Reactive Planning in a Fully Embodied Robot Arm

An honors project in Computer Science

By Daniel C. Churchill

Advised by Susan Fox

Macalester College

May 5, 2000

Abstract:

This paper describes the application of reactive planning techniques for controlling the motion of a two-dimensional robot arm, given the objective of finding and touching a light source. I compare reactive planning techniques with “traditional” artificial intelligence planning techniques, which historically have been unacceptable for operation in real-world environments. I present examples of systems that are essentially a hybrid mixture of traditional and reactive techniques. Finally, I describe the implementation of pure reactive planning on a robot arm controlled by a single Handy Board 68HC11-based controller. The results show that reactive planning is a viable alternative to traditional planning.

Acknowledgements

This project has been a long and somewhat arduous journey for me. And, considering that it will likely be sometime before I have the opportunity to make a statement like this again, assuming that I ever do, I want to recognize the people who, whether realizing it or not, have helped me along the way.

First and foremost, I want to thank my adviser, Susan Fox, for her patience in dealing with my tendency to procrastinate, and for all her help and suggestions along the way, most of which found their way into this project in some form or another. I would also like to thank my second and third readers, Dick Molnar and Tom Halverson, for their willingness to take the time to read and critique my work. Additionally, I would like to thank Jim Doyle in the Physics department, for letting us amateurs play with the soldering iron in the digital electronics lab when my sensors needed longer wires.

I want to thank my family and friends for their continued support and encouragement over the past two semesters: Mom and Dad, for providing a little extra money this year so that I didn't need to devote as much time to working for pay; the Vortex, for just being the Vortex; Molly for being the one who set an example work ethic that I should have followed, but didn't; Lynn for helping me to keep my priorities straight and for praying for me; TJ for his wacky commentary, as well as his prayers; Deb Kerkvliet for encouraging me through her own experience to not give up when deadlines seemed impossible to meet; Mikkel and Vivek for always being around to remind me that I wasn't in any worse shape than any other honors student in the Computer Science department; Toni at the Macalester switchboard for her role as "surrogate mother," as well as the rest of the crew in CIT; and any others who at some point said an encouraging word – you are too numerous to mention, but you know who you are.

Finally, I have one more acknowledgement, and that is to God, for seeing me through the long hours when no one else was awake, and for walking with me through the easy times, and carrying me in the difficult ones. To Him be all thanks and praise.

- DC

1 Introduction

This project grew out of a tour of the local Ford Motor plant in St. Paul as part of a managerial accounting course. Though it had nothing to do with what we were there for, I was struck by the complexity of the many robot arms that were in use in the plant for the various stages of building Ford trucks. It was obvious to me that a human somewhere had carefully programmed the arms; every motion, no matter how complex, was designed and performed with extreme precision. Welds that would be useless if the arm had been positioned even a millimeter one way or the other were performed perfectly time and again. Of course, these arms also had one other glaring characteristic: they were dumb. The only reason that they made those welds with such precision was that they had been programmed and calibrated to be exactly where they were, and the vehicle on the conveyor belt had been placed and held there with equal accuracy.

This illustration of industrial robots in action started me thinking about how much more effective such systems could be if they were able to exhibit intelligence, even on a level that might be called trajectory error-correction. In the case of the Ford plant, an arm coming toward a vehicle that was slightly misplaced on the conveyor could still be welded perfectly if only it were able to sense where the joint was and adjust its position accordingly. Of course, like nearly everything else in artificial intelligence, making that adjustment is far more difficult than it would at first seem that it should be. This paper compares and contrasts the two major approaches to planning: *Sense-Model-Plan-Act,* or SMPA, (commonly referred to as "classical planning") and a newer method called *Reactive Planning*.

The remainder of this paper discusses the implementation of a reactive planning system on a two-dimensional robot arm constructed of Lego blocks.

2 Background

As mentioned above, there are two main approaches to planning: Sense-Model-Plan-Act (SMPA) and Reactive Planning. SMPA is a much older approach, and is therefore frequently referred to as classical planning. In this model, the computer typically has a complex internal representation of the world. It generates the plan within this representation, and then issues the motor commands to get the robot to affect its world. Reactive planning systems have no complex internal representation of the world, though they may retain some state information. Its actions are based purely on the current state of its sensors and any internal state that the robot may be keeping.

Given these basic definitions, we can look at each planning framework more in-depth.

2.1 Sense-Model-Plan-Act Framework

The history of planning in artificial intelligence has tended to focus on getting the computer to reason out a solution for achieving a goal (Brooks 5). The emphasis has been placed on creating techniques for getting the computer to actually "think up" a solution. In particular, a common assumption was that getting the actual robot to perform the action was an engineering problem completely independent of planning the action.

This plan of attack resulted in the creation of systems that first needed to generate some sort of internal representation of the space in which they existed, then execute some sort of planning algorithm based on the state of that representation, and finally carry out the plans in the actual world. The STRIPS planner, used on Stanford Research Institute's Shakey is a prime example of just such a system. Shakey used a black and white television camera as its primary sensor, with image processing occurring on an offboard computer to convert the visual data to a first order predicate calculus model of the world. STRIPS then worked on this model and the plans were translated into calls to "atomic actions in fairly tight feedback loops," (Brooks, 8).

In at least some cases, researchers did not have the funding to create an actual robot. This resulted in the creation of simulators, which basically took into account only the execution of the planning algorithm. The results of the plan could only be applied to the internal representation, so there was no need for the conversion from a real-world situation to the internal representation. Nor was there a need to execute instructions to have motors move the robot to the state specified by the plan. Typically, this type of research relied on experimental results that demonstrated the feasibility of the input they assumed was available and the actions they expected the system to be capable of. An example of this is the work of Terry Winograd, who created programs to work in the blocks world after Patrick Winston had purportedly shown that it was possible for a system to perceive and manipulate such a world (Brooks 9).

Researchers working with the SMPA model addressed the problem of environmental modeling in two ways. The first was to assume a static (or mostly static) environment. The logic behind this decision was that current computers could render a representation of a static environment. Since computers of the future were continuing to get faster, it was felt to be only a matter of time before computers would be fast enough to perform enough static renderings in such a small amount of time that you could use them in a dynamic environment.

The second method for dealing with the computationally expensive environment rendering operation was to use off-board computers. This allowed the most powerful computers to be used. Thus, on-board computation was basically reduced to the collecting and sending of sensor data to the mainframe and the receiving and issuing of motor commands, as the majority of this research was done in mobile robotics. However, despite the use of the most powerful computers available *and* a robot situated in a static environment, SMPA systems still performed "excruciatingly slowly" (Brooks 2).

The major problems with the SMPA framework can be summed up as follows:

- **Sensing, modeling, and affecting the environment are hard and computationally expensive (Brooks 2).** A robot that is trying to respond to its environment in real time would have to update its model continuously, requiring tremendous amounts of computation, and in all but very specialized cases, *on-board* computation. In most cases, this computation is unnecessary, as the system is only concerned with a certain small portion of its environment, much as the human eye only "sees" a small portion of the entire field of vision. Similarly, changing a desired outcome into effector commands is an uncertain process due to the imperfection of servomotors, etc.
- **SMPA suffers from the horizon effect (Brooks 3).** The algorithms that SMPA uses are most-frequently based on traditional AI techniques, such as state space

search or problem decomposition (e.g. see chapters 11 and 12 of Russell and Norvig). While these algorithms are extensively researched and in some cases can be highly optimized, they make assumptions, see Table 1 below, that are very unrealistic for a robot acting in a real environment. The bottom line is that no matter how fast the processors of the future become, the basic limitations of SMPA will not be overcome. To render a system in operable simply requires situating the robot in an environment that is too complex and/or dynamic to allow an SMPA system to keep up.

