

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI[®]

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

DISSERTATION

**INTERACTIVE REPRESENTATION BASED MINIMALIST
ROBOT**

Submitted by
Keith L Bearden
Department of Mechanical Engineering

In partial fulfillment of the requirements
For the degree of Doctor of Philosophy
Colorado State University
Fort Collins, Colorado
Fall 2000

UMI Number: 3002066



UMI Microform 3002066

Copyright 2001 by Bell & Howell Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

COLORADO STATE UNIVERSITY

September 29, 2000

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED
UNDER OUR SUPERVISION BY KEITH L. BEARDEN ENTITLED
INTERACTIVE REPRESENTATION BASED MINIMALIST ROBOT BE
ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Mark Mowack

Kiroshi Sakurai

P. W. Chen

Q. R. O. T. X

Advisor

Allen Kimpf

Department Head

Abstract of Dissertation

Interactive Representation Based Minimalist Robot

Behavior-based robotics have made great advancements in the area of autonomous robotic control. The Subsumption Architecture and its derivatives have proven to be a most effective architecture for autonomous navigation in complex, dynamic environments. Subsumption, however, has no interconnection between the behaviors and no flexibility for the hard-wired behaviors, which drastically limit the ability of subsumption to achieve high level goals. Coupled to this lack of interconnection is the inability of subsumption, or any currently available architecture, to possess system detectable error.

This research proposes a fundamentally different type of representation called Interactive Representation. Systems that possess Interactive Representation do not explicitly represent the external environment; these systems represent interactive possibilities with the environment. This development, based only upon interactive possibilities, opens up the ability of a system to possess system detectable error. An architecture was developed based on Interactive Representation, the Interactive Representation Architecture (IRA). Error is based upon a functional definition of task achievement, not upon detection of a sensory fault. There is never any attempt to compare one sensor to another sensor for validation. An error is then defined as anything

that causes the circularity of the organization of a system to become broken. This definition of error is implicit instead of explicit, which allows the IRA based systems to truly possess system detectable error and to be able to recover from those errors without the use of an external observer, without resorting to plans, and without increasing the complexity or reducing the speed. Both real and simulated systems utilizing this implicit definition of error were proven to be superior to systems not possessing system detectable error even when there were no faults present.

Keith L. Bearden, Maj, USAF
Department of Mechanical Engineering
Colorado State University
Fort Collins, Colorado 80523
Fall 2000

Table of Contents

Abstract of Dissertation.....	iii
Table of Contents	v
List of Figures.....	x
List of Tables	xiii
1 Introduction	1
1.1 Research Goals	2
1.2 Dissertation Organization	3
1.3 Definitions	4
2 Philosophy	9
3 Architectures.....	13
3.1 Subsumption	13
3.2 AnSim Error Investigation.....	20
4 Interactive Representation	24
4.1 Interactive Possibilities.....	25
4.2 Motivation	26
4.3 Interactive Error.....	27

5	Error	29
5.1	Function	30
5.2	Definitions and Theory	31
5.3	Current Error Detection Methodologies	34
5.3.1	<i>Examples of Fault Detection</i>	36
5.4	Behavior Based Error Detection	39
5.5	The Medicinal Leech	42
5.6	The Human Model	45
5.7	Fault Detection	46
5.7.1	<i>Transition Matrix</i>	47
5.7.2	<i>Observation</i>	50
5.7.3	<i>Behavior Error Detection</i>	51
5.8	Error Detection Conclusions	52
6	Interactive Representation Architecture	55
6.1	Finite State Machines	58
6.2	Performance Demonstration	64
7	Autopoiesis	69
8	IRA AnSim Simplified Simulation	72
8.1	IRA Whisker Error	73
8.2	IRA Distance Sensor Error	74
8.3	The Byzantine Generals Problem	76
9	Full IRA AnSim Simulation	78

9.1	IRA AnSim Induced IR Error.....	78
9.2	IRA AnSim Induced IR and Whisker Error	81
9.3	IRA AnSim Two Behavior Simulation	84
9.3.1	<i>Two Behaviors – No Induced Errors</i>	85
9.3.2	<i>Two Behaviors – With Induced Faults</i>	87
9.4	IRA AnSim Three Behavior Simulation	88
9.4.1	<i>Three Behaviors – No Induced Faults</i>	88
9.4.2	<i>Three Behaviors – With Induced Errors</i>	93
10	Design Guidelines	95
10.1	The Organization	96
10.2	The Structure	98
10.3	Implementation.....	101
10.3.1	<i>Organization</i>	101
10.3.2	<i>Structure</i>	102
10.3.3	<i>Lessons Learned</i>	106
10.3.4	<i>Performance Results</i>	107
10.3.4.1	NO INDUCED FAULTS	108
10.3.4.2	PHOTORESISTOR FAULT	109
10.3.4.3	BUMPER FAULT.....	110
10.3.4.4	OVERALL PERFORMANCE RESULT	111
11	Redundant Measurement of a Vibrating Beam	113
11.1	<i>Mathematica</i> Beam Deflection Simulation	114
11.2	<i>LabView</i> Beam Deflection Experiment	119
11.2.1	<i>Experiment 1: All three gauges good</i>	122

