the intuition that a cooperating group of individuals – a society – will perform better at most tasks then the same group where each member ignores the others and acts as an individual.

Individual members of a system can benefit from cooperation, too. Disciplines such as artificial life, psychology, and ethology have used game theory as a model to find out why individuals cooperate. Research on a game called The Iterated Prisoner’s Dilemma has shown that if the proper conditions exist in the task and environment, cooperation can produce beneficial results for competing computer programs (Axelrod & Hamilton, 1981). The form of the Prisoner’s Dilemma game is shown in Figure 1.2. The game is played by two prisoners, who by cooperating or defecting will determine the jail sentence each will receive. The iterated form of the game means that the players will encounter each other more then once. The most successful algorithm for The Iterated Prisoner's Dilemma is generally some form of "Tit for Tat", a strategy which will always cooperate unless its partner defected in the last round of play. If Iterated Prisoner's Dilemma is a good model for certain problems facing natural systems, these results go a long way towards explaining cooperation of individuals among and between species. Dugatkin (1997) shows this to be the case in examples of altruistic behaviors from many different species, therefore giving another good theoretical reason for pursuing cooperation in robotic systems.



		Player 2	
		Cooperate	Defect
Player 1	Cooperate	1 year prison term	5 year prison term
	Defect	0 year prison term	3 year prison term

Figure 1.2. The prisoner's dilemma game. Payoffs are shown for player 1, but are the same for player 2.

In order to cooperate, agents must first communicate with each other, and then secondly they must use the information that they communicate to better their performance as individuals and/or as a group. Note, however, that communication is not the same thing as cooperation. Communication is necessary for cooperation, but not sufficient. That is, any system that cooperates must communicate, but not all systems that communicate are cooperating.

There are three broad categories of communication used in robotics (Parker, 1995). Explicit communication occurs when symbols representing data or commands are exchanged between agents. Implicit communication happens when agents gain useful information by observing each other perform actions in the environment. Stigmergic communication is when an agent gains useful information by observing the changes another agent has made to the environment. All three types of communication occur in the natural world. For instance, the honeybee's dance and human language are explicit. Implicit communication occurs when you approach a door at the same time as another person and they hold open the door to allow you to pass through first. Stigmergy is seen when one ant follows the pheromone trail another one laid to find food. Neither ant ever communicated directly with the other – they only changed or observed the change in the environment.

After communicating, the robots must use the information to better their performance. This can be individual or group performance, and generally is viewed as minimizing time or energy spent accomplishing a task. Alternatively, cooperation may be the thing that makes a task possible, such as when one robot alone is not big enough to move an object. When a robot's performance is viewed as some mixture of task completion and the time/energy to accomplish it, then we can enumerate three broad categories of cooperation. A robot may use its information to:

1) Share information that enables itself or another to either accomplish a task or accomplish it more efficiently in terms of time and/or energy costs.
2) Divide a task up spatially or temporally, so that it can be accomplished by more then one robot, thereby reducing time and/or energy expenditure.
3) Integrate more then one robot into a task so that it can be accomplished or reduce the time and/or energy to accomplish it.

1.3 The Foraging Task

Work with cooperative mobile robots has traditionally fallen into one of three "canonical task domains" (Cao, et. al., 1997), all displaying strong aspects of one or more of the reasons outlined in section 1.1 for using multiple robots:

- **Traffic Control** problems involving the physical organization of multiple robots such as collision avoidance, flocking, formation keeping, marching, and controlling roadway use.
- **Cooperative Manipulation**, where an object too large for a single robot can be moved by a group of robots working together.
- **Foraging**, a task where robots are required to find and pick up objects scattered in the environment and collect them to a central location.

Foraging is interesting to study for both its nature and its applications. It can be used to investigate the performance gain of cooperation in mobile robot systems quite easily. This is because foraging can be performed by each robot independently, by a group of robots working as individuals, and by a group of robots operating cooperatively. By looking at the difference between the average times to complete the task, it's fairly easy to compare time efficiency for the single robot, group of individuals, and cooperative cases. Also, the foraging task encompasses many applications that are of interest in current robotic designs such as cleanup, harvesting, search & rescue, and planetary exploration & sample retrieval.

There have been several different CMR systems created for the foraging task[1] and almost all of them use some inspiration or paradigm from biology to design their systems. This is not surprising since there is such a close parallel between the natural foraging task and the robotic one.

1.4 Bio-mimicry

However, many other areas of research are interested in trying to use the accumulated solutions in nature to solve problems. Materials engineering and bio-engineering, in particular, use the term biomimetic to mean a substance which has properties like a natural material but is not the same, or sometimes even close, chemically. Artificial Life research also uses this term (Meyer, 1997), and seems to mean by it any artificial construct inspired by a natural system. In behavior-based robotics research, designers often say they are "inspired by" a paradigm from a specific species or natural systems in general. Generally, this thesis will take bio-mimicry to be using a paradigm from a natural system to construct an appropriate parallel in the artificial world.

It is important to explore the nature of this translation. While following as exact a mapping from the natural to the artificial world as possible may yield a system with high fidelity of reproduction to the original, it may actually be too much to handle. The dynamics desired might be captured with a much simpler model of the natural system that is easier to understand. Control system theory uses this method often, modeling real non-linear systems with linear approximations.

Several foraging systems have tried to re-create the dynamics of an insect following a pheromone trail for navigation. One "low-fidelity" approach has been to use a chain of robots

[1] See (Cao, et. al., 1997) for a sample list

that form a beacon path for guiding other robots in the same way that the pheromone trail guides the insect (Werger & Mataric, 1996; Goss & Deneuborg, 1992). The opposite end of the spectrum is demonstrated by Kuwana, et. al. (1995), who attached the living antenna of a male silk moth to a mobile robot and demonstrated that it could follow a live female's pheromone trail. Somewhere in the middle lies the work of Russell, Thiel, and Mackay-Sim. (1994), who demonstrate a robot that follows a camphor trail by means of a gravimetric sensor crystal, which can measure the deposition of the volatile camphor molecules on its specially treated surface. These examples demonstrate the range of possibilities when translating a natural system into the artificial world.

Which one works the best? It will, of course, depend on the particular robot/task/environment trio it is implemented upon. However, it is somewhat misleading to speak of the "best" solution. The design space is infinite, and such global optimizations are incredibly difficult to conceive of, let alone actually find. Therefore it makes much more sense to speak of a solution which is "good enough" – one which gets the job done satisfactorily with a minimum of effort.

1.5 Statement of Thesis

This thesis develops a methodology for aiding the designer of behavior-based robotic systems by mimicking biology, called biomimetic design. It examines the issues involved in designing a robotic system with biomimetic methods, and presents an abstraction of the translation between the two worlds.

In order to accomplish this task three topics will be investigated. First, an abstraction of the biomimetic design process is developed. A set of principles and guidelines for biomimetic design will be proposed which will guide the design of the foraging system. Second, these principles will be applied to the design of a CMR system performing a foraging task. The biomimetic design method is used to find a natural system to emulate, determine the dynamics of its operation, and translate those principles into an actual robotic system. Finally, the task accomplishment of the design will be verified and characterized by a set of imposed performance standards.

1.6 Thesis Organization

Chapter Two elaborates on the use of bio-mimicry in the design of cooperative mobile robots. It surveys some of the work that has been done in designing biomimetic robots and draws principles of biomimetic design from the examples. An abstraction of the biomimetic design process is introduced, and a set of guiding principles for such design proposed. Together, these constitute a methodology for biomimetic design.

Chapter Three applies the method developed in Chapter Two to create a design for a cooperative mobile robot system solving a foraging task. The design is based upon mimicking the navigation and cooperation mechanisms of Saharan desert ants (genus *Cataglyphis*). The robotic platform consists of six modified TJ robots from Mekatronix, programmed using C language in subsumption architecture.

Chapter Four describes the fine-tuning and experimental validation of the design from Chapter Three. Individual robots are shown to be capable of navigating to and from their home base. Experiments with a group of six robots result in a large number of successfully navigated trips, but yields no actual target object retrievals. The successes and problems of the designed system are discussed.

Chapter Five is the conclusions of this thesis and recommendations for future work. Although the system did not perform to expectations its proposed that very simple modifications to the robot, task, and environment would yield substantial improvement. The problem areas are shown to have been predicted by the biomimetic design process and by other robotics research. The biomimetic design process is shown as a useful and promising tool.

Chapter 2 - Bio-mimicry in robotic design

Unimation produced the first servo-controlled robot in 1961. However, the servo-controlled robot to which we are accustomed is merely a type of automatic control system. One possible definition of a robot is an automatic controller moving some physical component in the real world. This type of system has been around for millennia, since the Egyptians first built water clocks and water powered figurines 5000 years ago (Hall and Hall, 1985). For hundreds of years before the invention of servomotors, automatic actuation was carried out by cams and gears using steam or water for power. With so much design history behind us, you would be excused for thinking that we would be well versed in the best methods of designing such machines. But this is not the case. There are several ways that robot design can be viewed, and these methods do not always agree on what is important in designing systems. This chapter creates a method based on bio-mimicry to help the robotic designer towards useful and practical systems.

2.1 Defining bio-mimicry

The concept of bio-mimicry was discussed briefly in the last chapter. But in order to use biomimetic tools in the design of robotic systems, we need to define what the process of bio-mimicry is and how it can be accomplished.

1. Problem definition $\Rightarrow$ Useful natural system to mimic
2. Natural system $\Rightarrow$ Model of the system's behavior
3. Model of natural system $\Rightarrow$ Model of robotic system
4. Model of robotic system $\Rightarrow$ Implementation

Figure 2.1. A set of transitions defining the process of biomimetic design.

Bio-mimcry is the process of taking inspiration, knowledge, or mechanisms from a natural system, which has some quality that we desire to emulate, to create an artificial system that has similar properties or dynamics. When bio-mimicry is defined this broadly it is easy to find examples of it in many common scientific and engineering fields. Everything from Astro-turf to neural networks are biomimetic when viewed in this manner. However, when used in this thesis, the term bio-mimicry will specifically mean a systemized and formal process of taking the dynamics from the natural world and translating them to the artificial one. This process can be captured in the transitions outlined in Figure 2.1.

Each of the transitions above is a step in the process of biomimetic design. To be successful however, each one of the transitions should really be a *translation* from one paradigm to the next. Each translation should have a level of fidelity sufficient to ensure that the useful properties of the beginning paradigm, problem definition, are present in the ending one, implementation. Insuring the fidelity of the resulting translations is, of course, the hard part.

2.2 Reasons to use bio-mimicry

There are two primary ways in which bio-mimicry can be useful. Sometimes a system is built merely to perform a task. On the other hand, it may also be built to increase knowledge and understanding about a certain type of system or effect.

Bio-mimicry for task accomplishment will be most familiar to the engineer. When one is given a complicated and often contradictory specification for a new product it can be difficult to turn this desire into a reality. Biomimetic design can provide guideposts for creating such a design. By properly using paradigms pulled from natural systems, the designer has the ability to find methods and concepts to accomplish the design specification. At a higher level of biomimetic fidelity, mechanisms used by various species might provide actual components for the design, whether these components are truly biological or merely artificial reproductions.

Research based reasons have a slightly different flavor. In pursuing knowledge about a particular aspect of biology, researchers will create models trying to describe the system. One of the best ways to test a model is to see if you can make predictions about the system's behavior using it. However, this may be very difficult to accomplish with members of a real species. The natural environment may be difficult to control, or the phenomena under investigation may be difficult to separate from other environmental variables. Perhaps the problem to be investigated requires in vivo studies, which may be hard to perform for ethical reasons. Thus, the creation of an artificial system that mimics the natural one under investigation can prove valuable for researchers.

Let us focus more closely on why bio-mimicry is useful in designing systems to accomplish a task. System design is never easy. Even for very simple problems, which are easily modeled by well-understood equations, creating a system can be a very difficult task. For robotics, where the systems are not well-understood or easily modeled, it can be very taxing and stressful to try and create a system to specifications. Traditionally, we have looked to classical methods of system design to create robotic systems. Robotic design encompasses dynamics, LTI control theory, state-space control, optimal control, and AI based methods. Successful robots have been created with all of these methods, and in fact these are still the bread and butter of most commercial robotic design today. However, all of these methods have a definitive bias towards viewing the object being designed as only the robot, and taking its task and environment to be simple, defined, well-modeled, and invariant with respect to time. The real environment is seldom any of those things, and task may prove similarly elusive to define.

Because natural systems exist in the same uncertain and dynamic world for which we wish to design robots, there might be something to learn in their solutions. That is not to say that biology has all the answers or that natural solutions are optimal. However, the designer cares more about whether the solutions work and are applicable to the design problem at hand. It is often useless to search for optimal solutions to a design problem because the state-space of possible solutions is so large. Additionally, the design specifications are usually under-constrained and there will be a large number of possible solutions. In a world like this, the designer must pick a set of constraints to impose on the problem in order to solve it. It makes as much sense to pick constraints based on a natural system as anything else. The designer then hopes that not only will using bio-mimicry constrain the problem adequately so it can be solved, but will also enable the system being designed to have some of the abilities to deal with a dynamic and uncertain world that are present in the natural system.

To better understand what bio-mimicry is, and how to use it as a design tool, it will be advantageous to see examples of its use.

2.3 Examples of bio-mimicry in designing robots

Biomimetic design has a long history in many fields, including robotics. This section will examine just a small sample of the many biomimetic robots. After mentioning two important early robots, this survey will examine several recent biomimetic robots, categorized by function of the system being mimicked, and briefly mention some of the lessons to be learned from each experience.

2.3.1 Early history of biomimetic robots

J. H. Hammond built the earliest known electrically actuated biomimetic robot sometime around 1915. Jacques Loeb, one of the most outspoken proponents for the mechanistic view of animal behavior, wrote about the robot as an example proof for his theories of animal behavior (Loeb, 1918). According to the text, Hammond developed an "artificial heliotropic machine", i.e. a light seeking robot. The device could exhibit both positive heliotropism[2] and negative heliotropism[3]. This is a clear case of bio-mimicry to further understanding of biology. By showing that a heliotropic behavior, similar that found in many lower animals, could be produced in a completely mechanistic system Hammond advanced the status of the then-new reflexive approach to animal behavior.

Another interesting example of an early biomimetic robot design is the work of Grey Walter in the early 1950's. Walter was a neurobiologist who was, like Loeb and Hammond, interested in how animal behavior is created. More specifically, he wanted to investigate how the interconnection of neural elements could generate complex, life-like behavior. Walter created a set of robots, known as tortoises, which displayed various phototropic[4] behaviors. The turtles were attracted to lights of moderate intensity, yet repulsed by those of higher intensity. They also had a small headlight on the front linked to their steering mechanism that enabled them to react to each other by means of their phototropism. Additionally, the robots were able to detect contact with an object and would enter a cycle of butting and withdrawal that would cause the robot to push aside small obstacles and eventually navigate around larger ones. These basic abilities, generated by the two-node neural structure designed by Walter, combined with the proper environment and the physical design of the robots yielded some startling behavior. The tortoises were able to demonstrate social behavior, "play" in front of a mirror, find and enter their hutches to recharge batteries, and make choices between similar alternatives (Walter, 1953). Most amazingly, this was done with no digital computation at all; these robots were completely analog

[2] Meaning that the agent is attracted to light and moves towards it.

[3] Meaning that the agent is repulsed by light and moves away from it.

[4] Different terminology then Loeb's heliotropism, but the same meaning.

electronics built individually and uniquely each time. In many respects they were like a real species, each member having their own quirks and abilities.

2.3.2 Mimicking physical abilities

In robotic design it is very common to mimic the physical abilities of a natural system, especially in the area of locomotion. In fact, many walking robots owe their designs to the field of gait analysis. Studies of the dynamic balancing methods used by various species have led to a host of walking machine designs[5]. These machines try to imitate the dynamics of a natural system as one *possible* means of providing support and mobility. Note that it is entirely possible to produce machines that are able to walk that have no parallel to any natural species. For instance, the Mars Micro-Rover consisted of two nested quadruped frames which could be lifted and translated independently (Ali, 1994), as seen in Figure 2.2. The robot walked well, but there is no species that has a locomotion mechanism even close to it.

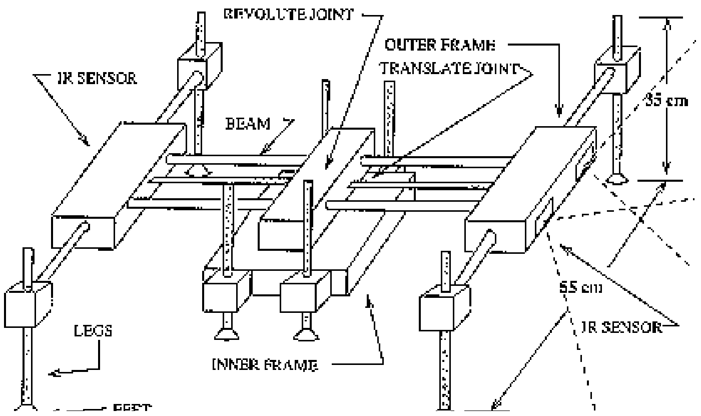


Figure 2.2. The physical design of the Mars Micro-Rover, which illustrates that useful designs do not have to come from biologic examples. From (Ali, 1996).

This is the first important principle of bio-mimicry for design: it does not generate unique or optimal solutions to a given task and environment set. Biomimetic design does, however, narrow down the range of possible solutions from the near-infinite to a more manageable set. As a bonus, these solutions are known to work given the correct set of agent, task, and environment. Thus, if the translation can correctly capture the dynamics of the natural system then the

[5] For example, see Espenscheid, et. al. (1994) and Playter (1994).

artificial design will also be able to complete its task reliably. This illustrates the importance of the translation and the how important it is to understand its nature, as was discussed in section 2.1.

The coupling of sensing and locomotion makes the problem more involved. A robot developed by Morse, Ferrée, and Lockery (1998), exhibits this kind of bio-mimicry by copying the chemotactic food-searching dynamics of a soil nematode. By closely examining the interaction of the nematode with its environment, they were able to characterize the nematode's food-searching dynamics as consisting of three essential parts:

1) The animal detects chemical gradients by using a point sensor located on its head and making temporal comparisons of the local concentration.
2) The direction of movement and rate of turn of the nematode is controlled by the angle of the head with respect to the body.
3) The nematode moves at a nearly constant speed during chemotaxis.

Morse, et. al. determined that these essential dynamics allowed them to construct a four-wheeled robot with Ackermann steering, constant velocity motors, and a single photocell that would have the same dynamics in phototaxis as the nematode did in chemotaxis.

How where they able to make such a drastic simplification and still have the system exhibit the same dynamics? The mechanistic view of animal behavior holds that sensing is tightly coupled to action. For an animal the perception of action leads directly to the action on the part of the robot. That is, it is important to study what an animal can perceive about its environment and how that perception triggers action (Ford, 1996; Duchon, Warren, & Kaelbling, 1998). By studying and properly describing that interaction, the solutions to mimicking the behavior may become obvious and simple. Thus, a second principle of biomimetic design is to examine closely the fit between the animal and its environment to find clues that will lead you to the basis of its behavior and how it can be re-created in the robot.

2.3.3 Mimicking intelligence and adaptability

Artificial Neural Networks [ANNs] are a particularly successful form of bio-mimicry. ANNs are models of biological neural networks, and like their biological counterparts have the ability to decompose complex information in parallel into basic elements. In this manner neural networks can react to complex input much faster then a normal control program. There are several different models of ANNs in use to mimic the activity of biological neural networks. However, the primary differences between these models are items such as the number of layers of the network, the learning algorithm, and how the neurons can interconnect. Almost every model shares the same concept of an individual neuron, as illustrated in Figure 2.3. The transfer function of the model is

$$O_i = F_i\left(\sum_{j=1}^{n} w_{ij}x_{ij}\right), \text{where the neuron's firing condition is } \sum_{j=1}^{n} w_{ij}x_{ij} \geq \Theta_i$$

and index *i* represents the neuron in question and index *j* represents the inputs from other neurons (Kartalopoulos, 1996).

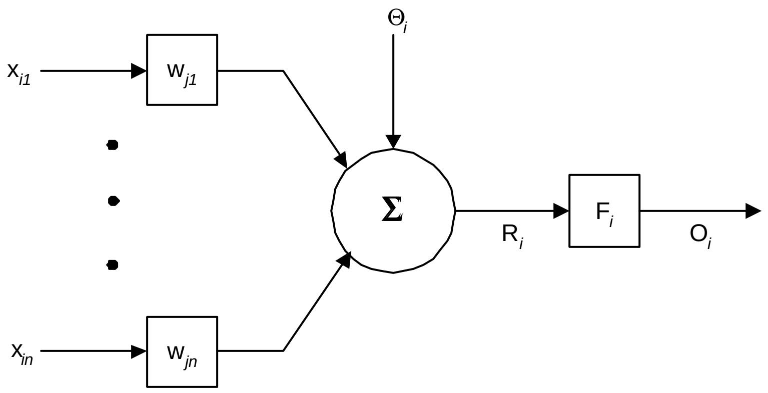


Figure 2.3. The basic model of an individual neuron.

This model has fundamental differences from the way a real biological neuron acts. Biological neurons do not output a steady value, instead their outputs come in the form of pulse trains of varying frequency. Additional complications of function come from the synaptic connections between neurons where the electrical pulses become chemical signals flowing across the gap. For instance, the synaptic efficacy changes as a result of the amount of activity, strength, and frequency of stimulation. Finally, many ANNs use very simple non-linearities to make analysis easier. One very popular non-linearity is a simple "on or off" threshold function which lacks all intensity information present in real neural structures. Only a very few attempts, such as (Beer, Chiel, & Sterling, 1990), have been made to create a more realistic neural model and apply it to a robot.

In spite of the many differences between ANNs and their biological counterparts, neural networks are a popular and useful solution to many difficult problems in control, signal processing, and data processing. In other words, even though ANNs don't operate at a very high level of fidelity they are useful models. Why then, are ANNs such loose translations? Because that is the level human understanding and technology can cope with. In order to build a more faithful reproduction one would need to either build a full-blown simulation of all the molecules involved or actually grow your own biological neurons. Therefore we end up with such low fidelity elements as threshold non-linearities. This is a third important principle of biomimetic systems: fidelity can only be as good as human understanding can make it. One must then build the system to find out if the level of fidelity possible is actually useful.

Another example of robotic bio-mimicry can be found in the design of various behavior-based control architectures. Behavior-based approaches to robotic control are believed to more closely approximate how animals make decisions and choose actions in their environment. The fields of ethology and environmental psychology both attempt to describe animal actions in terms of behaviors, which are generated by a releasing mechanism[6] or affordances[7] in the environment. Various control architectures have been proposed to make robots reactive to their environment in this way.

For instance, Arkin's schema architecture (1998) uses potential fields generated by various stimuli the robot recognizes in its environment. The shape, magnitude, and direction of these fields are dependent on the programming of the robot and the stimulus to which it is reacting. Then all the potential fields acting on the robot at any given time are summed together to yield the action the robot will actually take. This methodology makes the integration of a large number of behaviors very easy.

Connell's colony-style architecture (1990) also addresses the need to integrate a large number of functions and select the appropriate action. Brooks' original subsumption architecture design (1986) had a web of interlocking behaviors, which were difficult to add competencies to without modifying the already existing structure. Connell addressed this problem by eliminating direct communication between various behavior modules. That is, the only way in which various behaviors communicate is through changes they can mutually observe in the environment. This architecture also eliminates any concept of long-term state information. Thus, the robot is completely immersed in its environment and selects actions only by what it

[6] The term used in ethology for a set of stimuli that cause an animal to exhibit a certain behavior.

[7] The term used in environmental psychology for essentially the same idea.

currently knows about the world. It also enables the robot to be endlessly expandable without having to redesign the parts of the controller already in place.

These points bring us to a fourth, and final, principle of biomimetic robots: make sure that the components of the systems can be integrated into a whole. Unwieldy and poorly thought out methods can seriously hinder the modification and expansion of the system.

2.4 Biomimetic design

Drawing on the material in the sections above and in the previous chapter we need to create a notion of how a robotic system can be designed using bio-mimicry. In order to accomplish this, lets summarize what we've learned so far.

1) Bio-mimicry does not generate unique or optimal solutions to a given task and environment set. Biomimetic design does, however, narrow down the range of possible solutions from the near-infinite to a more manageable set. Hopefully, it will also lead to a set of solutions that have robust task accomplishment properties.
2) It's important to consider the fidelity with which a system is to be mimicked. Too much fidelity can lead to a design that is impossible to implement. After all, a perfect mimicking would be to completely recreate the species with just enough changes to perform the task. Too little fidelity can lead to a robotic system that does not exhibit the dynamics that were designed into it and is unable to accomplish the task set forth.
3) When performing a biomimetic design, closely examine the fit between the animal and its environment. This will lead to clues that will help you to find the basis of its behavior and how it can be re-created in a robot. By decomposing the system properly a solution may become obvious.
4) In designing a control architecture for a biomimietic robot care must be taken to ensure that the components of the systems can be integrated into a whole. Because the method is deconstructionist at its heart, reassembling the various design components, i.e. behaviors, into a functioning whole can be difficult if the proper care is not taken at an early stage.