2.2 Reactive Planning

Classical planning systems were virtually the only type of planning system that was in use up until the mid to late 80's. At this time, Rodney Brooks developed the subsumption architecture, which set up various different “behaviors” that were achieved by performing actions based upon sensor input, and possibly a small amount of internal state information. It is these behaviors that form the heart of the reactive planning model. Behaviors ideally are very simple responses to various input states. An example of a simple behavior, in the context of a mobile robot, might be to turn to the left as the result of running into an object that triggered a sensor on the right side of the robot.

One of the key principles of reactive planning is that by combining enough simple behaviors together, complex responses to the environment will emerge from the robot’s interactions with the world (Brooks, 3). It is through these interactions with the world that the intelligence of the robot will (or will not) emerge. The idea here is that the intelligence of the robot comes not from the method that it used to determine its actions, but from the results that those actions achieved in the world. Essentially, this is the key difference between reactive and classical planning, which relies on a more purely intellectual solution provided by a resolution theorem prover or some other method.

Following from this principle that reactive planning displays intelligence through its interaction with the world is the principle that it must be a fully embodied system. The main argument here is that it is impossible to have a system that is responding to world unless it has a body for the world to act on and to respond to the world with. In particular, simulated systems will not do, because it is impossible to truly represent the idiosyncrasies of the world in a simulated system.

In Brooks' research, the combination of behaviors was achieved using combinatorial circuits connected to a multi-layered network. This network was designed in such a fashion as to filter incoming sensor information and produce a combination of behaviors (Brill 7). Brooks' approach came to be known as reactive planning. The resulting system maps input to output, where the output can be modified by the internal state.

We can talk extensively about the characteristics of reactive planning, but an exact definition is difficult to pin down. For example, it is somewhat debatable whether a system should be called a reactive system if it maintains internal state, since it then is not acting solely on sensor information. I would suggest that the following characteristics are the two that are most important for distinguishing a reactive planning system from a classical planning system:

- **The system should not be concerned with maintaining a specific model of the world** as is done in the SMPA method, but it may have state (Brooks 17).
- **The system needs to be able to deal in and with the real world.** This means that the system must be embodied, because it must be able to effect change in its world, not just in an abstract internal representation (Brooks 3).

2.3 SMPA and Reactive Planning in an Example

Suppose that we have an arm consisting of two segments, functioning within a plane, as shown in Figure 2-1 on the next page. The goal of the system is to move the arm from its original position to the final position, touching the goal, represented by the large dot.

2.3.1 SMPA in the Example

The first thing to note about SMPA is that it would maintain an explicit representation of the arm's world. In this case, it could be as simple as representing objects by their location in the coordinate plane. SMPA would start by taking a sensory reading of its environment and mapping it into this representation. For this example, let's assume that the arm has some type of ranged sensor, perhaps visual input from a camera

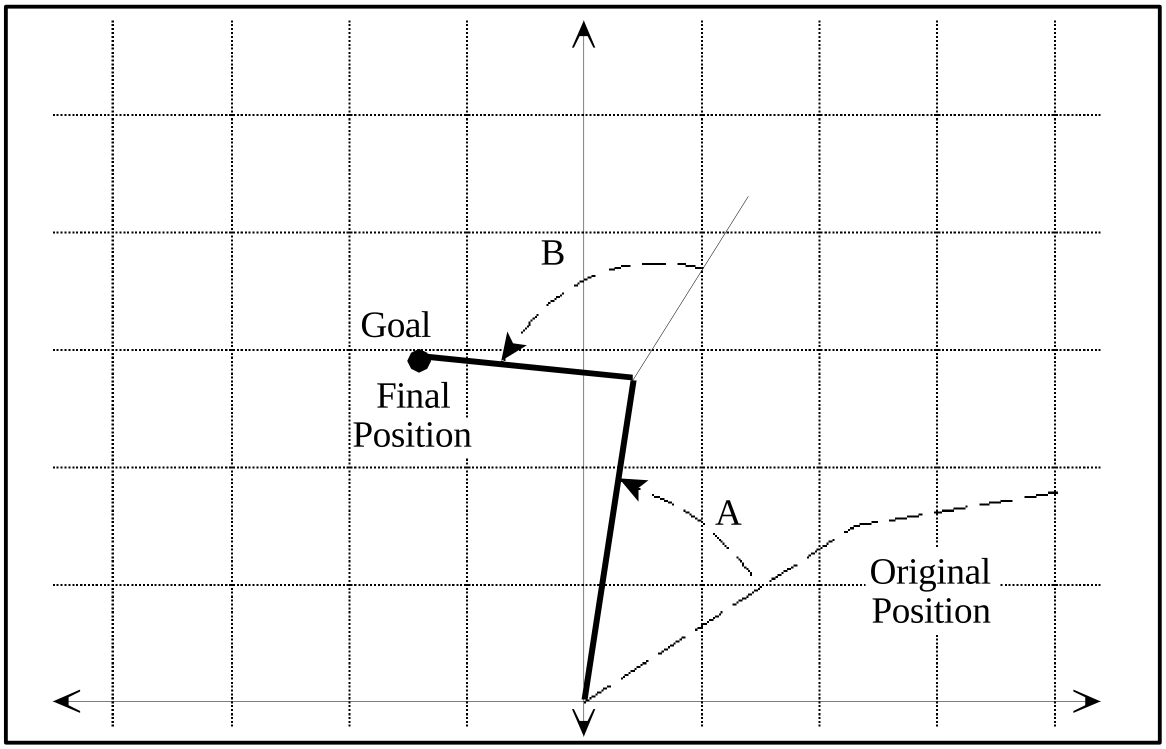


Figure 2-1 Example of the desired motion of a two-dimensional arm.

or sonar readings. Presumably the arm has sensors, which allow it to know its precise position.

After the SMPA system has put this information into its world model, it will execute some sort of planning algorithm. This could be something like state space search, where it would look at all the possible actions it could take and evaluate the results of each, choosing that which best satisfies the desired goal. Assuming that some sort of planner is in place that can do this, the design of which is difficult in its own right, the system is going to return a set of actions for the arm to take.

In this case, those actions are to rotate the base joint **A** degrees counterclockwise, and to rotate the outer joint **B** degrees counterclockwise. The final step is to map the planner's output to actual commands, which will cause motors to run just the right amount of time and in the right direction to get them to move the arm precisely to the desired location.

Now let us evaluate where problems could occur in this sequence of events. The first problem is that the planner is assuming the sensory input was perfect. This may be the case, but in most situations, sensors are not this accurate. So, the first problem is that the goal probably isn't quite where we think it is, even though we have to assume that it is for purposes of planning our moves.

Another problem is that the translation from the sensor readings to the world representation needs to be fast. As mentioned before, in a real-world environment, this is not necessarily (as well as necessarily not) the case. This means that it is going to take longer for the system to come up with a plan, and in that time, it's possible that something other than the arm has changed the world. Thus, the assumption that we're making here is that the only thing affecting the world of the robot is the robot itself.

