IMPLEMENTATION OF ADAPTIVE BEHAVIORS IN A SIMPLE INSECT-LIKE ROBOT

by MARK LANTHIER, B.C.S.

in partial fulfillment of the requirements for

the degree of Master of Computer Science

September 21, 1993

Carleton University

ABSTRACT

Recent approaches towards designing autonomous robots have concentrated on the bottom-up approach by programming them with instinctive behaviors in which the overall behavior of the robot emerges from the interaction of the robot with the environment. This thesis describes how these instincts can be coded using simple fixed weight neural networks which reduce the need for computational speed and power. Furthermore, with these simple low level behaviors, only simple sensors are required. The resulting robot is therefore simpler, smaller and cheaper. The potential of simply constructed robots that use a minimal complement of basic sensors is investigated. A hardware and simulated version of a robotic insect is presented which have the capability of adapting to a static, initially unknown environment. The hardware version was developed to determine if such a robot could operate in the unpredictable and noisy real world. Various problems were encountered during its development that points out the importance of developing a physical device. The robot is endowed with instincts pertaining to obstacle avoidance, wandering, vacancy and edge following. Instincts from the simulated version also include photokinesis, floor cleaning and landmark-based navigation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PHOTOGRAPHIC PLATES

~~~~~~

Chapter 1
Artificial Life Through Robotics

~~~~~~

There has been a growing amount of research in the area of autonomous mobile robots and *animats* [1]. Much of the work has been the designing of task-oriented mobile robots many of which use sophisticated sensors (i.e. cameras, range finders) and require much computational speed and power. While this approach seems to create robots that are able to impressively perform simple tasks, it usually results in a highly complex and expensive system that can only operate in a constrained environment. In addition, the sophisticated sensors suffer from noisy data, resulting in the need to alter the environment to reduce noise [2].

By creating robots with simple sensors, the noise factor is sufficiently reduced allowing the robot to function in an unaltered environment. The drawback of simplifying the sensors is that the information extracted from the environment is reduced and the resulting robot would only be able to perform simple tasks. However, the elimination of complex sensors allows the robot to be reduced in size since the computational power requirements diminish. With such a small size, perhaps many of these simplified robots could perform as a whole the same tasks as a more complex robot but at a fraction of the expense. Nanotechnology research may eventually lead to efficient nano motors and pumps, allowing these simplified robots to become reality [Flynn 87].

Although there has been some recent research in the area of colonies of nanobots [Dario et al. 91], [Lewis and Bekey 92], there is a need to determine just what types of complex

[1] The term animat is used to describe an artifical animal ; in this case, a simulated robotic life form.

[2] In the case of laser range finders, ambient light from outside the environment must be blocked off. For cameras, the obstacle contrasts must be altered in order for the camera to single out the desired objects. An example of this is creating a black and white environment.

behaviors can emerge from these simple robots individually. By experimenting with simplified robot control mechanisms and sensors, one may gain insight as to their usefulness.

The first step towards creating these individual robots is to determine the basic primitive behaviors that are needed for survival. At the lowest level of control, the robot can be programmed with simple behaviors such as obstacle avoidance, wandering, edge following, phototropism, photophobia and searching for food. Additional behaviors could then be added allowing the robot to learn and adapt to its environment as well as perform a repertoire of simple tasks. A robot of this nature takes precedence in keeping itself functioning by learning to adapt and survive in its environment, performing its given tasks only when its needs are met. Such a robotic system would represent a kind of mechanical artificial life form.

1.1 Traditional Robotics

Many researchers have taken a common approach towards designing so-called "intelligent" autonomous robots and robotic systems. Many of their robots were developed in a straight forward manner with software that instructed the robot to perform some action whenever a specific event had occurred. In this approach, the sensor data is fused into an internal model of the environment. The robot would often have a set of rules or strategies which completely governed its overall actions. The internal model was then used to reason about and plan intelligent actions. A main problem with using such an internal model of the environment is in the representation. The world must be represented with enough information to allow reasoning. When there is a lack of information, the reasoning and decision making process may not come up with an appropriate response for a given event. The need for accurate representations resulted in the need to increase sensor power. The addition of more complex sensors further required a large quantity of processing power in order to process the information.

The traditional programming approach used what is known as a top-down strategy in which a main controller was responsible for the overall actions performed by the robot. With this approach, it was difficult to add functionality to the robot since it involved making changes to the existing software and/or hardware. Moreover, by adding additional features to the robot, there was often a need to increase the computational power since in order to keep the robot operating with a reasonable response time. Due to the need for processing sensor data, this type of design is of no practical use in situations that demand a quick response time.

The computers on-board these traditional robots were often required to make complex calculations to solve problems related to 3D transformations, data analysis and extrapolation, and noise reduction techniques. In addition, the algorithms demanded the computer' s processing power for computational geometry and decision making problems as well as positioning and control for the robot' s actuators. During the development of such systems it was often the case that the computer could not meet the real-time demands of the control software, leaving the designers with the choice of increasing the computational power or reducing the computational requirements of the system. The latter solution leads to a reduction in functionality of the overall system.

Usually, as the robot went through its many design phases, the final version ended up being quite complex and loaded with massive computational hardware and sophisticated sensors. Needless to say, these robots were usually bulky, and heavy and often contained software and hardware problems that were difficult to predict ahead of time. Thus, the resulting robot was expensive, complex and often not very robust.

Perhaps the biggest problem was that of developing algorithms with the ability to handle the many different situations that could be encountered. Often the algorithms were improved upon to account for different situations, but this improvement often resulted in quite complex algorithms which were more difficult to code. Moreover, these algorithms could only be improved upon up to a certain point before they become too complex to implement in a robotic system.

Despite the many shortcomings, this traditional top-down approach to programming was used for many years. Perhaps the reasoning for this was that computers were becoming smaller, quicker, and cheaper, leading researchers to believe that eventually the computer would be quick enough to handle even the most demanding computations. But as these robots became larger, heavier and more heavily equipped with sophisticated sensors, it was clear that another approach had to be taken towards the design of autonomous systems.

1.2 The Subsumption Architecture - A Different Perspective

After a while, some researchers began to ask a simple question: "How could it be that a simple ant could outperform these highly complex and expensive robots ?". After all, an ant is equipped with only simple sensors and little or no computational power at all. Clearly, there was a problem with the traditional approach to robot design since their performance did not meet their expectations. Furthermore, there is a growing suspicion that the traditional approach to intelligent behavior is inadequate for systems that must operate in realistic environments, since they involve explicit reasoning by manipulating symbolic representations of the world [Beer 90].

One researcher in particular decided to make a radical change towards the design of robotic systems. This researcher is Rodney Brooks. In 1985-1986 R. Brooks published papers describing a radically new architecture for use in designing autonomous systems. This architecture is known as the *subsumption architecture* [Brooks 86]. This new architecture emphasizes a more direct coupling of sensors to actuators. It is a distributed and decentralized structure that provides a more dynamic interaction with the environment. Moreover, by having multiple parallel activities, and by removing the idea of a central representation, there is less chance that any given change in the class of properties enjoyed by the world can cause total collapse of the system [Brooks 91].

The subsumption architecture (SA) represents a bottom-up approach to the design of intelligent robotic systems. This meant that the robot was designed by building the simplest reactive components first, and then adding the higher level functioning. The SA allows the robot to be developed one piece (level) at a time such that after each level is completed, the robot is able to perform some simple behavior. Additional levels are added only when it is fairly certain that the existing layers are operating properly. With the SA, the behaviors are programmed in a priority oriented fashion in which higher levels of behavior subsume the lower levels. Thus the higher level behaviors can override or suppress the signals in the lower level behaviors. These higher levels of behavior correspond to higher levels of competence. The SA represents a parallel and distributed structure for connecting sensors to actuators as shown in Figure 1.1.

The architecture allows for incremental development, where additional layers can be "added on" to the existing structure without the need to modify the previous layers. The SA concentrates on getting the robot to function in a simple reactive manner, adding additional functionality on top of the simple behaviors. The result is a more robust and reactive system than with the traditional top-down approach. Furthermore, with the SA, there is no need for decision making as to when certain behaviors (actions) should be performed since it is automatically done through the subsumption process. This allowed the system components to be separated, reducing the computational requirements of the overall system.

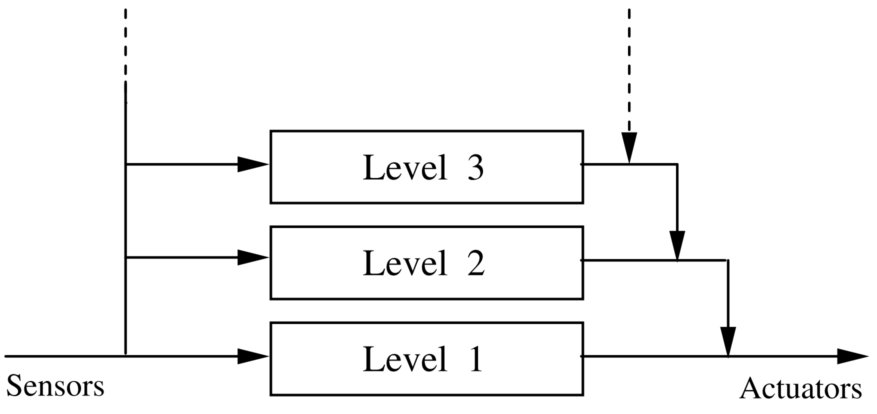


Figure 1.1 The subsumption architecture. Higher levels subsume lower levels when they wish to take control.

The main drawback of this approach is that there is no learning component. Traditional approaches, although far too complex and bulky, at least had the ability to learn. As it turns out, the SA shares a similarity to the notion of "instinctive behaviors" since they are built-in to the robot and no learning is required. Some say that the subsumption architecture is limited in that only simple behaviors can be implemented and that the approach could never have the ability to perform complex tasks that require reasoning.

The SA approach, although limited in its learning and task abilities, can lead to a robotic system that closely resembles a simple life form. As seen by observing ants, even a simple life form without the ability to learn or reason can perform rather well. Moreover, through cooperation, more complicated behaviors emerge from the system as a whole. It is the author' s opinion that the SA approach represents a significant leap towards the development of "intelligent" robots and robotic systems.

1.3 Artificial Life

Can a robot be created that simulates the life of a simple biological organism ? This question is not easy to answer. In fact, there are many different aspects that must be considered in the creation of artificial life. Before answering this question, one must have an understanding as to what artificial life represents.

Artificial Life (A.L.) is the study of man-made systems that exhibit behaviors characteristic of natural living systems [Langton 89]. It attempts to capture the behavior of the components of a living system and to endow artificial components with similar behaviors. If the artificial parts and behaviors are organized correctly, then the artificial system should exhibit the same dynamic behavior as the natural system. The approach of A.L. is not concerned with building systems that reach some sort of solution. For these systems, the ongoing dynamics is the behavior of interest, not the state ultimately reached by that dynamics. Traditional task oriented robots would be discarded from the area of A.L., since the resulting task performance is the only topic of interest. In essence, they are built as "slaves" with no desire for survival.

The key concept in A.L. is emergent behavior. Natural life emerges out of the organized interactions of a great number of non-living molecules, with no global controller responsible for the behavior of every part. Rather, every part is a behavior itself, and life is the behavior that emerges from the local interactions among individual behaviors. It is this bottom-up, distributed, local determination of behavior that artificial life employs in its primary methodological approach to the generation of life-like behaviors [Langton 89]. This bottom-up approach favors the subsumption architecture since simple behaviors can be created separately, allowing the more complicated behaviors to emerge.

1.3.1 Linear Vs. Nonlinear systems

Linear systems are those for which the behavior of the whole is just the sum of the behavior of its parts. Linear systems obey the superposition principle in that by studying the parts in isolation, we can learn everything we need to know about the complete system. *Nonlinear* systems on the other hand, do not obey the superposition principle. In these systems, the primary behaviors of interest are properties of the interactions between parts, rather than being the properties of the parts themselves, and these interaction-based properties necessarily disappear when the parts are studied independently.

Behaviors themselves can constitute the fundamental parts of nonlinear systems (virtual parts) which depend on nonlinear interactions between physical parts for their very existence. By programming a robot with many simple behaviors, complex behavioral patterns can emerge through the interaction of the robot with its environment.

1.3.2 Local Vs. Global Behavior

There are two basic approaches to creating a nonlinear robotic system. These are through *local* and *global* specification. With local specification (subsumption architecture), each basic component of the system has a set of local rules that determine its behavior. With global specification (traditional robotics), there is one set of rules that governs the behavior of all the components as a whole. It is easier to generate complex behavior from the application of simple local rules than it is to generate complex behavior from the application of complex global rules. This is because complex global behavior is usually due to nonlinear interactions occurring at the local level. With bottom-up specifications, the system computes the local, nonlinear interactions explicitly and the global behavior (which was implicit in the local rules) emerges spontaneously without being treated explicitly.

With top-down specifications, however, local behavior must be implicit in global rules. The global rules must "predict" the effects on global structure of many local, nonlinear interactions - something which we have seen is intractable, even impossible in the general case. Thus, top-down systems must take computational shortcuts and explicitly deal with special cases, which results in inflexible, brittle and unnatural behavior.

Furthermore, in a system of any complexity the number of possible global states is astronomically enormous and grows exponentially with the size of the system. Systems that attempt to supply global rules for global behavior simple cannot provide a different rule for every global state. On the other hand, systems that supply local rules for local behavior can provide a different rule for each and every possible local state. In addition, the size of the local state-space can be completely independent of the size of the system. The only special cases explicitly dealt with in locally determined systems are exactly the set of all possible local states, and the rules for these are just exactly the set of all local rules governing the system.

Consider modeling a colony of ants. One could create many instances of different classes of ants such that each class of ants has its own unique behavior. One could then start up a simulation of a simple 2-D environment by specifying an initial configuration of these classes of ants. Once started, the behavior of this system would depend entirely on the collective results of all the local interactions between individual ants and between each ant and the environment. There would be no "drill-sergeant" ant choreographing the ongoing dynamics according to some sort of high-level rules for colony behavior. The behavior of the colony of ants would emerge from the behaviors of the individual ants themselves, just as in a colony of biological ants.

1.4 Adaptive Behavior

Part of the definition of life itself is that the organism must have some way of finding and absorbing food. Many life forms have adapted to their environments through genetic selection, although there may be some degree of adaptability during their life cycle. This brings up the notion of adaptability in robotic behaviors. *Adaptive behavior* is behavior which is adjusted to environmental conditions. A robot with this type of behavior must be able to react in some appropriate manner to these changing conditions such that it remains functional. Perhaps, this adaptation may correspond to the fine-tuning of behaviors to suit the current environment, thus increasing the efficiency of the robot over time.

Learning is the process by which behavior remains adaptive throughout an agent's[3] life in the face of a non-stationary environment [Beer 90]. Conversely, if a robot is functioning in a non-changing environment, then the process of learning is not always necessary. Initially, the robot will have learnt the environmental conditions and since they are non-changing, the learning process will diminish since the robot has learned all it needs to know about the environment to function adequately. There is an interlocking of both learning and adaptive behavior since each relies on the other. It can be said that adaptive behavior is the direct result of learning.

A robot functioning in an environment may exhibit various forms of behavior. In fact, a short sequence of simple behaviors can sometimes appear as a complicated behavior. Researchers [Brooks 86] [Beer 90] agree that insects exhibit very simple *reactive* behaviors which interact to realize a complicated sequence of actions. These reactive behaviors take the form of either reflex behaviors or taxes. A *reflex* is a fast stereotyped response triggered by a particular class of environmental stimuli. The intensity and duration of the response is entirely governed by the intensity and duration of the stimulus [Carew 85]. *Taxes* on the other hand involve the orientation of an animal toward or away from some environmental stimulus such as light, gravity or chemical signals [Camhi 84].

[3] The term agent is synonymous with robot.

1.4.1 Types of Reflexes

Most reflexes are used for avoiding, escaping or minimizing the effects of noxious stimuli. The immediacy of reflexes, and their relative independence of the animal' s past history, make them easy to study. [Staddon 83] states that a reflex is the name for the properties of the relation between stimulus and response, and that these reflexes incorporate seven main properties:

1) *Threshold* : The minimum stimulus level required to elicit a reflexive response. (This may depend on the animal' s state of attention, motivational state and its past history).

2) *Latency* : The time between stimulus onset and the occurrence of the reflex response. (This may depend on stimulus intensity where a more intense stimulus elicits a more rapid response. The strength as well as the speed of most reflex responses is directly related to stimulus intensity).

3) *Refractory Period* : A brief period of time in which the threshold of a reflex is elevated after the reflex has occurred.

4) *Temporal Summation* : The situation in which two sub threshold stimuli, spaced closely in time, excite a reflex but separately they elicit no response. (A cat, for example, will turn towards a sound, looking at the direction of the sound with its ears pricked forward waiting for additional stimuli before responding).

5) *Spatial Summation* : The situation in which two individually sub-threshold stimuli excite a response when presented together. (A dog for example, may ignore a single itch but additional itching stimuli around the same area will result in a scratching reflex).

6) *Momentum* : The measure of time in which the reflex continues its excitation once the stimulus has been removed. (The scratch reflex is an example of a reflex with momentum).

7) *Habituation* : The decrease in vigor and eventual cessation of a response due to repeated elicitation of a reflex response.

Automatic variation in the strengths of these properties gives the organism a kind of adaptation to the environmental stimuli. In a sense, a higher level organism can become familiar with certain types of stimuli which may lead to more efficient reactions. For simple low level organisms incapable of learning, the strengths of these properties are usually fine tuned from the genetic process. This fine tuning represents a kind of evolutionary adaptation. The changing of reactions as a function of time also represents a form of adaptability; habituation and warm up effects are examples.

In addition to the properties mentioned, there are three principles of reflex interaction:

1) *Reciprocal inhibition* : The competition of incompatible reflexes for control.

2) *Cooperation* : A blending response of two or more reflexes when they are simultaneously excited.

3) *Successive Induction* : The sequential excitation of reflexes in which the former reflex induces the later.

It is these principles of interaction that determine which reflexes have control of the organism at any one time. This is closely related to the aspect of motivation and behavior selection which is discussed further in chapter 7.

1.4.2 Types of Taxes

Many forms of life posses either no, or only the most rudimentary nervous system. With their simple construction they are somehow able to survive by finding food and a proper habitat. Thus, orientation mechanisms (taxes) exhibit the properties of adaptive behavior in their clearest form.

The simplest example of an orientation mechanism is that of climbing plants. Plants grow vertically and seek the highest best-lighted point, unless there are other factors such as obstacle avoidance, predators and wind effects which may inhibit this process. This behavior can be explained by the combined effects of *negative geotropism* (orientation away from gravity), *circumnutation* (winding orientation) and *positive phototropism* (orientation towards light).

Another form of orientation is that of *chemotaxis*, which represents the ability to orient towards chemical stimuli. Bacteria use this type of orientation mechanism to move up and down chemical gradients. Since these bacteria have only simple chemical sensors and thus the only way to find a chemical source is by means of hill climbing. The bacteria are required to use kinetic orientation in that they must make successive comparisons.

[Staddon 83] distinguishes taxic reactions as being one of 4 main types:

1) *Klinotaxis* : The use of successive comparisons to orient towards a stimulus.

2) *Tropotaxis* : The outcome of a balancing process; the animal turns until the two receptors are equally stimulated and then proceeds forward.

3) *Telotaxis* : The orientation between two symmetrically disposed stimuli by directly approaching one of the stimuli. Here the animal is somehow able to identify the bearing of the stimuli.

4) *Light-compass reaction* : Maintaining a fixed angle between the path of motion and the direction of the stimulus.

These classes represent the basic types of orientation strategies found in animals. These types of orientation specify the basic underlying mechanisms of simple orientation reactions. There are many simple orientation reactions observed in animals; some of them being:

- Attraction to light or dark (skototaxis) areas and objects,
- Attraction to warm areas,
- Postural reactions to light and gravity (geotaxis),
- Reactions to physical contact (seeking out and conforming to crevices and corners),
- Reactions to fluid flow (rheotaxis),
- Reactions to chemical (chemotaxis) and humidity gradients .

These taxes allow the animal to orient towards certain stimuli while avoiding others. It allows the animal to seek out safe locations and food; the necessities for survival. These orientation mechanisms represent survival instincts and thus are a form of adaptive behavior. In some taxes, such as the Mysis crustaceans in the Naples Aquarius, periodic taxic reversals provide an additional degree of adaptability.

1.4.3 Integrating Behaviors

Despite the relative simplicity of each reaction analyzed in isolation, when combined, they can lead to complicated, "intelligent" behavior. [Staddon 83] states the following:

> "The function of reflexes is the integration of behavior, which would be impossible without well-defined rules of interaction".

Although reflexes and taxes can provide a repertoire of reactive behaviors, it is the interaction of these reactions that provides an integrated overall behavior of the organism. Thus, one must specify rules of interaction in order to integrate the reactions and produce complex behaviors. Cooperation and competition represent the two kinds of combination rules of behavior and these will be discussed in a later chapter. With a set of such well-defined reactions, one might think that the behavior of an organism is somewhat predictable; but the behavior is often unpredictable. One of the aspects of this unpredictability is due to the variability in behavior. In some cases, the variability is intrinsic to the mechanism and serves as a function of random sampling of the environment. A degree of variability will also prevent the organism from getting trapped in certain situations; like corners, local maxima and minima, and continuous circling. An example of this can be seen when a fly is observed buzzing at a window. The fly does not utilize a systematic searching strategy to find a way out, instead a degree of randomness is used which may eventually find an opening.

For conflicting reactive mechanisms, the animal must choose between the two reactions or issue a compromise. If this were not so, then when faced with a stimulus requiring one of many responses, the animal' s reactive behaviors would be in continuous conflict. As a result the animal would not react appropriately, leading to trappings, circulation or even fatality. A degree of variability allows the animal to avoid these conflicting situations. The protozoan Euglema, for example, is photo-positive in weak light and photo-negative in strong light. Consequently, these animals congregate in intermediate levels of illumination; this may represent a compromise between the bright light that provides energy and the dim light that provides greater protection from predators [Straddon 83].

There may also be a degree of systematic variation in which the rules of interaction depend on contextual variables such as the time of day, condition of light, presence of other animals, etc. In each situation the animal' s behavior may be predictable, yet the animal may be sensitive to a variety of situations resulting in a large repertoire of behaviors.

Reflexes and taxes both depend only on recent events which limits their overall usefulness towards adaptable behavior. Reflexes are further limited in that they are not readily modified by their consequences; they are almost independent of feedback. If an animal or robot is to be able to adapt to its environment, it must be able to cope with different situations which may require different reactions in different circumstances. With the ability of an animal to learn, adaptation can be enhanced allowing the animal to function more efficiently and possibly avoid fatal situations.

This learning ability plays a large role in higher level decision making to such an extent that past experience may drastically alter the previous functioning of the robot. This may pertain to paranoia, eagerness, reduction in curiosity, etc. Nevertheless, this learning ability is required for adaptive behavior since it may be crucial for survival.

1.5 Why an Insect Model ?

Animals are naturally able to adapt their behaviors to the environment in which they are embedded. As mentioned previously, through evolution, these behaviors are fine-tuned to suit their natural environment. Of all animals, insects perform amazingly well considering their small size. Their biological control systems are versatile and robust. These simple creatures have adaptive control systems that allow them to walk in complex terrains and even upside down. Insects are also able to adapt while coping with sensor damage and leg amputations [Graham 85].

In general, insects can have an instinctive ability for cooperation. Ant and bee colonies are rather efficient due to the cooperative efforts of the individuals. These cooperative efforts are instincts which are embedded into the control mechanisms of each individual. An example of such a mechanism is the receptors on a worker ant which allows them to follow a chemical residue left behind from other ants.

A colony of small cooperating robots would closely resemble a colony of biological insects in the way they function individually and the way they function as a whole. It makes sense to model each of these individual robots as an insect since they are biologically similar to a natural insect.

1.6 Problems With Simulated Robots

Many researchers have developed simulations of robots and animats. While their experiments clearly show that simple life forms can be mimicked by a computer, they are limited to the domain of their specialized "ideal" environments. Often, the designer creates a simplified environment such that the robot' s simulated sensors produce accurate data and the actuators always perform to specifications. This is hardly the case in the real world. To help validate their simulated experiments, some researchers have attempted to program errors into sensor readings by simulating noisy data. This attempts to bring the research a step closer to the development of real robots but it is never possible to accurately simulate the effects that are caused by the vast randomness of the physical world.

[Brooks 91] has concentrated his research on designing actual robots. His robots are created incrementally; at each step letting them loose in the real world. By doing this, the robots are designed to perform in an environment as opposed to altering the environment to suit the robot. He states:

> "When we examine very simple level intelligence we find that explicit representations and models of the world simply get in the way. It turns out to be better to use the world as its own model".

Although simulated robots and animats may provide insight as to the interaction of various behaviors and mechanisms, they would likely fail in the noisy and unpredictable world of reality.

1.7 RABI

This thesis presents a robot **RABI** (Robotic Adaptive Behavioral Insect) which was developed to investigate the usefulness of robots with simple sensors. It is important to study the usefulness of robots equipped with minimal control circuitry and minimal sensors so that future robots could be developed simpler and smaller. By using a similar architecture to [Brooks 86] and a simple control structure based on [Beer 90], a simplified robot can be created that exhibits similar behaviors to that of biological insects.

RABI represents a simplified robotic insect capable of learning in a static, yet initially unknown environment. There is both a hardware and a software version of RABI [4], both using touch sensors in the form of antennae and motor actuators to move the legs. The software version also contains a sensor and motor to detect a disk (used as a marker) which can be dropped and picked up by the robot. The hardware version was built to determine if such a robot could indeed function in a real world environment. As was expected, the walking mechanisms of the simulated robot had to be altered to control the hardware robot due to the unforseen nature of the robot' s actuators and sensor readings.

In addition, the software version has simulated taxic sensors to detect light areas and energy sources. Robots do not eat, instead they must consume some sort of electrical energy (battery charging) or light (solar powered). The energy sensor may be thought of as a sensor that detects power surges so that the robot would be able to find wall sockets and other types of electrical outlets. In outdoor environments, perhaps the only source of energy would be the sun. Here, the robot must be equipped with solar cells to extract energy from the light.

Chapter 2 discusses the issues involved in modeling a biological system. The remaining chapters discuss the implementation of RABI. Chapter 3 discusses the walking mechanisms. Chapter 4 discusses the programmed instinctive behaviors of the robot. Chapters 5 and 6 deal with the map learning and navigation aspects of RABI. Chapter 7 discusses the motivational aspects involved with behavior selection and Chapter 8 describes the hardware design of RABI.

1.8 Summary

Since *artificial life* is concerned with generating life-like behaviors, a good place to start is to identify the mechanisms by which behavior is generated and controlled in natural systems, and to recreate these mechanisms in artificial systems. Traditional approaches to robot design attempted to create a complex robot capable of performing some simple task using sophisticated reasoning about sensor data. This approach led to complicated, expensive and brittle robots that were not at all reactive. [Brooks 86] pioneered an approach using the subsumption architecture that utilized a bottom-up approach to programming. This new approach allowed for incremental development of a reactive-based robot capable of performing with levels of capability. Robots using this architecture are simpler, cheaper and more robust than the traditional approach.

[4] Chapter 8 discusses the design phase of the hardware version of RABI. The simulated version is discussed throughout the thesis.

By combining techniques of [Brooks 86] and [Beer 90], a robot can be developed that exhibits simpler reactive behaviors. These simple behaviors take the form of reflexes and taxes which provide the survival instincts of the robot. The more complex adaptive behaviors emerge from the interactions of these simple behaviors with the environment. By simplifying individual behaviors, the need for sophisticated sensors can be eliminated, resulting in a smaller, cheaper, and simple robot.

~~~~~~

Chapter 2
Modeling a Biological System

~~~~~~

The task of modeling a biological system is not at all trivial. [Linsker 90] states the following: "Unlike conventional computer hardware designs, neural circuitry is not hard-wired or specified as an explicit set of point-to-point connections. Instead, it develops under the influence of a genetic specification and epigenetic factors both before and after birth". This statement implies that the functionality of a neural network model arises from the interconnections which are learnt, rather that explicitly programmed. Thus, standard programming techniques do not provide a biologically feasible solution to the model being developed. Another strategy must be used to allow the model to develop, through learning, into a functional system.

Biological development processes are far too complex to warrant a complete understanding of the organizational and computational aspects of biological neural networks. In order to understand such a complex computational instrument, it helps to create simplified models which mimic various parts of the whole. Some feel that it may eventually be possible to combine the various parts to create a device capable of performing similar tasks to that of a biological system. By making simplifications and assumptions to the operations involved with biological neural networks, a complete understanding of the simplified model can be ascertained.

As stated by [Rosenblatt 58], if we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed or detected by the biological system ?
2. In what form is the information stored, or remembered ?
3. How does information contained in memory, influence recognition and behavior ?

The answer to the first question is fairly well understood compared to the other two. This may be due to the fact that the mechanisms related to sensory input have certain construction and connections that are readily visible, allowing a more complete study. The other two questions are not so easily answered. Biological memory is quite complex. It is capable of remembering detailed information while generalizing and matching to previous experiences. Amazingly enough, this detailed information is able to be extracted in an instant. Memory organization and operation is a baffling yet prominent area of research. Unfortunately, this area is beyond the scope of this thesis.

2.1 Artificial Learning

Due to genetic selection, animals are born with a certain degree of instinctiveness. Thus the animal is born with a set of associations between environmental signals and actions that will lead to satisfaction of its needs [Wilson 87]. The rest of the associations are learned through experience. For an "intelligent" autonomous robot (animat) one possible method of representing these associations is in the form of condition-action rules. Here an action is performed whenever its condition is met. It would be useful for the animat to use its past experience to update its set of rules. The learning of simple rules of behavior (i.e. associations) represents a kind of adaptability required for artificial life forms.

The learning problem is difficult because useful information is hard to obtain. First of all, environmental events relevant to particular rules come in arbitrary order and one by one. Thus, a truly "intelligent" animat must have some means of extracting the relevant cues in order to develop its rules. Secondly, there is no "teacher" indicating when the animat has performed the correct action in each situation. Thus, the animat must be self-taught by somehow rating its actions according to the degree of satisfaction it receives after they are performed. The term *payoff* is used to describe this rating. The payoff constraint actually poses another problem, since the animat may not be rewarded immediately after an action has been issued. It may take a sequence of correct actions in order to receive a payoff, which may come only after the last action is performed. This sequence of correct movements can be called "stage-setting". Thus, the animat should be able to learn under a payoff that may be delayed. Finally, since the environment is diverse, there may be many similar situations that call for similar actions. Thus, the animat should be able to discover significant combinations of the features, generalizing them as much as possible.

There are three popular approaches that represent artificial learning systems. These are *artificial neural networks* , *genetic programming* and *classifier systems.* The feasibility of these approaches is discussed in the sections to follow.

2.1.1 Neural Networks

Since learning is a task which is handled by the brain, it makes sense to model a brain for use in an artificial system. The most straight forward approach to mimicking a biological brain is to attempt to construct a model that closely resembles its structure and operating principles. A biological brain is known to be composed of *neural networks* . These massively parallel networks are composed of massive amounts of interconnected neurons which provide all brain operations in a massively parallel computational manner.[5] There are various portions of the brain which are understood better than others. In particular, as mentioned previously, the area associated with memory is not well understood.

[Rosenblatt 58] did some pioneering research with the idea of a perceptron. Since then, there has been a growing amount of research in developing neural networks that achieve different forms of learning and organization. A collection of papers describing the fundamental research in the area of neural networks is given by [Anderson and Rosenfeld 89].

Neural networks generally have an input layer, one or more internal (hidden) layers and an output layer, where each layer is composed of a collection of neurons as shown in Figure 2.1. Excitatory and Inhibitory links connect neurons from each layer in a highly interconnected fashion. Each of these links has a weight associated with it that indicates the strength of the connection.

5 From this point onward, the term "neural networks" will refer to artificially constructed (i.e. computer or electronically implemented) neural networks.

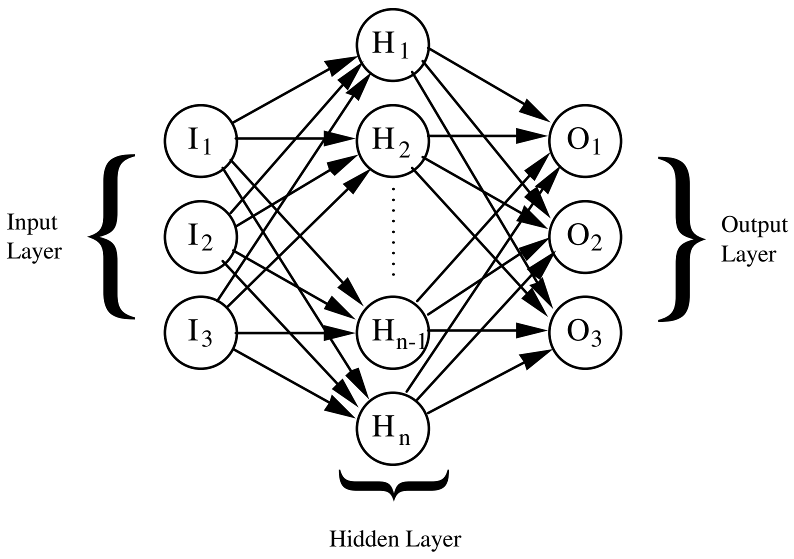


Figure 2.1 A neural network with one hidden layer.

The neurons are simple processing elements that merely compute their activation, perform a simple operation and emit an output [6]. Figure 2.2 depicts an example representing a "summation" neuron X that simply outputs a sum of the input neurons' signals multiplied by their weights.

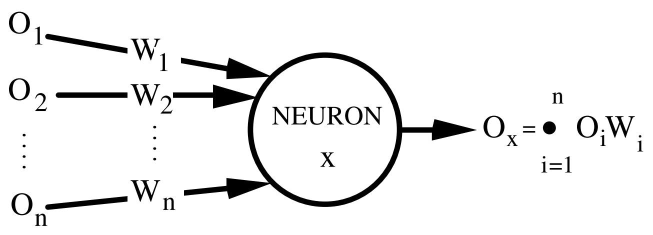


Figure 2.2 Model of a simple summation neuron.

There have been many variations in the processing aspect of this simple neuron model, however, they will not be discussed here. Again the reader is referred to [Anderson and Rosenfeld 89] for examples of neural network structures and the neurons therein.

[6] With computers, the inputs, weights and outputs are usually represented as floating point numbers between 0.0 and 1.0.

The networks are able to learn by matching specific input patterns to specific output patterns. The weights of the interconnecting links are altered to reflect the information that has been learnt. In fact, some networks actually begin with random weights assigned to all links, thus learning is required in order to set the weights to any meaningful values. After a period of learning (multiple input patterns are entered), the network maps the set of input patterns to a set of output patterns. These networks are capable of accomplishing pattern matching feats which would prove to be difficult to achieve through traditional straight forward programming. One shortcoming of neural networks is that they require the environment (or supervisor) to supply the correct value for each set of circumstances. This requirement can hinder the use of neural networks for animat purposes since in an unknown environment the robot rarely knows the correct response for a particular situation. At most, the robot can only take a best guess as to which action should be performed at any one instant. Moreover, it is not clear how feedback in the form of payoff could be used in these networks.

2.1.2 Genetic Algorithms

The theory of evolution states that complex animals evolved from lower, more primitive forms of life. This evolution results in physical and behavioral changes from generation to generation. In fact, some believe that sophisticated forms of life have evolved from other very primitive forms of life. If this is so, then the structure and organization of the sensory and control mechanisms must have evolved greatly. This physical evolution is the result of adaptation of the animal to its environment. It would be reasonable to assume that with adaptation, these animals would learn "rules of thumb" for survival in their environment. With genetic selection, these rules are passed onto future generations in the form of instincts. It is also reasonable to assume that these instincts are improved upon from generation to generation. [Staddon 83] gives a convincing argument for these assumptions, with the example of Orb-web spiders. Orb-web spiders have devised a most efficient net for catching flying insects, yet researchers can trace no history of trial and error in the life of an individual spider that could explain the excellence of the web' s design. Spider' s don' t learn how to weave good webs and no spider designs a variety of different webs, discarding all but the most efficient. This instant web-building perfection represents a solid example of an evolutionary instinct.

Due to the enormous complexity of biological neural networks, it seems reasonable to assume that an animal' s central nervous system is not completely determined genetically. Nevertheless, the assumption that all forms of life are endowed with instincts, through genetics, is also reasonable. Insects for example, do not have the brain capacity required for learning, and therefore rely mostly on instinctive mechanisms for survival. These instincts may pertain to hardwired reactive and performance mechanisms representing reflexive and taxic behaviors which eventually become fine tuned to environmental stimuli over time through evolution.

Genetic programming is a computational tool that attempts to mimic the process of genetic selection and evolution. Stated simply, the idea of this strategy is to create a population of agents [7], and select the ones with the best performance, for use in the next generation (population). Eventually, by continually selecting the best agents from generation to generation, the agents in the later generations will perform well.

A genetic algorithm (GA) usually represents an agent as a string of bits, called a *chromosome* (schema), where each bit represents some aspect of the agent. In the case of a simulated robot or animat, the bits may pertain to sensor and actuator information. The genetic process begins by creating a population of random schema and placing them into an environment for a specific period of time. After this time has elapsed some schemas are then selected on the basis of best performance. There must be some method of determining which schemas are better than others. To do this, the genetic algorithm incorporates what is called a *fitness function*. By applying this function to each schema, an indication as to their relative prosperity is acquired. For example, an agent attempting to find food may have a fitness function that measures the Euclidean distance from the agent to the food source. The agents with better performance would have a smaller distance value.

Using the fitness function, the best schemas are selected. These schemas are selected in pairs for reproduction of offspring. The process takes a pair of schema and performs a crossover in which a random split is made in the two strings of bits. The split partitions the parents into two pieces each. The right piece of one parent is concatenated to the left piece of the other to produce two new offspring as shown in Figure 2.3. These offspring then undergo a mutation in which one of their bits is randomly flipped. This allows for a degree of randomness in the genetic process. The resulting offspring are used in the next schema population.

[7] The term agent represents the object being studied (i.e. an animal, animat, robot, ...).

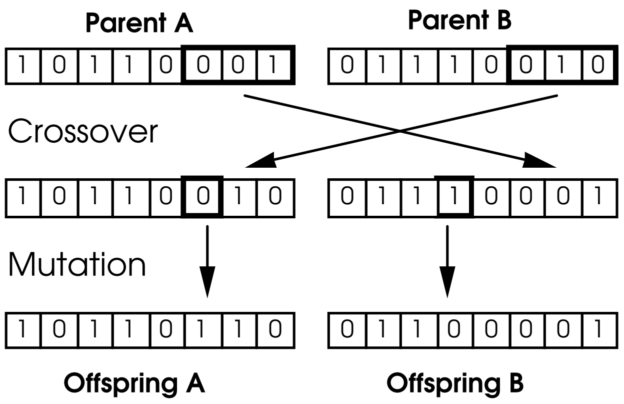


Figure 2.3 Generating offspring schema through crossover and mutation.

There has been some recent work in the area of genetically programming robot instincts. [Koza and Rice 92] have developed a simulation of a genetically programmed robot capable of the simple task of pushing a box from the middle of a room towards a wall. The robot is equipped with 12 sonar sensors which report distances to the nearest object and is capable of moving in the forward, left and right directions. The first few generations of robots are not always able to find the box. Those that find it, are not able to push it correctly towards the wall. However, after a few generations of robots, they are capable of finding the box and pushing it to the wall.

The foremost advantage of using this genetic programming technique is that very little programming needs to be done. Instead, most of the design work corresponds to defining the problem, in which a major part is determining a useful fitness function. The robot does not know a priori what the sensors mean nor what the primitive motor functions do. That is to say that the robot actually learns to perform the box pushing task from scratch.

However, there are some drawbacks to using genetic algorithms. GAs are subtle in that it often takes many generations of schema before a reasonable performance is achieved. Thus the need for high speed computations. Despite the impressive results of [Koza and Rice 92], the robot' s task is quite simple and may not prove to be useful. Another disadvantage of using a GA surfaces when the robot must perform a variety of tasks which may involve switching among them at different points in time. Such a system would require a large bit string size and may take a long time to evolve. Moreover, as the bit string becomes longer and more complex, the determination of a feasible fitness function becomes increasingly difficult.

2.1.3 Classifier Systems

A *classifier system* (CS) combines a rule-based approach with genetic algorithms. The system uses a set of classifiers (rules) to control an agent. These classifiers are similar to schemas of GAs in that they consist of a string of bits. Like GA applications to animat agents, these bits represent sensor information and actions. The classifiers consist of a condition/action pair representing a rule. The system keeps a list of all classifiers as shown in Figure 2.4. There is an input interface that translates the current state of the environment into standard messages, and an output interface that translates messages into effector actions. Finally, a message list keeps track of incoming and outgoing messages.

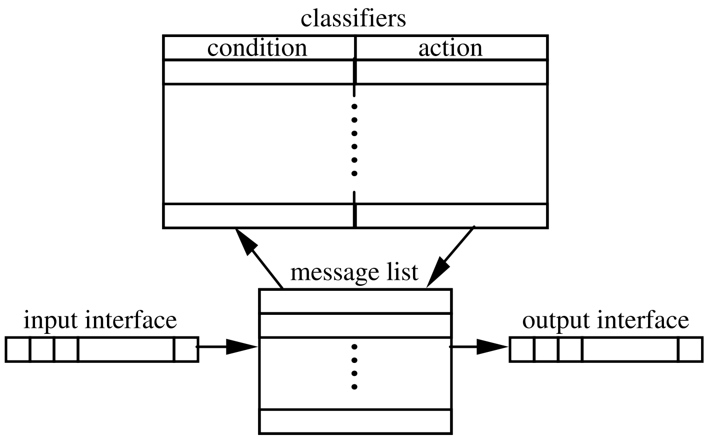


Figure 2.4 The basic parts of a classifier system.

As stated in [Booker et al. 89] the classifier system works as follows:

Step 1: Translate all incoming messages from the input interface to the message list.
Step 2: Compare all messages on the message list with the conditions of all classifiers and record the matches.
Step 3: For each match, post the message representing the action part of the classifier onto the message list; removing all previous messages on the message list.
Step 4: Combine the requirements of all the messages on the message list into one message and send it onto the output interface.
Step 5: Return to step 1.

The system allows an animat to perform according to the rules imbedded in the classifiers. Thus, by beginning with a set of basic classifiers, the animat is able to perform in an instinctive manner.

The system has two additional components used for learning purposes corresponding to credit assignment and discovery. Assigning credit to rules that achieve rewards is a difficult task since many of the rules need to be rewarded for their role in the "stage setting" process. By adding a payoff strength to each classifier, the classifiers are rated as to their relative usefulness in specific situations. The system uses a bucket brigade algorithm that adjusts the strengths to reflect the classifier' s overall usefulness to the system. Each time step, the classifiers make "bids" according to their strength. Only the highest bidding classifiers get their messages on the message list. The bucket brigade ensures that the "stage setting" rules eventually receive credit if they are coupled into sequences that eventually lead to payoff.

The discovery component of a CS uses a genetic algorithm, which applies genetic operators of crossover and mutation to classifiers selected according to their strength. The weakest classifiers are replaced by the new offspring. This process ensures that the animat will learn, over time, the rules which best solve the problem at hand. The credit assignment and discovery process are a little more involved than mentioned here. For further reading, the reader is referred to [Booker et al. 89].

[Wilson 85] used a classifier system for an animat that lived in an environment containing 92 distinct sensory vectors. The animat' s objective was to satisfy its need for food. With no initial knowledge of what food looked like, the animat evolved classifiers that led it to move towards visible food. It was also able to move in efficient paths towards food that could not be seen immediately, by using other objects as clues to the proximity of the food. This example shows that classifiers can prove to be quite useful for animat purposes.

[Wilson 87] suggests that a classifier system is suited to learn multiple disjunctive concepts incrementally under payoff. That is, classifier systems were designed to handle a complex and perpetually novel stream of data which makes them adaptively efficient. Moreover, in a CS, some relevant generalization of rules is achieved automatically. This may be due to the fact that a CS recognizes the importance of tightly coupling induction mechanisms with problem solving.

It is suggested that a CS works well when there is a large amount of sensor information, with relevant data being sparse. For simple robots with little sensor information, the system cannot take advantage of the environmental input and thus may not perform as well as a simple rule-based approach. A shortcoming of the CS is that the rules are written in a language that lacks descriptive power. This makes it difficult to debug and/or reason about. In addition, since several rules are allowed to be fired simultaneously, there are control issues that must be handled that do not appear in the other strategies.

2.1.4 Combined Strategies

Perhaps a compromise between neural networks and genetic programming may prove to be more efficient for robotic control. Such a system could make use of the neural network structure, while using genetic algorithms to specify the interconnecting links. [Beer and Gallagher 92] have experimented with combining neural networks and genetic programming. Their approach was to construct a neural network whose interconnected weights are determined genetically. The experiments involved the development of a *chemotaxis* behavior as well as a control technique for coordinated walking. The first set of experiments resulted in generations of robots that exhibited four classes of chemotaxis behavior. These four classes were observed as different movement patterns in the area of food patches. The results show that genetic programming, combined with neural networks can capture the subtle intricacies of biological behavior.

Classifier systems, themselves represent a combined strategy of rule-based systems and genetic algorithms. To my knowledge, the combination of neural networks and classifier systems has not been studied. [Booker et al. 89] gives a comparison between classifier systems and the connectionist approach (neural networks). Perhaps, all three strategies combined could produce a sophisticated architecture for use in A.I. problems. It is unclear however, if much will be gained from using the combined strategies as opposed to any single one.

2.2 The Hardwired Approach

There has been much research using neural networks, genetic algorithms and classifier systems to allow animats and robots to learn how to perform simple behaviors and control problems. Some examples are learning to walk, learning to orient towards a light source, learning to push a box etc. Since these behaviors are rather simple, it is possible to design a hardwired approach for each behavior. It is the author' s opinion that little is gained by allowing a robot to learn how to perform a simple behavior that could have been hardwired in a simpler, quicker and straight forward manner. A robot insect learning to walk with a neural network for example, would probably end up using a simple tripod gait. If this is so, then why not hardwire the tripod gait instead of wasting valuable resources trying to learn.

Sometimes however, unpredictable behaviors can arise from allowing the behavior to be learnt from scratch as in [Beer and Gallagher 92]. The variation in chemotaxis behavior in their research does resemble the unpredictability (chaos) of natural systems. Sometimes this chaos is an important part of life itself, and by omitting this randomness, it may not be possible to mimic life forms to any degree of satisfaction.

After a few generations, their robot was able to perform a chemotaxis behavior, and eventually the developed behavior was at a stage in which no further generations would improve upon it. At this point in time, the behavior remains relatively unchanged, thus fixed. The resulting neural network could then be hardwired into electronic hardware, thus reducing the size and cost of the robot and achieving the same performance.

Now consider a robot that is able to perform a variety of these "instinctive" types of behaviors. A neural network or classifier system would require a lot of processing power if it were to learn these behaviors from scratch. This processing speed and power may not be available for nanobots. It would be better to hardwire these behaviors into electronic gates so as to reduce the need for processing power. The foremost advantage, however, of employing predetermined instinctive behaviors as a control tactic is that the system in which they reside will have a quick response time when encountering environmental stimuli. Quick reactions may be necessary for systems that operate in potentially dangerous fast pace environments.

A main disadvantage of this hardwired approach would be that learning mechanisms cannot be hard wired directly into electronic hardware. Thus, the instinctive behaviors would be fixed and unable to be improved upon. This is not so bad since for an unchanging environment, the learning process of neural networks, GAs and classifier systems eventually diminishes as time passes. By choosing behaviors that have resulted from using any of these strategies, the performance of the hardwired robot will be very similar. If the environment is changing dynamically however, then the hardwired approach would not be able to handle the changes due to the lack of learning. Another problem is with robustness. The hardwired approach is not as robust as a neural network, since it is composed of a much, much, smaller number of processing units. That is, the death or malfunction of a single unit with the hardwired approach may lead to ill behavior or robot fatality, whereas in a neural network no change may even be observed, unless massive malfunction occurs (i.e. brain damage). However, for an average robot in a nonhazardous environment, if the instinctive neural circuitry is well designed and constructed, then the issue of robustness may not even be considered a problem.

2.3 Fixed Weight Neural Networks

If instincts are to be hardwired into electronic circuitry, it would be helpful to have the behaviors coded in some form that is similar to electronic gates. Of the strategies mentioned so far, neural networks are closest to electronics since they are composed of simple processing units which are similar to gates. The only aspect of neural networks that is difficult to code into electronics is the ever-changing weights associated with each link. Since these weights are used for learning purposes, they do not need to change over time if the network is used to code an instinctive behavior.

[Beer 90] presents a model of robot control consisting of a network of interconnected neurons that use fixed weights. The term *neuron network* is given to this model in order to make the distinction between standard artificial neural networks. The model differs from traditional neural networks in a number of ways:

1) The network does not learn; the weights of the interconnecting links have fixed values.
2) There is no input, middle and output layers per se. The activation through the network does however have a main direction of flow.
3) The number of neurons in a neuron network is much less than a neural network and therefore, a neuron network is usually quite smaller.
4) The neuron networks are not as heavily interconnected as a neural network.

5) A neuron network typically has a variety of different types of neurons whereas a neural network usually has just one type throughout.

Although the two types of networks have these differences, they share the same basic fundamental computational strategy in which neurons are excited or inhibited by others. A neuron network is made up of neuron [8] objects. Each neuron receives input signals from its adjacent "input" neurons, computes some simple function using these input signals and then produces an output. Like the formal neurons of artificial neural networks, the neuron model ignores the details of action potential generation and most of the complexities of synaptic and dendritic interactions. The neurons are most similar to those of [Hopfield 84], but differ in the choice of input/output function, inclusion of time-dependent properties and the nonuniformity of the network elements and their connections. The input/output function is characterized by three parameters: a threshold at which the neuron begins to fire, a firing frequency, and the gain. By varying these parameters, the behavior of the neuron can vary.

2.4 RABI's Neuron Networks

The neuron model of [Beer 90] allows the neurons to vary slightly in behavior by adjusting various parameters. It is reasonable to speculate that there may be a variety of different types of computational functions exhibited by biological neurons since there are various parameters associated with them. [Caudill and Butler 90] state that biological neurons display a large repertoire of input-combining treatments. They also state that there may be several kinds of special purpose direct inputs to a neuron and that the neurons may vary their input threshold. With this in mind, the neuron network model was slightly altered by allowing a variety of different types of neurons by essentially varying these parameters.

[8] The term neuron is borrowed from traditional neural networks, however, it is simplified greatly from actual biological neurons.

2.4.1 The Neurons

Since the neuron networks may eventually be hardwired, it would be useful to design neurons that easily convert to electronic circuitry. The interfacing would probably be digital and require binary signals. Thus, we could create neurons that emit binary signals of either 0 or 1 depending on whether they are excited. Borrowing from the notion of digital logic, we could create neurons that perform simple operations such as an AND gate, pulsing signals, flip-flops, comparators, summing, differentials etc. Figure 2.5 shows function diagrams of neurons that perform similar operations by connecting a variety of simple mechanisms.

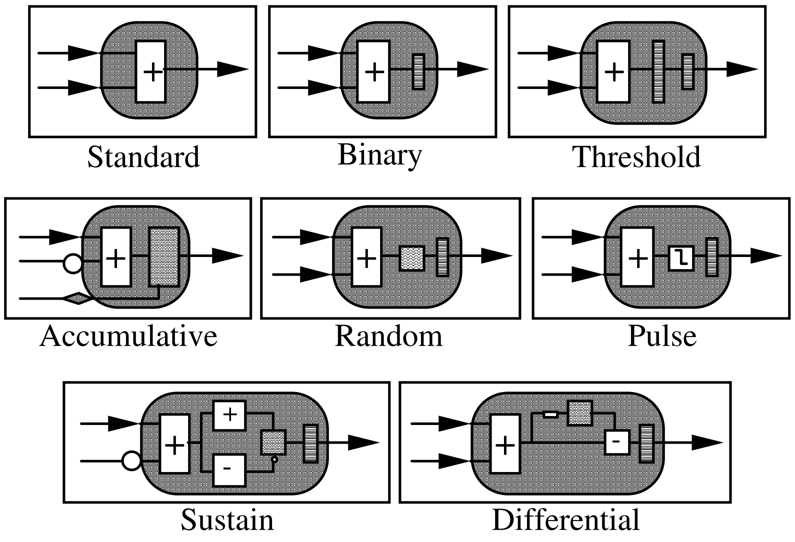


Figure 2.5 Functionality of the neurons in the neuron networks.

Each neuron takes a sum of its inputs and produces a single output representing the sum. The *standard* neuron emits this "analog" [9] signal as its output. *Binary* neurons on the other hand, pass this summated signal through a threshold mechanism. If the value is positive, the binary neuron emits a binary signal of 1; otherwise it emits a 0. Another similar neuron is the *threshold* neuron which uses a threshold on the input signals and on the output signals. Thus, the neuron emits a binary high signal if its input sum exceeds the input threshold. This neuron can be used as an AND gate where two or more high input signals are required in order to produce a high output signal.

[9] With computers, the inputs, weights and outputs are usually represented as floating point numbers between 0.0 and 1.0.

An *accumulative* neuron is similar to an accumulator as seen in microprocessors. It keeps a summation of all excitatory and inhibitory signals from the input lines and emits an "analog" signal indicating the current sum. This sum may be negative (i.e. in the case where inhibitory signals dominate the excitatory ones. The neuron has a special input signal that when high, resets the accumulated sum to zero. This neuron does not seem biologically feasible but nevertheless it provides a useful neural tool for counting which is required for measurement purposes.

A *random* neuron emits a random binary output whenever its input activation is positive and emits no signal otherwise. The neuron has an internal probability factor indicating the probability of outputting a binary high signal. The idea of a random neuron is not as readily accepted since there is no well known biological neuron that performs this function. All animals however have an unpredictable aspect related to chaos theory that endows them with a kind of randomness.

The *pulse* neuron is similar to a rising edge or falling edge [10] detector in electronics. It monitors the incoming signals and emits an energy pulse (a high binary signal) whenever the input sum switches from high to low or vice-versa. This neuron is used as an on or off switch that enables or disables other neurons occasionally.

The *sustain* neuron contains a state, thus it is similar to an electronic flip-flop. When the incoming signal summation is positive, the state is set to "on" and the neuron emits a binary high signal. The neuron continues to emit this high signal until the input summation is negative; it then sets the state to an "off" position and the neuron emits nothing. This neuron does not seem biologically plausible since it is capable of indefinitely emitting a high signal, even after the inputs are removed. This neuron is needed however, since there is often a need for maintaining a state (1 unit memory).

