MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo 1025 June 1988

A Behavior-Based Arm Controller

Jonathan H. Connell

ABSTRACT: In this paper we describe a working, implemented controller for a real, physical mobile robot arm. The controller is composed of a collection of 15 independent behaviors which run, in real-time, on a set of eight loosely-coupled on-board 8-bit microprocessors. We describe how these behaviors cooperate to actually seek out and retrieve objects using local sensory data. We also discuss the methodology used to decompose this collection task and the types of spatial representation and reasoning used by the system.

Acknowledgements: This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for this research is provided in part by the University Research Initiative under Office of Naval Research contract N00014-86-K-0685, in part by a grant from the Systems Development Foundation, and in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-85-K-0124. Mr. Connell is currently supported by a General Motors graduate fellowship.

© Massachusetts Institute of Technology 1988

1. Introduction

The controller described here is for use with an arm mounted on a mobile robot. Figure 1 shows the arm in action. The purpose of this mobile robot is to roam around our lab area looking for soda cans [Brooks, Connell, and Ning 87]. When it locates one, it picks the can up and brings it back to a home location. In this paper we will concentrate on the collection aspect rather than on the locating and navigation issues.

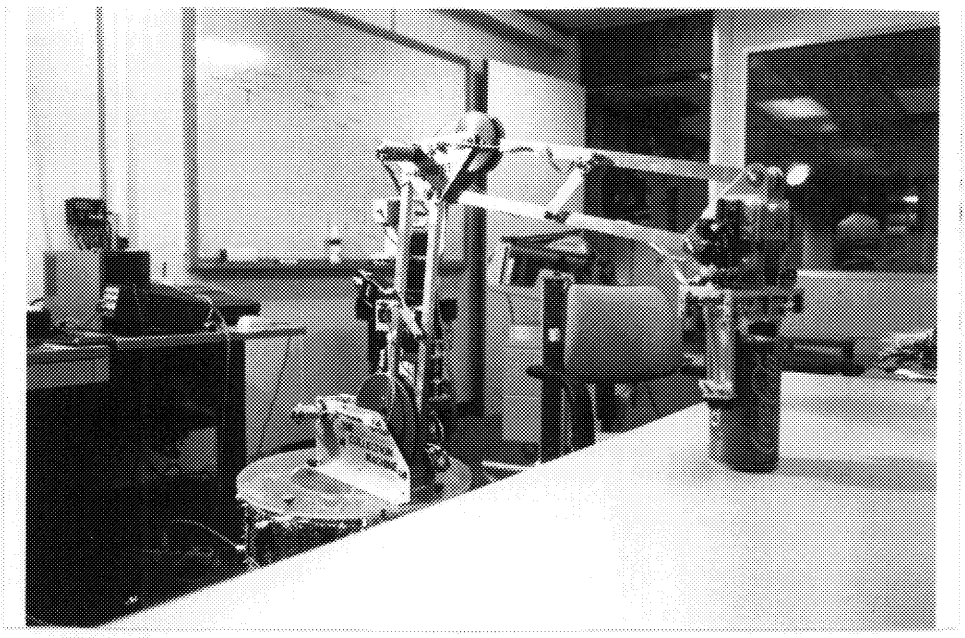

Figure 1. The arm is mounted on the top of our mobile robot. Its purpose is to collect soda cans from tables and from the floor.

A major difference between our approach and that of most other groups is that our robot is controlled by a collection of small behaviors rather than a complex centralized program. At first sight it is highly schizophrenic in the sense advocated by Minsky [Minsky 86]. However, just because there a number of independent, isolated agents there is no reason why they can not cooperate to achieve a specific goal. In the following sections we will detail how agents represent the portions of the world they are interested in, how they interact, and how they coordinate their actions.

2. The Behavioral Model

The control system for our robot is based on Brooks's *subsumption architecture* [Brooks 86]. This architecture uses a vertical decomposition of the control system. Instead of having one very complete perception system that feeds into

a comprehensive modeller that in turn feeds into a sophisticated planner, we break the system down into a number of parallel paths from sensors to actuators. Typically each path is concerned with achieving some particular task that the robot needs to perform. One advantage of this approach is that control systems can be built incrementally. We can start by building a wall-avoider and then debug this behavior until it works well. Afterwards, we can go on to build a door-recognizer without having to go back and change the wall-follower. This is because each behavior has its own separate perception, modelling, planning, and execution subsystems. Another advantage of the subsumption architecture is that each of these paths can be relatively simple. For example, if the task is avoiding walls then the associated control system does not need to know the color of the wall, its texture, its decomposition into generalized cylinders, etc. Each behavior only has to be aware of the relevant *indexical-functional aspects* [Agre and Chapman 87]. The word "aspect" is used to indicate that this is not a full, structured representation. "Functional" means things are defined by how one uses them rather than by intrinsic characteristics, while "indexical" means that there are no logical individuals but only classes of more or less interchangeable instances.