The final problem that I will mention with regards to SMPA in this example is that it assumes the mapping from desired output to motor commands is efficient, and perfect. The technology does exist that allows precise sensing of the angle that a joint is held at, but this technology is going to cost. A system that didn't have such sensors would have to rely solely on timed motor commands from where it thought it was before. This is problematic at best, because motors are physical devices, and their exact output in a given time period may change depending on such factors as the weight of the arm, the level of charge in the system's batteries, etc.

The bottom line is that SMPA is plagued by problems associated with assumptions that it makes regarding what it knows about the environment. This is a very simple world model, a very simple set of possible actions, and so on. Even given this simplicity, the SMPA model of planning has many shortcomings that make its use in such an environment nearly impossible.

2.3.2 Reactive Planning in the Example

Reactive planning, on the other hand deals quite well with most of the problems that SMPA has. In our example, the reactive planner would take a sensory reading of its environment. Remember that it has no internal representation of that environment. Assuming again that we have some sort of ranged sensor, the reactive planning system would "see" that there is something resembling the goal over to the left. So, it would immediately begin to move in that direction. However, it would be taking sensory readings the entire time, and would adjust its movement to follow the goal, even if the goal were to move.

The important difference here is that the reactive system has no concept of how far it has to move. Thus, any problems that we may have had with the motors not moving the arm the expected number of degrees is not a problem. There is no internal representation of the world, so there is no problem if the world changes during the process. As far as the reactive system is concerned, *the world is the model*, and all updates to its state are implicit in the events occurring in the world.

SMPA	Reactive Planning
Omniscience – the system has complete information	*Ignorance* – the system has limited information
Certainty – all information is absolutely true	*Uncertainty* – sensor readings are suspect, action may or may not work
Consistency – none of the information is contradictory	*Inconsistency* – different behaviors have different ideas about what to do
Sole cause of change – the system causes all changes	*Dynamic environment* – the world is always changing, not necessarily because of the agent, or as the agent intended
Atomic time – exactly one indivisible action occurs at a time	*Continuous time* – actions may be aborted at any time

Table 2-1 Assumptions of SMPA vs. Assumptions of Reactive Planning (adapted from Brill 5, 8)

Table 2-1 lists the major assumptions of SMPA and reactive planning. It is from these assumptions that the shortcomings of SMPA for robotics become apparent. It is also from these assumptions that we can see exactly why it is that SMPA tends to do well in a completely computerized environment, where it is not unreasonable to assume that we know all of these things.

2.4 Planning Practically – the Hybrid Approach

To this point, I have discussed the two extreme models for planning. Depending on how well your application fits the assumptions made by each of these, a system that implements pure SMPA or reactive planning may be the best choice. Practically speaking, these two are often combined into a hybrid planning system that implements some or all of the features of both methods. Though the work that goes into any of these hybrid approaches is itself very complicated, a brief overview of a couple of approaches that have been worked on will give the flavor of this hybrid approach.

2.4.1 Hybrid Approach 1

In his article "Design of a Reactive System Based on Classical Planning," John Bresina presents a system that uses the classical techniques of problem reduction and state space search to modify current behaviors and add new ones to improve the overall behavior of the system.

The basic idea of this work is to take the initial problem and break it down into *primitive subproblems* for which a *primitive reactive policy* is defined. How the problem is broken down is determined by the classical planning technique of problem reduction. After this breakdown is completed, the primitive reactive policies for each of the subproblems are combined to form a *non-primitive reaction policy* for the original top-level problem.

The system has other provisions that allow it to do a certain amount of refining of non-primitive reaction policies to avoid making choices that necessarily lead to failure, but we have enough here to see the how the hybrid system is functioning. Bresina is using the classical planning technique of problem reduction as the means to determine what behaviors are activated. Essentially, he is using problem reduction to perform the same function that Brooks' multi-layered network was providing.

2.4.2 Hybrid Approach 2

Another example of a hybrid system is called Reaction-First Search (RFS). RFS combines a reactive planner with a classical planner performing standard search space algorithms. The main point of this system is to allow the system to start performing an action before the search has completed (i.e. before the classical planner has been able to develop a complete plan). This is particularly important for a situated robot that has to start doing something before a complete plan is available.

RFS works by having the classical planner generate a partial plan from the problem search space. The reactive planner allows this partial plan to be executed, and then uses reactive policy to control the system. If at any time a partial plan is unavailable, the reactive planner will do the best it can, based on its current information (Drummond, Swanson, Bresina, and Levinson).

2.4.3 Final Comments on the Hybrid Approach

The work that is being done in the arena of hybrid planners is very complex, and there seems to be no generally accepted best way to combine the two types of planning. Indeed, the best way will depend on the variables involved with the problem to be solved, such as the environment of the system, the information available to it, and what the ultimate purpose of the system is. I have attempted to present a very brief overview of a couple specific examples, yet the researchers involved in these projects would probably shudder at the oversimplifications that I have made. Nevertheless, I think that we can get a feel for the general approach of combining reactive and classical planning.

3 Implementing Reactive Planning in a 2D Robot Arm

The main thrust of this project was not to simply research what other people have done in reactive planning, but to implement it on a robot arm. Unfortunately, much of the work that is done in reactive planning is done in mobile robotics, and not in the domain of stationary robot arms. Presumably this is because of the engineering complexity of robot arms and the resulting difficulty and expense involved in actually building one. Another reason could be that the technology available for robot arms, if one is willing to spend enough, allows them to be positioned at desired angles very precisely, eliminating much of the uncertainty with the exact position of the robot's body and making it a less interesting case for planning research. Whatever the reason for the scarcity of information on and designs for robotic arms, the fact remains that I could find no inexpensive kits or designs for a robot arm. Therefore, the first part of this project was concerned with what is basically an engineering problem – the design and construction of a robot arm.

3.1 Design of the Arm

3.1.1 Engineering Issues

A large portion of this project consisted of the raw engineering problem of developing a viable design for the arm. I chose to use Lego building blocks for the simple reason that they were the only viable resource for such a task that was available to me at Macalester College. The use of Legos required the construction of a fairly large arm, because the motors and gears that drive the arm need a sizable frame to be attached to. The motors themselves are relatively large, and need to be placed near the joint that they control. The size and weight, as well as the desire not to make this strictly an engineering project, dictated that this should be a two-dimensional arm that moves across the surface of a table. I designed a basic frame that could be replicated multiple times and connected in a chain. To keep the system complex enough to be interesting, but simple enough to be practical, I created and connected two of these frames together. See Figure 3-1 below.

