How to Build Complete Creatures Rather than Isolated Cognitive Simulators

Rodney A. Brooks
Massachusetts Institute of Technology, Artificial Intelligence Laboratory

INTRODUCTION

How can we build artificial creatures which inhabit the same world as us? To me this has always been the romance of Artificial Intelligence.

Artificial Intelligence as a discipline has gotten bogged down in subproblems of intelligence. These subproblems are the result of applying reductionist methods to the goal of creating a complete artificial thinking mind. In Brooks (1987) 1 have argued that these methods will lead us to solving irrelevant problems; interesting as intellectual puzzles, but useless in the long run for creating an artificial being.

Thus, my goal over the last few years has been to build complete creatures that can exist in a dynamic people-populated world.

But having rejected reductionism as an approach how can this goal be achieved? We can begin by making the following observation inspired by nature.

Trying straight up for human level intelligence is obviously difficult and is not necessarily the only valid approach. Evolution, after all, built a large number of prototypes; less intelligent than humans before incrementally reaching the intelligence of *homo sapiens*. In building artificial creatures we might well make progress by starting with low expectations and gradually building upon our experience. Note that we are not saying we should build creatures in simple worlds and then gradually increase the complexity of the worlds. Rather we are arguing for building simple creatures in the most complex world we can imagine and gradually increasing the complexity of the creatures.

One approach then, is to aim initially for the intelligence level of lower animals (e.g., insects) and respect the constraints that biology seems to have worked under. This approach may not produce the "optimal" intelligence in some sense, but it may have a chance where other approaches have failed.

In looking at lower animals one sees that most of their activity is concerned with rather mundane aspects of simply existing in the world (e.g., Moravec, 1984). Very little of their activity has an obvious component that would match any piece of existing work in Artificial Intelligence. To list just of few examples, it seems highly unlikely that a house fly is:

• recovering three dimensional surface descriptions of all the objects within its field of view,

• reasoning about threats from a human poised with a fly swatter, in particular about the human's goal structures, intents or beliefs,

• representing prototypes and instances of humans (or coffee pots, or windows, or napkins),

• making analogies concerning suitability for egg laying between dead pigs and other dead four legged animals, or

• constructing naive physics theories of how to land on the ceiling.

It seems much more likely that:

• there is very close connection of sensors to actuators (especially given the low speed of neural hardware and the fast reaction time of the fly)

• there are pre-wired patterns of behavior

• the fly uses simple navigation techniques

• and it is almost characterizable as a deterministic machine.

In this chapter we show how systems with such capabilities can be built from a collection of simple machines, with no shared representations, no central control, and only very low switching rates and low bandwidth communication.

Agre and Chapman (1987) have gone further and argued that much of human activity is simply a matter of following routines and that in fact very little of the traditional AI sorts of processes mentioned above go on in humans for much of their mundane day to day activity. They implement their systems in combinational circuits.

Like Minsky (1987) we believe that human level coherence during many activities may only be in the eye of the beholder; the behavior is generated by a large collection of simpler behaviors which do not have the rationality generating them that we might normally attribute to humans.

In fact, we hypothesize that all human behavior is simply the external expression of a seething mass of rather independent behaviors without any central control or representations of the world. Maybe there is only chaos from which order appears to emerge.

THE SUBSUMPTION ARCHITECTURE

The subsumption architecture Brooks (1986) is a parallel and distributed computation formalism for connecting sensors to actuators in robots. A traditional way of describing these connections would be to say the subsumption architecture provides a way of writing intelligent control programs for mobile robots.

One writes a subsumption program by specifying layers of networks of *augmented finite state machines.* These are finite state machines augmented with timers which can be set to initiate a state change after some fixed time period has passed.

The two key aspects of the subsumption architecture are that (a) it imposes a layering methodology in building intelligent control programs, and that (b) within each network the finite state machines give the layer some structure and also provide a repository for state.

Subsumption Details

Although there are a number of variations of the subsumption architecture in active use (see below), they all share a common base.

Each augmented finite state machine has a number of states and a set of input and output ports. Each input port has a buffer register that always contains the most recently arrived message on that port. The networks are built by wiring output ports of machines to inputs of others. Messages are sent over these wires. The messages on a given wire are all the same number of bits long (or wide).