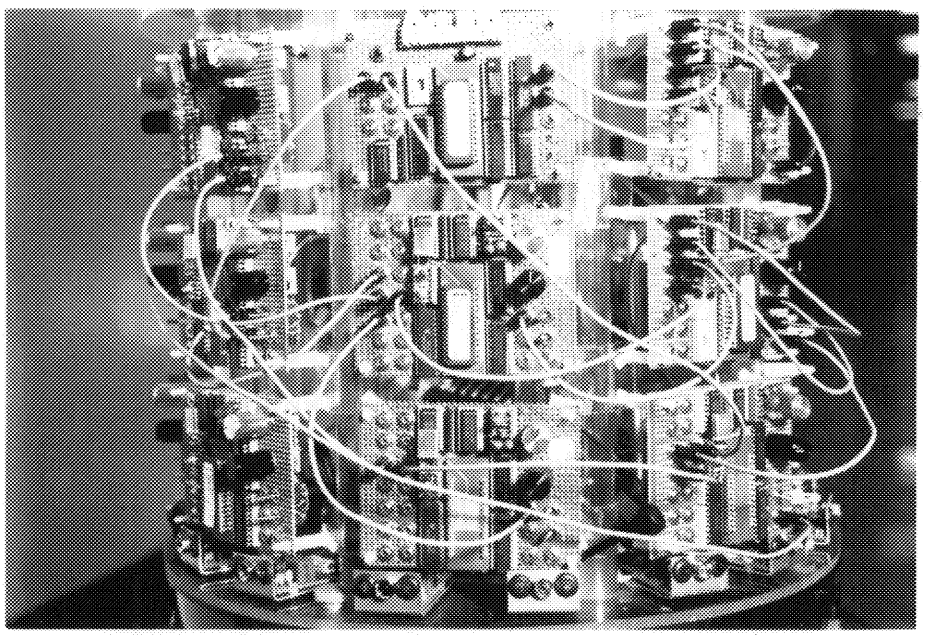

Figure 2. The 15 behaviors which control the arm are implemented using eight 8-bit processors mounted around the periphery of the robot.

The subsumption architecture is more than just a mindset; it also specifies a set of primitives which have proved useful for building multi-agent systems. First, the subsumption architecture declares that each behavior is composed of a small set of information processing modules which are interconnected by wires that carry message packets. Inside each module is a finite state machine which can perform a number of simple functions. Most important among

these are: wait for a particular signal, time-out after a certain period, test an input for a certain condition, and perform limited computation on the input values to generate an output value. In our system there is a one-to-one correspondence between modules and behaviors.

Modules interact using 3 special constructs. One module can *reset* another module by forcing its internal finite state machine to the initial state. A module can also *inhibit* the output of another module, that is, prevent it from generating outputs for a certain amount of time. Finally, the output of one module can *suppress* the output of another. In the case of suppression, the output from the dominant module overrides the output of the inferior module for a prespecified amount of time. This is a particularly powerful construct because it lets more competent behaviors take over from general-purpose behaviors when the circumstances warrant it. This language has proved very useful in our own research group and has been successfully used by others as well [Hywel-Snaith 87].

The circuit-like definition of the subsumption architecture lends itself naturally to a parallel processor implementation. Shown in Figure 2 are the eight on-board microprocessors that implement the 15 arm behaviors. Each 8-bit microprocessor simulates a small number of modules which are then connected by real, physical wires to other modules [Brooks, Connell, and Flynn 86]. Due to the nature of the parallelism, we can make this system arbitrarily large without saturating any resource. If we want more behaviors, we simply add more processors.

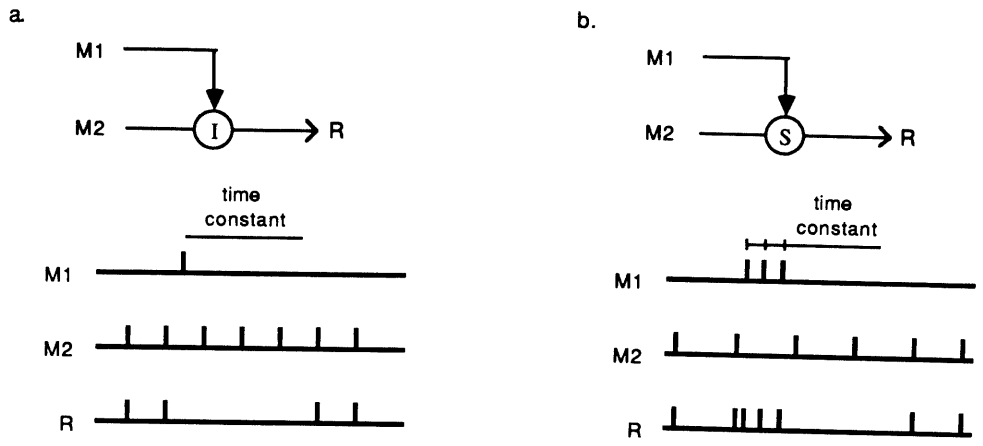

Figure 3. Inhibition and suppression. a. The effect of M1 inhibiting M2. b. The effect of M1 suppressing M2.