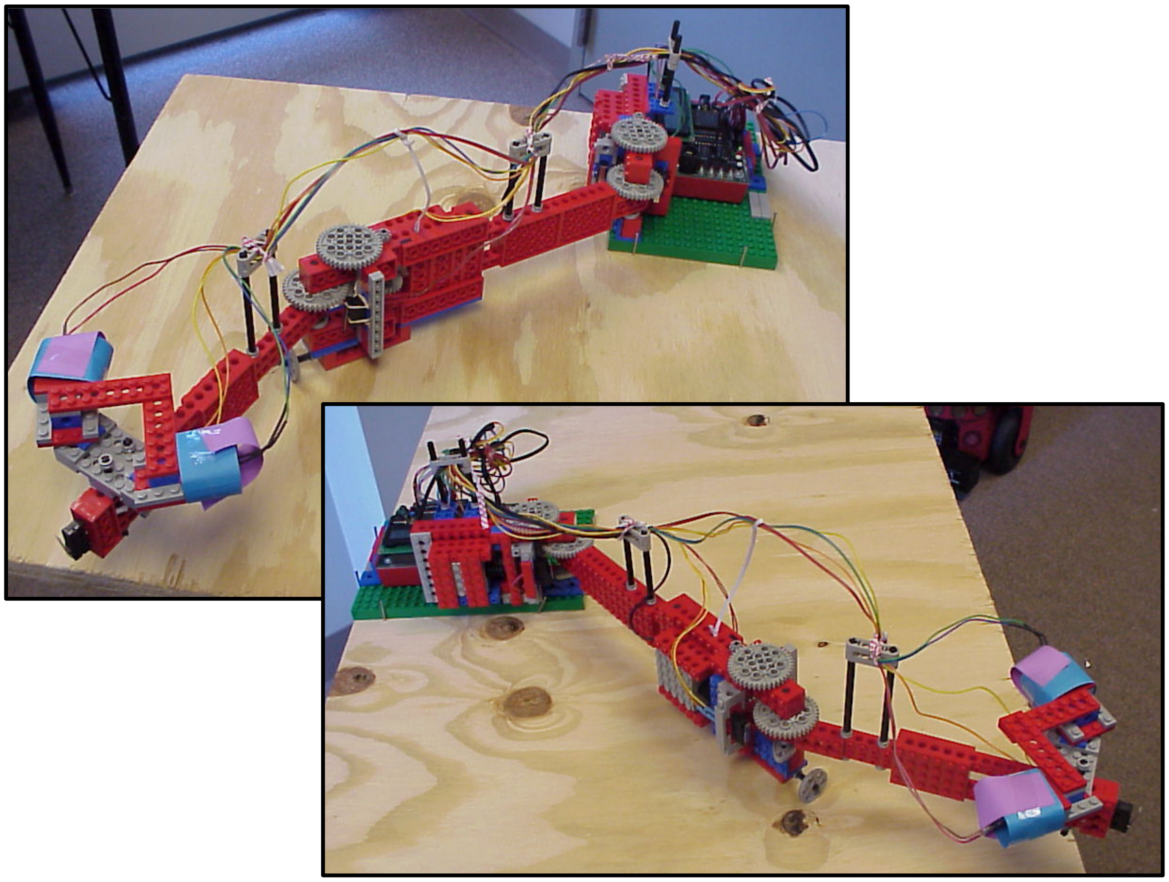


Figure 3-1 The robot arm built for this project

3.1.2 The Controller

The arm is controlled by a Motorola 68HC11 processor, running in a Handy Board controller developed at MIT. This controller was chosen because it was the only one available at Macalester College. These controllers are not easily networked with one another, so a single controller ran the arm. This board is limited to 4 motor controllers, 8 digital (i.e. touch) sensors, and 8 analog (i.e. light) sensors. This was not a limitation for me, although in a more realistic system (i.e. a system built for an application other than pure research, at considerably higher cost), there would be a need for many more than this.

3.1.3 Sensors and their limitations

The sensors available to me for use with the MIT Handy Board were limited to light sensors and touch sensors. I will discuss the uses and limitations of each of these below.

3.1.3.1 Touch Sensors

The arm has five touch sensors. One is located on the extreme end of the arm (see Figure 3-5 below.) It is used to determine when the arm has achieved its goal (see "Goals of the Arm" under "Theoretical Issues" below.) The other four sensors are split equally between the two joints. Their purpose is to sense when either joint is bent as far as possible to either the right or the left. These sensors provide the only sense of its own position that the arm has. They are absolutely essential, if for no other reason than to prevent the burn out of a motor or the self-mutilation of the arm when a joint has bent itself as far as is physically possible. See Figure 3-2.

This provides only a bare minimum of positional sensors, and in a sense represents the situation of operating with as little internal state information as possible. By including more sensors at each joint, it would be possible to sense certain other positions of the arm, such as when it is halfway between its full right and full left positions. Such information would be useful for providing more complex behaviors, but is unnecessary for basic functionality. In addition, such sensors would have been difficult to implement on the structure I designed. Thus, I chose to go with only two sensors at each joint, leaving myself with more limited information about the state of the arm.

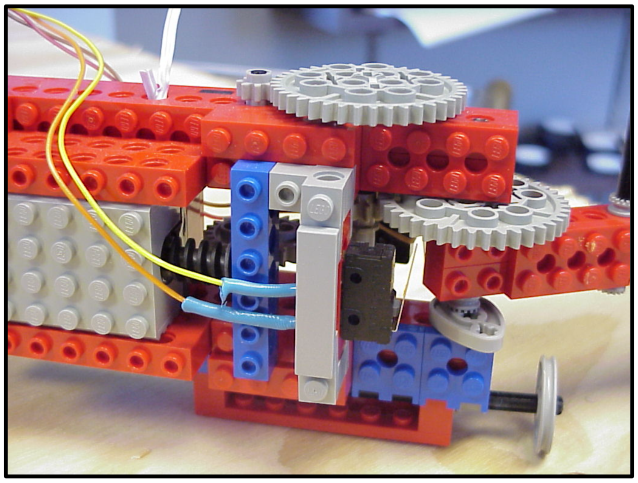
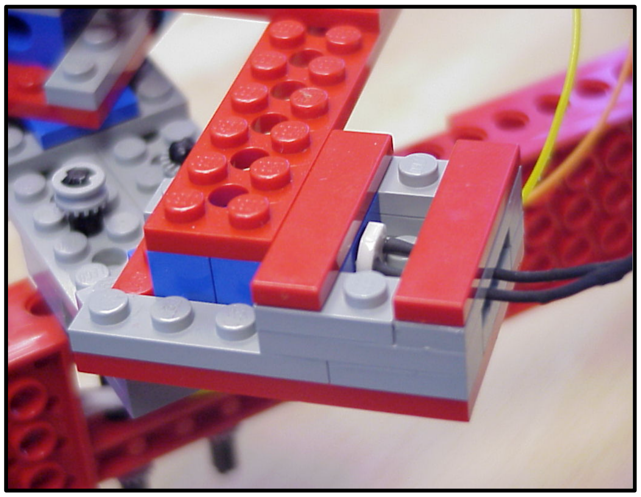


Figure 3-2 Close-up of one of the joints, showing the touch sensor that senses motion all the way to the right.

3.1.3.2 Light Sensors

Figure 3-3
Light sensor enclosure with top removed to show the light sensor inside.



The arm has two light sensors. They are mounted just behind the end of the arm and are responsible for controlling the actions taken by the "elbow" joint of the arm. Because of the high sensitivity of the sensors, the small enclosures in which they are contained have only a small, deep hole for light to pass through, as shown by Figure 3-3 and Figure 3-5. This design makes the light sensitivity highly directional, so that the sensor can only receive light from directly in front of it. This has both good and bad effects for how the sensor performs. The major benefit is that it gives definite, strong readings when a light source is lined up in front of it. However, this is also the greatest weakness. When the sensor is not lined up in front of the light source, we would like it to have a wide viewing area in order to pick out the most likely direction of the light source that we are looking for.