Beside input registers a finite state machine can have additional *instance variable* registers in which extra state can be stored.

There are four types of states possible in a finite state machine:

• An *output* state outputs a message to a designated port, then switches to a specified state. The message is a *peripheral function* of input and instance variable registers. In the early versions of the subsumption architecture a peripheral function was allowed to be an arbitrary supplied piece of Lisp code.

• A *conditional-dispatch state* tests the value of a peripheral function and conditionally branches to one of two designated states.

• A self state computes a peripheral function of input and instance variable registers, sets an instance variable register to the new value and branches to a designated state.

• An *event-dispatch* state waits in parallel for a number of different events and when one happens branches to the designated state. Each event is a boolean combination of message arrivals on input ports and the expiration of a timer initialized when the state was first entered.

Examples of all these types of states can be seen in Figure 8. 1. Additionally there is a *reset* line into each finite state machine; a message arriving on this line forces the machine into a distinguished state without resetting any of the register contents.

```
(defmodule avoid 1
  :inputs (force heading)
  :outputs (command)
  :instance-vars (resultforce)
  :states
     ((nil (event-dispatch (and force heading) plan))
     (plan (setf resultforce (select-direction force heading))
           go)
     (go (conditional-dispatch(significant-force-p resultforce 1.0)
                              start
                              nil))
     (start (output command (follow-force resultforce))
            nil)))
```

FIG. 8.1.

There are two other types of connection interaction allowed. An output port can have an *inhibiting* side tap added, where any message arriving on the side tap inhibits all output on the port from some specified time period. Any existing wire can have a *suppressing* side tap placed on it, where a message arriving on the side tap is propagated along the wire as though it had originated at the original source, and furthermore all messages from the original source, for some specified time period, are totally suppressed and discarded.

Figure 8.2 shows a schematic representation of a finite state machine with inputs, outputs and a reset, along with a suppressed input and an inhibited output.

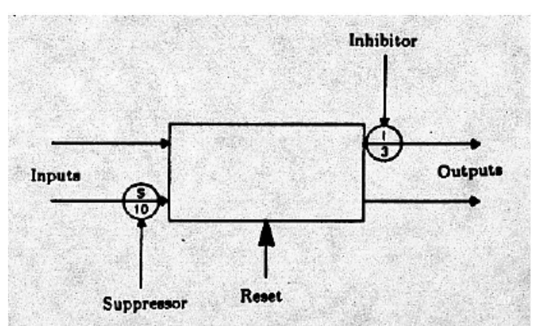

FIG. 8.2. A module has input and output lines. Input signals can be suppressed and replaced with the suppressing signal. Output signals can be inhibited. A module can also be reset to state NIL.

Variations on the Theme

The details of the subsumption architecture are very fluid, and indeed many people now use distinct versions, although all are strongly of the above described flavor.

Connell (1987) and Brooks (1988a) have explored the idea of simplifying all peripheral computations to the point where they are implementable in combinatorial logic or table lookup. For creatures with insect level intelligence this has not proved to be a serious constraint. It removes an ugly wart on earlier versions of the subsumption architecture by removing an escape mechanism into Turing-equivalent arbitrary computations, and hence puts a bound on the computational power necessary to implement a subsumption program.

Connell (1988a) has proposed it subsumption model where all messages have a continuous nature. When one layer wants to subsume another it must continually send messages to keep control. Messages might have the flavor of “go forward,” “go forward” “go forward,” etc. Once the higher layer is satisfied it stops sending this message and hence relinquishes control. This version of the subsumption architecture does not make use of inhibition or resetting and suppression nodes have no timeout period. The new version is implementable, in the original version however. Viola (1988) has reimplemented a number of earlier creatures in the continuous model of the subsumption architecture. It seems to simplify the subsumption programs markedly.

Cudhea (1988) has added a layer of abstraction to the subsumption architecture which lets users define programs in terms of instantiating finite state machine schemas. Ibis makes it easy to write subsumption programs where there are many instances of a single finite state machine.

Horswill and Brooks (1988) have augmented the underlying subsumption architecture with high bandwidth vision busses. Simple means of combination (such as MUXes and logical combination) of vision signals, along with local operators, delay elements and region to coordinate mapping functions, allow the implementation of a number of low level visual navigation techniques useful for insect level navigation. Standard subsumption architecture finite state machines monitor and switch the visual pathways, and translate the outputs into actuator commands.