While the subsumption architecture specifies the components which can be used, the programming style is still up to the implementer. In the control system described here, we have departed somewhat from the standard form of

the subsumption architecture. Instead of carrying discrete messages, the wires carry continuous signals. This allows us to remove the time constants from the suppressor nodes. The dominant module remains in control of the suppressor node as long as it is sending messages. However, as soon as it stops generating outputs, control reverts to the inferior module. The new semantics of a suppressor node are "use it or lose it". This lets us model the suppression network, the primary mode of interaction between behaviors, as a simple fixed-priority arbitration system. Note that the hardware we have developed for the direct implementation of the subsumption architecture can just as easily run systems written in this new style.

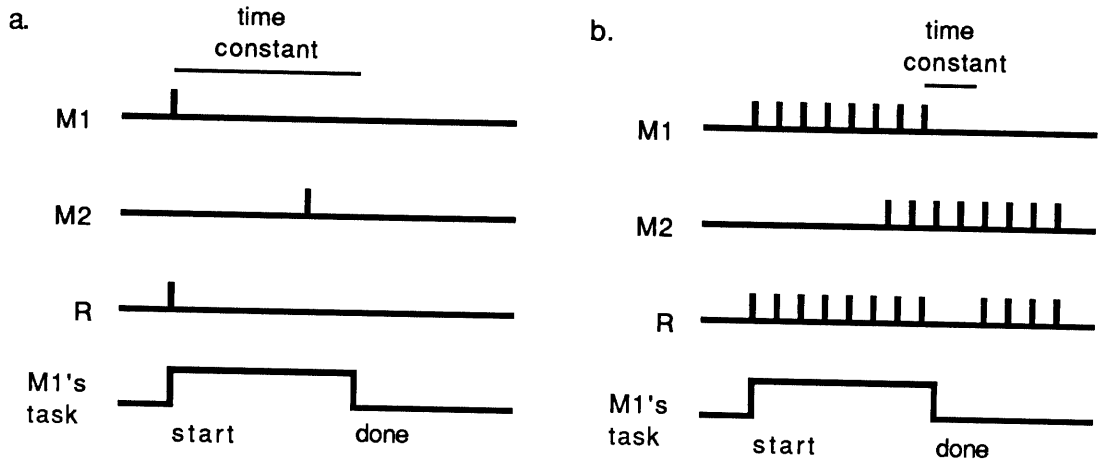

Figure 4. The bug in the packet model. a. Note that the command from M2 will never be processed. b. The same interaction is successful using the signal model.

The signal model of wires was invented to fix a bug with the previous discrete message formalism as shown in Figure 4a. In the original version [Brooks 86], a module would send a single message specifying what the robot should do and then remain silent as the robot carried out the directive. If the output of this module (M1) was connected to the dominant input of a suppressor node, it would commandeer the associated wire for T seconds. Now assume that the action takes much less than T seconds to complete, and that, after it is finished, the inferior module (M2) generates a command for the robot. Although the shared resource (the robot) is free to execute new commands, this command will *never* be acted on because it can not pass through the pre-empted suppressor node. Even after the node times out and returns control to the inferior module, the module's desires are never heard because it only sends one packet. In general, dropping a single packet can be catastrophic. In the signal model (Figure 4b), however, the command is repeated continuously thus alleviating the problem. Note that the command also does not have to stay the same; if, after it initiates an action, a module is able to give more accurate instructions, these will be reflected in the signal and thus can be acted on.

In general, the ability to specify time constants is very useful. Since we are no longer allowed to put this functionality into the suppressor nodes, we move it back along the signal chain and into the module which generates the messages. We do this by introducing a new internal construct for a module called a *monostable*. This device essentially stretches the interval during which a predicate remains true. Suppose we test the input of a module with some predicate and the predicate returns true. If this is part of a monostable construct, then the whole construct will also return true. The difference is that monostable will remain activated for some fixed amount of time after the predicate is no longer true. Furthermore, if, during this time, another input arrives and passes the specified test, then the length of time that the construct returns true is extended. The EXTEND module discussed in section 4 shows one way to use the monostable construct.

3. Hardware

The arm used in our studies is a planar, 2 degree-of-freedom design. However, it is structured so that the hand anywhere inside an 18 by 40 inch vertically oriented workspace (we can rotate the base to provide lateral motion). This large workspace allows the robot to manipulate objects at both floor and table top level. The hand is connected to the base through two parallelogram linkages and hence, no matter where we move the hand, the fingers always point straight down. The hand itself is a parallel jaw gripper which can open to approximately 5 inches. All three actuators, shoulder, elbow, and gripper, are controlled by analog servo-loops. The servos take a velocity command and integrate it to yield a desired position for a standard position controller. The arm can lift a payload of up to 1.5 pounds but it is fairly slow -- about 10 seconds from full down to full up at top speed.