Figure 3-4 Light Sensor Placement (view from top)

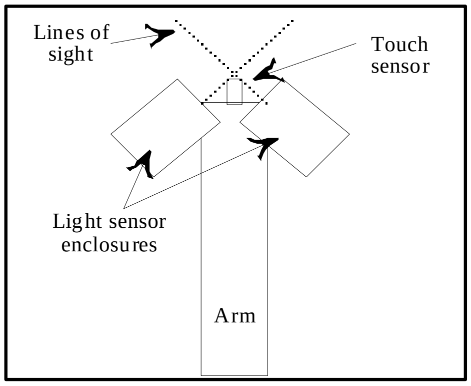
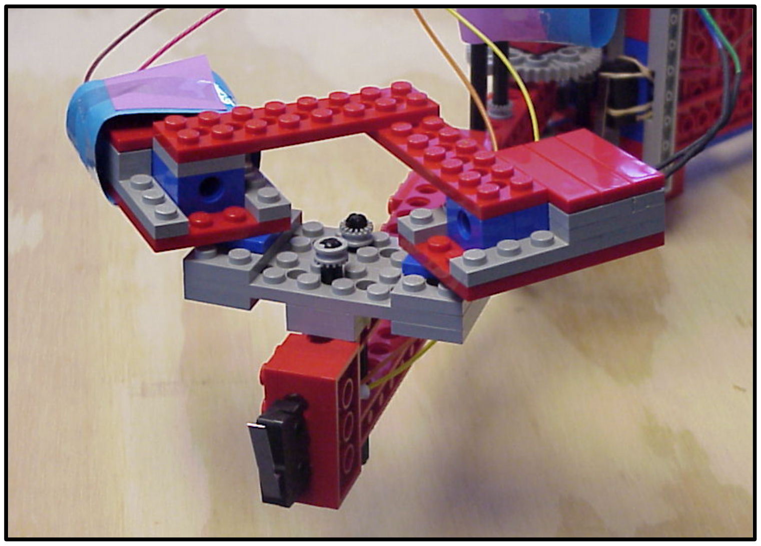


Figure 3-5 Light sensor enclosures and goal sensor



In placing the light sensors on the arm, I tried to utilize the directional capabilities to the fullest. The sensors are placed at right angles to each other and placed behind the end of the arm so that the point where their lines of sight cross each other is directly over the touch sensor. See Figure 3-4 and Figure 3-5.

There were several problems when dealing with the light sensors. The light sensors used with the Handy Board cause electrical resistance when exposed to light. The board puts a current out on one wire, and measures how much current returns on the other wire. Thus, if the sensor is exposed to a high level of light, the Handy Board will report a sensor reading near 0. If the sensor is in near-total darkness, it will report a much higher value. The first problem is that this “much higher value” varies significantly from sensor to sensor. This fact should not be allowed to cause a problem with our system, however, since reactive planning assumes the presence of imperfect sensors (Brill 8). One of the major challenges for a designer of a reactive system, as I discovered, was to get the robot to function in a desirable manner even with these finicky sensors.

The other main problem with these sensors is that they are extremely sensitive. This sensitivity has the undesirable effect of washing them out at a relatively low level of light. The enclosure design that I have created helps with this to a large extent, but the enclosures need to have a small opening in the back for the sensor wires, which allows a small opening for light to seep in. In addition, the Lego blocks are made of plastic, which, while not translucent per se, still allows a certain measure of light to seep into the enclosure and affect the reading of the sensor. To help combat this seepage of light into the enclosures, I found it somewhat helpful to surround the back portion of the enclosure with a piece of dark paper.

3.1.4 Validity of Hardware for Testing Reactive Planning

In terms of the hardware used, this system is not very advanced. The sensors, particularly the light sensors, are not very reliable, nor are they consistent with each other. The touch sensors are not as much of a problem in terms or reliability, but they are very limited in what they can tell you. In my case, they can only tell the system that a joint is completely bent one way or the other.

The motors on my system are a source of some uncertainty. The Lego motors spin quite fast, and thus it was necessary to use gearing to reduce the speed of the arm to a reasonable rate. This is a straightforward process of having small gears drive large gears.

Every time there are two gears that meet, there is a small amount of give, so that a change in direction of the arm isn't instantaneous. Also, the arm may move different distances depending on the charge of the controller's, how heavy the arm is, and whether it is located on a completely level surface or not. As a result of these characteristics, timed motor commands are virtually impossible to use, because you can never be certain how far it has moved at a given time. The actual distance moved will depend on the environment and the circumstances of use (e.g. has the arm been running for a long time and worn down the batteries.)

Even though the sensors and construction of this arm are all relatively simple, it is still entirely realistic to test reactive planning using it. This is precisely because reactive planning is supposed to deal well with all of these issues. Its assumptions include the expectation that sensors are going to frequently give incorrect and/or imprecise results and that its actions may or may not be carried out perfectly or in a timely fashion. Referring back to Table 2-1, these assumptions all fit into the uncertainty assumption of reactive planning. Therefore, this design is not only acceptable for testing the viability of reactive planning, but is actually desirable, precisely because a functional reactive planning system will be able to perform acceptably in spite of these limitation.

3.2 Theoretical Issues

3.2.1 Goals of the Arm

One of the earliest decisions I had to make while designing the arm was what I wanted it to do. The Lego blocks, processing abilities of a single Handy Board, and types of sensors available to me (touch and light sensors) dictated that the goal of the system needed to be fairly simple. Consequently, the goal of my system is simply to find and touch a light-emitting object that is within its reach. The assumption of a goal is not detrimental to the legitimacy of the reactive planning model. Brooks cites the work of Maja J. Mataric as an example of how reactive systems can and do implement goals and plans, even though they look nothing like the goals or plans of traditional AI (Brooks 20).

3.2.2 Programming issues

The programming of Handy Board controller is done in Interactive C, which is a subset of the C programming language, augmented by motor and sensor commands, as well as a few commands for spawning parallel processes. The Handy Board has only 32K of RAM for operating system and program code, so Interactive C offers a necessarily small set of features. However, it was designed primarily for research, so the features it does allow are useful and not particularly limiting for the purposes of research.

The Handy Board is capable of running multiple simultaneous processes. This is achieved by giving each process a slice of the processor's time in which to run. By having separate processes managing different behaviors, the Handy Board allows me to simulate the layered network of Brooks' model, providing an accurate rendering of reactive planning. With each parallel process controlling one particular behavior, the system's output is a combination of all the behaviors of the system, performing the same task as Brooks' layered network.

3.2.3 Modeling the Reactive Planning Model

Having determined the goal of this system, and realizing the restrictions and capabilities of the Handy Board, the main problem is to determine what behaviors the system should have. Since our goal is to touch a light source and the only sensors available to us are touch sensors and light sensors, the behaviors available are fairly limited. I chose to implement three behaviors that seemed to make the most sense given the capabilities of the arm I designed.

3.2.3.1 "Blind human" behavior

The first behavior utilizes the touch sensors on the arm. It is to blindly wave the arm about in all directions and hope that it touches its goal in the process. At first this does not seem like intelligent behavior, but if you think about it, it is exactly what a human who cannot see will do. Of course, humans usually have the added benefit of having a much more refined sense of touch, as well as the abilities to hear and smell.