THE PHILOSOPHY AND CONSEQUENCES OF SUBSUMPTION

Given these mechanics of the subsumption architecture a wide range of programming styles are possible. However there are some underlying considerations which distinguish a "good" subsumption program from a "bad" subsumption program.

The design of the subsumption architecture has been influenced by a philosophy of no global world models and no traditional Al planning. In turn, the experiments we have done with real robots controlled by the subsumption architecture have fed back on this philosophy, refining it and our understanding of the essential aspects of the subsumption architecture.

The underlying architecture is very distributed. There is no "free" communication network or any shared memory between computational elements. Any communication path must be made quite explicit by specifying a wire. It is thus difficult to maintain a central world model. Indeed it often becomes easier to use the world as its own model, and sense the pertinent aspects of the world when it is necessary. This is a good idea as the world really is a rather good model of itself. It automatically adds robustness to the system as there is neither a tendency for the world model to be out of date, nor are large amounts of computation poured into making sure that its not. We take this idea even further and often actually use the world as the communication medium between distributed parts of the subsumption program. Thus one layer senses what really happened in the world, rather than being told what another layer expects to happen.

Given that there is no world model there is also no place for traditional AI planning which examines a world model and reasons about consequences of actions. Rather, in the subsumption architecture it is more natural to locally react to sensed aspects of the world, and let a pre-wired priority scheme resolve any conflicts generated within the distributed system. It is entirely plausible for different parts of the system to "believe" wildly inconsistent things about the world. Of course belief is all in the mind of an outside observer as there are no explicit symbolic representations of any believed facts within the subsumption architecture.

Lastly, with no central world model there is no need for sensor fusion in the usual sense of the phrase. There is no "perception" system which delivers descriptions of the world to a "central" system which controls and "actuation" system, In the subsumption architecture the fusion of data from different sensors, or even from different processing applied to the same sensor (e.g. stereo and motion algorithms applied to the same carriers inputs) data, does not happen in the "perception" end at all. Individual strands of perceptual data are delivered to individual subsumption layers and then actuator commands are generated. Fusion happens in resolving conflicts between these commands.

EXAMPLES

In this section we briefly review some successful creatures built with the subsumption architecture and highlight the ways in which they have exploited or epitomize that architecture. Finally we outline a subsumption program for a complex visually guided creature (named Seymour) that is currently under development.

Allen

Our first robot, Allen, had sonar distance sensors and odometry onboard and used an offboard lisp machine to simulate the subsumption architecture. In Brooks (1986) we described three layers of control implemented in the subsumption architecture. The wiring diagram is shown in Figure 8.3.

The first layer let the robot avoid both static and dynamic obstacles; Allen would happily sit in the middle of a morn until approached, then scurry away, avoiding collisions as it went. The internal representation used was that every sonar return represented a repulsive force with an inverse square drop off in strength. The vector sum of the repulsive forces, suitably thresholded, told the robot in which direction it should move. An additional reflex halted the robot whenever there was something right in front of the robot and it was moving forward (rather than turning in place).

The second layer made the robot randomly wander about. Every 10 seconds or so, a desire to head in a random direction would be generated. That desire was coupled with the instinct to avoid obstacles by vector addition. The summed vector suppressed the more primitive obstacle avoidance vector, but the obstacle avoidance behavior still operated, having been subsumed by the new layer, in its account of the lower level's repulsive force. Additionally the halt reflex of the lower level operated autonomously and unchanged.