The subsumption architecture processors have access to various sorts of sensory data. The servos report the arm's joint angles and finger separation as well as the error voltage in each loop (a crude measure of torque). The hand also has a number of special purpose sensors as shown in Figure 3. Near the ends of the fingers there is an infrared beam that is broken when something comes between the fingers (we refer to this as the "finger beam" in later sections). There are also contact switches on the fingertips which signal when the fingers hit something (the "tip switches"). The wrist of the manipulator is also free to rotate slightly and there is a micro-switch which is activated if the wrist experiences any upward force. When the hand is grasping something this switch plays the same role as the tip switches for an empty hand, so in most cases we lump this signal in with that of the tip switches.

One of the most useful sensors is a pair of crossed infrared proximity beams located at the leading edge of the hand (the "crossed IRs"). These operate in two modes. In the basic proximity mode they just monitor how much of the light they are emitting is reflected. We threshold this quantity to determine if there is something in the beam or not. In the geometric mode we monitor how much of the light generated by one sensor is reflected to the other one. Once again this is threshold to yield a single bit of information. Note that since the

beams cross over each other about 2 inches in front of the hand, we can tell when an object is in this intersection.

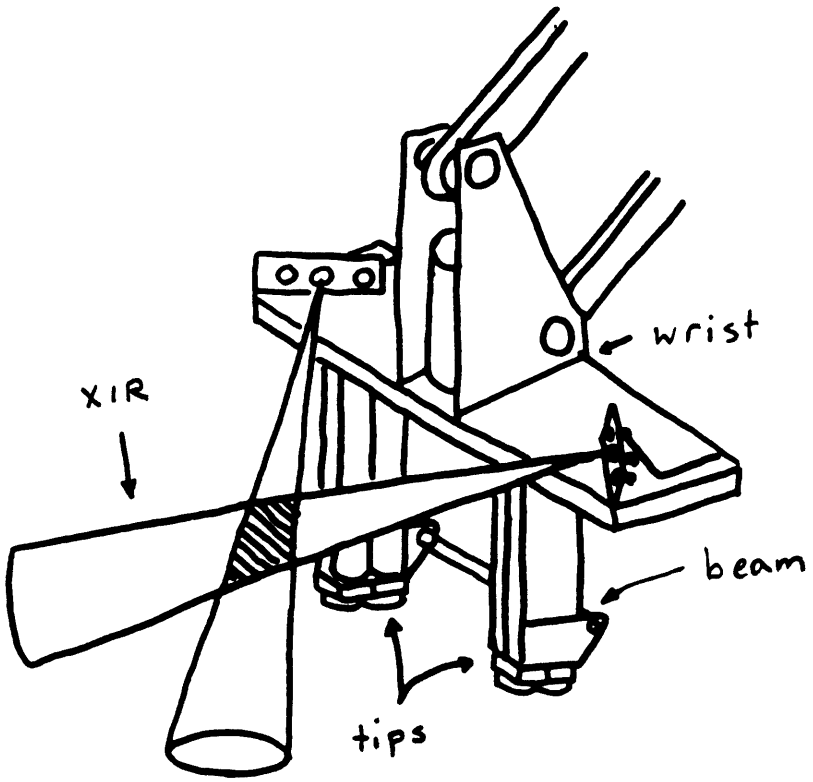

Figure 5. The sensors mounted on the hand.

4. Controlling the Arm

The complete set of behaviors and their interconnections is shown in Figure 6. Each box is a module which implements a specific behavior. All behaviors interact through suppressor nodes (the circles with an "S" inside). As mentioned earlier, the semantics of these nodes is that a signal coming into the side of a node can override the signal passing through the node. In one case there is a default node (a circle with a "D" inside). This is just like a suppressor node in which the dominant and inferior inputs have been reversed. Note that we have broken the collection down into 6 groups: **Cradle, Grip, Path, Park, Skim,** and **Local**, which we refer to as levels of competence. The idea is that the robot can perform useful actions with just the lower level implemented but its performance gets progressively better as more and more levels are added.