I have implemented this "blind human" or "waving" behavior into the base joint of the arm. Every few seconds, the base joint will move one direction or the other. It starts by going left until it has gone as far as possible. Then it will make its way back around to the right. If it makes it all the way to the right, it reverses and goes back to the left. It will repeat this process ad infinitum or until a the goal state is achieved.

There is at least one practical problem with this behavior for this robot. The touch sensor is located only at the very tip of the arm, so that the goal is achieved only if the tip of the arm touches the goal, which is very unlikely. This is one aspect of how the human sense of touch is more refined, because, in essence, humans have touch sensors along every point of their arm and not just at the tip of their index finger. Even if a blind human wants to touch something with their index finger, they can use the sensory information from touching something elsewhere along the arm to help guide the index finger to it. Since that kind of information is not available to the arm, we need some better, more refined way to locate the object we want to touch.

3.2.3.2 "Seek the light" behavior

The second behavior utilizes the light sensors on the arm and provides us with a way to locate the light source without simply moving the arm and hoping we just happen to run across it. The basic behavior is this:

- Take light sensor readings from the left and right light sensors.
- Compare the readings to see which side has brighter light (that is the direction in which we want to move the arm).
- Move the arm in the direction of stronger light, moving the arm more slowly when the left and right sensor readings are closer together.

These steps are deceptively simple because of the problems with the light sensors mentioned above in Section 3.1.2.2. The main problem is that the output values of different light sensors are different at the same light levels. To standardize the input of the light sensors, I have hard-coded a simple mathematical procedure for returning the "percentage" of darkness that the light sensor perceives. The Handy Board is not capable of type casting operations. Our sensor readouts are integer values, so the percentage needs to be found with integer calculations only. The calculation consists of simply taking the current sensor readout, multiplying it by 100, and dividing by the "maximum" value that a sensor ever returns:

Equation 3-1

$$\boxed{\text{Normalized Sensor Output} = \frac{\text{Current Sensor Reading} \times 100}{\text{Sensor Maximum Value}}}$$

I have placed "maximum" and "percentage" in quotation marks because the value need not represent an absolute maximum for the sensor, but rather is intended to be taken only as a readout that is typical of very high sensor readings, which correspond to very dark environments. Thus it is possible for the "percentage" to be over 100. This is not problematic because these readouts rely only on the relative values of the two sensors, not on the actual numerical values of the normalized output.

Getting the normalized sensor output is a good first step in using the output of the light sensors. These sensors, however, have one other annoying characteristic: they do not change values uniformly. For example, both sensors may give readings between 0 and 5 in bright sunlight and readings between 225 and 245 in a dark room, but in a room with medium light one may give readings from 40 to 50 and the other from 70 to 80. However, a reactive planning system needs to be able to deal with uncertainty in its sensors, including when they drift in calibration (Brooks 17).

To deal with this, my arm performs a light sensor calibration for every complete left-to-right sweep of the arm, caused by the blind human behavior. If it hasn't found the light source after this, there is most likely a problem with the light sensors. This is important because we don't want to change the light calibration unless we are certain that there is a problem with it in the first place.

The question then becomes how to perform a calibration. In my case, the solution is quite simple because I am using a normalized light sensor reading. Therefore, if I want to change the light sensor reading, I simply need to modify the calculation of the normalized value. Since the only parameter to this calculation (Equation 3-1 above) is the maximum sensor reading, it follows that this is the parameter that should be changed. The calibration should bring the normalized outputs of the two light sensors closer together, since the assumption is that one of them is giving a reading that is *always* significantly higher than the other, even at the same actual light levels.

Since there is no information to tell which sensor is giving the abnormally high (or low) reading, the best we can do is to bring the higher value down some, and raise the lower value. To raise the value of the normalized output, which has an inverse relationship to the maximum sensor reading parameter, we must decrease the value of the maximum sensor reading. Conversely, to decrease the value of the normalized output, the maximum sensor reading must be increased.

Thus, the calibration method is quite simple. When invoked, it takes the current sensor readings from the two light sensors and compares them. The maximum sensor reading parameter for the higher-valued sensor is increased by 1, and the maximum sensor reading parameter for the lower-valued sensor is decreased by 1. For convenience, both maximum sensor values are initially set to 250, which was approximately the highest value that either of the sensors I used ever returned.

Using this method of calibration, the normalized outputs slowly converge until a change in the light level will enable the arm to home in on a light source.

3.2.3.3 "I don't bend that far" behavior

This behavior is the simplest of the three. It simply watches the touch sensors that inform us of having a fully bent joint. When one of the sensors is triggered, it causes the arm to go back in the opposite direction just long enough to turn off the sensor. Making it move the arm off the sensor was a mostly practical measure. Initially, I thought that this behavior should just turn off the motors when the sensor was tripped, and that I could have another behavior move it off the sensor. However, because I wanted to leave this process running all the time, simply turning off the motors was not good enough. This process checks the sensors so often that it never lets another process run the motor long enough to release the sensor. Thus, the process for this behavior needs to move the arm off the sensor by itself.

3.2.4 The arm's world

Given that the arm is supposed to be modeling reactive planning, in which no explicit world representations are kept, I will just briefly mention the arm's world here. Essentially the arm can touch items in the area of a half-donut centered around the base joint. The radius of the inner edge of the donut is the length of the inner segment of the arm; the radius of the outer edge the combined length of both segments of the arm. The item that we are trying to touch should be located within the area of the half-donut.

3.3 Testing of the Arm

Most of the time spent testing the arm was used to perfect some of the fussy details about physical design, such as the best angle and placement of the light sensors. Testing was really a straightforward process of turning on the flashlight, turning on the arm, and watching it go. It is very difficult to quantify the results of these experiments in any sort of numerical fashion that is conducive to bar graphs or charts. About the best we can do is to observe the various behaviors in action and decide whether they are working or not.

3.3.1 Results and Analysis (a.k.a. Behaviors and Misbehaviors)

The arm had good behavior homing in on the light source, once it was within range of the light sensors, which were annoyingly shortsighted. A nice feature of the way I have implemented the behaviors is that it is possible to clearly see the effects of each one. It was easy to see that the light seeking behavior was not doing so well because that particular behavior controlled only the elbow joint. It was similarly easy to see that the waving behavior was performing on the base joint as designed, if not always as desired. Ultimately, one probably would not want to be able to discern exactly which behavior is responsible for a specific action, but in an initial study of the method, I think it was very useful to see.

One particular (undesirable) characteristic of the way the behaviors interacted on the arm was obviously a result of having the behaviors implemented on separate parts, instead of on all parts. As the waving behavior brought the arm around, it was often the case that the outer joint was facing the wrong direction, so that the light sensors wouldn’t pick up the light until after the waving behavior had taken the arm too far past the light to touch it. If the behaviors had been implemented on all joints, two things could have worked out better. First, the light seeking behavior would have become more important than the waving behavior and taken over the operation of the arm when its reading gave a strong indication of where the light was. Second, if the waving behavior had been implemented on the outer joint as well as the first, it may not have been pointed the wrong way in the first place.

3.3.2 Possible Improvements

There are several possibilities for improvements to the implementation of reactive planning on this arm. Unfortunately, time and the available hardware permit did not permit me to implement any of these possible improvements, mostly because they would require an extensive amount of rewriting and expanding of the code.