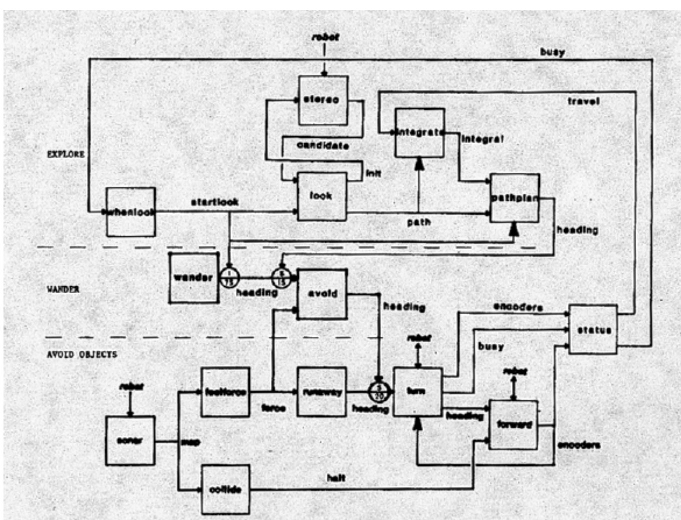

FIG. 8.3.

The third layer made the robot look (with its sonars) for distant places and try to head towards them. The third layer monitored progress through odometry, generating a desired heading which suppressed the direction desired by the wander layer. It was thus fed into a vector addition with the instinctive obstacle avoidance layer. The physical robot did not therefore remain true to the desires of the upper layer. The upper layer had to watch what happened in the world, through odometry, in order to understand what was really happening in the lower control layers, and send down correction signals.

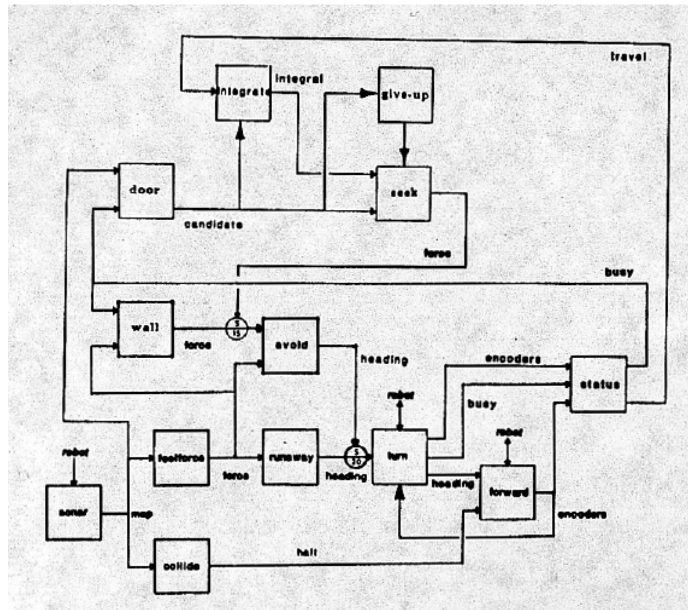

FIG. 8.4.

In Brooks and Connell (1986) we described another set of layers for the robot Allen. See Figure 8.4 for the wiring diagram. The first was identical; avoiding obstacles both static and dynamic. The second layer implemented wall following by treating a wall to the right, say, of the robot as an attractive force slightly ahead and to the right of the robot. This attraction fought with the repulsive force of the wall, and together they formed an attractive vector which made the robot hug the wall. The wall follower breezes right past open doors. A third layer examines the sonar data looking for doorways. When it sees one it sets up a goal as in the previous set of experiments, and servoes through the doorway using odometry. Now however the repulsive forces from the lowest obstacle avoidance layer are crucial to line the robot up with the center of the doorway so that it can squeeze through. The door finder only has a very rough idea of where the door is, so the third layer only has a very crude world model. The lowest layer uses the world as its own best model in order to get the robot through the door. Note that the upper layer does not involve the lower layer as a subroutine. The lower layer is just doing its job as best it understands it as it always has. Figure 8.5 shows the behavior of a simulated version of Allen with one, two, and three control layers activated.

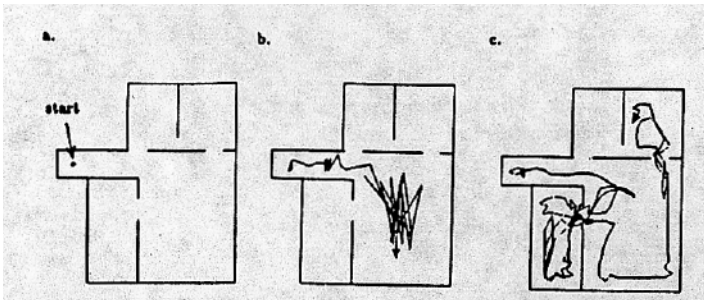

FIG. 8.5. The path followed by the robot. a. The starting point in a suite of four rooms. b. The wandering system. c. The wall following system.