Finally, the *differential* neuron is used as a comparator, by comparing the current input with the last input. If the difference is positive, the neuron emits a binary high signal, otherwise it emits a low. The neuron is capable of remembering the last input summation for use in comparison. Thus, this neuron uses a kind of volatile memory which could be implemented as a feedback loop with a delay.

[10] The rising-edge and falling-edge neurons are identified by a _|¯ and ¯|_ symbol respectively.

2.4.2 The Networks

RABI' s programming involved the creation of simple instinctive behaviors. Since there is no learning aspect to the behavior mechanisms, they were coded with neuron networks using the repertoire of neurons mentioned in the previous section. The various networks are described throughout this thesis with diagrams showing their components. The visual form of neurons of Figure 2.5 can cause a neuron network diagram to look cluttered and complex; therefore their appearance has been altered for the network diagrams for the sake of keeping the diagrams simple. The appearance of the various neurons are shown in Figure 2.6.

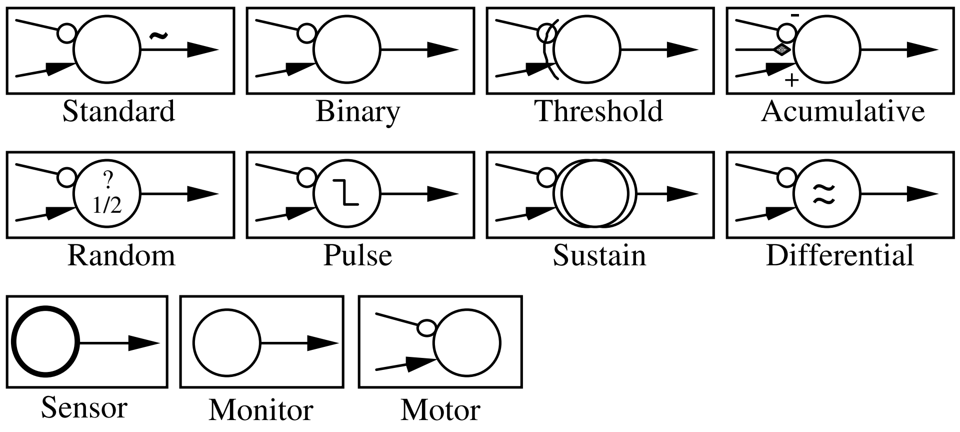


Figure 2.6 Appearance of the neurons in the neuron networks.

Note the addition of *sensor, monitor* and *motor* neurons. These neurons represent the input and output neurons similar to traditional neural networks respectively. The sensor (or monitor) neurons receive input directly from a sensor (or monitor) which they are connected to, and output a signal whose strength reflects that of the sensor (or monitor) reading. Similarly, a motor neuron connects directly to an actuator. When the motor neuron receives an input activation, this activation directly affects the actuator.

Interconnecting the neurons are links as shown in Figure 2.7. Each link has a weight associated with it. *Inhibitory* links result in a negative activation whereas *excitatory* links result in positive activation, each with an associated weight. Most of the interconnections are composed of these two kinds of links. The additional *negating* link provides a method of negating a neuron' s output. Any neuron that receives positive activation from a negating link will negate its output. This is useful for changing directions of a motor neuron. Finally, the *resetting* link is used to reset the accumulated energy of an accumulative neuron. When an accumulative neuron receives a positive activation from a link of this type, its internal stored energy is reset to zero.

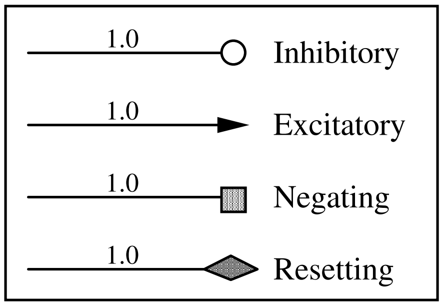


Figure 2.7 Various interconnecting link types used in RABI' s neuron networks.

Since these neuron networks are simulated, the various timing and delay properties of the neurons can be directly controlled. With direct hardware implementations, however, there are certain neuron characteristics (such as feedback) that require additional care so that they have similar performance to the software versions. The technicalities of the construction of these neuron networks in electronic hardware has not been investigated. This is a future area of study towards the development of a fully self contained version of RABI.

2.4.3 Neuron Networks and the Subsumption Architecture

The subsumption architecture(SA) of [Brooks 86] provides a form of task-level decomposition in which the various basic functions (or instincts) can be programmed separately and then linked together. The SA in its original form uses state machines to code the behaviors. Previous robots designed using the SA required microprocessors to program and control these state machines which controlled the overall performance of the robot. RABI presents a simplified approach towards coding these simple behaviors. By using neuron networks, processors are not needed since the networks can be hardwired into electronic circuitry. By interconnecting the neural circuits in a hierarchical subsumption-style fashion, the overall structure of the system would have a similar performance as the SA, but with reduced size and complexity.

2.5 Summary

Many techniques have been used to develop "intelligent" robotic systems. Among these, neural networks, genetic algorithms and classifier systems are the most widely used for learning purposes. Much of the research using these techniques has focused on learning simple behaviors from scratch. While these methods of learning provide insight as to the mimicking of the behaviors of biological systems, they often require much computational power. This high computational requirement can lead to larger, more complex and expensive robotic systems.

By hardwiring the simple behaviors into the robot, the computational requirements diminish, allowing the robot to be reduced in size and complexity. This hardwired approach could be implemented using neuron networks, which are similar to the notion of fixed weight neural networks [Beer 90]. By using a repertoire of simple neuron-like elements, these neuron networks can provide a variety of different functional behaviors. The arrangement of these behaviors in a subsumption-style arrangement allows the overall system to perform in a similar manner to the robots of [Brooks 86] but with the possibility of reduced size and complexity.

Chapter 3
Walking and Coordination

All autonomous robots require some form of movement. More often than not, this movement is realized through wheeled motion. However, this form of movement is not a natural form of locomotion. Most visible forms of life use some form of joint movements in order to achieve motion. Insects for example have a simple yet efficient mechanical structure. An example of the efficiency is the fact that ants can lift many times their own body weight. In fact, insects perform rather well considering their very small size. An insect is defined to have 6 legs, however, there are insect-like organisms with 4, 6, 8 or more legs that perform similarly.

Giving a robot the ability to walk has its advantages but adds a great deal of complexity to the aspect of control. A walking machine has the ability to step over small obstacles of various kinds as well as being able to perform efficiently in rough terrain environments. Legs allow the robot to access environments that wheeled robots would normally fail to perform in due to obstacles, ruts, uneven surfaces, etc. Conversely, robots with wheels have the advantage that they can move quicker, provided that the terrain is suitable, and require a less complex control strategy. The reason for this ease of control is due to the small number of degrees of freedom (i.e. usually just two motors forward and backward) as well as the simplified terrain necessary for wheeled locomotion. A robot that is required to adapt to its surroundings would prove to be more useful if it were able to handle a variety of terrains. There is a tradeoff, however, since legged vehicles usually require more complex control techniques and complicated mechanics as well as heavier power requirements.

Controlling walking robots is a difficult problem due to complications such as maintaining stability, choosing an appropriate gait and coordinating the legs [Bernstein 67]. In many cases, the robot must be precisely controlled and may be required to deliver payloads. This causes additional problems related to force, compliance, energy, stress and power computations. RABI does not take on these extra computational burdens since it is only walking around in a simplified environment.

3.1 Gaits

When designing a legged robot, all the legs must be coordinated otherwise the legs will just flop around and the robot will not achieve motion. Let the *transfer phase* of a leg be the period in which the foot is not on the ground. Similarly, the *support phase* of a leg is to be the period in which the foot is on the ground, providing support. A leg does not exhibit continuous motion. The leg must be lifted at the end of each stroke and placed down again in preparation for another stroke. This alternating phenomenon creates a phasing problem that is described by the term *gait*. A gait represents the combined coordinated motion of the legs, defined as the time and location of the placing of each foot. There are various types of gaits that may be used by a legged robot, each with its own unique movement pattern.

3.1.1 Gait Selection

Selecting an appropriate gait depends upon certain conditions such as the terrain condition, stability requirements, ease of control, smoothness of body motion, speed, mobility and power requirements. The selection depends greatly on the type of terrain that is to be traversed by the robot. Consider dividing the terrain into foot-sized cells. Each cell can be labeled as a valid or forbidden zone. A forbidden zone is a zone that does not provide support. These may be cells with weak soil structure, steep gradients, interference between terrain and legs, etc. Terrains can be categorized as perfect, fair or rough depending on whether they have none, few or many forbidden cells respectively.

At this point it will help to define two characteristics that a gait may possess. A gait is *symmetric* if the motion of the legs of any right-left pair is exactly half a cycle out of phase. A gait is *periodic* if similar states of the same leg during successive strokes occur at the same interval for all legs. That interval is called the *cycle time*, T, which is the time that it takes to complete one cycle of leg locomotion.

Periodic gaits work well in perfect terrain since they are easily implemented. There are 3 main types of periodic gaits described as follows.

1. Wave gaits provide optimum stability. In a wave gait, stepping motion occurs in a wave-like fashion from the back legs to the front legs.

2. Equal phase gaits can easily distribute the placing and lifting events in a locomotion cycle, minimizing fluctuations in power consumption.

3. Continuous follow the leader gaits require that the two front legs are specifically placed, the middle and back legs follow in the same footsteps.

In rough terrain, periodic gaits are not very useful since there are too many locations that the feet cannot be set down in. *Large obstacle crossing* gaits are commonly used for situations such as ditch- crossing, hill climbing, step mounting, or overcoming an isolated wall as depicted in Figure 3.1. Follow the leader gaits are useful in these situations since only the front two legs must be carefully placed and the others fall into place. Various obstacle crossing gaits are discussed in detail by [Song and Waldron 89]. A robot utilizing solely periodic gaits would be unable to perform with these physical obstacles, and as a result the regions would be labeled as non-traversable.

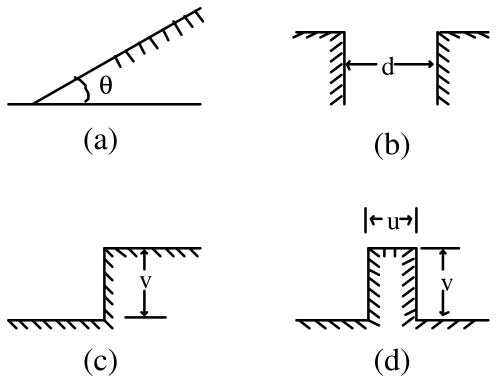


Figure 3.1 Geometric representations of standard obstacles. (a) a gradient (b) a ditch (c) a vertical step (d) an isolated wall.

If the terrain becomes too complex the robot may be required to use a *free gait* [11]. A free gait requires that each leg is lifted and placed one at a time. Usually, a robot would examine the

[11] [Pal and Jayarajan 90] give a discussion of an optimal free gait for 2-D generalized motion of a quadruped walking machine.

environmental terrain and scan for a place to put down each foot. This approach requires much computational power and additional sensing techniques. This type of gait is chosen only as a last resort when all other gaits are ruled out.

Ideally, a robot should be able to make use of multiple gaits. Such a robot would use a periodic gait for perfect terrain and then switch to an obstacle crossing gait when needed. Creating a terrain-adaptable robot with gait switching abilities was not the intention of this thesis. The intention was to create a simply constructed robot that would achieve walking in a simplified 2-D environment. A more in-depth study would be required in order to design a robot with terrain adaptability.

3.1.2 Gait Analysis

The study of leg design and coordination can be quite perplexing. One aspect of complexity is the notion of gait analysis. Various types of gaits can be analyzed to determine their ease of use, distribution of power requirements and measuring the stability of the robot as a whole. This section will just slightly touch upon these issues. For a more in-depth study of these concepts, the reader is referred to [Song and Waldron 89] and [Todd 85].

Let the *duty factor*, β_i, be the time fraction of a cycle time in which leg i is in the support phase as follows:

$$\beta_i = \frac{\text{time of support phase of leg i}}{\text{cycle time of leg i}} \tag{3.1}$$

A *regular gait* is a gait with the same duty factor for all legs.

$$\beta_i = \beta_j = \beta \quad \left\{ \begin{array}{c} \text{i, j} = 1, 2, \ldots, n \\ n \text{ is the leg number} \end{array} \right.$$

Let the *leg phase,* ϕ_i, be the fraction of a cycle period by which the contact of leg i on the ground lags behind the contact of leg 1 (i.e. front left). The legs are numbered from left to right starting at the front.

In order to achieve constant motion, the robot must remain stable. Every gait has a margin of stability associated with it that indicates how stable the robot will be throughout its locomotion cycle. [McGhee and Frank 68] have shown that for a 2n-legged gait, the gait stability is maximized by a regular and symmetric gait defined as:

$$2n = 4,\ \phi_3 = \beta, \qquad 3/4 \leq \beta \geq 1 \tag{3.2}$$

$$2n = 6,\ \phi_3 = \beta,\ \phi_5 = 2\beta - 1, \qquad 1/2 \leq \beta \geq 1 \tag{3.3}$$

Since stability [12] is very important, the robot designed for this thesis utilizes the regular symmetric gait defined by equation 3.3. Equations 3.2 and 3.3 represent wave gaits. By varying the duty factor β, various stepping patterns can be realized. These patterns can be graphically displayed using what is known as a gait diagram. A set of gait diagrams for hexapod wave gaits is shown in Figure 3.2. The legs are numbered from front to back, with L or R denoting the left and right sides of the body. The darkened lines indicate the period of the support phase. The beginning and end of a darkened line correspond respectively to the placing and lifting of a foot.

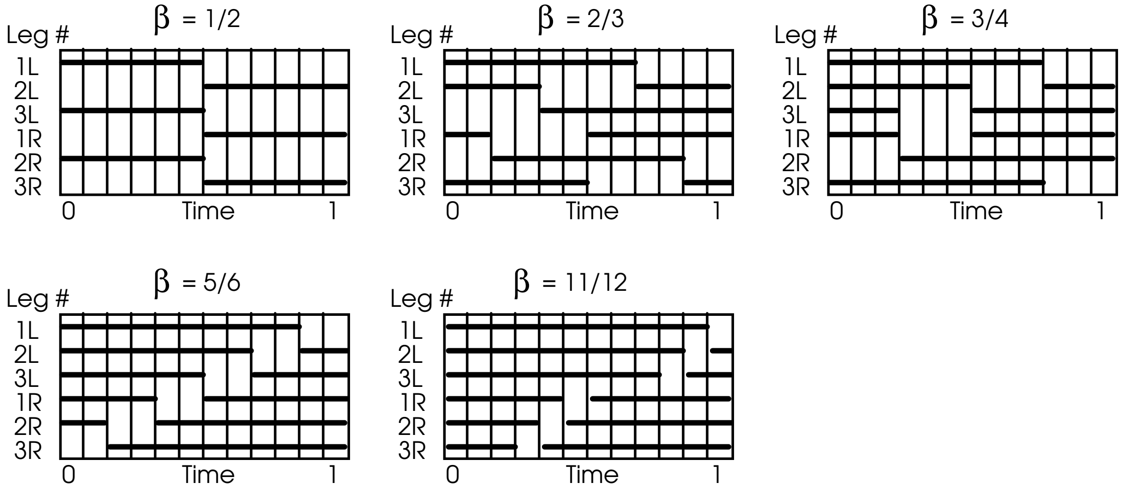


Figure 3.2 Gait diagrams of hexapod wave gaits showing the effect of varying the duty factor, β.

To help understand the role of the leg phase, consider the artificial insect of Figure 3.3. The leg phase for each leg is shown with respect to the duty factor. From the diagram, it is easy to see that by setting β to 1/2, legs 1L, 2R and 3L move in phase as well as 1R, 2L and 3R. This dual phase represents what is commonly termed a *tripod gait.* The tripod gait was chosen for

[12] For other methods of maintaining stability, [Messuri and Klein 85] discusses the notion of body regulation.

RABI since it allows the robot to maintain stability while issuing the minimum number of legs needed to be down at any one time. This gait has been observed in insects [13] when they require fast motion, perhaps to avoid being eaten.

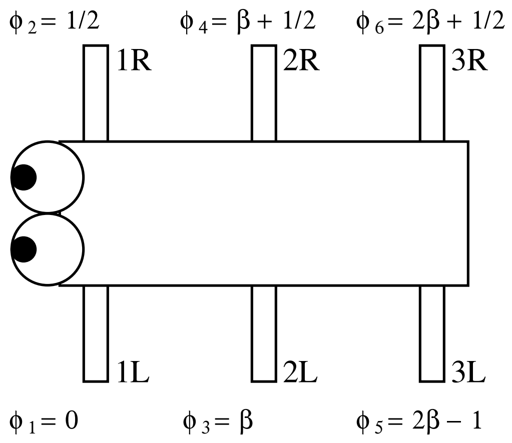


Figure 3.3 Leg phases representing the general wave gait for a hexapod robot.

In the case of a quadruped, their must be at least 3 legs down at a time in order to maintain static stability. This is indicated in equation 3.2 which states that the duty factor must be at least 3/4. A gait diagram representing a quadruped wave gait with the duty factor of 3/4 is given in Figure 3.4.

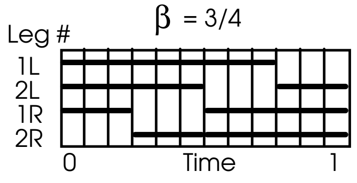


Figure 3.4 Gait diagram of a quadruped wave gait with duty factor 3/4.

[13] [Wilson 66] investigates the various gaits observed in insect locomotion.

3.1.3 Quadruped Vs. Hexapod

The 6-legged gait of Figure 3.2 where β=1/2 and the 4-legged gait of Figure 3.4 were both investigated in different versions of RABI. In practice, it has been observed that the 6-legged gait is more stable than the 4-legged gait. This observation could not have been made without the development of physical mechanical robot. The problem is related to the relative positioning of the supporting legs over time.

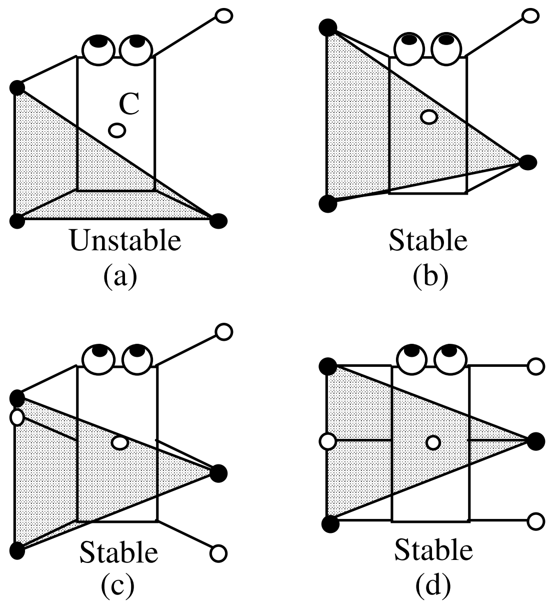


Figure 3.5 Instability problem caused by leg placements. The quadruped (a) becomes unstable if the shaded polygon becomes too narrow, and remains stable when the polygon widens (b). The hexapod (c) and (d) remains stable throughout the cycle.

In order to explain the phenomenon responsible for the instability of the 4-legged robot, we can loosely define the stability in terms of the center of gravity. Consider the 2-D projection of the center of gravity, C, onto a plane containing a *foot-fall* polygon formed from the foot positions of the robot' s supporting legs. If this projection falls withinthe polygonal boundaries, then the robot is considered to be stable (i.e. standing). There are many factors left out such as forces, slippage, inaccuracies in the construction of the legs, etc. that also play a role in stability but do not need to be analyzed to point out the stability problem encountered. Indeed what is happening is that in the 4-legged gait, the center of gravity sometimes falls outside the border of the foot-fall polygon as shown in Figure 3.5. This stability problem can be subdued if the swinging arc of the legs is reduced, or if legs on one side of the body remain far apart. This would keep the center of gravity within the foot-fall polygon. Notice that the center of gravity shifts slightly with the leg movements.

The hardware version of RABI went through 3 major mechanical design changes, including both a quadruped and hexapod configuration. These leg designs are discussed in more detail in chapter 8.

3.2 Gait Implementation

Once a gait is selected, it must be implemented in either hardware or software. Provided that the gait is simple enough, it may be implemented directly into hardware. The tripod gait is easily implemented since it is periodic, regular and symmetric. There have been a variety of methods for implementing gaits on legged machines.

3.2.1 Previous Approaches

One of the pioneering approaches to insect-like robot design was that of [Brooks 89]. He used a form of interconnected finite state machines (FSM's or *modules*) each with internal timers to achieve walking. These FSM's are connected in a network such that signals are passed among them. The network contains modules for walking, balancing, force control, simple obstacle detection and steering. Each leg has a set of modules controlling the basic movements. A single central module emits triggering pulses that control each leg module. By varying this controlling module, various wave gaits can be realized. Brooks gives no indication of how complex each of these FSM's are, but does mention that 57 are used in an incremental fashion. This "add-on" property allows the steering, and obstacle detection modules to be removed without preventing the robot from functioning in a simple walking manner. His subsumption architecture allows additional behaviors to be added with ease, as well as providing a robust system where components can break down without halting the entire system.

[Ayers and Crisman 92] have implemented what is called a CCCPG (Command Coordinating Central Pattern Generator) for generating an omni-directional gait similar to that observed in the American lobster. The generator has a clock which specifies the period of stepping, and neural circuitry specifying the detailed pattern coordination of each joint. The legs are controlled individually by these CCCPG's and additional neural connections coordinate them.

By varying the clock pulse rate, various stepping patterns are attained. A similar approach was taken by [Beer 90]. Beer' s control strategy consists of a network of interconnected neurons. Each leg has its own local network as shown in Figure 3.6.

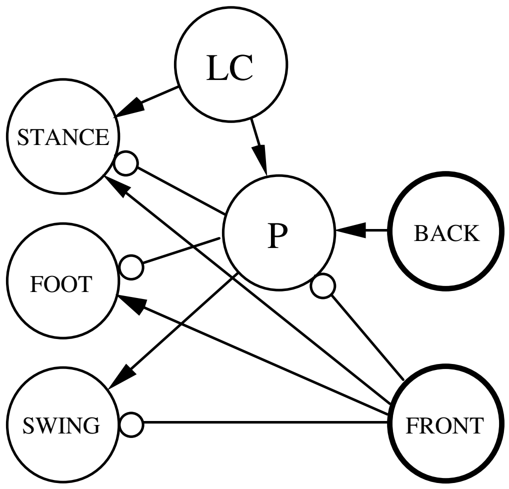


Figure 3.6 Local leg network from [Beer 90]. The pacemaker neuron P, emits a burst of energy at constant intervals which interchanges the leg between stance and swing phases.

In this network, there are three basic types of neurons. The BACK and FRONT neurons are *sensor* neurons. Sensor neurons are directly connected to the robot' s sensors. They emit signals corresponding to the intensity of the sensor readings. In this case, the BACK and FRONT neurons send out a binary signal indicating whether the leg is all the way back or forward respectively. The STANCE, SWING and FOOT neurons are called *motor* neurons. These neurons are connected directly to the robot' s actuators. The SWING and STANCE neurons move the leg forward and backward respectively, while the FOOT neuron lifts and places the leg down. Perhaps the most interesting neuron is the P neuron which is called a *pacemaker* and emits a continuous pulse at constant intervals. By varying the rate of emission, the leg moves at different speeds. When the leg reaches all the way back, the P neuron is triggered, causing the leg to swing forward and the foot to be lifted (i.e. the transfer phase). Once the leg travels all the way forward, the FRONT neuron disables the P neuron, causing the foot to be placed down and the leg to stance backwards (i.e. the support phase). The P neuron also emits pulses on its own which essentially triggers the transfer phase. The LC neuron allows enabling and disabling of the walking mechanism as well as providing a pulse rate to P.

Although the network produces the desired motions of leg movements, it does not provide coordination among the legs. To provide the coordination required for walking, additional connections are made between the pacemaker neurons such that each pair of adjacent pacemakers inhibit each other as shown in Figure 3.7.

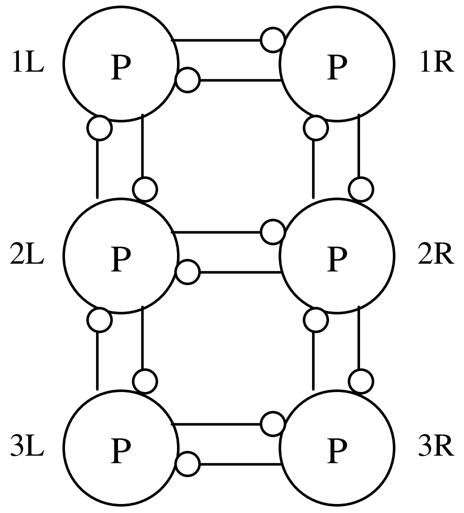


Figure 3.7 Additional connections in which adjacent pacemaker neurons inhibit each other allowing coordinated walking.

[Chiel et al. 92] have shown that this method of control is quite robust. In fact, if all sensor input is disabled, the robot is still able to walk. This is possible since the internal pulsing of the pacemaker neurons provide the necessary excitatory activity for the motor neurons to operate. The robot remained functional even after a couple of links connecting the pacemaker neurons were disabled.

This approach to walking is simple and straight forward and has been shown to be robust. RABI takes a similar approach to Beer' s model. If only a single gait is desired, then the idea of a pacemaker is not needed. In other words, the legs can be fixed to exhibit a single gait, only changing phases when the leg limits are reached.

3.2.2 RABI's Walking

RABI's legs each have two motors. One motor moves the leg in the horizontal direction, the other in the vertical direction (See Chapter 8). In reality, motors do not move at the same speed nor with the same efficiency. Thus, when coordinating the legs, the lagging of the motors must be taken into account. In addition, each foot takes time to be placed. Thus, the legs must be restrained from swinging and stancing until the foot is lifted or placed down appropriately. The easiest way to do this is to add additional circuitry that will delay legs from swinging and stancing until the foot is in place. Figure 3.8 shows the revised network similar to that of Figure 3.6.

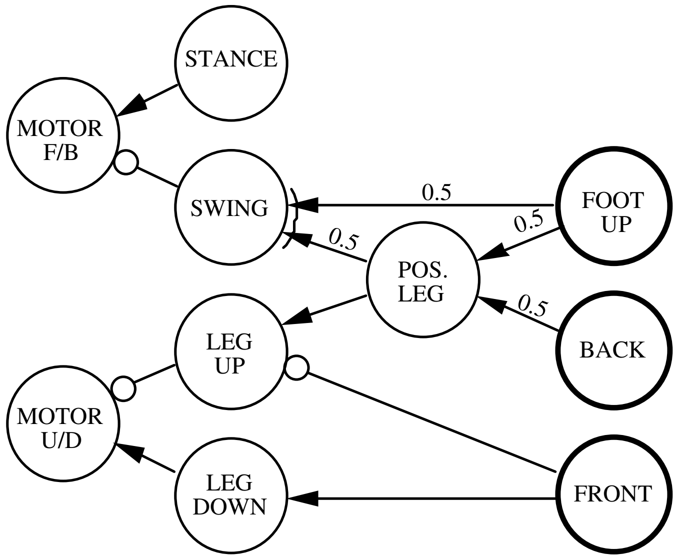


Figure 3.8 Revised leg network accounting for stepping delay.

In the revised network, I have added a FOOT UP neuron that indicates whether or not the foot is touching the ground. A micro switch placed at the bottom of the foot serves as a simple sensor providing a binary input. Two motor neurons control the horizontal and vertical placement of the foot. In this network, the POS. LEG neuron acts as the P neuron in the previous model with the exception that it does not produce its own bursts internally. Here, when the leg reaches the back-most position, the POS. LEG neuron is excited and remains excited as long as the foot remains up. The SWING neuron consequently is excited but being a threshold neuron, it does not allow motor movement unless the FOOT UP neuron is also excited. This provides a method of delaying the swing phase until the foot is up off the ground. The POS. LEG neuron also excites the LEG UP neuron. This allows the leg to be lifted immediately.

Once the leg swings to the frontal limit, the FRONT neuron causes the leg to be placed down. Once the leg is placed down, the POS. LEG neuron becomes inactive (no more active inputs) and the leg stances backwards.

Notice in Figure 3.8 that there are no inputs to the STANCE neuron. One more additional neuron is needed to accomplish movement. A single WALK neuron provides excitatory input to each STANCE neuron. This allows disabling of the walking mechanism altogether. Figure 3.9 shows these connections.

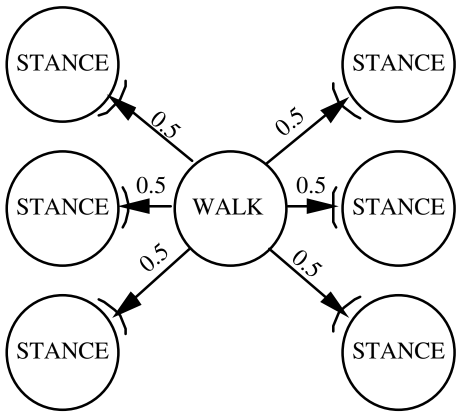


Figure 3.9 Connecting the WALK neuron which provides the excitatory signals needed for walking.

Walking is enabled by exciting the WALK neuron which provides excitation to the STANCE neuron of each leg. When the WALK neuron is not excited, the STANCE neurons are in a sense "disabled". Note again that the STANCE neurons are threshold neurons that receive only a 0.5 excitatory signal. The additional 0.5 needed to trigger it is explained in the next section.

With the networks shown so far, the basic leg movements for walking are achieved but the lack of coordination prevents walking. A similar approach to coordination to that of [Beer 90] was used. In this approach, the POS. LEG neurons inhibit each other. Additional inhibitory links were added to specify the tripod gait. These were needed in order to ensure synchronized alternating phases so that the robot did not break into a generalized wave gait. Figure 3.10(a) shows the connections required to coordinate a hexapod robot, while 3.10(b) shows the connections for a quadruped robot. In the case of the quadruped, the connections are made such that no leg can be lifted unless all others are on the ground. This is needed in order to ensure stability as mentioned previously.

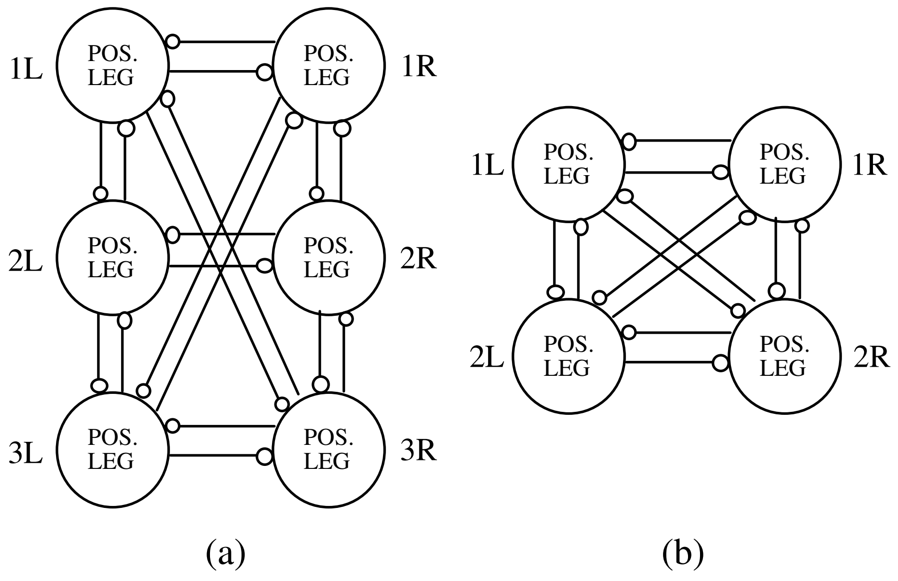


Figure 3.10 Interconnections of leg networks providing coordination between the legs. (a) hexapod tripod gait connections (b) quadruped wave gait connections.

3.3 Turning

Once the robot is able to walk, the next step is to get it to turn. There are a few ways to implement turning capability. One mechanical approach is to design the legs in such a way that the feet pivot towards the desired direction of turning. This would allow turning since each step would slightly alter the trajectory causing the robot to veer off from a straight line. This approach does not allow sharp turns since each step allows only small directional changes. Another approach to steering is to design each leg such that they are capable of thrusting the body in both the forward/backward and right/left directions. The CCCPG model of [Ayers and Crisman 92] uses this method which allows the robot to move from side to side if needed. This approach, although biologically feasible, requires a more sophisticated leg design which could increase the weight and size of the robot.

A simpler method of steering was taken by [Brooks 89] and [Beer 90] which involved instructing the legs on one side of the body to not swing as far back as the legs on the other side. With a legged insect, the smaller the swinging angle, the smaller the forward translation of the robot. Thus by keeping the forward translation on one side of the body small compared to the other side, the robot will turn away from a straight line path [14].

3.3.1 Turning With Simple Sensors

If the robot is only equipped with sensors that detect forward and backward leg limits (as opposed to position measurement sensors), then a simpler method is needed. RABI achieves turning by first stopping, then pivoting the body. To pivot the body, the legs on one side of the body are instructed to reverse direction. Thus, a stance phase in reverse would cause a backward translation. This allows the robot to make sharp turns since all legs are cooperating in the turning process. Figure 3.11 shows the result of such a method.

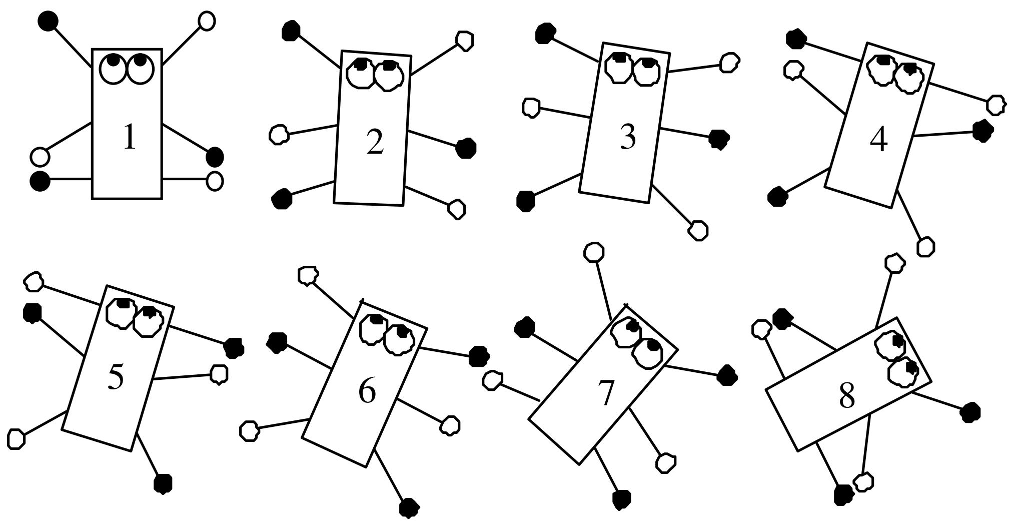


Figure 3.11 Snapshots showing the displacement of a hexapod robot using a pivoting tripod gait to turn right.

The diagram shows the different snapshots of a hexapod using the pivoting method for turning right. Note that there is no angular displacement between steps 4 and 5 since this is when the tripod gait crossover occurs.

Although this method is simple and is able to accomplish turning, there is a disadvantage that the robot must stop in order to turn. This is not as biologically feasible as previous

[14] This is analogous to a car running into a puddle. The wheels that hit the water will slow their revolution while the wheels on the other side remain at full speed. At quick speeds the car will suddenly turn drastically towards the puddle.

approaches to walking since observed insects do need to stop in order to turn. The method does however allow the robot to turn in cornered areas.

3.3.2 RABI's Turning

In order to achieve turning, the legs on one side of the body must be reversed. To do this, a REV. neuron is added to each leg network by connecting it to the SWING and STANCE neurons and the BACK and FRONT sensor neurons as shown in Figure 3.12.

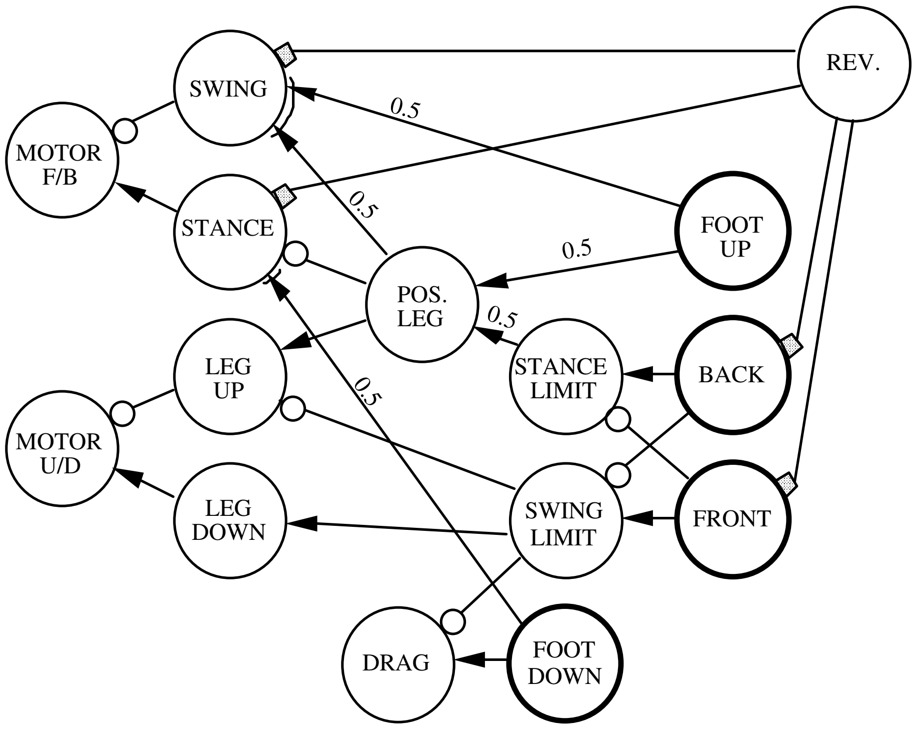


Figure 3.12 Expanded leg network showing additional neurons and connections needed for turning. The DRAG neuron is required for additional coordination.

The REV. neuron provides a special connection into the SWING, STANCE, BACK and FRONT neurons. When these neurons receive a positive activation from this special connection, they negate their output. Thus, when in reverse, the STANCE and SWING neurons will cause the motor to move in a reversed direction. When not moving in reverse, the leg will remain in a stance phase until the leg reaches the back most limit. Similarly, the swing phase will continue until the front most limit is reached. When in reverse however, the role of these two limit sensors is reversed. For this reason the BACK and FRONT limit sensor neurons needed to be swapped. The STANCE LIMIT and SWING LIMIT neurons allow this swapping to occur.

When reversed, the excitatory connections from a neuron become inhibitory and vice versa for the inhibitory connections. This allows the STANCE LIMIT and SWING LIMIT neurons to receive a signal from the appropriate sensor.

Since each leg is given a reverse neuron, some sort of mechanism is needed to decide which legs are to be in reverse at any given time. This mechanism is shown in Figure 3.13.

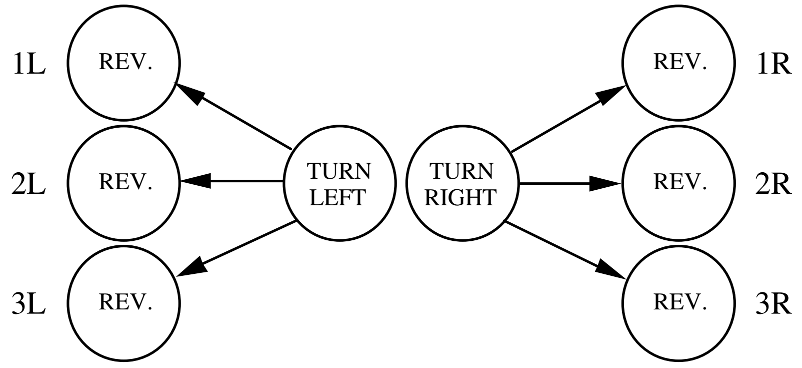


Figure 3.13 A mechanism for controlling the reverse neurons.

Using this mechanism is simple. Turning left or right requires only that the TURN LEFT or TURN RIGHT neurons be excited. Once excited, these two neurons turn on the appropriate reverse neurons, thus executing a turn. Additional neural circuitry can be connected to the TURN LEFT and TURN RIGHT neurons to provide higher level control over this low level mechanism.

The DRAG neuron of Figure 3.12 was added to add further coordination of the legs. This neuron becomes excited whenever a leg is considered to be dragging on the ground. This will happen whenever the leg has its foot down unless it is at its swing limit (i.e. waiting for the other legs to catch up so that a stance phase can be performed).

The additional circuitry to allow the extra coordination is shown in Figure 3.14. Here, the tripod gait is built-in. Essentially, the network prevents opposite phase legs from stancing if any of the other phase legs are still in the stance phase. In other words, it makes sure that all the legs are synchronized to start their phase.

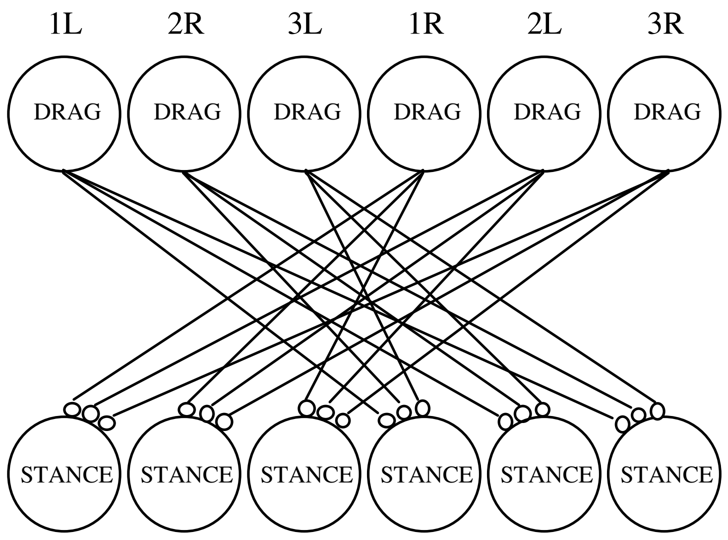


Figure 3.14 Connecting the DRAG neurons such that no leg begins a stance phase until all others of opposite phase are ready for a swing phase.

The addition of the DRAG neurons was necessary for the hardware version of RABI only. The legs of the hardware version moved at different speeds and consequently, there was a need for some form of additional synchronization. The simulated version is obviously precisely timed and thus such additional synchronized connections were not needed. Again, the importance of building a physical robot surfaces.

3.4 Summary

Various approaches to coordinated walking and turning have been discussed. The previously research methods provide multiple gait capability at different speeds. They also provide turning while walking. RABI uses a simple approach that uses simpler sensors and coordinating techniques which may be hardwired into electronics due to the simple design. Although RABI uses only one gait, the tripod gait, it is capable of simple walking and pivoting without adding additional complications to the hardware. The use of neuron networks keeps the implementation simple and proves to be adequate for walking purposes. Furthermore, the network design allows easy interface with additional circuitry from higher level functioning such as instinctive behaviors.

Chapter 4
Instinctive Behaviors

All animals are endowed with some sort of instinctive behaviors which provide basic functioning and reflexes required for survival. An *instinct* is a type of behavior that does not depend on experience. These behaviors are a result of evolutionary progress. The genes from the ancestors that survive are passed onto future generations allowing the instinctive behaviors to be fine tuned from generation to generation. The behaviors are fine tuned for the environment in which the animal lives.

Since these instincts provide basic functioning, it is reasonable to program them into robots as a means of controlling the robot' s low level decisions and actions. The first step required for this type of programming is to identify the basic behaviors (instincts) needed to keep the robot functioning. The robot must at least have instincts for avoiding injuries, exploration and obtaining food if it is to survive in an environment. These behaviors provide a foundation on which to build higher level behaviors for a more versatile and competent robot.

Usually, robots are designed to operate in specific environments. If this is the case, then it is possible to program specific instincts suited for this environment. Once programmed, all that remains is to organize them in such a way to be able to handle situations in which two or more conflicting behaviors arise. The resulting robot (agent) should be able to survive to some extent in the environment in which it functions.

4.1 Avoiding Injuries

Locomotion gives a robot the ability to translate its body from point to point. This is necessary for autonomous systems, moreover, it is necessary for survival. It can however, produce a variety of hazardous situations which would not have occurred if the robot were stationary. Terrain features such as uneven surfaces, obtrusions, ruts, holes, liquids, non-supporting surfaces, etc. all provide a level of danger for locomotives by threatening to induce bodily harm through collisions, tipping, rolling, submersion and plummeting. Moreover, quick moving locomotives are further threatened by high impact collisions. For simple, flat, indoor environments, the most common injuries would arise from collisions with obstacles.

Most animals and insects have a mechanism that prevents them from hitting obstacles in order to avoid self injuries. Likewise, an autonomous mobile robot must have some means of detecting an obstacle in its path and turning appropriately to avoid contact. This brings up two issues, that of *obstacle detection* and *collision avoidance*.

4.1.1 Obstacle Detection

In order to detect an obstacle, some kind of sensor information must be processed. Animals generally use visual information when available since it provides a wealth of data. Animals and insects that rely on visual sensory information often have difficulty in preventing a collision with transparent objects such as glass. Flies, moths and mosquitoes, for example, are often seen flying head-on into obstacles such as lights and windows. Any autonomous robot relying on solely *visual* [15] information would also collide with transparent objects. The main drawback of visual sensors is their expense and the processing power required to extract useful information from noisy images.

A more widely used approach to obstacle detection is that of *ultrasonics.* Bats use this type of strategy to measure distances. An ultrasonic burst is emitted in a direction towards the obstacle. The time that it takes for the burst to return is measured, giving an indication of the distance to the object. These electronic ultrasonic devices offer good range data but offer poor directionality since they detect obstacles inside a cone of approximately 30 . Multiple objects cannot be detected inside this cone. Usually, multiple sensors are arranged in a ring fashion providing a 360 panoramic view, resulting in a low resolution obstacle detection system. Another problem with these sensors is that they often receive noisy data due to spurious reflections from smooth surfaces, previous bursts and crosstalk from other sensors. To help

[15] Visual information extracted from a camera or similar sensor.

alleviate the problem, occupancy grids or histograms [16] are used to determine the likelihood that an object is present.

Optical range finders are another kind of sensor used on robots. Essentially, they emit a beam of infrared light which is reflected back if an obstacle is present. These sensors require fine tuning to their environment and are susceptible to ambient infrared light such as that which comes through a window from outside. As a result, the incoming data is noisy and must be filtered. Typical range finder data for a 360 view requires about a second to obtain. [Cox 91] discusses the use of a range finder for navigation.

While all these sensors are being used on large "garbage can" sized robots, they are not able to be incorporated into smaller robots. Furthermore, smaller robot life forms are more likely to require very quick reflexes in order to survive. The 1 second delay for collision avoidance may be fatal for the robot.

For these reasons, designers of small robots are turning towards simpler sensors such as proximity, antennae and touch sensors [Fylnn 87], [Brooks 89], [Beer 90], [Beer and Gallagher 92], [Koza and Rice 92] and [Mahadevan and Connell 92]. These simple sensors are less prone to noisy data and can be made very small for use in small microbots or nanobots. Antennae sensors are a form of mechanically activated proximity sensors found on all insects and in other forms on animals [17]. These sensors are better than "bump" sensors in that they do not require the robot' s body to contact the obstacle. They provide a flexible kind of physical interface with the robot and the obstacle.

RABI incorporates 6 antennae sensors as shown in Figure 4.1. The four frontal antennae are used to detect collisions with obstacles during forward motion. The side antennae are used to detect contact with an edge during edge following behavior. The side antennae are not used for collision avoidance since the detection of obstacles from either of these antennae does not indicate an obstacle obstructing the forward motion of the robot.

[16] [Borenstein and Koren 91] and [Lang et al. 89] give descriptions on ultrasonic ranging, occupancy grid and histogram techniques.

[17] Some animals have whiskers and hairs which have essentially the same performance as antennae with the exception that some antennae are used as actuators which can "feel" obstacles. The antennae can obtain active data as opposed to passive data as seen with hairs and whiskers.

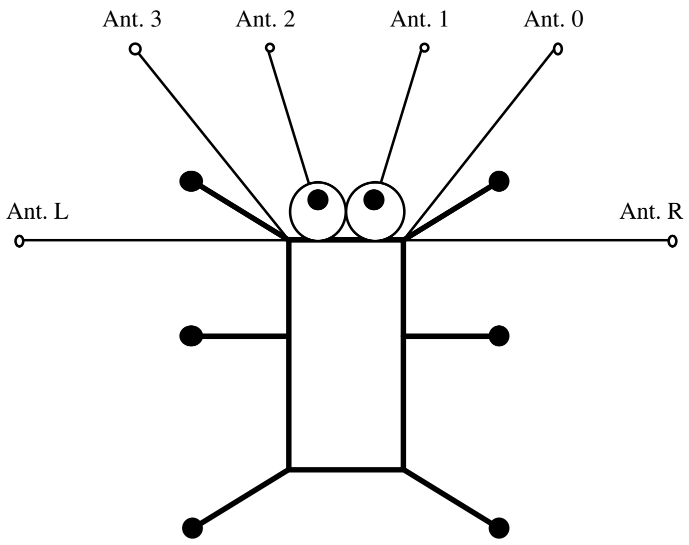


Figure 4.1 The 6 antennae incorporated into RABI.

The antennae on the hardware version of RABI are made of piano wire which provides a flexible binary "switch" whenever an object brushes up against it. As a result, the antennae are able to detect obstacles before the body becomes too close. The software version simulates the antennae as points representing their tips. These points are at a fixed distance away from the robot and therefore they do not have the same bending/flexible property of the hardware version. They do however, provide a quick means of detecting collisions with obstacles in the simulation. Four basic obstacle features can be detected as shown in Figure 4.2.

Since there are only 4 antennae at the front of the robot, there is a limit to the resolution that can be detected. Consequently, such an arrangement is not able to detect small obstacles which may pass between the antennae. In essence, altogether they can provide 4-bits of data depicting features of the obstacle encountered.

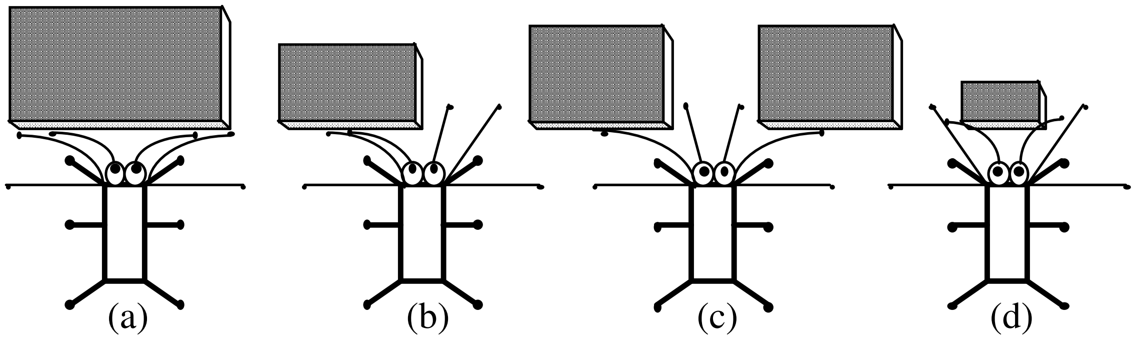


Figure 4.2 The four basic types of detectable obstacle features. (a) large obstacle surface detected, (b) obstacle corner detected, (c) corridor detected and (d) small obstacle detected.

4.1.2 Collision Avoidance

Once an obstacle is detected in the path of the robot' s motion, some sort of *collision avoidance* [18] motion must be executed in order to prevent the robot' s body from coming in physical contact with the obstacle. The ability to detect some low resolution obstacle features allows the robot to respond in a more efficient manner. The detection of these features calls for two basic collision avoidance responses. In Figure 4.2(a), 4.2(c), and 4.2(d) the detection of the obstacle is spread symmetrically among left and right antennae. In such a situation, the robot should randomly chose to turn left or right to avoid hitting the obstacle [19]. In Figure 4.2(b) the robot detects an obstacle on one side of its body only. The appropriate response here would be to turn in the direction in which the obstacle is not detected. That is, for the situation in 4.2(b) the robot would turn right to avoid the obstacle detected on its left. If the robot were to turn left, then it would be walking into the obstacle, defying the purpose of collision avoidance altogether.

Figure 4.3 shows a table of the 16 possible frontal antennae readings and the appropriate response needed to avoid contact. A blackened box indicates antenna contact with an obstacle. From the figure, it can easily be seen that most of the collision avoidance task results in a specific left or right turn, only choosing a random turn when there is no distinction between left and right antennae readings. Furthermore, it can be seen that the direction chosen is indicated by a majority of antennae contact readings (i.e. blackened boxes from Figure 4.3) on the opposite side of the body.

[18] The term *collision avoidance* used here indicates a simple directional change in order to avoid contact with an obstacle lying directly ahead. This term as well as the term of *obstacle avoidance* has been previously used to describe the notion of avoiding multiple obstacles while incorporating a path planning strategy. The use of the term here does not touch upon any aspect of path planning.

[19] If some sort of navigational behavior is present then this behavior may render a desired direction of travel whenever the robot is faced with making a random turn.

LEFT	RIGHT	RANDOM	NONE
□□□■	■□□□	■□□■	□□□□
□□■□	□■□□	□■■□	
□□■■	■■□□	■■■■	
□■□■	■□■□		
□■■■	■■■□		
■□■■	■■□■		

Figure 4.3 The 16 possible frontal antennae readings and the appropriate response needed to avoid contact. Each set of 4 boxes represents the antennae readings where the rightmost box is from the rightmost antenna.

The task of collision avoidance is more than providing an appropriate directional response for each antennae state encountered. With only a direct sensor to action response, the robot may enter into an oscillating pattern when encountering a corner as shown in Figure 4.4. When at a corner, the antennae on one side will detect an obstruction. To avoid collision, the robot will turn away from the detected obstruction, causing the antennae on the opposite side to detect the other wall of the corner. Consequently, the robot will again turn back towards the original obstruction and the oscillating pattern will begin.