The first observation is that the calibration of the sensors occurs very slowly, which can mean a long delay before the arm has gone through enough sweeps to get the sensors giving readouts relatively close to one another. These sweeps of the arm are not producing what anyone would call intelligent behavior, so we would like to eliminate them. Eliminating them means that we need to find a way to make the calibration happen more quickly. This could be accomplished in several ways:

- Perform increment/decrement operations as in the current implementation, but do it at both extreme bends instead of just at the right one. This would cause calibration to happen slightly more quickly.
- Modify the maximum sensor values based on how large the difference between the two sensors is. If the difference is large, adjust the maximum value by a proportionately large amount; if it is small, adjust it only a little (or maybe not at all).
- Implement the light sensor calibration as a behavior in its own right, possibly, but not necessarily, completely independent of the number of sweeps of the arm.

The second type of improvement that I would consider would be to extend the behaviors to affect both joints instead of only one. Currently, the outer segment implements only the light seeking behavior, and the base segment implements only the "random" waving behavior. Ideally, you would have a certain amount of each type of behavior in both segments. This would primarily be a reprogramming issue that would require one of two choices.

The first choice is to continue using the current paradigm of controlling individual behaviors with individual, parallel processes. It would require a lot of inter-process communication through shared variables, which is not easily supported on the Handy Board (globally accessible variables are easily accessible, but controlling concurrency of access is not.) The second choice would be to reprogram the behaviors using an entirely different method to moderate the output. One might have a single process in command of all others, which would give each behavior a programmer-specified portion of operating time to give it a set of motor commands, and then combine all those motor commands into one command. This would require some sort of check to make certain that a behavior that wanted to go left, and one that wanted to go right couldn't cancel each other out indefinitely. The reactive planning model allows this type of checking (Brooks, 18).

Finally, there is the whole engineering side of the project. There are many physical changes and additions that could be made to improve the performance of this robot. For instance, the light sensors right now are attached so that they can only face out from the end of the arm. They could be utilized much more effectively if they were to be placed on a rotating axis, so that they could be turned in any direction to look for the strongest source of light, without moving the arm at all. Another possibility would be the addition of sensors to the arm. It could be useful, for instance, to add a sensor to each joint that would tell when it was halfway between the left and right extremes of its motion. Additional touch sensors on the sides of the arm could tell us when a portion of the arm other than the sensor at the very tip had run into an object, and possibly help us to find that object with the tip of the arm more quickly. Additional light sensors that were designed to have a wide field of view could also be helpful for determining an initial direction of motion, to get the current light sensors within the range where they function the best.

4 Conclusion

In this paper, I have given an overview of classical and reactive planning methods, with special attention to their applications in robotics. I have also given a brief overview of some systems that attempt to combine reactive and classical planning into a hybrid system. The research I have done is not extensive enough to be able to say anything for or against their validity as a general planning method, but many feel that this is the best approach to take.

Finally, the greater portion of this paper has covered my own work on a purely reactive system controlling a robot arm, a project that I feel has been quite successful. Working on it has alerted me to a fair number of the difficulties that one encounters with the implementation of a reactive system. In general, these difficulties tend to center around sensory input, and in particular, the imperfection of sensory input. The assumptions of reactive planning, listed in Table 2-1, seem to be perfect, until one starts to realize the difficulty of implementing a system that is based on them. However, these assumptions are more realistic for the world in which robots must operate, and so I strongly believe that work in reactive planning should continue. Though its actual performance was only fair, my arm is probably the simplest implementation of reactive planning that can be of any interest. Therefore, this project has convinced me that reactive planning has the potential to be a viable alternative to the classical planning methods, particularly when the reactive planning assumptions are more realistic for the application of the system.