Tom and Jerry

Tom and Jerry (Connell, 1987) were two identical robots built to demonstrate just how little raw computation is necessary to support the subsumption architecture. A three layer subsumption program for the robots is shown in Fig. 8.6. All data paths are one bit wide, and the whole program is implemented on a single 256 gate PAL (Programmable Array of Logic). Tom and Jerry physically are toy cars with three one bit infrared proximity sensors mounted on the front of them, and one identical sensor mounted at the rear. The sensors are individually tuned to a proximity distance at which they will fire. The central front sensor fires only on much closer objects than the two others which point slightly outwards.

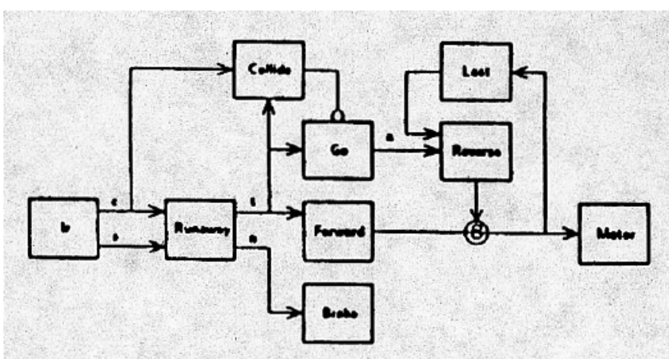

FIG. 8.6.

The lowest layer of Tom and Jerry implements our standard pair of behaviors, using a vector sum of repulsive forces from obstacles and a halt reflex to stop when something is too close ahead, as detected by the central front looking sensor. There are extra complications in that we need to use the subsumption architecture to implement an active braking scheme because of the high speed of the robots relative to their sensor sensitivities. Tom and Jerry's second layers are much like Allen's original second layer—an urge to wander about, implemented by an attractive force which gets added to the repulsive forces from obstacles. The third layer detects relatively moving objects using the front three sensors. When something is detected it is attracted towards them. The lower level collide behavior stops the robot from actually hitting the target however. While the robot is chasing a target the wander behavior is suppressed.

We see with Tom and Jerry both the notion of independent behaviors combining without knowing about each other (chasing obstacles but staying back from them a little ways) and the idea again of using the world as its own best model. It demonstrated that the subsumption architecture could be compiled (by hand) down to the gate level, and also that it could be run at clock speeds of a few hundred Hertz. This has inspired us to automate the compilation process (Brooks, 1988b).

Herbert

Herbert (Brooks, Connell, & Ning, 1988) is a physically much more ambitious robot which is now physically complete. It has a 24 processor distributed loosely coupled onboard computer to run the subsumption architecture. The processors are slow CMOS (low electrical power; an important consideration when carrying batteries around to power them) 8 bit microprocessors, which can communicate only by slow serial interfaces (maximum 10 packets each 24 bits wide per second). Onboard Herbert the wires in the diagrams shown in this paper for subsumption programs have physical embodiments as copper wires which provide the medium to support the serial sensing of messages.

Herbert has 30 infrared proximity sensors for local obstacle avoidance, an onboard manipulator with a number of simple sensors attached to the hand, and a laser light stripping system to collect three dimensional depth data in 60 degree wide swath in front of the robot out to a range of about 12 feet. A 256 pixel wide by 32 pixel high depth image is collected every second. Through a special purpose distributed serpentine memory, some number of the onboard 8 bit processors are able to expend about 30 instructions to each data pixel. By linking the processors in a chain we are able to implement quite good performance vision algorithms.

Connell (1988b) is programming Herbert to wander around office areas, go into peoples offices and steal empty soda cans from their desks. He has demonstrated obstacle avoidance and wall following, real-time recognition of soda can like objects and desk like objects, and a set of 15 behaviors (Connell, 1988b) which drive the arm to physically search for a soda can in front of the robot, locate it and pick it up. These fifteen behaviors are shown as fifteen separate finite state machines in Figure 8.7.

Herbert shows many instances of using the world as its own best model and as a communication medium.