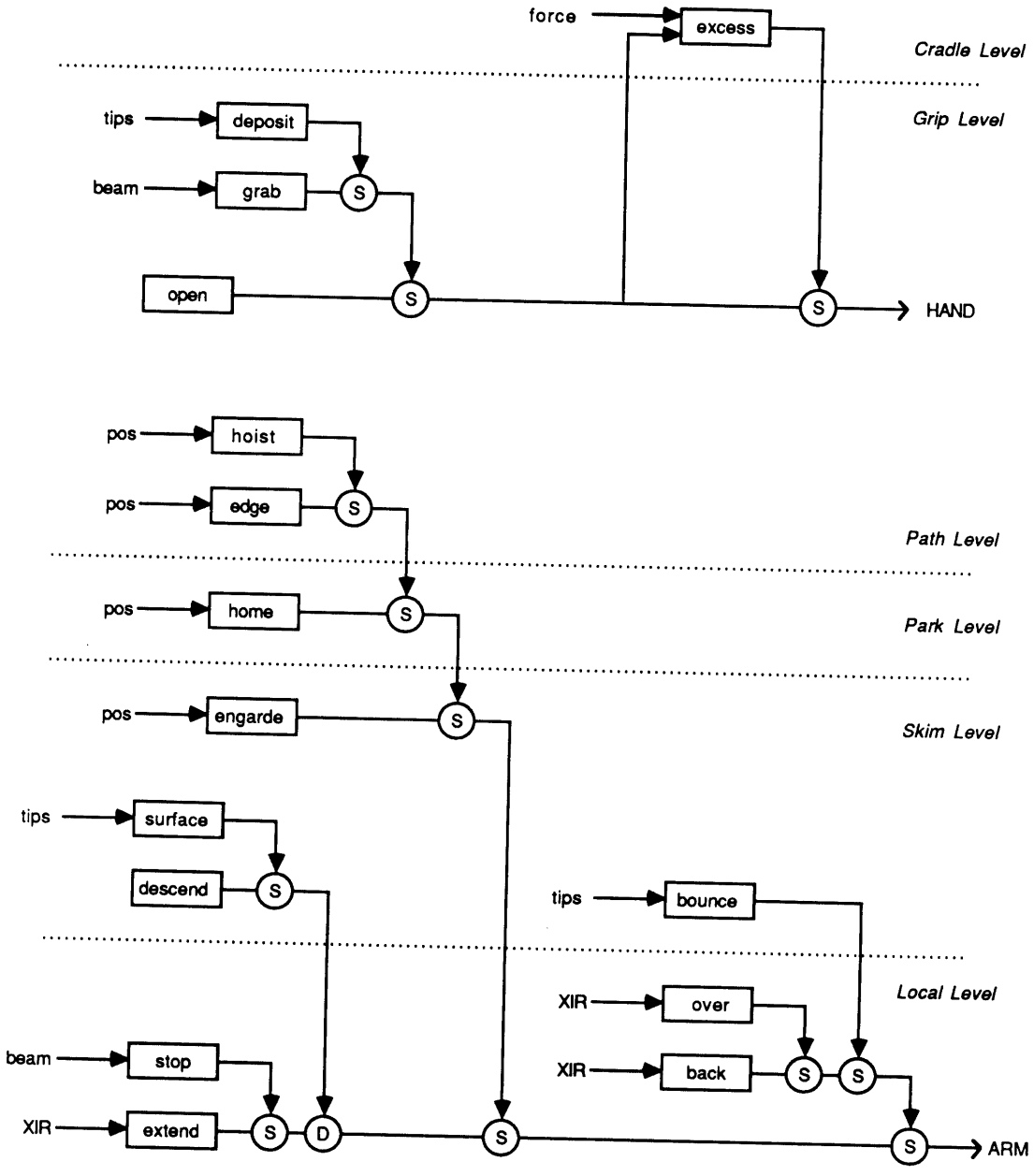

Figure 6. The control system for the arm is composed of 15 separate behaviors. These can be grouped into 6 task-specific levels.

Let us start with the **Grip** level. The most basic behavior in this level is to unconditionally open the hand. This is accomplished by the OPEN behavior but is modified using GRAB. The GRAB behavior monitors the finger beam and directs the hand to close so long as the beam is broken. Thus, anytime something comes between the fingers, the hand tries to close. Note that the hand could not care less whether this is a soda can or something else, it just knows to close in this situation. The GRAB behavior is in turn modified by DEPOSIT which forces the hand to open if either tip switch or the wrist sensor is activated. This is how the robot puts things down: by moving the hand down until a contact between the grasped can and the table causes sufficient force to activate the wrist sensor. It is also useful in its own right as a general protective behavior. For instance, the wrist sensor is also activated when the bottom of the can catches on a protruding obstacle or when the robot has grabbed a person's hand and they attempt to shake it loose.

The next level, **Cradle**, contains a single behavior called EXCESS. The purpose of this behavior is to regulate the force exerted by the gripper. EXCESS monitors the error in the finger servo-loop and when the force gets above a set threshold, takes control of the hand and reduces the force to a reasonable level. This is important because squeezing things too hard tends to damage them, while trying to open the fingers beyond their limit-stops heats up the motor. EXCESS also monitors the commanded velocity of the fingers to determine when to relinquish control. It drops out when it sees that the hand is being commanded to move in the direction which reduces the force. If we did not have this feature, the hand would become stuck either open or closed due to the fixed priority arbitration scheme implicit in the set of suppressor nodes.

Let us turn our attention now to the behaviors controlling the arm, in particular the **Local** level. Much of this level is devoted to interpreting the data from the crossed IRs. For instance, the EXTEND behavior says that if either IR has seen something recently, the robot should extend the arm straight forward in an attempt to locate the sensed object once again. The notion of "recently" is implemented using the retriggerable monostable construct discussed earlier. As each IR packet comes in, we test whether either of the IRs is currently detecting an object. If an object has been spotted, we trigger the monostable which then directs the hand to go forward. This means the hand will go forward as long as one of the IRs sees something, and then *continue* to go forward for a short time after the signal disappears. This is a reasonable algorithm, since, in practice, when the hand chases objects it is often capable of catching them. Unfortunately, the fingers take a long time to close. So, if the hand keeps moving when the object is in reach, it may actually go beyond the object before the fingers have closed sufficiently. Therefore, we have added the STOP behavior to freeze the hand for awhile if the robot detects any change in the finger beam. This module also uses the monostable construct to accomplish the "awhile" part. Note that the stop command takes precedent over any extension commands because the output of STOP suppresses the output of EXTEND.