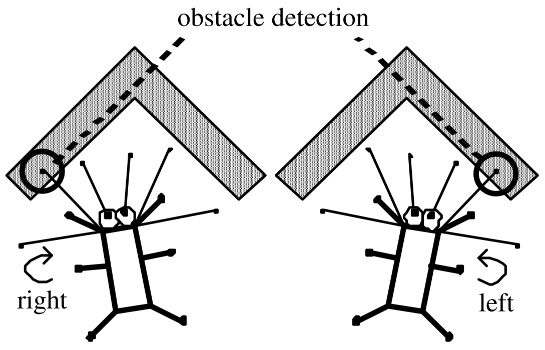


Figure 4.4 Diagram showing the oscillating turning problem encountered during collision avoidance at a corner.

RABI is endowed with a reflexive collision avoidance behavior which provides a control mechanism for turning away from the obstacle in the appropriate direction. This mechanism provides the appropriate responses as shown in Figure 4.3 while avoiding the oscillating turning pattern shown in Figure 4.4. Figure 4.5 shows the neural circuitry to accomplish the overall collision avoidance behavior. The circuit reads in the status of the four frontal antennae and decides the appropriate response in order to avoid encountered obstacles. Since the circuit contains two sustain neurons, it provides a temporal response as opposed to a direct sensor to action response.

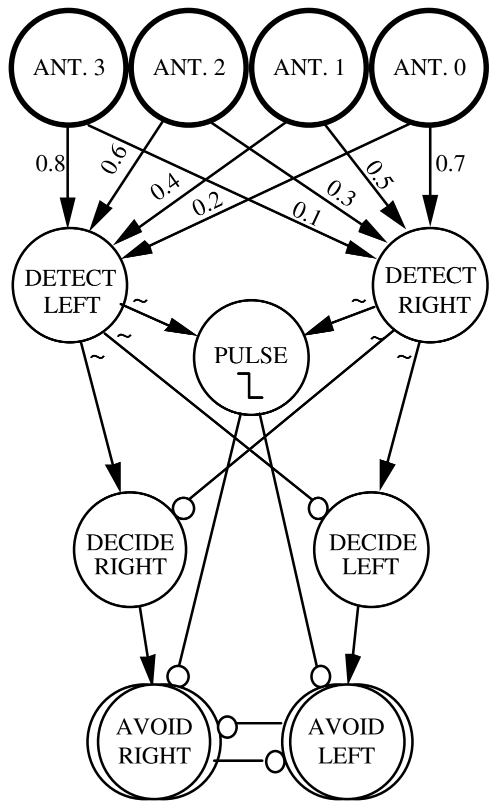


Figure 4.5 The neural circuitry depicting the collision avoidance behavior.

The sensor neurons read in the status of the four frontal antennae, ANT. 0 being the rightmost antenna. These sensor neurons connect to standard neurons DETECT RIGHT and DETECT LEFT which act as summing neurons giving an indication of the amount of detection on each side of the body. Since they are standard neurons, they have analog (real number) output which is fed into the DECIDE RIGHT and DECIDE LEFT neurons. These two neurons act as "majority" operators in that they become enabled whenever the majority of the sensor readings are obtained from their side of the body. These neurons connect to the AVOID RIGHT and AVOID LEFT neurons, turning on the appropriate directional response.

The AVOID RIGHT and AVOID LEFT neurons provide the resultant action desired by the collision avoidance behavior. Moreover, the neurons mutually inhibit each other such that only one direction can be active at a time. This allows the cornering problem to be alleviated by essentially choosing a direction to turn in and "sticking with" the decision until free from obstruction. The remaining PULSE neuron provides a method of turning off the two sustain neurons whenever the sensors stop reading collisions.

In the case when the sensor readings do not bias on either side, RABI chooses a right turn. In reality, an insect would more than likely chose a truly random turn with other external and internal factors playing a role in the decision. However, in the absence of additional internal and external information, a right turn is equally acceptable to a left turn. Therefore, to simplify the circuitry by avoiding the use of additional neurons for random turning, a right bias was chosen. This bias can be seen by observing the weights of the links from the sensor neurons. The weights are stronger for those detecting obstacles on the left, which results in a right turn dominance for collision avoidance. With these weights there is no sensor situation which will result in equal detection on the left and right sides, except when there is no obstacle detection at all. Thus, a left or right decision will be made even for symmetric sensor readings. Moreover, the weights have been chosen such that the antennae provide biased readings for detection on the left and right side. For example, the leftmost antennae ANT.3 provides a strong excitatory signal to the neuron which detects left obstacles and a weak excitatory signal to the neuron which detects right obstacles.

The network allows RABI to walk around in the environment without hurting itself by avoiding collisions with walls. The AVOID LEFT and AVOID RIGHT neurons are connected directly to the TURN LEFT and TURN RIGHT neurons of Figure 3.13 that control the turning mechanism of RABI' s walking behavior. This connection will allow the robot to walk forward turning away only when an obstacle is detected in its path. This collision avoidance technique has been designed to avoid collisions with stationary obstacles. The information would need to be processed dynamically in order for the robot to safely avoid moving obstacles. The antennae neurons themselves do not provide the dynamic information necessary for the detection of moving obstacles.

4.2 Wandering : A Basis for Exploratory Behavior

Perhaps the most basic form of exploratory behavior is that of *wandering*. By wandering, a robot is able to obtain information from environmental locations that may never have been reached with calculated paths. If, for example, a robot has been programmed to weave between obstacles while performing a point to point path plan, it may never detect environmental stimuli lying along edges of the obstacles. These environmental stimuli may be critical to the survival of the robot (i.e. power source). By allowing the robot to wander around in the environment, it is more likely to find such important stimuli [20].

4.2.1 Implementing Artificial Wandering Behavior

For a non-changing environment, truly random wandering will eventually cover the entire accessible portion of the environment. Although random wandering is a means of searching for stimuli, most animals are more efficient than this, in that they rely on other information. Some insects such as ants, emit a chemical residue that allows them to detect where they have been before. This additional information could prevent the ant from wandering into previously explored regions. [Steels 90] describes a simulation in which a robot is to map out an environment through wandering behavior. The robot is able to leave behind a trail indicating previous path choices. The wandering behavior is random only when there is a lack of information. That is, the robot is able to sense where it has been before and therefore chooses to wander into locations that have not been visited.

Computers and electronics are capable of producing reasonably random effects which can be used as a control mechanism for wandering behavior. All that is required for random wandering is a mechanism that causes occasional directional changes in the robot' s forward motion. Although these random directional changes produce a wandering illusion, this effect is not necessarily observed in real insects. Cockroaches, for example, utilize a combination of circling and relatively straight movements in its wandering behavior [Bell and Adiyodi 81]. Nevertheless, a form of random wandering provides a reasonable coverage of the environment.

[20] An assumption is made here that the robot has no prior knowledge as to where the critical stimuli are located, and therefore may only encounter the stimuli "by chance".

[Beer 90] incorporates a wandering network in his simulated insect in which two neurons randomly emit bursts of energy that directly control the turning mechanism of the insect. This approach requires only two additional neurons which directly connect to the turning mechanisms. RABI uses a similar approach by using a simple neural circuit as shown in Figure 4.6, containing three random neurons that produce a form of random wandering behavior.

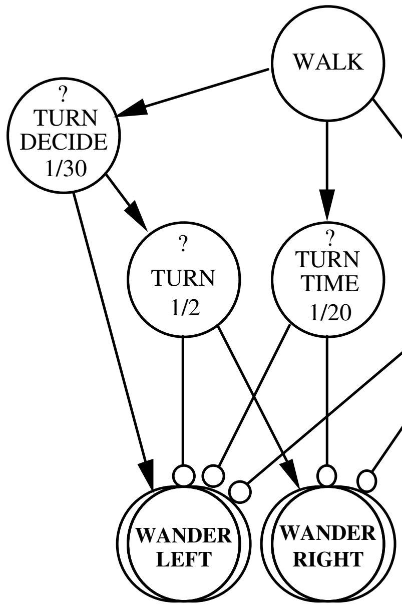


Figure 4.6 The neural circuitry depicting the wandering behavior The WALK neuron enables the TURN DECIDE and TURN TIME neurons that, through random output, selects when and for how long to turn.

When the circuit is enabled by exciting the WALK neuron, it randomly enables and disables the WANDER LEFT and WANDER RIGHT neurons which are used as directional change mechanisms providing wandering behavior. The WALK neuron acts as an "on" switch which enables the TURN DECIDE and TURN TIME neurons that select when and for how long to turn respectively. The TURN DECIDE neuron, when excited, emits a binary high output with a probability of 1/30, otherwise emitting a low signal. The output of this neuron is used to enable the TURN neuron which essentially chooses the direction to wander in. It does this by enabling the appropriate sustain neuron. The TURN TIME neuron is a random neuron that emits a binary high output with a probability of 1/20. This output is used to disable the turning process by inhibiting both sustain neurons WANDER LEFT and WANDER RIGHT. Finally, the PULSE neuron emits a burst whenever the explore neuron is turned off which inhibits the sustain neurons so that the wandering influence to the overall system is eliminated.

The wandering behavior becomes functional by connecting the WANDER LEFT and WANDER RIGHT neurons directly to the TURN LEFT and TURN RIGHT turning control neurons of Figure 3.13. The result is a form of wandering behavior that allows the robot to walk forward while occasionally turning off into random directions.

4.2.2 A Wandering Bias Towards Vacant Areas

Some animals rely on visual information such as landmarks and "open" areas that provide a global sense of location which may directly affect the actual wandering behavior. If an animal has previously found certain stimuli (i.e. food) in an open area, then it may chose to steer clear from all obstacles. In this case, the animal may choose to explore "open" (vacant) locations as opposed to "crowded" (obstacle ridden) locations with its wandering behavior. [Anderson and Donath 90] describe a method of using various attraction behaviors that allows a robot to wander in and out between obstacles such that the robot tends to explore the vacant areas of the environment. The implementation of their robot utilizes ultrasonic and camera sensors which are not feasible for nanobot purposes. Nevertheless, the idea of repelling from obstacles in order to remain in vacant areas can prove to be quite fruitful if the environment contains most of its stimuli in these vacant areas. [Beer 90] utilizes a recoiling feature that allows his simulated insect to "bounce away from" encountered obstacles. In his simulation, the insect first backs away from the obstacle and then turns off into a new direction. The result is not unlike the familiar 3-point turning technique of vehicle driving. His approach is based on observations of cockroach recoil. This technique requires that the robot be able to move backwards as well as being able to advance and turn simultaneously providing a nonlinear trajectory. This provides further complications in the control of the insect' s walking behavior and thus a simpler approach was used by RABI.

A robot equipped with simple proximity or touch sensors does not have the ability to detect obstacles until it is very close to them. Moreover, simple sensors do not provide distance information that may be needed to steer away from upcoming obstacles. RABI can make use of all six of its antennae sensors to detect obstacles and turn away from them. The basic collision avoidance behavior incorporated by RABI allows the robot to turn away from obstacles that it encounters in its path until it no longer detects the obstacle. This collision avoidance behavior provides part of a *vacancy* behavior in that it turns the robot away such that the obstacle does not obstruct the path. Usually, once the robot has turned using the collision avoidance behavior, a side antennae remains in contact with the obstacle. All that remains to do is to further turn away until the side antennae sensors detect no collisions either, and then head off away from the obstacle. Figure 4.7 shows the 3 basic steps that specify vacancy behavior.

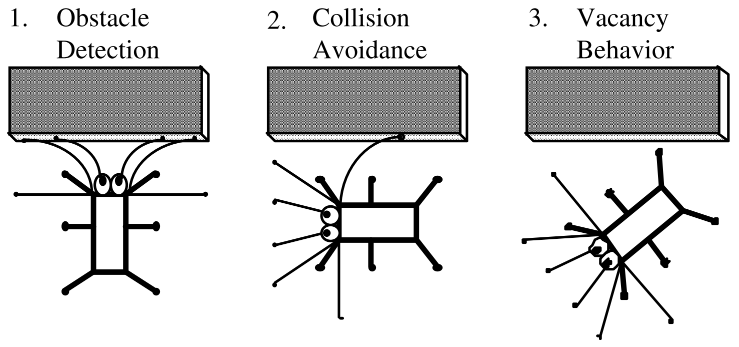


Figure 4.7 The 3 basic steps of vacancy behavior. 1) Obstacle is detected. 2) Collision avoidance turns to prevent collision. 3) Vacancy behavior makes additional turns to repel the robot from the obstacle.

This vacancy behavior incorporates the collision avoidance instinct. When the robot detects an obstacle in its path, the collision avoidance instinct causes the robot to turn away. Once the robot has turned such that its front antennae no longer touch, the collision avoidance behavior ceases. If the vacancy behavior is enabled, then this behavior would make additional turns such that the robot' s side antennae also do not touch the obstacle. Once the antennae no longer sense obstacles, it wanders forward into the vacant area by heading away from the obstacle.

Implementation of the vacancy behavior requires that the left and right side antennae be observed for collisions with obstacles. Since the collision avoidance instinct handles all collisions, then this behavior need not worry about the frontal antennae collisions. In fact, for any frontal antennae collisions, the collision avoidance behavior should "kick-in" and maneuver the robot to avoid collision. RABI does exactly this. A neural circuit providing the vacancy behavior is depicted in Figure 4.8.

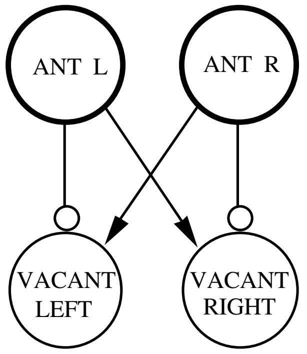


Figure 4.8 Neural circuitry depicting the vacancy behavior. The network allows the robot to turn away from any detected obstacle along its side such that it heads back into vacant areas of the environment.

In this circuit, ANT L and ANT R are sensor neurons corresponding to the side antennae. They are able to turn on the vacancy behavior by exciting the VACANT LEFT or VACANT RIGHT turning neurons. The interconnections are made such the robot turns in the direction opposite that which detects an obstacle. That is, if an obstacle is detected on the left, the robot turns right towards vacant territory. When both side antennae detect obstacles (i.e. a narrow tunnel or corridor) then neither turning neuron is excited and the robot proceeds straight ahead. This allows the robot to exit the corridor into a more vacant area. Once again, the vacancy behavior is consummated by connecting the VACANT RIGHT and VACANT LEFT neurons to the TURN RIGHT and TURN LEFT neurons of Figure 3.13.

4.3 Edge Following Behavior

A robot that wanders around aimlessly avoiding obstacles does not learn anything about the structure of its environment. Sometimes stimuli may be known to exist along obstacle borders, such as power outages and wall sockets in a typical indoor environment. If a robotic life form is to survive in such an environment, then clearly there is a need to narrow the search space when looking for such stimuli. One possible method of reducing this random searching is to provide the robot with an *edge following* behavior. Such a behavior would allow the robot to follow along obstacle boundaries (i.e. walls, boxes, foundations etc.) keeping its body parallel to the obstacle. Such an instinct is more suitable for traversing complex environments than random wandering. Moreover, it is also biologically plausible since [Bell and Adiyodi 81] state that cockroaches in particular are known to spend the majority of their time within antenna contact of an edge. This edge following behavior also allows environmental mapping to occur.

The main issue in edge following behavior is keeping the body close and parallel to the edge. If the robot begins moving away from the edge, it must turn back towards it such that it realigns itself. Likewise, if the robot begins to head into the obstacle it must turn away slightly to avoid a collision. This indicates that some sort of angle measurement may be needed for efficient parallelism. Finally, if the robot loses contact with the edge, it should attempt to regain contact again.

[Beer 90] uses a simple neural circuit for edge following behavior with his simulated insect. However, the antennae of his insect are able to produce analog output such that a direct head-on collision of the antennae provides a strong output signal while a brush against an obstacle provides a weaker signal. Consequently, this analog output gives the insect an indication of the angle that the antennae makes with the edge. Since the antennae are fixed on the body, this provides a kind of measurement of the angle between the edge and the insect's body. With RABI's simple binary antennae, this is not possible since the binary output does not provide any indication of the angle between the edge and the body. Instead, RABI uses its front antenna closest to the edge to detect when the robot comes too close to the edge. Veering away from the edge is detected when the side antenna loses contact.

4.3.1 Modes of Edge Following

There are three basic "modes" of edge following behavior. The first mode is the *follow edge* mode in which the robot just walks forward as long as the side antenna remains in contact with the edge and no frontal obstructions are detected. If the robot encounters a head-on obstacle obstructing its forward motion, then the robot must *orient* itself to this new obstacle's edge. Lastly, if the robot then loses contact with the edge, it must turn to regain contact in what is called the *realign* mode.

When in the follow edge mode, no special behavioral changes occur since the robot is merely walking forward along an edge. The directional adjustments must be made only when the robot loses edge contact or when it encounters an obstacle. The loss of contact can occur for one of two reasons. Either the robot is slowly moving away from the edge because it is not quite parallel, or the features of the boundary being followed has changed (i.e. a convex corner). These two situations are depicted in Figure 4.9(a) and 4.9(b).

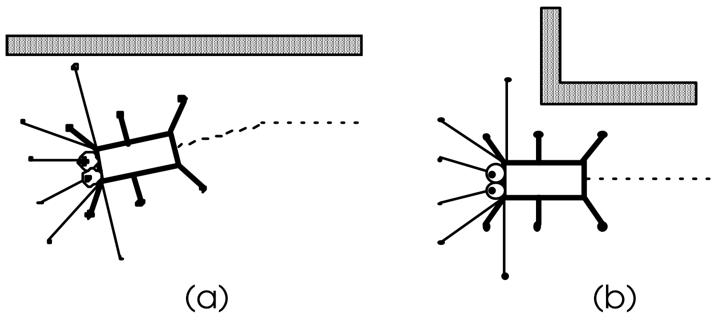


Figure 4.9 Situations causing loss of edge contact. (a) misalignment, and (b) obstacle feature changes (convex corner).

In (a), the robot must turn back to regain contact with the edge. In (b), the robot must also turn back towards the edge but will end up re-gaining contact with a different edge on another side of the obstacle. Usually, a sharp corner such as 90 will require a sequence of realignment steps since the side antenna will continually lose and regain contact with the edge during the sharp turn. This amount of fluctuation between edge follow and realign modes can be reduced by positioning the side antennae further back towards the rear of the robot. The further back the side antenna is placed, the longer it will take the robot to detect a loss of edge contact when traveling past a convex corner. Consequently, the robot will be further out from the obstacle when it starts the realignment. If far enough out, the robot may require only one realignment phase. There is a limit to how far back the antennae may be placed, however, since if the robot travels too far out past the obstacle, it may not be able to regain contact with the edge. Figure 4.10 depicts antenna placement constraints.

The diagram shows the distances from the pivotal center [21] to the tip of a frontal antenna and two possible side antennae. Antenna A_1 is suitable since its distance d_1 is less than d_0 even when fully bent backward. Antenna A_2 however, is not valid since when fully bent backward, its distance d_2 exceeds d_0. With A_2 as a side antenna, the robot would walk too far past the convex corner before losing contact and the robot would not be able to regain contact with the edge through turning.

[21] The pivotal center is the point from which the robot pivots during a turn. Usually this is the center of the robot if the robot is constructed symmetrically and efficiently.

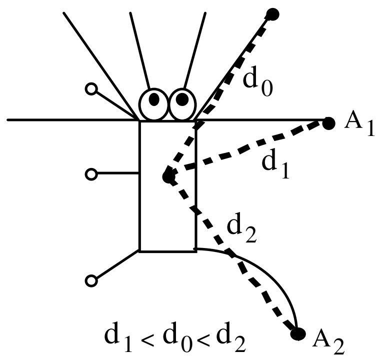


Figure 4.10 Placement of a side antenna. A_1 is safely placed since the distance d_1 does not exceed d_0 but when A_2 is bent, its distance d_2 exceeds d_0. Thus, A_2 should be moved forward.

If the robot detects any frontal antennae collisions while engaged in edge following behavior, it should turn away from its followed edge and re-orient itself to the new edge encountered, then continue on. There are a variety of frontal collisions that may occur and these are depicted in Figure 4.11.

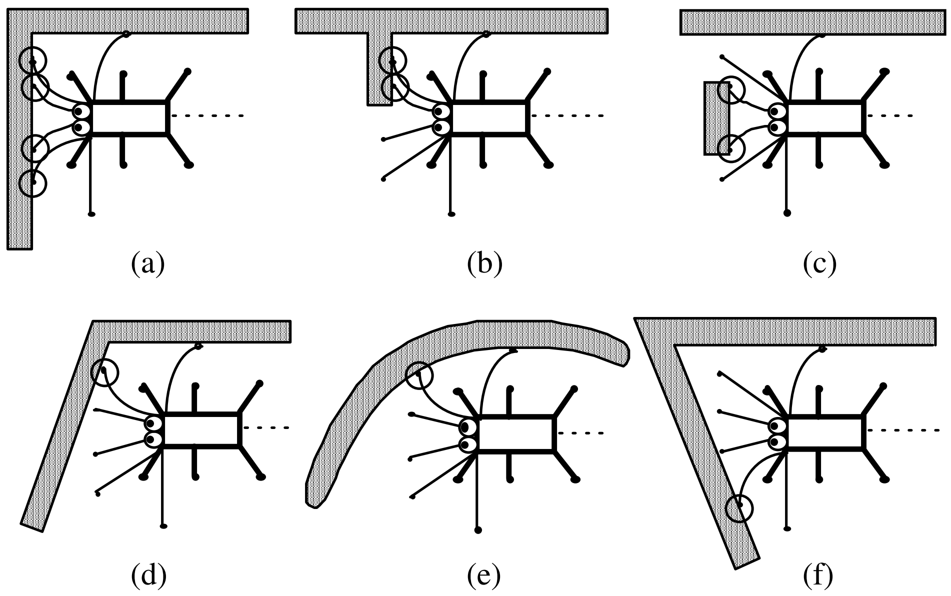


Figure 4.11 The different types of frontal antennae collisions calling for reorientation to a new edge. (a) full surface contact, (b) edge obtrusion, (c) small object contact, (d) obtuse angular edge change, (e) curved surface, (f) acute angular edge change.

In (a), the robot makes full contact with its front antennae. A similar situation appears in (b) and (c) with the exception that only a couple of its sensors make contact. The situations in (d), (e) and (f) are all similar in that only an outer antenna makes contact. In fact, since the antennae provide binary output, curved edges as in (e) will be recognized as a polygonal chain and thus (d) and (e) represent the same situation.

Each of these types of collisions call for the robot to reorient itself to the new contacted edge. Once realigned with the new edge, the edge following behavior continues. There are two additional situations involving the frontal antennae. In Figure 4.12(a), the robot is not aligned properly with the edge being followed and eventually becomes too close to the obstacle resulting in a collision with its outer antenna. Here, the robot merely needs to adjust itself to the edge by turning away until the collision no longer exists. This happens often since the robot' s antenna sensors provide no indication as to the distance from the edge. Consequently, alignment adjustments can only be made once the robot becomes too close and collides with the edge.

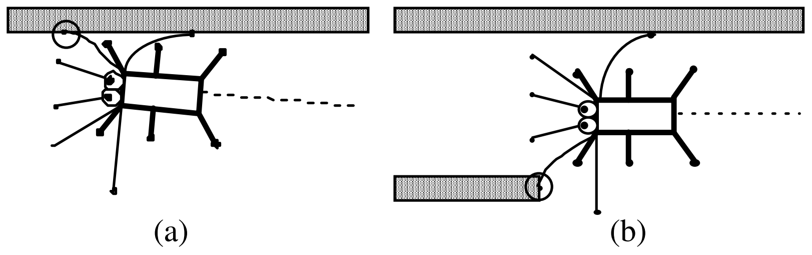


Figure 4.12 Additional frontal antennae collision situations. (a) misalignment with the edge and (b) a narrow passage.

In Figure 4.12(b), the robot' s outer antenna detects a collision. Here, it may be possible for the robot to *squeeze* in between the edge and the detected obstacle such as the case of narrow passages or tunnels. This squeezing may be a requirement if the environment is complex and dense with obstacles. If the squeeze is unsuccessful, due to additional antennae collisions, then the robot should assume that the passage is not traversable and treat the initially detected obstacle as an edge.

This "squeezing" process is necessary for efficient traversal in an unknown environment. Without this squeezing process, the robot could enter into an infinite looping process when following edges as shown in Figure 4.13. Here, the robot follows along the border of the environment. As it approaches the inner obstacle, the robot' s right-most antenna would make contact, causing the robot to turn right and follow along the inner obstacle' s edge. The robot will then remain circling the inner obstacle counter clockwise since there is no antenna collision causing it to join up with the original border. This would lead to inaccurate mappings of the landmarks.

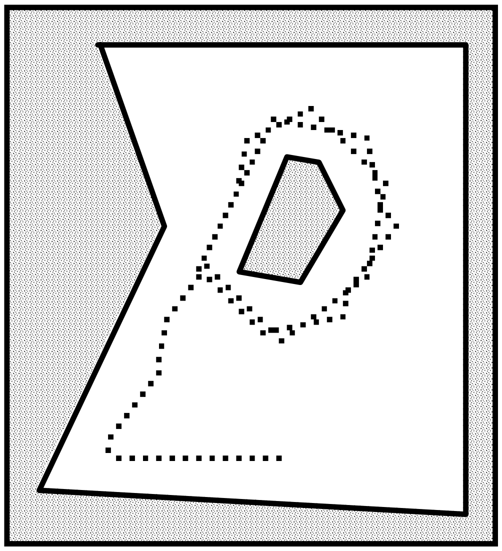


Figure 4.13 Circling problem during edge following without the squeeze mode.

A more important problem that may be encountered is that of entrapment. The robot may be able to fit easily through a passageway when coming from one direction but not from the other. Thus, the robot may get trapped in an area that it cannot get out of. Figure 4.14 shows such a situation where the robot becomes trapped in the center of a spiral-shaped environment. In fact, this kind of entrapment happens to animals such as insects; spiral designs are even used in some fish traps.

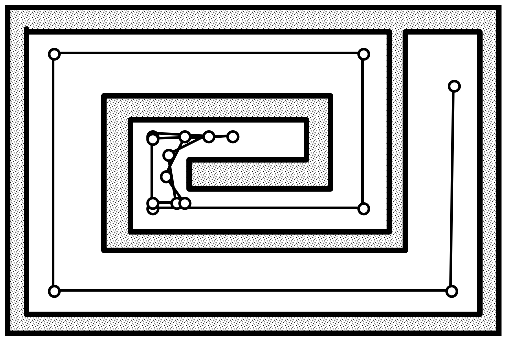


Figure 4.14 Entrapment problem during edge following without the squeeze mode.

Here, the robot' s constructed mapping is shown with line segments and circles representing edges and corners [22]. The robot is able to travel into the spiral center, but when attempting to follow the edge to get out, the frontal antennae collide causing the robot to turn back inwards. A similar circling pattern emerges. This presents a problem since a trapped robot is likely to run out of energy and die. Giving the robot the ability to squeeze will prevent this entrapment from occurring.

One final problem that may occur without a squeeze facility is that of discrepancies in mapping representations. That is, the robot may be able to squeeze into certain areas on some occasions, and unable in others. Thus, edge lengths and corners may vary depending on the width of passageways in the environment. A narrow environment is shown in Figure 4.15 along with two mappings constructed from RABI before the squeeze mode was developed.

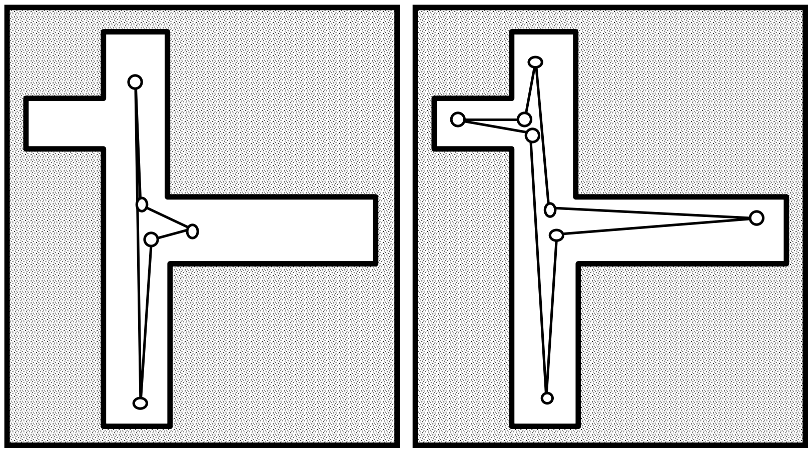


Figure 4.15 Mapping discrepancies caused by narrow passageways.

In the first mapping, the robot was unable to fit into the narrow passages, where as in the second mapping the robot was able to enter and map out the passages. Note that the right passage was mapped in both cases but since the robot went into the passage at a different angle, the mappings are different. All three problems just mentioned can be solved by giving the robot an ability to squeeze into narrow passages.

The three modes of edge following combined with an additional mode for attempting a squeeze are all required for a generalized edge following behavior. The interactions between these modes can easily be shown by use of a state diagram. Figure 4.16 shows the state diagram

[22] The map building strategy is discussed further in chapters 5 and 6.

of the 4 modes required by RABI' s edge following technique. This state machine solution has the ability to handle all the situations mentioned above.

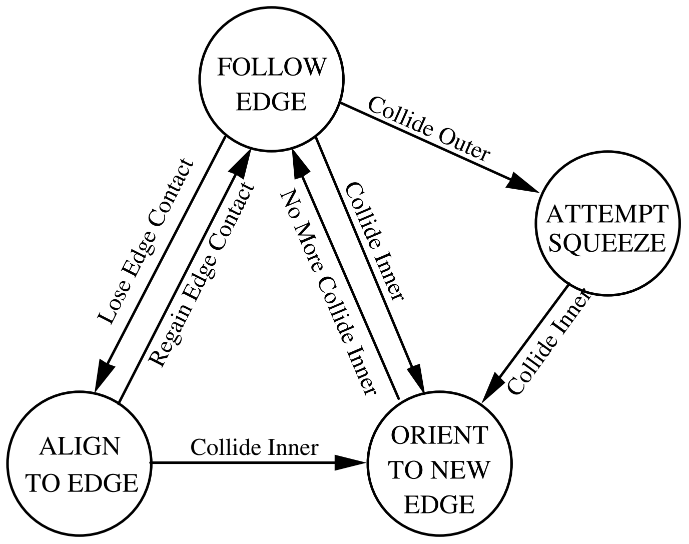


Figure 4.16 A state diagram depicting the edge following process. The attempt squeeze is an additional mode required for the robot to maneuver into tight situations.

The circles in the diagram represent the modes (states) while the links indicate the events required to transfer from one state to another. The terms "collide inner" and "collide outer" represent the events in which one of the 3 antennae closest to the edge and the one antennae furthest from the edge detect a collision respectively. The terms "lose edge contact" and "regain edge contact" represent the detection state of the side antenna that is being used for edge following. The remainder of the diagram is self-explanatory.

4.3.2 A Neural Implementation of Edge Following

The state diagram of Figure 4.16 depicts the edge following process and the events required to switch between modes. Direct programming of this state machine may not be a robust approach to implementing the behavior. There may be a collection of unexpected situations which can arise that may result in poor edge following behavior and perhaps even failure. Since RABI is capable of collision avoidance, it may be useful to use this behavior as part of the edge following process. For example, if the robot is following an edge and encounters an obstacle, then the obstacle avoidance behavior could "kick-in" to turn the robot away from the obstacle [23]. With this in mind, a neural circuit was designed which resembles the state diagram of Figure 4.16. A neural circuit for following right edges is shown in Figure 4.17.

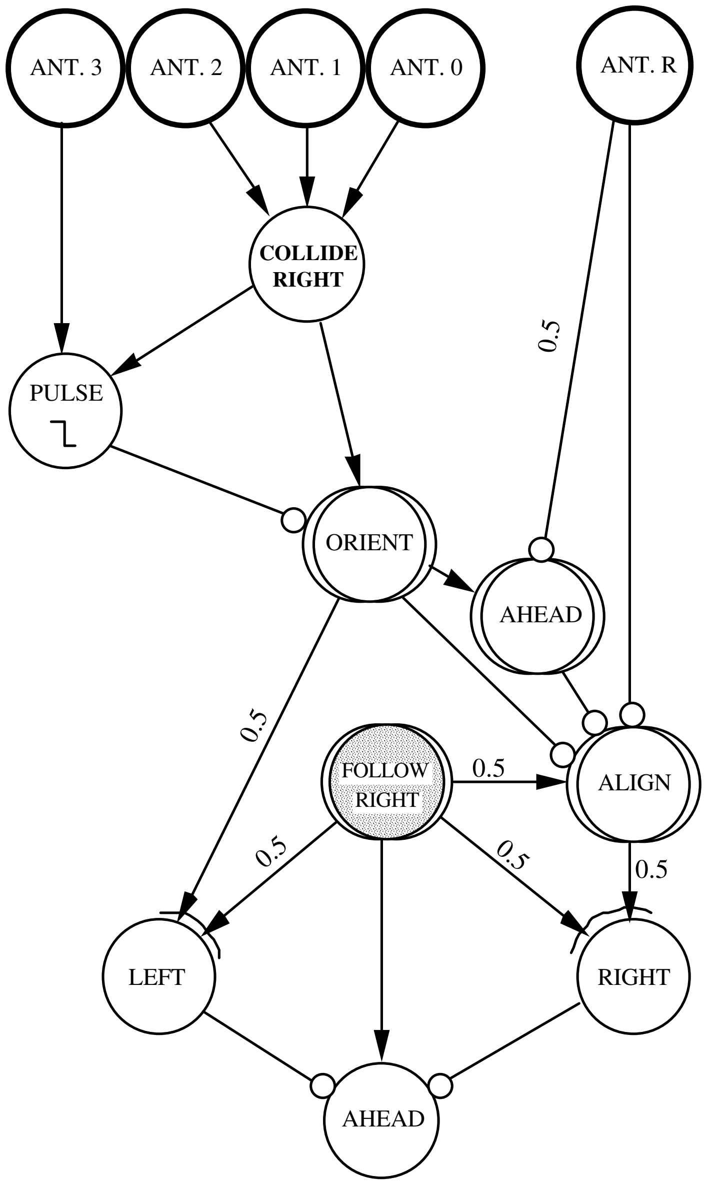


Figure 4.17 A neural circuit depicting right edge following behavior. The FOLLOW RIGHT neuron enables the circuit by providing the additional excitation required for the direction neurons LEFT, RIGHT and AHEAD.

23 The use of the collision avoidance behavior is required for the squeeze mode of the edge following behavior.

The upper part of the circuit utilizes the same sensor neurons as the collision avoidance network with an additional neuron ANT.R for the right side antenna. Another familiar set of neurons is the ORIENT, ALIGN and AHEAD sustain neurons which represent the "orient to new edge", "align to edge" and "follow edge" modes of the edge following behavior respectively. These neurons are readily identified as the 3 state neurons since they are sustain neurons which have the ability to retain energy, thus they can represent a "state" when switched on. The LEFT, RIGHT and bottom AHEAD neurons represent the direction in which the robot should travel during its edge following process.

The behavior begins when the FOLLOW RIGHT neuron in excited. This neuron is a behavioral neuron (shaded) that enables the 3 directional neurons. The ORIENT and ALIGN neurons decide when to turn LEFT and RIGHT respectively as in the state diagram. The robot only walks straight ahead if neither of the LEFT and RIGHT directional neurons are active. The rest of the connections follow directly from the state diagram where the ANT.R neuron decides if the robot loses contact with the edge, and the COLLIDE RIGHT neuron decides when the robot' s frontal antennae collide with an obstacle. Notice that the outer antennae is not included with the others during the detection phase. This is because the robot must be able to attempt squeezes in between narrow passages and therefore the outer antenna must be handled differently. In software, this antenna is ignored in this circuitry and handled by the collision avoidance circuitry. The PULSE neuron provides a method of disabling the edge orientation mode whenever the antennae stop detecting collisions (i.e. detects stop since it' s a falling edge neuron).

The circuit of Figure 4.17 allows the robot to follow edges on its right side once the directional neurons LEFT and RIGHT are connected to the TURN LEFT and TURN RIGHT neurons of Figure 3.13 as done for the other instinctive behaviors. If the robot wishes to detect obstacles on the left side of its body also, a similar symmetrical circuit is required. The combined left and right edge following behavior circuitry is shown in Figure 4.18. Here the edge to be followed is decided by the FOLLOW RIGHT and FOLLOW LEFT neurons which mutually inhibit each other since only one could occur at any moment. The remainder of the circuit represents a dual version of Figure 4.17.

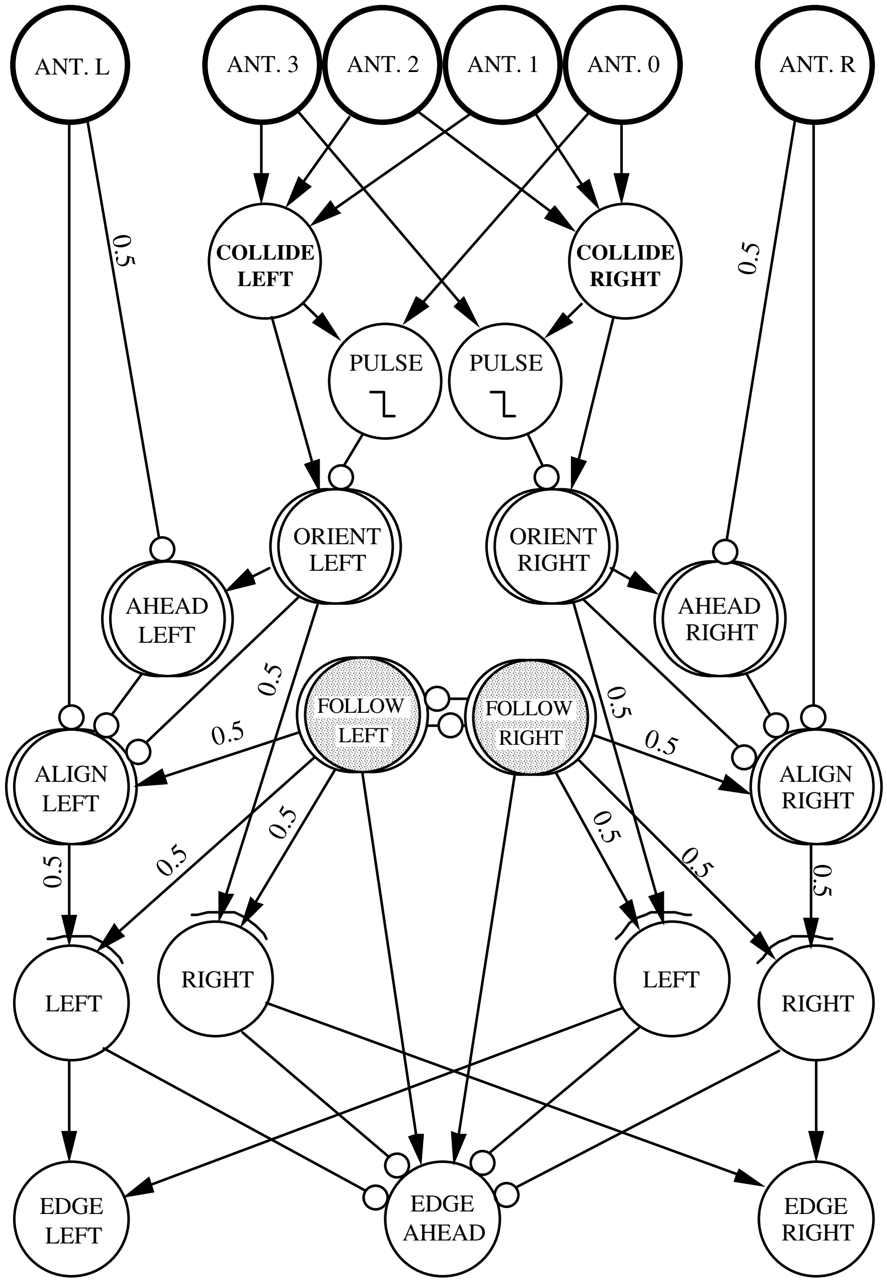


Figure 4.18 The combined left and right edge following circuitry. Two additional directional neurons were added in order to combine the LEFT, RIGHT and AHEAD decisions from both circuits into one set of signals.

4.4 Light Orientation: A Source of Energy

Many insects have a form of taxic behavior that attracts them to light (phototropism). The attraction towards light allows the insect to seek out warmer environments and even energy

[24]. Some insects avoid light (photophobic) since it makes them a better candidate for predation. This simple form of phototaxic behavior (photokinesis) requires two receptors that are sensitive to light. [Braitenberg 84] describes a simple crossover connection that essentially simulates fear and aggression taxic behaviors. In one of his examples, a simple 2-wheeled vehicle is controlled by this simple crossover. For attraction, the right receptor connects directly to the left wheel and the left receptor to the right. The more light received by the left receptor, the stronger the excitatory signal sent to the right motor, causing the vehicle to turn left towards the light. For light avoidance, the receptors are connected directly to the motor on their side, with no crossover.

RABI uses a similar crossover approach which is embedded into a neural circuit as shown in Figure 4.19.

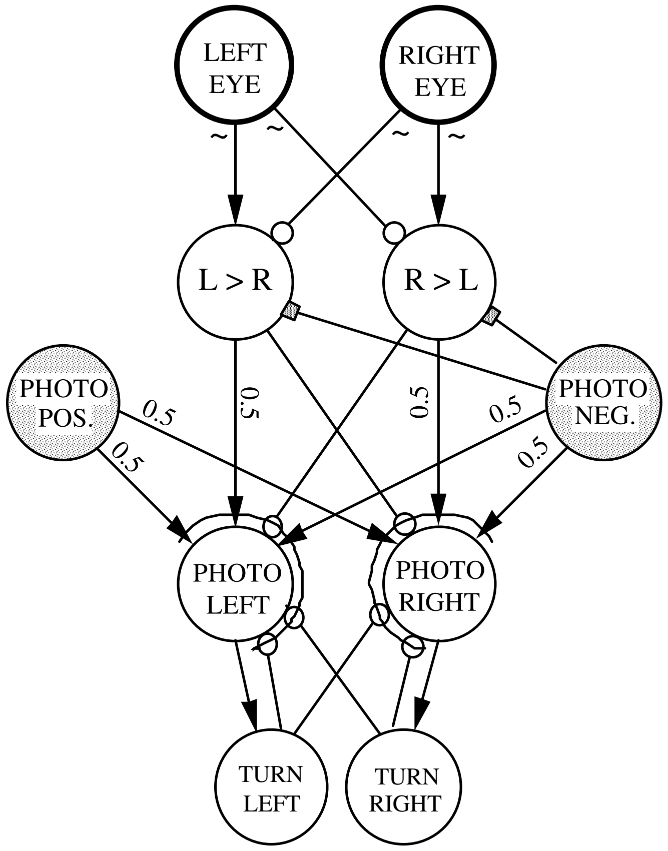


Figure 4.19 A neural circuit for photokinetic behavior.

[24] Plants seek energy from the sun.

The robot has two light sensors at the front of its body that are oriented 5 outwards from the center of the robot. Thus, the sensors are 10 apart from each other, and facing in opposing directions.

The LEFT EYE and RIGHT EYE are sensor neurons which are connected directly to the two light receptors. These neurons emit an analog signal representing the light intensity detected by the sensors. The L>R and R>L neurons detect which signal is stronger. This is done by comparing the two receptor readings. Since these neurons are binary, one of them will emit a high signal while the other emits a low. These two neurons then connect to two threshold neurons essentially providing the crossover as seen in the vehicles of [Braitenberg 84]. These two neurons connect to the TURN LEFT and TURN RIGHT neurons similar to the other behaviors. The PHOTO POS and PHOTO NEG neurons are controlling neurons that allow the behavior to be turned on or off. Note that the PHOTO NEG neuron also negates the two binary neurons. This allows a crossover to occur, reversing the behavior [25]. Additional inhibitory connections were added from the TURN LEFT and TURN RIGHT neurons preventing the robot from making consecutive turns in the same direction while seeking light. This was needed since the robot does not move forward while turning. Without these additional connections the robot would essentially toggle back and forth in one location without advancing. An example of the light following behavior is easily shown with a screen snapshot of the simulated environment as in Figure 4.20.

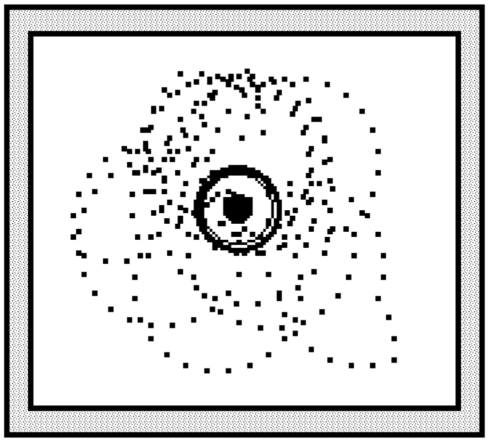


Figure 4.20 Screen snapshot of the phototropic behavior.

[25] This crossover is reversed from [Braitenberg 84] since the TURN LEFT and TURN RIGHT neurons provide a form of crossover themselves.

The snapshot shows the robot' s path (black dots) during 1 minute of performing the light attraction behavior. The two circles in the center represent the light source. Notice that the path consists of circular trajectories surrounding and occasionally crossing the light source.

One of the problems in implementing the light sensors is simulating the effects of the light sources in the environment. This simulation takes a simple approach in that the light is capable of passing through the environmental walls (i.e. glass walls). Clearly, any useful simulated light sensor data must depend on three factors: the distance from the light, the angle towards the light and the intensity of the light. Moreover, if there are multiple light sources, they must also be taken into account. The method for determining each sensor reading is as follows:

$$\frac{\mathop{\bullet}\limits_{i=1}^{\mathbf{n}} \text{abs}(\mathbf{a}_i) * \frac{1}{\mathbf{d}_i^{\mathbf{2}} - \frac{\mathbf{f}_i}{2}}}{\mathbf{n}}$$

where,

$\mathbf{n}$	is the number of light sources
$\mathbf{a}_i$	is the angle between the sensor and light source i
$\mathbf{d}_i$	is the distance from the sensor to light source i
$\mathbf{f}_i$	is the intensity of light source i.

This equation allows the robot to rely on the distance information when far away and on angular information when close to the source. Thus facing a close light source would contribute strongly to the overall signal, while facing a far light source will barely affect the signal. This allows the closest light source to have the greatest effect on the sensor. The intensity **f** plays a role in the distance information. If the robot becomes too close to the source (i.e. within half the intensity) then the equation gives a negative signal causing the light source to have a negative effect. This prevents the robot from staying directly on top of the source. In the simulation, the intensity is an integer from 10 to 100 representing the radius of a circle around the light in pixels. The distance is also in pixels.

4.5 Food Orientation: Finding and Absorbing Energy

Finding food is an essential behavior for all forms of life. In order to find the food, the robot must have an ability to orient itself towards the food source. With just simple sensors, this is more difficult than it seems. Many animals (and humans) use some form of "smell" sensor which produces a form of klinotaxis. The animal is usually required to wander around, making successive comparisons of sensor readings. This is essentially a form of hill climbing in which the robot has no accurate sensor readings indicating the direction of the food at any single location. Furthermore, smell sensors can usually only detect the odor when within a certain distance from the source, and this distance varies depending on the odor intensity. Thus, a mechanism for orienting towards the energy sources must be able to compare a sensor reading with the previous value in order to determine a course of action. Figure 4.21 shows a neural circuit that does just this.

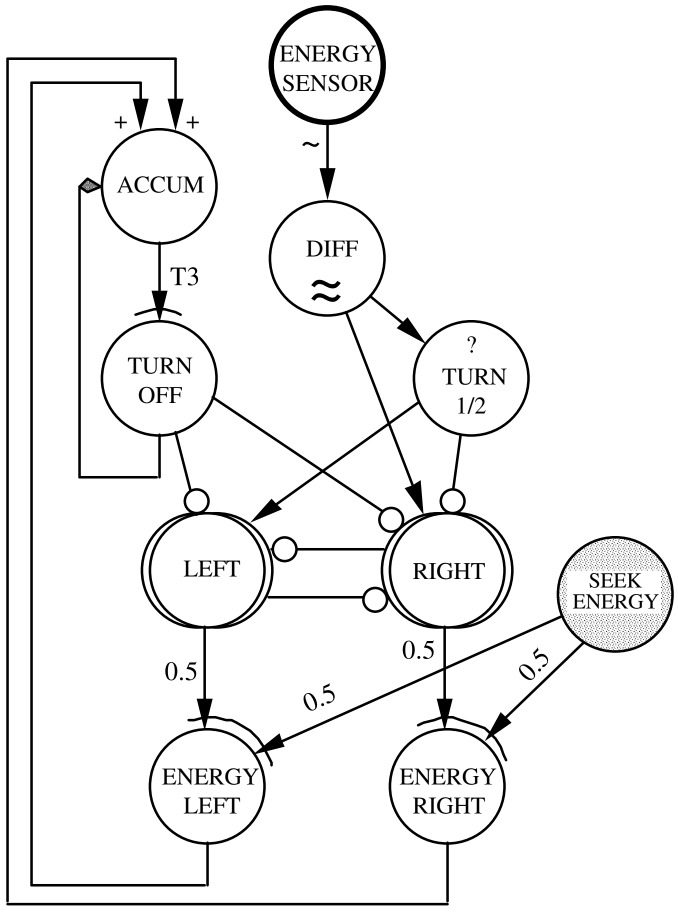


Figure 4.21 A neural circuit for the energy seeking behavior. The DIFF neuron provides the successive comparisons necessary for the energy seeking behavior.

The SEEK ENERGY neuron is used to enable the circuit. There is only one main energy sensor used as input to the circuit. The DIFF neuron is a differential neuron that compares the incoming sensor signal with the previous reading and emits a high binary signal whenever the new reading is lower than the last one. This single neuron allows the robot to detect when it is moving closer or further away from the energy source [26].

Once the DIFF neuron has detected that the robot has moved further away from the energy source, the random TURN neuron enables either the LEFT or RIGHT turning neurons. These two sustain neurons enable the ENERGY LEFT and ENERGY RIGHT neurons which are connected directly to the TURN LEFT and TURN RIGHT neurons of Figure 3.13 as in the other instincts. Each time the robot turns, the ACCUM neuron is excited. This neuron excites the TURN OFF threshold neuron with a weight of T3. T3 represents the number of 15 turns that the robot will make before it moves ahead. Thus if T3 = 0.2, the robot will make 1/0.2 = 5 turns of 15 for a total directional displacement of 75 . Once the TURN OFF neuron detects the completion of the turn, the LEFT and RIGHT neurons are disabled, allowing a new direction to be chosen. The effect of changing the value of T3 is shown in Figure 4.22.

The figure represents 4 screen snapshots showing the path that the robot traveled while using the energy seeking behavior. The rectangle in the center of the environment represents the energy source. Notice that with larger turning angles (i.e. low values of T3), the robot is able to remain near the source and occasionally travel over it. In the case of high value of T3, the robot does not come in contact with the energy source and eventually loses the sensor readings by traveling outside the intensity range of the source. The value of 0.20 was chosen for T3 since the path allows the robot to home in closer to the source while maintaining a degree of randomness.

As with the light sources, the effects of the energy sources must be adequately implemented. As with smell sensors, an energy sensor receives its readings depending mainly on the distance from the energy source, and again multiple energy sources must be taken into account.

[26] In practical applications, the robot may need to sample the environment less often since the readings may be identical until the robot becomes significantly closer of further from the source.

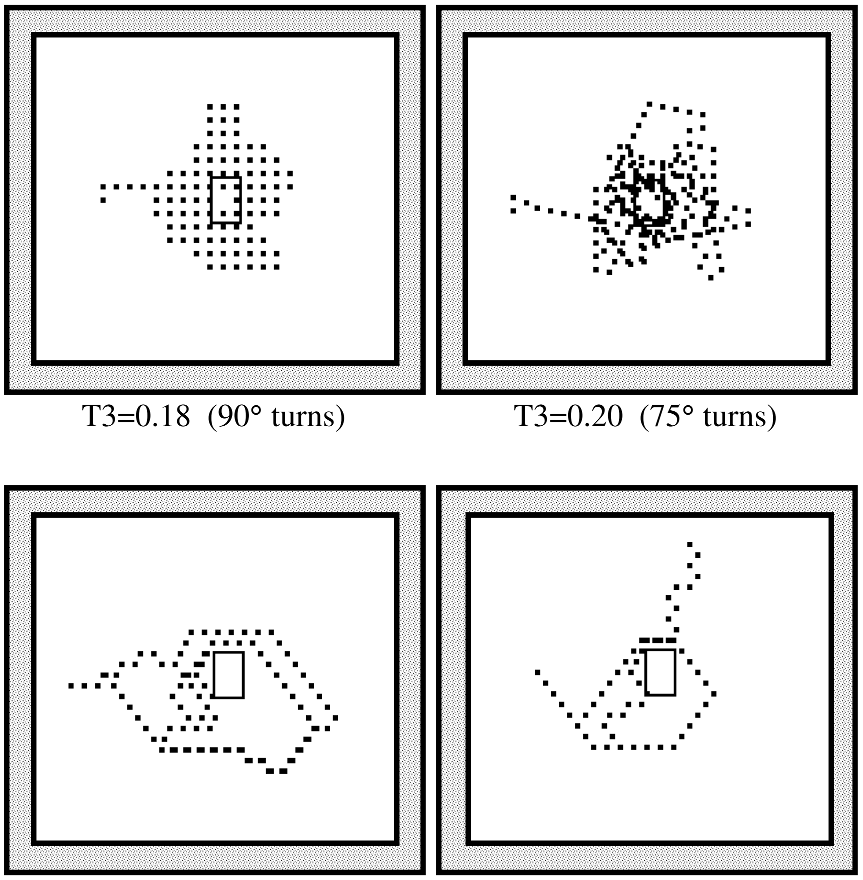


Figure 4.22 Screen snapshots showing the effect of varying T3 during the energy seeking process.

The method used to determine the sensor reading is as follows:

$$\frac{\mathbf{1}}{\overset{\mathbf{n}}{\underset{i=1}{\bullet}} \sqrt{\frac{\mathbf{1}}{\mathbf{d}_i}}} \qquad \text{where } 0 < \mathbf{d}_i < \mathbf{e}_i$$

and,

$\mathbf{n}$	is the number of energy sources
$\mathbf{d}_i$	is the distance from the sensor to energy source i.
$\mathbf{e}_i$	is the intensity factor of energy source i.

This equation shows that the sensor reading depends solely on the distance from the energy source. Thus by standing in one location, there is no indication as to the direction of the energy source. As in the light sources, the intensity **e** plays a role in the distance information.