The laser-based *table-like-object* finder initiates a behavior which drives the robot closer to a table. It doesn't communicate with any other subsumption layers. However when the robot is close to a table there is a better chance that the laser-based *soda-can-like-object* finder will trigger. In turn it centers the robot on the detected object, but does not communicate anything to other subsumption layers. The arm control behaviors notice that the robot is stationary, and reach out looking for a soda can. The advantage of this approach is that there is no need to set up internal expectations for what is going to happen next; this means that the control system can both (a) be naturally opportunistic if fortuitous circumstances present themselves, and (b) it can easily respond to changed circumstances, such as some other object approaching it on a collision course.

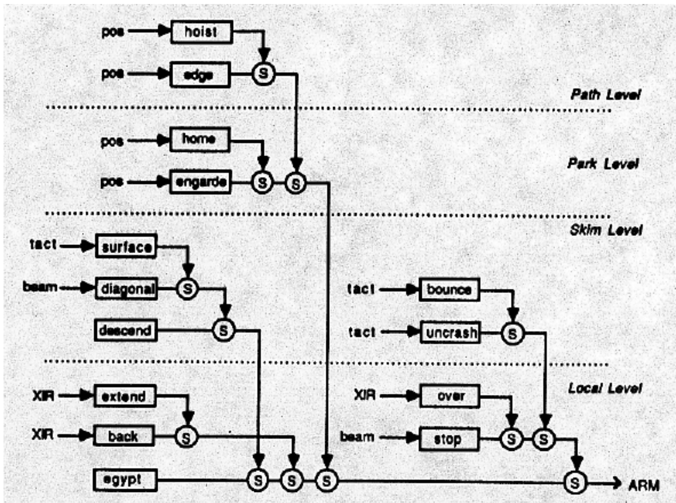

FIG. 8.7.

Likewise the arm and hand do not communicate directly. The band has a grasp reflex that operates whenever something breaks an infrared beam between the fingers. When the arm locates a soda can with its local sensors it simply drives the hand so that the two fingers are on either side of the can. The hand then independently grasps the can. Given this arrangement it is possible for a human to hand a soda can to the robot. As soon as it is grasped the arm retracts—it doesn't matter whether it was a soda can that was intentionally grasped, or one that magically appears.

Seymour

Seymour is a new robot we are building with all onboard processing to support vision processing of 8 to 10 low resolution cameras at approximately 10 frames per second. The cameras will feed into different subsumption layers which will act upon those aspects of the world they perceive. For instance, one layer might use a camera looking at ceiling lights to direct the robot down a corridor. Another might use motion stereo to detect obstacles in the path of the robot and force deviations from the simple corridor following path.

While the robot and its computers are under construction we have begun testing a number of low level visually guided navigation routines on Allen using an offboard Lisp machine to do the computations. We will transfer these routines to the robot when it is built.

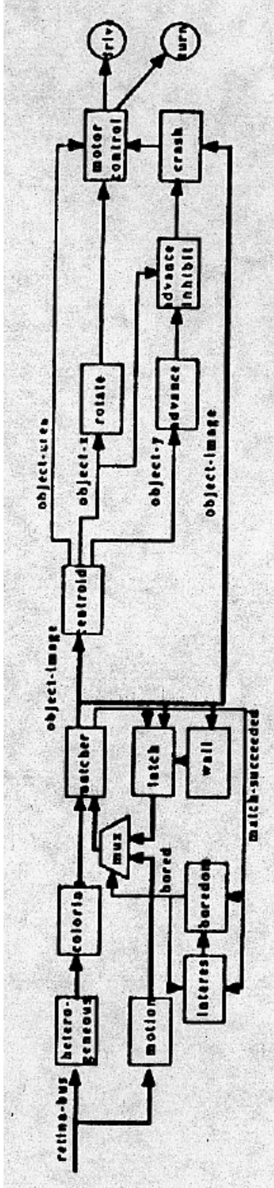

FIG. 8.8. Diagram of approach-and-follow system.