2.4.1 Models of design used in robotics

There are three models of robotic design that we will find particularly useful to developing a biomimetic design method. There are, however, many other methodologies out there, based off of programming languages (Parker, 1995), a basic set of behaviors (Matariæ, 1993), and the like.

We have already noted that examining the environment and the task are just as necessary as designing the robot for a successful system. There are two existing models of robotic design which take this into account. Essentially, they have the same objective – integrating the process of robotic system design with the nature of the environment the robot will be operating in.

Kube and Zhang (1997) describe a method for modeling robotic tasks based upon Finite State Automata [FSA]. By extracting from the environment certain perceptual cues to trigger a task-accomplishing behavior in the robot, they can specify both the sensors needed to accomplish the task and the FSA controller needed to accomplish the task. In other words, by decomposing the task into an ordered series of steps they can determine what the characteristics of the environment's state should be at each step. Thus they can determine what set of cues should be used to trigger any particular step. Using an FSA, they can direct the triggering of any given step and the sequencing between them. This approach recognizes the basic interconnectedness of the robot and its environment and provides a tool for taking advantage of that interaction to accomplish a task.

Ford (1996) recognized an additional part of the puzzle. Not only are environment and the robot interconnected, but we can design either one. While Kube and Zhang's method is restricted to designing the robot given the environment it is in, Ford uses the environmental psychology method called Theory of Affordances to show how either robot or environment can be a design variable. Basically, as in the Kube and Zhang model, there are certain sets of affordances that can be used to trigger a task-accomplishing behavior. But these affordances are not just cues as before, they are also the properties of the environment that allow the robot to actually accomplish the task. For instance, an object that is rigid, horizontal, extended in size relative to a human's pelvis, and roughly knee high can be used as a seat. It does not matter what color, texture, or material it is made of, those properties do not change its suitability for use as a seat. Those properties afford a human sitting and those are the properties that a human extracts from the environment to find a seat. A robot and its environment can be designed by creating the match between the robot and the affordances of the environment, so that the affordances can be used to trigger task-accomplishing behavior.

2.4.2 Locating complexity in the system

This thesis will make an assumption that the designer is trying to create a system that will perform a useful task. Since this kind of design is task-specified, then the robot and its environment will be the primary design variables. In most system design problems the design task must be successfully completed or else the system is a failure. There will therefore be very little leeway in modifying the task. Depending upon the situation there may also be very little ability to modify the environment. However, in most cases the designer may have more ability to modify the environment then they realize. They will certainly have the ability to change the design of the robot. In both areas there is a complexity tradeoff to be made.

Given a task of a certain level of complexity, there will be a minimum amount of complexity that has to be present in the robot and environment in order to solve the task problem. That fixed amount of complexity can be distributed between the robot and the environment in almost any manner that the designer wishes. For instance, there are two possible approaches to solving a navigation task. A robot could be equipped with a complex vision system for recognizing landmarks. Alternatively, a set of barcodes could be placed at known locations in the environment and the robot could use a simple laser striper to read them and triangulate its position. In other words, complexity can either reside in designing the robot or in designing the environment.

Once a given amount of complexity is located in the robot, the designer can distribute it between either the hardware or software as they please. This second complexity tradeoff is present in all computer-controlled machinery. One may either put a great deal of the function in a complex mechanical design, or a more complicated control algorithm can be used. For instance, a hexapod walking robot traditionally has some multiple of six actuators, which require complex controllers to synchronize properly. However, one Colorado State entry into the 1999 Walking Machine Decathlon uses a mechanism that utilizes two actuators running parallelogram four-bar mechanism legs through a system of spur gears (Derringer, et. al., 1999). In this way, it relocates much of the complexity of hexapod gait from the controller to the mechanical design of the legs.

2.4.3 A method of biomimetic design

The biomimetic design method comes in two parts: process and principles. By following the process and applying the principles a system can be designed using this biomimetic design method. The method is described in Tables 2.1 and 2.2.

A few more comments should be made on applying the biomimetic design process. As mentioned previously, biomimetic design does not necessarily yield an optimal design. When working in such a design space it is futile to try to optimize the design. Instead, the designer should concentrate on making the design “good enough.” If the design fits all the reasonable performance criteria laid out in the problem definition then it is a successful design.

Using this formalized process may enable a designer to get more repeatable results with each design then has previously been the case. This is because design is the art of managing tradeoffs to get the desired outcome. As in the complexity trade-off issue, there is no right or wrong answer, but realizing that the ability is there can make an untenable situation solvable. Each individual designer must use their judgement so that the design falls into a form that they can deal with using their personal knowledge base. In the end, the final result will be as original as each designer. Therefore, the biomimetic design method can aid in obtaining useful systems more often.

Note that this method of design does not replace the traditional model of engineering design, such as illustrated in Figure 2.4. Rather, it provides a framework inside it, giving the designer guidance that might prove valuable in creating a workable and robust design.

Table 2.1. The principles of the biomimetic design method.

Principles	
I.	It is important to maintain enough fidelity in each translation that the dynamics desired are maintained, but not so much that it overwhelms the understanding and technology of the designer.
II.	An agent (robotic or natural), its task, and environment are an inseparable trio.
a.	When analyzing a system, decomposing the relationships between the agent/task/environment trio is critical to understanding its function.
b.	When designing a system, it is important to remember that any one of the trio can be designed, not just the robot.
III.	The design architecture must allow for the re-assembly of the deconstructed parts back into a useful and synergistic whole.
IV.	Complexity in the system can be traded-off between various components, such as robot and environment.

Table 2.2. The process of the biomimetic design method.

Procedure		
Step	comments	principles used
1) Define the problem	Try not to overconstrain the problem unnecessarily.	IIa.
2) Find a useful natural system to mimic	Ethology, ethnology, sociobiology, and environmental psychology are all useful fields to find information from. If more then one system fits the problem definition use the one which has been studied the most.	I, IIa.
3) Create a model of the natural system's behavior	Preferably this is merely a matter of finding such a model from the literature, and the designer won't have to do this from scratch.	I, IIa.
4) Translate the model of the natural system into a robotic model.	In this step principle I has to be carefully balanced against technical and economic feasibility.	I, IIb, III, IV
5) Implement the robotic model in a real system	Experience in this step will lead you to create more realistic models in the first place, reducing the number of times iteration will take place between steps 4 and 5.	I, IIb, III, IV
6) Analyze the robotic system and make sure it meets the problem specification sufficiently.	Iteration may be necessary back up to any step in the process if it fails to do so.	IIa.

In fact, steps two through five of the biomimetic design model become the synthesis step of the traditional design model, and steps one and six take their respective positions in problem definition and analysis.

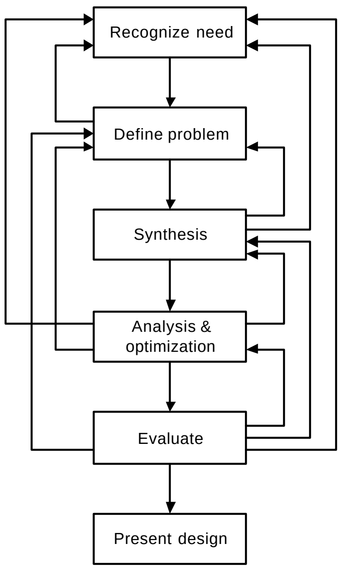


Figure 2.4. The traditional model of engineering design, which is complemented and not replaced by the biomimetic model of design presented here.

Similarly, we also wish to explicitly include a notion of hierarchy in the biomimetic design model. That is, robotic systems can be designed by dividing complex problems into functional sub-problems, solving those, and assembling the various solutions into a workable whole. This is a powerful design tool to deal with the limitations of human apprehension and the complexity of the designs that we are seeking. Simon (1998) expressed the idea of hierarchical composition in design most succinctly:

> The basic idea is that the several components in any complex system will perform particular subfunctions that contribute to the overall function... To design such a complex structure, one powerful technique is to discover viable ways of decomposing it into semi-independent components corresponding to its many functional parts. The design of each component can then be carried out with some degree of independence of the design of others, since each will affect the others largely through its function and independently of the details of the mechanisms that accomplish that function.

Finally, we are interested in why the biomimetic design model is particularly applicable to cooperative mobile robot systems. While it should be applicable to all robotic design tasks, bio-mimicry can be of particular use for the same reasons that CMR design is more difficult then single-robot design. CMR systems are even harder to model, predict, and diagnose then single robot systems because of the unpredictable interaction of its robotic members. Consequently, these systems have traditionally been classified as more difficult to design and less experience has been accrued in the area. It has already been indirectly explored why biomimetic design is good in an unknown design area – because it offers the designer guidance in an under-constrained design problem that they otherwise would not have had. Also, it has been noted before that members of various species seem to cooperate within their own family, clan, or species; and sometimes even with animals outside their species. By exploring the methods and reasons by which this cooperation occurs we hope to learn more about coordinating the workings of CMR system.

Chapter 3 – Biomimetic design of a CMR system

This chapter will apply the biomimetic design model developed in the previous chapter to create a real robotic system. By documenting not just the design, but the design process this thesis can: 1) vividly illustrate the usage of the design model, 2) evaluate the usefulness of the model for creating a real system, and 3) more completely document the design itself by including its design history.

3.1 Problem definition

Recall that robot, task, and environment are an inseparable trio; therefore the problem definition includes aspects of all three. It is easy to believe that problem definition in robot design is only the specification of the task. However, in most cases there are constraints upon aspects of the robot and its environment as well. It is important to explicitly recognize those at an early stage so they can be available to constrain and drive the design of the system. This section will describe the constraints that were chosen at the beginning of this design. In most design problems, constraints like these are typically driven by forces external to the designer, e.g. customer requirements or design specifications. In this case some of them are simply invented to create an interesting problem to solve.

Since the Department of Mechanical Engineering at Colorado State University already possessed a set of small mobile robots, they were chosen as the development platform. The robots used to for this project were six Talrik Juniors [TJs] from Mekatronix (Doty, 1999). Physically, they consist of an 18cm diameter wood base with holonomic steering. Two hobby servomotors “hacked” to rotate continuously are controlled with pulse width modulation to provide propulsion. The original sensor suite was two infrared [IR] transmitter/receiver pairs for obstacle avoidance and four bump sensors[8]. Permission was obtained to perform extensive modification of these robots.

[8] Located front center, front left, front right, and rear

The robotic platform is constrained to be homogeneous. That is, all robots will be as similar physically as is possible, and they will use the same program for control. Robots will not be differentiated in any way other then by name.

The task has already been defined to be foraging, but some additional constraint is needed. The environment will contain 10 target objects. Task performance will be quantified in terms of both the number of target objects retrieved in a given time and the ratio of successful returns to home base to the number of times a robot becomes lost.

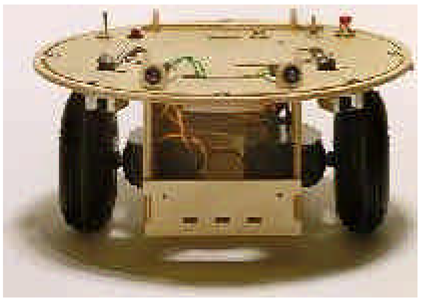


Figure 3.1. The Talrik Junior robot platform before modification.

The robots will have to be able to operate in environments which have both evenly distributed and tightly grouped target objects. In the tightly grouped case the target objects will be located in a single clump.

Knowing the general capabilities of the robotic platform can also be used to add some specifications to the environment. For instance, target objects to be collected will have to be in the 20mm to 70mm on-a-side size range for the robots to be able to handle them. They will be cubes so that orientation of the target object will not be part of the collection problem. It is also desirable that the robot positively manipulate the target objects, e.g. by a gripper, rather then simply pushing them around. This allows the focus to be on the foraging task, and not the dynamics of pushing objects around.

Any other object in the environment must necessarily be an obstacle. The physical platform is incapable of travelling over discontinuities of greater then 5mm in height, and yet cannot sense objects less then 80mm in height. Therefore, obstacles in that size range must not be present in the environment.

Finally, the speed and size of the robot can dictate the size of the environment so that the task might be completed in a reasonable amount of time. The environment will consist of an enclosed area of eight meters by eight meters, where the walls will be of a material that will enable the robot's reflected infrared sensors to detect them before a collision.

3.2 A useful natural system to emulate: Cataglyphis

Several species were examined to find one that might be suitable to emulate in the foraging situation defined above. Because of their relative simplicity of behavior, insects were the focus of this investigation. This was still too large an area to survey effectively. Therefore, the requirement for cooperation in the problem definition was used as a feature to further narrow the choices. The eusocial insect species are both renowned for their cooperation and have been studied extensively. This group includes bees, ants, termites, and wasps.

Originally, interest focused on the insects best studied and most famous for their social order – honeybees. Their methods of communication and cooperation are now well understood. Honeybees communicate explicitly by a dance language that has been successfully translated (von Frisch, 1967). They cooperate by recruiting other individuals once they've returned to the nest from a successful foraging trip. A returning worker will perform a dance that communicates the location of the food source they just found. Unoccupied workers who witness this dance will know the location of the food source and be able to exploit it themselves.

Bees also have a very useful navigation system that enables them to make forays in the kilometer range and still be able to find the nest (Seeley, 1995). They can see polarization patterns in the air and use them to form a light compass. That is, using the polarization pattern in the air, the bee can determine its compass angle to the sun. Thus, as long as the foraging trip is short compared to the progression of the sun across the sky, the bees have an accurate method for determining their heading. They can then use optic flow, the apparent speed of objects passing them, to determine the distance they travel. To increase their accuracy they use landmarks to fix their position.

Since honeybees communicate explicitly in order to cooperate, any robot mimicking them would have to do so as well. There were two options for communicating with the robotic system specified in the problem definition: IR modulation or the addition of a serial radio frequency link between robots. However, this soon proved infeasible with the time and funding available. Unfortunately, even though a large chunk of development on the robot's navigation system had been performed, it was not feasible to mimic the cooperation of honeybees.

Serendipity is a valid part of the design process. Luckily, one of the original papers that was pointed out as a resource on navigation using the polarized light compass was actually on a type of ant that also uses the same mechanism (Wehner, Michel, & Antonsen, 1996). Saharan desert ants, of the genus *Cataglyphis*, have essentially the same method of navigation as bees. They utilize a polarized light compass and optic flow to determine their position approximately and then increase their certainty using learned landmarks. *Cataglyphis* is unlike other ants in that it does not use pheromone trails for navigation or recruitment for food sources, but relies solely on the navigation method outlined above.

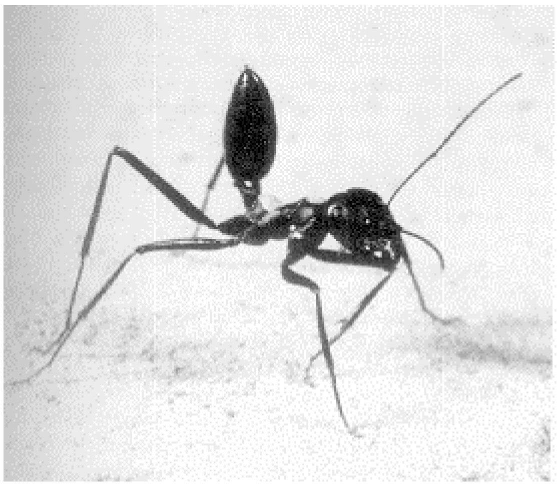


Figure 3.2. A forager of the species Cataglyphis bicolor.

Members of this genus are solitary foragers who do not appear to explicitly communicate the location of food sources among themselves as honeybees do. Rather, the colony uses a stigmergic communication mechanism to distribute its resources (Hölldobler and Wilson, 1990). Individual ants tend to forage in the same radial direction from the nest as their last trip if it was successful. The ant will tend to abandon that direction and search in a new one with a probability that is inversely proportional to the number of previously successful foraging trips in that direction. Thus, when food is concentrated in the environment the ants will focus on more productive sites. When food is more evenly distributed the ants will disperse their efforts to a corresponding degree. There is enough "noise" in the repetitive journeys that unpredictable deviations occur, allowing the ant to find food items previously missed by its foraging trip to the same area. In this way, even though each ant will probably never vary outside of a 40 degree arc in its individual foraging trips, the colony will cooperatively redistribute foraging trips to concentrate its efforts where it will yield the most effect.

Because this stigmergic method of cooperation is possible without modification to the robots, and it retained all the same navigation facilities that had already been developed, the desert ant was selected as the species to emulate for this robotic system.

3.3 A model of Cataglyphis foraging

Saharan desert ants can search tens of meters from the nest, find a food item, and return it to the nest quite accurately. To accomplish this they use a five-part strategy to: 1) select a foraging direction, 2) search for a food item, 3) navigate back to the vicinity of the nest, 4) search for the nest, and 5) recognize and enter the nest.

An individual ant makes its selection of a foraging direction based upon previous experience. One possible model was proposed in the section above. That is, ants might tend to forage in the same direction as the last trip and only abandon it and search in a new one with a probability that is inversely proportional to the number of previously successful foraging trips in that direction. Another possible model is that the search direction should be the mean of all previous radials from the nest upon which food has been found (Harkness and Maroudness, 1985). Note that in both cases this direction is simply the mean direction of the search as the ant leaves the nest. The real search path will be more complex then simply following a radial away from the nest. In the special case where the individual ant has no experience, i.e. its first foraging trip, then it should select its initial direction at random.

The search for a food item is carried out in the mean direction of the initial search as selected above. It has been proposed that an ant performs small random heading changes while following the mean initial search direction. Alternatively, Harkness and Maroudas (1985) propose that the ant simply tries to follow the initial search direction and go the same distance as its previous find. At this point it enters a random-walk type search pattern.

In the same paper, Harkness and Maroudas also propose that an ant’s search strategy would try to keep it within a certain distance of the nest. That is, in order to minimize time and energy spent searching a certain amount of area it makes sense to limit the maximum radius from the nest that the ant will achieve. In this manner, searching a wider and shorter area can be more time efficient then searching a narrow path around a long radial. This is because the return time after finding the food will be shorter. They propose that the density of search should fall off approximately as a Gaussian exponential from the nest. This algorithm causes the ants to self-limit the distance of their expeditions from the nest and concentrates their search closer to the nest area.

There is another safety device built into the search algorithm to prevent the individual ant from wasting time and energy. After approximately 30 minutes of unsuccessful searching an ant will return to the nest empty-handed. However, only about five percent of foraging trips end up in this manner.

After completing a search in one way or another, the ant must find its way back to the general area of the nest. During the outbound leg, the ant has been integrating the path and remembering the direction and distance back to the nest. Note that the ant is not keeping a large metric map of everywhere it has been. It has only been keeping track of the vector pointing back to its nest. Müller and Wehner (1988) suggest that the ant is not even performing a complete vector summation, but is actually using a step-wise approximation that consistently yields an error that actually helps the ant find its nest. At each successive step *n*, the ant calculates its direction, *ö*$_n$, and distance, *d*$_n$, from the nest in a way that can be modeled by the equations:

$$\boldsymbol{j}_n = \boldsymbol{j}_{n-1} + k\frac{(180^\circ - \boldsymbol{d})(180^\circ + \boldsymbol{d})\boldsymbol{d}}{l_n}$$

$$d_n = d_{n-1} + 1 - \frac{\boldsymbol{d}}{90^\circ}$$

where *ä* is the angle between the ant's current heading and *ö*$_n$, and *k* is a fitted constant to force the fraction term to a maximum of 90 degrees equal to 4.009 x 10^{-5} deg^{-2}. These relationships are illustrated in Figure 3.3a.

Because the algorithm is only an approximation, it consistently generates an error angle in the direction back to the nest. However, if the ant makes roughly as many right turns as left then the error will tend to cancel out. If this is not the case, then the error will be proportional to the angles steered in the outbound search. This phenomenon, illustrated in Figure 3.3b, will actually help the ant because it will return onto the outbound path from the nest. Thus, the ant may use landmark recognition in the familiar area to pinpoint the nest exactly and not have to risk overshooting the other way and missing any familiar area at all.

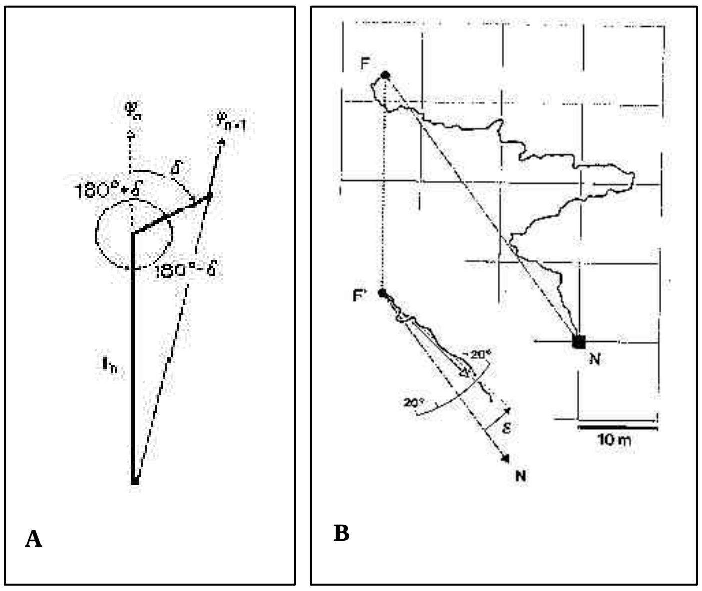


Figure 3.3. (**A**) A geometric representation of the ant's approximation to path integration. The angle from the nest is *ö*, the distance from the nest is *l*, and the angle between the ant's heading and *ö* is *ä*. (**B**) The natural foraging path of an ant showing the error angle generated by the path integration approximation system. N is the nest and F is the location of the food item found. Both figures from (Müller and Wehner, 1988).

This algorithm is not sufficient to get the ant exactly to its nest in a reliable fashion. Therefore, the ant needs a search method that enables it to find the nest site assuming that its path integration scheme has functioned well enough to bring it into the correct general vicinity. At this point, the ant initiates an outward spiral search pattern (Müller and Wehner, 1994). The ant now assumes that the start of the spiral is the tip of its home vector. It begins an outward spiral, i.e. one where the angle between the ant's current path and its home vector is greater then 90 degrees, and continues to use its path integration system to keep track of the spiral's origin. Periodically, the ant resets its home vector by walking back to where it believes the spiral started. Because the path integration scheme is not perfect it will not actually reach the place where it started the spiral. This imperfection also ends up causing the ant to execute a series of creeping loops rather then the spirals it believes it is making. Just as it was with the path integration algorithm, this error actually makes the method more robust. The errors cause the ant to traverse the same territory more then once, increasing its chance of finding home base if the ant if in the right vicinity. The ant will also continue to search most intensely around the area where it thinks the nest is, while slowly expanding the coverage of the search.

The following steps describe the spiral algorithm:

1) The ant selects the value of ä$_r$, the angle between ö$_n$ and the ant's current heading. Selecting ä$_r$ > 90 degrees defines the first segment of the outward spiral.
2) At a certain ratio of radial distance to path length, d/l, the ant performs the reset maneuver by marching inwards a distance, d_r, towards the origin. The critical value of d/l is on the order of one-tenth. After such a manuever the value of ä$_r$ may change. In general, the value of ä$_r$ increases after each reset.
3) After a certain period of time, t_r, the ant may switch the spiral direction but not its magnitude. This value appears to be on the order of tens of minutes.

Finally, the ant uses vision to recognize its nest and other landmarks. It uses this ability to help eliminate navigation errors over long distances and to perform the fine navigation necessary to enter the small nest hole. The ant does not need any feature extraction or other complex visual processing algorithms to accomplish this task. Instead it uses an unprocessed, two-dimensional, "visual snapshot" of the scene around the goal to recognize it (Wehner, Michel, & Antonsen, 1996). It is well known that the visual systems of many species often use these sorts of retinotopic representations in the processing and storage of images. For instance, hypercolumns have been observed in the primary visual cortex of a monkey, where adjacent one millimeter square areas become activated when adjacent areas of the retina are stimulated (Goldstein, 1996). In the ant, this stored retinotopic image is compared against the current visual scene and if it is similar enough the ant moves in such a way as to reduce the discrepancy between the current image and the stored visual snapshot. *Cataglyphis* uses this visual navigation system both to form landmark-based paths to aid in navigation, and as a way to find the nest entrance.