Only energy sources that are within a specified distance are entered into the equation. This need to be close to the source is related to the intensity, which can be thought of as the degree of detectability of the energy field. In the simulation, this intensity is a constant. As a result, the robot can only detect the energy source when it is within a certain distance.

Once the robot finds the energy source it must have some mechanism to acquire the energy (i.e. recharge the batteries). In the case of a wall socket, the robot would need to accurately position itself in order to plug itself in. It is therefore helpful to add a simple sensor that can detect when the robot is in an appropriate position for plugging in. This is discussed further in chapter 7. RABI simplifies this process by assuming that the robot can gain energy whenever it lies on top of the simulated socket. This assumption would be useless in practical robot applications since the robot would need to be more accurate. One possible solution is to customize the sockets with some form of funneling system that would allow the robot to become aligned properly to the socket as it gets closer. A better approach may be to create some form of docking bay in which many of these robots could enter into a docking zone. Here, a different robot or machine could have a mechanism which would have the ability to track incoming robots and recharge them by plugging them in. These ideas are clearly hypothetical but may eventually become reality with colonies of simple microbots or nanobots.

4.6 Cleaning: A Task-Oriented Behavior

If a robot is to be useful, it must have some behaviors that allow it to perform a specific task. One simple task is that of cleaning. Consider a robot that is required to collect dirt by scooping up morsels and bringing the dirt to the edges of its environment. This task can be performed with a simple scooping mechanism requiring very simple sensors. The robot may have a simple scooping mechanism fastened below it. The scoop itself would hover close to the floor similar to a dust pan. As the robot travels in a dirty environment, the scoop becomes full with dirt morsels. A simple micro switch can be used to detect when the scoop contains a full load of dirt. The robot could then walk to the nearest edge and empty the scoop using a simple dumping mechanism.

This "scoop and dump" process represents a cleaning task that the robot could perform whenever it is able, that is, when it is not in need of energy. The simple neuron network in Figure 4.23 represents the cleaning behavior. The SCOOP FULL neuron is a sensor neuron that detects when the scoop is full of dirt. The neuron excites the CLEAN AHEAD neuron which causes the robot to walk straight forward towards an edge [27]. This CLEAN AHEAD neuron inhibits the TURN LEFT and TURN RIGHT neurons similar to the EDGE AHEAD neuron of the edge following behavior.

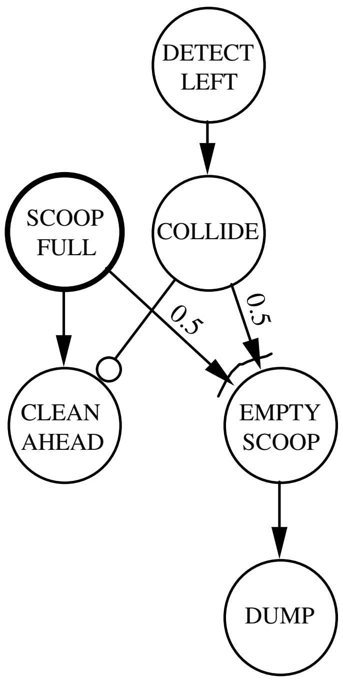


Figure 4.23 A neural circuit for the cleaning behavior.

The DETECT LEFT neuron is from the collision avoidance network. It excites the COLLIDE neuron whenever the robot' s frontal antennae touch an obstacle. When this happens, the CLEAN AHEAD neuron is disabled and the scoop is emptied. The DUMP motor neuron represents the dumping mechanism that is used to empty the scoop.

This method of cleaning is not the most effective since the robot has no sensors that allow it to seek out food morsels. Instead, the morsels of dirt are only detected when the scoop becomes full. As a consequence, there may be some areas in the environment that do not get cleaned. By wandering around, the robot is able to clean a significant portion of the environment. Figure 4.24 below represents screen snapshots of the simulated environment before and after the cleaning behavior was used for approximately 5 minutes. The wandering and vacancy behaviors were used in combination to allow the robot to obtain the morsels in a

[27] Walking straight ahead does not always find the closest edge, but it is the easiest one to find since the environment is closed off and the robot is sure to find an edge by walking straight.

somewhat random fashion. In the second snapshot, most of the morsels have been moved to the outer edges of the environment.

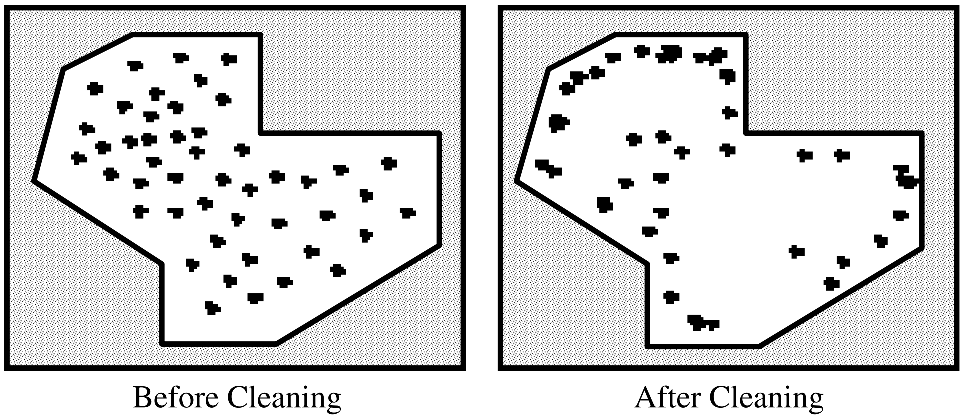


Figure 4.24 Screen snapshots before and after the cleaning behavior was used.

As seen in the snapshots, the cleaning behavior can be efficient when combined with other behaviors. This method of combining behaviors is described in the next section.

4.7 Behavior Selection

Although an animal may exhibit a variety of instinctive behaviors, sometimes the animal must choose between two or more conflicting behaviors. It is easier to simulate a behavior switching agent by limiting the agent to performing only one behavior at a time. In such an agent, there needs to be some mechanism and overall structure that provides a flexible and robust means of switching between behaviors. [Maes 91] and [Tyrrell and Mayhew 91] present methods of selecting behaviors according to internal factors as well as external stimuli. These methods allow the agent to switch between behaviors using motivational aspects based on its internal monitors. This will be discussed further in chapter 7.

4.7.1 Behavior Hierarchy

The subsumption architecture of [Brooks 86] provides a method of switching between behaviors in a layered fashion. The architecture provides a method of selecting a behavior which controls the overall performance of the system where the higher levels have a higher priority in that they may subsume the roles of the lower levels.

A similar architecture was incorporated into RABI through neural circuitry. Each of the instinctive behaviors are connected to the TURN LEFT and TURN RIGHT neurons which essentially control the actuators directly by way of the walking circuitry. The instinctive behaviors act as different levels of competence similar to the subsumption architecture. Moreover, these instincts can be added on in an incremental fashion. The remaining task of prioritizing these behaviors becomes simple with neural circuitry. The connecting architecture of the instinctive behaviors is shown in Figure 4.25.

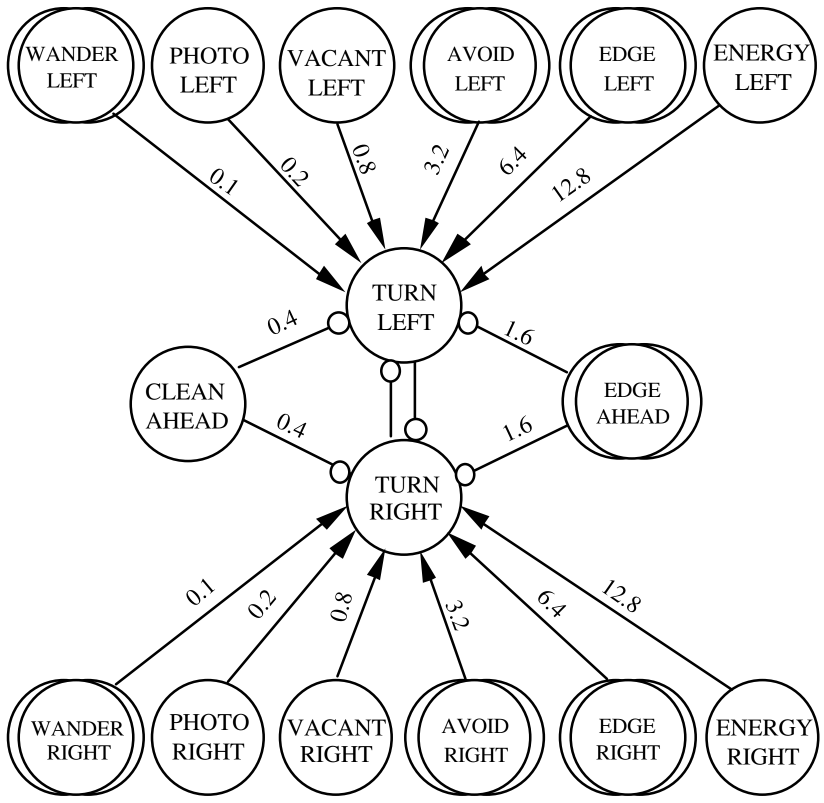


Figure 4.25 Prioritized connections for instinctive behavior selection. Each behavior competes for overall directional control.

In this figure, the TURN LEFT and TURN RIGHT neurons are the neurons from Figure 3.13 that provide the mechanism for steering the robot. These two neurons inhibit each other since they represent conflicting actions [28]. As a result, only one of these two neurons will be active at a time. If both are inactive, then the robot walks straight ahead.

The weights from the directional neurons were chosen in a priority oriented fashion. They begin at 0.1 and increase by factors of 2 (i.e. 0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.4 and 12.8). This factor allows each behavior to override the ones below it in the hierarchy. That is, the energy seeking behavior dominates all lower level behaviors since it has a weight which is higher than all others combined. In a sense, this allows the energy seeking behavior to subsume all of the lower level behaviors. Similarly, the collision avoidance behavior overrides the vacancy behavior, which both override the wandering behavior etc...

The EDGE AHEAD, EDGE RIGHT and EDGE LEFT neurons all represent the edge following behavior, however, the EDGE AHEAD neuron has a smaller weight than the other two. This lower weight was selected such that the collision avoidance behavior could subsume it whenever the robot needed to squeeze into a narrow passageway. Also, the EDGE AHEAD neuron provides an inhibiting link as opposed to an excitatory link as with the others, allowing it to disable the left and right directional changes resulting in straight forward motion. The CLEAN AHEAD neuron is connected in a similar manner.

4.7.2 Emergent Behaviors

The simple hierarchical structure allows different behaviors to be added or removed without altering the others. By combining various behaviors through their connections to the TURN LEFT and TURN RIGHT neurons as in Figure 4.25, a variety of more complex behaviors can emerge. Take for example, the wander and vacancy behaviors. Figure 4.26 shows a screen snapshot depicting the effects of combining the two behaviors during a 1 minute time period.

[28] Actually, in software, these two inhibiting links were not implemented. Instead, both neurons receive inhibiting signals from the directional neurons on the opposite side such that it is disabled whenever the opposite side has dominating signals.

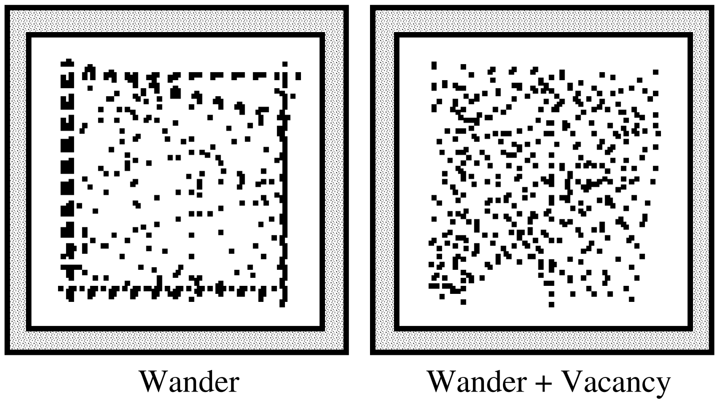
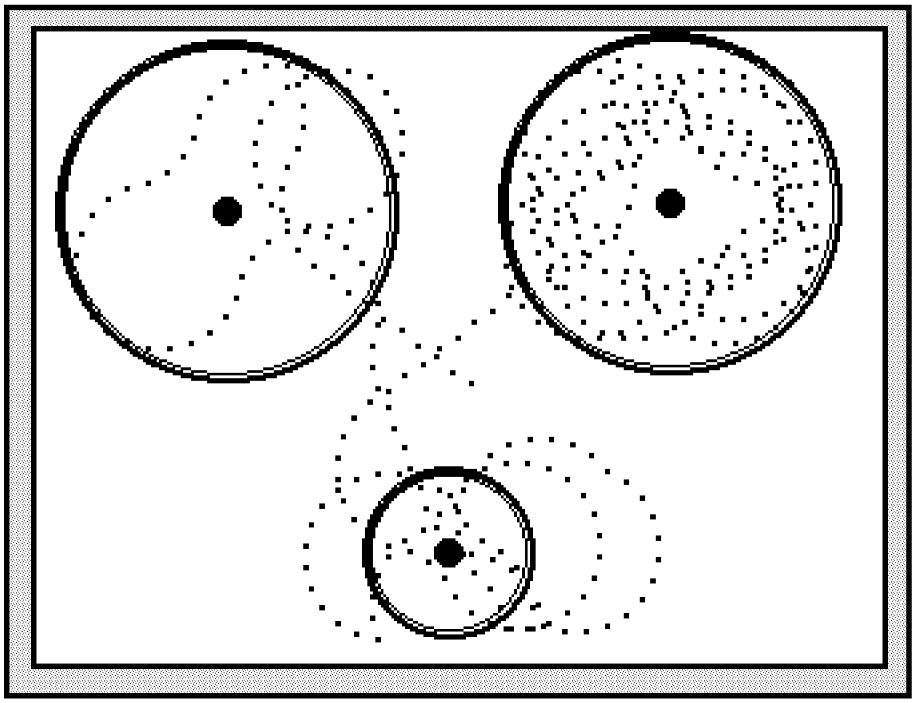


Figure 4.26 Screen snapshots depicting the effects of combining the wandering and vacancy behaviors.

Both situations are using the collision avoidance instinct to stay within the environment. As the images show, using only the wandering behavior, the robot spends much of its time along the environmental boundaries. When the vacancy behavior is added, it acts as a recoil which keeps the robot away from the walls. Clearly, this addition allows the robot to spend most of its time away from the borders. Neither of these behaviors direct the robot to any particular location, instead the resulting trajectory is random and has no purpose except exploration. Now consider the effects of combining the wandering and light seeking behaviors as in Figure 4.27.

Both cases were allowed to run for 2 minutes. In (a) the robot started near the top left light source, making a few circles around its center. It then ventured down to the smaller light source for a few laps before proceeding to its final destination, the top right light. The robot remained at the top light source since the light seeking behavior had ample sensor information to keep the robot moving in circles. With the addition of the wandering behavior in (b), there is clearly a difference in behavior. The wandering behavior allowed random movements causing the robot to stray from its otherwise predictable circulating path. As a consequence, the robot spent much of its time traveling between light sources. Less time was spent near the smaller light source since its low intensity did not attract as much attention as the higher intensity sources.


(a)

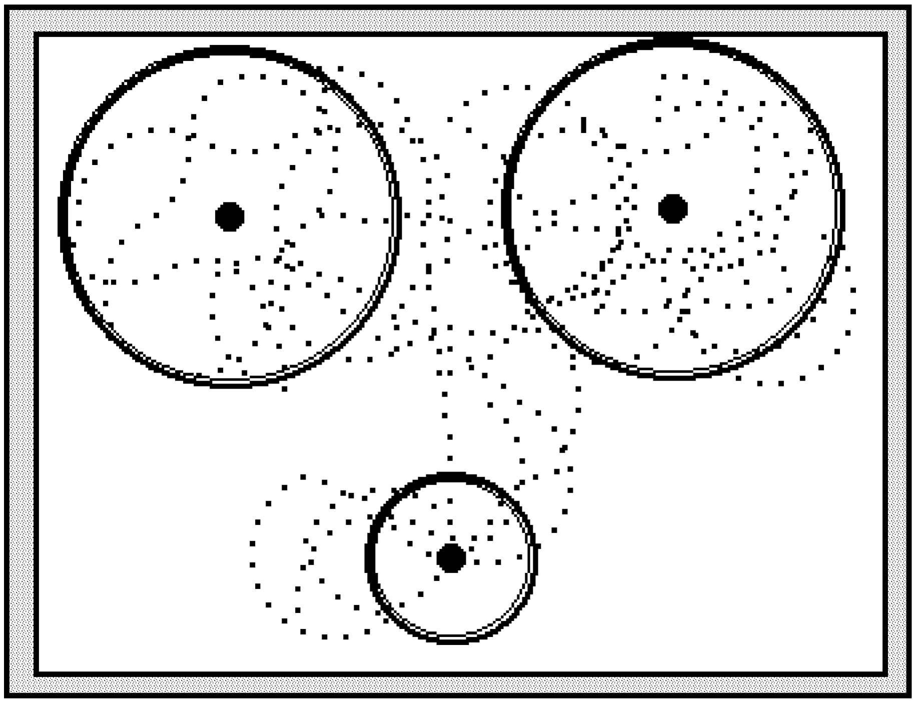

(b)

Figure 4.27 Screen snapshots showing the wander and light seeking behaviors. (a) light seeking only, and (b) wandering + light seeking.

A last example of emergent behavior is shown in Figure 4.28. Here, the effects of the wander, vacancy and light seeking behaviors are shown for a simple environment with an external light source. The environment represents a situation similar to that of a fly trapped in a box with light coming in from one wall of the box.

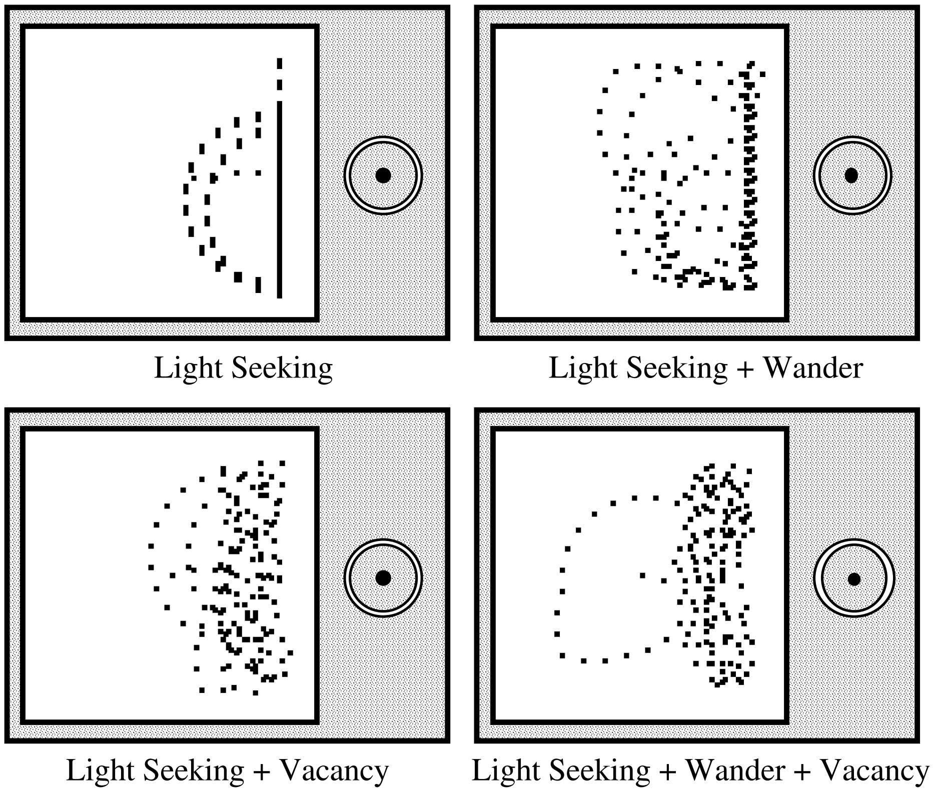


Figure 4.28 Screen snapshots showing the integration of the wander, vacancy and light seeking behaviors.

In the first snapshot, only the light seeking behavior is used. Here, the robot often presses up against the glass attempting to get to the light source. The robot' s path is very regular and repetitive causing the robot to rub up along the wall. In the next snapshot, the wander behavior is added. This addition results in a behavior that allows the robot to vary its path. The robot still presses up against the wall but not as often.

By using a vacancy behavior instead of the wandering behavior, as in the third snapshot, the robot no longer presses up against the wall. Instead, each time it hits the wall, it is repelled away from the wall. A close look at the image shows that the circular patterns are still visible. Finally, in the last snapshot, all three behaviors are used. The resulting emergent behavior appears random. Here, no circular paths can be seen and no pressing up against the wall is observed. At one instant, the randomness even allowed the robot to stray away from the source altogether.

4.7.3 Making Behaviors More Efficient

It is possible to increase the efficiency of individual behaviors by combining them with other behaviors. Some behaviors work well with others. Take for example, the cleaning task behavior. When just wandering around aimlessly, the robot may not be very efficient at its cleaning task since it stumbles upon dirt morsels by chance. A more efficient cleaning behavior could make use of the phototropic behavior. If a light source is placed at the dirtiest parts of the environment, by using the phototropic behavior, the robot will seek out the light source, hence spending most of its time near the light. By staying near the light, the robot is more likely to find dirt morsels resulting in a more efficient cleanup. By varying the location of the light source, the robot could clean up various portions of the environment one at a time. Figure 4.29 shows two screen snapshots of the cleaning behavior combined with a light seeking behavior before and after the test.

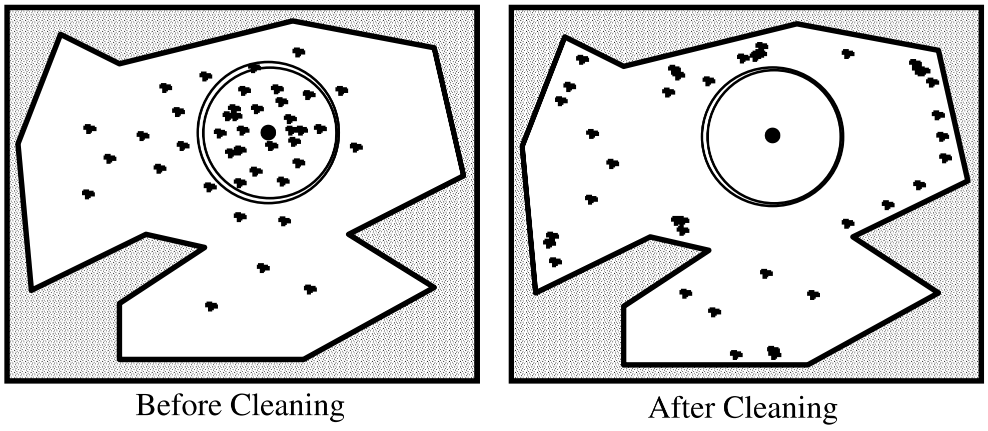


Figure 4.29 Screen snapshots showing the effects of combining the cleaning and light seeking behaviors.

Notice that the cleanup concentrated on the area surrounding the light source. In fact, the 3 leftmost morsels and the 3 bottom-most morsels were not touched since the robot did not venture into these areas.

Another less effective method of improving the cleaning efficiency is to make use of the vacancy behavior. By adding the vacancy behavior, the robot will spend most of its time away from the environmental borders allowing a quicker cleanup by remaining in the middle of the environment where the morsels are.

4.8 Summary

Providing a robot with basic underlying mechanisms corresponding to simple instinct-like behaviors provides a flexible and robust method of control. The instincts of collision avoidance, wandering, vacancy and edge following allow the robot to function safely and explore the environment in a flexible fashion. The light and energy seeking behaviors complete the system by providing most important survival behavior, which is that of being able to find and obtain food in the form of energy. A cleaning behavior is easily added, giving the robot a sense of usefulness.

Each of these behaviors are implemented as neural circuits allowing them to be easily integrated together. The behaviors interact through a simple subsumption style architecture also implemented with neural circuitry. This architecture allows the various behaviors to be added and removed without affecting the operation of the others. Moreover, by adding behaviors in combination, more complicated behaviors can emerge resulting in a more flexible and efficient system capable of a variety of otherwise unpredictable behaviors. Moreover, by using various combinations of behaviors, the robot can become more efficient.

~~~~~~

Chapter 5
Mapping out the Environment

~~~~~~

Exploration refers to the active behavioral processes by which an animal assimilates information about its environment. Animals often closely investigate novel objects in their environment and in some cases, they regularly patrol their environment and pay particular attention to any changes that have occurred in it. Such patrolling allows the animals to detect new sources of food [Toates 86].

In order to detect changes, the animal must have some idea regarding the state of the environment ahead of time. Usually, the state of the environment corresponds to some sort of mapping; which may be precise or very generalized (landmarks). This internal map is built up through exploration and stored in some form of memory. As mentioned previously, it is not at all clear how memory contents are stored or how this information is retrieved. Thus, most robots use some form of simplified 2D mapping techniques. To my knowledge, there has not been much research in the area of 3D mapping applied to robotics. This extra dimension presents a host of new problems and is not touched upon by this thesis.

5.1 Mapping Strategies

In order for a robot to map out an environment, it must have some sense of relative distance. The robot, for example, must be able to estimate the distances between two locations so that it may store this information in its map. This is essential so that the robot would know which locations it may travel in and which locations are not traversable.

There are two approaches to mapping out the traversable areas: that of mapping the *free space* (open areas) and that of mapping out the *obstacle space*. These are discussed in turn.

5.1.1 Mapping Free Space

Many existing robots map their environment by determining the locations that the robot can reach. The easiest method of mapping this way is to create a 2D grid, where each grid unit represents either a free location or an occupied location (obstacle). In a maze, for example, a "micro mouse" robot would map out the maze as a set of consecutive free units. The micro mouse does not need to know the locations of walls, instead it needs to know the unoccupied locations in which it may travel.

Using a simple 2D grid and marking locations as occupied or unoccupied, may not be very efficient in some situations. For an environment with very few obstacles, there would be large amounts of free space. A better method of grid mapping is that of using quadtrees [29]. This method combines similar adjacent units in groups of 4 into a larger unit of size 1. The result is a more efficient grid which requires less memory since the number of grid units is decreased. Figure 5.1 shows an example of an environment mapped with both the standard grid technique and the quad tree technique.

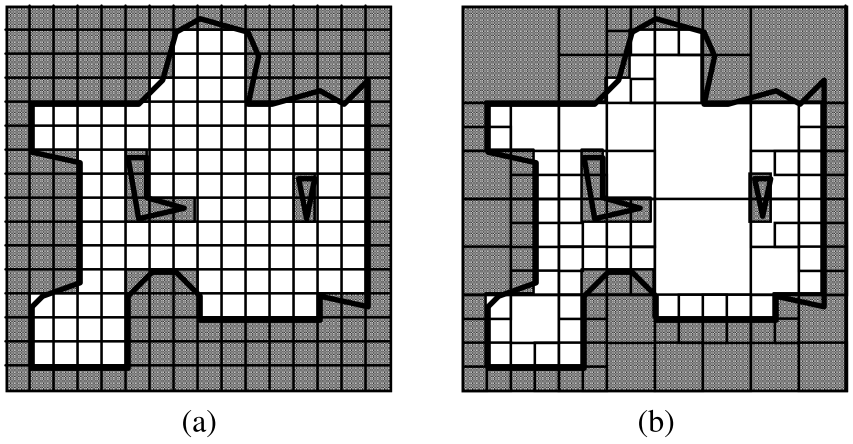


Figure 5.1 Two variations of grid-based mapping. (a) a straight forward grid of equal sized units, and (b) a more efficient grid using a quadtree structure with units of various sizes.

The thick lines represent the borders of the obstacles and the boundary. The standard grid requires 16x16 = 256 units to store the map while the quadtree requires only 129 units (approximately half). It is clear that in this situation, the quadtree mapping is more efficient.

[29] [Zelinsky 92] presents a method of building quadtree maps by exploring the environment. His method also incorporates a shortest path strategy.

The efficiency of the quadtree method will vary depending on the density and complexity of the environment. Crowded environments for example, would not contain large units since for any location there is likely to be locations of different types nearby. Complicated environments such as mazes, also cannot contain large units since there are many thin closely spaced obstacles which prevent integration of smaller units. Thus, for some environments, the choice of a quadtree over a standard grid may not provide an advantage and may even provide a disadvantage due to overhead.

The quadtree grid embeds additional information. Since similar units were grouped together, then by storing the size of the new unit, one can obtain information about the relative "openness" of the environment. That is, the robot can identify large open spaces [30].

The quadtree representation could be improved upon even further by allowing an offset for each unit. The quadtree representation uses a symmetric dissection of the environment into grid units. By allowing a different dissection, the number of units can be further decreased. For example, the tunnel of small units on the left of Figure 5.1(b) contains 3 groups of 4 adjacent units. These could be combined, reducing the whole tunnel to just 3 units. Figure 5.2 (a) shows the results of using this offset dissection method. Notice that the squares do not always line up together. This further reduces the bits needed to 109, which is only 20 units less than the normal dissection method.

The quadtree representation provides further advantages during navigation since a dual tree can be constructed in which each node represents a grid unit. Here, the resulting dual tree would represent all possible paths within the free space. Figure 5.2 (b) shows the dual tree of the offset dissection quadtree in (a).

As far as implementation is concerned, the standard grid based approach to mapping is simple to implement since all the units are of the same size, the robot merely needs to fill in the bits. The quadtree approach is not as easy to implement. It requires an examination of the units and grouping them together.

Despite their simplicity, all of these grid-based approaches suffer from two main problems. First, an approximate grid size must be known. In some cases, knowledge of the starting point (unit) is also required. As a result, the grid-based approach is mostly useful in

[30] Large open areas may represent the different rooms in an indoor environment.

situations where there is some prior knowledge as to the structure of the environment. In cases where the world size is unknown, some sort of map growing and appending technique must be used.

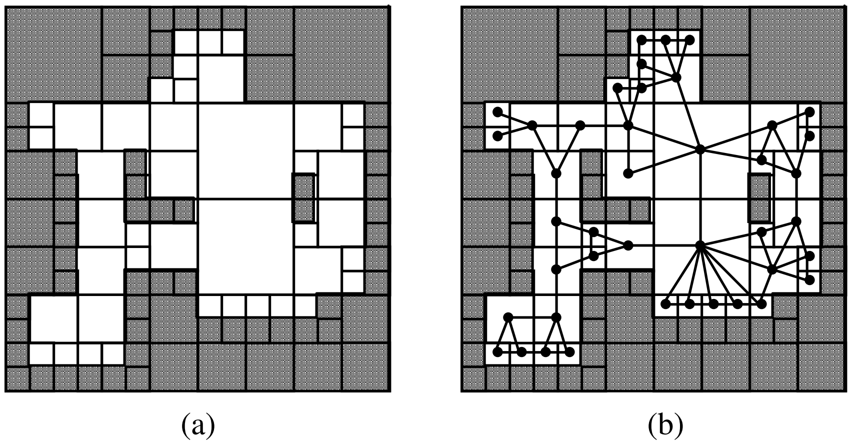


Figure 5.2 The offset dissection of a quadtree mapping. (a) the revised quadtree combining additional units, and (b) its corresponding dual tree.

By far the biggest problem of the grid based approach is that of positioning errors. As mentioned in [Brooks 91], absolute coordinate systems for a robot are the source of large cumulative errors. The robot must be able to acknowledge the fact that it has left one grid unit and entered another. Every physical system has some degree of inaccuracy, often preventing precise position measurements to be made. To account for errors, the robot must be able to "get back on track" once it becomes lost or when its position uncertainty becomes too large. This position error is a problem faced by all mapping techniques, however, it is particularly troublesome with the grid techniques since the entire map is based on evenly spaced units.

5.1.2 Mapping the Obstacle Space

There are some animals and insects that live under rocks, in dark areas, in small holes, etc. These creatures spend most of their time close to and inside objects. It is rare that these insects would wander freely in open areas unless they are exploring the environment or en route to another location. This is somewhat of a survival instinct for insects since they tend to get squished if seen wandering in an open area. The obstacles provide protection and shelter for the insects, and possibly even sources of food.

For animals such as insects, it makes little sense to map out the free regions of the environment. Instead, mapping out the obstacles would prove to be more helpful since this is where the insect regularly travels. By identifying the obstacles and the relative distances and directions between them, the insect can survive by limiting the amount of time spent wandering in open areas. [Staddon 83] identifies some organisms that exhibit this free space avoidance behavior. Perhaps these organisms tend to stay near obstacles because they receive rich sensor information from them. In open space, there is a lot less information about the surroundings. For insects, their antennae are used to provide a wealth of information about obstacle sizes and shapes but they cannot present much information from a flat and open surface.

To map out obstacles, there must be some way of distinguishing between the different obstacles in the environment. Many robots use ultrasonic transducers, cameras and laser range finders to detect obstacles and map out the environment. These sensors provide distance information in the form of an environmental scan, which results in a set of points. Operations are performed with these points (such as averaging and extrapolation) and line segments are computed. Thus, the resulting map is a set of line segment chains. By traveling around the obstacles, these line segment chains could then be joined, creating a set of polygons representing the obstacles as shown in Figure 5.3.

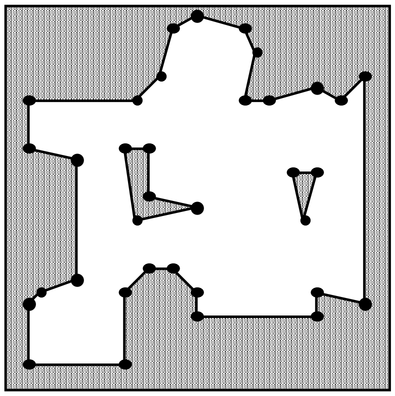


Figure 5.3 A polygonal mapping of a simple environment.

Although this method of mapping can produce a fairly accurate representation of an environment, it requires accurate odometry on behalf of the robot and significant noise reduction techniques to be performed with the sensor data. This technique would require much computational power as well as sophisticated sensors; both of which may not be possible on smaller sized robots. For small robots, a mapping technique must be developed which uses simple sensors, and requires little computational burden. Landmark detection and identification is one possible solution for mapping with simple robots.

5. 2 Landmark-Based Mapping

Bees are known to navigate by using landmarks and image patterns. The bee uses these landmarks as a reference for orientation and position estimation. A bee stores its landmarks as low resolution images with space set aside for information on each landmark or image. For example, when identifying flowers, there is space set aside for the color, odor, shape etc. [Gould and Marker 87] discuss the instinctive image detection scheme of honey bees. Landmark detection and mapping are commonly integrated as a single behavior; once a landmark is detected through feature extraction, it is stored in memory. After all, the only reason for detecting landmarks is so that they may be stored for future reference. This method of mapping requires a mechanism for extracting various features from landmarks and storing the information.

5.2.1 Previous Approaches

[Nehmzow and Smithers 91] present a method of mapping out the inside perimeter of a rectilinear environment using a real robot. The robot is able to identify its location in the environment by recognizing sequences of convex and concave corners. The sequences of corners act as landmarks. Their method incorporates a self-organizing neural network which stores the corner features within it. The input vector of the network contains information regarding the present corner, previous corners as well as the distance traveled from the previous corner. After traversing the perimeter a couple of times, the robot is able to identify some of the corners (landmarks). A drawback of using the self-organizing neural network is that corners can only be identified if they are significantly different from other corners with respect to the corners around it. That is, a completely symmetric environment would not allow any of the corners to be distinguished from the others. In Figure 5.4, environments A and B are examples of symmetric environments that have similar corners. No corner can be distinguished. In C and D however, the corners can be distinguished since they all differ in terms of distances between consecutive adjacent corners. Their method was designed to map out a simple perimeter and there is no mention of mapping out inner obstacles. Moreover, by using their neural network, they cannot directly extract corner and edge information since the network is self-organizing. The technique does, however, allow the robot to map out an obstacle perimeter using simple sensors.

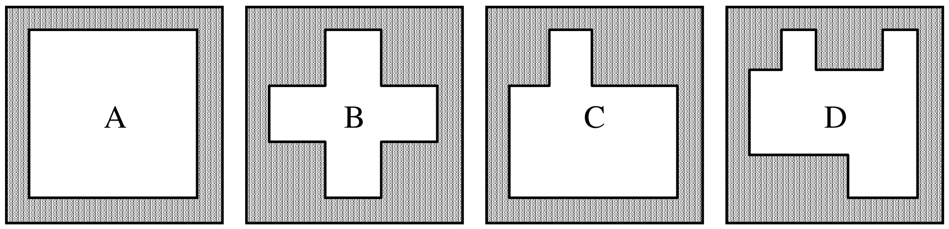


Figure 5.4 Identifiable and unidentifiable environment shapes. A and B are symmetric environments in which corners can' t be distinguished. Non-symmetric environments C and D allow corners to be identified.

[Mataric 91] presents a different approach to landmark-based mapping. His method was tested on a real robot equipped with sonar sensors and a compass. The system is able to identify various landmarks in the environment such as walls, corridors and long irregular boundaries. Moreover, each of these landmarks have an attribute indicating its orientation in the environment. His technique was tested in a cluttered environment but did not have the ability to map out inner obstacles. Instead the method maps out the boundaries specifying the areas of the room that are blocked off by obstacles. The mapping strategy used here was that of topological links between landmarks indicating their physical spatial adjacency. In essence, the map is similar to a graph whose nodes are landmarks and whose links specify spatial adjacency in the environment. The use of range sensors (sonar) and orientation mechanisms (compass) makes this method less attractive for the purposes of nano technology. The research does however, present the interesting notion of topological landmark-based mappings.

5.2.2 RABI's Landmark-Based Mapping

RABI uses a similar approach to the techniques mentioned above. The robot records corners and edge lengths as [Nehmzow and Smithers 91] have done, however, the landmarks are stored in a memory similar to that of [Mataric 91].

5.2.2.1 Landmark Feature Identification

Biological neural networks are able to quickly and efficiently learn input patterns by extracting features from the sensory input [31]. These networks are able to extract and analyze features from their input and then efficiently organize the patterns in memory [32]. It would be useful to endow a robot with some kind of feature extraction technique such that it could determine the different features from the obstacles within its environment. The difference in features would allow one obstacle to be distinguished from another. With this ability, a robot could map out its environment by identifying *landmarks* . Bees are known to employ such a technique [Gallistel 90].

Insects have the simplest form of obstacle sensor: the "antenna". Due to the varied locations of antennae, whiskers and hair, obstacles can be detected all around the body. If the antennae are moveable, as in most insects, then features from the obstacle itself can be identified such as height, width, motion, etc. This simple sensor can be used to detect features of large obstacles through continuous displacement along the boundary of the obstacle (i.e. edge following). This is what is known as "active" touch, as opposed to stationary detection which is "passive". [Gibson 62] gave quantitative results showing that active touch is superior to passive. In his experiments, shapes were to be identified by a human subject using only the sense of touch. The passive touch tests involved pressing the shape into the palm of the hand. The active touch tests allowed exploration by the fingertips. The results showed that passive touch received only half as many correct matches as active.

[Hochberg 68] proved, at least for vision, that the serial presentation of sequential views of a shape was sufficient for its identification. He performed experiments in which a human subject was instructed to identify objects while looking through a small hole such that only a small piece of the object, such as a corner, was visible at any one time. The object was the rotated so that the subject viewed the corners in sequence. Consider the following scenario:

> "If you were to take your little brother, tie his hands and blindfold him, move him to a different location in his home and wake him up, he would probably be annoyed. If you then spin him around and tell him to find his way back to bed, he would probably walk until he hits the nearest obstacle. More than likely, he would then feel his way along walls and obstacles until some of these obstacles and structures are recognized. Once an object is identified, he would then be aware of his location and would then have little difficulty finding his way back to his bedroom."

---

[31] In visual systems, there exists feature analyzing components that detect corners, edges, curves, motion, contrast, etc.

[32] [Linsker 88] discusses the concept of feature-analyzing cells in self-organizing neural networks.

This scenario points out the technique that is commonly used when sensor information is reduced. Obviously, a person would make use of their sophisticated sensors such as sound, smell, temperature and feet sensors (detect floor, rug, tile). However, the person as well as simple robots must rely on the structural information gathered from the surroundings in order to identify the exact location in the environment. By detecting and recognizing obstacles, structures and the relative distances between them, a robot would have an adequate representation of the surroundings.

RABI uses a feature recognition method derived from the results of [Gibson 62] and [Hochberg 68]. The robot is able to identify an obstacle by traversing its boundaries, identifying the corner angles and edge lengths in sequential order, and then matching it with existing landmarks in memory. The edge following behavior is used to follow along the perimeter of an obstacle, and a separate neural circuit is used to identify the features of the obstacle in terms of the lengths of each edge and the angle between consecutive edges. Storing only corners and edges is sufficient for mapping a 2D environment provided that ample detail is extracted. Each corner must be identified by both an angle (magnitude) and an orientation (concave or convex). For example, figure 5.5 shows three environments with identical edge lengths and corner magnitudes, but some of the corners are oriented differently. Clearly, A and B should be distinguishable, resulting in the need to store corner orientation. For A and C however, it is not as clear.

The sequential features of A and C are identical in every way. The only difference is their global orientation with respect to the outside world. If A and C are closed off from the outside world, then they are essentially identical. If however, there exists stimuli outside of the environment such as lighting, noises, etc., then these two environments may need to be differentiated. More on this in chapter 6.

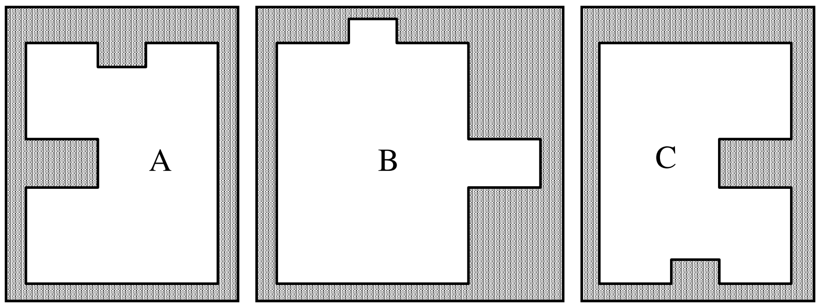


Figure 5.5 Three environments with identical edge lengths and corner magnitudes.

5.2.2.2 Odometry and Angle Measurements

The measurement of angles and edge lengths requires odometry. Since RABI uses only a tripod gait for walking, the robot either moves ahead one unit or turns one unit [33]. A simple neural circuit allows the robot' s movements to be rounded off into time units. This circuit is shown in Figure 5.6.

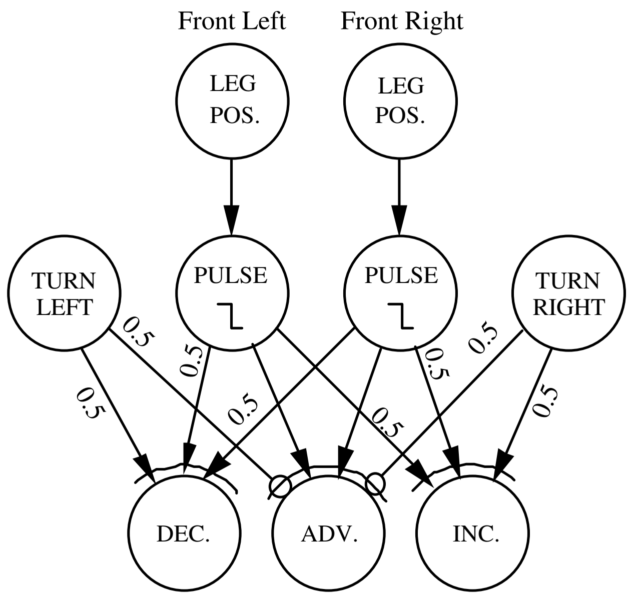


Figure 5.6 Position measurement circuitry. The DEC and INC neurons indicate a left and right angular unit change. The ADV neuron indicates a position change of 1 unit forward.

Since only a tripod gait is used, then the robot will move forward once during the stance phase of each front leg, thus twice per walking cycle. The LEG POS neurons indicate a stance phase. The pulse neurons make sure that the positioning neurons DEC, INC and ADV, only receive a count once per stance phase. The TURN RIGHT and TURN LEFT neurons are the directional control neurons of Figure 3.13.

The legs are disabled in the software version of RABI, in order to speed up the simulation. One time unit in the simulation corresponds to the updating of each neuron network exactly once. It is assumed that in each of these time units, the robot walked either forward, left or right. Thus, the circuit of Figure 5.6 was reduced to Figure 5.7 for the software version.

[33] One unit is approximately 2 inches and occurs twice per walking cycle. When turning, one unit is approximately 15 degrees.

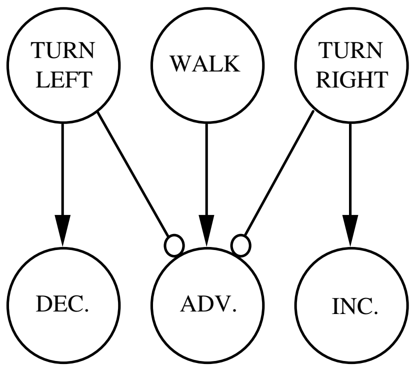


Figure 5.7 The position measurement circuitry for the software version of RABI.

With both positioning circuits, the robot is able to measure angles and edge lengths by counting the number of times that the DEC, INC and ADV neurons are excited. For example, if the robot turned left 6 times sequentially then the DEC neuron would be excited 6 times indicating a 6 x 15 = 90 angle.

5.2.2.3 Feature Extraction

When following an obstacle, the lengths of the edges can be determined by counting the number of stepping units (via the ADV neuron) from endpoint to endpoint of the edge. The robot needs to determine where an edge ends and where the next edge begins. This determination can be made when the robot enters the re-align mode during its edge following. Due to the simplicity of the antennae, the robot may occasionally need to turn slightly towards or away from the edge it is following in order to remain relatively parallel to it. In this case, the re-align mode is also used to keep the robot parallel to the edge and thus the re-aligning mode may not always indicate an edge endpoint. If however, the robot makes a significant turn to regain contact during re-aligning then, more than likely, this represents an edge endpoint or convex corner of the obstacle.

When misaligned, the robot requires only a slight turn to regain its parallelism with the edge. By using some form of threshold when counting turns, the robot could distinguish between slight misalignment adjustments and corners. For example, if this turn threshold is set at 3 units, then turning 1 or 2 units during re-aligning would represent an attempt to align to the edge whereas turning 3 units would indicate a change in edges (i.e. a convex corner on the obstacle boundary). Similarly, concave corners are distinguished from misalignment with a threshold. Concave and convex corners are identified by negative and positive angles respectively.

Figure 5.8 shows the basic neuron network for obstacle feature extraction. This network interprets positioning and sensor information regarding the obstacle' s edge lengths and angles. The INC, DEC and ADV neurons are the positioning neurons from Figure 5.7. The ANGLE and DISTANCE neurons are accumulative neurons that count the number of consecutive turns and advances respectively. The DEC and INC neurons decrease and increase the total angle sum respectively. At sharp corners, the robot may alternate between turning and advancing since it is unable to turn and advance simultaneously. Thus, the robot must distinguish its forward advances along an edge and its forward advances while turning a corner. The ACCUM EDGE neuron counts the number of forward advances since the last turn was made. Once this accumulation reaches a threshold of T1, it represents the fact that the robot is following along an edge; as opposed to turning a corner. This threshold, T1 represents the minimum number of forward units to be advanced in order for the robot to consider it an edge. In essence, T1 represents the minimum recognizable length of an edge. A similar threshold is used for corner detection. Since occasional slight turns may be the result of a misalignment problem, the number of consecutive turns must have a threshold so that the slight directional changes can be distinguished from the larger turns at corners. T2 is the minimum number of turns required in order to be considered a corner. That is, T2 is the minimum detectable angle.

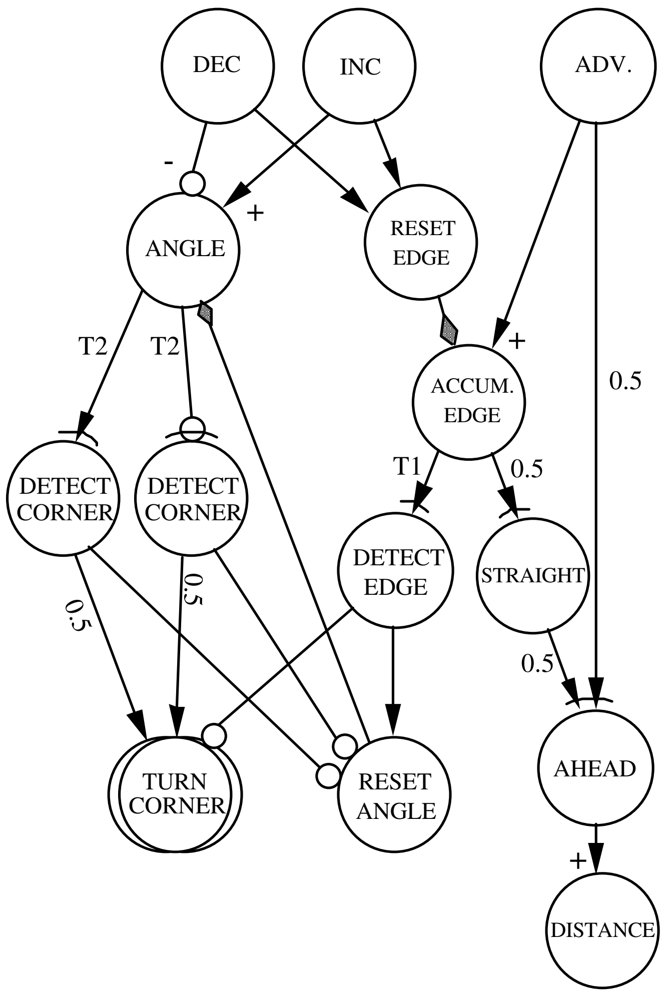


Figure 5.8 The neural circuit for feature extraction.

By decreasing the value of the cornering threshold T2, the feature detection circuit becomes less sensitive, resulting in fewer detectable corners. Similarly, if the threshold is kept high, the circuit is able to detect every corner with greater precision. Figure 5.9 shows the results of varying the cornering threshold. The map of an environmental border is shown for corner threshold values of 1.0, 0.5, 0.34 and 0.25. The darkened lines represent detected edges and the white circles represent detected corners. With these threshold values, the robot can detect corners of 1, 2, 3 and 4 angular units, which is approximately 15 , 30 , 45 and 60 angles. Notice that with a threshold value of 1.0, every turn is detected, even turns that were issued along an edge due to misalignment. Moreover, the larger-angled corners are detected as many small corners. Clearly, this value is too precise. At the other end, a value of 0.25 can only detect angles of 60 or more. This value omits many of the corners, resulting in a very general and imprecise mapping representation. Values of 0.34 and 0.5 are intermediate choices that provide a reasonable representation closely resembling the actual environmental structure. RABI uses a value of 0.34, allowing the detection of 45   angles.

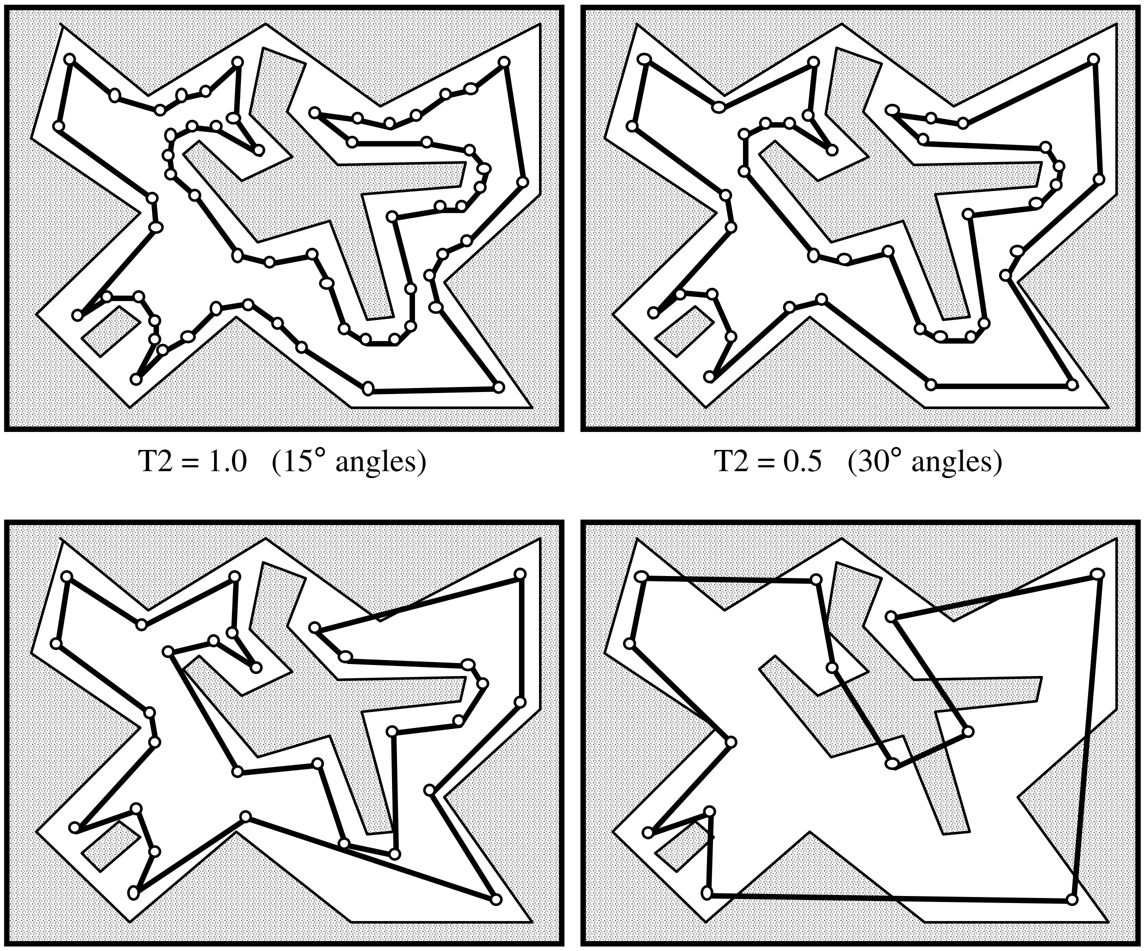


Figure 5.9   The effects of varying the cornering threshold.