Horswill and Brooks (1988) describe a subsumption program that controls two simple and unreliable visual processing routines to producer a reliable behavior which follows moving objects using vision. The subsumption network is shown in Figure 8.8. One vision process tracks a single moving blob. It gets bootstrapped by another process which overlays the blob image with an indication of where motion is seen. The robot then tries to servo a selected blob to stay in a fixed location in image coordinates. The blob tracker often loses the blob it is tracking. The motion finder produces a lot of noise especially when the robot is moving. But between the two of them they let the robot reliably follow a moving object (any moving object; we have seen the robot chase a pink plastic flamingo and a black trash can dragged by a string, a radio controlled toy blue car on a blue floor, a grey notebook on a grey carpeted floor, and a drinking mug moved around by hand), by switching back and forth between the visual routines as either one fails. The subsumption program nowhere has the notion of an identifiable object internally, but to an outside observer it certainly appears to follow a moving object well.

CONCLUSION

By trying to build complete creatures we have found it useful to place a different emphasis on many aspects of intelligence than has been traditional in AI research. The key problems in building a complete creature are:

• providing fast and adequate response in a dynamic changing environment,

• providing a way to make sense of sensory data in an incompletely understood world, and

• providing a mechanism where the goals of a creature are not routinely overwhelmed in dealing with the interruptions provided by the world.

At least for simple creatures we have found little use for complete world models, planning, search, explicit knowledge representation, truth maintenance or control of reasoning.

Inspired by Minsky (1986) we hypothesize that the same will hold true as we move up the evolutionary train in the complexity of creatures we build. In our more radical moments we believe that this will hold true all the way through to human level intelligence. We are further inspired by the following observation.

Nature has shown us that intelligence is possible with very low switching speeds in the substrate used for computation. Indeed *every single example* of biological systems with any form of cognition operates on hardware that can propagate signals at no more than a kiloHertz or two. Yet all of these systems can react and operate in a fraction of a second. But they can do more than just react in those time frames. Humans, for instance, can often perceive, reason, understand and plan actions in subsecond time frames. In fact they can produce continuous streams of speech at many words per second. Thus we have an existence proof that it is possible to achieve intelligence with very shallow computational processes. Interestingly we have no existence proof that it is possible to do it any other way. The subsumption architecture similarly provides intelligence to simple creatures with hardware that need only be clocked at a few hundred Hertz (Brooks, 1988b).

ACKNOWLEDGMENTS

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the research is provided in part by the University Research Initiative under Office of Naval Research contract N00014-86-K-0685, in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-85-K-0124.

REFERENCES

Agre, P., & Chapman, D. (1987). Pengi; An implementation of a theory of activity. *AAAI-87*, Seattle, WA, 268-272.

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

Brooks, R. A. (1987). Intelligence without representation. Preprints of the *Workshop on Foundations of Intelligence.* MIT: Endicott House.

Brooks, R. A. (1988a). Simple Computations Suffice, *in preparation*.

Brooks, R. A. (1988b). A Silicon Compiler for the Subsumption Architecture, *in preparation*.

Brooks, R. A., & Connell, 1. H. (1986). Asynchronous Distributed Control System for a Mobile Robot. *SPIE Vol. 727 Mobile Robots*, 77-84.

Brooks, R. A., Connell, 1. H., & Ning, P. (1988). Herbert: A Second Generation Mobile Robot. Report No. MIT AIM-1016.

Connell, J. H. (1987). Creature Building with the Subsumption Architecture. *IJCAI-87*, Milan, Italy, 1124-1126.

Connell, J. H. (1988a). A Behavior-Based Am Controller, to appear *SPIE Mobile Robot Conference,* Cambridge, MA, Nov.

Connell, J. H. (1988b). Task Oriented Spatial Representations for Distributed Systems. MIT Ph.D. thesis in preparation.

Cudhea. P. H. (1988). Describing the Control Systems of Simple Robot Creatures. MIT S.M. thesis, June.

Horswill, 1. D., & Brooks, R. A. (1988). Situated Vision in a Dynamic World: Chasing Objects, to appear *AAAI-88,* St. Paul, MN.

Minsky, M. L. (1986). Society of mind. New York: Simon and Schuster.

Moravec, H. P. (1984). Locomotion, vision and intelligence. In Brady & Paul (Eds.), *Robotics Research* (pp. 215-224). Cambridge, MA: MIT Press.

Viola P. (1988). A Syntax for Mobot Description with Strong Semantics: Evolution the Easy Way. MIT S.D. thesis.