Aside from the exploratory behaviors, this level also has two protective behaviors. EXTEND is constantly trying to drive the arm forward; yet this is not

always a smart thing to do, especially if there is something directly in front of the hand. One case is when the hand has reached a wall. Here both IRs will see something and BACK will command the arm to retreat slightly to avoid collision. If, on the other hand, something is detected at the intersection of the two beams, the OVER behavior forces the hand to go straight up. Once again the relative priorities of these behaviors can be determined by examining the wiring of the suppressor nodes. When the hand comes up to a can, the combination of the EXTEND and OVER behaviors causes the hand to rise slightly above the top of the can and then go forward causing the can to pass between the fingers. This in turn activates the grasp reflex. Note that, although the crossed IR detector can obviously determine the location of a can, it does not communicate this directly to the finger controller. Rather, it sets up the world such that in the course of the robot's normal actions the grasp reflex will be triggered.

The next level of competence is the **Skim** level. This group of behaviors contains an operational representation of the fact that objects often rest on surfaces. ENGARDE starts off by moving the hand from the tucked travelling position up to near the top of its workspace and out towards the inner edge of the workspace. When the hand reaches the edge of the workspace ENGARDE shuts down and lets the exploratory behaviors take control. For instance, EXTEND may have sensed an object and started generating motion commands. Yet because it has lower priority than ENGARDE (as can be seen from the suppressor nodes), its commands can not get through until ENGARDE stops producing outputs. Usually, however, EXTEND has not been activated so instead the default behavior, DESCEND, simply drives the hand straight down. This action is modified by the protective behavior BOUNCE which forces the hand to go back up if one of the tip switches has been activated. When the tip switches are activated there is a good chance that the hand has come in contact with a support surface. Therefore, if SURFACE sees that the tip switches have been hit it directs the hand to go forward and slightly down for awhile. Eventually, if the surface is more or less flat, this causes the tip switches to be activated again. In this manner the hand hops along a surface hoping to find a can to collect.

There is more to collecting cans than just finding and grabbing them. One must also bring them back closer to the body in order to transport them to another location. This is the purpose of the HOME module in the **Park** level. The evolution of this module is particularly interesting because it was originally designed as a protective behavior. Occasionally the robot gets into a situation where it should simply give up. For instance, the arm might reach the edge of its working envelope or become stuck somewhere. In these circumstances, a reasonable thing to do is to retract the arm. Observing that in both cases the arm ceases to move, we trigger the HOME behavior whenever the hand is stopped at a non-home location. The HOME module then attempts to bring the hand back to a special parking location, and keep it there for a certain amount of time. Remember, however, that we also deliberately stop the hand when the finger beam changes state to allow the fingers time to open or close. This naturally triggers the HOME behavior which brings the arm back after the robot has grasped or ungrasped and object. In this case, we did not

need a new, separate behavior; we were able to tap into an existing reflex by generating its local environmental activation pattern.

The final level, **Path,** modifies the basic retrieval behavior. HOIST is activated by the same environmental conditions as HOME, but it does not head directly toward the parking location. Rather, it first lifts the can almost to the top of the workspace, then brings it straight back to the inner edge. At this point it drops out and lets HOME take over. This is a better retraction strategy when the hand grabs an object at a height above that of the parking location. In such a situation, if the hand tried to go straight it would stand a good chance of running into whatever surface had been supporting the grasped object. The up-then-back strategy used by HOIST eliminates this difficulty. The other behavior in this level, EDGE stops the hand (and hence causes retraction) when the hand attempts to go beyond the far edge of an inscribed rectangular workspace. While the hand can actually reach beyond these limits many times, it can not rise straight up all the way to the top of the rectangular workspace if it goes any further. EDGE's purpose, therefore, is to make sure the hand does not go beyond the point where the HOIST strategy will work.

5. Spatial Representation

Note that our arm can collect cans from a variety of surfaces without requiring any detailed description of what a can is or what the surfaces look like. Given our multi-processor implementation, there would be no place to put such a representation anyway. What we have instead is a collection of partial descriptions spread among various behaviors. Consider what we would do if we had a complete internal model of the world. We would, in that model, measure various parameters such as the height of the top of the can and its distance in front of the hand. So why bother with the model? Why not *use the world as its own model*, and simply measure these quantities directly? This is exactly what we do by carefully arranging the sensors and then monitoring their response over time. For instance, we can tell when the can is in position to be grasped because this will cause the finger beam to be broken. We can tell when the hand is directly above a surface because the tip switches will be activated. We can determine how high the top of the can is, and hence how high to raise the hand, by looking for the disappearance of the crossed IR signal. In some cases, such the door-finder described in [Brooks and Connell 86], we can even use the effects of the behaviors themselves to pare down our representations.