A similar effect is observed by varying the edge threshold value of T1. By changing the edge threshold, the minimum detectable edge length can be set. Thus, with a high threshold value (1.0), small edges of 1 unit length [34] can be detected. With a low threshold value, only larger edge sizes can be detected. The result of varying this threshold is similar to that of varying the cornering threshold in that the smaller edges are combined to produce more general representations. RABI uses a value of 0.2 so that the robot must move straight along the border for approximately two body lengths in order for an edge to be detected. Perhaps, it would be better to alter these threshold values over time. This would allow the robot to create specific

[34] One unit length is approximately 5 pixels in the simulation. This is about half the robot body length.

mappings for new or dense regions and use lower resolution mappings for the familiar or less dense areas. This adaptivity in resolution was not incorporated into RABI but is an interesting topic for future research.

Note that the DEC and INC neurons increase and decrease the accumulated energy of the ANGLE neuron. Consequently, a left turn will cancel out a right turn and vice versa in the overall angular sum. When turning corners, however, one direction is dominant over the others and the sign of this angle will prevail. Each DETECT CORNER neuron detects a corner as a dominant positive or negative angle. Whenever an corner is detected, the TURN CORNER neuron is excited indicating that the robot is turning a corner. The robot remains in this cornering mode until a significant number of forward advances is detected. That is, the DETECT EDGE disables the cornering process whenever the corner has been turned and the robot begins to follow a new edge.

The STRAIGHT and AHEAD neurons are used to ensure that the robot moves at least two forward units before it begins adding the forward units to the distance sum. This ensures that alternating turn and forward motions (as seen when turning a sharp corner) does not register as part of the edge length. In a sense, it reduces the error on the
measurement of the edge lengths.

While following an edge, the robot may often become misaligned, resulting in many small direction adjustments which may build up energy in the ANGLE accumulative neuron. Whenever an edge is detected, the accumulated energy of the ANGLE neuron is reset by the RESET ANGLE neuron. This prevents the minute direction adjustments from interfering with the angular data at the next detected corner.

Sometimes, once a corner has been turned, there may be another turn in the opposite direction (zig-zag shape) as shown in Figure 5.10. Since corners are only stored when the robot begins following a new edge, then the corner C_1 would not be stored because it is not followed by an edge. Instead, C_2 will undo the angle made by C_1, and when E_2 is reached, the angle will be 0.

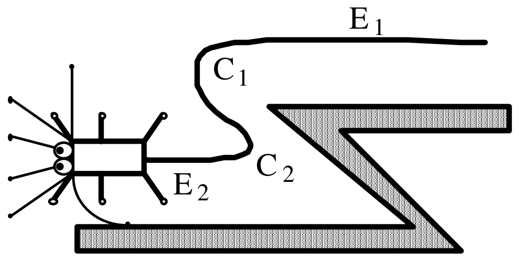


Figure 5.10 Zig-zag path with consecutive corners C_1 and C_2 of opposite types.

A mechanism is needed to detect consecutive corners of opposite direction so that each corner could be stored. This mechanism is realized by adding more neurons and connections to Figure 5.8 as shown in Figure 5.11. The TURNING LEFT and TURNING RIGHT neurons are sustain neurons that indicate if the robot is turning a left or right corner. The CHANGE RIGHT and CHANGE LEFT are used to detect when the robot turns from left to right or right to left without advancing forward. When a change from one direction to another is detected, one of these neurons will be excited and inhibit the TURN CORNER neuron. Consequently, if there is a significant angle built up, then the corner will be stored.

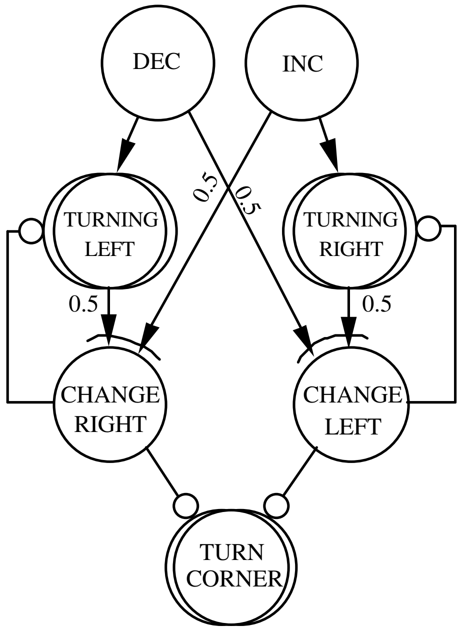


Figure 5.11 Additional neurons in the feature identification network.

5.2.2.4 Landmark Memory

RABI' s memory is arranged in a linear fashion, essentially one long list of memory neurons. Each neuron can store a value [35] representing a corner angle or edge length. Moreover, adjacent memory neurons alternate between *corner neurons* and *edge neurons*. Links join the neurons that represent adjacent features of a landmark. Figure 5.12 depicts a simple landmark mapping. The values of the neurons correspond to the corner angles (15 units) and the edge lengths (forward units). The bottom link connects the first and last neurons, due to the closed nature of perimeters.

In the general case where an environment is cluttered with obstacles, there is a need to separate the landmarks in memory. The simplest way to do this is to fill up the memory in a linear fashion keeping pointers to each separate obstacle. Figure 5.13 depicts the memory contents after 3 landmarks have been identified. Note that one of these landmarks may represent the environmental boundary but it is not distinguished from the others.

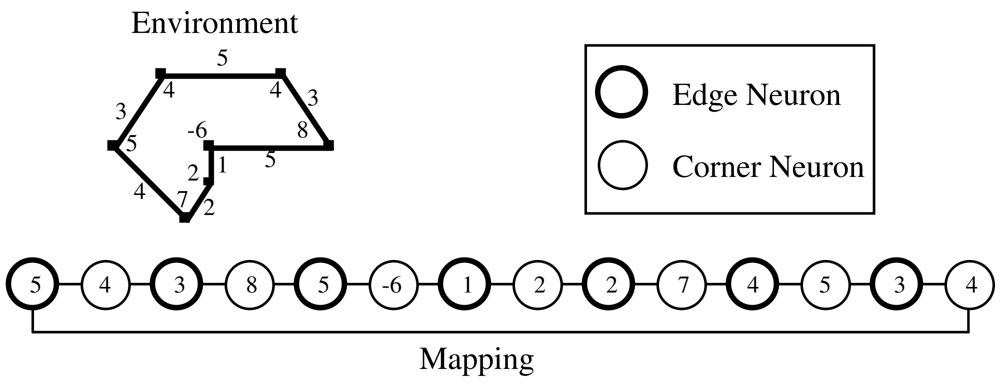


Figure 5.12 The mapping of a simple environment.

This simple linear memory has the ability to store neurons in a simple and quick manner. There is no ordering of the different landmarks. The landmarks are stored linearly as they are encountered. Since human memory is known to be self organizing, this method is not as biologically plausible as that of [Nehmzow and Smithers 91]. It does however, allow a quick and simple implementation with simple operations.

[35] Biologically, the value may be stored as a threshold.

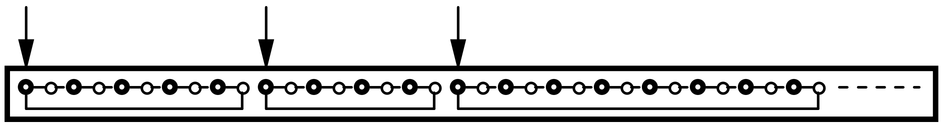


Figure 5.13 Storing multiple landmarks in memory keeping pointers to each landmark. The circles represent memory neurons and the connecting lines indicate spatial adjacency.

RABI actually uses a dual memory system consisting of a *short term memory* (STM) and a *long term memory* (LTM). The STM has a capacity of 8 neurons whereas the LTM has an unlimited capacity. When tracing out a landmark, the corner and edge information is stored in the STM only. Once this memory becomes full, it is transferred to the LTM for permanent storage. The STM is used as a means of temporary storage for use in comparing "chunks" of sequential features to identify landmarks. The robot does not begin storing information in LTM unless it believes that this information pertains to a new landmark. Essentially, the robot fills up the STM and compares it with existing landmarks looking for a match. If a match is not found, the robot then begins a complete trace. More is explained in the next chapter.

When tracing a landmark' s perimeter, a new pointer is created and new neurons are appended to the memory for each corner and edge encountered. Appending neurons can be done in O(1) time. Once learnt, the neurons remain in the memory [36].

5.2.2.5 Storing Data in the Memory

During the edge following behavior, the feature extraction circuitry will detect corners and edges. These corners and edges must be stored sequentially in memory. Because of the alternating nature of the landmarks, each time a corner has been turned, the corner and the last edge traveled must be stored in memory. Figure 5.8 and Figure 5.9 show the mechanisms responsible for detecting the corners. Figure 5.14 shows the additional neurons required to instigate the storage process.

[36] There is an assumption here that the environmental landmarks remain unchanged.

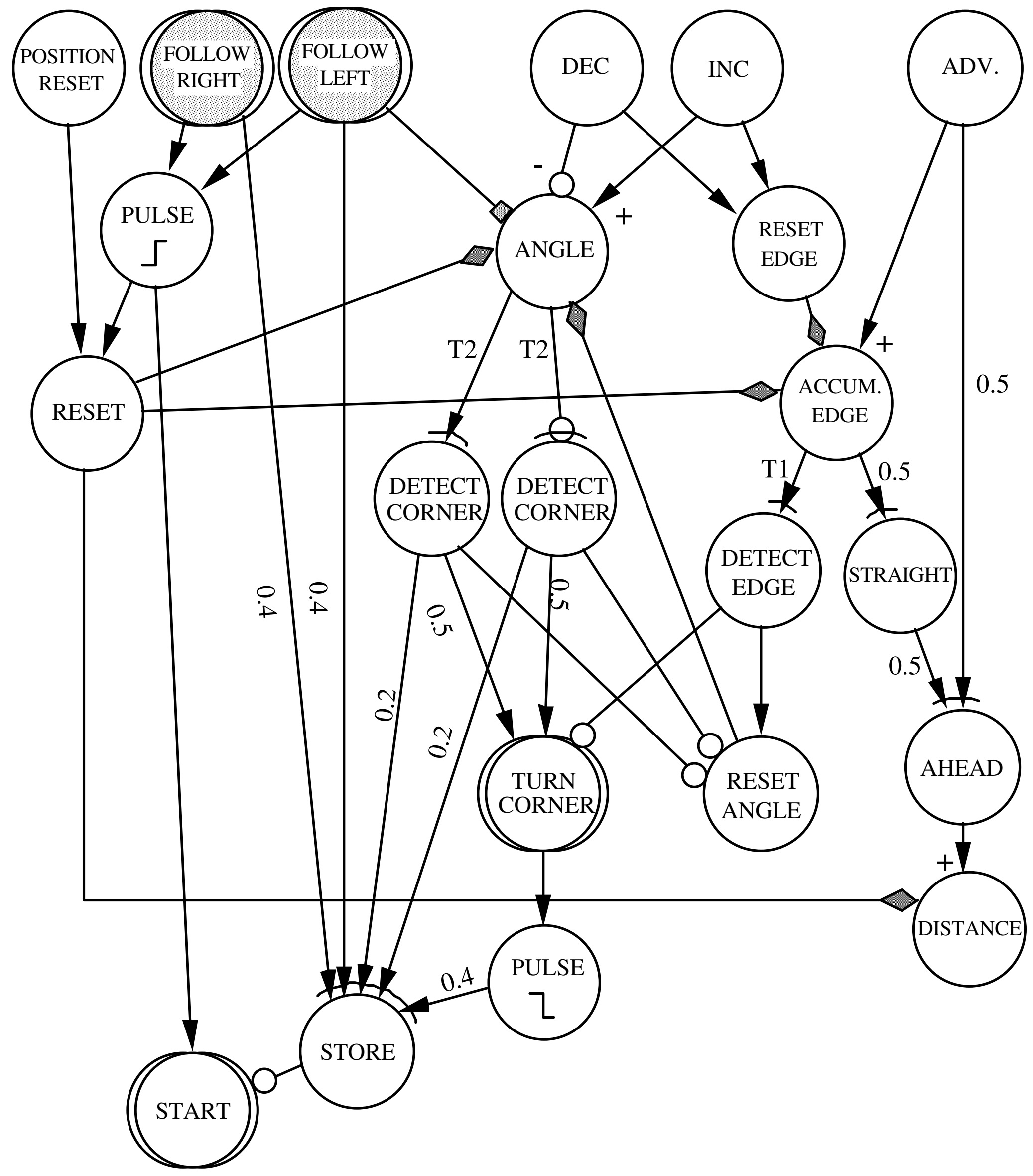


Figure 5.14 The expanded neural circuit for feature extraction. This circuit includes neurons to instigate memory storage.

In the circuit, a PULSE neuron is excited by the TURN CORNER neuron. This falling-edge pulse neuron emits a high signal whenever the TURN CORNER neuron is disabled. This occurs once each time a corner is turned. The STORE neuron is used as a threshold to make sure that the new corner and edge information should indeed be stored. Since each neuron in the circuit is always computing, there is a need for a mechanism to disable the storage process unless the robot is tracing an obstacle (i.e. exhibiting the edge following behavior). If the robot is performing the edge following behavior and there is corner information present, then the STORE neuron will emit a high output, resulting in memory storage. The START neuron is used as a flag to indicate that a new landmark is being traced. A PULSE neuron enables this start neuron whenever the edge following behavior begins. The START neuron remains "on" until the first corner/edge pair is stored.

The memory was not implemented as a neural circuit since the process of explicitly creating new memory locations and storing data within it is not a trivial task for a neural circuit . Furthermore, it is not clear how the landmark pointers would be kept within the memory. A memory system has been implemented which was coded with a top down strategy. The system is responsible for storing and matching memory neurons as well as navigation. The memory system was created as a separate unit which interfaces to the neural circuitry through a handful of neurons as shown in Figure 5.15.

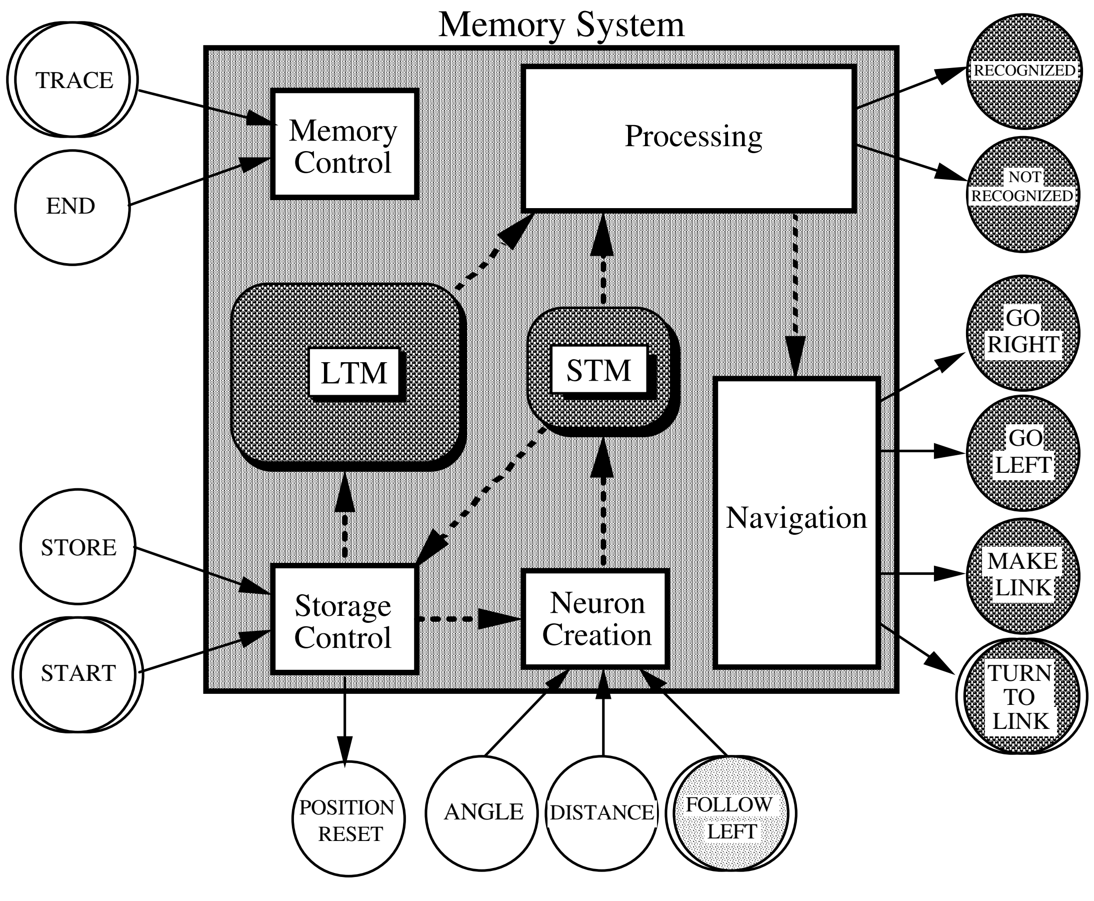


Figure 5.15 The external neurons connected to the memory unit.

When the memory receives a high signal from the STORE neuron, new angle and edge neurons are created with values pertaining to the stored energy in the DISTANCE and ANGLE neurons. The START neuron indicates whether or not the new corner and edge is part of a brand new landmark. The POSITION RESET neuron is excited by the memory in order to reset the accumulative neurons to begin measurements for the next corner and edge. The FOLLOW LEFT neuron inputs a signal to the memory so that when storing the information, the proper sign of the angles are used. That is, depending on the direction traveled along the obstacle (clockwise or counter clockwise), the angles will differ in sign. An angle of 60 during clockwise traversal for example, would represent an angle of -60 during a counter clockwise traversal. The TRACE and END neurons are described in the next section; essentially they are used to determine if the robot is tracing and obstacle and when the robot has completed the traversal.

Once the memory becomes full, a processing unit compares the STM with the LTM to look for a match. The RECOGNIZED and NOT RECOGNIZED neurons are thus excited according to whether or not the contents of the STM matched in the LTM. These neurons are used to enable and disable appropriate instinctive behaviors. As will be mentioned below, the NOT RECOGNIZED neuron is responsible for enabling the boundary tracing process. The GO LEFT, GO RIGHT, MAKE LINK and TURN TO LINK neurons are all used for navigation purposes; they are discussed in the next chapter.

5.2.2.6 Completing a Landmark

In order to map out a perimeter, the robot must be able to detect when it has completely gone around the obstacle once. One method of detecting this "full loop" is to store the initial position and orientation when starting to follow the perimeter. While mapping out the border, the position and orientation of the robot is updated with respect to the starting position. Once the robot arrives back at the starting position with the starting orientation, the perimeter has been completed. Though simple, this method encompasses positioning problems. Due to the inaccuracies of robot motion, there will be a growing amount of positioning and orientation error while traveling [37]. This could lead to false detection of the start position as shown in Figure 5.16. In the diagram, the accumulated error during traversal causes the measured path to be slightly off from the actual traveled path, and consequently the start position is believed to have been reached when indeed it has not.

[37] Weight shifting in walking robots could sometimes affect the amount of turning and forward movement per time unit. That is, in one turn the robot may change its orientation by 10 and 14 in another.

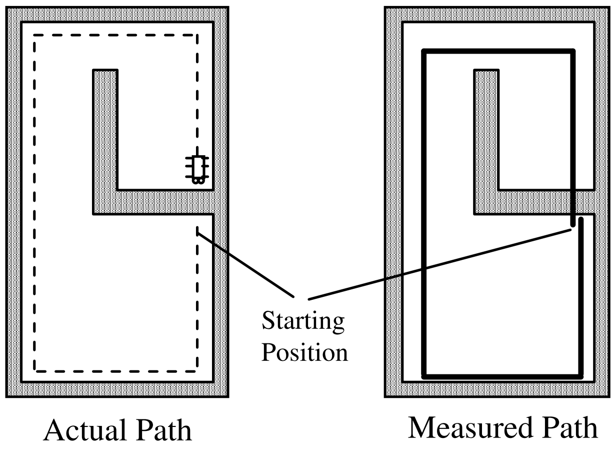


Figure 5.16 False detection of the starting position.

Another possible method of attempting to identify a loop completion is to traverse the boundary more than once, attempting to match up the features during the second lap. This method will not work unless the obstacle has distinct features. A square perimeter, for example, will always detect identical edges and corners no matter how many laps around the border. Complex perimeter shapes also cause a problem if there are any repetitive sequences in its features.

Perhaps the best way of detecting a completion of the perimeter is to leave behind something that can be detected at the end of the loop. This is the only feasible approach given the lack of precise metric information and distinct landmark features. A chemical residue left at the starting point would be the choice of ants, but a robot would need additional sensors to be able to detect the chemical. Furthermore, the robot would need to replenish its chemical supply, unlike ants whose chemical is biologically replenished. It is easier to create some sort of electronic sensor that could detect a metal rather than a chemical. A robot could leave behind a marker in the form of a small metal disk along an edge of the perimeter. [Dudek et al. 91] use a similar type of marker as part of an exploration strategy. All that is needed is a small actuator to deploy the disk and lift it back up once the trace is completed. One more sensor is also needed to detect the disk when it is underneath the robot [38]. The neural circuit of Figure 5.17 contains the simple mechanism for disk dispensing.

[38] The software version of RABI simulates such an actuator and sensor. Due to time constraints, the hardware version was not blessed with such a feature.

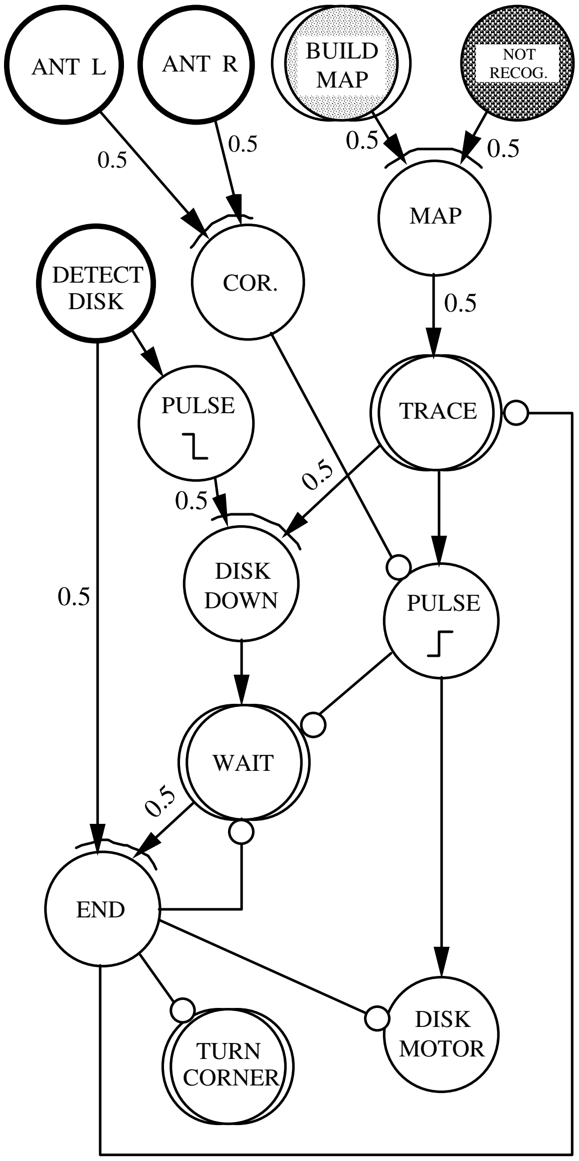


Figure 5.17 Obstacle tracing and disk dispensing network. The detection of the disk during tracing indicates a completed traversal.

The TRACE neuron is excited whenever the robot is about to begin tracing out a landmark during the map-building process. When this sustain neuron is first excited, a PULSE neuron excites the DISK MOTOR neuron, which is a motor neuron connected directly to the disk dispensing actuator. By exciting this neuron, the disk is dropped. The DETECT DISK neuron connects directly to the disk sensor. Since the disk is dropped beneath the robot, the robot' s disk sensor will detect the disk immediately. The disk must be ignored until the robot has completed the tracing of the landmark. The DETECT DISK neuron excites a PULSE neuron whenever the disk is no longer detected (i.e. just left behind). Once excited, this falling edge pulse neuron "turns on" a WAIT neuron which is used to indicate that the robot is now waiting until the trace is complete before detecting the disk again.

The WAIT neuron is initially disabled by the rising edge pulse neuron when the tracing begins. When the disk is detected again, the combined output of the DETECT DISK and WAIT neurons turns on the END neuron which represents the end of a trace. The END neuron then inhibits the DISK MOTOR neuron, resulting in the disk being picked up. This END neuron disables the TRACE neuron and disables the TURN CORNER neuron of the feature extraction network in Figure 5.8 to ensure that the last edge and corner are stored. The MAP neuron is used to start the tracing process whenever the robot encounters environmental features that are not recognized while it is building a map.

The idea of dropping a disk as a marker seems simple and efficient. There are, however, a few problems that can arise. The robot must be able to detect the disk once it arrives back at the starting location. If the robot is a different distance away from the wall than when it started, the disk may not be detected. This is not a problem with a simulated robot since there are no positioning errors. For real robots this would be a problem unless the robot had some sort of mechanism to locate a disk that is nearby, perhaps within a few inches. Another problem occurs when the robot drops the disk within a corridor or hole as shown in Figure 5.18 (a) and (b) respectively.

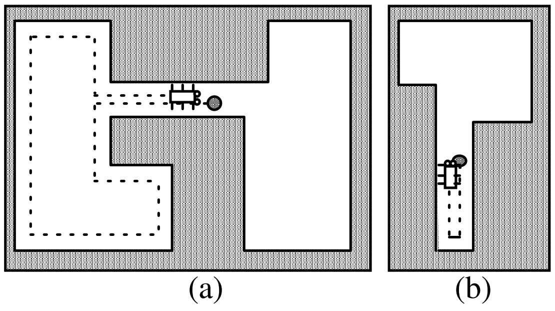


Figure 5.18 Dropping a disk in a corridor or hole.

If the disk is dropped in a narrow passageway, the robot will detect it when coming out of the passageway. Thus, the disk will be picked up before a complete trace is made and an incomplete representation of the perimeter is stored in memory. The easiest way to prevent this from happening is to disallow the robot from dropping the disk inside a narrow passageway. That is, if the robot detects obstructions on both sides of its body, then it should not drop the disk until it is free from one of the obstructions. One additional link is needed to solve this problem.

The COR. neuron of Figure 5.17 corresponds to the neuron from the vacancy network. When a corridor is detected, the PULSE neuron from the disk dispensing network is inhibited, preventing the robot from dropping the disk until exiting the corridor.

5.3 Combining Mapping Techniques

Grid-based mapping techniques can provide a simple and accurate map provided that the robot has precise odometry mechanisms. Landmark-based mapping is useful when precise positioning is not possible. Perhaps an integration of both techniques could provide an elegant solution in which the robot could take advantage of both methods. Local grid maps could be developed and embedded into a global landmark-based map or vice versa. With the quadtree approach, open areas can easily be identified as large grid units. These large open spaces can be labeled as landmarks and combined with a landmark-based map. This would, for example, allow a robot to identify "rooms" in a building by comparing their sizes and structure. Figure 5.19 shows a topological mapping for an indoor environment. Each ring represents a network of corner and edge neurons representing the boundaries of a room. The links connecting the rings correspond to topological adjacencies of the rooms.

This integration of mappings would also allow the robot to chose between free space or obstacle space. If the robot wished to remain in vacant areas, then the free space (grid) map would be used since it maps out all paths that may be traveled. When the robot wishes to remain close to obstacles, it would then utilize the landmark map.

The foremost advantage of this combined map, would be that of landmark identification. The relative location in a grid would aid in identifying similar landmarks. The problems associated with landmark identification are discussed in the next chapter. This dual mapping strategy was not implemented for RABI due to time constraints. It is, however, a promising area for future research.

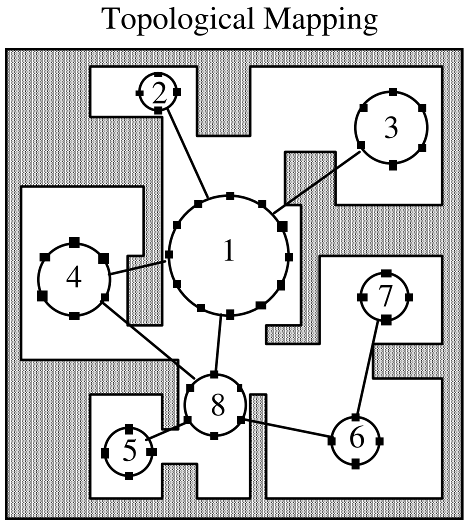


Figure 5.19 A topological mapping of an indoor environment. The circles represent landmark mappings.

5.4 Summary

There are a variety of mapping techniques used to map out 2D environments. Among these, landmark-based techniques can provide adequate mappings without the common positioning problems encountered more often in the other methods. Moreover, landmark-based mapping is possible with very simple sensors and requires less memory in general to store the information. This approach to mapping is not without problems since the features may be similar in a many of the landmarks, causing a problem related to landmark distinction.

The memory system used by RABI interconnects with the neural circuits through a handful of neurons. As a result, the memory system is essentially a "black box" whose contents can be altered without having to change the neural circuits. The use of a dual memory allows the features of landmarks to be grouped together in "chunks" for partial matching of landmarks. This allows the robot to ignore landmarks that have already been investigated so as not to waste time when mapping. It can also allow the robot to detect a landmark without having to trace the entire perimeter. By mapping the environment, the robot is able to learn the locations of food sources and obstacles. This learning process provides a form of adaptivity since the robot is essentially adapting to the environment by learning the areas important for its survival needs.

Chapter 6
Navigating in the Environment

The use of internal maps can directly affect the behavior of an animal. These maps can contain topological and spatial data that can allow the animal to efficiently navigate from one location to another. The maps can also associate specific locations with certain types of stimuli such as food, energy, hazardous regions, dead ends, unfavorable climates etc. Knowledge of the locations of such stimuli may be crucial for survival as well as being valuable sources of information that could improve the efficiency of the robot. A robot can use an internal map for the same reasons, learning the locations of various energy sources etc.

In order to use a map, a robot must handle three subproblems. First, the robot must be able to identify its location within the map. This presents a problem due to data discrepancies, imprecise map data and ambiguities between landmarks. The use of a landmark-based mapping strategy provides only an estimated position and therefore the robot' s location is never known to precise detail. Second, the robot must have some sort of navigational system that continually updates its approximate position within the internal map as it moves among landmarks. Lastly, the robot needs a mechanism that allows it to travel efficiently from point to point. This point to point navigation may require the robot to travel from one landmark to another. Therefore, it may be useful for the robot to determine where it is with respect to the other obstacles in the environment. All of these problems are discussed in this chapter along with solutions that were implemented by RABI.

6.1 Navigation Strategies

When navigating in an environment, the robot needs some sort of indication of where it is in the environment so that it can travel efficiently from point to point. The position may not need to be known precisely, sometimes a rough estimate is adequate to provide a clue to the robot' s location. As long as the robot has a general idea as to where it is in relation to other objects and locations in the environment, then navigation is possible. Usually the robot builds up a map from which it is able to compute a path from one location to another.

For simple mappings such as the grid-based approach, the robot relies on dead reckoning. With this method, the robot usually has an odometer and orientation sensor. Every time the robot moves forward or turns, the robot' s position in the map is updated with respect to some known starting location. Due to the inaccuracy of robot sensors and imprecise mappings, there is an accumulating error associated with each position. This error must be reduced (reset) occasionally by verifying a precise location in the environment. [Rosten and Krotkov 92] give a description of the dead reckoning technique. Due to the imprecise measurements of RABI' s forward and angular movements, the dead reckoning approach is not reliable.

Once the position is known, the robot could then compute a path from its current location to its final destination. For the shortest Euclidean path, Djikstra' s algorithm provides an elegant solution. Sometimes, there are certain regions in the environment which are more traversable than others. In this case, there may be weighted regions in the environment which could affect the overall desired path. [Mitchell 89] describes the problems of navigating with weighted regions. This method is useful for the polygonal mapping techniques, for which again there is a problem with position estimation.

Consider landmark-based mapping. A landmark-based map contains information about the environmental structure. If these structures (landmarks) can be distinguished from one another, then a robot could determine its location in the environment by identifying a single landmark. Thus, a robot that is picked up and placed down in a different location would easily be able to relocate itself by first finding an obstacle edge and tracing its boundary until the obstacle is identified (matched in memory). Once matched, the robot has a fairly accurate indication as to where it is in the environment and where the other landmarks are with respect to its present location. Clearly, this approach does not require accurate position estimates since the landmark shapes are the only cues as to the robot' s location. This approach brings with it many problems associated with landmark identification.

6.2 Landmark Identification

When tracing out landmarks as mentioned in the previous chapter, the robot builds up a map containing representations of the obstacles in the environment. These landmarks are stored in memory for future reference. In order to make use of this stored information, the robot must be able to identify the landmarks it encounters by comparing it with the ones in its map. This matching process presents a host of problems dealing with discrepancies, identical objects and inaccurate representations.

6.2.1 Data Discrepancies

During the tracing of an obstacle perimeter, the robot makes rough estimates of edge lengths and corner angles. With binary collision data, there is no way of accurately measuring these angles and distances. Moreover, since the robot cannot turn while advancing, convex corners must be partially passed before turning begins. This presents a problem with tracing borders clockwise and counter-clockwise. Figure 6.1 shows a snapshot of the mappings produced from a clockwise and counter-clockwise traversal of a simple rectilinear environment.

The snapshot shows that there are differences in edge lengths and corner angles. In addition, there is a kind of symmetry between the two mappings. This symmetry is the direct result of the cornering technique of the edge following behavior. These two mappings are very similar indeed. Getting them to match with each other may not be a problem since the differences are small.

A bigger problem arises due to the binary characteristic of the antennae. Since the antennae have no concept of proximity, then there is a little bit of unpredictability as to how close the robot will get to an edge before it is detected. Thus, traversing a perimeter in the same direction can yield a different mapping if the robot is closer or further from the edge the second time around. This presents a misalignment problem as shown by the snapshot in Figure 6.2.

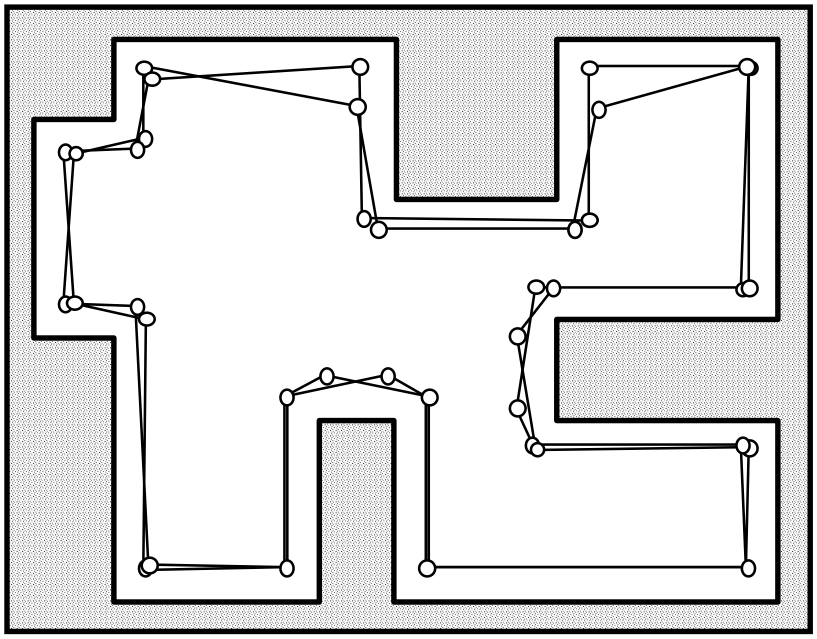


Figure 6.1 CW and CCW mappings of a rectilinear environment.

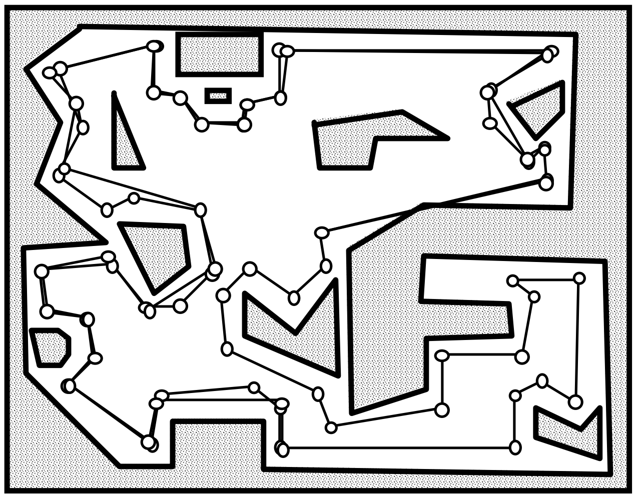


Figure 6.2 Misalignment problem caused by the lack of proximity detection.

The snapshot shows the differences in the mappings obtained from three laps around the environment. Only two "paths" can be seen in the image, and in the bottom right portion of the environment, only one path is visible. This is because after one and a half times around the environment, the robot became "aligned to the edges" This alignment ensures that any more laps around the perimeter will result in an identical mapping. The binary characteristic of the antennae and the accuracy of the turning and forward movements of the simulated robot are responsible for this phenomenon. This example shows that small differences in the distance between the robot and the obstacle can significantly affect the mapping that is produced.

Another problem that emerges from the binary nature of the antennae is that of estimation. A mapping of a curved surface will be estimated as a polygon. An environment such as the circular environment in Figure 6.3 would be detected as a polygonal shape. The consequence of this estimation is an imprecise representation of the environment.

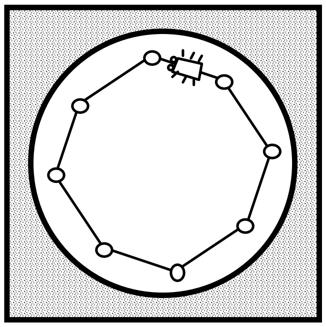


Figure 6.3 Estimating a polygon from a circular environment.

Furthermore, such an environment does not allow corner identification since anywhere along the border, the edges and corners are of the same magnitude. Thus, no corner is distinguishable from the others and there is no way of determining, with any accuracy, the robot' s location in the environment. In fact, this problem occurs for any environmental shape which is symmetric as mentioned previously in Figure 5.4 (a) and (b).

Each of these discrepancies must be overcome or handled such that the robot is able to identify the landmarks solely on the basis of consecutive edge and corner information. Since most of the errors are due to minor angle and edge length differences, the problems can be fixed by allowing an error when matching neurons. In the case when different mappings are created for the same obstacle as in Figure 6.2, the robot must be able to match up both paths in memory. Since these paths have a different number of neurons, the robot should be able to generalize the two paths by recognizing that they indeed represent the same obstacle.

6.2.2 Distinguishing Between the Inner Obstacles and the Border

The border of an environment is a unique type of landmark. It is different from all the other landmarks since it surrounds them. It would be useful to be able to distinguish the landmarks as being either an inside landmark or the environmental border landmark. Insects when trapped, spend much of their time searching the enclosing boundaries in order to determine a way out. In the case of indoor robots, the energy sources may lie on walls (i.e. sockets) which are part of the environmental border.

As it turns out, this task is easily accomplished. Each landmark can be represented as a simple polygon. Due to the winding (spiraling) properties of polygons, a traversal of the interior angles of a polygon will yield an angle sum of either 360 or -360 . This is proved by [Carmo 76]. Furthermore, since the outer environmental perimeter contains all other obstacles, it is similar to a polygon with holes in which the border has an opposite orientation. Following the boundary of an inner obstacle on the left will always result in a counter clockwise traversal. The outer border however, would result in a clockwise traversal.

The winding property of polygons ensures that by summing all the values of all the corner neurons, their sum should be 24 (i.e. 15 x 24 = 360) or -24 units [39]. Assuming that all tracing keeps the border to the left of the robot, then the outer border will yield a -24 unit sum where as the interior obstacles will yield a +24 unit sum; with an error associated with each value of course. This angular sum provides a method of distinguishing the outer environmental perimeter from the inner obstacles. Actually, the sign of the angle sum is adequate for making the distinction.

6.2.3 Identical and Similar Obstacles

When mapping out an environment, the robot may encounter two identical objects. If this is the case, then there is no way of distinguishing between the two obstacles since the landmark-based mapping technique does not record spatial adjacency. In the environments of Figure 6.4 for example, only two obstacles can be distinguished; the outer border and the inner obstacle. Moreover, due to the lack of spatial information and allowable error, both environments would appear identical.

[39] In practice, the sums are not usually too accurate resulting in approximately + or - 10 .

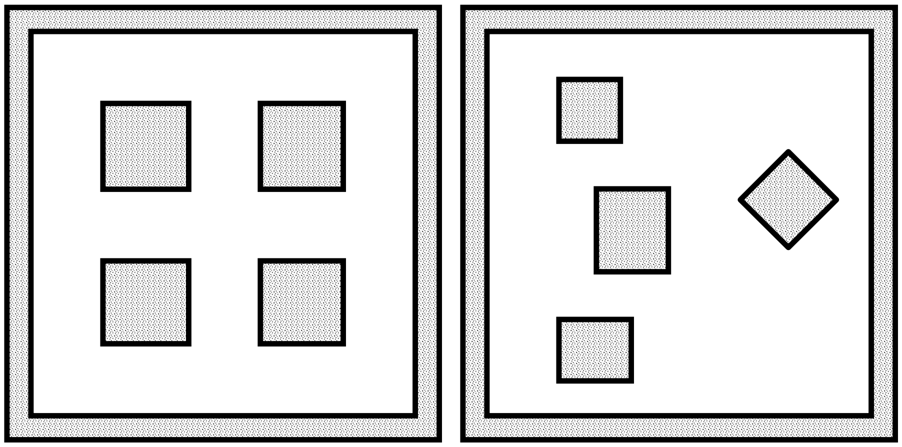


Figure 6.4 Two environments that appear identical due to the lack of spatial information and the allowable error during matching.

This example points out an important shortcoming with the landmark-based approach. As mentioned in the last chapter, perhaps a combination of different mapping strategies can combine the landmark map with a spatial map obtained from dead reckoning. This combination would enhance the overall map allowing the two environments to be distinguished. This is a topic for future research but for now , an assumption is made that the environment is sufficiently complex with obstacles of different shapes.

6.2.4 The Matching Process

The problem of matching features with an internal map is not as trivial as it may seem. Due to the discrepancies discussed, the robot may actually map out the same obstacle more than once, each time producing a slightly different map. An efficient pattern matching process should be able to somehow recognize that two slightly different mapping sequences actually represent the same obstacle. It is impossible to match sequences with 100% accuracy since the amount of discrepancy of any one mapping is unknown. The best that the robot could do is to allow a specific error in the corner and edge sizes during the matching process. As a result, if any two mapping sequences match within the error allowed, then they are considered to be mappings of the same obstacle. For this reason, similar obstacles cannot be distinguished using solely their features.

Given two sequences of obstacle mappings, it must be determined if they represent the same obstacle. If they do, then the two mappings are combined into one map, storing the differences. This process represents a form of generalization in which two or more mappings are generalized into one map with multiple paths. Figure 6.5 gives an example of three similar mappings of the same obstacle and the generalized map resulting from the matching process.

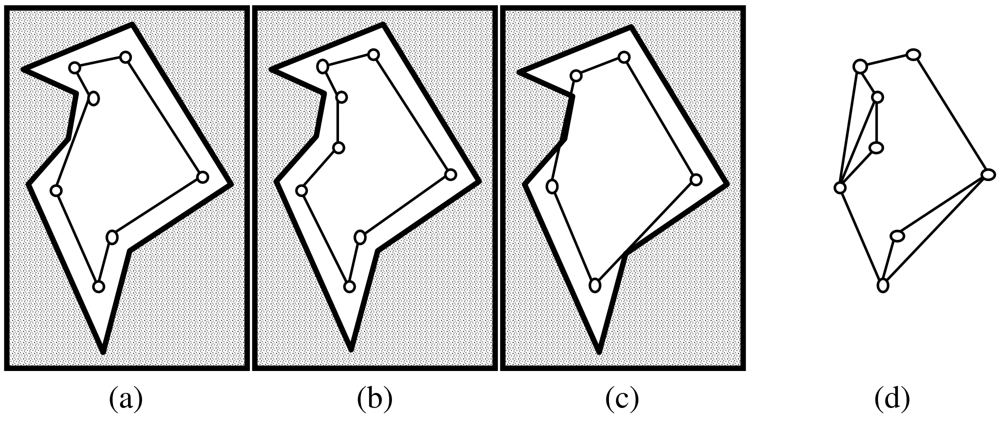


Figure 6.5 Three similar obstacle mappings (a), (b) and (c) and the resulting generalization (d).

Note that the three mappings have small differences in the corner angles. These differences cause the mappings to have a different number of neurons since some corners are occasionally not detected (due to their small angle). The generalized map combines the similar paths of all three mappings and adds additional paths that correspond to the differences. The memory neurons in these additional paths are termed *bypass neurons* since they bypass the existing path. This generalization technique prevents the memory from storing duplicate copies of mappings.

The generalization process is essentially a matching problem in which neurons from two layers [40] are matched sequentially until there is a discrepancy or until all neurons have been matched. The pseudo code for the basic matching algorithm is described with three routines which are presented in Appendix A. The algorithm attempts to match each layer in memory with the new layer. It does this by matching the neurons sequentially one by one. Since the layers are circular linked lists of neurons, there may be a problem in determining the starting point for matching the two layers. That is, a complete match can only be determined whenever the two starting neurons indeed represent the same landmark. Figure 6.6 for example, shows

[40] A layer here indicates a sequential list of memory neurons representing edges and corners from an obstacle mapping.

two sequences of neurons that are indeed identical, but they are offset due to the circular nature of the lists.

34	3	17	5	8	-4	20	6	4	-6
-4	20	6	4	-6	**34**	3	17	5	8

Figure 6.6 Two identical matching sequences which are offset due to the circular nature of the linked list.

In order to rectify this offset problem, the algorithm chooses the first neuron from each layer and attempts a match. If no match is found, the next neuron in one of the layers is chosen as a starting offset and then a match is attempted again. In worst case, each neuron of a layer is used as a starting point requiring the matching process to be attempted once for every neuron in the layer. Since each neuron in each layer may be processed for matching, this method has a worse case time complexity of O(nm^2) where n is the number of layers and m is the maximum number of neurons per layer.

The generalization algorithm uses the basic matching algorithm discussed with additional processing during the matching process. Since there may be two mappings of an obstacle that have a different number of neurons, then there may be a need to match a single neuron from one layer with multiple neurons from another layer so that slightly different mappings will match and be generalized as seen in Figure 6.5. To do this, the algorithm adds an additional condition to the matching process. If a neuron does not match another, the algorithm looks ahead to determine whether several neurons can be combined to make a match. Figure 6.7 (a) and (b) represent the layered neurons from two obstacle mappings. The generalized mapping is shown in (c). Note that the two layers match (within a small error) except for the darkened edges. When doing a sequential mapping, the generalization algorithm will match darkened edge 14 of layer (a) with the 7, 2, 7 piece of layer (b) since the combined edge length of the piece matches (within a small allowable error) with the edge length of 14 from layer (a). Similarly, the 6, 3, 14 piece of layer (a) matches with edge 21 of (b) for the same reasons. The algorithm keeps track of these "special piece-matches" and uses them to create a combined, generalized map as shown in (c).

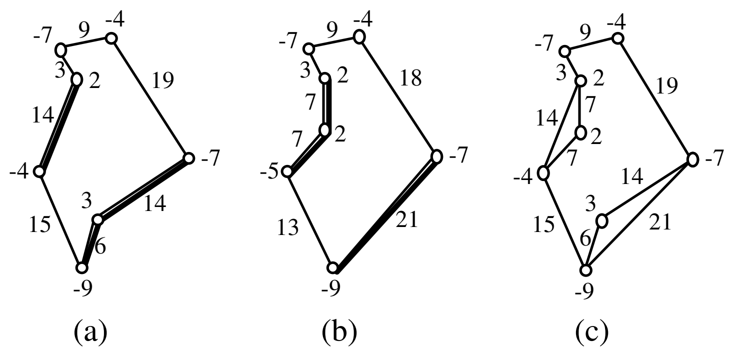


Figure 6.7 Two layers of neurons (a) and (b) that represent the same obstacle mapping. The darkened lines show the areas that differ. The resulting generalized mapping (c).

This process of matching pieces with a single neuron does not add to the overall time complexity since each neuron is being examined only once per match attempt. Once combined, the two original layers are discarded and the combined layer is stored. This generalization process is useful since it eliminates duplicate information from similar obstacle mappings, storing only the differences.

6.3 Improving Identification Through Additional Sensor Information

By now it is clear that the landmark-based mapping technique is unable to accurately represent the environment in some situations. It is easy to restrict the robot to certain environments in which the landmark-based mapping technique flourishes. If the robot is required to operate in a simple environment, then a problem arises with using this technique. The problem is mainly due to the lack of sensor information. The robot is equipped with only simple sensors and thus must make many assumptions to fill in the gaps. By adding additional sensors such as a compass, gyroscope, beacon detector etc, the robot may be able to better distinguish the environmental obstacles. This thesis however, is not aimed at developing a complex map building robot, instead the topic of interest is to determine just how well a robot can perform with a minimal amount of sensor information. Nevertheless, this section presents a couple of ways in which the landmark identification process can be improved upon by adding additional sensors.

6.3.1 Global Orientation and Outside Influence

One method of improving the identification process is to determine spatial adjacency. This can be done by using external cues such as natural magnetic fields (north on a compass), lights or beacons. Overhead lights for example, have been known to affect the way in which a mouse learns and travels through a maze. In fact, in some cases, the internal map of a rat was found to be highly dependent upon such external cues. It is possible for a robot to make use of an outside influence to give it a sense of global orientation similar to that of a compass. An external light source can act as a direction indicator which would allow the robot to determine the orientation of the obstacle edges relative to one another. This additional orientation mechanism could also provide a useful tool for determining spatial adjacency. The robot could determine the corner of an obstacle that is closest to the outside source. The dotted lines in Figure 6.8 represent the path from the closest corner of each obstacle towards the light source.

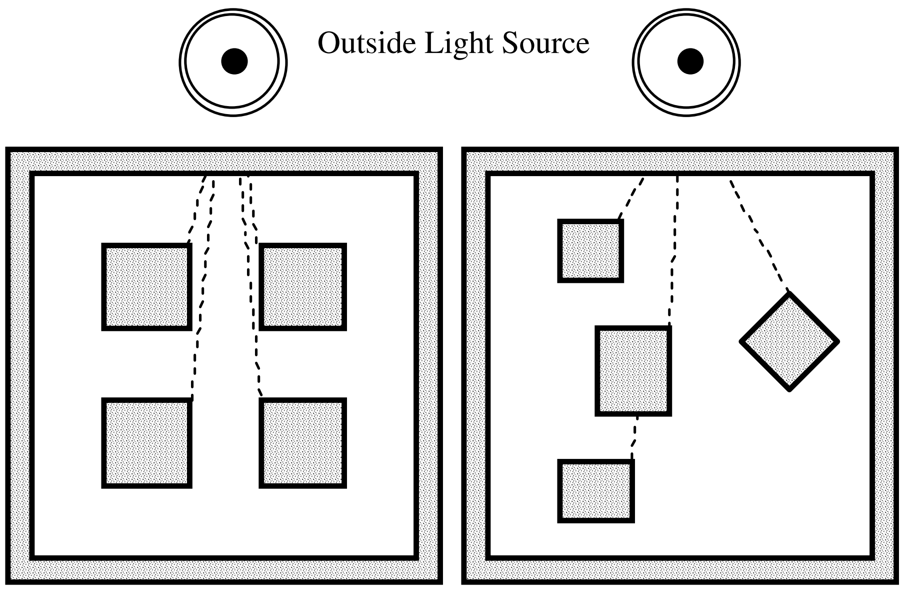


Figure 6.8 Additional spatial information obtained from an external cue.

In the first environment, the top two obstacles are distinguishable from the lower two since their corners are closer to the external source. Furthermore, the obstacles can be distinguished from left to right by the angle that each corner makes with the source. In the second environment, all obstacles are distinguishable in a similar manner. This method may be better than using a simple compass since the additional angular information is not present in a compass.

It is clear that external cues can play an important role in the gathering of spatial information which is useful for map building. The external cue however, must be fixed and easily identified. Using the sun as a guidance tool may cause problems due to the earth' s rotation. In addition, clouds and nightfall can prevent the robot from obtaining the external cue information. Indoor environments also have problems with lighting as there may be many light sources within it. Perhaps a better external cue could take the form of a beacon which emits ultrasonic bursts or infrared light, but these require the environment to be altered which may not be suitable for many nanobot applications.

6.3.2 Single, Dual and Infinite Disks

It has been shown that a disk can be used as a marker for detecting the end of a complete traversal of an obstacle. With only one disk, the disk must be picked up after each obstacle trace for use in the next trace. This means that a robot has no indication as to whether or not it has traced out an obstacle except by comparing it with the obstacles in memory. It would be useful to have the robot leave behind a special marker at each obstacle so that it knows if the obstacle has been traced before without having to retrace.

Consider a robot equipped with N + 1 disks in an environment with N obstacles (including the border). Now, the robot does not need to pick up a disk that has been laid down since the robot has enough to leave one at each obstacle border. Clearly, the robot now has an indication as to whether it has visited an obstacle border before, since a disk will be encountered along the edge during a traversal. If all these disks are indistinguishable however, the robot would not be able to use them as a means of detecting a full trace. As in Figure 6.9, the robot would begin a trace of obstacle A by dropping disk d_1. During the traversal of obstacle A, the robot would detect disk d_2 which had been left behind from a previous traversal and would not be able to distinguish it from d_1. Thus, the obstacle will be assumed to have been completely traversed resulting in an incomplete traversal.

This is indeed a problem since the existing tracing algorithm depends on the detection of the disk laid down at the start of a trace. Thus, if multiple disks are to be used, they should be of a different type. That is, a type 1 disk could be used for tracing as done previously and all other N disks (of type 2) could be used as markers indicating a previous traversal. Thus, disk d_2 would not present a problem since the robot wouldn' t detect it as a type 1 disk and could continue on until d_1 is found.

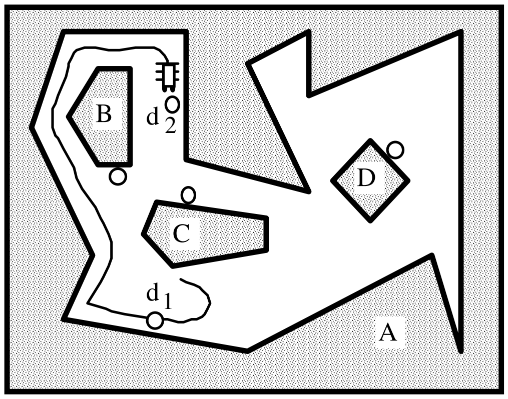


Figure 6.9 Tracing problem with similar disk types.