On its first few foraging trips the ant knows nothing of its surroundings. As it starts to become familiar with the landmarks in its preferred search area, the ant will be able to use them to fix its location. For instance, if an ant is taken from its nest and deposited in an area with familiar landmarks, it will begin to find its way back to the nest even though its path integration scheme has no vector to provide for navigation. The ant will perform an extensive spiral search until it accidentally encounters a familiar landmark. Then the ant will follow its previously learned landmark-based route home. Wehner, et. al. (1996) suggest that the ant has simply learned to keep certain landmarks on one side of its body while travelling home, or it might have stored a set of snapshots along the route where each one triggers a certain response with no need to have an expected sequence between them.

Even on its first foraging trip the ant must be capable of finding its nest site. The ant takes a visual snapshot at the start of each foraging trip, perhaps reinforcing its previous images if it has any. When it depletes its home vector during path integration, it will enter into the spiral search pattern described above. While searching, the ant is constantly trying to match its nest snapshot with the current visual scene. This matching can be fairly sophisticated since the ant is able to recognize its home irrespective of direction it is viewed from. Once the match between the visual scene and the remembered snapshot is sufficient, the ant will move in such a way as to reduce the remaining discrepancies between the two.

While the model above cannot completely explain the navigational behavior of *Cataglyphis*, it does provide enough information to construct a working robotic model. Unfortunately, it is impossible to know how faithful this model is to the real ant's behavior even though it represents the bulk of the current thinking about a very well studied species. It does describe certain aspects of ant navigation very completely, such as the approximate path integration scheme. But in other areas, such as the use of visual information for landmark recognition the model is still very skeletal.

In summary, there are six algorithms that can be used to describe components of *Cataglyphis* foraging behavior.

1) Search direction selection – The ant picks a radial direction from the nest to search for food based upon one of two algorithms. Either it picks (a) the mean direction of previous successful trips, or (b) the direction of the last trip if successful and otherwise pick an adjacent direction with some probability inversely proportional to the number of successful trips in the last direction.
2) Search path – The distance of the ant from the nest is proportional to the square root of time. The path is modeled by going the distance of the last successful foraging trip in relatively straight line and then executing a random walk search.
3) Navigation – The ant uses an approximation to path integration, using a compass sense, visual flow to measure distance traveled, and a vector memory to keep track of direction and distance to the nest.

4) Landmark paths – *Cataglyphis* uses of visual landmarks to create a path that the ant can follow home, even if its path integration home vector is zero. This scheme is based upon keeping certain landmarks on one side of the ant's body.
5) Spiral search – When the ant returns to where it's path integration system believes home is located, it will initiate an expanding spiral search for the nest location. The spiral turns into a series of loops due to the inaccuracies of the ant's path integration approximation.
6) Recognizing home – To find the nest entrance, the ant tries to move in such a way as to minimize the differences between the current visual scene and a stored retinotopic snapshot of the area around the nest.

3.4 A robotic model of Cataglyphis-like foraging

Each of the aspects of the natural model must be purposefully translated into the model of the CMR system being built. This does not mean that every feature present in the model described in section 3.3 needs to exist in the robotic model. One possibility of translation is to drop a feature entirely. Another possibility is to substitute a different feature, which serves a similar function, in place of the actual one in the natural model. Finally, one may also add features to the robotic model that did not exist in the natural one. Unfortunately, any of these methods might end up modifying the dynamics of the original system you desire to emulate. Thus, it is always wise to be able to test your system and have it be easily reconfigurable to enable the design iteration that will have to take place.

This system will be built in a subsumption architecture where there is no communication between the behaviors (Connell, 1990). Recall biomimetic design principle III, which states that the design architecture must allow for the reassembly of the deconstructed parts into a useful and synergistic whole. At this point in the process, the designer is faced with putting together those separate bits of algorithm into something that functions as a coherent whole – a central principle in Simon's (1998) hierarchical composition of design. The colony-style architecture, developed by Connell, enables the recombination of behaviors and the addition of new ones without having to redesign the behaviors already in place.

The robot can be decomposed into two separate physical systems that need not communicate directly with each other. We can therefore generate a Connell-type architecture by separating the locomotion and gripper control parts of the robot design. That is, those behaviors that control the motion of the robot need not interact with those that control the manipulation of target objects except through the environment they mutually perceive.

Before translating the model of *Cataglyphis* foraging into a robotic system, we will need to add some features that enable the robots to survive in a dynamic environment. Because of their small size and physiology, there is not much terrain in the world that is truly an obstacle to an ant. Ants even crawl over each other, especially in dense traffic situations like inside the nest. Unfortunately, the robotic platform used in this project does not have those abilities. The robot design must begin with a set of components that will enable the robot to work in its environment without smashing into objects, and also allow it to extricate itself from situations that impede its mobility. Thus, we define the following two behaviors, which are NOT based upon the ant *Cataglyphis*.

Behavior: Avoid

Avoid uses the return from the infrared object sensors to move the robot away from objects that are too close. The robot possesses two such sensors, mounted on the front and pointing about 30 degrees left and right of centerline. If an object is close to the left side of the robot then the left sensor will experience a high return value, while the right sensor has a low value. If the speed of the right wheel is reduced from +100%, full forward, to -100%, full reverse, in proportion to the return of the left sensor, the robot will turn away from the object on its left in proportion to the nearness of the object. To generate a smoother avoidance path, the motion of a wheel is retarded in proportion to the square of the return of the opposite sensor. This is similar in conception to Braitenburg's Vehicles 2a and 3b (1984).

Behavior: Disengage

If the robot bumps another object it will activate one of the contact sensors. If the contact is on the front of the robot, it will back away for a short distance, turn to a new heading, and try again. In this way, the robot can avoid getting hung up on a stationary obstacle or another robot that was missed by **Avoid**.

Since physical contact is more immediate then an object a short distance away, it makes sense for **Disengage** to have a higher priority then **Avoid**. Therefore, the survival section of the locomotor program is defined as in Figure 3.4, where **Disengage** subsumes the output of **Avoid** if activated.

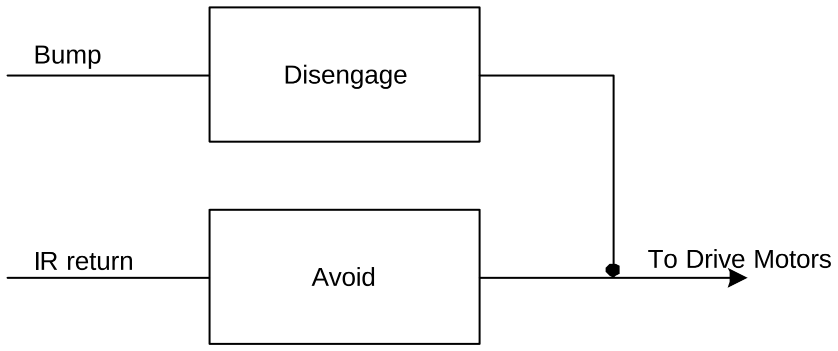


Figure 3.4. The robot survival section of the locomotion control architecture. The round connection between **Disengage** and **Avoid** indicates that the output of **Disengage** subsumes, or replaces, the output of **Avoid**.

Next, the components of the ant's foraging behavior need to be translated into the robotic model. The previous section defined six components that together make up a desert ant's foraging behavior. Because of various implementation constraints, this design will not use those components that require visual recognition of objects, such as landmark recognition. However, the rest of the ant's navigation system will be translated.

The path integration scheme discussed earlier is vital to the functioning of the *Cataglyphis* style navigation system. In order to use this scheme the robots will need some sort of compass sense. A digital magnetic compass with eight points of resolution[9] and a distance measurement system based upon speed commands to the motor are employed to solve that problem. Further details of these systems can be found in section 3.5.

In subsumption architecture, it is logical to break down a sequential task by allowing the first goal-driven behavior to be lowest in the architecture. Then each successive goal-driven behavior will be higher in the architecture and subsume the previous one when the correct preconditions are met. The foraging task is sequential in nature, and composed of the following goal-driven behaviors.

[9] That is, it can discriminate between N, NW, W, etc.

Behavior: Search

First, **Search** selects the radial direction from the nest to initiate the search. This decision is random on the first search performed by the robot on each run. However on successive searches the robot will chose the previous direction if foraging was successful, or choose the previous direction or one of the two adjacent directions with equal probability. That is, if the previous direction N did not yield a target object then the robot will choose NW, N, or NE with equal probability. This behavior therefore contains the stigmergic cooperation mechanism for the CMR system.

While searching for the target objects the robot will initially steer the radial direction selected above. After a random number of steps it will turn either 90 or 135 degrees in either direction. It will continue on this course until the maximum amount of time allowed for searching is reached.

Behavior: Homing

When the robot either finds a target object or exceeds its timeout limit it will proceed back to home. The robot does this by following the direction of the home vector stored in memory.

Behavior: Spiral

Once the length of the home vector becomes zero, then the robot will enter into an expanding spiral search pattern trying to find its home. Steering a heading slightly more then 90 degrees off from the vector towards home creates the expanding spiral.

Under certain conditions the robot will modify the spiral. When a critical distance to path length ratio is exceeded then the robot returns to where it believes home to be located. On those resets the robot may also reverse the direction of its spiral.

When these behaviors are set up in a subsumption architecture, where each successive goal-driven behavior subsumes the previous ones, they can successfully execute a task requiring the sequential accomplishment of those sub-goals. These behaviors make up the foraging section of the drive motor control program, described in Figure 3.5.

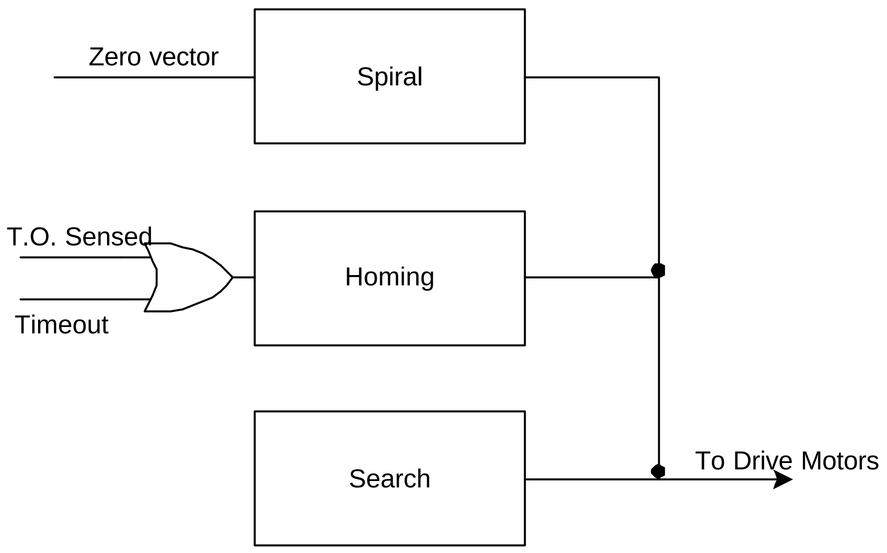


Figure 3.5. The foraging section of the locomotion control architecture.

Note that the navigation capabilities of the robotic model are not the same as the natural one. There was no attempt to add the ability to navigate known paths by landmark as the ant does. This sort of visual processing would be impractical on the computational resources available on board. It is likely that this exclusion will make it harder, but not impossible for the robot to perform its task. When a *Cataglyphis* forager emerges from the nest for its first trip, it does not posses any landmark path knowledge either. This path knowledge is something that is acquired only by experience, and in this case the system designed is actually mimicking the capabilities of a new forager.

So far, there has been no description in any of these behaviors of the actual updating of the home vector used in the navigation algorithms. Because this is a continual process, which needs to run regardless of what behavior is active at any given point, updating the home vector is modeled as a process separate of the behavior architecture. The process can be described as a very simple finite state automata, as illustrated in Figure 3.6.

A separate architecture for controlling the gripper mechanism of the robot is needed. To be faithful to the Connell-style architecture there should be no direct communication between the locomotion control and gripper control architectures. All communication between the different architectures must occur through perceptual cues in the environment; e.g. through the presence or absence of a target object in the gripper.

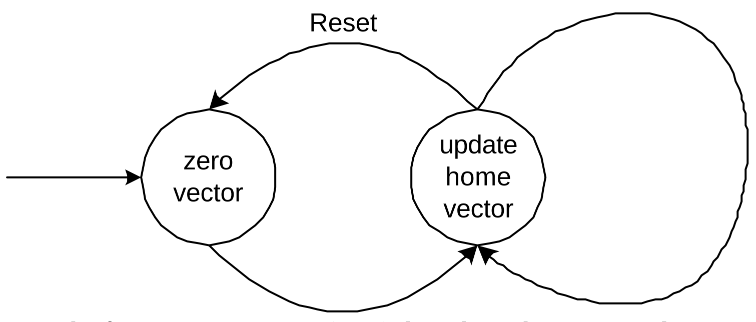


Figure 3.6. The FSA describing the vector update process. The state zero vector sets the home vectors length to zero. The update home vector state uses the current heading, speed, in an algorithm, e.g. the *Cataglyphis* approximation to path integration, that calculates the next home vector.

The gripper control architecture must be able to sense two distinct occurrences in the environment, the presence of a target object in the gripper and the arrival of the robot at its home. The exact design of both sensors is left to the next section, but a short overview is used here to design the gripper control architecture.

The ant model did not include any specific mention of food detection. No research was found that had a specific formal model, but it is easy to guess that such detection might come from scent or visual stimuli. Harkness and Maroudas (1985) state, from informal observations in the field, that detection range for *Cataglyphis* is around 10cm. Note that in the locomotion architecture there was no facility for the robot to approach a target object detected at a distance. It would not be hard, however, to insert another behavior into the foraging section of the locomotion architecture to enable the proper response. In this case, it was decided early on to leave the detection of target objects to tactile range only, in order to simplify the implementation process and avoid complicated sensor arrays.

The home sensor is also a radical departure from the model of the ant's behavior. Rather then using a real visual landmark recognition ability, the robot recognizes home by the amount of infrared light the floor reflects. This sensor is activated in a binary fashion, either showing that the robot is or is not at home. Again, there was no facility designed into the locomotion architecture to drive the robot towards its home from a distance. As with target object sensing, this could be easily added to the locomotion design, even though it is currently designed to operate at a tactile range only.

The gripper has only two actions it should take: close when a target object is present and open when the robot reaches home. These are obvious choices for behaviors. Adding a "don't move the gripper" behavior will enable us to trigger off of perceptual cues that are transient.

Behavior: Steady

Keeps the gripper at its current position. However, in order for the robot to be able to pick up a target object on its first run, the gripper must be initialized to the open position before the control program begins.

Behavior: Grip

When the robot detects a target object the gripper picks it up.

Behavior: Drop

When the robot detects that it is at home it releases the target object.

With **Steady** as the base behavior, the gripper system will still operate even when the signals for target object detection and being at home are not steady. The resulting gripper control architecture can be seen in Figure 3.7.

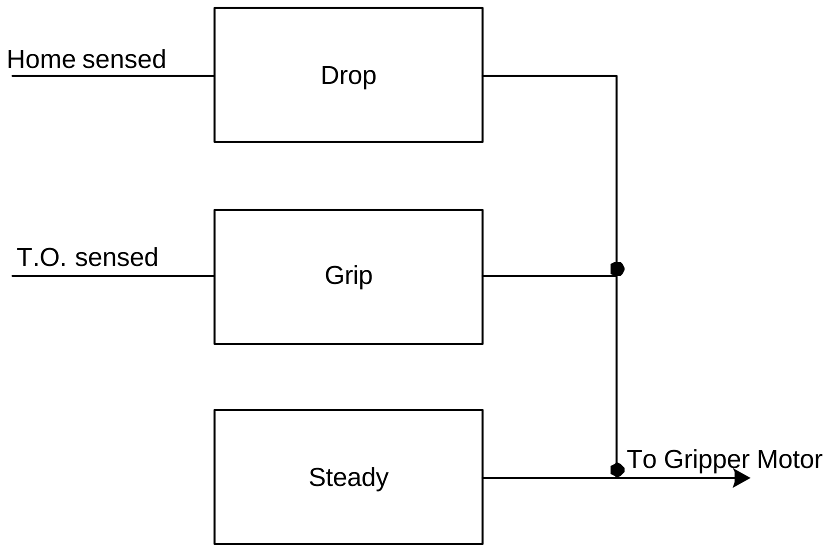


Figure 3.7. The gripper control architecture.

The various parts of the robotic model need to be combined into a unified and functional whole. The foraging and survival sections of the locomotion control architecture can be unified by having the output of the survival section subsume the output from foraging. This subsumption is vital to the continued functionality of the robot since it prevents it from pursuing potentially dangerous strategies while trying to accomplish its foraging mission. To enable the robot to leave its home without disturbing the collection of target objects, it should back out of home base before heading for the next foraging trip. This is similar to the **Disengage** behavior, therefore it is reused by providing an extra trigger to it when the robot senses that it is at home. The FSA for updating the home vector will run as a separate process whose value will be available for the rest of the control system to read. Several other continuous sensor processes, such as IR return, target object sensors, and the like will also run as continuous processes whose values are available to the rest of the system. This is the completed robotic model of the system, and is depicted schematically in Figure 3.8.

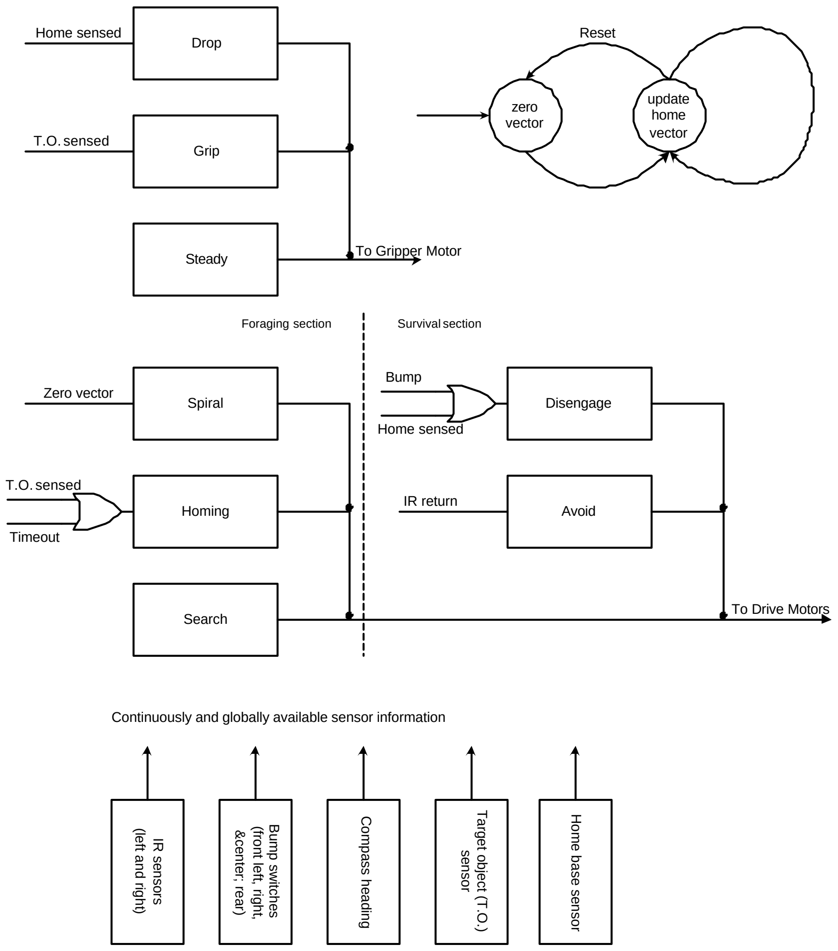


Figure 3.8. The complete robot control model. This model is a Connell-style subsumption architecture. All operations should happen in parallel, with negligible latency between a change in the environment and the change in the output of the model.

3.5 Implementation of the robotic system, task, and environment

"He either fears his fate too much,
Or his deserts are small,
That dares not put it to the touch
To gain or lose it all." - The Marquis of Montrose (1612-1650).

Implementation is the only real test. Building robots in the real world is much harder then simulation due to the world's inherent "messiness" – its non-linearity, time variance, uncertainty, and breakdowns. This is because the true functioning of a design comes from the interaction of the robot, its task, and its real environment. This section presents the implementation of all three for the robotic model developed in the previous section.

The TJ robotic platform was deemed insufficient for the task assigned without undergoing major modification. It needed upgrades in three areas: computational power, actuation, and sensing. The original controller possessed only 2K of program memory. The robot also had no means of manipulating a target object. Finally, the sensor suite did not have the capabilities of recognizing target objects, home base, or the direction of robot travel.

The TJ robots originally had a controller board based on a Motorola MC68HC811E2, an 8-bit HCMOS single chip microcontroller [MCU]. The MC68HC811E2 has 2 Kbytes of EEPROM and 256 bytes of RAM on the chip. It was thought that this might not be enough memory for a program complex enough for the task. In particular, it was desired that the robot be able to multi-task so that the subsumption architecture could actually function in a parallel fashion as it should in theory. Therefore, the original controllers were replaced with another Mekatronix product, the MTJPRO board. This board is based on a Motorola MC68HC11A8 MCU, which has 512 Kbytes EEPROM, 8 Kbytes ROM, 256 bytes RAM on the chip. This board also has an additional 32 Kbytes of RAM that the chip can address when operating in expanded mode. The off-chip RAM is interfaced through PORT B and PORT C of the MCU. The controller therefore looses those ports as general inputs and outputs, but the resulting increase in memory is worth the loss of I/O capability for this project.

In order to fulfill the robotic model of the previous section, the robot also needs the ability to manipulate a target object. Given the size of the robot, there is not much room for a typical gripper arrangement. Also, anything protruding in front of the bump ring would destroy its capability to determine frontal collisions. Yet if there were no gripper then the robot would have to rely on being able to maintain positive contact on the target object while pushing it. This would be a large risk, so a design was created for a simple gripper using only a single hobby servo for actuation. The full design and conversion process can be found in Appendix A. This gripper can pick up cubic objects in the 20mm-70mm on-a-side size range and rotate them slightly to clear them off the ground. It does not protrude beyond the bump ring and also protects the wheels from any possible entanglements while the gripper is open. The design can be seen in Figure 3.9.

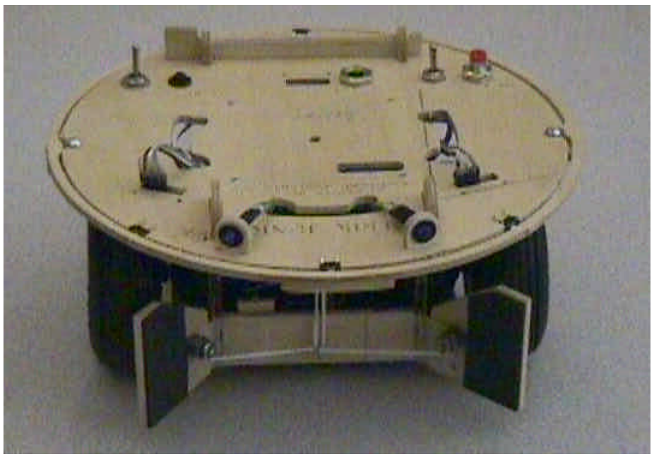


Figure 3.9. A TJ modified with a gripper. The design was developed by Jonathon Hurst, an undergraduate assistant on this project, in conjunction with the author. The gripper design and conversion process is located in Appendix A.

There are three ways in which the original platform was deficient in sensing for the required task. It was unable to sense the direction the robot was heading, unable to differentiate when the robot had a target object in the gripper, and it couldn't tell when it was at home base. To correct this, three sensor systems were designed and constructed.

Cataglyphis compass sense is based on polarization patterns of light in the sky and the position of the sun. To implement this sort of sensing would require the development of a custom sensor and restrict the robot to operate outdoors during the daylight hours only. On the other hand, there are other ways to discern the heading of an agent, such as magnetic compass, external beacons, and wheel-encoder information. The Dinsmore No. 1490 digital magnetic compass was the lowest cost option that could perform as required. The No. 1490 is small, light, and draws little power. It operates in the proper voltage range to be interfaced directly to the controller board. The compass is composed of four Hall Effect sensors placed 90 degrees apart around a regular compass needle. As the needle rotates it activates the various Hall Effect sensors. Combinations of the activation of the four sensors give eight cardinal points on the compass rose for resolution. That is, the No. 1490 can differentiate between North, Northwest,

West, etc., for a resolution of roughly 45 degrees. This compass is mounted on a specially designed board that provides easy connection and the necessary ancillary components. The compass specifications and board design can be found in Appendix B.

The design of the target object and home sensors was simplified by being able to control the environment closely. Recall that the environment is a valid part of the design if not constrained in the problem definition. Therefore, the problem can be solved by designing a simple sensor and then creating an environment to match it. By structuring the environment in this way some of the complexity going that would otherwise reside in the design of the robot is transferred to the environment.