One often cited reason for maintaining a comprehensive, centralized world model is that it allows us to do sensor fusion [Giralt, Chatila, and Vaisset 84; Flynn 85; Shafer, Stentz, and Thorpe 86]. Since we do not have such a model, our answer is to use *behavior fusion*. For instance, using a traditional approach we could use the information from the tip switches, the crossed IR beams, and the joint encoders to build up a local model of obstacles to avoid. Instead, we have two separate behaviors, BOUNCE and BACK, which employ different sets of sensors. BACK takes care of avoiding objects in front of the hand while BOUNCE prevents the robot from jamming its fingers into the table. If there is ever a conflict between the two, BOUNCE takes precedent. Note that we have achieved all that we could hope for if we had fused the data from the two sensors. We have done it, however, using behaviors that only pay

attention to one set of sensors. The advantage of our approach is that it allows us to avoid the difficult task of trying to calibrate the two sets of sensor readings against each other.

The kinematic knowledge that we need for the can grabbing task is also very slight. We certainly do not need to know the *x-y* position of the hand relative to the robot. Our actual requirements are shown in Figure 7a. We need to know the hand's qualitative direction relative to a single point, "park", used by HOME. Beyond this, we need to know where the hand is relative to the three edges of the workspace. The outer boundary is used by EDGE, the top boundary by HOIST and ENGARDE, and the inner boundary by HOIST. This makes a total of only 7 binary predicates (4 to delimit the "park" location and one for each of the edges of the workspace) that we either have to calibrate or learn -- certainly not an unmanageable number and much easier than dealing with the full kinematics of the manipulator.

We have also adopted a minimalist attitude toward effector commands. Just because the hand can be moved in any direction, it is not necessary to use all directions. As a matter of fact, we only ever move the arm in one of 8 directions, as shown in Figure 7b. These motions are all in the vertical plane which is the arm's workspace. Once again, this makes calibration or learning of the inverse kinematics easier since we only have to solve for these special cases.

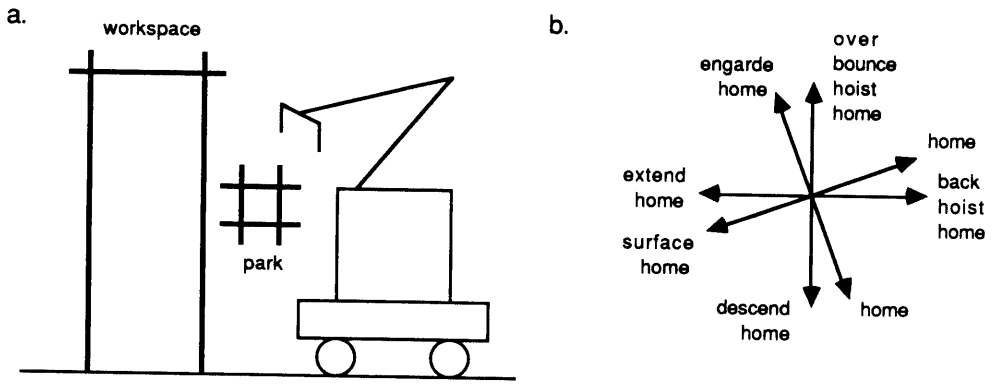

Figure 7. The spatial knowledge employed by the system. a. One distinguished place and the borders of the workspace. b. The 8 local directions of movement.

Finally, note that our use of the subsumption architecture differs from previous results as to whether different behaviors can share intermediate level modules. In the system described here, we forgo any overlap at all, whereas, heretofore, it was quite common for a higher level behavior to tap off the outputs of certain modules in a lower level. The reason we eschew this practice is for the sake of modularity. There may be a number of ways to implement a particular behavior; however, only a small number of these will

have the correct internal structure to allow the higher level to operate. Thus, we may get a viable debugged low-level behavior only to find that we have to radically restructure it so that the appropriate internal signals are available to higher levels. This seems to defeat the claim of incremental extensibility. This is especially irksome when we mix levels of implementation. As we have shown elsewhere [Connell 87], the subsumption architecture can be compiled down to the gate level. Unfortunately, when we do this we often have to cut corners so that the intermediate signals simply do not exist. This obviously rules out any possibility of higher levels using them.

For these reasons, we have adopted the maxim "let the behavior be the interface". That is, do not depend on the internal structure of some behavior, depend, rather, on this behavior's effect on the world. This way, as long as the behavior achieves the same task, it does not matter how it happens to be implemented. However, in many instances behaviors need to communicate with each other in order to coordinate their actions. If each behavior is a monolithic entity, the only way remaining is to *communicate through the world.* We have already seen one example of this, namely, stopping the arm triggers the HOIST and HOME behaviors. Remember that the arm may be frozen by STOP if the finger beam changes state, or by EDGE if the arm tries to extend itself too far. This is how these two modules communicate their desire to retract the arm to the behaviors which know how to do this.