There is one small matter of efficiency that must be dealt with. Assume that the situation in Figure 6.9 has occurred where the robot detects disk d_2. The robot now knows that it has traced out this obstacle border before, therefore further tracing is not needed. But disk d_1 must be picked up again for use in the next traversal. Thus, even though the robot does not need to trace the obstacle again, it must continue the trace in order to arrive back at d_1 so that it can be picked up again. This is rather inefficient. Perhaps it would be more efficient if the robot could detect if an obstacle has been traced previously without having to trace it out until the marker is found.

One possible method of immediate detection is to place disks all along the border of an obstacle as it is traced as in Figure 6.10. When the robot encounters any edge which has been traced previously, it will immediately detect a disk and thus will not need to trace out the border at all. This method, however, would require a large supply of disks since they are required throughout the environment. Chemical residue may provide a more efficient marker since the robot could easily spray the chemical along its path, similar to ants. The chemical would have to be strong enough with a slow decaying factor so that it is easily detectable over a longer period of time. This spraying would require some form of pumping system and "smell" sensor which may increase the size of the robot.

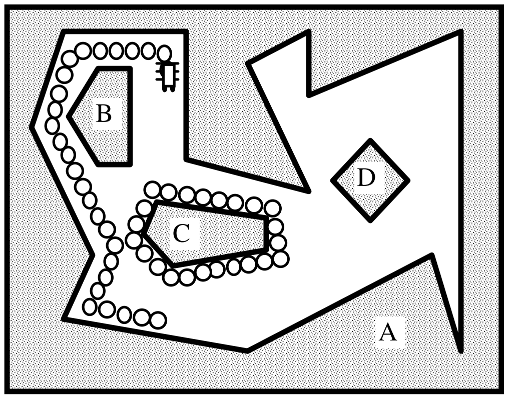


Figure 6.10 Using an infinite number of disks for tracing out obstacles.

Despite the need for more complicated sensors and actuators, this method of using a large disk supply would make the exploratory behavior much more efficient since new obstacles can be found without having to trace out any of the obstacles previously traced. This would allow the robot to trace out all obstacles even if their features are identical.

6.4 Simplified Spatial Adjacency

As mentioned in Chapter 5, [Nehmzow and Smithers 91] and [Mataric 91] present methods of landmark-based mapping techniques that map out only environmental perimeters. Their methods do not handle the mapping of interior obstacles. Clearly, this limits the scope of the usefulness of their techniques. A more realistic and useful mapping should be able to map out the inner obstacles as well. The mapping out of interior obstacles is a simple task when using RABI' s technique. All obstacles are automatically mapped and generalized as encountered. For a robot that must compute efficient point to point paths however, the additional mappings of interior obstacles presents a small problem. The robot may not need to know exactly where these inner obstacles are with respect to the other obstacles and environmental border, but it may need to know how to get from one to another. If for instance, the only energy sources lie at obstacles which are in the middle of the environment, the robot must be able to reach these locations. It would be useful to know which edges of an obstacle' s border lie near the energy sources so that the robot could shoot out in the appropriate direction towards the energy sources when needed. Essentially, the robot needs to recognize the spatial adjacency among the obstacles.

6.4.1 Adjacency Links

This simple form of spatial adjacency can be stored as special links that connect adjacent obstacles as shown with dotted lines in Figure 6.11. Here the dotted lines represent the paths that need to be taken to get from one obstacle to another in the environment. They are termed *adjacency links* and are stored in the memory mapping. The endpoints of these links are termed *adjacency points.*

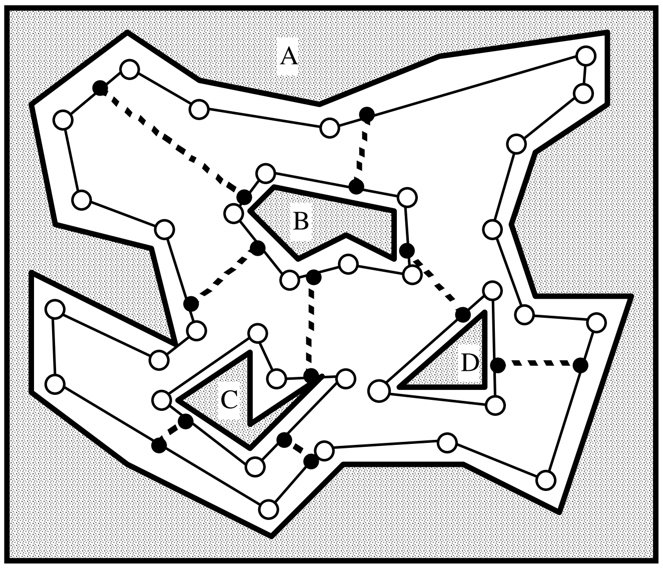


Figure 6.11 Adjacency links (dotted) required to record spatial adjacency among obstacle mappings.

The simplest method of determining these links is to traverse outwards from the inner obstacles. Since the environment is closed off, the robot could walk away perpendicular to an inner obstacle edge until colliding with another obstacle. As long as the robot knows which edge it walked away from and which edge it just encountered, then a link can be created joining these two edges in memory.

The robot will eventually use this adjacency information for obstacle to obstacle path planning. The question that must now be answered is: How does the robot travel along an adjacency link when there is no physical sensor information to guide it ? As will be seen later, the answer to this question involves dead reckoning. By just creating a link that joins two edges in memory, not enough information is stored since one edge may lead to many others as seen in the example environment of Figure 6.12.

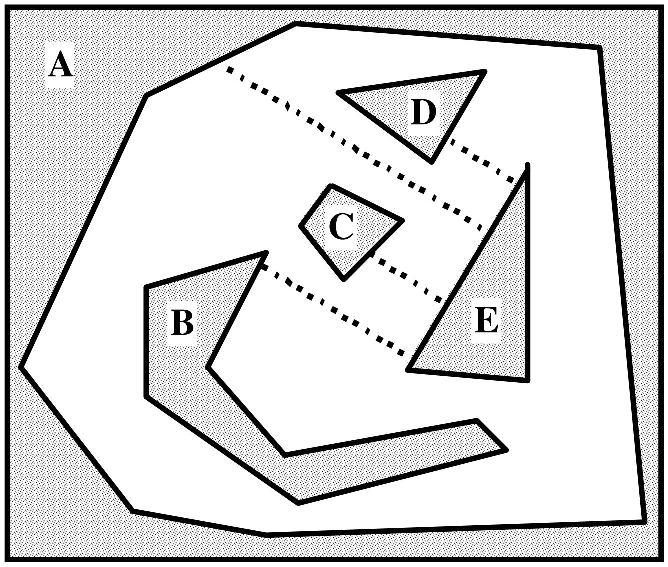


Figure 6.12 The edge displacement problem. One of the edges of E can lead to an edge of A, B, C or D.

The example shows that an edge of obstacle E can lead to many other obstacles depending on where along the edge the robot turns away. This presents a problem if the robot wanted to go from E to D for instance. The robot needs to know how far along the edge of E it must go before turning away so as to reach obstacle D. For this reason, the adjacency link stored in memory must also record the distance along E that the robot must travel before "shooting out" towards the desired obstacle.

To keep things simple, the robot could always turn perpendicular to the inner obstacle edge when creating the adjacency link. The other end of the link however, generally will not have the same 90 angle. The link from E to A in Figure 6.12 for example starts with a 90 angle and ends with approximately a 115 angle. When traveling from A to E along the same link, the robot would need to turn 115 to get to E. Thus, the adjacency link must also store the angles that the adjacency points make with the edges.

This information can be stored in memory by creating *adjacency corner* and *adjacency edge* neurons which are similar to the corner end edge neurons in the basic memory system. Figure 6.13 shows an example of how the adjacency links are added. In the example, the mappings of two obstacles (an inner obstacle and a border) is shown before and after the addition of two adjacency links.

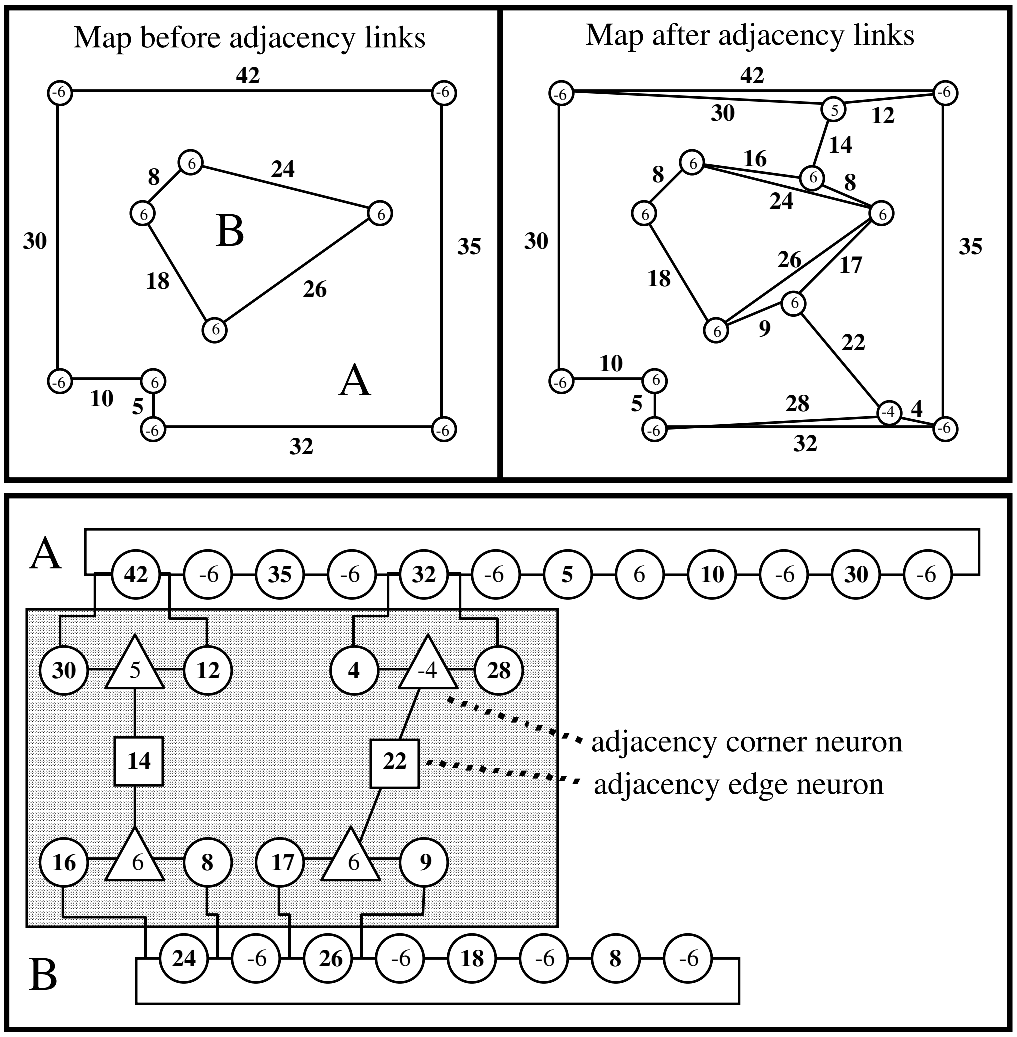


Figure 6.13 Environmental mappings before and after two adjacency links are added. The shaded region contains the additional connections required to store the adjacency links.

Each addition of an adjacency link creates two additional edges corresponding to the partitioning of the edges on which the endpoints of the link lie. These two edges are joined by an adjacency corner neuron in a similar fashion to the basic memory system. Two adjacency corner neurons of an adjacency link are joined by means of an adjacency edge neuron. This method of adding adjacency links allows the already existing memory to be unaltered. Instead, the link is added as a set of bypass neurons as done in the generalization process.

6.4.2 Creating an Adjacency Link

In order to create these adjacency links, the robot needs to have a mechanism that allows it to turn perpendicular to an inner obstacle edge, walk straight until another edge is detected, turn until it is parallel to the detected edge and record the distance and angle information at the adjacency points. This entire process is easily coded with a neural circuit which can be initiated by exciting a single neuron. The circuit is given in Figure 6.14.

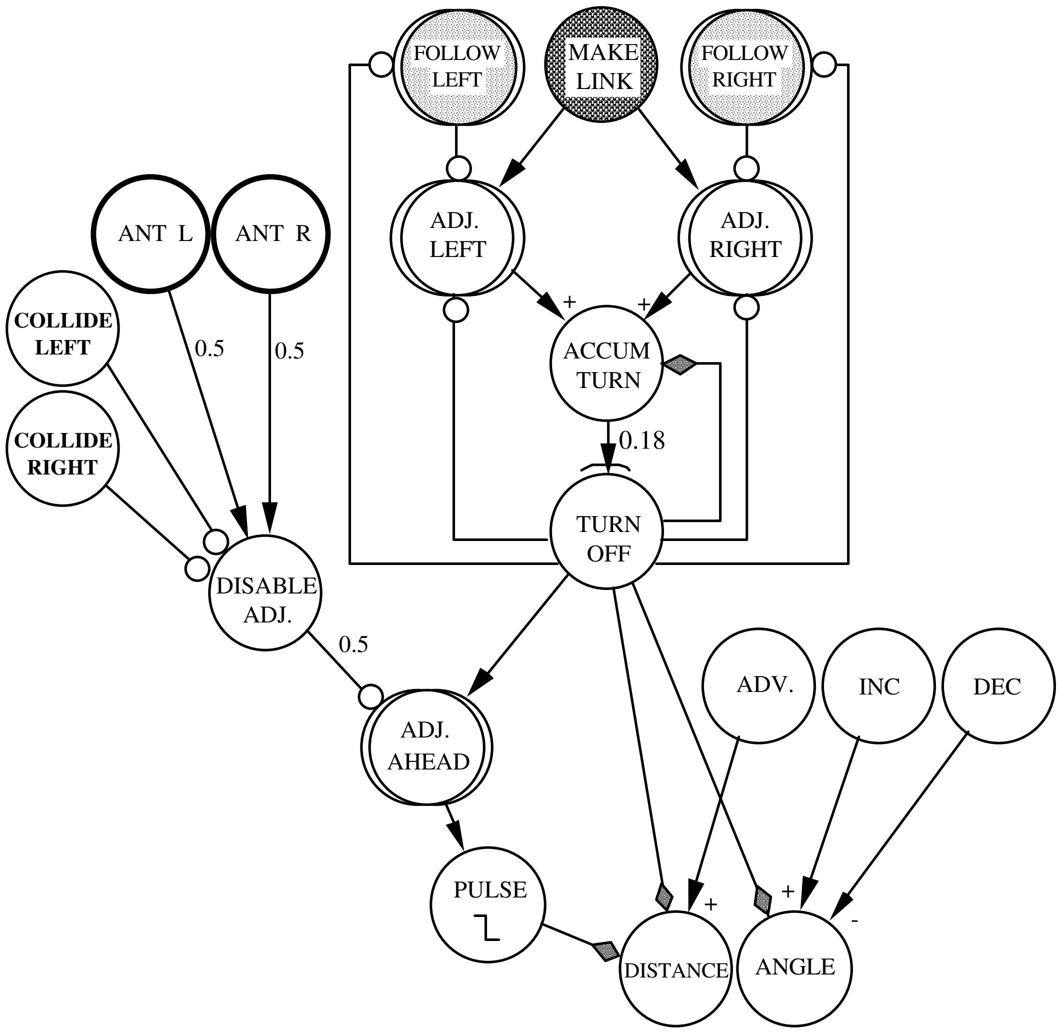


Figure 6.14 The neural circuit for the adjacency link mechanism.

The MAKE LINK neuron is responsible for instigating the adjacency link creation process. This neuron is excited by the memory system whenever the robot recognizes a sequence of features [41] from an inner obstacle. The signal excites the ADJ. LEFT and ADJ. RIGHT neurons which provide the turning mechanism. Only one of these neurons is excited depending on the side of the obstacle that the robot was following. These neurons connect to the TURN LEFT and TURN RIGHT neurons as with the instinctive behaviors with a weight equal to the weight of the follow edge behavior neurons. The ACCUM TURN neuron counts the number of turns made and the TURN OFF neuron emits a high signal whenever the robot makes 6 turns (i.e. 0.18 x 6 ♠ 1) representing 90 .

Once the robot has finished turning, the TURN OFF neuron inhibits the follow edge neurons as well as the turning neurons and excites the ADJ. AHEAD neuron. The ADJ. AHEAD neuron connects to the turning network similar to the EDGE AHEAD neuron. Once this neuron is excited, the robot continues walking straight. It is disabled whenever one of the robot' s side antennae touches an obstacle. The COLLIDE RIGHT and COLLIDE LEFT neurons are from the edge follow network. They make sure that the robot turns parallel to the detected edge before disabling the adjacency behavior [42].

The DISTANCE and ANGLE neurons are accumulative neurons that are used to measure the length of the adjacency link and the angle that the link makes with the ending edge. They are reset when the robot begins forward motion. The memory system monitors the ADJ. AHEAD neuron and stores the distance and angle information whenever it is turned off. When turned off, the DISTANCE neuron is reset again so that it may be used to measure the distance along the ending edge for partitioning purposes.

Once the adjacency behavior is disabled (when the robot reaches the outer obstacle), then the edge following behavior "kicks in" allowing the robot to trace a small portion of the obstacle so that the edge can be identified. Once the edge is identified, the link is created using the method previously mentioned.

Figure 6.15 shows an example of the path that the robot needs to follow (darkened) in order to make an adjacency link from the top edge of the inner obstacle to the top edge of the outer border. The robot is assumed to have already created a mapping of the obstacle borders. In this example, the robot started to follow the edges of the inner obstacle in a clockwise

[41] This occurs when the short term memory becomes full.

[42] The robot is assumed to be parallel provided that a side antenna has contact and the frontal antennae do not.

direction at S. Once it had traced out 4 complete edges (i.e. a full STM), the robot recognizes the features as being part of an inner obstacle. Then at point A, the robot begins to create the link by turning perpendicular to the current edge and walks straight ahead until colliding. The robot then turns right until it is parallel to the new edge. At this point, the robot has stored the distance of the adjacency link and the corner angles. The robot then continues to follow the new border in a clockwise direction until the features are recognized at point E. If the features are recognized, then a new adjacency link is created as described previously. If no features are recognized then the adjacency link is abandoned.

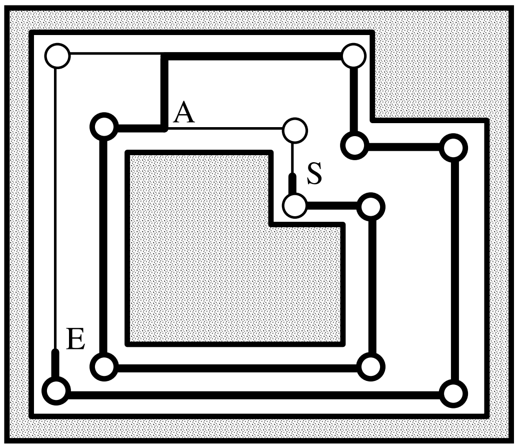


Figure 6.15 Path traveled during the creation of an adjacency link. The robot begins tracing the obstacle at S, starts the creation of the adjacency link at A, and then completes the link at point E.

6.5 Point to Point Navigation

Any animal that builds an internal map of its environment must use it for some purpose. The map gives the animal a sense of where it is with respect to other locations in the environment. Internal maps are usually used for point to point navigation. That is, an animal can use the map to navigate from one location to another. The use of such structured path planning can result in a quicker and more efficient animal. It could allow an animal to remember locations of food and navigate back to these locations when it is hungry. Similarly, a robot could remember locations of wall outlets and navigate back to them in order to receive a battery recharge.

6.5.1 Self Location

The first step in navigating the environment is to determine the robot' s current location. To do this, the robot must compare nearby features with an internal map to find a match. As already discussed, this matching task can pose problems due to discrepancies in feature characteristics between sensor readings and the internal map.

Assuming that the robot has a fairly accurate map, the robot could attempt to make a match whenever it encounters a corner feature. Since there may be many similar corners in the environment, the matching process may yield a collection of matching corners. Thus, the robot may be unable to determine an exact location by examining just one corner. A better method of matching is to trace out a small portion of the environment, obtaining a group of sequential features. RABI does this by filling up its short term memory as it travels along a border. Once the memory becomes full (i.e. 8 neurons = 4 corners and 4 edges), then this sequence of features is compared with the features in the long term memory to determine a match. Again, the sequence may match more that once in the memory, but the chance of duplicate matches is reduced. Assuming that the environment is sufficiently complex with few similarities in the sequential features, then an exact match would likely be found. Once a match has been made, the robot knows its present location with respect to the other edges and corners of the environment.

Lets assume that the robot performs the matching operation which results in a list of plausible locations. By continuing onward tracing the border, the robot would receive additional sequential feature information in the form of neurons from the short term memory. This additional information can be used to trim the list of plausible locations. Consider Table 1, which shows a set of 5 sequences of neurons in memory. All of the sequences begin with a corner neuron having a value of -4. Assuming that the robot has matched up to the -4 neuron, the robot must be at a location pertaining to one of the 5 sequences in the table. Now, by continuing onward along the border, assume that the robot encounters features as follows: 20, 6, 4 -6, 34, 3, 17, 5, 8. Thus, we can see that the first sequence represents the path traveled. By encountering the features one by one, the list of plausible locations can be reduced as sequences stop matching. Sequence 5 for example, will stop matching as soon as neuron 20 has been encountered. Thus, sequence 5 is eliminated from the list. Continuing onward, receiving a 6 and then a 4, sequence 3 will stop matching and thus will be eliminated. Sequence 2 and 4 are eliminated in a similar manner. Thus, after encountering the initial -4 feature, the robot must trace out 7 more sequential features before it is able to identify its location since after this additional tracing, sequence 1 will be the only one remaining.

ID	Stored Memory Sequence											
1	···	-4	20	6	4	-6	34	3	17	5	8	···
2	···	-4	20	6	4	-6	22	3	30	-6	17	···
3	···	-4	20	6	11	-3	21	4	34	8	30	···
4	···	-4	20	6	4	-6	34	3	22	-6	3	···
5	···	-4	5	6	34	4	4	3	11	-12	5	···

Table 1 Possible matching memory sequences.

Assuming that the environment is sufficiently complex (without symmetries and many similar edge lengths), the robot should always be able to identify its location using this method of sequential feature extraction. Once the robot knows its position, it simply needs to compare sequential features to ensure that its location is valid. As it moves along a border, the current location will also move adjacently in the memory. Thus, provided that the robot has an accurate mapping, the position can be determined while the robot moves along the borders.

6.5.2 Determining a Path

When traveling from point to point in the environment, the robot should choose the most efficient path such that it does not waste time and energy. The shortest path from one location to another depends mostly upon the distance traveled. Perhaps in some cases, the shortest path should take into account the number of turns made, since for walking robots, the turning process can extract additional energy that would not have been used up during straight motion. For simplicity sake, RABI examines only the Euclidean distance; the additional turning constraints are easily added.

With neurons, the simplest way of determining a shortest path is to use a form of spreading activation [43]. That is, by exciting a neuron in memory, a chain of activation spreads outwards from its adjacent neurons. This activation can be increased or decreased as it passes

[43] [Mataric 91] uses spreading activation in his robot to determine the shortest path between landmarks.

through a neuron. An edge neuron for example can add activation corresponding to its stored energy (i.e. the edge length).

In order for this to happen, each memory neuron must have an adjustable output (set by the spreading activation) as well as a stored energy value (representing the corner angle or edge length). The stored energy value remains constant while the output changes according to the spreading activation.

Determining a shortest path from one point to another is a trivial problem. The problem becomes more interesting as multiple destinations are possible. A robot, for example, may choose between a dozen wall outlets to receive energy from. It would be most efficient if the robot were able to choose the closest outlet. This problem of selecting the closest destination point is easily handled with the spreading activation concept. Essentially, the activation spreads outwards from each destination.

The activation algorithm consists of a handful of simple functions which are given in Appendix A. The algorithm works by initially giving the destination neurons an output of 1 and all other neurons an output of zero. Activation then spreads outwards from the destination neurons in such a way that any neuron that receives an activation less than its current output uses this smaller activation value as its output. When an edge neuron receives an activation, it adds its stored energy and spreads the new activation to adjacent neurons. Depending on the direction that the activation is passed (i.e. forward or backwards in the memory), the sign of the activation is altered. Activation going backwards through the memory has a negative sign, while activation going forward has a positive sign.

Once the activation stops, each neuron will have an output corresponding to the activation that has passed through it. Since the edge neurons added activation pertaining to an edge length, the output of each neuron corresponds to the Euclidean distance required to get from one neuron to the other during navigation. Moreover, the sign of the output indicates the direction (left or right) that the robot must travel in order to get to the nearest destination.

Figure 6.16 depicts an example showing the effects of selecting 1, 2 and 4 possible destination points. The destination neurons are shown with blackened boxes around them. The number above each neuron represents the output of the neuron after the activation process has completed. Note that the top three neurons represent bypass neurons that represent another possible path in the memory.

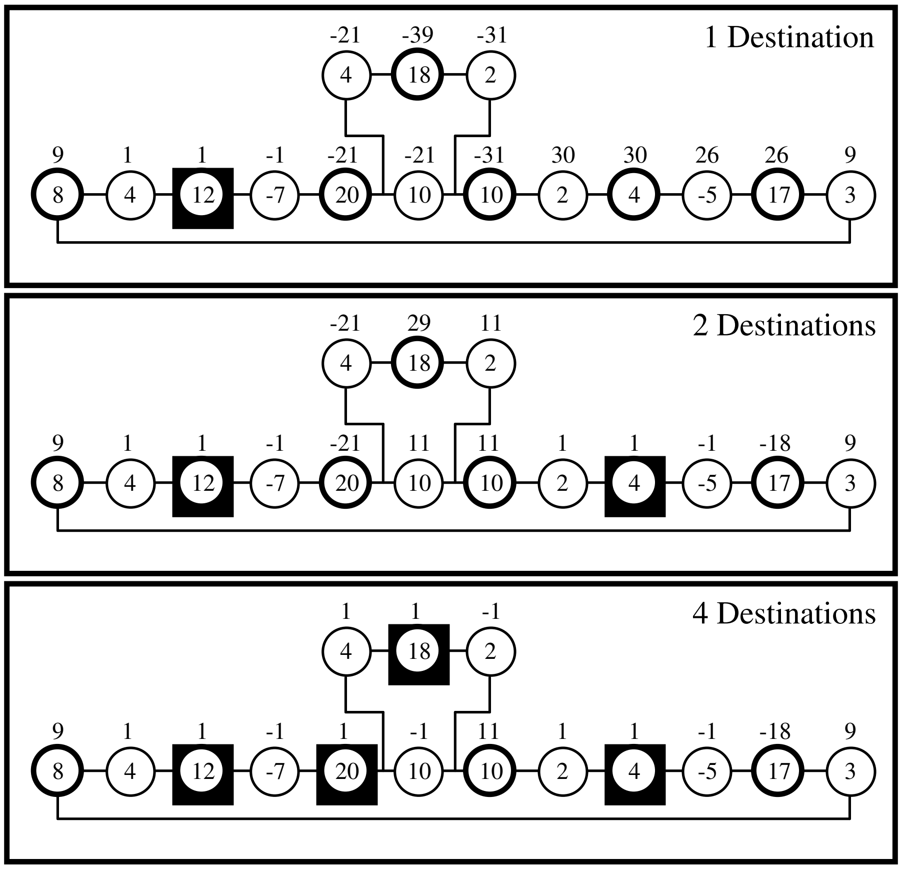


Figure 6.16 Results from spreading activation for 1, 2 and 4 destination points.

In the diagram, each of the destination neurons is an edge neuron with an output of 1 indicating that they are indeed destinations. For each corner neuron in the diagram, their output indicates the distance (in terms of edge length sums) and direction required to travel from that location to the nearest destination edge. Thus as soon as the current location is identified, the robot can use this activation information to choose a direction in which to travel which will result in the shortest path to the destination. This method of point to point navigation is simple to implement and provides an efficient path to the desired destination.

This method is also able to incorporate the adjacency links during the spreading activation process since the neurons are part of the same memory. The activation through an adjacency corner or adjacency edge neuron is similar to the regular corner and edge neurons. Since this activation process results in an efficient path plan, the robot can easily travel from obstacle to obstacle using the adjacency links.

6.5.3 Selecting the Destination Points

More often than not, animals have many choices as to where they can obtain food. When this is so, the animal must choose only one location at a time in which to eat. The selection of destination may depend upon other factors such as the amount of food, quantity of food, variety of food or other external factors such as danger zones which make the food risky to obtain. A robot may be in a similar predicament. These complexities require a weight to be placed upon each location indicating the strength of the tendency to travel to that location. This weighted selection strategy was not implemented since it is more complicated and the objective was to keep the robot as simple as possible.

When in need of energy , the robot must prepare for navigation by instigating spreading activation from all sources of food. This would allow the robot to determine which direction to travel to reach an energy source. There is a need for some type of mechanism that maps energy sources to environmental locations. This can be done by adding special links that activate memory locations that contain the desired source. The mechanism required to do this consists of just two neurons as shown in Figure 6.17.

The SEEK ENERGY neuron is used to excite the RESET and LOCATE neurons whenever the robot is searching for food. The RESET neuron provides a special signal to the memory unit that resets all of the memory neurons to have zero activation whenever the robot stops seeking energy. This gets the memory set up for the next time by resetting any activation from a previous navigation. The LOCATE neuron emits a special signal through its output links whenever the robot begins to seek energy. This special signal initiates spreading activation to each memory neuron that is connected to the LOCATE neuron. Once the activation has been spread, the memory is set up for navigation purposes and the memory neurons will retain their output until the robot no longer seeks energy.

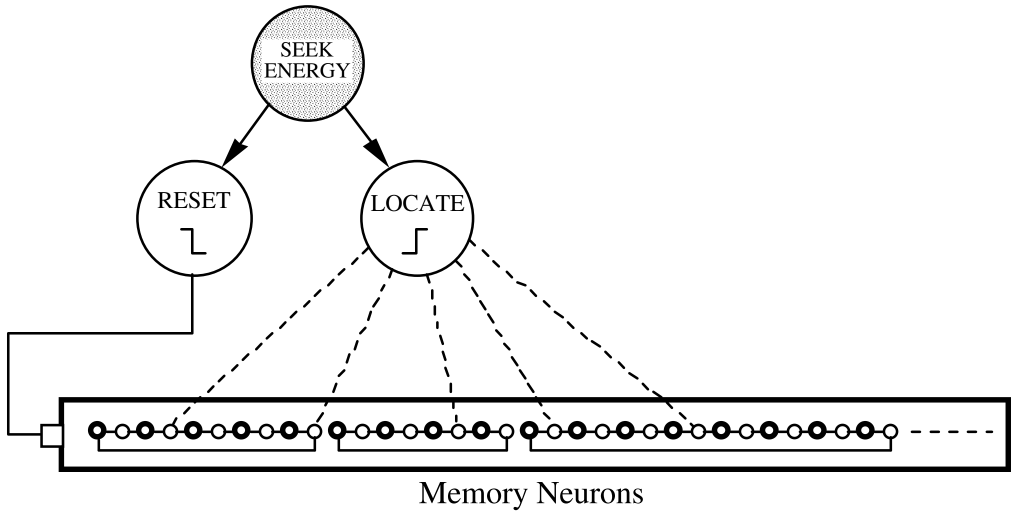


Figure 6.17 Connecting neurons that control the activation process.

Initially, there are none of these special links connected to the memory neurons. The links are added as the robot encounters locations that induce an energy signature. That is, whenever the robot senses an energy source when traveling along an edge, it adds a link from the LOCATE neuron to the memory neuron representing the current edge. This linking strategy represents a form of associative learning in which energy sources are paired with environmental locations.

Assuming that each food (or energy) location has equal weight, there may still be other problems related to conflicting motivations. A robot may find energy at one location, work at another and other items at yet another. In this case, the robot must choose which item to seek out and spread activation corresponding only to this type of item. The spreading activation algorithm will not work when destinations of different types are allowed to interfere with the activation of other types. Thus, each time the robot decides to seek a specific type of item, the memory must be reset so that previous activation data does not interfere with the new activation information. This can be done by adding inhibitory links between the neurons that represent different destination types as shown in Figure 6.18.

In the diagram, the neurons 1, 2 and 3 represent different types of item seeking behaviors similar to the SEEK ENERGY neuron. The R and L neurons correspond to the RESET and LOCATE neurons. Each of the LOCATE neurons connects independently to the memory neurons as before. Sometimes the links may actually share a memory neuron which indicates that there are multiple items at one location. The circuit works the same with the exception that the RESET neurons inhibit the LOCATE neurons of other types. This ensures that the memory is reset before the spreading activation occurs. RABI is only able to associate energy sources with locations and thus does not use these extra inhibitory links.

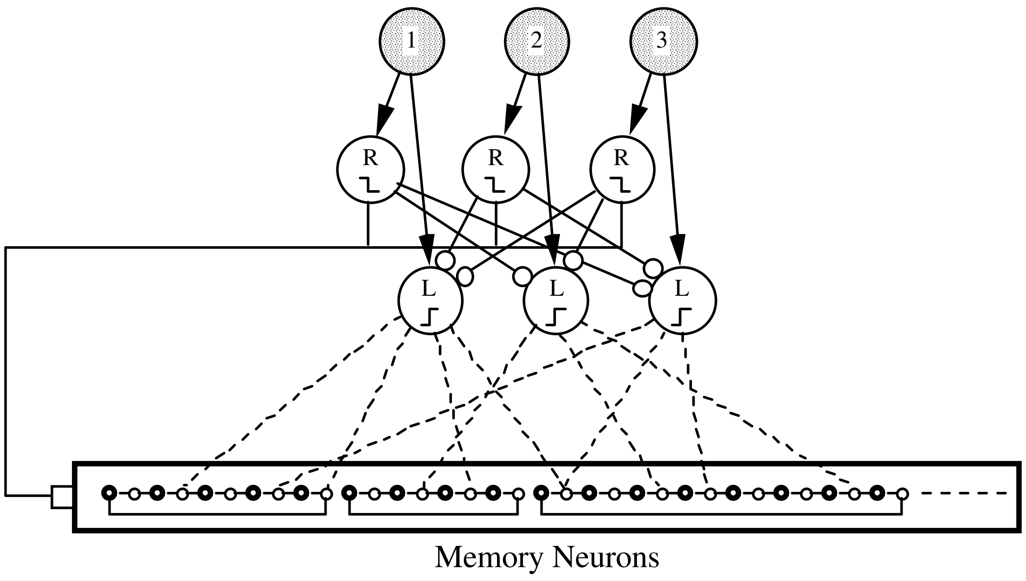


Figure 6.18 Additional neurons needed to handle conflicting destination types.

6.5.4 Turning Around

When the robot is navigating in an environment by following edges, it may need to turn around and follow the edge with the other side of its body. This would occur after the robot has determined that it is going the wrong way during point to point navigation. The task of turning around is simple. It involves turning 180 and switching from following left to following right or vice versa. The circuit of Figure 6.19 accomplishes this.

The GO RIGHT and GO LEFT neurons connect to the output of the memory system. They are excited when the robot decides to go right or left during navigation. The AROUND RIGHT and AROUND LEFT neurons are excited whenever the robot is going in the wrong direction. They connect directly to the TURN LEFT and TURN RIGHT neurons with a weight of 6.4 so that they override all but the edge follow and energy seeking behaviors. The remaining neurons are similar to those of the adjacency mechanism. They are responsible for counting 12 turns (180) and then disabling the turning process and disabling the edge following. The edge following will start up in the correct direction at the next time step.

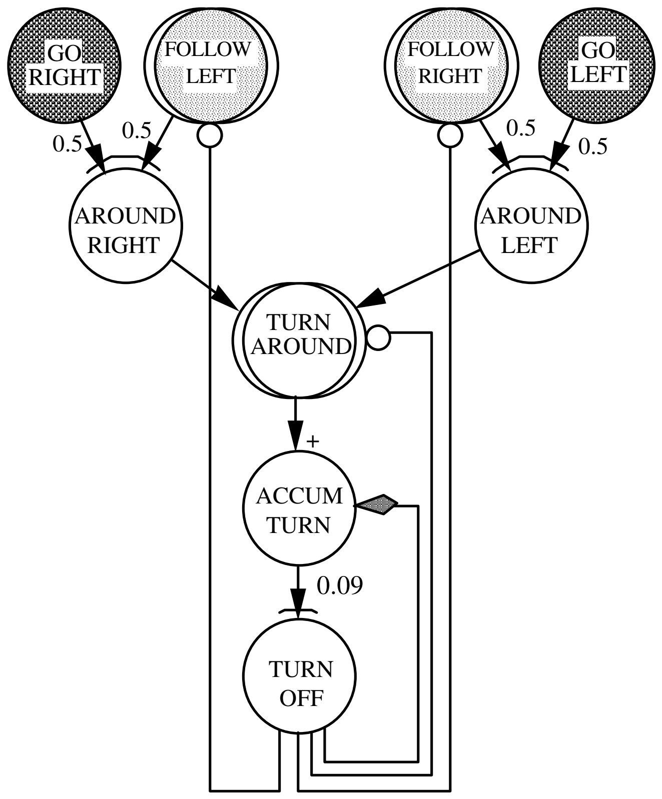


Figure 6.19 Neural circuit for turning around.

6.5.5 Using an Adjacency Link

Once an adjacency link has been created, it is stored permanently in memory. In order to use this link path during navigation, the robot must be able to extract the distance and angle information so that it may leave one obstacle and travel directly towards another. Thanks to the spreading activation process, the required information is embedded within the memory neurons.

Whenever the robot is navigating along an obstacle border it uses the spreading activation results to determine which direction to follow along the obstacle. A memory neuron' s output is either negative or positive indicating a desired left or right traversal. Since the output can be only positive or negative, there is no way of indicating a third direction (outwards away from the obstacle) which is required for adjacency link traversal. Therefore, the navigation system must handle adjacency links separately. The navigation system must constantly look ahead to determine if it needs to stop following the edges of one obstacle and start following the edges of another.

Figure 6.20 shows an example of a simple environmental mapping and the results of spreading activation across an adjacency link. Note that the sign of the output of the adjacency corner neurons does not matter since these corners are handled specially.

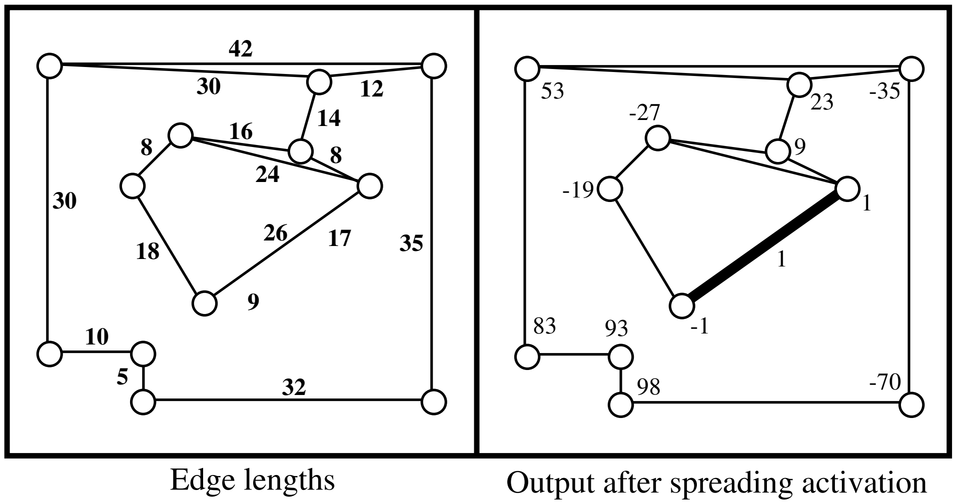


Figure 6.20 Example showing the output of neurons after activation is spread across an adjacency link. The darkened line in the second diagram represents the point at which the activation is initiated (i.e. an energy source lies close to this edge).

By examining the top right and top left corners of the outer border, it is clear that their output differs in sign. If the robot did not have a mechanism that allowed it to travel across an adjacency link, then it robot would pace back and forth along the top edge of the environment due to the change in sign of the edge corners.

By adding one more simple neural circuit, the robot could shoot out into the appropriate direction along the adjacency link so that it can reach the inner obstacle. This simple circuit is shown in Figure 6.21.

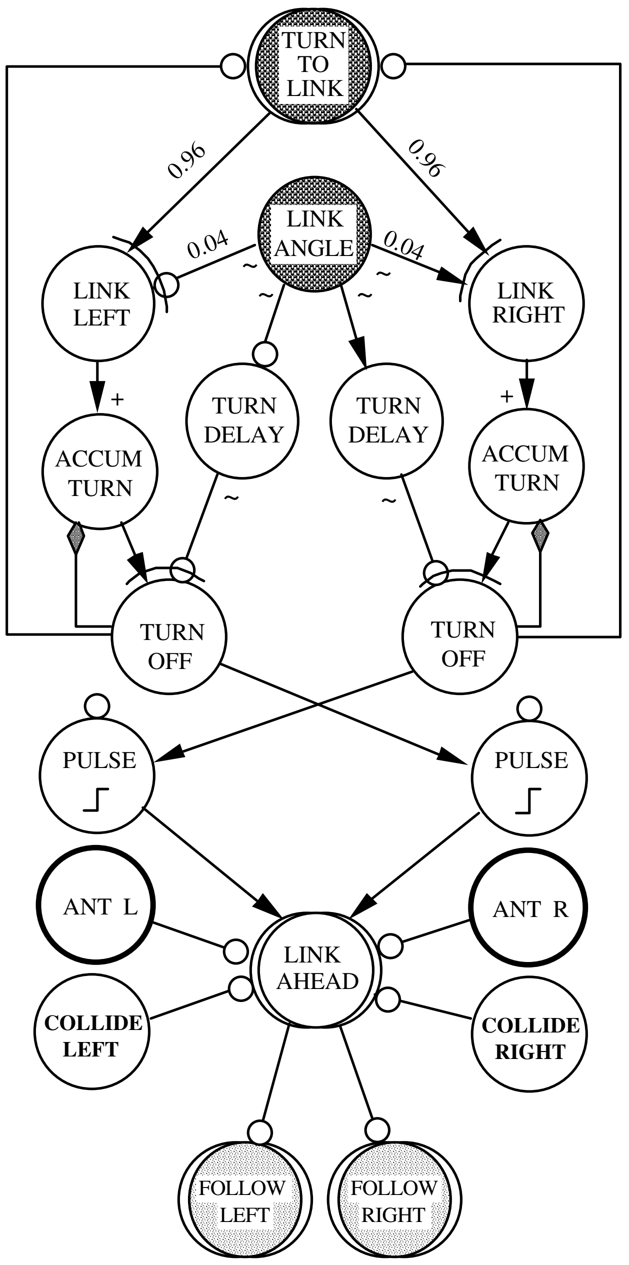


Figure 6.21 A neural circuit for using the adjacency links.

The TURN TO LINK neuron is excited by the navigational system whenever the robot decides to take the path along an adjacency link. This would occur whenever the robot has reached the adjacency point. The LINK ANGLE neuron is a standard neuron which is set by the navigation system and has a value equal to the number of turns required to align the robot along the adjacency link. This number is derived directly from the stored angles of the adjacency corner neurons.

Once these two neurons are excited, one of either the LINK LEFT or LINK RIGHT neurons is excited depending on the sign of the output of the LINK ANGLE neuron. The threshold values are 0.96 and 0.04 which represent values of approximately 23/24 and 1/24. These values ensure that both neurons need to be excited in order to enable the turning process [44]. The LINK LEFT, LINK RIGHT and LINK AHEAD neurons all connect to the TURN LEFT and TURN RIGHT neurons in a similar fashion to the network for adjacency link creation. The ACCUM TURN and TURN OFF neurons are responsible for counting the appropriate number of turns. Once the appropriate number of turns are made, the turning is disabled and the LINK AHEAD neuron is excited, subsuming the follow edge behavior. The behavior is disabled whenever the robot comes in contact with the new obstacle. Once the robot reaches the new obstacle, it continues to navigate from its current location towards the desired location.

This method of traveling among adjacency links does not necessarily produce shortest paths since the adjacency links act as bridges in which the robot may cross at certain locations. Over time however, as the robot explores its territory, multiple adjacency links will eventually be created and the robot will become more efficient.

6.6 SUMMARY

Internal maps can store a great deal of spatial, topological and associative data which can prove to be advantageous to an adaptive robot. The neuron style memory employed by RABI allows efficient navigation through the use of spreading activation. Moreover, this navigation technique is easily interfaced with the neural circuitry of the instinctive behaviors. Due to the non-spatial nature of the landmark-based mapping technique, various problems occur when attempting to identify the robot' s current location. Provided that the environment is sufficiently complex, the landmark identification process of matching sequences of neurons is a sufficient method for determining the robot' s location. With the navigational technique used by RABI, the robot is able to identify various edges and corners in the environment. By using a simple form of spatial adjacency, the robot is able to efficiently travel from landmark to landmark.

[44] The value of 24 represents 24 turns (24 x 15 = 360). The LINK ANGLE neuron will never have a value above 24 or less than -24.

~~~~~~

Chapter 7
Motivation and Behavior Selection

~~~~~~

Many of the instincts presented in chapter 4 can be combined to produce more complicated emergent behaviors. Sometimes however, there are certain behaviors that cannot be performed at the same time. Phototropic and photophobic behaviors for example, are conflicting behaviors that cannot be performed simultaneously. A less obvious example may be the energy seeking and map building behaviors. A robot may search for energy while building a map, but if the energy is in the center of the environment, the map building behavior may prevent the robot from attaining it.

It is clear that some kind of behavior switching technique must be incorporated into the robot such that conflicting behaviors are not active at any one time. Furthermore, the behaviors must be selected in such a fashion so as to keep the robot functioning ("alive") at all times. This area of selecting behaviors at appropriate times is strongly related to the area of *motivational systems* . This chapter discusses the notions of motivated behavior and behavior selection, and also describes the mechanisms required to implement them in RABI.

7.1 Motivated Behavior

What causes an animal to behave the way it does ? This question brings up the notion of *motivated behavior*. [Toates 86] defines motivation as follows:

"Motivation is the strength of the tendency to engage in behavior when taking into account not only internal factors but also appropriate external factors."

The statement suggests that motivation arises as a function of both internal state (or drive) and external incentives. This would mean that an animal' s behavior will change as its internal state changes or if external cues are presented. This is partially intuitive since many animals will only engage in foraging behavior when their internal energy state decreases (when they are hungry). Furthermore, this foraging behavior becomes more apparent when external energy sources are present (near food).

[Toates 86] states that although reflexes constitute a crucial part of behavior, they clearly cannot account for all aspects. The consequences of behavior are such as to change the internal state underlying motivation and hence reduce the neural activity of the motivation circuits. It would seem that certain behaviors are selected and performed depending on some motivational aspect whether internal or external.

7.1.1 Homeostatic and Externally Driven Behaviors

Homeostasis is the term used to refer to the ability of the body to maintain its essential parameters near constant and to take corrective action to return them to normal following a disturbance. As mentioned in [Toates 86], some feel that homeostasis is a key element of motivational systems. An example of homeostatic behaviors would be the eating and drinking behaviors associated with a decrease in an internal state such as energy depletion and dehydration.

Not all behaviors are directed towards maintaining an internal state. Other behaviors such as sex, exploration, aggression, etc., depend largely upon external stimuli such as sexual partners , environmental structures or intruders. In some cases, it is not always obvious as to what the external cues are; such as when an animal just gets up and goes. Exploration is an example of a behavior which is not accentuated by internal state or external stimuli. Rather it is an external stimulus in the context of the animal' s expectations about the environment that causes exploratory behavior. New environments however, do present a form of external stimuli.

7.1.2 The Role of Past Experiences

Many experiments have been performed to show that animals can be conditioned to respond to certain types of stimuli with a certain type of behavior. This conditioning may take the form of reinforcement learning where an animals response to a specific stimuli is strengthened. For example, if a particular response (turning left in a maze) is followed by a particular event (presentation of food), then the frequency with which the animal turns left increases. In this example, the food reinforces the response. This example shows that past experiences play an important role in motivated behavior.

7.2 Behavior Selection

At any one time, an animal is performing a behavior according to some form of motivational selection. There are lower level reflexes and mechanisms that are performed as part of a higher level behavior. A foraging behavior for example, may be composed of a set of lower level reflexes or mechanisms such as walking, avoiding collisions, seeking light, etc. An animal must have some method of selecting which mechanisms to enable or disable for any given behavior. In addition, there may be conflicting motivations in which two behaviors compete for overall control of the animal' s muscles or actuators. For robots, this problem of conflicting behaviors must be handled such that only one overall motivated behavior can be active at any one time.

It is clear from observations that a change in an animal' s external environment may cause a change in behavior. But animals also change behavior even in the absence of external cues. Hence, it seems that the process of behavior selection should depend upon both the external stimuli as well as internal state.

[Maes 91] presents a bottom-up mechanism for behavior selection in an artificial creature. In her simulation, the creature is endowed with various behaviors pertaining to obstacle avoidance, exploration, fighting, fleeing, eating, sleeping and drinking. Each of these behaviors has a corresponding motivation associated with it. This motivation is in the form of a monitor which keeps a strength value pertaining to the desire to satisfy the motivation. In her system, the highest strength motivation is selected and the corresponding behavior is performed. The system also contains a set in links that allows the creature to enable and disable certain behavioral mechanisms. If hungry, for example, the creature would explore until it sees food, then go towards the food, and finally eat the food. Some of these links are innate while others are learnt. Her system for behavior selection is robust, efficient, reactive and flexible. The system is also situation oriented and exhibits opportunism.

[Tyrrell and Mayhew 91] also present a simulation for behavior selection. They concentrated on creating a significantly complex environmental model so as investigate the mechanisms behind behavior selection. They discuss certain issues of opportunism, conflicting behaviors and cooperative behaviors.

While both of these simulations appear successful in mimicking the behavior of a possible biological creature, they deal with issues that will not be necessary for robotic purposes. Drinking, fighting, sex and fatigue for instance, are not issues that need to be dealt with in robotic systems. Thus, to some extent, the simulations have a degree of overkill. Their purpose was to investigate behavior switching strategies with multiple conflicting motivated behaviors. Their research may prove useful for robots that must be able to efficiently select among several behaviors, although simplified robots such as nanobots will more than likely contain hardwired instincts. If this is the case, then it may be simpler and more efficient to hardwire a behavior selection strategy within the hardware.

7.3 RABI's Motivation and Behavior Selection

For a robot that is "artificially alive" it must be highly motivated to keep itself functioning. This motivation is responsible for finding energy sources. Since RABI was developed to concentrate on the survival aspects, there are only 3 types of motivation: obtaining energy, exploration and the motivation to work. As the robot is given additional instinctive behaviors (other tasks to perform), then there will be motivation to exhibit other types of behavior.

Since RABI does not get tired, it should always be moving unless it is re-charging. Thus, every behavior should keep the robot walking. To do this, the motivated behaviors excite the WALK neuron as in Figure 7.1.

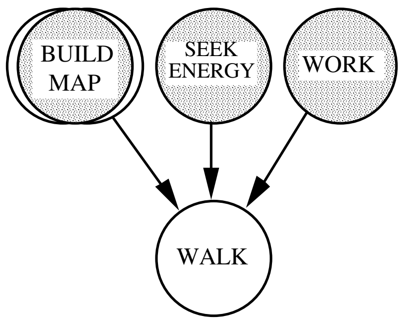


Figure 7.1 Excitation of the WALK neuron by the motivated behaviors.

The simplest method of handling multiple motivated behaviors is to allow only one to be selected at a time. Therefore, the robot must decide which motivated behavior to pursue at any one time. There is a need to determine when a behavior should be enabled or disabled and which behaviors are more important than others.

7.3.1 Obtaining Energy

The energy seeking behavior of RABI is the most important behavior since it allows the robot to seek out energy sources which are crucial for survival. If the robot were to spend all of its time seeking out energy however, then the robot would be of no use for any practical application. There should be some type of mechanism that allows this behavior to be selected at the appropriate time; when the robot' s energy becomes too low. The motivation for the energy seeking behavior should depend on both the internal energy level and the external cues such as the presence of energy sources. In order for this to happen, the robot must have some form of energy monitor indicating the current level of energy. When this energy lowers to some threshold value, the robot should then begin to search for energy sources. This process is easily implemented with neural circuitry as shown in Figure 7.2.

The ENERGY MONITOR is a monitor neuron which emits an output signal (0.0 to 1.0) reflecting the current energy level of the robot. The neuron excites the HIGH neuron with a weight of T4. This value of T4 represents the threshold at which the robot is considered to have adequate energy (i.e. a battery low indicator) [45]. The signals from the HIGH, ENABLE and DETECT SOCKET neurons are responsible for instigating the ingestive process. The DETECT SOCKET neuron is a sensor neuron that ensures the robot is at an energy source in order for the ingestion to take place. Once the START neuron is excited, the ingestion begins. During ingestion, the WALK, TURN LEFT and TURN RIGHT neurons are disabled (i.e. remain still while charging) and the ABSORB motor neuron is excited.

[45] T4 ranges from 1 to •. If for example, T4 was set at 4, then an energy value of 0.25 or more would be required to excite the HIGH neuron. Thus, the low indicator would be at 25%.

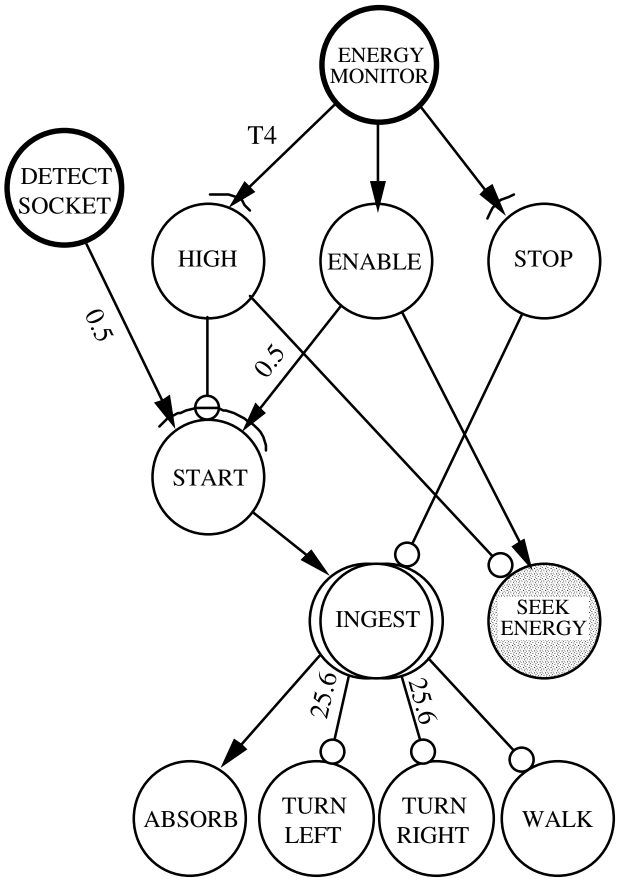


Figure 7.2 Neural circuitry representing the ingestion mechanism. The energy seeking behavior is enabled when the energy monitor reads a significantly low value.