In the last section, the sensor signals in question were defined to be binary values, indicating either the presence or absence of the desired object. The simplest sensor that will work reliably is one where the target object makes creates a contact between a high logic voltage rail and a sense rail. The sense rail has a pull-down to ground, and thus will be low unless a target object closes the contact across the rails. This design defines a target object to be any object that is: (a) lower then the height of the bump ring, (b) will fit in the gripper, (c) high enough to force the contact closed, (d) heavy enough to force the contact closed, and (e) is light enough to be picked up by the gripper. At this point, the form of the target objects can be picked: wood cubes, 45 mm on a side. An earlier version of this sensor depended on the cubes being conductive, and therefore the ones used in the tests of Chapter Four were unnecessarily covered with aluminum foil. The schematic design of the target object sensor can be found in Figure 3.10.

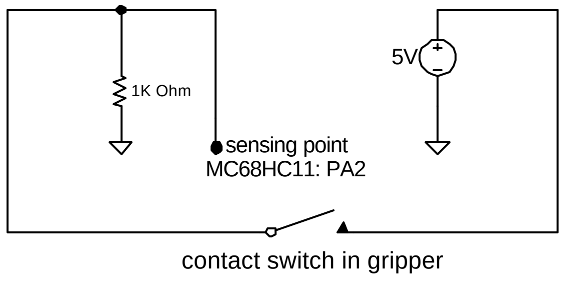


Figure 3.10. The schematic representation of the target object (T.O.) sensor.

Like the target object sensor, the home sensor should also be robust in operation and simple in design. In this case, a commercially available reflective infrared sensor was employed.

The sensor is pointed downwards at the floor. When the robot rolls over a highly reflective material the sensor will detect it. Thus, the home base area can be defined by aluminum tape on the floor.

The software to implement the robot control architecture, described in the previous section, was written in ICC version 5.1– a subset of the ANSI C language for the 68HC11 family of processors. Ideally each behavior would be a thread on a multi-tasking system. While ICC does have a simple round-robin style multi-tasker, attempts to integrate it with the timing functions needed for the navigation algorithms failed repeatedly. Therefore, a serial processing approximation to the subsumption architecture was developed. Pseudo-code for the control program can be found in Table 3.1.

Table 3.1. Pseudo-code for the behavior based robot control architecture. Each behavior may actually be a separate function called by the main program.

```
declare and initialize variables
declare functions
initialize communication with servos and sensors
loop forever
    if bump_signal or home_sensed or disengage_latency then
        left_motor_command, right_motor_command, disengage_latency ← behavior disengage
    else if IR_return or avoid_latency then
        left_motor_command, right_motor_command, avoid_latency ← behavior avoid
    else if zero_vector or spiral_latency then
        left_motor_command, right_motor_command, spiral_latency ← behavior spiral
    else if TO_sensed or timeout or homing_latency
        left_motor_command, right_motor_command, homing_latency ← behavior homing
    else
        left_motor_command, right_motor_command ← behavior search

    if home_sensed then
        gripper_motor_command ← behavior drop
    else if TO_sensed then
        gripper_motor_command ← behavior grip
    else
        gripper_motor_command ← behavior steady

    update the home vector
    send motor commands to servos
```

The various activation conditions of each behavior are the predicates to each if-statement and the resulting commands are assigned to variables in the then-clause. The various locomotion commands often require a series of motor commands to take place in a given sequence. Therefore, these behaviors also have a latency variable that allows them to remain active for the relevant amount of time.

The environment consists of the physical properties of the foraging area and the robot's home base. It also includes the general conditions the robot is expected to operate under. The home base is a 60cm x 60cm area of floor covered with reflective aluminum tape. All the robots will start an experiment inside the base and then back out to start their search. The base is large enough for all six robots. Home base sits in the middle of a square foraging area eight meters on a side. A wall of white foam board, high enough to trigger the robot's IR sensors, surrounds the foraging area. The floor surface is an industrial type carpet which does not produce excessive wear on the robots' plastic tail-skids, but still has enough friction push a target object hard enough against the sensor to trigger it. This carpet is marked with gridlines one meter apart to enable reconstruction of the robots' paths from videotape.

Several general conditions need to be met when the environment is set up. The temperature must be between 60 and 90 degrees Fahrenheit to ensure the proper operation of the electronics and batteries. The environment should not be excessively moist. There should not be drop-offs or immovable obstacles of a height lower then the bump ring on the robot. Lastly, the area should have a near uniform magnetic field, so the compasses will function properly.

Chapter 4 – Experiments with the CMR system

In order to evaluate the biomimetic design model presented in Chapter Two, the robotic system designed in the previous chapter was constructed and evaluated. Both individual robots and groups of robots were tested to determine if the designed system could accomplish the task set for it in the environment for which it was designed. Individual robot tests were used to select and validate a path integration algorithm for the navigation system. Group tests were done to evaluate the CMR system's performance in certain parameters while accomplishing it's foraging task, and to discover if the dynamics were qualitatively similar to those displayed by *Cataglyphis* during it's own foraging.

4.1 Single robot experiments: validation of the navigation system

In translating from the natural to the artificial world, one has to worry about the fidelity of every step along the way. To keep this from becoming an insurmountable problem, one can test the implementation in stages in the same way a computer programmer tests code in pieces. As each level of the design hierarchy is completed, it can be tested for individual functionality. That way, the final debugging cycle is testing only the integration of the various subsystems, not their functionality as well. If this process is not used it might be nearly impossible to find the source of any particular malfunction in a large and complex robotic system. Therefore, at every step along the way, the system was tested in an environment similar to the one it would be run in for its foraging task. For instance, one of the first tests involved making sure that the survival section of the drive motor architecture functioned properly by itself. These tests were not recorded, but were important to the process of actually making the CMR system functional.

Since this design and test cycle was already necessary, the robot control architecture designed in Chapter 3 was left purposefully vague concerning certain details of the navigation system's functioning. Instead of specifying a specific methodology at design time that may not work, the design and test cycle was used to evaluate the functioning of different path integration algorithms. Therefore, a series of individual robot experiments were performed to test and evaluate two path integration algorithms. There are two easily available choices, the full mathematical formulation and the approximation thought to be used by *Cataglyphis* (Müller and Wehner, 1988).

The full mathematical formulation is simple vector addition. Each time the drive motor commands were updated the distance the robot traveled is calculated from magnetic heading, the last commanded motor speeds, and the time duration for which the commands were executed. The *Cataglyphis* approximation to path integration is described in section 3.3. This method is based on using small difference terms of angular deflection to iteratively modify the homeward vector.

Experimental results for both algorithms can be seen in Figures 4.1 and 4.2. In all experiments the robots ran a two-leg course with a 90 degree turn, similar to the one used to investigate ant navigation by Wehner, et. al. (1996). The robot was programmed to run for a fixed duration in each of the two directions. After completing the course, the robot attempted to navigate home using the stored vector information. Each plot shows the ending location where the home vector ran out for each of the six trials. The plots are marked with a typical path that the robot took during one of the six trials.

Each robot is slightly different in its construction and the orientation of its compass. Thus, each one of the three robots in the experiments has a different shape to its path. In the outdoor experiment shown in Figure 4.1, only one of the three robots was used to test both algorithms. Since each robot has a different compass arrangement, this might produce skewed results. The need to use a different robot for the ant algorithm test arose after Rayborn's compass suffered a mechanical failure. At that point, Krusty was substituted to finish the experiment sequence. Because that action might make it difficult to compare the performance of the robots, a second set of experiments was performed using Krusty only and is shown in Figure 4.2. The same programs were loaded onto the robot and the results charted as in the previous experiment. In this case, the experiment was run indoors – in Engineering B109. This room was determined to have very little magnetic distortion, and was therefore suitable for the experiments. The indoor location was used because of the difficulties of keeping the robots secure while outside and the problems of wear and tear caused by running on bare concrete.

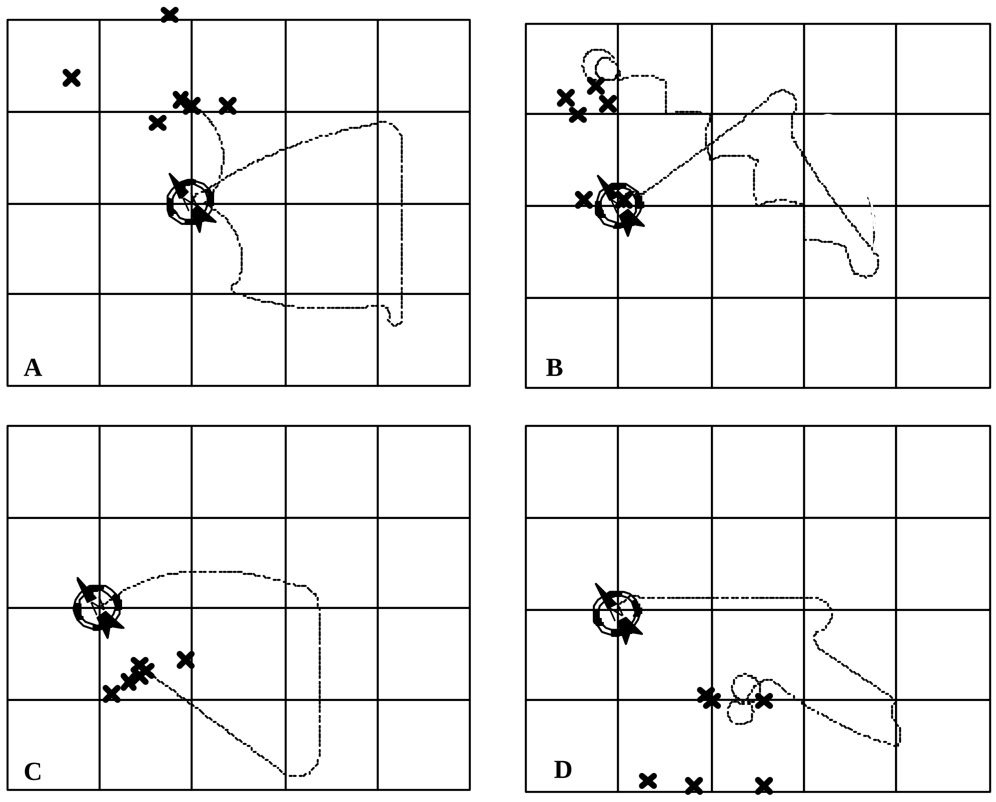


Figure 4.1. Navigation performance of individual robots using two different navigation algorithms. Each robot used its internal compass to run at top speed for 5s to the east and 5s to the south. The robot then attempts to navigate back to its starting point. Th compass rose denotes both the starting location of each run and the direction of magnetic north there. An X denotes the ending location of each run. A dashed line in each figure shows the path of a randomly selected run. **A** and **B** are the robots Proteus and Krusty, respectively, navigation with the *Cataglyphis* approximation algorithm (Müller and Wehner, 1988). **C** and **D** are the robots Proteus and Rayborn, respectively, using the full mathematical solution to path integration. Figure is to scale, with each gridline being 50cm apart.

The paths shown are typical of the most runs in each experiment. Note that the figures were drawn from videotape of the runs that occurred on the pictured gridlines, rather then by measurement during the run, and are therefore not perfectly accurate.

The results show that the dynamics of the two algorithms are quite different. The full mathematical path integration algorithm tends to produce ending locations that are tightly clustered, but some distance off from the starting point. The ant's approximation to path integration yields a larger spread of locations, which are closer to the desired ending point. In fact, in many of the runs the ant algorithm actually passed nearer to the starting point then its final position. In two out of three comparisons, the mathematical path integration method is further off on average then the *Cataglyphis* approximation algorithm.

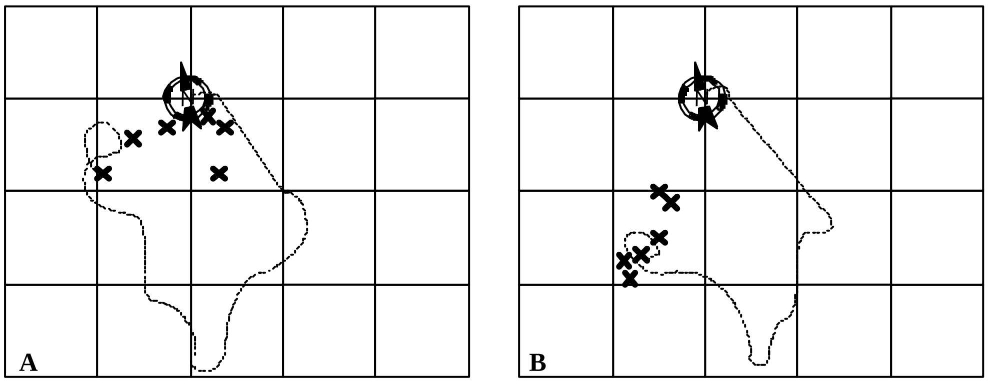


Figure 4.2. A second set of navigation experiments. Both experiments were performed indoors, in a room with minimal magnetic disturbances. The robot Krusty was run, using the exact same programs as seen in Figure 4.1. Again, the grid is 50cm on a side and the plot markings have the same meanings as before. **A** shows the robot running the Cataglyphis navigation algorithm. **B** is the robot using the full mathematical solution to path integration.

Table 4.1 shows that it is difficult to determine the better algorithm by mean distances alone. Proteus' mean distance is smaller using the mathematical algorithm then with the ant approximation. However, the reverse is true of Krusty and Rayborn. When the total errors are added up and averaged for all episodes on each algorithm, the ant approximation method comes out the winner by a small amount. Since the ant approximation algorithm also had closer passes of the typical path to the starting location, was generally easier to fine tune, and produced richer behavior it was selected for use in the foraging program.

Table 4.1. Summary statistics for the experiments shown in Figures 4.1 and 4.2. These numbers represent the distance and magnetic heading of the six stopping points in relation to home base for each experiment.

Robot	*Cataglyphis* apprx. path integration	Distance (m)	Heading (deg)
Proteus (Fig4.1)	Mean	0.69	350
	Median	0.57	354
	Standard Deviation	0.23	22
	Range	0.52	64
Krusty (Fig4.1)	Mean	0.44	343
	Median	0.56	344
	Standard Deviation	0.26	41
	Range	0.63	126
Krusty (Fig4.2)	Mean	0.34	185
	Median	0.31	189
	Standard Deviation	0.18	49
	Range	0.49	105
All	Mean	0.49	52
	Median	0.54	357
	Standard Deviation	0.26	103
	Range	0.96	316

Robot	Mathematical path integration	Distance (m)	Heading (deg)
Proteus (Fig4.1)	Mean	0.45	149
	Median	0.44	148
	Standard Deviation	0.06	17
	Range	0.17	51
Rayborn (Fig4.1)	Mean	0.92	144
	Median	0.94	138
	Standard Deviation	0.22	17
	Range	0.58	48
Krusty (Fig4.2)	Mean	0.82	203
	Median	0.85	203
	Standard Deviation	0.22	4
	Range	0.54	9
All	Mean	0.73	165
	Median	0.69	158
	Standard Deviation	0.27	31
	Range	0.87	86

4.2 Multiple robot experiments: large-scale foraging

The multiple robot experiments were carried out in a large-scale environment in order to test the navigation system to its limits. In a confined, closed space it is fairly easy for a robot wandering randomly to pass over a certain point with regularity. Therefore, the environment used here is an eight-meter by eight-meter enclosure constructed of white foam board six inches high. The foam board is easy for the robots' infrared obstacle detectors to see, and is easily constructed. A one-meter grid was laid out on the floor to aid in recording of the experiments. This setup was located in Lory Student Center Room 228, one of the few rooms of sufficient size on the whole campus which did not have severe magnetic disturbances. Ten target objects were clumped together in an area approximately two meters long by one meter wide, located about 1.5 meters northeast of the home base and oriented with its long axis on a magnetic heading of roughly 135 degrees.

Four separate algorithms were tried, resulting in the four different experiments shown in Table 4.2. All four, however, started with the same basic algorithm found in Experiment #1.

All experiments start with six robots inside the home base area. When turned on, the robots back out of the base and randomly select their radial search heading. This heading is then followed for a randomly selected number of "steps" between 0 and 30. A "step" is the amount of distance covered by a robot running full speed ahead between one update of the home vector and the next one. Since the robots are not programmed for real-time control, the amount of time involved is uncertain, as is the step size, too.

After completing its outbound radial course, a robot randomly selects a heading of 90 or 135 degrees in either direction from its current one. This heading is maintained until 15 seconds have elapsed. This departure from the ant's method ensures that the robot will remain close enough to the home base that it has a good chance of returning. A robot using this algorithm will reach distances of up to three meters from the home base. However, the search will be more concentrated near the home base.

Once the 15 seconds are finished, the robot uses its accumulated home vector to try and navigate homewards until it reaches the spot at which it believes home to be located. As we know from the previous section, this rarely coincides with the actual location of the home base. Thus, the robot enters its spiral search pattern.

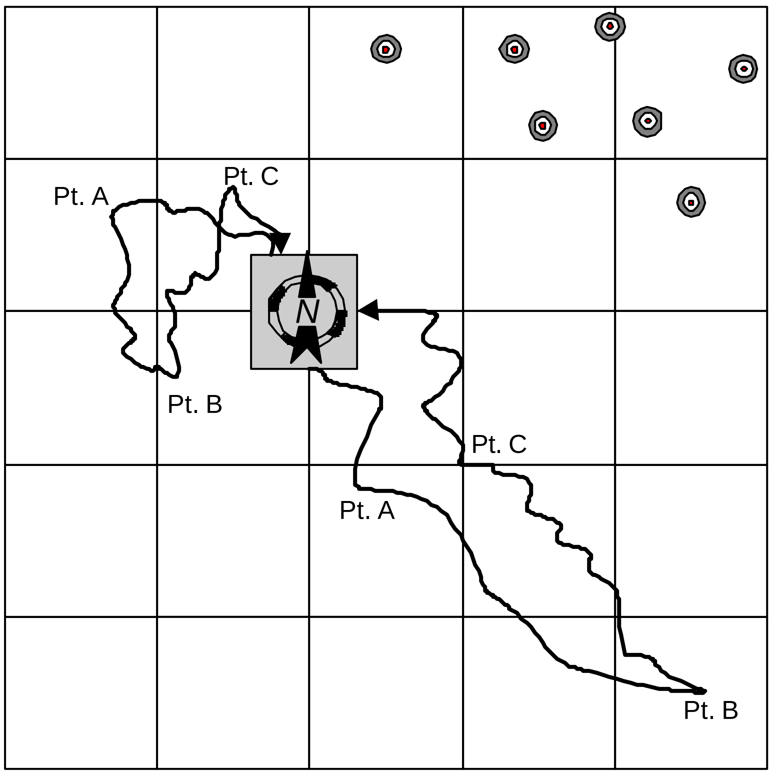


Figure 4.3. Two typical successful returns from the multiple robot experiments. These paths illustrate the two-leg search structure. Search is active for 15s after leaving the base (shaded box). The robot follows a radial heading for 0-30 "steps", up to Pt. A. The robot then turns 90 or 135 degrees in either direction and proceeds until the 15 seconds expires, at Pt. B, or it finds a target object. It uses its path integration to navigate to where it believes home to be, Pt. C, and then enters a spiral search pattern until encountering the real home base. The drawing is to scale and the grid squares are 1m on a side. The bulls-eyes denote the location of the seven target objects visible in this drawing.

This base navigation algorithm was run in three recorded tests and in several more unrecorded, informal runs. The three recorded tests are lumped together as Experiment #1 in Table 4.2. The homing and spiral behaviors do not return the robot to its home base every time. When a robot is unsuccessful it will typically end up travelling across the environment in a spiral search, getting steadily farther away from home base. Eventually, its progress is stopped by the environment's enclosure where it ends up continually butting against the wall, trying to continue its travel. From observations during the tests, it appears that this process will continue indefinitely. At this point the robot becomes trapped and useless, with little chance of getting back to home base again.

The results for Experiment #1 show a raw success ratio of 0.72. This is probably an anomalous number. During one of the runs a single robot successfully returned seven times in a row before becoming lost, an occurrence never observed at any other time. If that run is excluded the success ratio is one-third. This later number is similar to the ratio observed in unrecorded experiments and general observations during the design and test cycle. Typically, ratios of one-third to one-half were observed. The foraging task, however, went unfulfilled. Although five robots picked up a target object, none were ever successfully returned to home base.

Table 4.2. The results of four multiple robot experiments using different versions of the navigation system. Experiment #1 uses the base navigation algorithm. It is a lumped experiment of three separate runs, whereas the other experiments are only one run each. Experiment #2 adds the forward **Wander** behavior, which activates after 2 min. Experiment #3 changes the **Wander** timeout to 1.5 min. changes it to cycle from 20s forward to 1 min. spiral search. Experiment #4 is the same as #3 with 30s forward and 20s spiral search. Time normalized measurements are created by linearly scaling the successful returns with the experiment time to 10 minutes. Time normalized values are not applicable to Experiment #1 because there was no chance of a lost robot in those runs recovering with time.

	Experiment #1	Experiment #2	Experiment #3	Experiment #4
No. of successful returns of a robot	13	4	7	10
No. of incomplete returns	18	6	6	6
No. of returns by wandering rather then navigation	N/A	2	3	3
No. of target objects picked up	5	3	4	5
No. of target objects retrieved	0	0	0	0
Time elapsed (min)	20	11.5	7	10
Raw success to failure ratio	0.72	0.67	1.17	1.67
Normalized success to failure ratio (scaled to 10 min exp. duration)	N/A	0.58	1.67	1.67
Normalized success rate (robots returned per 10 min)	N/A	4	10	10

In an effort to improve the successful return ratio, a new behavior, called **Wander**, was added to the architecture. The resulting complete behavior architecture can be seen in Figure 4.4. **Wander** becomes active after a set amount of time, in the case of Experiment #2 it was two minutes. At that point it subsumes all navigation behaviors, but not the survival behaviors. In Experiment #2, **Wander** consisted only of full forward commands to both drive servos. This resulted in a new dynamic for lost robots. After a while of butting against the enclosure, they would turn away from it as **Avoid** became active; and then rather then turning back into the enclosure, **Spiral** would be subsumed by **Wander** and the robot would go straight. Because all of the robots exhibit a slight bias in one direction or the other when attempting to go straight, the robot would eventually end up contacting the enclosure again at some point down its path. Thus, the overall behavior of **Wander** created an emergent wall-skipping dynamic that was not expected during design.

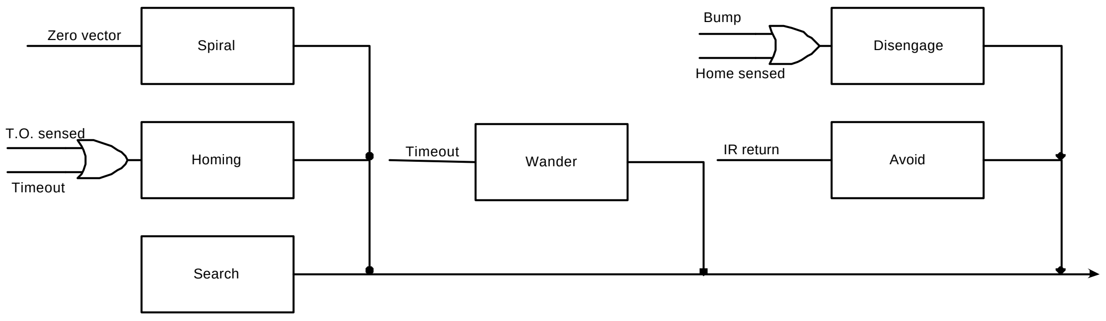


Figure 4.4. The modified behavior architecture used in Experiments #3 and 4 to help recover lost robots.

To aid in getting the robots out into the open, a set of four, half-meter long protrusions were placed along the walls. They were affixed pointing towards the home base and spaced 90 degrees apart on the center of the enclosure walls. These protrusions were designed to redirect a wall-skipping robot to head out into the middle of the environment. However, when robots did head towards the interior of the experimental environment they would take a relatively straight path.

The results from Experiment #2 showed a lower success ratio then Experiment #1: 0.58. But again, this may be attributable to an anomalously high number of returns at one point in the that experiment. This experiment is measured using a ratio normalized over 10 minutes of experiment time, in order to enable comparisons between the experiments that use the **Wander** behavior. This is necessary because these experiments can continue to generate successful returns indefinitely, but the maximum number of unsuccessful runs is always the number of robots, six. Without a time normalized measure of success ratio, an algorithm would appear to do better if it was run for a longer time then the others. Again, the foraging task was unsuccessful. Three robots picked up target objects, but none were returned to home base.