Appendix: Interactive C Code Used in the Arm

The following code is what drives the behavior of the robot arm. It consists of four files, which are loaded in the order presented here. The file motor.c consists of basic motor commands and constants which allow consistent reference to whatever touch sensor or motor port a sensor or motor is connected to on the controller. It also contains the procedure `watch_sensors()`, which controls the behavior that watches for the arm's joints to be fully bent. The lights.c file defines names for the actual physical port that the light sensors are connected to and has the routines for getting normalized light sensor readings and for calibrating the light sensors. The wave.c file contains the routine for controlling the waving behavior of the base joint. Finally, the main.c file contains the main function call. It watches the start and stop buttons on the controller, and is responsible for checking for the goal state and starting and stopping the `wave()` and `watch_sensors()` processes. The main function is also responsible for the light seeking behavior.

```
/**********************************************************************/
/*  File: motor.c                                                     */
/*  Author: Dan Churchill                                             */
/*  Created: January 21, 2000                                         */
/*  Last Modified: April 12, 2000                                     */
/*  Description: A file of constants and routines to use with         */
/*               Interactive C in the programming of the handy_board  */
/*               for use with the robot arm.  This file should        */
/*               contain all necessary control functions for the arm. */
/**********************************************************************/

/* Define some constants based on where the motors & sensors are plugged in */

int m_1 = 1;     /* The motor at the base joint                     */
int s_l1 = 7;   /* Sensor to stop counterclockwise rotation        */
int s_r1 = 8;    /* Sensor to stop clockwise rotation               */

int m_2 = 2;     /* The motor at the first joint                    */
int s_l2 = 9;    /* Sensor to stop counterclockwise rotation       */
int s_r2 = 10;  /* Sensor to stop clockwise rotation              */

/* Motor speeds */
int fast = 100;
int slow = 40;
int stop = 0;

int s_l1_cnt = 0;   /* the number of times we've hit the left sensor (base
joint) */
int s_r1_cnt = 0;  /* the number of times we've hit the right sensor (base
joint) */

/* Sensor watching process
       This procedure is intended to be spawned at the beginning of execution
       and be allowed to run until execution stops.  It watches the digital
       sensors and stops the appropriate motor when one of them is tripped. It
```

```
    then moves the arm just enough in the opposite direction to release the
    switch.
*/
void watch_sensors() {
    while(1) {
        if(digital(s_l1) ) {
            s_l1_cnt++;
            motor(m_1, stop);
            motor(m_1, slow);
            while(digital(s_l1));
            motor(m_1, stop);
        }
        if(digital(s_r1) ) {
            s_r1_cnt++;
            motor(m_1, stop);
            motor(m_1, (0_slow));
            while(digital(s_r1));
            motor(m_1, stop);
        }
        if(digital(s_l2) ) {
            motor(m_2, stop);
            motor(m_2, slow);
            while(digital(s_l2));
            motor(m_2, stop);
        }
        if(digital(s_r2) ) {
            motor(m_2, stop);
            motor(m_2, (0_slow));
            while(digital(s_r2));
            motor(m_2, stop);
        }
    }
}

/* Extends everything as far to the left as it goes.
    It assumes that watch_sensors is running in a separate process. */
void full_left() {
       /* Start both motors       */
    motor(m_1,(0 _ fast));
    motor(m_2,(0 _ fast));

       /* Loop until both sensors are tripped  */
    while(!digital(s_l1) || !digital(s_l2) );
}

/* Extends everything as far to the right as it goes.
       It assumes that watch_sensors is running in a separate process.    */
void full_right() {
       /* Start both motors */
    motor(m_1,slow);
    motor(m_2,fast);
    /* Loop until both sensors are tripped  */
      while(!digital(s_r1) || !digital(s_r2));
}

/* Starts to bend the specified joint to the left at the specified speed.
    Will return execution to calling procedure immediately.  Stopping the
    arm must be done elsewhere. */
void left(int joint, int speed) {
    motor(joint, (0_speed));
}

/* Starts to bend the specified joint to the right at the specified speed.
    Will return execution to calling procedure immediately.  Stopping the
    arm must be done elsewhere. */
void right(int joint, int speed) {
    motor(joint, speed);
}

/* Bends the specified joint to the left for the specified number of msecs
       It will NOT check to see if it has hit a sensor. */
void timed_left(int joint, int speed, long time) {
    motor(joint, (0_speed));
    msleep(time);
    motor(joint, stop); /* Stop the motor */
}

/* Bends the specified joint to the right for the specified number of msecs.
    It will NOT check to see if it has hit a sensor. */
void timed_right(int joint, int speed, long time) {
    motor(joint, speed);
      msleep(time);
    motor(joint, stop); /* Stop the motor */
}
```

```
/**********************************************************************/
/*  File: lights.c
/*  Author: Dan Churchill
/*  Created: March 22, 2000
/*  Last Modified: April 12, 2000
/*  Description: The light sensor routines and constants.
/**********************************************************************/

int s_ll = 0;   /* Left light sensor in analog port 0 */
int s_lr = 1;   /* Right light sensor in analog port 1 */

int ll_max = 250;   /* Highest reading of left light sensor */
int lr_max = 250;   /* Highest reading of right light sensor */

/* A procedure to return normalized light sensor readings
    A positive reading means that there is more light to the right
    of the current sensor position, a negative reading means more
    light to the left of the current sensor position. */
int light_reading() {
    int left_percent = analog(s_ll)*100/ll_max;
    int right_percent = analog(s_lr)*100/lr_max;

    printf("\nL%:%d, R%:%d", left_percent, right_percent);

    return (int)left_percent _ (int)right_percent;
}

/* This procedure is called when the outer arm is stuck at one extreme
   position.  The assumption is that one light sensor's percentage is
   significantly higher than the other and needs to have its max increased
   accordingly.  This procedure looks at the current difference in the
   sensors.  Whichever sensor percentage is higher has its max incremented,
   and whichever sensor percentage is lower has its max decremented.  Both
   maximum values are hardcoded to start at 250. */
void calibrateLights() {
    if (light_reading() > 0) { /* the left percent is higher */
        ll_max++;
        tone(440.0,0.2);
        lr_max__;
        tone(360.0,0.2);
    }
    else {
        ll_max__;
        tone(360.0,0.2);
        lr_max++;
        tone(440.0,0.2);
    }
    /* printf("\nllmax: %d, lrmax: %d", ll_max, lr_max); */
}
```

```
/*******************************************************************/
/*  File: wave.c
/*  Author: Dan Churchill
/*  Created: April 12, 2000
/*  Last Modified: April 12, 2000
/*  Description: The routine for controlling the "waving" behavior
/*               of the robot arm.  Assumes that motor.c and lights.c
/*                          are loaded first.
/*******************************************************************/

void wave() {
    /* Since we default to go left, an increase in the number
       of times we've hit the right indicates a full sweep of the arm
       without luck.  Therefore, it is probably the case that we need to
       calibrate the light sensors.  The following helps us to remember
       how many sweeps of the arm have been made.  After every sweep, we
       will modify the light sensor max values a little bit.  Eventually
       the readings should get to a point where the outer segment of the
       arm will move away from a fully bent position. */
    int last_s_r1_cnt = s_r1_cnt;

    while(1) {
        reset_system_time();    /* Set the clock to 0 */

        while(mseconds() < 2500L);  /* Give the outer time to find the goal */

        if( s_l1_cnt <= s_r1_cnt ) {
            /* we've gone right more than left, so go left this time */
            /* if they're equal, go left first */
            timed_left(m_1, slow, 1000L);

        }
        else {
            /* we've gone left more than right, so go right this time */
             timed_right(m_1, slow, 1000L);
        }

        if( s_r1_cnt > last_s_r1_cnt ) {
            calibrateLights();
            last_s_r1_cnt = s_r1_cnt;
        }
    }
}
```

```
/*********************************************************************/
/*  File: main.c
/*  Author: Dan Churchill
/*  Created: March 22, 2000
/*  Last Modified: April 12, 2000
/*  Description: The main routines for my honors project.
/*               This code assumes the presence of motor.c
/*********************************************************************/

void main() {

    int waveproc;   /* the process number of the wave process */
    int running = 0; /* Is the arm active? 1=True 0=False */
    int lights;

      /* start the watch_sensors process
      *      _ runs for 1 tick before giving up the processor (default is 5)
      *      _ stack size of 50 bytes (default size is 256)
      */
    start_process( watch_sensors(), 1, 50);

    /* Run an infinite loop */
    while(1) {
        if(!running && start_button()) {    /* look for a start signal */
            running = 1;

            /* start the wave process which will move our base joint if we
don't
               find our goal */
            waveproc = start_process( wave() );

        }
        else if(running && stop_button()) {  /* check for a stop signal */
            running = 0;

            /* stop the wave process */
            kill_process( waveproc );
        }
        else if(running) {                /* Seek the light */
            if (digital(11)) {            /* then we've found the light */
                kill_process( waveproc ); /* stop the wave process */
                ao();                     /* turn off all of the motors */
                tone(440.0,0.1); msleep(50L); tone(440.0,0.5); /* play some
music */
                printf("\nTa da!");       /* celebrate */
                running = 0;              /* rest */
            }
            else {
            /* get light sensor readings */
              lights = light_reading();

              if(lights < 0) {
                if(lights > _6 && !digital(s_l2)) left(m_2,20);
                if(lights > _16 && !digital(s_l2)) left(m_2,30);
                if(lights > _31 && !digital(s_l2)) left(m_2,40);
                if(lights < _30 && !digital(s_l2)) left(m_2,50);
             }
                    if(lights >= 0) { /* Random decision to default to the
right */
                    if(lights < 6 && !digital(s_r2)) right(m_2,20);
                    if(lights < 16 && !digital(s_r2)) right(m_2,30);
                    if(lights < 31 && !digital(s_r2)) right(m_2,40);
                    if(lights > 30 && !digital(s_r2)) right(m_2,50);
                }
            }
        }
    }
}
```

Bibliography

Bresina, John L. "Design of a Reactive System Based on Classical Planning." Online. Internet. Available http://ic-www.arc.nasa.gov/ic/projects/xfr/backgound/jb-spring-93.html.

Brill, F. Z., 1996. "Representation of Local Space in Perception/Action Systems: Behaving Appropriately in Difficult Situations." Ph.D. Dissertation. University of Virginia. Online. Internet. Available ftp://ftp.cs.virginia.edu/pub/dissertations/9604.pdf.

Brooks, Rodney A. "Intelligence Without Reason." A.I. Memo. No. 1293. MIT, April 1991.

Brooks, Rodney A. and Stein, Lynn Andrea. "Building Brains for Bodies." A.I. Memo. No. 1439. MIT, August 1993.

Drummond, M., Swanson, K., Bresina, J., and Levinson, R. "Reaction First Search." *Proceedings of IJCAI '93*. Chambery, France: Morgan-Kaufman.1993.

Martin, Fred G. *The Handy Board Technical Reference.* 1998. Online. Internet. Available http://el.www.media.mit.edu/projects/handy-board/.

Russell, Stuart J. and Norvig, Peter. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, NJ: 1995.