The ABSORB neuron represents some form of physical mechanism responsible for increasing the robot' s energy. This may be a simple plug-in actuator or perhaps something more complicated. As in the case of docking situations discussed in chapter 4, this motor may not be needed at all since the energy refill may be performed by some external mechanism.

Once the energy monitor detects a full energy reading, the STOP neuron is sufficiently excited so as to disable the ingestion process. This neural mechanism is linked to the energy seeking behavior by connecting it to the SEEK ENERGY neuron from the energy seeking network in a manner such that the behavior is enabled as soon as the energy level becomes too low.

7.3.2 Seeking Energy Vs. Map Building

Since the robot can only detect energy sources when they are close by, the robot may wander around the environment for a long time without finding an energy source. It would be advantageous for the robot to explore the environment to search for energy sources. These energy sources may lie along walls and thus, it would be useful to build a map of the obstacle borders. This map building should take place before the robot becomes energy deficient so that it has time to locate the energy sources. Thus, when not "hungry", the robot should engage in exploratory map-building behavior.

The robot must not however, spend all of its time seeking energy and building maps since it was also given a task to work by cleaning up the environment. The map building behavior should therefore be both enabled and disabled during the robot' s life time. In fact, animals spend much of their time exhibiting exploratory behavior [Toates 86]. Moreover, the exploratory behavior is highly active when the animal is placed down into a new environment. It would be prudent to devise such a scheme for robots.

RABI uses a curiosity monitor that essentially indicates the amount of exploration that should be performed. The curiosity is initially very high when the robot starts out and decreases very slowly over time. Whenever the robot traces out a portion of the environment that is not recognized, this curiosity is increased. When the robot recognizes certain environmental features, the curiosity is decreased [46]. Figure 7.3 shows the connections required to select one of the three motivated behaviors. Notice that the cleaning behavior has no excitatory signals. That' s because it is always enabled. Hence, whenever the robot is not seeking energy or building a map, then it is cleaning up the environment.

[46] The monitor value ranges from 0.0 to 1.0. The time decrease is approximately 0.001. The increase when the features are not recognized is 0.1 . When features are recognized, the decrease is 0.1 also.

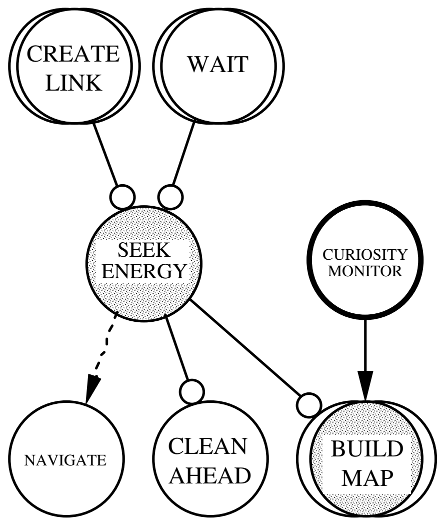


Figure 7.3 Neural circuit for selecting the motivated behaviors.

In the network, the map building behavior is enabled whenever there is any amount of curiosity. This behavior is dominated by the energy seeking behavior which is selected according to the energy monitor as mentioned previously. Both of these behaviors override the cleaning behavior since they are more important for survival. There is no need for the map building neuron to inhibit the clean ahead neuron since the robot is always following edges when building a map and the edge following behavior already overrides the cleaning behavior. The energy seeking behavior is disabled by the CREATE LINK and WAIT neurons. This ensures that the robot has completed an adjacency link or obstacle trace before turning off the map building behavior. This is important so that the robot does not leave behind its disk (along the edge of an obstacle being traced) and wander off; otherwise the disk could not be found easily.

An excitatory link is shown from the SEEK ENERGY neuron to the NAVIGATE neuron. This link is initially absent and is permanently added if and when the robot finds an energy source near an obstacle edge (as opposed to the center of a vacant region). The link allows the robot to use navigation to find an energy source.

When initially placed in the environment, the robot spends most of its time tracing out obstacles and building a map. It occasionally finds dirt morsels and brings them to an obstacle border. Eventually the robot begins to recognize the obstacles and the curiosity decreases significantly. Once the curiosity diminishes, the robot begins performing its cleaning task, occasionally taking a break to obtain energy. With so few instinctive behaviors, the robot' s overall behavior is somewhat predictable. As more and more instinctive behaviors are added (more tasks) then the global behavior would be more difficult to predict.

7.3.3 Enabling and Disabling the Edge Following Behavior

Since the robot should not always follow edges (i.e. there may be interesting things in the middle of the environment), there must be some method of enabling and disabling the edge following behavior. The simplest method of instigating the behavior is to begin edge following whenever a side antenna comes in contact with an obstacle. The robot could then trace out the obstacle' s edges and turn away once it has completely traced the obstacle or when it has motivation to perform some other behavior. Figure 7.4 shows the connections responsible for enabling the edge following process.

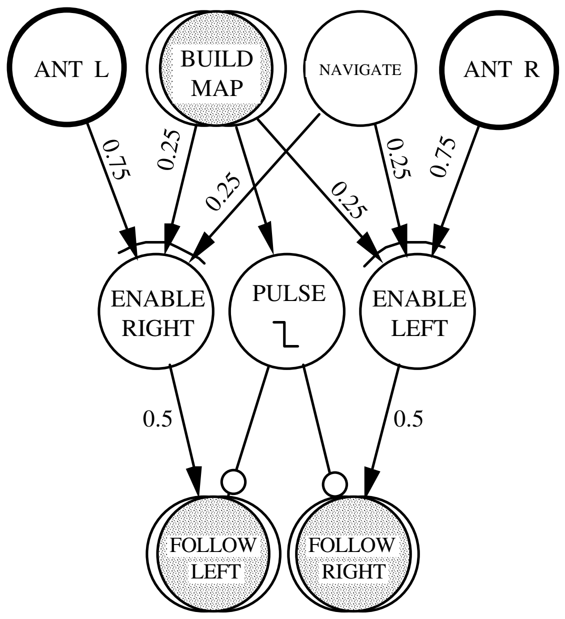


Figure 7.4 Neural connections responsible for the enabling of the edge following process.

The side antennae provide a strong excitatory signal that enables the appropriate edge following neuron; provided that the robot is building a map or navigating. A pulse neuron is used to disable edge following whenever the map-building ceases.

When navigating, the robot uses the spreading activation to determine which edge to head towards. It may be the case that the energy source does not lie along the edge, but instead it lies close by. If this is so, the robot should stop following edges and head towards the energy source using its energy seeking network. Figure 7.5 shows the additional circuitry for disabling the edge following whenever the robot is close to an energy source.

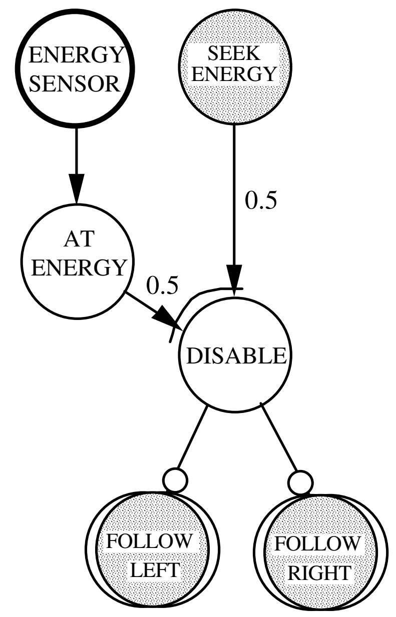


Figure 7.5 Neural circuitry to disable edge following when near an energy source.

The ENERGY SENSOR and SEEK ENERGY neurons from the energy seeking network are used in combination to disable edge following. The AT ENERGY neuron produces the required binary output for the threshold neuron.

The presence of energy is not the only reason to stop following an edge. The robot should stop following an edge when it finishes tracing an obstacle or when it recognizes a portion of an obstacle while exploring. In addition, when the robot is navigating, it should turn away whenever it arrives at a location that has no activation level. That is, if an edge of an obstacle mapping has no activation level, then there is no known path from this current location to an energy source. The robot should therefore turn away from the obstacle.

Turning away from an obstacle is the action performed by the vacancy behavior. This behavior is always active but the edge following behavior subsumes it. If the robot wants to turn away from an obstacle it should therefore disable the edge following behavior. Figure 7.6 shows a neural circuit that does this disabling.

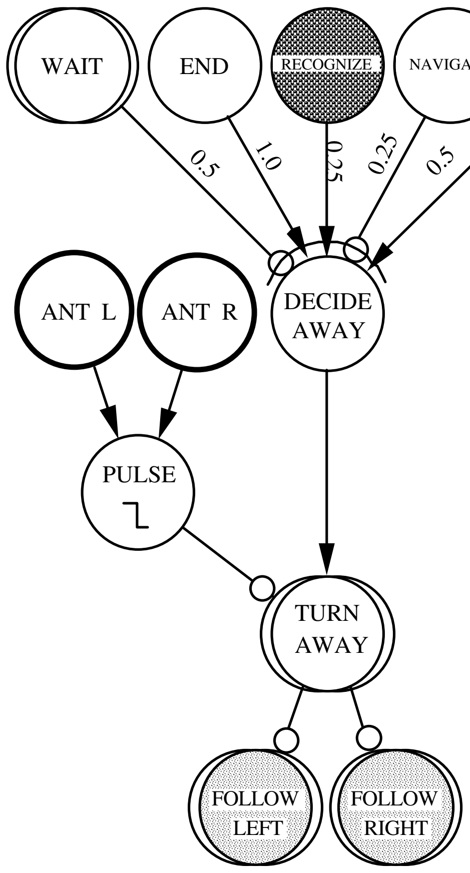


Figure 7.6 A neural circuit to disable the edge following behavior.

The DECIDE AWAY neuron determines when the robot needs to turn away from the edge it is following. The neurons at the top of the network do this as described. The WAIT neuron is also used to inhibit the disabling since the robot should not stop following an edge if it has left a disk behind. With the edge following behavior disabled, the vacancy behavior will take over and turn the robot away from the edge. The side antennae are used to disable this network whenever the robot no longer has contact with an edge.

7.3.4 Instinctive Behavior Selection

Nothing has been mentioned about the photokinetic behaviors. When are these behaviors selected? Ideally, an adaptive robot should be able to learn various associations such as: light leads to energy, dark leads to morsels, loud noises lead to danger, etc. Since RABI has only antennae, energy, light and dirt sensors, then there is not much interesting to be learnt. Essentially the robot could learn only four associations:

LIGHT ---> ENERGY
DARK ---> ENERGY
LIGHT ---> DIRT MORSELS
DARK ---> DIRT MORSELS

Moreover, the associations may be valid at some point in time and invalid later in the robot' s life time. Dirt for instance, may be cleaned up from around a light source when the LIGHT ---> DIRT MORSELS association is used. Once the dirt around the light is cleaned up, the association is no longer valid. Since there are only four associations to learn, the learning process was not incorporated into RABI Instead, they may be selected by hardwiring additional links in the neural networks. Figure 7.7 shows the four links that enable these instinctive associations.

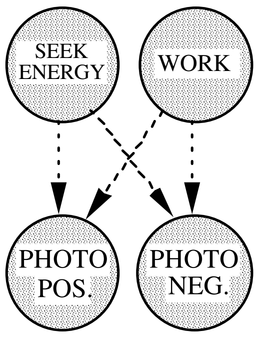


Figure 7.7 Neural links used to associate light with energy and dirt morsels.

Note that only one link should be added from each of the SEEK ENERGY and WORK neurons since the other link represents a conflicting association. With the addition of any of these links, the robot' s wandering behavior is subsumed by the phototaxic behavior allowing the robot to go towards or away from light. Thus, without the mapping and navigational circuitry, the robot could still function in the environment by placing energy sources in light or dark areas and maneuvering the light sources near or away from dirty locations.

7.4 SUMMARY

Motivation and behavior selection are important issues that must be considered when designing autonomous artificial life forms. Care must be taken to ensure that the robot takes precedence in remaining "alive" as opposed to performing some task. Traditional robots concentrate on task performance, allowing the robot to stop functioning at regular intervals. This is not acceptable for colonies of robots that must survive on their own.

RABI has a strong motivation to remain functioning. When its energy level drops too low, it seeks out energy. When it is not seeking energy, it explores the environment in search for energy sources and remembers the locations. Only when it has sufficiently explored the environment, does the robot concentrate on the given task of cleaning. With colonies of nanobots, their given task is spread among hundreds or thousands of others. It is not wise for an individual nanobot to risk its "life" by performing a task when it is low on energy. It would be more efficient if the nanobots could remain functioning for extended periods of time, maybe even working in shifts.

~~~~~~

Chapter 8
Hardware Construction of RABI

~~~~~~

The hardware version of RABI is a 6-legged robot platform allowing various robot control and behavioral techniques to be tested in a real environment. As already mentioned, it was important to create a physical device since these robots must operate in the real world once completed. By basing a robot on only simulated research, the robot ends up being brittle, inefficient, faulty and often fails due to unpredicted situations.

Approximately 6 months of work went into building this physical robot and another 3 months just getting it to simply walk around. RABI has undergone a tremendous amount of redesign and construction, each version improving upon the last. This chapter explains the mechanical changes encountered in each version and points out the problems with each failed design. The electronic control and interface circuitry is also presented along with the communication protocol required to control the robot.

8.1 Frame Construction and Materials

Most of the materials used for construction are made of plastic or aluminum of various sizes which were all hand drilled and cut. Very small nuts and bolts are used to hold these pieces together. Two basic types of wire were used to connect the electronics. The power and ground wire is twisted pair wire from a long microphone cable. Jumpers were made from cut-up computer ribbon cables, and used to interconnect the electronic boards. The robot was built as light as possible by using only plastics and aluminum.

Each leg uses two geared down 3v dc motors allowing 2 dimensional movement. These motors come from Tamiya gear box kits which contain a set of gears to provide enough power to move the legs. The housing and all the gears are plastic providing a light weight gear box. The bulk of the weight is the motor itself and the turning shaft to which the gears are connected, which is made of metal. These motors make up about one third of the robot' s total weight.

The worm gear kit of Figure 8.1 contains a worm gear that allows the turning shaft to lock into place when the motor is at rest. This locking is important so that the weight of the robot does not cause the motor shaft to turn. The gear set provides a 336:1 gear ratio providing enough power to lift the robot' s legs. There is however, a considerable amount of slack in the gears hindering precise position measurements. Nevertheless, the gear box provides enough strength.

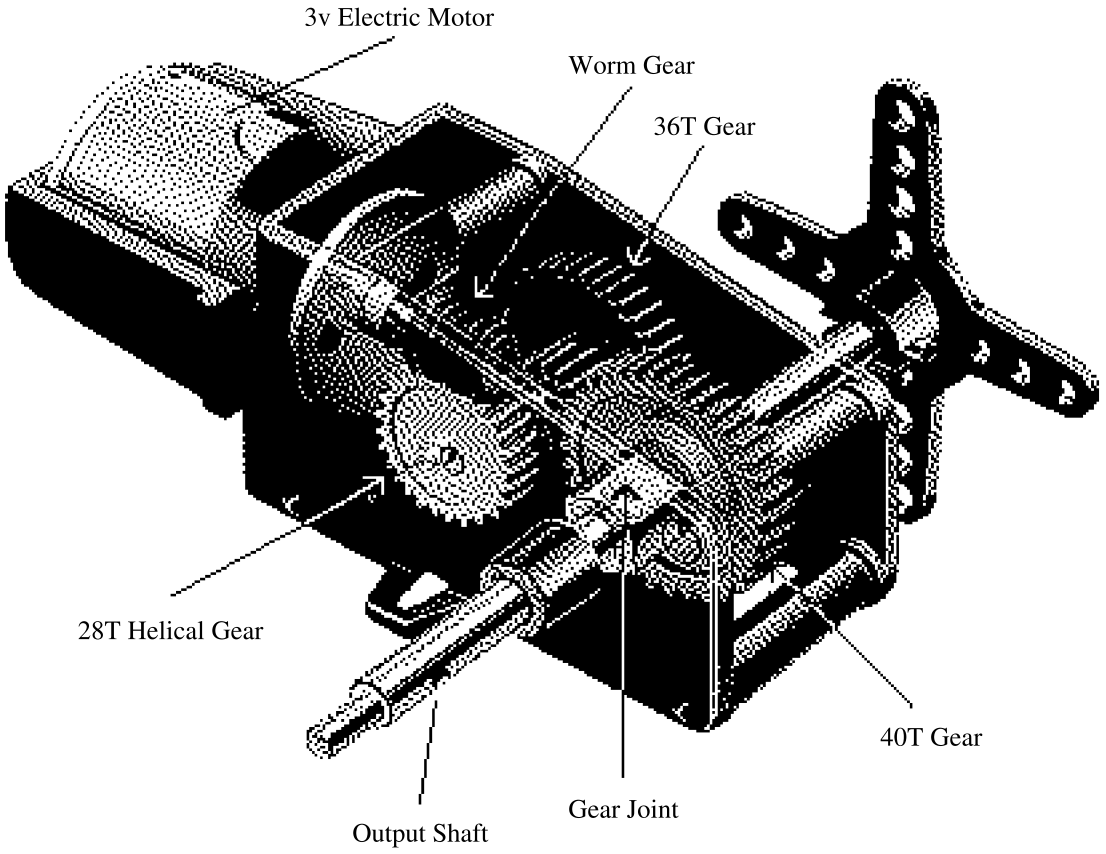


Figure 8.1 Worm gearbox set from Tamiya.

The planetary gear box kit of Figure 8.2 contains many small gears. The gears are layered onto each other so as to provide a higher reduction ration while keeping the housing small enough. These gears provide a 400:1 gear ratio satisfactory for horizontal leg movements. The shaft for this gear box is short and less rigid than the worm gear box. Three long bolts hold the housing layers together. These bolts must be tight enough to hold the housing together, but not too tight as to prevent the gears from movement. In fact, the bolts had to be kept reasonably loose, resulting in occasional gear slippage. Consequently, some of the gear housing had to be glued into place.

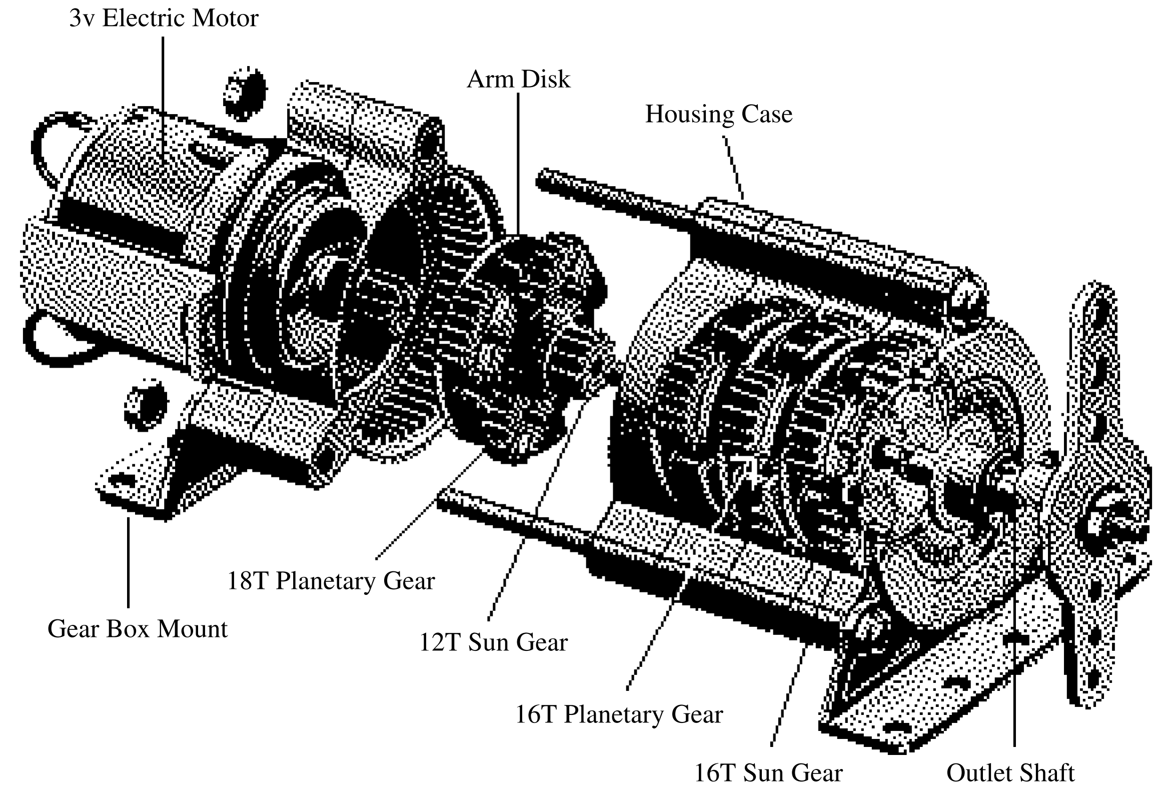


Figure 8.2 Planetary gearbox set from Tamiya.

The frame of the final version of RABI consists of side shafts made of square plastic tubes obtained from a hobby store. Aluminum cross bars are fastened horizontally across the shafts to hold the side bars together. There is an upper and lower level each identical in construction. The upper and lower layers are held together by the 6 planetary gear boxes which provide the horizontal movement of the legs. All electronics and wiring are attached to this two layer frame as well as the head motor which is currently not being used. Figure 8.3 shows the two layer frame with the planetary gear boxes holding them in place.

Each version of RABI has ping-pong ball eyes attached. Photocells were fastened inside them so that they may be used as light sensors. Although the light sensor circuitry is not used at the moment, the eyes provide aesthetics. These eyes were fastened to the frame of the robot and in some versions, fastened to a head motor.

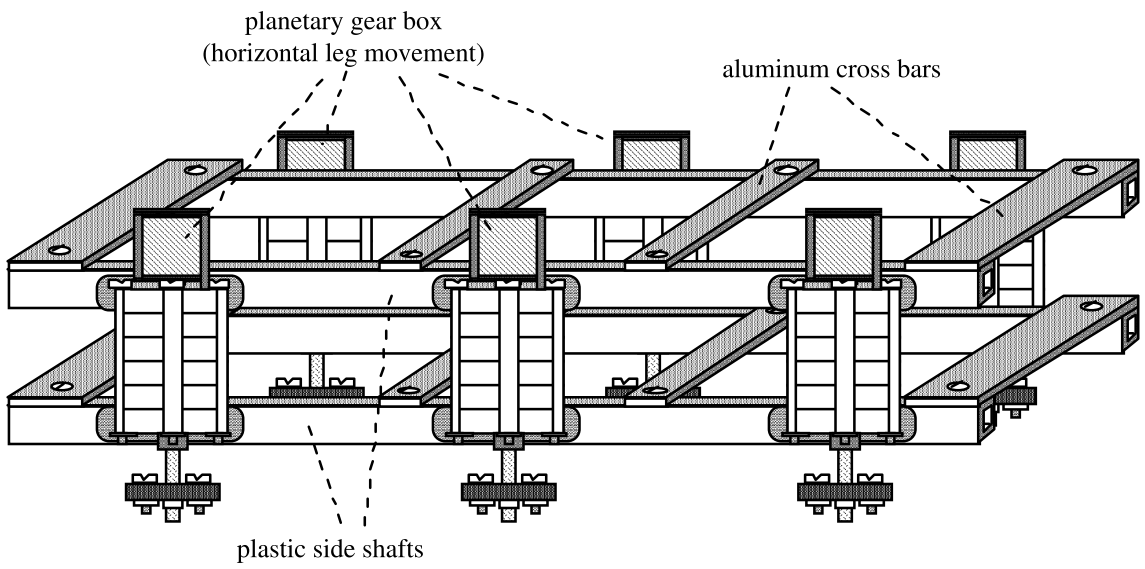


Figure 8.3 The two layer mechanical frame construction on which all electronic boards, sensors and wiring are attached.

8.1.1 Version 1.0 - The Inauguration

The very first version of RABI used 12 worm gear sets, two for each leg. One gear box was used to pivot the leg horizontally, the other was attached to this pivoting gear box, providing vertical pivoting motion. The initial frame was constructed with thin aluminum in the shape of a topless box. The horizontal motors were fastened to the inside walls of the frame with the shafts protruding through the bottom. The second set of gear boxes were then attached to these shafts. The leg itself was a 1 inch rigid aluminum plate with half a ping-pong ball for a foot. The ping-pong ball was glued to the end of a push button switch which was fastened to the leg. This leg design is shown in Figure 8.4. Each leg initially had 3 micro switches with extended levers. These switches were to be used to detect collisions in the front back and outwards directions. The switches were eventually discarded since the legs were unable to provide precise positioning measurements. These "bump" sensors were later replaced by an antennae system mounted at the front of the robot.

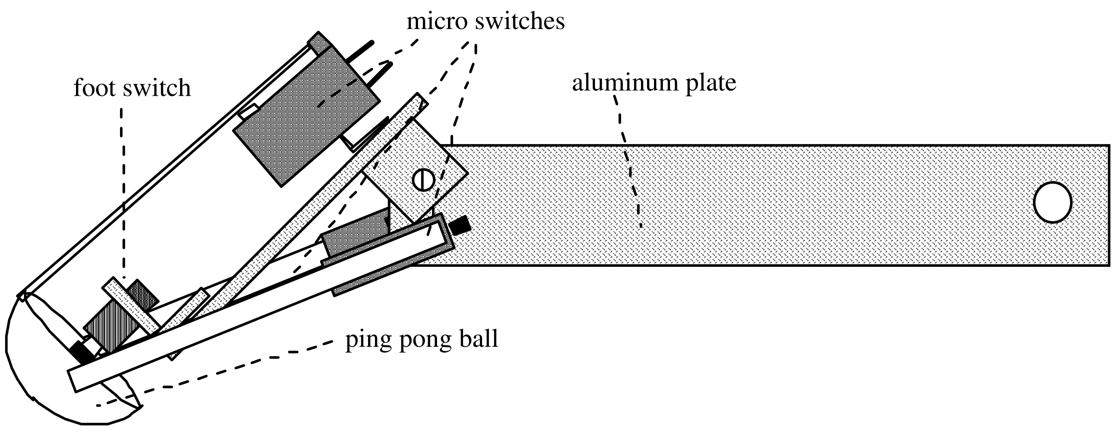


Figure 8.4 Initial leg design.

Once the legs were attached, the motors were connected to a power supply to determine if they had the ability to lift the robot up. The legs were able to support the robot' s weight but were not strong enough to lift the body. Moreover, the entire robot was very wobbly and unstable. Much of this flimsiness was due to the slack in the gears, the plastic shaft fastening end piece and the leg design. It was clear that some changes had to be made. Plate 1 shows a photograph of the first version of RABI without the feet or microswitches attached.

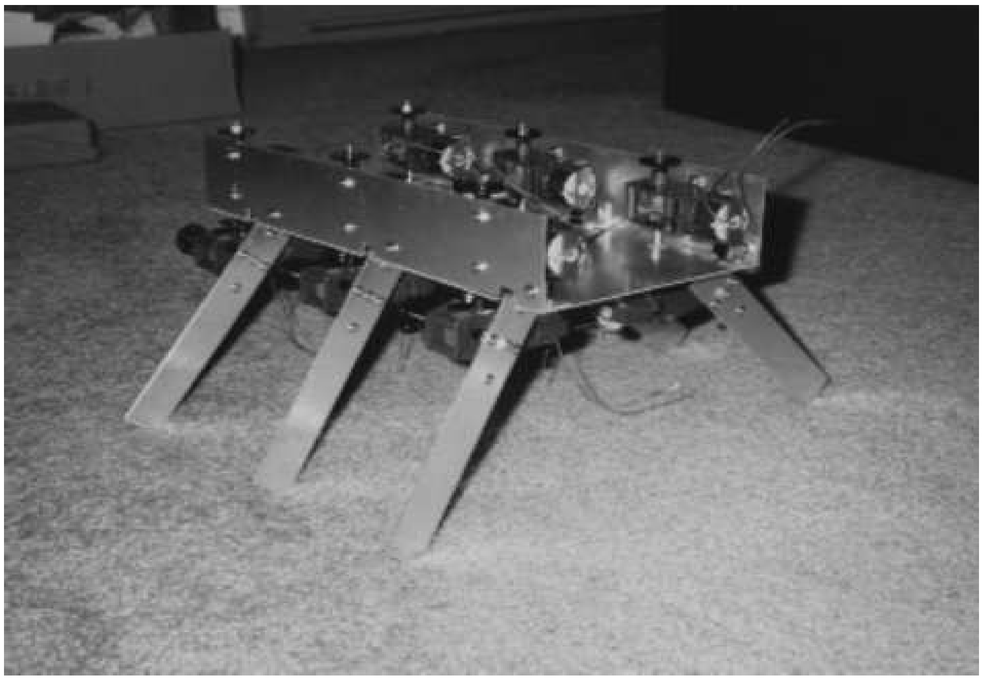


Plate 1 Photograph of RABI version 1.0.

8.1.2 Version 1.1 - Beetle

The next version of RABI was an attempt to solve the body lifting problem encountered in the version 1.0. The worm gear motors providing the vertical leg movement were replaced by planetary gear box units since the gear boxes provided a larger gear ratio and thus more power. These motors were attached to the shaft of the horizontal worm gear motors as before. The frame of the robot remained unchanged. Once connected, these new planetary gear motors had the ability to lift the body but they were not without problems. This new gear kit does not have a worm gear and thus, when the motor was turned off, the weight of the robot caused the motor shafts to slowly turn. As a result, the robot would slowly sink to the ground when the motors were turned off. A photograph of the mechanical design of version 1.1 is shown in Plate 2. The horizontal worm gear motors are hidden inside the casing however, the tops of the shafts can be seen.

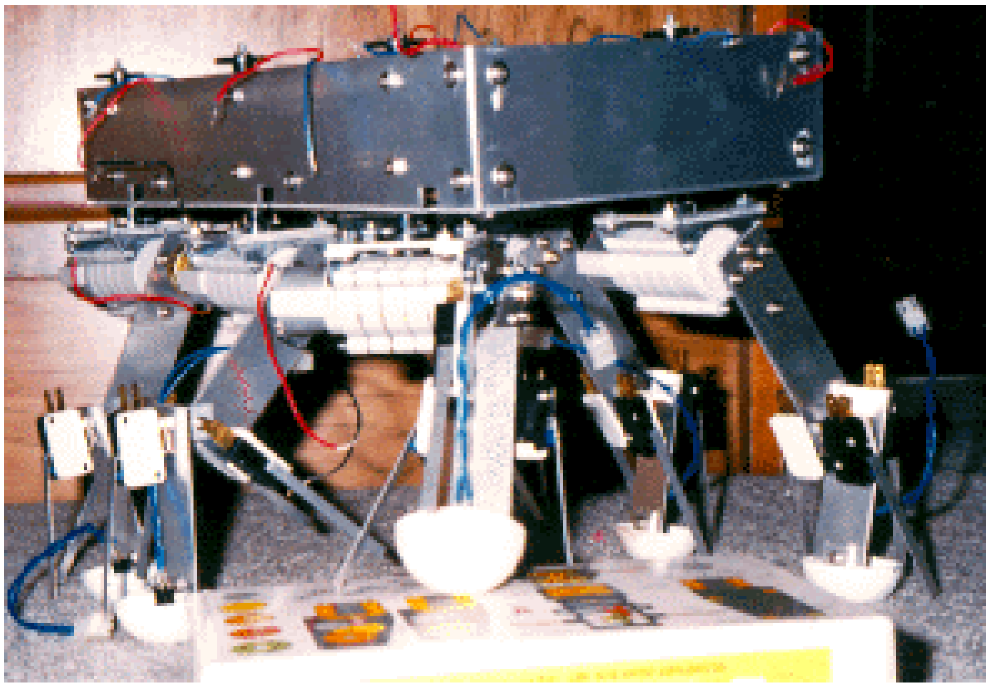


Plate 2 Photograph of RABI version 1.1.

8.1.3 Version 2.0 - Spider

After the two failed attempts at leg construction, it was clear that a different leg design was required. The motors were not able to provide adequate torque needed to move the leg vertically since the legs were connected directly on the shafts. Furthermore, pressure on the foot of a leg easily turned the motor shaft since the design allowed enough leverage. This prompted a new leg design in which the motors were used in a push/pull fashion. A diagram of this leg design is given in Figure 8.5. A threaded rod was attached to the shaft of the planetary gearbox. The legs were re-constructed from rigid plastic shafts in which part of the shaft was locked in place, allowing a vertical pivot. A nut (in the form of a threaded square aluminum chunk) was then attached to the top of the leg and screwed onto the threaded rod. The nut was attached using a holding pin on each side such that it was able to tilt. The feet were held on with elastics, allowing the foot to pivot and providing a more flexible surface contact.

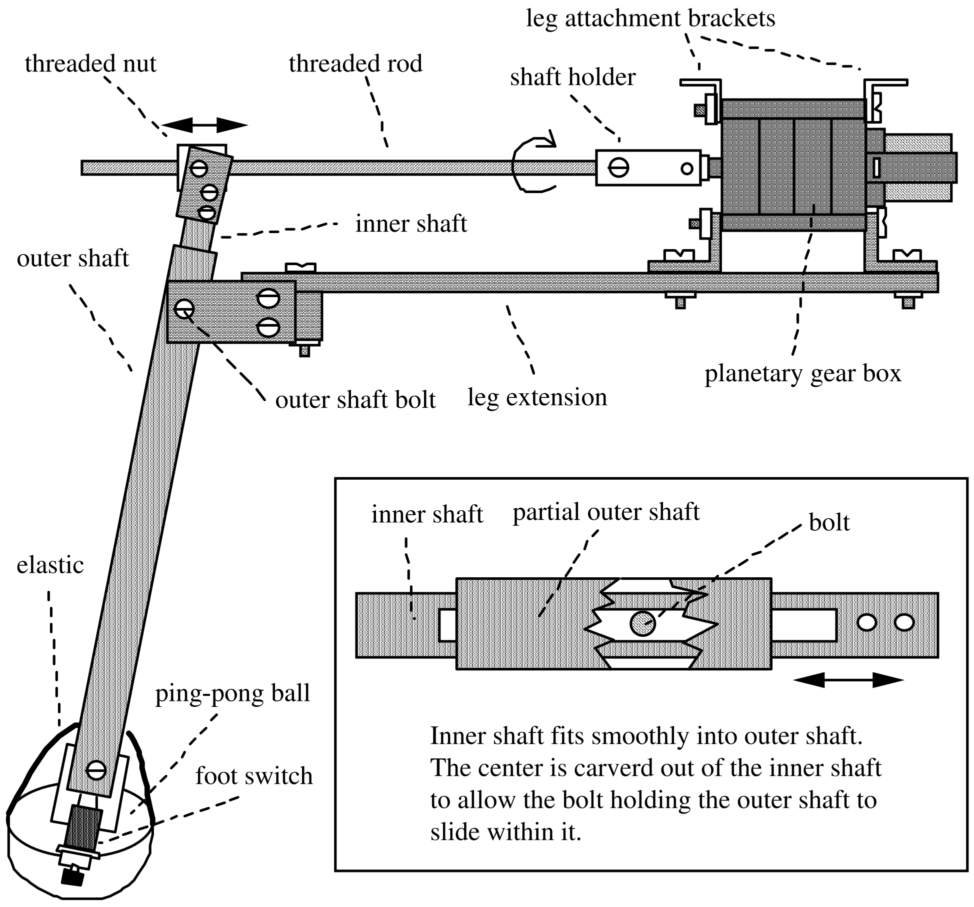


Figure 8.5 Leg design of RABI version 2.0. The motor spins the threaded rod causing the threaded nut to move in the horizontal direction, resulting in vertical leg movement.

When the leg contracts and expands, it must have some way of stretching since it is fixed at one point. To allow stretching, a smaller shaft was placed inside the outer leg shaft. The middle of the inner shaft was carved out such that the bolt holding the outer shaft did not prevent the leg from stretching. This leg design resulted in slow vertical leg movements. Moreover, the inner plastic shaft was easily bent and the leg was not very rigid. After a while, the threaded nut became worn out and the leg began to slip and snag. Once all the electronics and wiring were placed on the robot, it weighed too much and the legs were not able to lift the body. Thus the saying: "back to the old drawing board". Two photographs of RABI version 2.0 are shown in Plate 3 and Plate 4. Notice that this version of RABI used proximity sensors as obstacle detectors instead of antennae. These proximity sensors never quite worked correctly due to their sensitivity of ambient light.

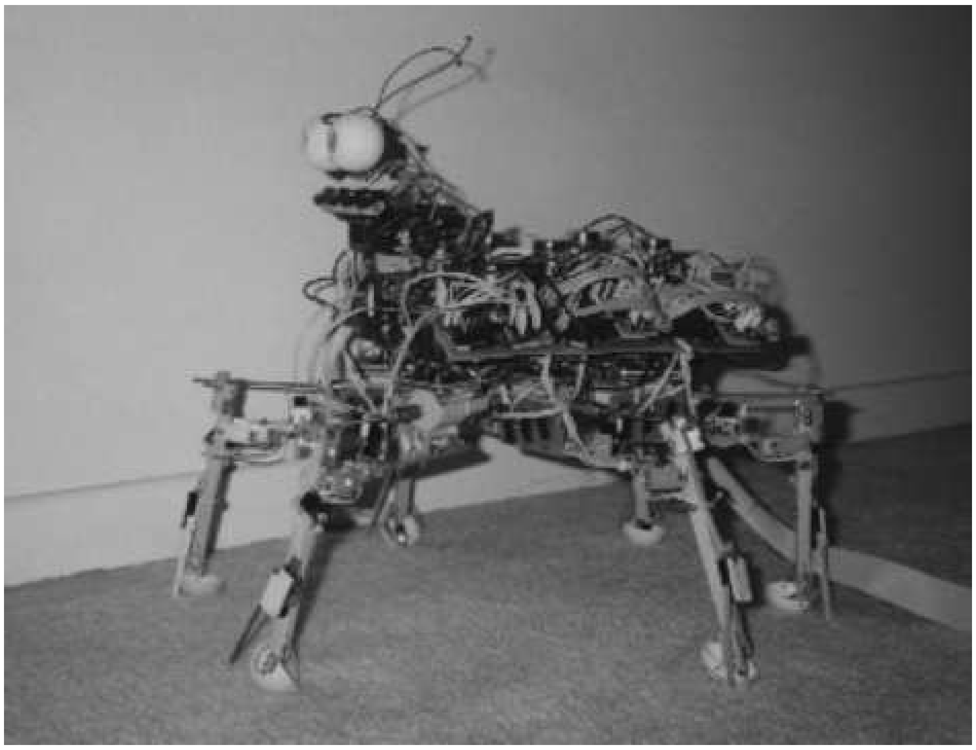


Plate 3 Photograph of RABI version 2.0 - side view.

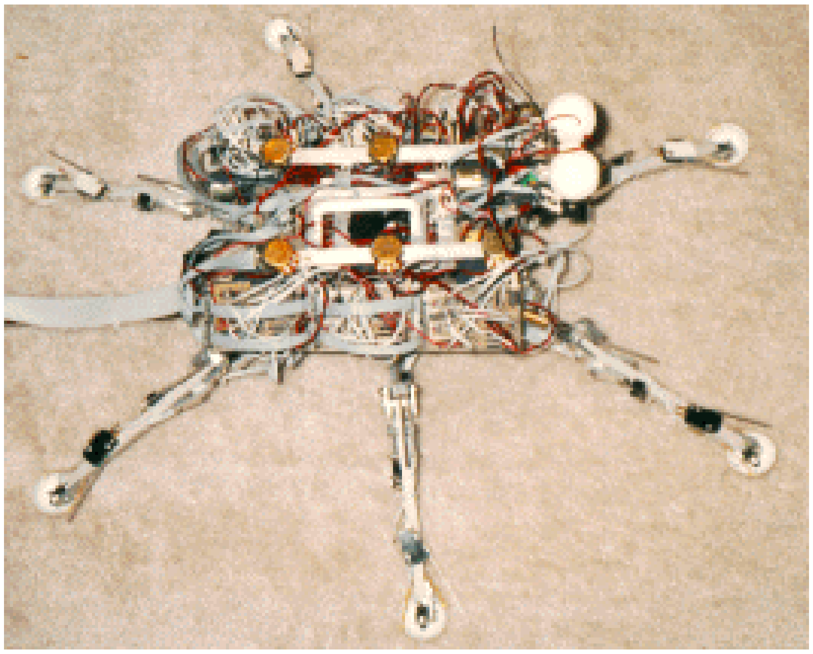


Plate 4 Photograph of RABI version 2.0 - top view.

8.1.4 Version 3.0 - Quadruped

Versions 1.0, 1.1 and 2.0 all had one thing in common; a weight problem. In all 3 trials, the legs were not strong enough to lift the body and remain stable. Thus, an attempt was made to reduce the overall weight. Since most of the weight is attributed to the legs and motors, two of the legs were eliminated resulting in a quadruped robot. Moreover, the legs were redesigned as shown in Figure 8.6.

All previous versions of legs shared the same basic type of movement. All of the leg joints were rotational, requiring the motor to produce a strong torque to achieve lifting. Version 2.0 had the additional problem of friction between the foot and the ground since the legs were designed to slide outward during the process of standing up. It was clear that a rotational joint would not suffice for vertical leg movement. As a result, a prismatic joint was constructed based on a "rack and pinion". This design was successful and provided enough power to elevate the body.

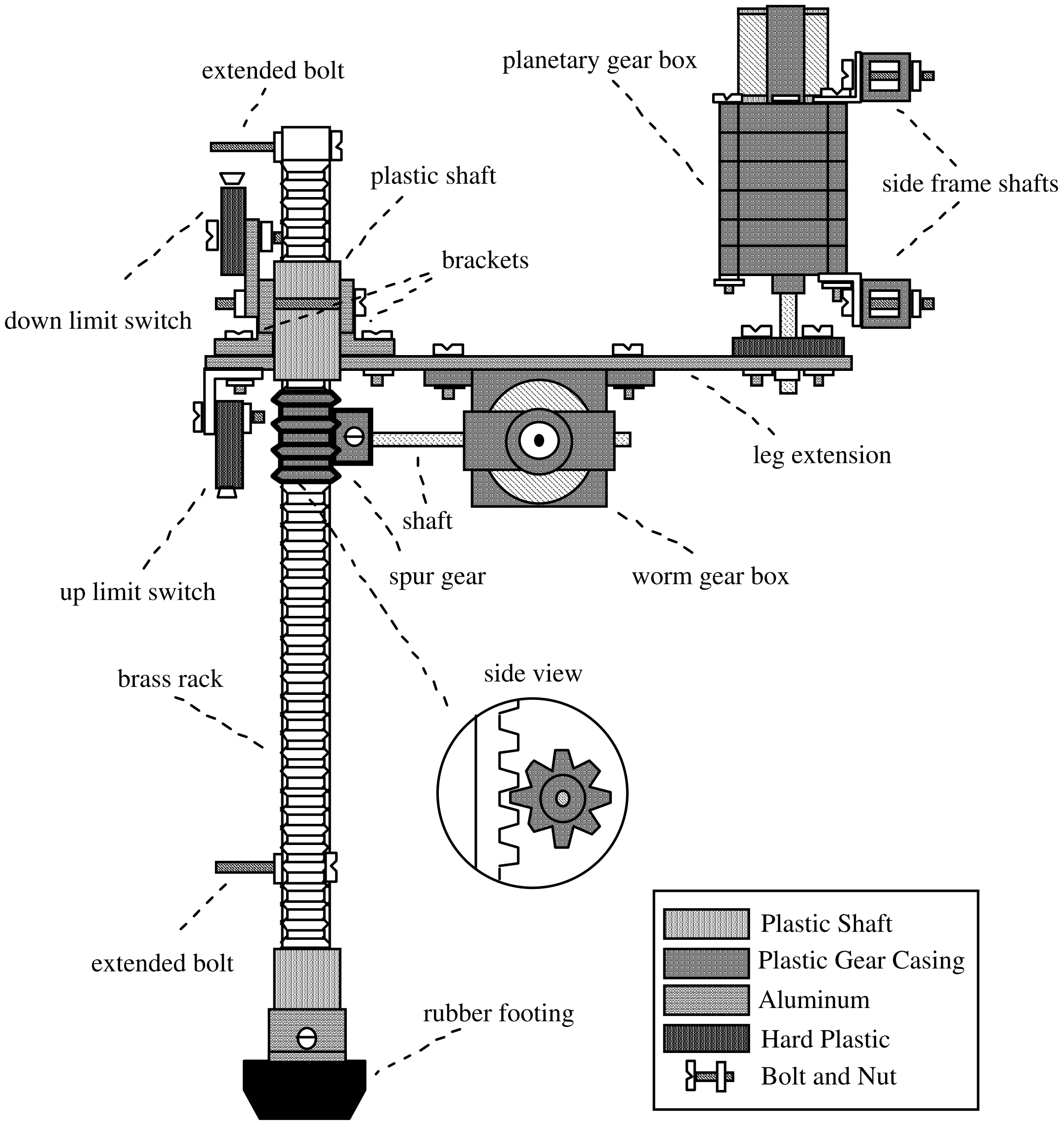


Figure 8.6 The prismatic leg joint for vertical movement.

The leg itself consists of a brass rack that translates up and down by a turning spur gear that interlocks with the rack. An attempt was made to construct a rack made of aluminum by cutting evenly spaced groves in a square aluminum bar. Due to the lack of sophisticated cutting equipment, the resulting aluminum rack contained uneven slots of imprecise depths and shape. The aluminum racks were initially used on the leg but they did not function smoothly, and the gears often jammed in the misguided groves. Heavier brass racks were eventually used since they were available and professionally constructed.

The brass rack slides freely through a plastic shaft which is bolted down with brackets. There are two extended bolts that prevent the brass rack from sliding out from the shaft. These two bolts are also used a triggers for the leg limit micro switches. When the leg is placed down (up) far enough, the bolt closes the down (up) limit switch causing the motor to stop. At the bottom of the leg is a rubber footing as found on the bottom of most large electronic devices. This rubber footing was only used in the final version of RABI; it is shown here only to avoid duplicating the diagram in the sections to follow. Version 3.0 actually utilized the same footing structure of version 2.0 with the exception that the elastics were removed, and the ping-pong ball was fastened to the switch using silicon. Like the elastics, this silicon also allowed a somewhat flexible foot to surface contact.

This leg was fastened to a planetary gear motor shaft which was attached to the plastic side shafts as in previous versions. The completed quadruped version with electronics attached is shown in Plate 5 and Plate 6 [47].

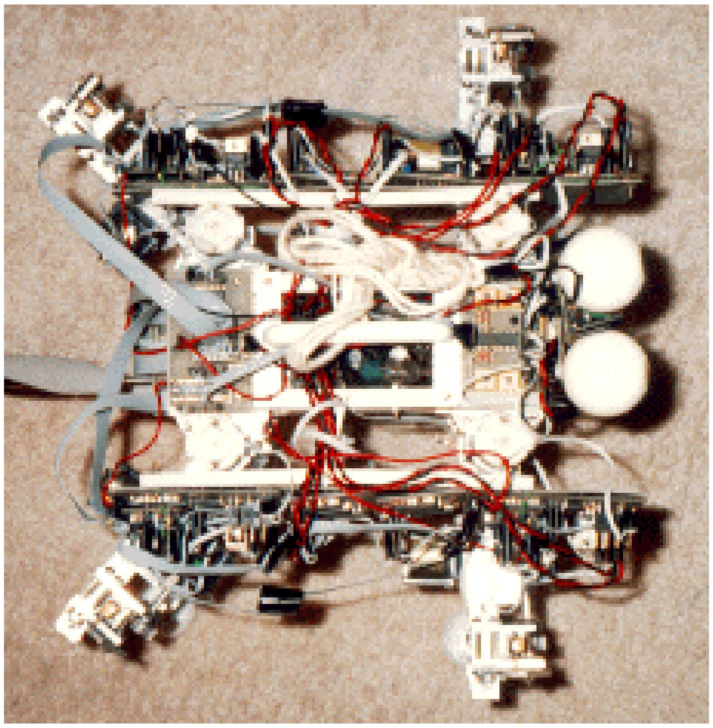


Plate 5 Photograph of RABI quadruped version 3.0 - top view.

[47] The string in the center of the robot was used to hang the robot up during construction and repair.

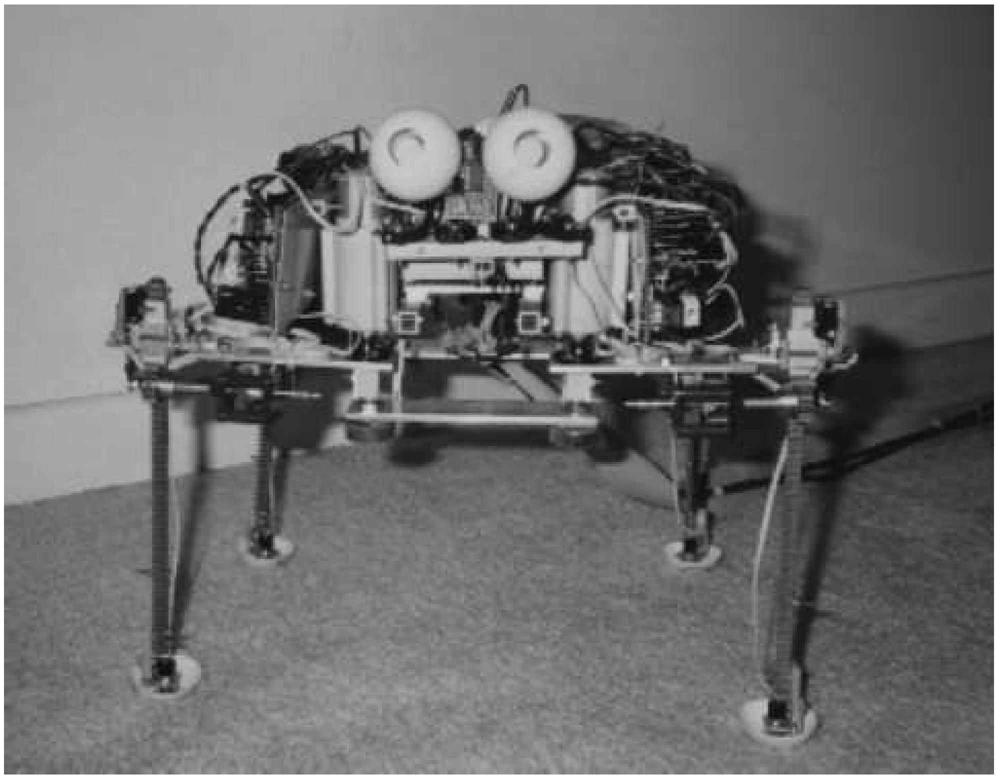


Plate 6 Photograph of RABI quadruped version 3.0 - front view.

Once constructed, the quadruped had the ability to lift itself up with ease. However, there were problems with stability (see chapter 3). The robot often tipped in all directions and was not able to walk continually without falling. After rigorous testing and failed attempts at maintaining stability, it was decided that 6 legs were necessary.

8.1.5 Version 4.0 - Hexapod

The success of the leg design in version 4.0 simplified the conversion of the quadruped into a hexapod. Two more legs were duplicated and the body was extended. Plate 7 shows the completed mechanics of the revised hexapod without the electronics. This version of RABI was capable of walking and turning without falling over. Occasionally, a leg twitch would cause tipping, but RABI's software usually recovered.

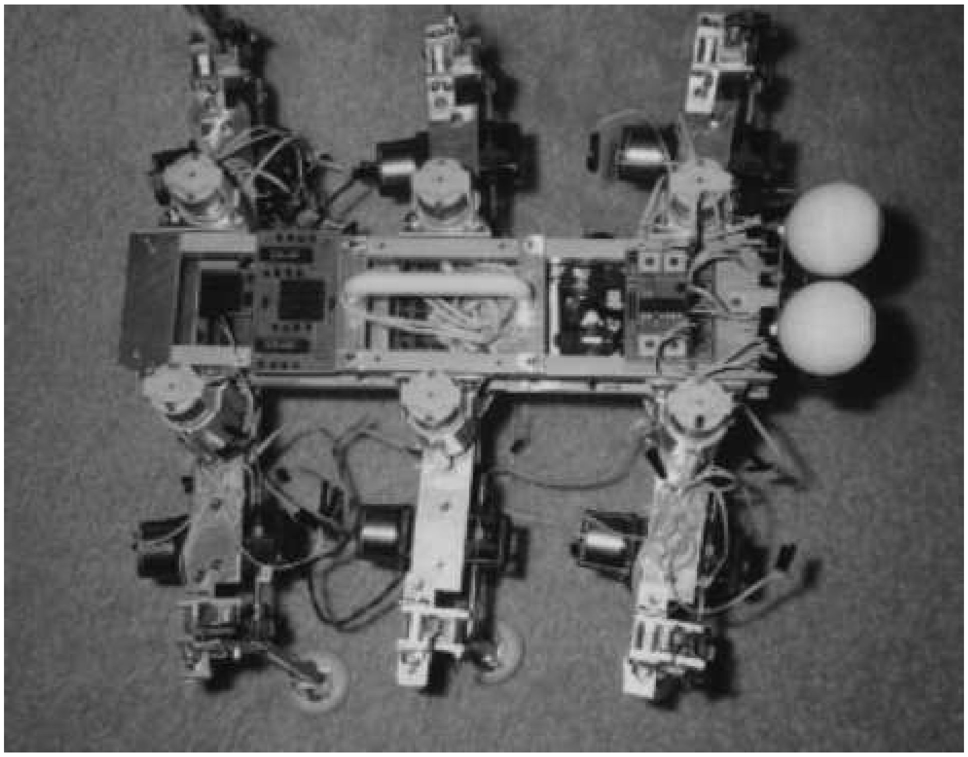


Plate 7 Photograph of the revised hexapod; RABI version 4.0.