Because the previous algorithm had such a straight path through the center of the environment it ended up exploring on a very narrow section of it. It therefore had little chance of encountering home base by chance. The algorithm for Experiment #3 was designed to address this problem. To fix it, the **Wander** behavior was modified to cycle between 20 seconds of forward travel and one minute of spiral search. The spiral search was executed assuming that the point where forward motion ended was home base. This modification helped Experiment #3 achieve a much higher success ratio than previous experiments.

Results showed that this experiment had a time normalized success ratio more than double both previous attempts: 1.67. This difference is strong enough to leave little doubt that this is an important improvement. However, there were still no successful returns of target objects to home base. Four robots picked up target objects and were unable to return them.

Table 4.3. Detailed results from Experiment #4. Two minutes after leaving the base the robots assume that they are lost and execute a wandering behavior consisting of 30s forward motion followed by 20s of spiral searching to ensure that they will not get stuck trying to endlessly navigate to someplace outside the experimental environment. The experiment lasted 10min, at the end of which there were no robots permanently stuck against a wall or out of commission in any other way.

No. of successful returns of a robot	10
No. of approaches within 1m of the center of home base	21
No. of approaches within 1m while carrying a target object	1
No. of times a robot physically interfered with another robot	13
No. of times there was physical interference within 1m of the center of home base	5
No. of target objects picked up	5
No. of target objects knocked out of the gripper by mischance	4
No. of successful returns of a robot while carrying a target object	0

Observations of Experiment #3 left the impression that the robots were perhaps spending more time spiral searching then was necessary. Therefore, Experiment #4 was the same as before, but with **Wander** consisting of 30 seconds of forward travel and 20 seconds of spiral search. However, this yielded no change in the time normalized success ratio. This indicates that there may be little difference between the division of time of the algorithm, but rather that it's the combination of forward and spiral paths that make it effective. Further details from Experiment #4 can be found in Table 4.3. Again, no target objects were returned, but five were picked up and one passed rather close to the home base. Four of these objects were lost by the robots holding them, either through bouncing off of the enclosure walls or from collisions with other robots.

4.3 Discussion of results

The high number of approaches within one meter of home base shown in Table 4.3 seems to indicate that great gains might be possible if the accuracy of the navigation could be increased only a small amount. For instance, if a beacon or other method of landmark recognition were employed.

That table also points out the area of greatest concern – the crowding and physical interference between robots. Even though there was 64 square meters of area inhabited by only six 18cm diameter robots, there were still 13 collisions or other cases of physical interference. Five of these happened in a one-meter radius around the home base. That is, nearly 40% of physical interference occurred in only five percent of the environment's area.

There were no successful cases of target object retrieval. In the 17 cases where a target object was acquired by a robot, they either lost the object in a collision [12], never navigated home [4], or experienced an erasure of program memory from what was most likely static electricity discharge [1]. However, there are two mitigating factors. First, the environment was purposefully designed to be as challenging as possible. A smaller environment would make target object retrieval much more likely. Secondly, a number of robots in possession of target objects passed literally within centimeters of the home base. This includes the robot which had its memory erased by a static discharge. It had actually been on its way into the home base when this happened, but was incapacitated before it could get the target object fully inside the base. It therefore seems likely that if an experiment were run for an increased amount of time, say one hour, several target object returns might be likely even in the large environment used for these experiments. In experiments performed before the modifications discussed in this thesis, the TJ robots demonstrated the ability to run for over 1.5 hours on a full battery charge. While their current endurance is certainly less due to the new electronics and sensors, it may very well be possible to run them for an hour from a full charge.

Finally, the dynamics of the robots' navigation are very similar to those of a *Cataglyphis* forager. Comparing the typical paths in Figures 4.1, 4.2, and 4.3 with Figure 4.5 below, shows that qualitatively the ant's path may be slightly more tortuous. It also consists, as mentioned before, of a single outbound radial direction that is modified by a biased random walk forcing the distance from the path to be proportional to the square root of time. Instead, the robot has both an outbound radial direction and a 90 or 135 degree turn leg. This causes the algorithm to cover a different area then the ant would. However, it is similar to the ant's algorithm in that the distance from the nest does not increase linearly in time.

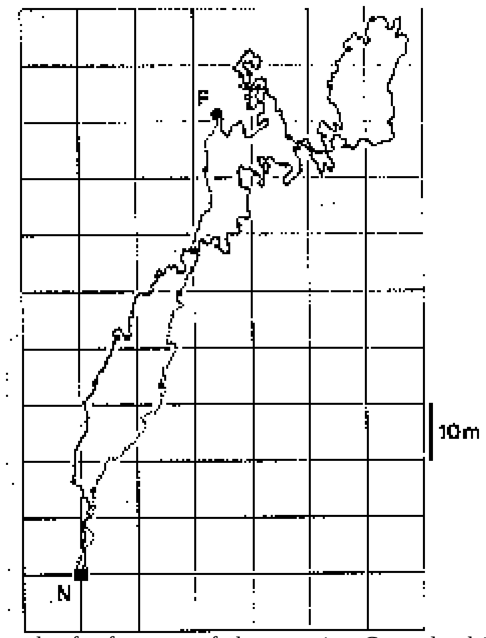


Figure 4.5. An example path of a forager of the species Cataglyphis fortis from (Müller and Wehner, 1988). Outbound path is solid and the inbound path is stipled. N is the location of the nest and F is the food item found by the forager.

Chapter 5 – Conclusions

5.1 Regarding the CMR system designed

Overall the robotic system designed in this thesis performed satisfactorily. It demonstrated the basic competencies necessary to accomplish a foraging task. Although it never actually retrieved a target object, small design changes or an extended experiment would certainly result in improved foraging.

Most of the problems experienced in foraging were caused by two errors in the fidelity of translation from the natural to the artificial system. These problems were in the areas of navigation and physical interference. It is interesting to note that this type of error is predicted and warned against in the biomimetic design method.

First, there was a problem with low fidelity of translation from the biological navigation model to the robotic one. The navigation system implemented did not have any landmark recognition capabilities other then at zero distance (touching). However, the *Cataglyphis* forager does have the ability to learn to use landmark recognition. In fact, if a forager has been to a place before, it can navigate back to the nest from there even if its homeward vector store is zero. Therefore, if the robotic system had the ability to recognize landmarks, like the home base, from a distance of even one meter the successful return rate might increase by as much as a factor of two.

This result is backed up by previous research in the area of robot and animal navigation (Nehmzow, 1995). This research has indicated that a combination of topological maps, learned canonical paths, and landmark recognition is a robust solution to navigation. It is employed in many examples of animal navigation and also works well in the robot domain. In this system although a topological map is employed (the homeward vector), there is no ability to follow a canonical path or to recognize a landmark from a range greater then zero. The extension of this system with those abilities would likely increase the success rate tremendously.

Secondly, there was a problem of low fidelity of translation from the natural system to the biological model. There are no existing models of how *Cataglyphis* ants interact with others of their own kind without physically harming one another. The robotic system experienced a high degree of physical interference, which resulted in several robots either losing their grip on target objects or being unable to enter the home base. Matariæ (1997) found that learning behaviors such as yielding and proceeding could enhance a CMR system's ability to accomplish a foraging task. The robotic system used here only avoids sources of infrared light, either reflected off other objects or directly from another robot's IR emitter. If another form of robot interaction was used it might improve task performance by increasing successful return ratio and decreasing the amount of target objects lost after being picked up.

5.2 Regarding the biomimetic design method

The method developed here was a useful tool in creating a cooperative mobile robotic system for a foraging task. The biomimetic design method aided in both conceiving of a design capable of solving a given problem, and in implementing that design in a robotic system. In this case following the method successfully allowed the designer to create a workable real world system – the ultimate goal of robotics.

The system created was functional and able to work in a dynamic and uncertain environment. Changing the environment configuration slightly before, during, or in between experiments would have little effect on the task accomplishment of the system. Of course, large changes such as removing the aluminum tape marking home base or putting a hole in the enclosure wall would likely have a large impact on task accomplishment. However, small changes such as moving target objects, adding obstacles, or moving home one meter would likely have little effect. Thus, the biomimetic design methodology is capable of generating robust, real world solutions.

However, the proposed method is not without its pitfalls and problems. As it exists, the method has little to say about quantities instead of qualities. While it is known that the fidelity of a translation is important there is no way to figure out just what level of fidelity is needed in any given situation. Nor are there any ways to know what parts of a biological model may safely be excluded and still create the dynamic desired.

Another problem is that this method requires knowledge in two disparate disciplines – biological sciences and engineering. Thus, difficulties can arise in working outside of one's area of expertise. This has traditionally been overcome in the past by collaboration of two or more people who have expertise in each area individually. The emerging trends in Artificial Life research, however, are pointing the way towards designers that can operate comfortably in both bodies of knowledge.

What the biomimetic design method offers most of all is guidance and the beginnings of formalism. To create highly competent robots in a reliable manner we need a science of robot design. Such a science would consist of a knowledge base and a set of principles that could be used to explain and/or predict the behavior of robotic systems. While the method proposed here is a long way from such an achievement, it does provide a possible framework for viewing robotic system's design. If the abstraction presented here turns out to have merit in design practice, it could be fleshed out into a more useful form by continued research.

5.3 Future work

Finally, there are several avenues that seem worth pursuing in both developing the CMR system and the biomimetic design method.

The robotic system described in this thesis is capable of becoming a workhorse system for research at Colorado State University. Its competencies should continue to be developed.

- As mentioned previously, some form of landmark recognition would greatly increase the system's utility. This might take the form of a beacon operating in some frequency of the EM spectrum, a simple visual system, or an ultrasound ranging system.
- A radio-frequency serial port transceiver would be of great aid to any further work on the system. It would enable easier debugging, explicit inter-robot communication, and better experimental data collection.
- A simple, round-robin multi-tasking kernel is available for this platform. It was used successfully on one robot, but problems interfacing it with the timing system were never solved. If implemented, it would create a control architecture much more closely coupled with the environment.
- The compass module could be redesigned to use a resistor network to combine the four digital output lines into a single analog line. This would open up three input ports on the robot controller for future expansion.
- Research into social behaviors like yielding and proceeding could reduce the physical interference of robots and enable them to perform their tasks better.

- A redesign of the gripper's double-pull actuation system could yield higher gripping forces and enable the robot to retain target objects better in the case of a collision.

There are two broad categories of research that can be pursued to develop the biomimetic design methodology – empirical and theoretical. Empirical developments that might be worth pursuing include:

- A more complete survey of previous biomimetic robots and their design process.
- Creating more robotic systems using the process and then evaluating their successes and failures. Trends and sticking points could be traced back to changes necessary in the method.
- A specific species could be picked and mimicked by a robotic system. Researchers familiar with the species in question would then evaluate the system. The design would be iterated on several times to improve it and the necessary improvements might yield insight into the biomimetic design process.

Theoretical developments of the process would be of a much more difficult nature at this stage. But some possible avenues of research include:

- Pursuing work in creating more realistic simulation worlds that can start to approximate the dynamic and uncertain nature of the real world.
- New methods of task and environment modeling could help develop a better understanding of agent-task-environment interaction. This would be beneficial to pursuing studies in both animal behavior and robotics.
- Finally, work in the far future could concentrate on developing models of how animal behavior is created on a neural level. Such a model might have broad implications for recreating complex behavior in an artificial system.

References

Ali, M. M. (1994). *Exploration-based design synthesis of behavior-based autonomous robots.* Ph.D. dissertation. Colorado State University, Ft. Collins, CO.

Arkin, R. C. (1998). *Behavior-Based Robotics.* Cambridge, MA: The MIT Press.

Axelrod, R. and Hamilton, W. D. (1981). The evolution of cooperation. *Science.* 211:1390-96.

Beer, R. D., Chiel, H. J., & Sterling, L. S. (1990). A biological perspective on autonomous agent design. *Robotics and Autonomous Systems.* 6:169-186.

Braitenburg, V. (1984). *Vehicles: Experiments in Synthetic Psychology.* Cambridge, MA: The MIT Press.

Brooks R.A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation,* RA-2:14-23.

Cao, Y., Fukunaga, A., & Kahng, A. (1997). Cooperative mobile robotics: antecedents and directions. *Autonomous Robots,* 4:7-27.

Connell, J. H. (1990*). Minimalist Mobile Robotics: A Colony-style Architecture for an Artificial Creature.* San Diego, CA: Academic Press.

Derringer, D., Cook, S., Madrill, B., Good, J., Burt, A., & Kedrowski, P. (1999). *Team Triad.* Technical Report [Senior Design]. Department of Mechanical Engineering, Colorado State University, Ft. Collins, CO.

Doty, K. L. (1999). *The Talrik Junior^{TM} Assembly Manual.* Gainesville, FL: Mekatronix.

Duchon, A. P., Warren, W. H., & Kaelbling, L. P. (1998). Ecological robotics. *Adaptive Behavior.* 6:473-507.

Dudek, G., Jenkin, M., Milios, E., & Wilkes, D. (1996). A taxonomy for multi-agent robotics. *Autonomous Robots,* 3:375-397.

Dugatkin, L.A. (1997). *Cooperation Among Animals: An Evolutionary Perspective.* New York, NY: Oxford University Press.

Espenscheid, K., Quinn, R. Chiel, H., Beer, R. (1994). Biologically-inspired hexapod robot control. In *Proceedings of the Fifth International Conference on Robotics and Manufacturing.* 89-102.

Ford, P. (1996). *Description of a Robot, Task, and Environment Using the Theory of Affordances.* M.S. thesis. Colorado State University, Fort Collins, CO.

Goldstein, E. B. (1996). *Sensation and Perception* (4th ed.). Pacific Grove, CA: Brooks/Cole.

Goss, S. and Deneubourg, J. (1992). Harvesting by a group of robots. In *Proceedings of the European Conference on Artificial Life.* Cambridge, MA: The MIT Press.

Hall, E. L. and Hall, B. C. (1985). *Robotics: A User-Friendly Introduction.* New York: Holt, Reinhart, and Winston.

Harkness, R. D. and Maroudas, N. G. (1985). Central place foraging by an ant (*Cataglyphis bicolor* Fab.): a model of searching. *Animal Behavior.* 33:916-928.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press.

Hölldobler, B. and Wilson, E. O. (1990). *The Ants.* Cambridge, MA: Belknap.

Kartalopoulos, S. V. (1996). *Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications.* Piscatawy, NJ: IEEE Press.

Kuwana, Y., Shimoyama, I., and Miura, H. (1995). Steering control of a mobile robot using insect antennae. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems.* Los Alamitos, CA: IEEE Computer Society Press. 2:530-535.

Kube, C. R. and Zhang, H. (1997). Task modelling in collective robotics. *Autonomous Robots.* 4:53-72.

Lindgren, L. and Nordahl, M.G. (1995). Cooperation and community structure in artificial ecosystems. In, C. G. Langton, (Ed.), *Artificial Life: an overview.* Cambridge, MA: The MIT Press.

Loeb, J. (1918). *Forced Movements, Tropisms, and Animal Conduct.* Philadelphia, PA: J. B. Lippincott.

Matariæ, M. J. (1997). Learning social behavior. *Robotics and Autonomous Systems.* 20:191-204.

Matariæ, M. J. (1993). Designing emergent behaviors: from local interactions to collective intelligence. In, J.A. Meyer, H.L. Roitblat, & S.W. Wilson, (Eds.), *From Animals to Animats 2: Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior.* Cambridge, MA: The MIT Press. 432-441.

Meyer, J. A. (1997). From natural to artificial life: biomimetic mechanisms in animat designs. *Robotics and Autonomous Systems.* 22:3-21.

Morse, T. M., Ferrée, T. C., & Lockery, S. R. (1998). Robust spatial navigation inspired by chemotaxis in *Caenorhabditis elegans.* *Adaptive Behavior.* 6:393-410.

Müller, M. and Wehner, R. (1988). Path integration in desert ants, *Cataglyphis fortis.* *Proceedings of the National Academy of Sciences.* 85:5287-5290.

Müller, M. and Wehner, R. (1994). The hidden spiral: systematic search and path integration in desert ants, *Cataglyphis fortis.* *Journal of Comparative Physiology* A. 175:525-530.

Nehmzow, U. (1995). Animal and robot navigation. *Robotics and Autonomous Systems.* 15:71-81.

Parker, L.E. (1995). The effect of action recognition and robot awareness in cooperative robotic teams. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics and Systems.* Los Alamitos, CA: IEEE Computer Society Press. 1:212-219.

Playter, R. R. (1994). *Passive Dynamics in the Control of Gymnastic Maneuvers.* Ph.D. dissertation. MIT, Cambridge, MA.

Russell, A., Thiel, D., & Mackay-Sim, A. (1994). Sensing odour trails for mobile robot navigation. In *Proceedings of the IEEE Conference on Robotics and Autonomous Systems.* pp. 2672-77.

Simon, H. A. (1998). *The Sciences of the Artificial* (3rd ed.). Cambridge, MA: MIT Press.

Seeley, T. D. (1995). *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies.* Cambridge, MA: Harvard University Press.

von Frisch, K. (1967). *The Dance Language and Orientation of Bees.* (transl. L.E. Chadwick). Cambridge, MA: Belknap.

Wehner, R., Michel, B., and Antonsen, P. (1996). Visual navigation in insects: coupling egocentric and geocentric information. *Journal of Experimental Biology*. 199: 129-140.

Werger, B. B. and Matariæ, M. J. (1996). Robotic “food” chains: Externalization of state and program for minimal agent foraging. In, *Proceedings of From Animals to Animats IV, SAB-96*. Cambridge, MA: The MIT Press.

Walter, G. (1953). *The Living Brain*. New York: W. W. Norton.

Wilson, E.O. 1971. *The Insect Societies*. Cambridge, MA: Belknap.

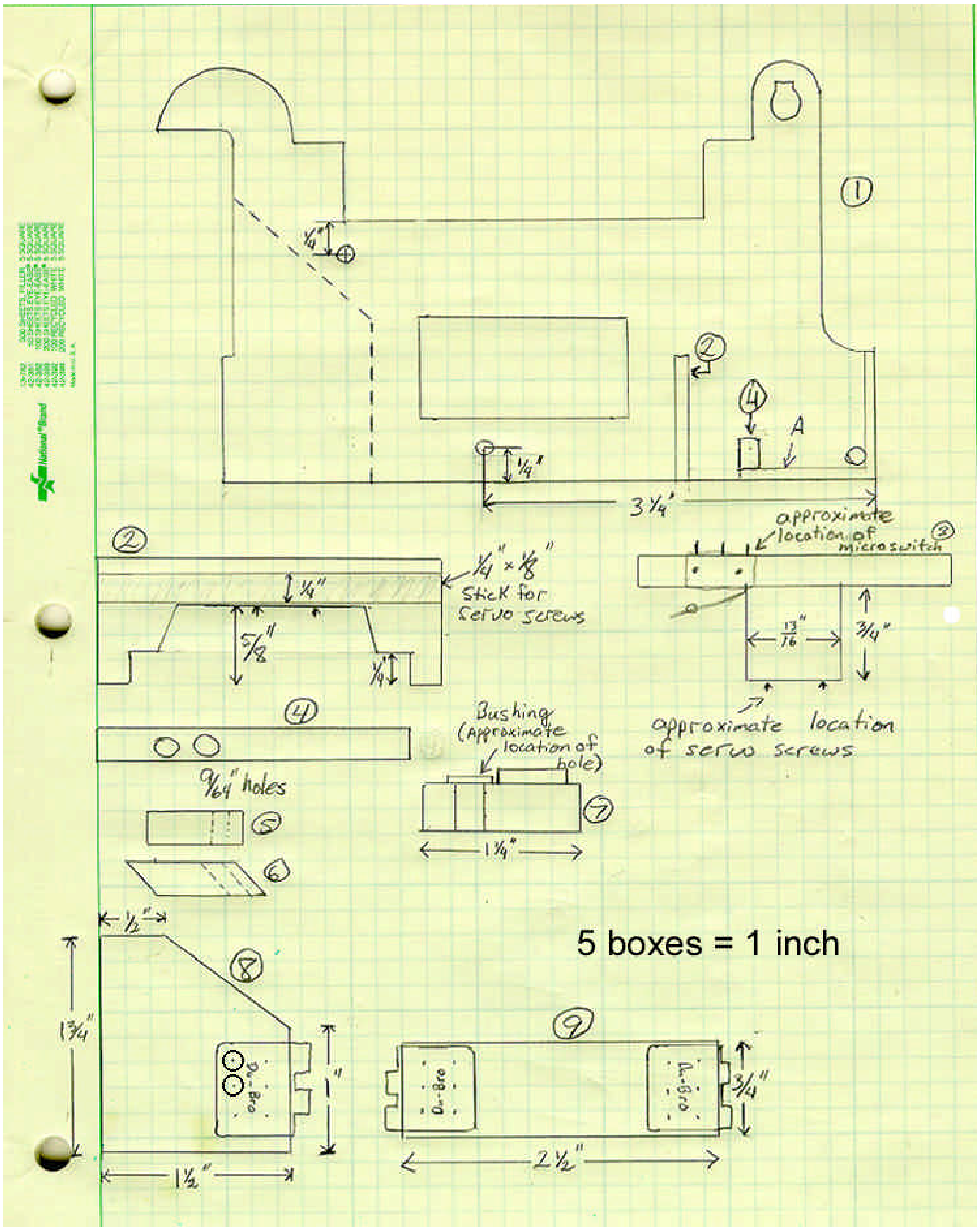
Appendix A – The Talrik Junior Gripper Modification

By Jonathan Hurst

Reprinted with permission.

Parts List

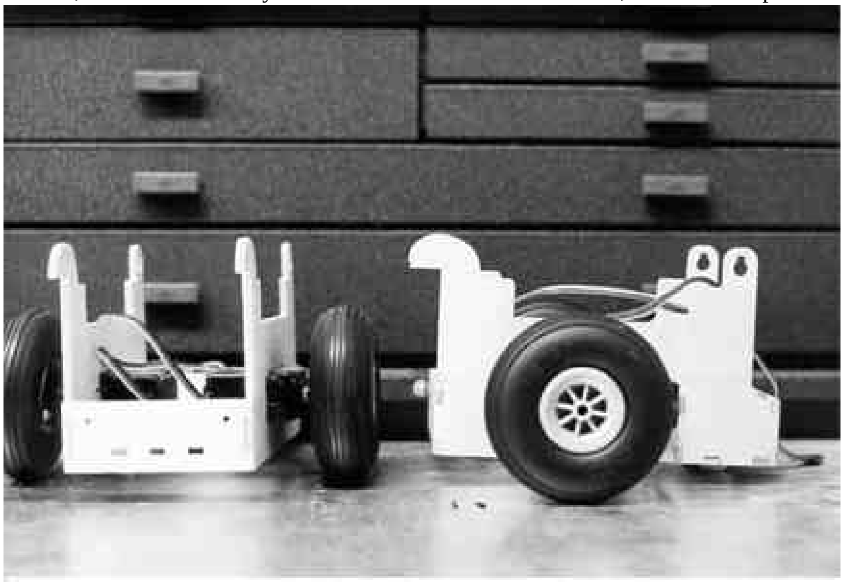
3” $^{3}/_{16}$” music wire
6” $^{1}/_{8}$” music wire
36” ¼” x $^{1}/_{8}$” stick
36” ¼” x ¼” stick
1 standard model airplane servo
20 feet of string – kevlar or nylon
36” pushrod tubing
1 oz thin CA glue
1 oz thick CA glue
small torsional spring
microswitch
12 #0 wood screws
plastic bushings
nylon hinges
sticky-back foam for grippers
brass string holder, with screws, attaches to servo arm
plastic tube, fits inside brass string holder
2 ½” 4-40screws
6 4-40 nuts
4 #4 washers
2 4-40 T-nuts



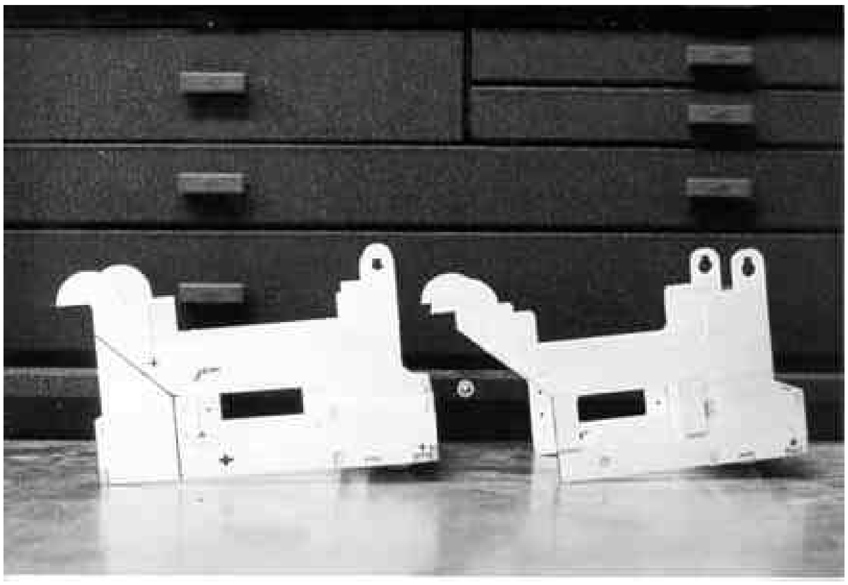
1
1/4"
1/4"
3 1/4"
2
4
A
2
1/4" × 1/8" stick for servo screws
1/4"
5/8"
1/4"
approximate location of microswitch
3
13/16"
3/4"
approximate location of servo screws
4
9/64" holes
5
6
Bushing (Approximate location of hole)
7
1 1/4"
1/2"
1 3/4"
8
1"
Du-Bro
1 1/2"
9
Du-Bro
Du-Bro
3/4"
2 1/2"
5 boxes = 1 inch

A. Body preparation, servo installation

These modifications to the Talrik Junior robot add a powered set of grippers, so the robots can pick things up. All physical modifications are done to the lower half, the top computer platform is not altered. Any time a direction is stated, such as up, down, right, or left, the direction refers to the robot being upright, and the reader facing the gripper or the front of the robot. Any time a glue joint is to be permanent, use thick CA glue. The Thin CA glue is for saturating wood to prevent splitting. Be sure to read through the entire instructions first, then read as you build. Below, the TJ's as they come from the manufacturer, with the top removed.