The beauty of communication through the world is that we can have infinite fan-in and fan-out without knowing who is generating the signal and who the recipients are. Already there are three behaviors which stop the arm and two behaviors that watch for the arm to be stopped. We could easily add some other behavior which, say, desired the retraction of the arm if it had been away from home for too long. It would signal its desire just like the other two behaviors, by stopping the arm. We could also add a new retraction behavior, like lifting the arm only a little before retraction if the arm was near the floor, which also triggered off the same external signal. Compare this to the case of having an internal desire-to-retract wire. Each new retraction method and retraction requestor must know where this wire is inside the control system, once again causing us to rely on a particular internal structure.

We have used a similar setup to coordinate the actions of the base and that of the arm [Connell 88]. First of all, we do not want the base to move if the arm is extended. We prevent this by a adding a simple behavior which suppresses all base motor commands when the arm is not at its home location. So once the arm is extended, the base must wait for it to complete its cycle before moving again. We use a corresponding behavior to signal the arm that it is time to extend. The control system described in section 4 assumed that it was always alright to move the arm. We now add a new behavior which overrides all arm commands (we do not care about the hand) if the base has moved recently. Thus to cause the arm to extend, we park the base at some location for awhile which signals, through the world, that it is time to extend the arm.

We can go one step beyond this and use the same form of communication for hand-eye coordination. Instead of figuring out how arm coordinates translate into image coordinates we can use our knowledge about the composite

behavior of the arm controller. The way it is set up, if we can cause an object to be in the arm's workspace, we are guaranteed that our controller will retrieve it. Now all we need to do is figure out how the edges of the workspace map to eye-space. We drive the robot until the image of the object appears inside the projected image of the workspace of the arm then simply stop the robot. The arm will do the rest by itself.

This is in stark contrast to conventional approaches. Typically, one would first correct for any distortions in the camera image, then perform the inverse optics calculations to find the target's coordinates. After this, one would plan a path to bring the robot to the desired point, and then perform the inverse kinematic calculations to find the necessary joint angles for the arm. Not only does our system achieve the same net effect as this method with substantially less work, it is also more robust in its performance since it can locally plan and replan the trajectories of both the robot and the arm.

6. Conclusions

We have described a working behavior-based control system for a mobile robot's arm. This controller has been successfully implemented on a collection of small on-board processors. The system described is extensible both in terms of hardware and in competence. The multi-agent approach has allowed us to incrementally build and debug progressively more sophisticated control systems. To use this approach, however, we have had to invent new forms of spatial representation and new schemes for the coordination of independent agents. Currently, there seems to be no limit on the potential size of such systems.

References

[Agre and Chapman 87] Phil Agre and David Chapman, "Pengi: An Implementation of a Theory of Action", *Proceedings of AAAI-87*, Seattle WA, 268-272.

[Brooks 86] Rod Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal Robotics and Automation*, RA-2, April, 14-23.

[Brooks and Connell 86] Rod Brooks and Jon Connell, "Asynchronous Distributed Control System for a Mobile Robot", *SPIE Proceedings*, Vol. 727, October, 77-84.

[Brooks, Connell, and Flynn 86] Rod Brooks, Jon Connell, and Anita Flynn, "A Mobile Robot with On-board Parallel Processor and Large Workspace Arm", *Proceedings of AAAI-86*, Philadelphia PA, 1096-1100.

[Brooks, Connell, and Ning 87] Rod Brooks, Jon Connell, and Peter Ning, "Herbert: A Second Generation Mobile Robot", MIT AI Lab, Cambridge MA, AIM-1016.

[Connell 87] Jon Connell, "Creature Design with the Subsumption Architecture", *Proceedings of IJCAI-87*, Milan Italy, 1124-1126.

[**Connell 88**] Jon Connell, "Task-Oriented Spatial Representations for Distributed Systems", MIT Dept. of Elect. Eng. and Comp. Sci., Cambridge MA, Ph.D. thesis (forthcoming).

[**Flynn 85**] Anita Flynn, "Redundant Sensors for Mobile Robot Navigation", MIT AI Lab, Cambridge MA, TR 859.

[**Giralt, Chatila, and Vaisset 84**] G. Giralt, R. Chatila, and M. Vaisset, "An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots", in *Robotics Research: The First International Symposium,* M. Brady and R. Paul (eds.), MIT Press, Cambridge MA, pp. 191 - 214.

[**Hywel-Snaith 87**] Martin Hywel-Snaith, "A Reductionist Study into the Behavioural Approach to Autonomous Robots", The Technology Applications Group, Alnwick UK, TAG discussion paper MHS/AR1.

[**Minsky 86**] Marvin Minsky, "The Society of Mind", Simon and Schuster.

[**Shafer, Stentz, and Thorpe 86**] S. Shafer, A. Stentz, and C. Thorpe, "An Architecture for Sensor Fusion in a Mobile Robot", *Proceedings of the IEEE International Conference on Robotics and Automation,* pp. 2002 - 2011.