8.1.6 Version 4.1 - Insect With Antennae

During the programming of the instinctive behaviors, it became clear that the infrared proximity sensors were not adequate for detecting obstacles. They constantly received spurious data from stray light sources and did not operate at the desired proximity. This problem could only be detected through experimentation with a physical device. This situation demonstrates the importance of designing physical systems rather than simulated systems. The infrared sensors were replaced by an antennae system as shown in Figure 8.7.

This antennae system consists of 4 antennae made of piano wire. The piano wire provided a flexible means of detecting data. The piano wire was fastened at one end with a nut and bolt and the other end protrudes outwards. The antennae passes through a washer which is fixed to a plastic shaft. Wire leads are placed at the fastened point of the antenna and on the washer. When the antennae makes contact with the side of the washer, a current passes through one of the wire leads into the other thus acting as a binary switch. Due to the small opening of the washer, the antenna had to be finely adjusted such that it hovered in the center of the washer hole when at rest.

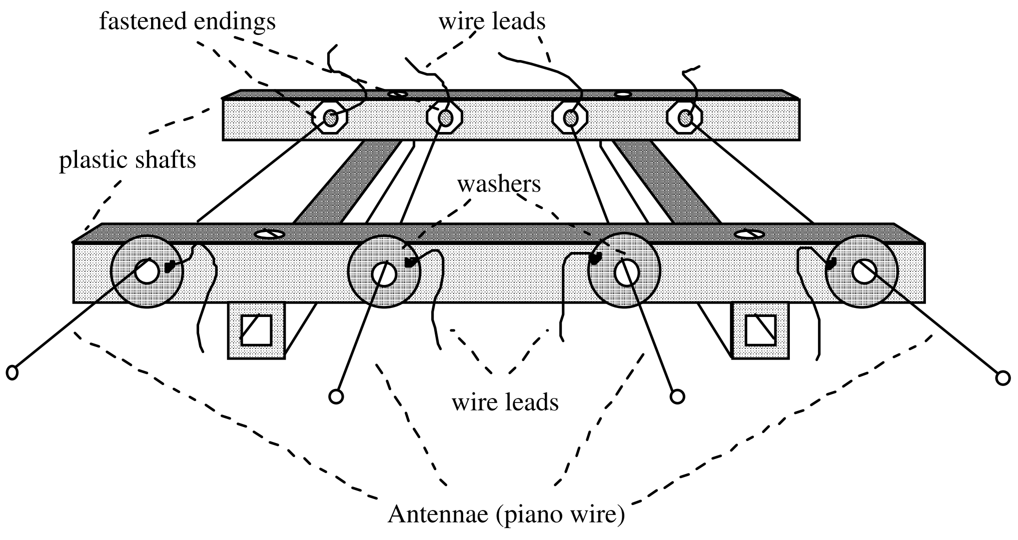


Figure 8.7 The 4-antennae system for detecting obstacles.

8.1.7 Version 4.2 - The Final RABI

The final (current) version of RABI has a number of improvements from the last version. The first improvement was the antennae. The adjusting of the previous antennae was difficult, involving slight bending of the piano wire with a pair of pliers. The antennae of version 4.1 were improved upon allowing easier adjustments to be made to the antennae. The new design is shown below in Figure 8.8. This design left access holes allowing the antennae to be adjusted easily. The design also reduced the size of the fastening system and gave it a more aesthetic appearance. This version of RABI also included side antennae with similar construction. These antennae extend from underneath the robot, fitting between the first two legs on each side.

Another improvement was in the foot design. The foot switch on the bottom of the leg often provided inaccurate readings. The switch would not make contact unless a sufficient amount of weight was issued on the leg. Furthermore, the flexibility of the ping pong ball caused the switch to snag and miss contact. These problems were ignored since it was not a major problem. Eventually, one of the foot switches broke off resulting in a decision to redesign the feet. The pin pong ball was discarded and replaced by a rubber footing as mentioned in version 3.0. The foot switch was replaced by a micro switch at the top of the leg such that it made contact when the leg was fully extended (down). The placement of the switch will prevent it from breaking off, but results in a different function. This final version of RABI is depicted in Plate 8.

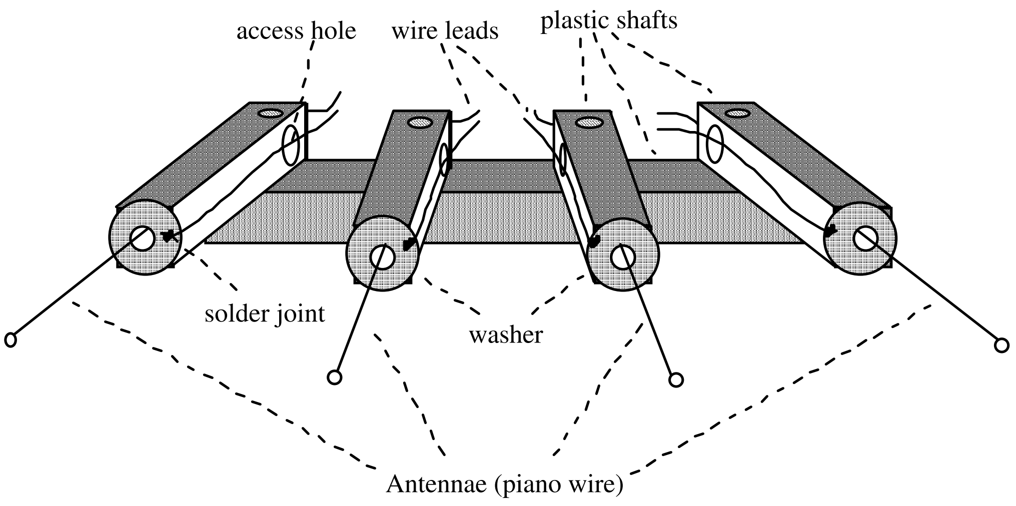


Figure 8.8 The improved antennae system design.

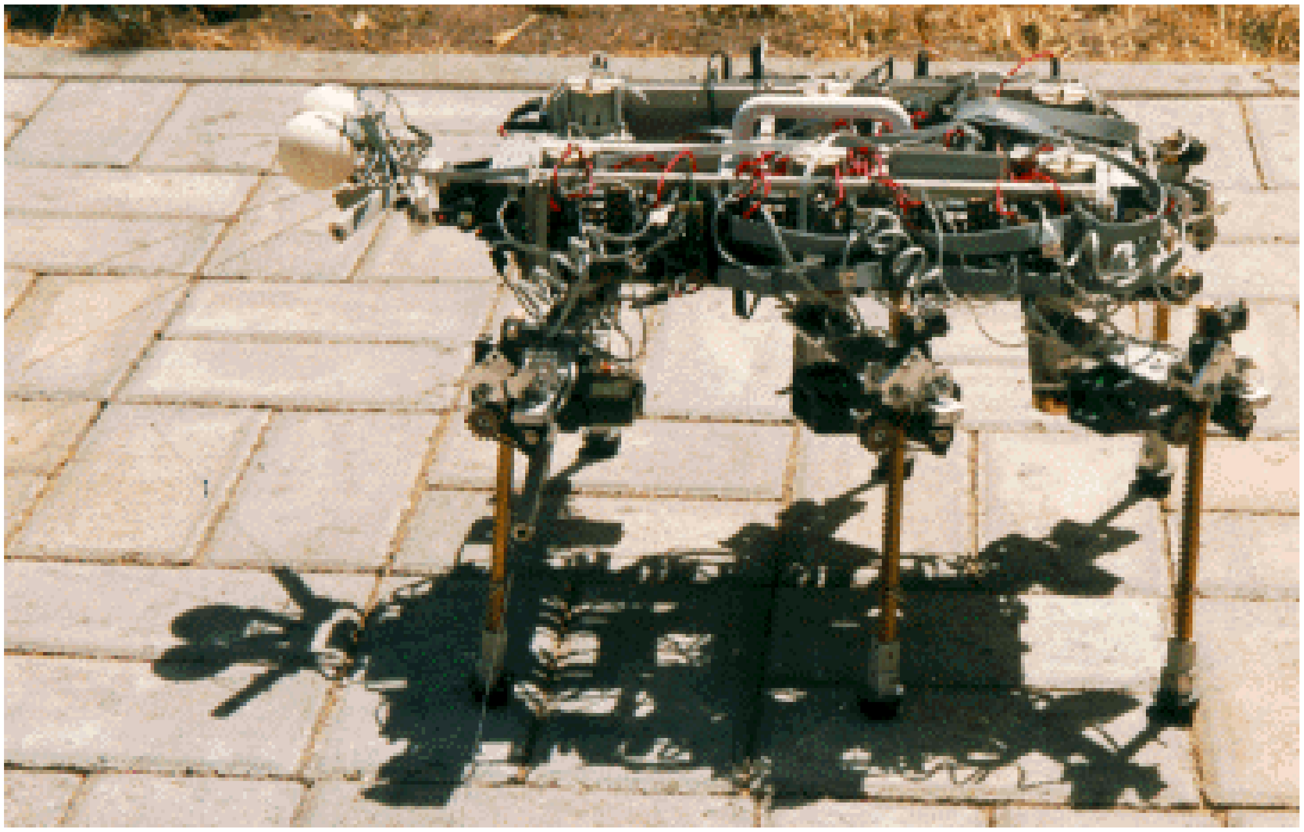


Plate 8 Side view of the final version of RABI.

The new switch does not detect when the foot has contact, instead it detects when the leg is down [48]. This new functioning is acceptable since RABI only uses a tripod gait which works in level environments. Therefore, if the environment has no holes, there will not be a problem using this new switch in the same manner as the previous foot switch.

8.2 The RABI / Computer Interface

The hardware version of RABI interfaces to a PC compatible computer. More specifically, the robot is actually connected to a 386 computer via a Quatech PXB-241 interface card which is a 24 buffered digital input/output adapter. The robot has a tether of 20 lines representing 3 buses which plug directly into the PXB-241. In addition, there are two power lines attached to the tether providing +5v power, +3v power and ground to the robot. This power comes from a Condor 5v power supply and an EDLaboratory regulated DC power supply respectively. Figure 8.9 displays the setup.

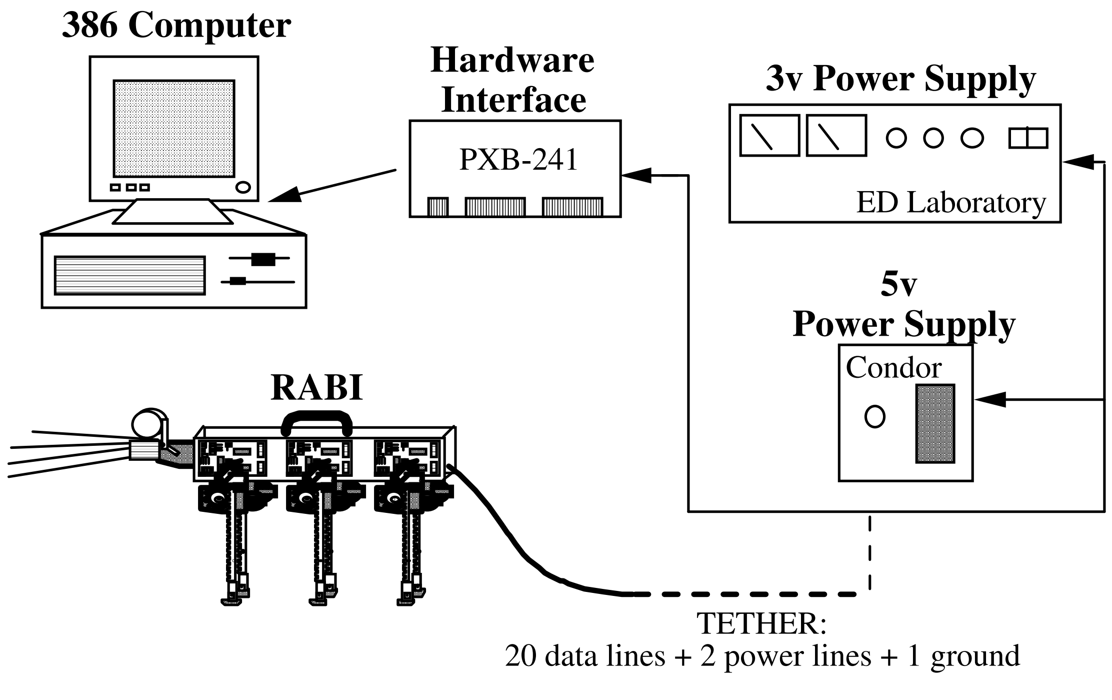


Figure 8.9 The hardware setup for RABI. A tether connects the robot to the PXB-241 interface for control, and to 2 power supplies for power.

[48] This switch provides the same function as the leg down limit switch. The leg limit switch could not be used as foot down detection since electronically, it is used as a cutoff switch.

By keeping the power supplies off the robot, considerable weight problems are eliminated. Furthermore, by keeping the controlling software separate from the robot, it is easier to make modifications to the various networks and algorithms. As a consequence of keeping the computer off-line, the robot must have some method of communicating with it. The simplest method was to connect it, using a tether, to a commercial interface card which provides the necessary electronics and buffering.

The communicating interface consists of 3 buses. An 8-bit *input* bus reads sensor information from the robot and an 8-bit *output* bus sends data to the robot for actuator control. Finally, data from a 4-bit *control* bus sends a command to the robot' s control electronics telling the robot to either read its output bus or send something onto the input bus.

All of RABI' s software is written in Smalltalk[49]. This language provides a rich object oriented environment suitable for robotic software design. It also allows easy access to the PXB-241 interface board by using just two DOS instance methods: **outByte:toPort:** and **inByteFromPort**. These two methods allow data to be sent to and read from the interface card, thus allowing the sending and receiving of data from the robot. There is however, a limit to the transfer speed of the data through the PXB-241 ports, and thus additional electronic circuitry was added to the robot to reduce the transfer rate required for control. Most existing robots require real time control, but RABI is a relatively slow walking machine that does not require real time interaction.

8.3 Electronics

Most of the robot' s electronics consists of simple IC' s, transistors and resistors. Since all of the behaviors are programmed by the interfaced computer, the electronics merely need to read in actuator commands and communicate sensor values back to the computer. There are separate electronics boards for leg positioning, leg interfacing and control, sensor interfacing and leg selection. Additional boards were created for head positioning and control, proximity detection and light sensing but these are not used on the final version of RABI and will thus not be discussed.

[49] Smalltalk V/286 version 1.1 from Digitalk Inc.

8.3.1 Leg Positioning

For all types of actuator control, some form of position feedback is necessary. For RABI, there is a need to know roughly where each leg is at any one time so that coordinated walking is possible. On level surfaces, a precise indication of the vertical position of a leg is not required. The horizontal position of the leg is more important in this type of environment, thus there is a need to measure the horizontal position of the leg. A simple method is to construct an analog to digital converter using simple IC' s and some resistors as shown in Figure 8.10. By connecting a potentiometer to the shaft of the geared down motor, an analog signal can be obtained giving an indication as to the leg' s position. This analog signal can easilybe converted into digital 3-bit data.

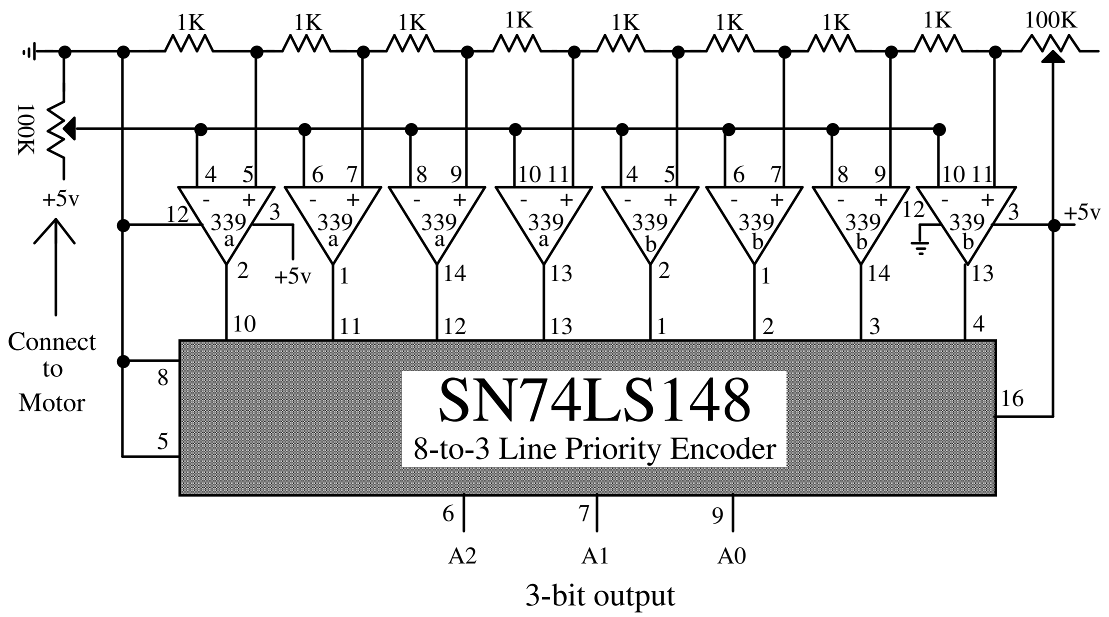


Figure 8.10 The schematic diagram for a 3-bit analog to digital converter used to obtain horizontal positioning information for each leg.

In the diagram, the left most 100K potentiometer connects directly to the geared motor shaft. The analog voltage from the potentiometer passes through a series of 1K resistors and then to another 100K potentiometer used to adjust the sensitivity. By sampling the voltage at different points in this resistor series, different voltages are obtained. In fact, the voltage is cut in a linear fashion such that the voltage drop after each resistor is approximately equal. By connecting a series of voltage comparators (LM339) at these different sampling points and comparing the voltage to a common reference voltage, an indication to the amount of resistance in the potentiometer is obtained. Each of these voltage comparators emits a high signal when the voltage of the potentiometer exceeds their reference voltage. An 8-to-3 line priority encoder (74LS148) then reads in the 8 signals from the voltage comparators and outputs 3-bit data indicating the position of the potentiometer, hence the position of the leg. For leg movement in the vertical direction, only 3 states can be identified: fully extended, fully contracted or somewhere in between. This positioning information can be obtained by reading the leg limit switches, hence no positioning circuitry is required.

8.3.2 Leg Control

Each of RABI' s 6 legs has a small circuit board allowing it to interface to the input and output buses as well as interpreting commands and moving the leg appropriately. The schematic for this interface is given in Figure 8.11.

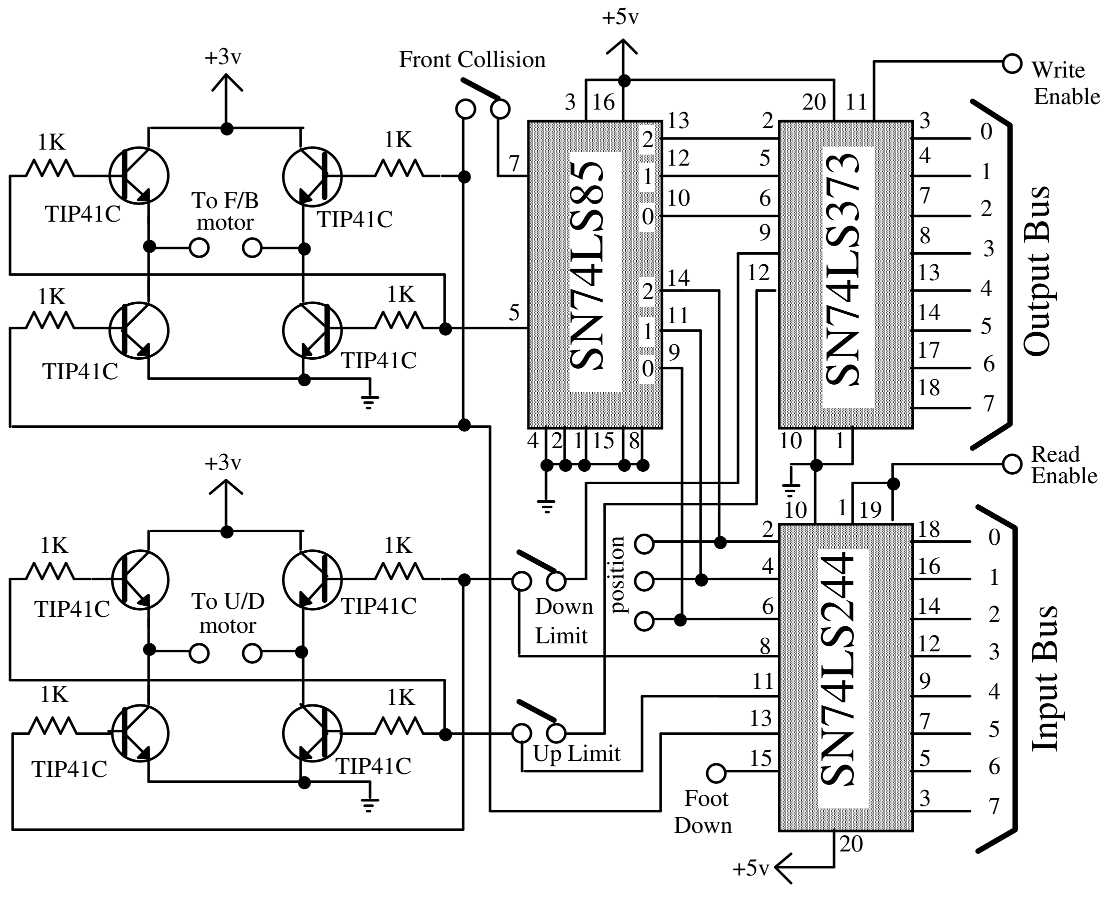


Figure 8.11 The schematic diagram for the leg control circuitry.

The circuit can be decomposed into pieces making analysis easy. The two left most chunks of circuitry represent the motor control circuitry using an "H" network of transistors [50]. This piece of circuitry requires a 2-bit binary signal indicating the direction of the motor. A binary input of 01 or 10 causes the motor to move in clockwise and counter-clockwise directions. An input of 00 causes the motor to shut off. An input of 11 is not allowed since it will attempt to turn the motor on in both directions causing overheating of the transistors. The 74LS85 is a 4-bit magnitude comparator which handles the horizontal positioning of the leg. The output bus provides a desired 3-bit horizontal position for the leg. The comparator compares the desired 3-bit horizontal position with the actual position and produces a less than, greater than or equal to resulting signal. The less than and greater than signals provide the necessary 2-bit information required for motor control. Thus, the leg is always being positioned unless the comparator receives a match between the actual and desired position.

During construction, an overshooting problem arose due to this method of positioning. During leg movement, the comparator shuts off the motor once the leg has reached the desired position. When shut off, the motor does not stop immediately, instead it slows down to a stop. While slowing down, the leg passes the desired position, causing the comparator to reverse the motor in order for the leg to backup to the desired position. Once reached again, the motor is shut off. Again, the leg overshoots and the process enters into an oscillating pattern around the desired leg position, never stopping at the desired position. This problem causes a kind of "twitching" in the legs making walking impossible. Since the leg motors had much slack in the gears, the legs are not accurate enough for 3-bit precision. Thus, the positioning circuitry was reduced to 2 bits to help alleviate the oscillation problem. Actually, for the tripod walking used, only the 2 most extreme leg positions are required.

A further problem was encountered with the strength of the motors. The earlier designs of RABI all failed due to the accumulated weight of the robot, causing the motors to struggle and even fail to support the weight. A part of the problem was in the leg design itself but another part of the problem lied in the electronics. The controlling circuitry was not powerful enough to drive the motors. As a result, the motors were pulling excessive current from the electronics, causing failure. To fix the problem, an additional circuit was designed for each leg to provide enough current to the motors so they would not draw current from the electronics. The schematic for this circuit is shown in Figure 8.12(a) and 8.12(b).

[50] This motor control circuit is based upon the circuit given by [McComb 87] page 99.

There are two copies of each of these circuits for a total of 4 additional circuit pieces that connect to the motor control "H" network. Essentially, the inverters (74HC04) provide low current buffers for incoming signals from the electronics. For (a), this incoming signal comes from pins 9 and 12 of the 74LS373 which specify the vertical direction to move the leg. For (b), the incoming signal comes from pins 5 and 7 of the comparator which controls the horizontal leg direction. The signal then passes through a 10K biasing resistor which drives a small switching MPSA13 NPN transistor. For the vertical positioning, the signal also passes through the limit switch allowing disabling of the motor as shown in (a). The transistor acts as a switch providing direct power to the motor control circuitry, solving the problem of current draw from the electronics. In (b), an additional inverter is added to undo the inversion of the first inverter so that the binary signal remains unchanged. That is, a signal of 00 must not be inverted to 11 otherwise an illegal state will occur for the motor control circuitry.

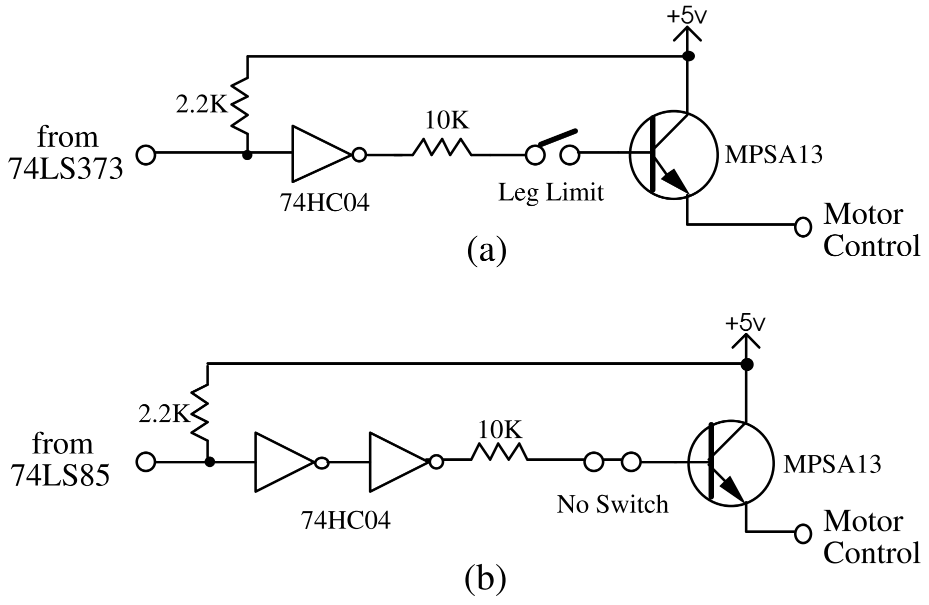


Figure 8.12 Additional circuitry providing the necessary current to the motors. (a) circuit for vertical motor and (b) circuit for horizontal motor.

The two right most ICs of Figure 8.11 handle the interfacing to the input and output buses. In this circuit a tri-state buffer (74LS244) was used to interface sensor data to the input bus. When the READ ENABLE line is set low, the sensor data is buffered onto the input bus and remains on the bus until the READ ENABLE line is set high again. The output bus uses a tri-state octal D-type transparent latch (74LS373). When the WRITE ENABLE signal is set high, the data on the output bus is latched and stored in the IC. This data remains stored until new data replaces it with a high WRITE ENABLE signal again.

The 7 bits of data buffered onto the input bus is sensor data corresponding to the leg position and switch settings as shown below in Figure 8.13. The 5 bits of output data are also shown in this figure.

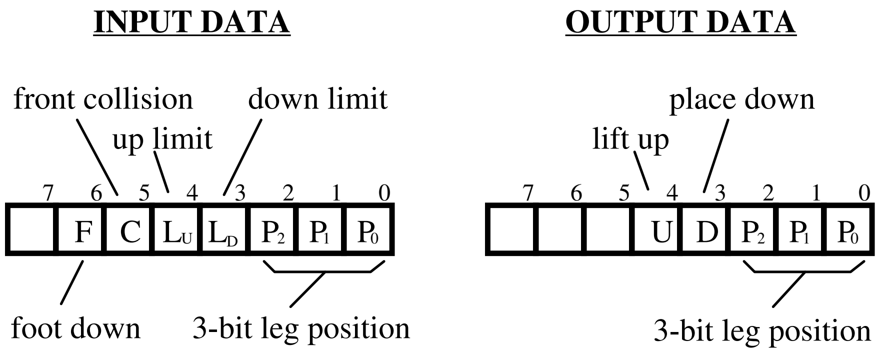


Figure 8.13 The 8-bit data format for the input and output buses. Note that bit 7 is null on the input data and 5, 6 &7 are ignored from the output data.

Here, the F, C, L_U and L_D bits are all readings for the foot, collision, leg up limit and leg down limit switches respectively. The F bit is low if the foot is touching the ground and high otherwise. Similarly, the C bit would be low if the leg has collided or bumped into an object and high otherwise. The circuitry was designed such that forward leg movement is halted whenever this switch is touched. Since the robot walks by lifting its leg and swinging it forward at the same time, the leg would continue to lift when the switch is closed. This allowed the forward leg movement to wait until the leg fully lifted over the obstacle before it completed its swing. This notion is analogous to a reflexive reaction to an obstacle stimulus. In essence it is an electronic reflex. The collision switch was eliminated from the final version of RABI since, the legs were not accurate enough to warrant this additional sensor. The L_U and L_D bits indicate whether the leg is fully up or down respectively. The leg limit switches are in a "closed" state normally and become "open" when they are touched. This allows the switches to cut off power to the motor circuitry preventing the leg from extending past its limit.

The data on the output line represents actuator commands needed to move the leg to the appropriate position. If the U bit is low, the leg will lift up. Similarly, if the D bit is low, the leg will be placed down. Care must be taken not to send a binary 0 for both these bits at the same time, otherwise the leg will not move and the transistors would overheat. Bits 0, 1 and 2 represent 8-bit data for the desired leg position. Once this data has been latched, the leg will begin to move to the desired position and up or down as specified. The electronic circuitry will halt leg motion once the desired position has been reached. This "self-moving" method of leg control frees the programming software from having to continually specify consecutive positions, reducing the need for real time control.

8.3.3 Robot Interface Circuitry

Each leg must be able to extract information from the output bus and send sensor information onto the input bus. At any one time the data on the output bus contains a command for only one leg, and thus the other legs should ignore the data. Furthermore, only one leg can place data onto the input bus at a time, otherwise the data would clash resulting in garbage information and perhaps circuit burnouts. It should be clear that there must be some circuitry directing the data on these two buses. For this reason, a third bus, called the control bus was created. The control bus is a 4-bit bus that contains controlling information allowing the various legs to be selected for input or output data transmissions. The circuitry required for connecting this bus is shown in the schematic diagram of Figure 8.14.

In this circuit, there are two 3 to 8 line decoders (74LS138) are used to decode the control bus signals into 8 control lines each. One decoder is used to turn on the WRITE ENABLE for the appropriate leg [51]. The other decoder is used to turn on the READ ENABLE for the appropriate leg or sensor to be read. The high bit 4 of the control bus is used to select either a read or write operation with a 0 and 1 value respectively. The use of the inverters (74LS04) is required since the decoders emit a low signal on the selected line; the OUTPUT ENABLE signal must be set high to latch the data in the leg control circuitry.

[51] There is room for expansion here if additional actuators are to be added, such as head movements.

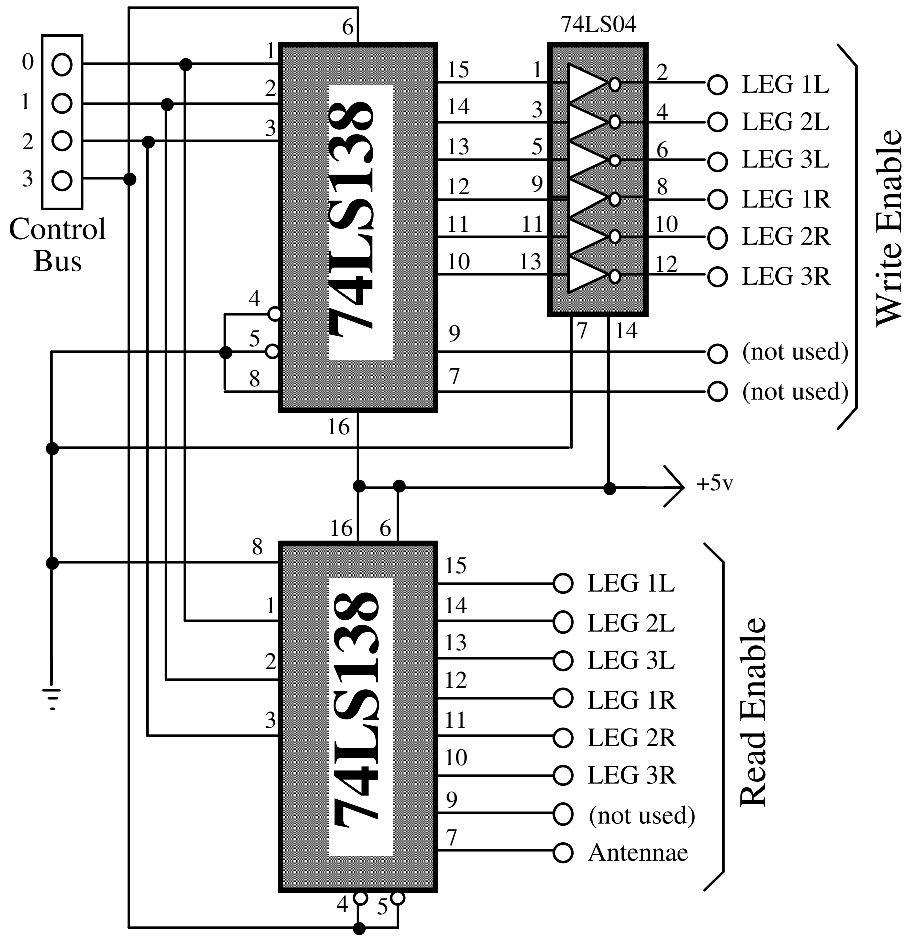


Figure 8.14 A schematic diagram for the control bus circuitry.

The Antennae signal coming from the bottom decoder of Figure 8.14 is used to enable the reading of the robot' s body sensors. This signal is fed into an additional circuit that latches various sensor data onto the input bus. A schematic for the additional circuitry is given in Figure 8.15.

There are actually two copies of this circuit; one for head sensors and one for body sensors. Only one is being used now since RABI only needs 6 lines for its antennae, leaving 10 data lines for expansion. The 6 antennae signals come directly from the sensors and are latched onto the input bus when the READ ENABLE line is set low.

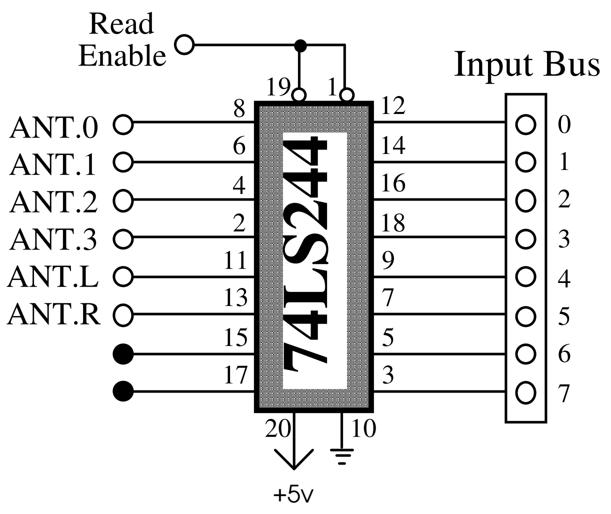


Figure 8.15 A schematic diagram for the sensor latching circuitry.

With both these circuits the PXB-241 interface allows RABI' s software to send commands on the output bus resulting in leg movement, and read in sensor values from the input bus. Table 2 depicts all possible functions (commands) that may be sent to the robot. The first half of the commands represent input commands in which the computer can read sensor and switch values from the robot. The second half of the commands represent actuator or control functions allowing the legs to be moved to a specified horizontal position and vertical direction.

8.4 Future plans

If RABI was to be redesigned, all of the electronic circuitry would be modified and some even eliminated. Some of the electronic circuits could have been simplified greatly since the resolution of the horizontal leg position was reduced to 2 bits. Furthermore, the software only makes use of the most extreme leg positions. Thus, the position sensor circuitry is not needed since limit switches would perform the same task.

All of the basic neural circuits for the instinctive behaviors would be electronically constructed. This would allow the computer interface to be greatly simplified since the interface would only need to communicate through a higher level protocol. For example, the computer could send out commands such as "walk forward", "stop", "turn left", "turn right", "follow edges", "wander", "stay in vacant areas", "go to light", etc. The electronic neuron networks would actually perform the desired functions and send back only sensor information such as antennae readings and light sensor readings. In fact, if the robot is to be programmed with only instinctive behaviors and no learning or map building is required, then the computer interface is not needed and the addition of batteries to the robot would make it completely autonomous and self contained.

Function	Cntrl.	Input Data	Output Data
Read leg L1 sensors	0000	- F C L_U L_D P_2 P_1 P_0	- - - - - - - -
Read leg L2 sensors	0001	- F C L_U L_D P_2 P_1 P_0	- - - - - - - -
Read leg L3 sensors	0010	- F C L_U L_D P_2 P_1 P_0	- - - - - - - -
Read leg R1 sensors	0011	- F C L_U L_D P_2 P_1 P_0	- - - - - - - -
Read leg R2 sensors	0100	- F C L_U L_D P_2 P_1 P_0	- - - - - - - -
Read leg R3 sensors	0101	- F C L_U L_D P_2 P_1 P_0	- - - - - - - -
(Unused)	0110	- - - - - - - -	- - - - - - - -
Read antennae	0111	- - A_R A_L A_3 A_2 A_1 A_0	- - - - - - - -
Move leg L1	1000	- - - - - - - -	- - - U D P_2 P_1 P_0
Move leg L2	1001	- - - - - - - -	- - - U D P_2 P_1 P_0
Move leg L3	1010	- - - - - - - -	- - - U D P_2 P_1 P_0
Move leg R1	1011	- - - - - - - -	- - - U D P_2 P_1 P_0
Move leg R1	1100	- - - - - - - -	- - - U D P_2 P_1 P_0
Move leg R3	1101	- - - - - - - -	- - - U D P_2 P_1 P_0
(Unused)	1110	- - - - - - - -	- - - - - - - -
(Unused)	1111	- - - - - - - -	- - - - - - - -

Table 2 All possible robot commands. For commands from 0 to 7 the resulting input data is described. Also, for each command from 8-15, the required output data format is specified.

As for future mechanical changes, the entire robot would be reduced in size. Since the robot walks using only a tripod gait, then only two motors are actually required to achieve this. Consider a small tank-like robot with 3 wheels per side. The legs could be attached to each of these wheels such that they operate in a kind of stirring motion with three legs down at any one time. This reduction in motors would reduce most of the weight problems allowing the use of smaller motors, therefore reducing the overall size of the robot. With nanotechnology and VLSI technology, it is possible to create a very small robot with the performance of RABI. It is unclear, however, as to the reducibility of the map building and learning features of the software version of RABI. Thus, more experimentation needs to be done with the reducing of software, hardware and power requirements needed for learning.

8.5 Summary

The various versions of RABI provided insight into the mechanical and physical problems of legged robot design. As the robot evolved, the leg designs became more efficient and stable. The electronic circuitry required to control RABI is simple and inexpensive, and provided a suitable interface for robot / computer interaction. With the use of simple mechanical and electronic sensors, the overall robot is reduced in size and weight. Eventually, these sensors may be improved upon so as to reduce their size and increase their efficiency, resulting in an even lighter weight design. RABI' s physical design has the potential to be drastically reduced in size through the elimination of the computer interface. Such a reduction is a step towards smaller and smaller robots of the future.

~~~~~~

Appendix A

~~~~~~

A1 Spreading Activation

The following is a list of pseudo code routines required for spreading activation between the memory neurons (see section 6.5.2). The routines are used to spread activation such that a shortest path can be determined to multiple destinations.

```
InitiateSpreadingActivation($D_1$, $D_2$, $D_3$, ..., $D_n$)
   {
     "Initiate the spreading of activation from each of the given neurons"

     Reset all memory neurons to have zero activation.
     FOR each destination neuron $D_i$ DO
          {
            Set output of $D_i$ to 1.
            SpreadActivationAdjacent ($D_i$).
          }
   }

SpreadActivationAdjacent (N)
   {
     "Spread activation adjacent to neuron N"

     FOR all prev neurons $P_i$ of $D_i$ DO
          SpreadActivation (Pi)
     FOR all next neurons $N_i$ of $D_i$ DO
          SpreadActivation ($N_i$)
   }

SpreadingActivationFrom (N)
   {
     "Spread the activation from a neuron N"

     IF N is an edge neuron THEN
          SpreadActivationFromEdge (N)
     ELSE
          SpreadActivationFromCorner (N)
   }
```

SpreadingActivationFromEdge (N)

```
{
  "Update the activation of an edge neuron N and continue the activation spreading it adjacent"

  LET $P_n$ = previous neuron of N.
  IF DominatesEdge ($P_n$, N) THEN {
      IF output of $P_n$ > 0 THEN
          set output of N to (output of $P_n$ + storedEnergy of N) * -1.
      ELSE
          self output of N to (output of $P_n$ - storedEnergy of N).
      SpreadActivationAdjacent (N) }

  LET $N_n$ = next neuron of N.
  IF DominatesEdge ($N_n$, N) THEN {
      IF output of $N_n$ > 0 THEN
          set output of N to (output of $N_n$ + storedEnergy of N).
      ELSE
          self output of N to (output of $N_n$ - storedEnergy of N).
      SpreadActivationAdjacent (N) }
}
```

SpreadingActivationFromCorner (N)

```
{
  "Update the activation of a corner neuron N and continue the activation spreading it adjacent"

  LET $P_n$ = previous neuron of N .
  IF DominatesCorner ($P_n$, N) THEN {
      IF output of $P_n$ > 0 THEN
          set output of N to output of $P_n$ * -1.
      ELSE
          self output of N to output  of $P_n$.
      SpreadActivationAdjacent (N) }

  LET $N_n$ = previous neuron of N .
  IF DominatesCorner ($N_n$, N) THEN
      set output of N to output of $N_n$
  SpreadActivationAdjacent (N)
}
```

DominatesCorner (N_1, N_2)

```
{
  "Return whether or not $N_1$ has an activation that dominates $N_2$"

  IF output of $N_1$ is 0 THEN  RETURN (false).
  IF output of $N_2$ is 0 THEN RETURN (true).
  IF abs(output of $N_1$) < abs(output of $N_2$) THEN  RETURN (true)
  ELSE RETURN (false)
}
```

Dominates Edge(N_1, N_2)

```
{
  "Return whether or not $N_1$ has an activation that dominates $N_2$"

  IF output of $N_1$ is 0 THEN RETURN (false).
  IF output of $N_2$ is 0 THEN RETURN (true).
  IF (abs(output of $N_1$) + stored energy of $N_2$) < abs(output of $N_2$) THEN RETURN (true)
  ELSE RETURN (false)
}
```

A2 Generalization

The following is a list of pseudo code routines required for the basic matching algorithm that determines whether or not two layers of neurons match (see section 6.2.4).

Match (L_1, L_2)

```
{
  "Determine whether the two layers match"
  LET S = the first neuron of layer $L_1$.
  FOR each neuron $N_i$ of layer $L_2$ DO
      MatchFrom($N_i$,S)
}
```

MatchFrom (N_1, N_2)

```
{
  "Determine whether the two layers match from these two neurons"
  IF Matches($N_1$, $N_2$) THEN
      RETURN (MatchFrom ($N_1$ next , $N_2$ next)
  ELSE RETURN (false)
}
```

Matches (N_1, N_2)

```
{
  "Determine whether the two neurons match"
  IF $N_1$ is a different type of neuron than $N_2$  (i.e. corner and edge) THEN
      RETURN (false)
  IF the stored energy of $N_1$ is within an allowable error from the stored energy of $N_2$  THEN
      RETURN (true)
  ELSE
      RETURN (false)
}
```

References

[Anderson and Rosenfeld 89] Anderson, James A.; Rosenfeld, E., *Neurocomputing: Foundations of Research*, 1989, MIT Press, London.

[Anderson and Donath 90] Anderson, Tracy L.; Donath, Max, Animal Behavior as a Paradigm for Developing Robot Autonomy, *Designing Autonomous Agents*, 1990, MIT Press, Elsevier, pp. 145-168.

[Ayers and Crisman 92] Ayers, Joseph; Crisman, Jill, Biologically-based Control of Omnidirectional Leg Coordination, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 1, July 1992, Raleigh, NC, pp. 574-581.

[Beer 90] Beer, Randall D., *Intelligence as Adaptive Behavior*, 1990 Academic Press, London.

[Beer and Gallagher 92] Beer, Randall D.; Gallagher, John C., Evolving Dynamical Neural Networks, *Adaptive Behavior* , 1992, MIT Press, London, pp. 91-122.

[Bell and Adiyodi 81] Bell, W.J.; Adiyodi, K.G., *The American Cockroach*, 1981, New York: Chapman and Hall, pp. 372-376.

[Bernstein 67] Bernstein, N. A., The Co-ordination and Regulation of Movements 1967, Oxford: Pergamon Press.

[Booker et al. 89] Booker, L.B.; Goldberg, D.E.; Holland, J.H., Classifier Systems and Genetic Algorithms, *Artificial Intelligence*, Vol. 40, 1989, pp. 235-282.

[Borenstein and Koren 91] Borenstein, Johann; Koren, Yoram, Histogramic In-Motion Mapping for Mobile Robot Obstacle Avoidance, *IEEE Trans. on Robotics and Automation*, Vol.7, No.4, August 1991, pp. 535-539.

[Braitenberg 84] Braitenberg, Valentino, *Vehicles*, 1984 Bradford Books, MIT Press.

[Brooks 86] Brooks, Rodney A., A Layered Intelligent Control System for a Mobile Robot, *Third International Symposium on Robotics Research*, Vol. 3, 1986, MIT Press, London, pp. 365-372.

[Brooks 89] Brooks, Rodney A., A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network, *Neural Computation*, Vol. 1, No. 3, 1989, pp. 253-262.

[Brooks 91] Brooks, Rodney A., Intelligence Without Representation, *Artificial Intelligence*, Vol. 47, 1991, Elsevier, pp. 253-262.

[Camhi 84] Camhi, J.M., Neuroethology, 1984, Sunderland, MA, Sinauer Associates.

[Carmo 76] Carmo, M.P., Differential Geometry of Curves and Surfaces, Prentice-Hall, Eaglewood Cliffs, NJ, 1976, pp. 30-32, 267, 396.

[Carew 85] Carew, T.J., The Control of Reflex Action, *Principles of Neural Science,* 1985, E.R. Kandel and J.H.Schwartz, New York, Elsevier, pp. 457-468.

[Chiel et al. 92] Chiel, Hillel J.; Beer Randall.D.; Quinn R.D.; Espenschied K.S, Robustness of a Distributed Neural Network Controller for Locomotion in a Hexapod Robot, *IEEE Transactions on Robotics and Automation*, Vol. 8, No.3, June 1992, pp. 293-302.

[Cox 91] Cox, Ingemar J., Blanche - An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle, *IEEE Transactions on Robotics and Automation*, Vol.7, No.2, April 1991, pp. 193-204.

[Dario et al. 91] Dario, P.; Ribechini F.; Genovese V.; Sandini G., Instinctive Behaviors and Personalities in Societies of Cellular Robots, *Proc. of the 1991 IEEE Int. Conf. on Robotics and Automation*, April 1991, pp. 1927-1932.

[Dudek et al. 91] Dudek, Gregory; Jenkin, M.; Milios, E.; Wilkes, D., Robotic Exploration as Graph Construction, *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 6, Dec. 1991, pp. 859-865.

[Fylnn 87] Flynn, Anita M., Gnat Robots, *AI Expert,* Vol.2, No.12, Dec. 1987, pp. 34-41.

[Gallistel 90] Gallistel, Charles R., *The Organization of Learning*, 1990, MIT Press.

[Gibson 62] Gibson, J.J., Observations on Active Touch, *Psychological Review*, Vol.169, 1962, pp.477-491.

[Gould and Marker 87]	Gould, James L.; Marler, Peter, Learning By Instinct, *Scientific American*, Vol. 256, No. 1, Jan. 1987, pp. 74-85.
[Graham 85]	Graham, D., Pattern and Control of Walking in Insects, *Advances in Insect Physiology*, Vol. 18, 1985, pp. 111-114.
[Hochberg 68]	Hochberg, Julian, In the Mind' s Eye, Contemporary Theory and Research in Visual Perception, 1968, R.N. Haber, ed. New York: Holt, Rinehart and Winston, pp.309-331.
[Koza and Rice 92]	Koza, John R.; Rice, James P., Automatic Programming of Robots using Genetic Programming, *Proc. 10th National Conf. on A.I.,* San Jose, July 1992, MIT Press, London, pp. 194-201.
[Kweon et al. 92]	Kweon, I.; Kuno, Y.; Watanabe, M.; Onoguchi, K., Behavior-Based Intelligent Robot in Dynamic Indoor Environments, *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Vol.2, Raleigh, NC, July 1992, pp. 1339-1346.
[Lang et al. 89]	Lang, S.T.; Korba, L.W.; Wong A.K.C., Characterizing and Modelling a Sonar Ring, *Proc. of SPIE, Advances in Intelligent Robotics Systems: Mobile Robots IV*, Philadelphia, PA, Nov. 1989, pp. 291-304.
[Langton 89]	Langton, Christopher G., *Artificial Life VI*, 1989, Addison-Wesley, Santa Fe Institute.
[Lewis and Bekey 92]	Lewis, Anthony M.; Bekey, George A., The Behavioral Self-Organization of Nanorobots Using Local Rules, *Proc. of the 1992 IEEE/RSJ Int.Conf. on Intelligent Robots and Systems*, Vol.2, July 1992, Raleigh, NC, pp. 1333-1338.
[Linsker 88]	Linsker, R., Development of Feature-analyzing Cells and their Columnar Organization in a Layered Self-adaptive Network, *Computer Simulation in Brain Science*, 1988, New York: Cambridge University Press, pp.416-431.
[Linsker 90]	Linsker, R., Self-Organization in a Perceptual System: How Network Models and Information Theory May Shed Light on Neural Organization, *Connectionist Modeling and Brain Function*, 1990, MIT Press, London, Ch.10, pp.351-392.
[McComb 87]	McComb, Gordon, *The Robot Builder's Bonanza: 99 Inexpensive Robotics Projects*, 1987, Tab Books, Blue Ridge Summit, PA.

[Maes 91]	Maes, Pattie, A Bottom-Up Mechanism for Behavior Selection in an Artificial Creature, *From Animals to Animats: Proc. of the 1st Int. Conf. on Simulation of Adaptive Behavior*, 1991, MIT Press, London, pp. 238-246.
[Mahadevan and Connell 92]	Mahadevan, Sridhar; Connell, Jonathan, Automatic Programming of Behavior-Based Robots using Reinforcement Learning, *Artificial Intelligence*, Vol.55, 1992, Elsevier, pp. 311-365.
[Mataric 91]	Mataric, Maja J., Navigating With a Rat Brain: A Neurobiologically-Inspired Model for Robot Spatial Representation, *From Animals to Animats: Proc. of the 1st Int. Conf. on Simulation of Adaptive Behavior*, 1991, MIT Press, London, pp. 169-175.
[McGhee and Frank 68]	McGhee, R.B.; Frank, A.A., Some Finite State Aspects of Legged Locomotion, *Mathematical Biosciences*, Vol. 2, No.1/2, 1968, pp.67-84.
[Messuri and Klein 85]	Messuri Dominic A.; Klein, Charles A., Automatic Body Regulation for Maintaining Stability of a Legged Vehicle During Rough-Terrain Locomotion, *IEEE Journal of Robotics and Automation*, Vol. RA-1, No.3, September 1985, pp. 132-141.
[Mitchell 89]	Mitchell, Joseph S.B., An Algorithmic Approach to Some Problems in Terrain Navigation, *Geometric Reasoning*, 1889, MIT Press, pp. 171-201.
[Nehmzow and Smithers 91]	Nehmzow, Ulrich; Smithers, Tim, Mapbuilding Using Self-Organizing Networks in "Really Useful Robots", *From Animals to Animats: Proc. of the 1st Int. Conf. on Simulation of Adaptive Behavior*, 1991, MIT Press, London, pp. 152-159.
[Pal and Jayarajan 90]	Pal, P.K.; Jayarajan, K., A Free Gait for Generalized Motion, *IEEE Transactions on Robotics and Automation,* Vol. 6, No. 5, Oct. 1990, pp. 597-600.
[Rosenblatt 58]	Rosenblatt, F., The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, *Psychological Review*, Vol. 65, 1958, pp.286-408.
[Rosten and Krotkov 92]	Roston, Gerald P.; Krotkov, Eric P., Dead Reckoning Navigation for Walking Robots, *Proc. of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, NC, July 1992, pp. 607-612.

[Song and Waldron 89]	Song, Shin-Min; Waldron, Kenneth J., *Machines That Walk*, 1989, Mit Press, London, Ch. 3.
[Staddon 83]	Staddon, J. E. R., *Adaptive Behavior and Learning*, 1983, Cambridge University Press, Cambridge, Ch. 2 & 3.
[Steels 90]	Steels, Luc, Exploiting Analogical Representations, *Designing Autonomous Agents*, 1990, MIT Press, Elsevier, pp. 71-88.
[Toates 86]	Toates, Frederick M., *Motivational Systems*, 1986, Cambridge University Press, Cambridge.
[Todd 85]	Todd, D.J., *Walking Machines: An introduction to Legged Robots*, 1985, Kogan Page Ltd, London.
[Tyrrell and Mayhew 91]	Tyrrell, Toby; Mayhew, John E. W., Computer Simulation of an Animal Environment,*From Animals to Animats: Proc. of the 1st Int. Conf. on Simulation of Adaptive Behavior*, 1991, MIT Press, London, pp. 263-272.
[Wilson 66]	Wilson, D.M., Insect Walking, *Annual Review of Entomology* Vol. 11, 1966, pp.103-122.
[Wilson 85]	Wilson, Stewart W., Knowledge Growth in an Artificial Animal, *Proc. of an Int. Conf. on Genetic Algorithms and Their Applications,* 1987, Pittsburgh, PA., pp.16-23.
[Wilson 87]	Wilson, Stewart W., Classifier Systems and the Animat Problem, *Machine Learning,* Vol. 2, 1987, pp.199-228.
[Zelinsky 92]	Zelinsky, Alexander, A Mobile Robot Exploration Algorithm, *IEEE Transactions on Robotics and Automation,* Vol. 8, No. 6, December 92, pp. 707-717.