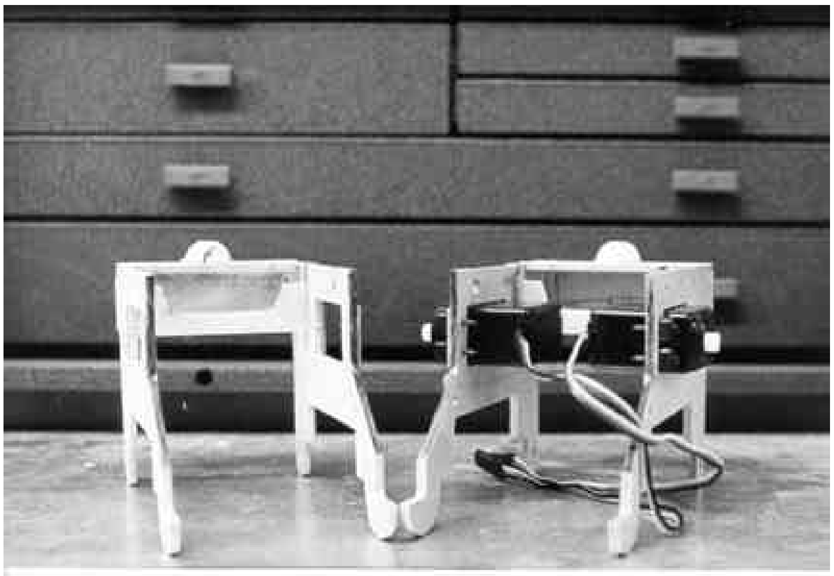


1. Remove wheels, servos
2. Make cuts following the dotted lines and drill holes for part 1 on the schematic
3. Add $^{1}/_{32}$" plywood spacer to each servo mount

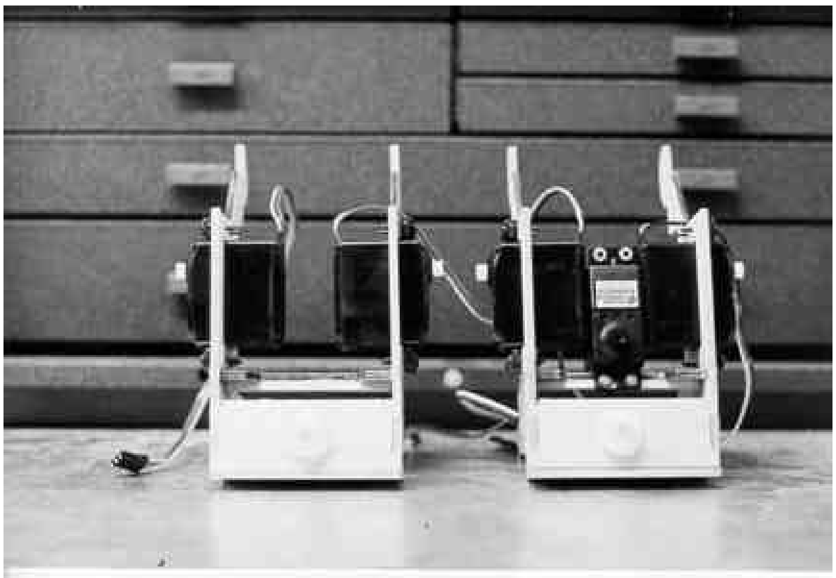


4. Draw 2 lines, ¼" from bottom and $^{5}/_{8}$" from bottom, on the horizontal plate that separates the servos and the battery. It is labeled part 2 on the schematic.
5. Cut all the way across on the ¼" line, and in the middle about half the width at the 5/8" line. A dremel with a grinding wheel is recommended for this cut.

6. Replace servos on part 1, screwing the screws through the $^1/_{32}$" spacer and into the original screw holes.



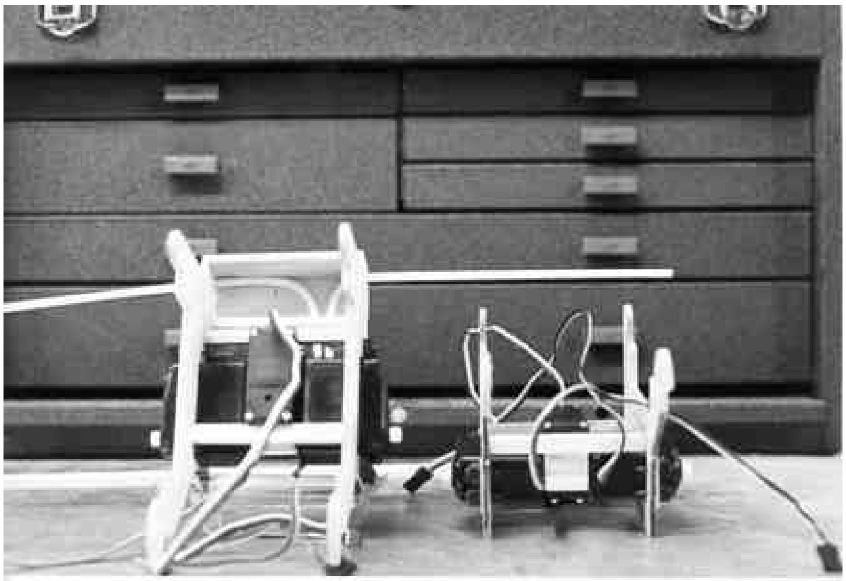
7. Add $^1/_8$" x ¼"stick for third servo – this is glued to the back of part 2 so the bottom of the stick is parallel to the $^5/_8$" line.
8. Drill small pilot holes, at most 1/16," to prevent splitting, and screw the servo in. the shaft of the servo should be close to the plate, nearer the battery. The opposite side is not yet supported.



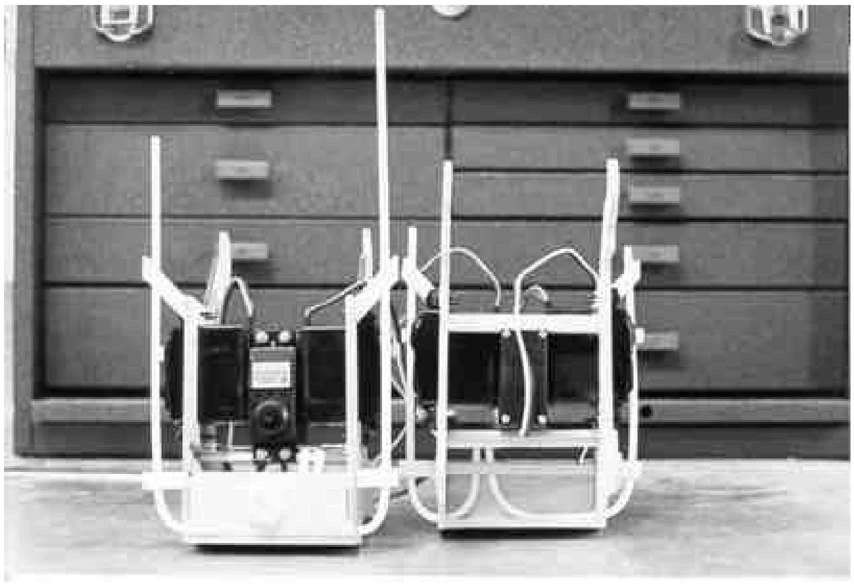
9. Create part 3 of the schematic by building it out of pieces. Add ¼" x ¼" stick directly over the two remaining servo screw holes, resting on the drive servos. Add ¼" thick wood between the servo and the ¼" x ¼" stick so the servo is flat when resting on the added wood. It should be approximately ¾" from the servo rails to the ¼" x ¼" stick. Glue this piece to the ¼" x ¼" stick, drill pilot holes, and screw the servo down.

B. String guide installation

10. Measure a ¼" x ¼" stick to the inside width of the TJ. Drill two holes off center and next to each other – these are where the string will enter from the servo. Glue it in at the edge of the lower battery support plate. It is part 4 on the schematic.
11. Take pushrod tubing, pull it though the hole in the rear corner of the TJ, and insert into thehole in the stick that was glued in the previous step. One tube goes to each side. At every place the stick is through a hole in the frame, glue the tube to the stick so it doesn't slip.



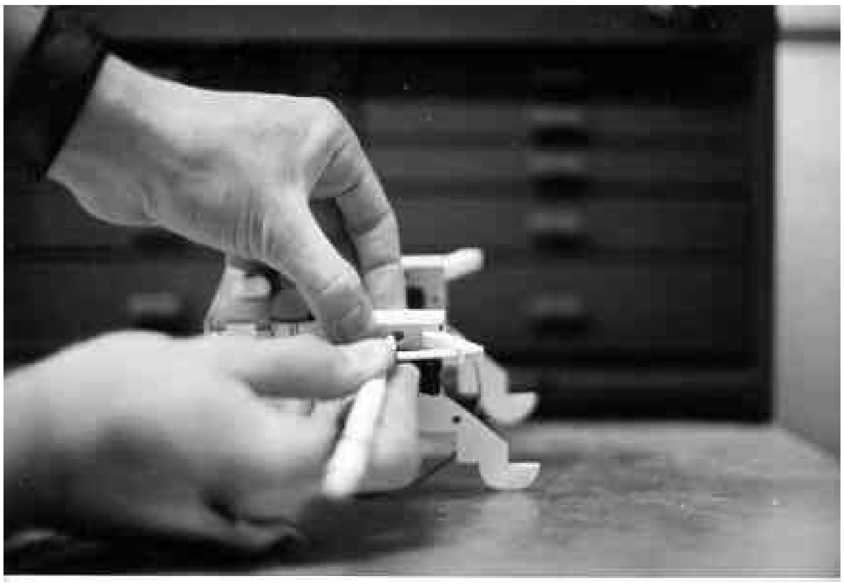
12. Slide guide pieces 5 and 6 (one set on each side) onto the tubing, and glue them to the tube and the side of the TJ at the rear and front, respectively.



C. Gripper

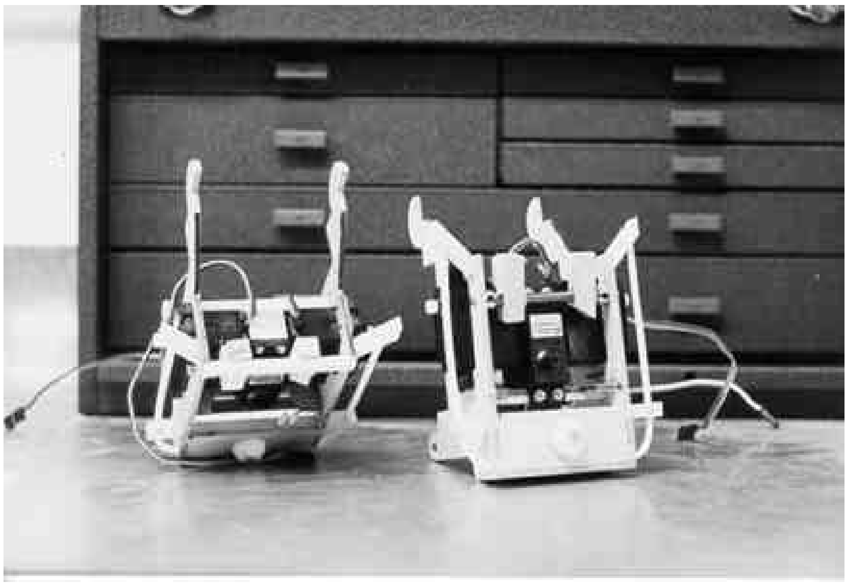
13. Cut two pieces of $^3/_8$" x $^3/_8$" wood 1½" long. These are piece 7 on the schematic. With the TJ upside down, place them on the drive servos next to part 1, and even with the front cut. With a pencil, mark on the stick where the center of the ¼" hole is. Take the sticks, place them in

a vise to avoid splitting, and drill a ¼" hole. If plastic bushings (recommended) are available, drill a 9/32" hole, and insert the bushing.

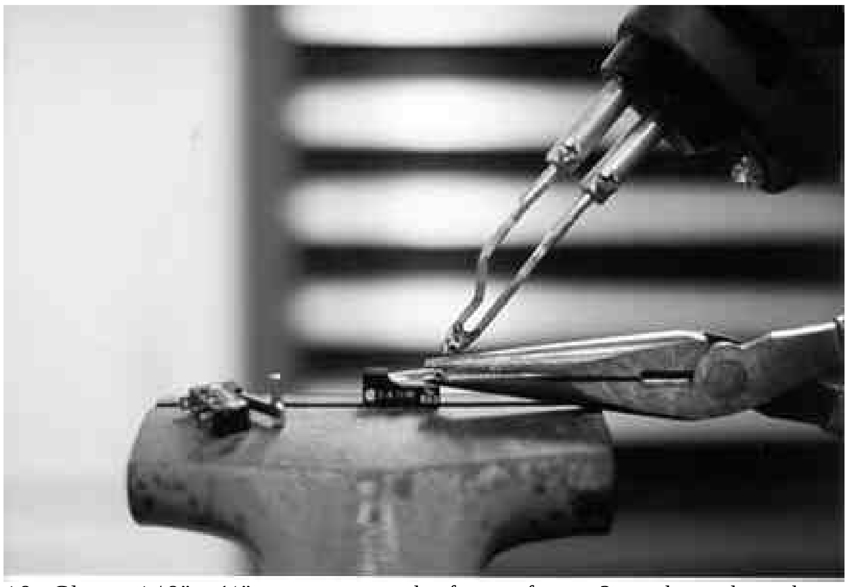


14. On one of the pieces, glue a piece of ¼" x $^1/_8$" stick on the side where the spring will be located. Put the spring, temporarily, on the metal shaft, and slide it into the bushing, so you can see where the point of the spring rests on the wood. Make a small, visible depression. Take the shaft out, and drill a $^1/_{16}$" hole where the depression is. Saturate the piece with thin CA glue, to harden the wood.

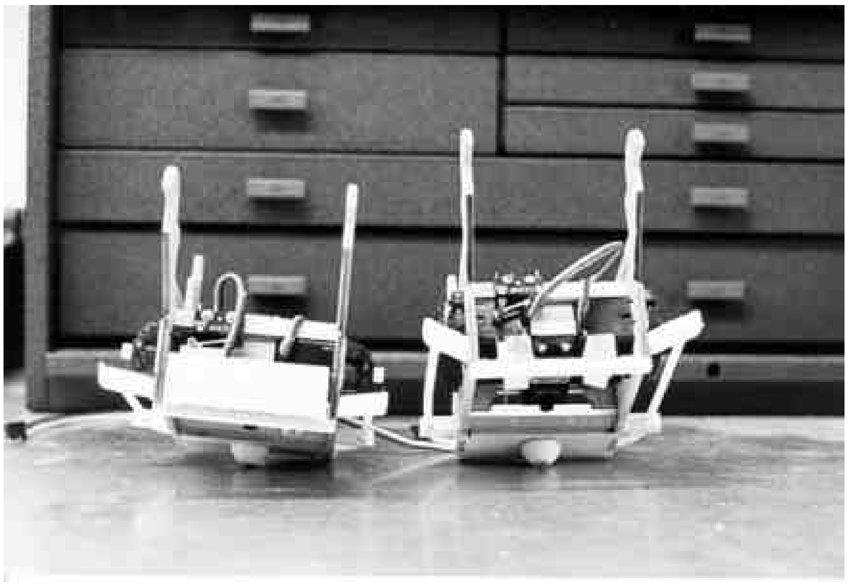
15. Put the shaft in the hole on one side of the TJ, and as you slide it through, put these pieces on it in this order: wheel collar, stick that does not hold the spring, stick that does hold the spring, spring, wheel collar, other side of TJ. Tighten the wheel collars, and insert the spring catch into the hole.
16. Glue a $^1/_8$" x ¼" stick across the bottom, on the edge near the font of the TJ, to prevent the spring-loaded piece from going below it. When the glue has dried, the spring-loaded piece should rest on the newly added restraining stick.



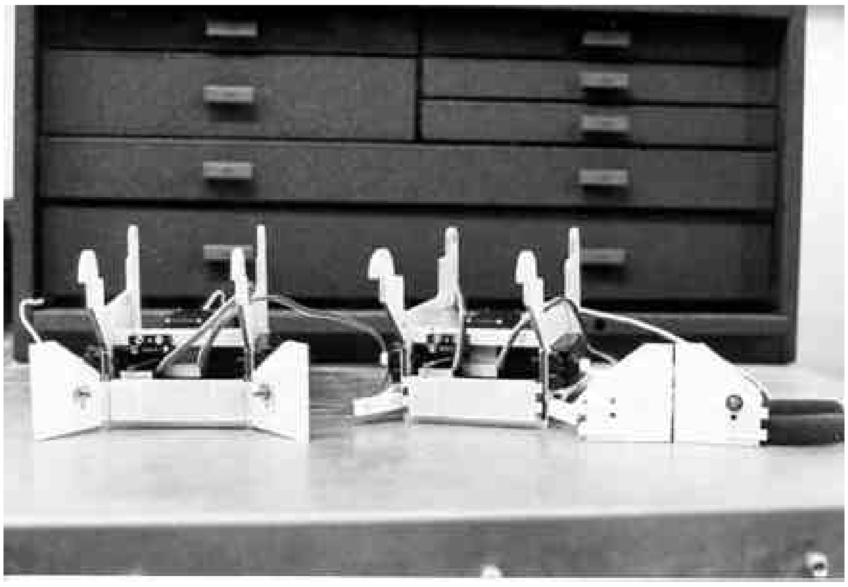
17. Solder a short piece of copper tubing onto the microswitch, perpendicular to the lever arm. Flatten the end of the tubing that will be soldered to the switch. This is so the gripper can activate the microswitch, while the switch is out of the way.



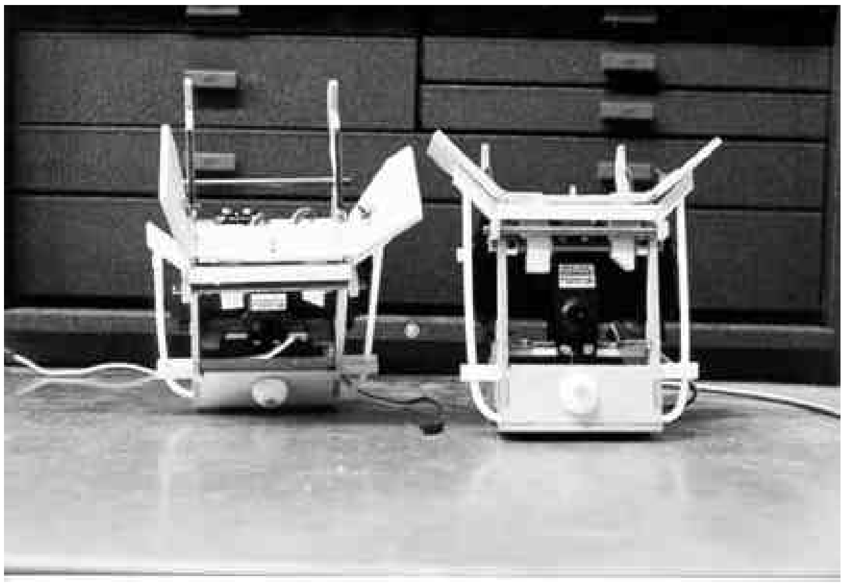
18. Glue a 1/8" x ¼" spacer onto the front of part 3 so that when the microswitch is screwed on it doesn't hit the servo. Drill small pilot holes, and screw the micsroswitch to the spacer.
19. Create a gripper back from 2½" x ¾" piece of $^{1}/_{8}$" thick plywood, and glue it to the front of the pieces supported by the shaft, labeled part 7. Take care to ensure they are even, and parallel with the restraining stick. Glue a small piece of scrap ply to the gripper back, so when it moves upward, the microswitch is activated. If needed, the switch arm can be bent to so it will switch at the proper gripper angle. This gripper back is shown as part 9 on the schematic.



20. Glue one half of the two-piece hinges to the edge of the plywood gripper back.
21. Cut the grippers according to the plan. They are part 8 on the schematic Glue the other half of the hinges to the grippers, and once the glue has dried, drill the top two holes out to $^1/_8$” in the second row away from the pin. There should be six small holes in the hinge, and these are used as pilot holes.
22. Add a T-nut and string-holding screw according to the picture. The T-nut and head of the crew should both be on the side of the gripper that faces away from the front when the gripper is open. Tighten a nut down to keep the screw from unscrewing, and to tighten the T-nut into the wood.
23. Place the grippers on the gripper back, and insert the pins.

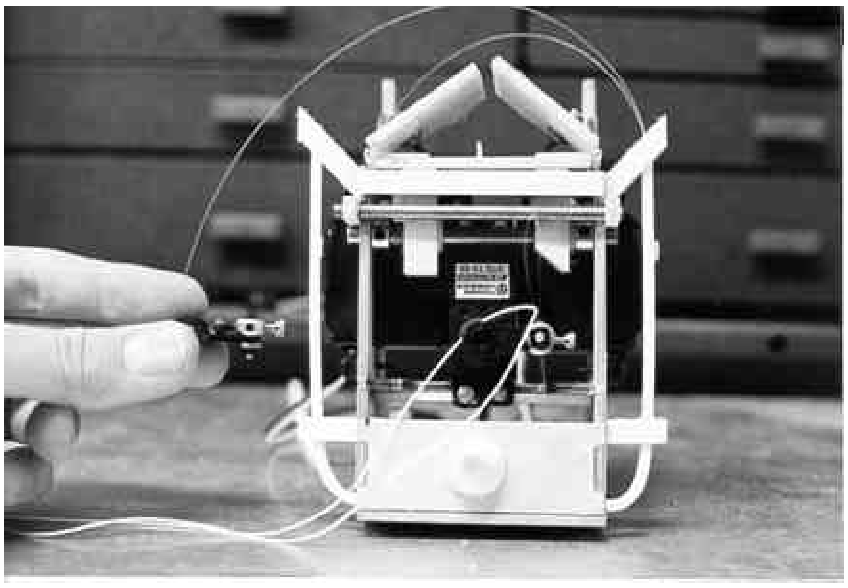


24. Add a small eye screw ¼” from the bottom, and in the direct center of the gripper back.



D. Threading the strings

25. Clip three arms off the four-sided servo arm included with the servo, so one arm remains. Drill the hole out at the correct length (on the servo arms that are included with the servos, the hole farthest out is the correct lever arm) so the brass fitting fits in the hole, insert it, and secure it with the included retainer. Cut a small piece of plastic tubing (I used electrical shrink tube) and insert it into the brass fitting. This is to prevent the screws from cutting the string when you tighten it.
26. Cut a long piece of monofilament line, about and arm’s length, and fold it in half. Use this length to lead the string through. The line should first go through the two white pushrod tubes. Feed the two free ends through the brass fitting, and into one of the white pushrod tubes. Push it through until it comes out near the gripper. Cut a piece of string, again about an arm’s length, and loop it through the fishing line, so the two lines meet at the mid point of each line. Pull the fishing line through, so the loop of the string comes out near the gripper, and take the fishing line out.



27. Feed the string through the hole in the gripper, and loop it over the screw on the inside of the gripper. Add a washer and a nut, and tighten it lightly, so the string doesn't come undone when there is no slack.
28. Repeat for the other pushrod tube and gripper.
29. Slide a piece of $^1/_8$" music wire, slightly longer than the width of the TJ, into the two holes near the top of the TJ body.
30. Make two short sticks with $^1/_8$" holes drilled in them at the ends, similar to part 5, and glue them near the base of the TJ on the side, right behind the gripper back, and right ahead of the two servos. The holes should be underneath the TJ's, so a piece of $^1/_8$" music wire will slide through both holes, and cross the underside of the TJ parallel to the gripper back, and right behind it. The string should be able to go from the servo arm, over this rod, over the rod on the top of the TJ, and down to the screw eye and grippers arms without touching anything in between.
31. Thread the two ends of the loop of fishing line through the brass connector
32. Loop two pieces of string through the fishing line
33. Pull the fishing line through the brass connector, with the strings, being careful not to pull the plastic tube out of place.
34. Lead the strings under the bottom piece of music wire, and over the top piece, and down to the eye screw on the back of the gripper. Thread one loop of string through one side, and the other loop through in the other direction, so one string goes to one gripper arm and the other string goes to the other gripper arm. Thread the loop through the hole in the grippers, and place the loop over the head of the screw.
35. Move the servo arm to one extreme, and place the grippers at the same extreme, and pull all strings tight. When they are all tight, loosen the strings slightly that cause the gripper to open. This is because the angle and tension changes when the gripper closes, and the increased tension will cause the gripper to lift before it closes unless there is some slack.. Again, be careful not to pull the plastic tube out of the brass fitting. When the string tensions are correct, tighten the screw so they will not move.
36. Cut a piece of foam tape to the approximate shape of the gripper arm, and attach it firmly for better grip.

Appendix B – Compass board design

The Dinsmore Model No. 1490 Digital Compass Sensor is composed of four Hall Effect sensors arrayed around a magnetic compass needle. Each Hall Effect sensor is driving an open collector output on pin 3. Heading is determined by which sense lines are high. For instance, if the all sense lines are high except the forward one then the robot is facing north. If the rear and left lines are high, and forward and right lines are low, then the robot is facing northwest.

A schematic diagram of the compass board can be found in Figure C.1, and its PCB layout mask is shown in Figure C.2. The specifications for the Dinsmore No. 1490 can be found in Figures C.2 and C.3.

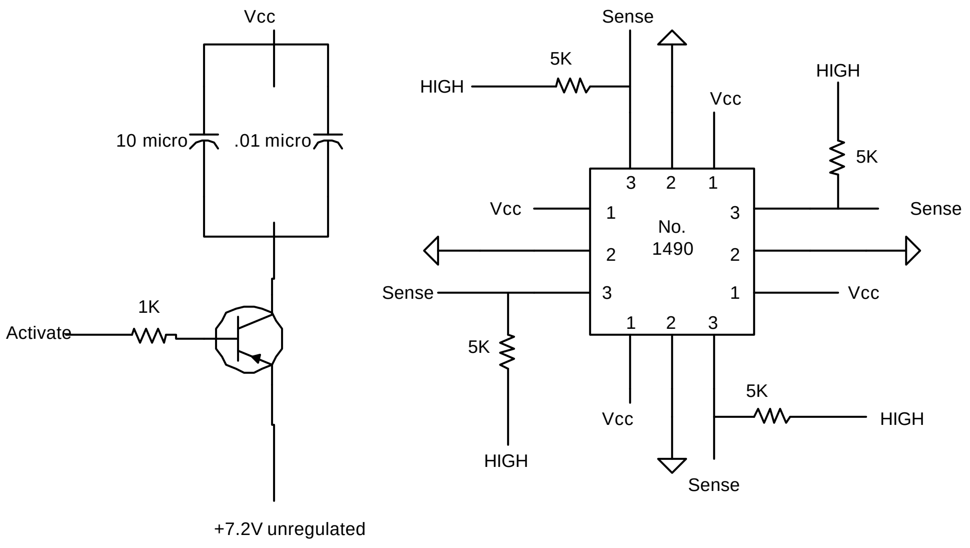


Figure C.1. The Dinsmore compass board schematic.

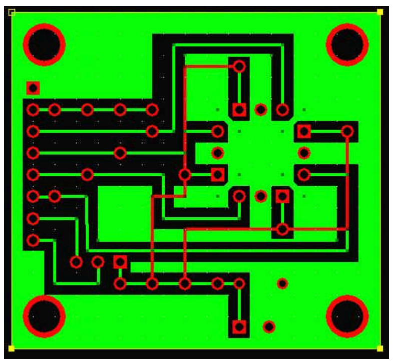
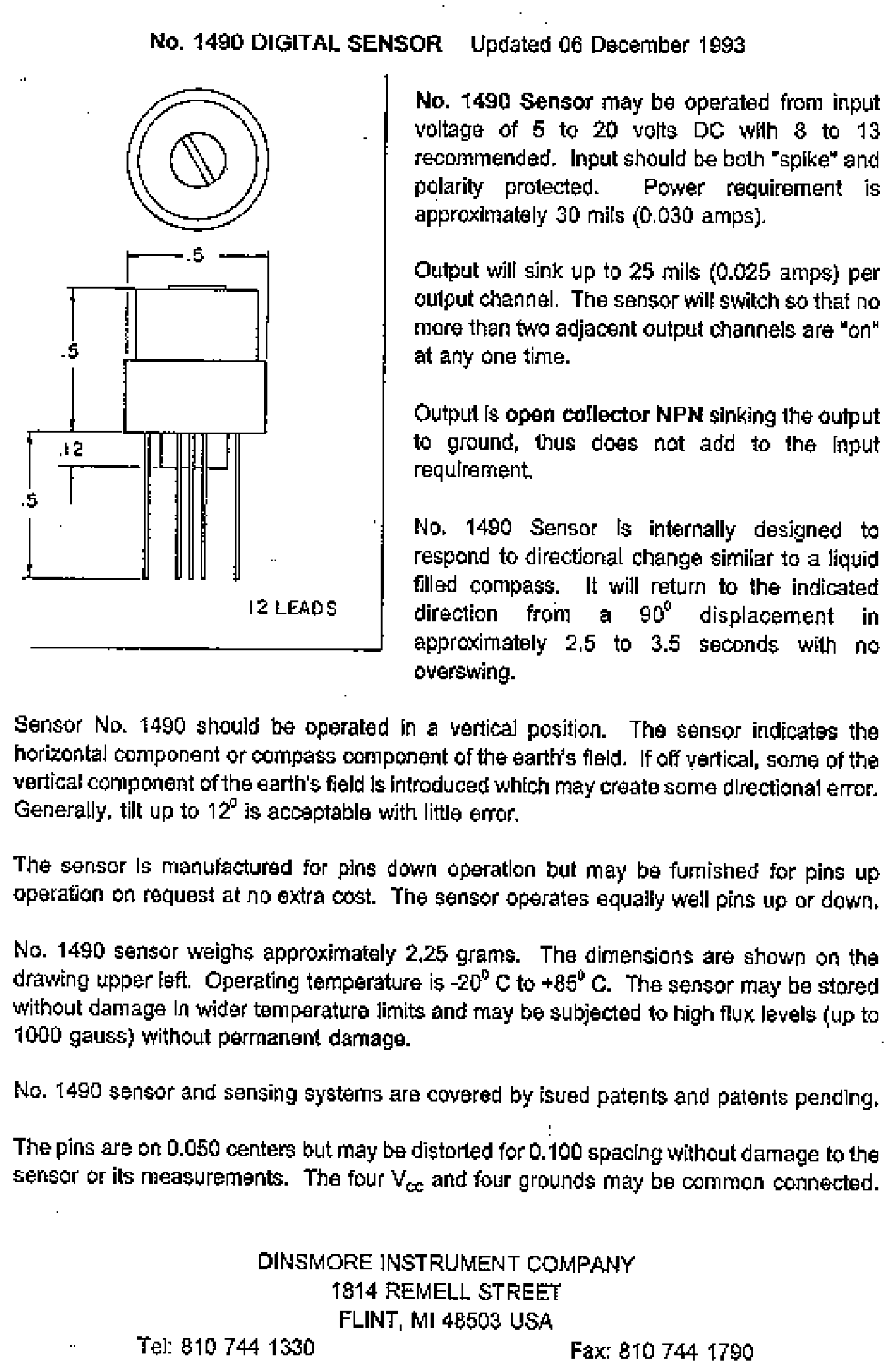
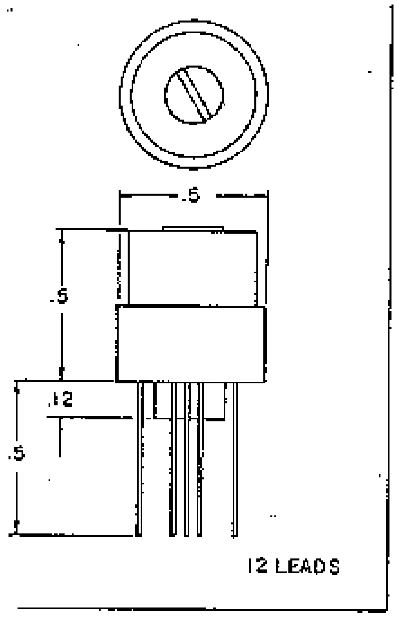


Figure C.2. Printed circuit board mask for the Dinsmore compass board. Layout by Dennis Clark. The compass mounts in the box array of 12 pins, which should be on 0.100 inch centers. The pins on the left side of the board are for a male 0.100 inch connector, whose lines are the following from top to bottom: ground, +5V regulated, signal 1, signal 2, signal 3, signal 4, activate, +7.2V unregulated.



No. 1490 DIGITAL SENSOR Updated 06 December 1993



No. 1490 Sensor may be operated from input voltage of 5 to 20 volts DC with 8 to 13 recommended. Input should be both "spike" and polarity protected. Power requirement is approximately 30 mils (0.030 amps).

Output will sink up to 25 mils (0.025 amps) per output channel. The sensor will switch so that no more than two adjacent output channels are "on" at any one time.

Output is **open collector NPN** sinking the output to ground, thus does not add to the input requirement.

No. 1490 Sensor is internally designed to respond to directional change similar to a liquid filled compass. It will return to the indicated direction from a 90^0 displacement in approximately 2.5 to 3.5 seconds with no overswing.

Sensor No. 1490 should be operated in a vertical position. The sensor indicates the horizontal component or compass component of the earth's field. If off vertical, some of the vertical component of the earth's field is introduced which may create some directional error. Generally, tilt up to 12^0 is acceptable with little error.

The sensor is manufactured for pins down operation but may be furnished for pins up operation on request at no extra cost. The sensor operates equally well pins up or down.

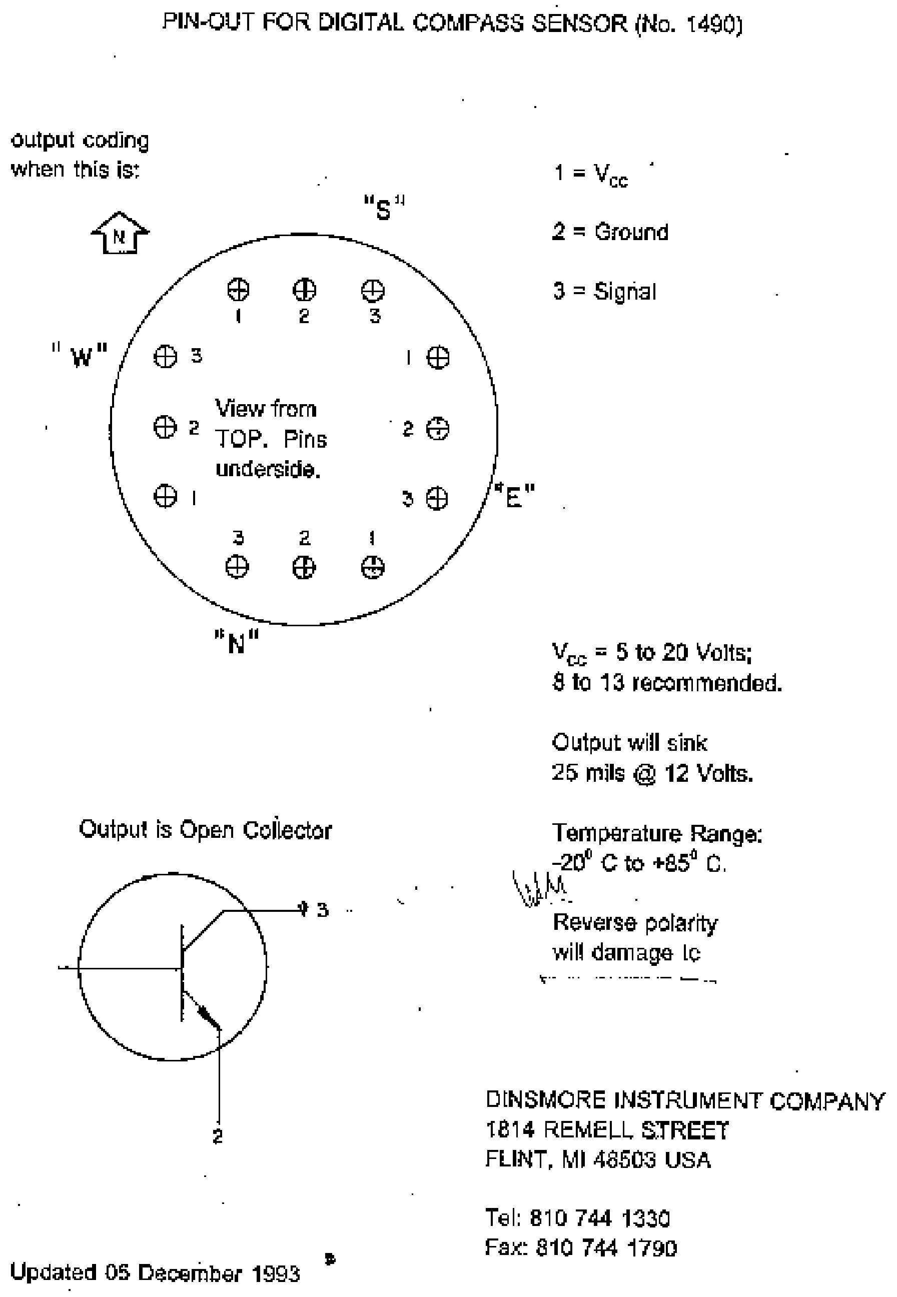
No. 1490 sensor weighs approximately 2.25 grams. The dimensions are shown on the drawing upper left. Operating temperature is -20^0 C to $+85^0$ C. The sensor may be stored without damage in wider temperature limits and may be subjected to high flux levels (up to 1000 gauss) without permanent damage.

No. 1490 sensor and sensing systems are covered by isued patents and patents pending.

The pins are on 0.050 centers but may be distorted for 0.100 spacing without damage to the sensor or its measurements. The four V_{cc} and four grounds may be common connected.

DINSMORE INSTRUMENT COMPANY
1814 REMELL STREET
FLINT, MI 48503 USA
Tel: 810 744 1330 Fax: 810 744 1790

Figure C.2. The specification sheet for the Dinsmore No. 1490 digital compass.



PIN-OUT FOR DIGITAL COMPASS SENSOR (No. 1490)

output coding when this is:

1 = V_{cc}

2 = Ground

3 = Signal

V_{cc} = 5 to 20 Volts; 8 to 13 recommended.

Output will sink 25 mils @ 12 Volts.

Output is Open Collector

Temperature Range: -20° C to +85° C.

Reverse polarity will damage Ic

DINSMORE INSTRUMENT COMPANY
1814 REMELL STREET
FLINT, MI 48503 USA

Tel: 810 744 1330
Fax: 810 744 1790

Updated 05 December 1993

Figure C.3. The pin-out diagram for the Dinsmore No. 1490 digital compass.

Appendix C – Foraging code for the CMR system

FORAGE4.C: The main program for Experiment #4

```
/********************************************************

*

* forage4.c

*

* Platform:  ALFs - Ant-Like Foragers, a set of modified TJ

*            mobile robots.

*

* Function:  Behavior-based control architecture for a

*            exploring an unknown area, finding target
*            objects, and returning them to home base.
*            The robots cooperate by distributing their
*            search areas via stigmergic communication.
*
* Programmer: Jason Fleischer
* Created:    April 5, 1999
*
* Revision History-----------------
*            v0.1  4/5/99-4/7/99
*            Wrote program.  Several stub functions in place.
*            v0.11 4/7/99 (Compilation fix)
*            Changed all the pass-by-ref C++ style to pointer ops
*            v0.12 4/8/99
*            Changed speedl, speedr to global variables, fixed HOME and
*            TO_sensor #defines.
*            v0.13 4/8/99
*            Added serial communication to troubleshoot control problems
*                 and fixed problems with set_speed & homing.
*                 v1.0 4/12/99  WORKS!!!
*            Added "depart"; random dir selection,no sqrt of time:dist
*            relatioship in search.  No speed scaling. Even dir distrb.
*            v1.1 4/15/99
*            Added "seek" to enable the robot to find the home base
*            Must now change the home sensor to top mounted IR!
*            v1.2 4/23/99
*            Changed home sensor to downward pointing IR.  Added speed
*            scaling to the navigation algorithm
*
* This version was used in the single and multi-robot experiments on
* 5/2/99.  It is a 15 sec outbound leg with one random dir change.
* it also has the wander straight "I'm lost" mechanism
* it also has a periodic spiral in the "i'm lost" behavior
*
********************************************************/

#include <alf.h>  // for various #defines
#include <math.h>     // various functions used in navigation
#include <stdio.h>      // for printing things over serial port

/* This section is used to set sensitivity values for the IR
   obstacle sensors.  These are A/D conversion values in the
   range 0-256, corresponding to 0-5V from the sensor.      */
#define AVOID_THRESHOLD 90
#define AVOID_HIGH 120
#define AVOID_NORMALIZER (AVOID_HIGH-AVOID_THRESHOLD)


/*  Sets the distance at which the robot considers itself
    "close enough" to home.  At this point it begins its spiral
    search pattern, if neccessary.  Distance is in arbitrary
    "steps" - each step being the distance covered between
    vector updates                                          */
#define DIST_THRESHOLD 3


// Global Variable Declarations
// (things all the independant behavior processes can see)
int speedr=0, speedl=0, gripper=OPEN; // servo command values
int irdr, irdl;               // IR return values
double phi;              // angle of home vector
double dist=0;           // distance of home vector
int going_home=0;             // 1 when the robot is trying for home
int last_food_dir=999; // last dir food was found; 999 means not yet
int last_trip_successful=0;  // if robot found a TO last trip
int rand_seed;                // a seed for the rand # generator;

// Function Declarations
void update_home_vector(int, unsigned int, int);
int home_dir();
void set_speed(int);
int rand(int);

void disengage(int *);
void avoid();
void spiral(int *, int);
void homing(int *);
void search(int *);
void depart(int *);
void wander(int *, int);

void main()
{
       int activated=1;
       int avg_speed, temp;
       int latency_disengage=0;
       int latency_spiral=0;
       int latency_homing=0;
       int latency_search=0;
       int latency_depart=0;
       int latency_wander=0;

       /* VT100 clear screen */
       char clear[]= "\x1b\x5B\x32\x4A\x04";

       /* VT100 position cursor at (x,y) = (3,12) command is "\x1b[3;12H"*/
       char place[]= "\x1b[1;1H";    /*Home*/

       struct time last_time_step, timeout_val, lost;

       // if its more then 1:30 mins you're lost!
       lost.milli=lost.secs=30;
       lost.mins=1;

       // turn back timeout occurs at 15 seconds.
       timeout_val.milli=0;
       timeout_val.secs=15;
       timeout_val.mins=0;

       init_analog();
       init_serial();
       init_servos();
       init_clock();

       printf("%s", clear);
       printf("%s", place);

       printf("\tTitle\t\tforage1.c\n"
       "\tProgrammer\tJason Fleischer\n"
       "\tDate\t\tApril 8, 1999\n"
       "\tWhy don't the home and to_sensors trigger proper behavior?\n\n\n"
```

```
"HEAD  ERR    Dist.       Phi          Delta        Return angle\n\n\n"
"Active behav.     lat_disg     lat_sprl     lat_homn     lat_srch\n");

IRE_ON;                                           // IR emitters on
rand_seed=phi=read_heading();         // init nav algorithm

while(activated)
{
       avg_speed=(speedr+speedl)/2;  // speed of last time step

       irdr = RIGHT_IR;                     // reads right IR sensor val
       irdl = LEFT_IR;                      // reads left IR sensor val

   // Normalize IR values in the range of 0-10
   irdr = 10*((limit_range(irdr,AVOID_THRESHOLD,AVOID_HIGH)-
   AVOID_THRESHOLD)/AVOID_NORMALIZER);
   irdl = 10*((limit_range(irdl,AVOID_THRESHOLD,AVOID_HIGH)-
   AVOID_THRESHOLD)/AVOID_NORMALIZER);

       /* Gripper control architecture

              The gripper is setup so that it does not need any
              latencies as the drive motors did because none of
              the sub-tasks require a sequenced series of actions.

              The Steady behavior enables the control architecture
              to trigger off transitory rather then continuing signals.
       */

       // Drop the target object if at home and reset the timeout
       // and navigation clock for another run
       if (HOME)
       {
              gripper=OPEN;
              reset_clock();
       }

       // Grab a target object if one is found
       else if (TO_FOUND)
       {
              last_trip_successful=1;
              gripper=CLOSED;
       }

       // Hold the gripper steady otherwise
       /* no need for the code really, but here it is...
       else
              gripper=gripper;
       */


       move(GRIP_SERVO, gripper);


       /* Drive motor control architecture

              Each drive motor behavior evaluates if the correct
              preconditions exist.  When they do, the correct behavior
              that is highest in the subsumption architecture loads
              the next motor speed commands into the speedl & speedr
              variables.

              Latencies are used to enable this serial implementation
              to imitate the truly parallel subsumption architecture.
              Some of the behaviors require a sequence of actions
              to be performed.  The latency variables allow them to
              give a command and relinquish processing to the
              architecture again.  If a latency variable is set then
              the behavior will activate again and finish its task,
              provided that another higher priority behavior hasn't
              activated.  The latency variable is also used to ensure
```

```
        continued execution of a behavior given a transitory
        activation input.
*/

// Back out of home base once there
// Also, if the latencies aren't reset the robot might try
// to navigate back home as soon as it gets to far out
// on the next trip.  Also reset home vector for nav.
if (HOME || latency_depart)
{
        latency_search=latency_homing=latency_spiral=latency_disengage=0;

        depart(&latency_depart);

        printf("\x1b[11;1H          ");
        printf("\x1b[11;1H%s","depart");
}

// Disengage if bumped
else if (FRONT_BUMP || latency_disengage)
{
        disengage(&latency_disengage);

        printf("\x1b[11;1H          ");
        printf("\x1b[11;1H%s","disengage");
}

// Avoid obstacles if IR shows any
else if (irdr || irdl)
{
        avoid();


        printf("\x1b[11;1H          ");
        printf("\x1b[11;1H%s","avoid");
}

//zoom around until you have to turn looking for home
else if (time_compare(lost,curr_time))
{
        wander(&latency_wander, avg_speed);
}

// Spriral search if near home
else if (((dist<DIST_THRESHOLD) && (going_home))
|| latency_spiral)
{
        spiral(&latency_spiral, avg_speed);

        printf("\x1b[11;1H          ");
        printf("\x1b[11;1H%s","spiral");

        // Don't let the homing reactivate!
        // Is this really necessary? I think maybe Muller &
        // Wehner got it wrong and the homing is active all the
        // time or reactivates periodicaly or ??
        latency_homing=latency_search=0;
}

// Navigate home if an object found or its been too long
else if (TO_FOUND || (time_compare(timeout_val,curr_time)) ||
latency_homing)
{
        homing(&latency_homing);
        going_home=1;

        printf("\x1b[11;1H          ");
        printf("\x1b[11;1H%s","homing");

        latency_search=0;
}
```

```
		// Otherwise search for an object
		else
		{
			search(&latency_search);

			printf("\x1b[11;1H          ");
			printf("\x1b[11;1H%s","search");
		}


		update_home_vector(read_heading(),msec_diff(curr_time,
		last_time_step),avg_speed);

		move(LEFT_SERVO, speedl);
		move(RIGHT_SERVO, speedr);
		last_time_step=curr_time;

		printf("\x1b[11;22H%d", latency_disengage);
		printf("\x1b[11;35H%d", latency_spiral);
		printf("\x1b[11;48H%d", latency_homing);
		printf("\x1b[11;61H%d", latency_search);

	}
}


//double x_cord=0, y_cord=0;
//const double threshold=50;

void update_home_vector(int heading, unsigned int t_diff, int speed)
{
	const double k=0.00004009;
	double delta;

	delta = (double) heading - phi;
	if (delta<-180) delta+=360;
	else if (delta>180) delta-=360;

	if (dist==0) phi=heading;
	else	phi+= k*(180+delta)*(180-delta)*delta/dist;

	dist+= ((double)speed/100)*(1 - fabs(delta)/90);

	if (phi>=360) phi-=360;
	else if (phi<0) phi+=360;

	printf("\x1b[8;1H    ");
	printf("\x1b[8;1H%d", heading);
	printf("\x1b[8;14H        ");
	printf("\x1b[8;14H%f", dist);
	printf("\x1b[8;24H        ");
	printf("\x1b[8;24H  %f", phi);
	printf("\x1b[8;37H        ");
	printf("\x1b[8;37H  %f", delta);

}


// Returns compass heading towards home base
int home_dir()
{
	int ret;

	if (phi>180) ret= (int) (phi-180);
	else ret= (int) (phi+180);

	printf("\x1b[8;50H        ");
	printf("\x1b[8;50H%d", ret);

	return ret;
```

```
}


// Sets speeds according to err from desired heading
void set_speeds(int des_dir)
{
	int err;
	err = head_error(des_dir);

	printf("\x1b[8;8H    ");
	printf("\x1b[8;8H%d", err);

	speedr=speedl=100;

	if (err>90) speedr=-100;			// hard right turn
	else if (err>22) speedr=0;			// med right turn
	else if (err>0) speedr=(100-2*err);		// fine adjustment right turn
	else if (err<-90) speedl=-100;			// hard left turn
	else if (err<-22) speedl=0;			// med left turn
	else if (err<0) speedl=(100+2*err);		// fine adjustment left turn
}

void depart(int *latency)
{
	static int first_call;
	int temp=0;  //dummy var to pass to trigger first disngage

	going_home=0;

	if (!(*latency))
	{
		first_call=1;
		(*latency)=1;
	}

	if (HOME)
	{
		speedr=-100;
		speedl=-100;
	}

	else if (first_call)
	{
		 disengage(&temp);
		 first_call=0;
	}

	else disengage(latency);

}

// Disengage robot from obstacle if bumped
void disengage(int *latency)
{
	static struct time mark1, mark2;
	static int i;  // used for random numbers of various types

	// set the start times on the first trip through
	if (!(*latency))
	{
		(*latency)=1;
		i=rand(1024);

		// how long the robot backs up for
		mark1=time_addm(curr_time, 1500);

		// how long the robot turns for (semi-random 250-1024ms)
		if (i>250)
			mark2=time_addm(mark1, i);
		else
```

```
                        mark2=time_addm(mark1, 250);
        }

        // while current time is less then mark1 backup
        if (time_compare(curr_time,mark1))
        {
                speedl=-100;
                speedr=-100;
        }

        // between mark1 and mark2 turn randomly left or right
        else if (time_compare(curr_time,mark2))
        {
                if (i & 0x0001)
                /*turn left*/
                {
                        speedr=100;
                        speedl=-100;
                }
                else
                /*turn right*/
                {
                        speedr=-100;
                        speedl=100;
                }
        }

        // otherwise its time to relinquish control and start forward
        else
        {
                speedr=100;
                speedl=100;
                (*latency)=0;
        }
}


// Avoid obstacles if IR shows any
void avoid()
{
        // Use square law relationship for smooth avoidance
        // Scaled to enable a small rearward motion at max avoidance
        // speeds: -30 : +50
        speedl=100-1.3*(irdr*irdr);
        speedr=100-1.3*(irdl*irdl);
}

void spiral(int *latency, int speed)
{
        static int dir;
        static double path_length;
        static int reset;
        static int des_dir;
        static struct time mark;

        if (!(*latency))
        {
                (*latency)=1;
                dir=rand(1);   // random choice of spiral dir
                path_length=0;
                reset=0;

                mark.milli=mark.secs=mark.mins=0;
        }

        path_length+=(double)speed/100;

        // if mark<=current time set a new direction
        if (time_compare(mark,curr_time))
        {
                if ( ((dist/path_length<0.1) && (dist>10)) || reset || (dist>30))
```

```
		{
			if (dist>DIST_THRESHOLD) reset=1;
			else reset=0;

			path_length=0;

			des_dir=home_dir();
		}
		else	if (dir) des_dir=home_dir()+90;
		else des_dir=home_dir()-90;

		// Make sure des_dir is in the proper range of compass headings
		if (des_dir>=360) des_dir-=360;
		else if (des_dir<0) des_dir+=360;

		mark=time_addm(curr_time,500);
	}
	// else don't change the desired direction.


	// Set speed according to the error from des_dir
	set_speeds(des_dir);

}

// Navigates the robot towards home
void homing(int *latency)
{
	static struct time mark;
	if (!(*latency))
	{
		(*latency)=1;
		last_food_dir=home_dir();
		mark=time_addm(curr_time,200);
	}
	if (time_compare(curr_time,mark))
	{
		speedr=100;
		speedl=100;
	}
	else	set_speeds(home_dir());

	if (dist<DIST_THRESHOLD) (*latency)=0;
}

// Searches for target objects
void search(int *latency)
{
	static int how_far;
	static int direction, direction2;
	int current_direction;

	if (!(*latency))
	{
		(*latency)=1;

		// calculate search direction

		// first time either 45, 90, 135, 180, 225, 260, or 315 deg
		if (last_food_dir=999) direction=rand(7)*45;

		// otherwise go the same direction again
		else if (last_trip_successful) direction=last_food_dir;

		// or choose new direction slightly off from old
		// will be + or - 45 from old, evenly distributed
		// note this will tend towards zero deflection! need bias?
		else
		{
			direction=rand(2);
			while ((direction-1)==0) direction=rand(1);
```

```
                        direction=(direction-1)*45 + last_food_dir;
                }

                last_trip_successful=0;

                how_far=10*rand(3);

                direction2=rand(3);
                if (direction2==0) direction2=direction-135;
                else if (direction2==1) direction2=direction-90;
                else if (direction2==2) direction2=direction-90;
                else if (direction2==3) direction2=direction-135;

                if (direction2>=360) direction2-=360;
                if (direction2<0) direction+=360;

        }

        if (dist>how_far) current_direction=direction2;
        else current_direction=direction;

        // This section calculates motor speeds for given direction
        set_speeds(current_direction);
}

// Psuedo-random number generator
// Generates integers in the range of 0-top where top>2
// uses algorithm from p.122 of Discrete Mathematics and its
// Applications, by K.H. Rosen
int rand(int top)
{
        int multiplier = 2;
        int increment = 3;

        rand_seed=(multiplier * rand_seed + increment) % (top+1);

        return rand_seed;
}









void wander(int *latency, int avg_speed)
{
        static struct time straight_done, spiral_done;
        static int spiral_latency;
        struct time mark1, mark2;

        if (!(*latency))
        {
                (*latency)=1;

                // 30 sec of straight
                mark1.milli=mark1.mins=0;
                mark1.secs=30;

                // 20 sec of spiral
                mark2.milli=mark2.secs=20;
                mark2.mins=0;

                straight_done=time_addt(curr_time, mark1);
                spiral_done=time_addt(straight_done, straight_done);
        }

        // if its after mark1 then we spiral
        if (time_compare(straight_done,curr_time))
        {
                // if its after mark2 do it again from the top
                if (time_compare(spiral_done,curr_time)) (*latency)=0;
                else spiral(&spiral_latency, avg_speed);
        }
        else
        {
                speedr=100;
                speedl=100;

                dist=0;
                phi=0;
        }

}
```

ALF.H: The main #include file for the foraging programs

```
/*********************************************************
*
* alf.h
*
* Platform:  ALFs - Ant-Like Foragers, a set of modified TJ
*            mobile robots.
*
* Function:  Contains defines and includes for use in ALF programs
*
* Programmer: Jason Fleischer
* Created: April 7, 1999
*
*********************************************************/


#include <analog.h>
#include <servos.c>
#include <compass.c>
#include <alf_clk.c>
#include <serialtp.h>
#include <isrdecl.h>
#include <vectors.h>


#define BUMPER     analog(0)
#define RIGHT_IR   analog(2)
#define LEFT_IR    analog(3)

#define HOME !(PORTA & 0x0001)

#define TO_FOUND (PORTA & 0x0004)

#define FRONT_BUMP  (BUMPER>10)&&(BUMPER< 120)
#define BACK_BUMP  BUMPER>120

/*Turn the IR emitters ON and OFF*/
#define IRE_ON  *(unsigned char *)(0x7000) = 0x07
#define IRE_OFF       *(unsigned char *)(0x7000) = 0x00
```

COMPASS.C: Compass and navigation routines

```
/*************************
*
* compass.c
*
* Jason Fleischer
* created: 1/18/99
*
* Contains all the functions for using the compass on CSU's TJPros.
* The compass is the Dinsmore #1490, a physical magnetic compass using
* 4 hall effect sensors at the cardinal points.  Combinations of the
* cardinal points can give 8 points (45deg) of compass resolution.
*
* The compass is decoded using the lines from the compass as bits in
* a 4-bit number.  The back line is MSB followed by left, front, and
* right.  This becomes an index into the direction array.
*
* Users will usually call heading_error(desired), where desired is
* the magnetic heading in degrees of the desired travel direction.
* The function will return the error as up to +/- 180 degrees, which
* can then be used to steer the robot.
*
* The compass has the following lines attached to the robot:
*
* Inputs
* PE1 - right
* PE4 - left
* PE6 - forward
* PE5 - back
*
* Outputs
* PA4 - activate compass
*************************/

/*************************
* Function descriptions
*
* int read_heading() - returns mag heading in degrees.
*   inputs: -
*   ouputs: int heading in magnetic degrees
*   uses:
*   registers: PORTA, PORTE
*
* int heading_error(int desired) - returns error between current
*  and desired heading as 0 to +/- 180 degrees. Errors requiring
*  a CW correction are positive, CCW correction are negative.
*   inputs: -
*   ouputs: int heading error in degrees
*   uses: read_heading(), sign(), abs()
*   registers: -
*
* int sign(int num) - returns a unit value of the sign of num.
*   inputs: int number
*   ouputs: int unit value number
*   uses: -
*   registers: -
*
* int abs(int num) - returns the absolute value of num.
*   inputs: int number
*   ouputs: int positive numer
*   uses: -
*   registers: -
************************/

/***********************
* Revision history
*
* 1/21/99 - corrected COMP_x defines and eliminated timing from
*           read_heading().
* 1/22/99 - finished commenting.
************************/

#include <hc11.h>


// COMP_X is true when the corresponding line is high on the compass
#define COMP_RIGHT (analog(1)>128)
#define COMP_LEFT (analog(4)>128)
#define COMP_FORW (analog(6)>128)
#define COMP_BACK (analog(5)>128)

#define COMPASS_ON PORTA|=0x10
#define COMPASS_OFF PORTA&=0xEF

/* Direction vector holds degree headings lookup table
   NOTE: -1 is an indexing error!
   Yes, its inefficient but it only wastes 16bytes of mem and in
   return we get a very easy solution. */
int direction[16]={-1,-1,-1,315,-1,-1,45,0,-1,225,-1,270,135,180,90,-1};


int read_heading();
int head_error(int);
int sign(int);
int abs(int);

// returns current magnetic heading of the robot
int read_heading()
{
       int index;
       COMPASS_ON;

       /* index is a 4-bit binary number composed of the values of the
          compass's binary lines with back being MSB and right the LSB */

       index= COMP_RIGHT + 2*COMP_FORW + 4*COMP_LEFT + 8*COMP_BACK;
       COMPASS_OFF;
       return direction[index];
}

// returns the robot's navigational error 0 to +/-180 deg
// an error needing a CW correction is positive
// an error needing a CCW correction is negative
int head_error(int desired)
{
       int err=desired-read_heading();
       if (abs(err)>180)
              err=-(sign(err)) * (360 - (sign(err))*err);
       return err;
}

// returns unit number of the same sign as num
int sign(int num)
{
       if (num<0) return -1;
       else return 1;
}

// returns absolute value of num
int abs(int num)
{
       if (num<0) return -num;
       else return num;
}
```

SERVOS.C: Drive and gripper servo routines

```
/*-------------------------------
* servos.c
*
* Jason Fleischer
* created: 1/7/99
*
* Low level ICC servo handling routines for CSU's TJ-based robots.
* These robots use a 68HC11A1 processor in a Mekatronix MTJPRO board.
* Based off of the PWM algorithms in Mobile Robots: Inspiration to
* Implementation by Joseph Jones and Anita Flynn.
*
* PWM is accomplished by using Output Compare (OC) registers as timers
* to provide control for up to 4 servos on PA3-6. PA7 remains
* open as a digital output (OC1 does not fire on it) for other uses.
* Currently its setup for 3 servos on PA3,5,6; PA4 is also open right
* now.  If a fourth servo (PA4) is added follow the instructions in
* the commenting embedded in the code below.
*
* DANGER! If you add a fourth servo on PA4/OC4 the servos will
* conflict with the MTASK library!
*
* TCNT is the 16-bit timer on the 68HC11.  At the 2MHz E-clock rate it
* overflows every 32.77ms - which we'll use as the period for PWM on
* the servos. The hobby servos we're using take 1-2ms pulse width
* (2000-4000 E-clock cycles).  1ms is full CCW/-180deg., 1.5ms is
* stoped/center, and 2ms is full CW/180deg.
*
* OC1 is put in supervisory control of OC2-OC5.  OC1 fires at TCNT=0
* and sets OC2-OC5 outputs (PA6-PA3) high.  Registers OC2-OC5 are set
* to fire at TCNT=desired pulse width and turn their respective
* outputs low.  In this manner OC2-OC5 can be controlled individually
* merely by writing the desired pulse width in terms of the number of
* E-clock cycles to each individual register address.  The CPU does
* not have to service any interrupts.
*------------------------------*/


/*-------------------------------
* Function Descriptions
*
* void init_servos() - sets up all the OCs as mentioned above.
*      inputs: -
*      outputs: -
*      uses: #define'd variables from hc11.h
*      registers changed: OC1M; OC1D; TOC1,2,3,&5; TCTL1
*
* void move(int ID, int command) - changes the OC timer val for
*  a new motor command (-100% to +100%) to servo #ID.
*      inputs: #define'd hex int servo ID; int command val -100 to +100
*      outputs: -
*      uses: limit_range()
*      registers changed: TOC2,3,&5
*
* void stop() - turns off PWM
* void go() - turns it back on (NO Initialization of PW!!)
*
* int limit_range(int value, int low, int high) - forces all
*  values to stay in the range low to high.
*      inputs: integer value, low limit, and high limit
*      outputs: integer within inclusive range low:high
*      uses: -
*      registers changed: -
*------------------------------*/


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

```
* Revision history
*
* 1/11/99 - eliminated OC4 usage to maintain compatibility with MTASK.
* 1/14/99 - cleaned up code and made remaining OC4 issue changes.
* 1/15/99 - fixed gripper constants for proper PWM and parameterized
*           motor addresses in #defines.
* 1/20/99 - added stop and go.
* 2/8/99  - turned all register assignments into |= to make the routines
*           "play nice" in case other routines access those registers.
*
*------------------------------*/

#include <hc11.h>

//define servo hex adresses
#define RIGHT_SERVO 0x101E            // TOC5 - PA3
#define LEFT_SERVO 0x1018             // TOC2 - PA6
#define GRIP_SERVO 0x101A             // TOC3 - PA5

/* Uncomment and modify below if one more servo is added
#define ?_SERVO 0x101C                // TOC4 - PA4
*/


#define MAX_PWM 3800                  // not quite full CW
#define MIN_PWM 2200                  // not quite full CCW
#define CENTER_PWM 3000                       // aproximate center/stop PWM

// Multiplier to turn -100% to +100% commands into pulse widths
#define SPEED_SLOPE (MAX_PWM-MIN_PWM)/100

#define OPEN 90                               // gripper opened % PWM
#define CLOSED -90                            // gripper closed % PWM
#define OPEN_PWM (CENTER_PWM+OPEN*SPEED_SLOPE)      // raw PWM for open


int limit_range(int value, int low, int high)
{
       if (value<low) return low;
       else if (value>high) return high;
       else return value;
}

void init_servos()
{
       OC1M|=0x68;           // OC1 changes the values of PA3,5,&6
       OC1D|=0x68;           // OC1 turns on PA3,5,&6 when it fires
       TCTL1|=0xA2;          // OC5,3,&2 turn off PA3,5,&6 respectively
       TOC1=0;               // OC1 fires when TCNT=0 (every 32.77ms)

       // Variable casts to fill in starting values for the motors.
       // Assigns the number of clock cycles until the relevant
       //   OC fires and turns off its pin.
       *(unsigned short *)(RIGHT_SERVO)=CENTER_PWM;
       *(unsigned short *)(LEFT_SERVO)=CENTER_PWM;
       *(unsigned short *)(GRIP_SERVO)=OPEN_PWM;


       /* Uncomment the below statements and comment out the above
          if another servo is added.
       OC1M|=0x78;           // OC1 changes the values of PA3,4,5,&6
       OC1D|=0x78;           // OC1 turns on PA3,4,5,&6 when it fires
       TCTL1|=0xAA;          // OC5-2 turn off PA3-6 respectively
       TOC1=0;               // OC1 fires when TCNT=0

       // Variable casts to fill in starting values for the motors.
       // Assigns the number of clock cycles until the relevant
       //   OC fires and turns off its pin.
       *(unsigned short *)(RIGHT_SERVO)=CENTER_PWM;
       *(unsigned short *)(LEFT_SERVO)=CENTER_PWM;
       *(unsigned short *)(GRIP_SERVO)=OPEN_PWM;
```

```
        *(unsigned short *)(?_SERVO)=CENTER_PWM;  */

}

void move(int ID, int command)
{
        command=limit_range(command,-100,100);       // ensure OK command value

        /* since the left motor turning CW would be backwards
               from the right motor we need to make sure
               that the commands are reversed */
        if (ID==LEFT_SERVO) command=-command;

        command=CENTER_PWM + SPEED_SLOPE*command;    // calc PW in clock cyc.
        *(unsigned short *)(ID)=command;             // enter PW in correct OC register

}

void stop()
{
        OC1D &= 0x10;  // OC1 won't turn on PA3,5,&6

        /* Uncomment the below statements and comment out the above
           if another servo is added.
        OC1D = 0x00;           // OC1 turn on PA3,4,5,&6 */
}

void go()
{
        OC1D|=0x68;            // OC1 turns on PA3,5,&6

        /* Uncomment the below statements and comment out the above
           if another servo is added.
        OC1D|=0x78;            // OC1 turn on PA3,4,5,&6 */
}
```

ALF_CLK.C: Timing and time-keeping routines

```
/***********************************************************************
 *                                                                         *
 * Title        alf_clk.c                                              *
 * Programmer   Jason Fleischer                                        *
 * Date         Feb 19, 1999                                           *
 * Version      1.3                                                    *
 *                                                                     *
 *                  BASED off of Mekatronix clocktjp.c                 *
 * Description                                                         *
 *                                                                     *
 *   MOD for v 1.1: uses OC4, MOD for v 1.2 time struct and assc fcts   *
 *     This file contains all routines and interrupt drivers               *
 *      necessary for TJ clock control. The clock keeps track of       *
 *      milliseconds, seconds, minutes                                 *
 **********************************************************************/


/*************************** Includes *********************************/
#include <hc11.h>
#include <mil.h>


#define CLOCK_TICK 2000  // 2000 clock cycles = 1ms

/**************************** Globals *********************************/
struct time {
       unsigned int milli;
       unsigned int secs;
       unsigned int mins;
};

struct time curr_time;

unsigned int loc;



/************************** Prototypes ********************************/
#pragma interrupt_handler clock_isr
void init_clock();
void clock_isr(); /*Interrupt service routine to generate clock ticks*/
void pause(unsigned int);   /* Wait for int milliseconds */
int time_compare(struct time t1, struct time t2); /* returns 1 for t1<=t2 */
struct time time_addm(struct time t1, unsigned int milliseconds); /* t1 + msec */
struct time time_addt(struct time t1, struct time t2); /* t1 + t2 */
unsigned int msec_diff(struct time t1, struct time t2);
void reset_clock();



void init_clock()
/***********************************************************************
 * Function: Initializes all necessary variables for the timer.        *
 * Returns: None                                                       *
 *                                                                     *
 * Inputs                                                              *
 *   Parameters: None                                                  *
 *   Globals:    None                    *
 * Outputs                                                             *
 *   Parameters: None                                                  *
 *   Globals:    None                    *
 *                                                                     *
 * Registers: HPBRIO, TOC4,TMSK1                              *
 * Functions called: None                                              *
 * Notes: This routine must be executed to enable the clock.           *
 ***********************************************************************/
{

  INTR_OFF();

  // The followings statements modify the normal TJ interrupt vectors
  // mem location in order to have clock_isr service OC4 interrupt
  loc=(unsigned int)(&clock_isr);
  asm("ldd %loc\n"
        "std 0xffe2");

  SET_BIT(HPRIO,0x0E);       /* Promote OC4 interrupt to highest */
  CLEAR_BIT(HPRIO,0x01);

  curr_time.mins=0;
  curr_time.secs=0;
  curr_time.milli=0;
  TOC4 = 0;

  SET_BIT(TMSK1,0x10);         /* Enable OC4 interrupt */
  INTR_ON();
}



void pause(unsigned int msec)
/*********************************************************************
 * Function Description: Busy wait for msec milliseconds             *
 * Returns: None                                                     *
 *                                                                   *
 * Inputs                                                            *
 *   Parameters: msec, the time to pause.                            *
 *   Globals:    curr_time                                           *
 * Outputs                                                           *
 *   None                                                            *
 * Functions called: None                                            *
 * Notes: unsigned int up to 65355, therefore waits for up to 65 secs!*
 *********************************************************************/
{
       struct time mark=time_addm(curr_time,msec);
       while (time_compare(curr_time,mark));
}



void clock_isr()
/***********************************************************************
 * Function:                                                           *
 *   Interrupt service routine to generate clock time variables. This  *
 *   isr executes every millisecond and increments the global variables *
 *   msec, seconds, minutes, hours, days appropriately.                *
 *                                                                                 *
 * Returns: None                                                       *
 *                                                                     *
 * Inputs                                                              *
 *   Parameters: None                                                  *
 *   Globals:    curr_time, timer                                      *
 * Outputs                                                             *
 *   Parameters: None                                                  *
 *   Globals:    curr_time, timer                                      *
 *                                                                     *
 * Registers:  TOC4, TFLG1                                                           *
 * Functions called: None                                              *
 * Notes: None                                                         *
 ***********************************************************************/
{
   TOC4 += CLOCK_TICK;     /* Set the timer to interrupt again in 1msec. */
   ++curr_time.milli;               /* Increment the msec timer */
   if (curr_time.milli == 1000) {curr_time.milli = 0; ++curr_time.secs;};
   if (curr_time.secs == 60) {curr_time.secs = 0; ++curr_time.mins;};
   CLEAR_FLAG(TFLG1,0x10);          /* Clear OC4I flag */
}




int time_compare (struct time t1, struct time t2)
/************
* Function: Returns 1 if t1 <= t2.  Otherwise returns 0.
*
************/
{
	if (t1.mins > t2.mins) return 0;
	if (t1.secs < t2.secs) return 1;
		else {
			if (t1.secs > t2.secs) return 0;
			if (t1.milli < t2.milli) return 1;
			else {
				if (t1.milli > t2.milli) return 0;
				else return 1;
			}
		}
}


struct time time_addm(struct time t1, unsigned int milliseconds)
/************
* Function: adds milliseconds to time struct and returns sum time struct
*
************/
{
	struct time sum;


	sum.milli= t1.milli + milliseconds;
	sum.secs= t1.secs;
	sum.mins= t1.mins;

	while (sum.milli > 1000) {sum.milli-=1000; ++sum.secs;};
	while (sum.secs > 60) {sum.secs-=60; ++sum.mins;};

	return sum;

}

struct time time_addt(struct time t1, struct time t2)
/************
* Function: Adds two time structs and returns time struct sum
*
************/

{
	struct time sum;

	sum.milli= t1.milli + t2.milli;
	sum.secs= t1.secs + t2.secs;
	sum.mins= t1.mins + t2.mins;

	if (sum.milli > 1000) {sum.milli-=1000; ++sum.secs;};
	if (sum.secs > 60) {sum.secs-=60; ++sum.mins;};

	return sum;

}


unsigned int msec_diff(struct time t1, struct time t2)
/************
* Function: Returns millisec time diff t1-t2.
*
* Note: assumes not more then 1 min diff
```

```
************/
{
        return (1000*(t1.secs - t2.secs) + (t1.milli - t2.milli));
}

void reset_clock()
{
        curr_time.milli = curr_time.secs = curr_time.mins = 0;
}
```