11.2.2	<i>Experiment 2: One bad gauge</i>	123
11.2.3	<i>Experiment 3: Two bad gauges</i>	125
11.3	Vibrating Beam Conclusions	127
12	Dynamic Control of a Vibrating Beam	130
12.1	Experimental Set up	131
12.2	Experimental Algorithm	133
12.3	Experimental Results	135
12.3.1	<i>Active Damping with No Error Detection</i>	135
12.3.2	<i>Active Damping with Error Detection</i>	138
12.4	Statistical Analysis	139
12.4.1	<i>Statistical Analysis of Results with No Error Detection</i>	140
12.4.2	<i>Statistical Analysis of Results with Error Detection</i>	140
12.4.3	<i>Detection to No Error Detection – No Error</i>	141
12.4.4	<i>Error Detection to No Error Detection – Induced Error</i>	142
12.5	Active Damping Conclusions	142
13	Discussion	145
14	Conclusion	150
15	Future Work	155
	References	156
	Appendix A: IRA vs Subsumption	163
	Appendix B: Goal Oriented Results	171

Appendix C: Minimalist Robot Control Code	180
Appendix D: AnSim Users Manual.....	189
Basics.....	189
Input Files	190
<i>World.dat</i>	190
<i>Animat.dat</i>	192
<i>Behavior.dat</i>	194
Animat.cpp	196
Controller.cpp	196
Conclusion	197
Appendix E: <i>Mathematica</i> Beam Deflection Simulation	198
Appendix F, LabView Virtual Instruments.....	221
No Error Correction.....	221
Comparison Error Detection.....	222
Behavior-Based Error Detection	223
Appendix G, Minimalist Robots	225
Appendix H: Active Damping Virtual Instruments	229
Appendix I: Determination of Damping Ratio.....	234
Appendix J: Hypothesis Tests and Power Determination.....	239

List of Figures

Figure 2.1, Linkages	10
Figure 3.1, Generic Subsumption Flow Chart	14
Figure 3.2, Trapped Trajectories.....	20
Figure 3.3, Animat Baseline	21
Figure 3.4, Whisker Error	22
Figure 3.5, IR Error.....	23
Figure 5.1, Leech Behavior.....	43
Figure 5.2, Explore Flow Chart	48
Figure 6.1, Interactive Representation Architecture	56
Figure 6.2, Subsumption Flow Chart.....	58
Figure 6.3, Transition Diagram Subsumption.....	59
Figure 6.4, Trapped Trajectories with Interconnection	62
Figure 6.5, Transition Diagram, IRA.....	63
Figure 6.6, Distribution of States.....	64
Figure 6.7, Subsumption Goal Directed	65
Figure 6.8, IRA Goal Directed.....	66
Figure 6.9, IRA vs Subsumption.....	67
Figure 8.1, IRA Whisker Error	73
Figure 8.2, IRA IR Error.....	74
Figure 8.3, IRA Escape from a Corner	75
Figure 9.1, AnSim Subsumption.....	78

Figure 9.2, IRA & Subsumption No Errors	79
Figure 9.3, IRA & Subsumption with Induced IR Error.....	81
Figure 9.4, IRA & Subsumption with IR and Whisker Error	82
Figure 9.5, IRA & Subsumption with IR and Whisker Failure	83
Figure 9.6, Two Behavior Subsumption Flow Chart.....	85
Figure 9.7, Two Behaviors No Errors.....	85
Figure 9.8, Two Behaviors – 30% Coverage.....	87
Figure 9.9, Three Behavior Subsumption Flow Chart.....	88
Figure 9.10, Three Behaviors, No Induced Errors.....	89
Figure 9.11, IRA Animat No Errors	90
Figure 9.12, Subsumption No Errors	90
Figure 9.13, Close-up of Trap.....	91
Figure 9.14, Three Behaviors – Trapping Region	92
Figure 9.15, Three Behaviors – 31% Coverage.....	93
Figure 10.1, The System	97
Figure 10.2, Action-Centered Design Model.....	98
Figure 10.3, Minimalist Robot.....	103
Figure 11.1, Beam Model	114
Figure 11.2, Three Good Gauges.....	115
Figure 11.3, Gauge One Fails	116
Figure 11.4, One Faulty, With and Without Error Correction.....	117
Figure 11.5, Two Faulty, With and Without Error Correction	118
Figure 11.6, CEA-13-240UZ-120 Data Sheet	119

Figure 11.7, Experimentation Setup	120
Figure 11.8, VI Front Panel	121
Figure 11.9, Tip Deflection, No Induced Errors	123
Figure 11.10, Tip Deflection, One Bad Gauge	124
Figure 11.11, Tip Deflection, Two Bad Gauges	125
Figure 12.1, QP20N Characteristics	131
Figure 12.2, Actively Controlled Beam.....	132
Figure 12.3, Active Damping Experimental Setup.....	133
Figure 12.4, Active Damping Front Panel.....	134
Figure 12.5, Active Damping with No Error Detection.....	136
Figure 12.6, Active Damping with Error Detection	138

List of Tables

Table 5.1, Transition Matrix	49
Table 5.2, System Level of Awareness.....	53
Table 6.1, Performance Test	67
Table 10.1, Demo Transition Matrix	104
Table 10.2, Minimalist Robot Parts List.....	107
Table 11.1, Experimental Equipment List	120
Table 12.1, Active Damping Equipment List	132
Table 12.2, Damping Ratio with No Error Detection.....	137
Table 12.3, Active Damping with Error Correction	139
Table 12.4, Hypothesis Test Results.....	143

1 Introduction

Behavior-based robotics has received a lot of attention in the last several years. The main goal of behavior-based robotics has been to develop truly autonomous systems. “There is a growing controversy about the extent to which intelligent actions are planned, as opposed to being quick reactions to immediate problems arising in the android’s current situation.” (Ford, 1995) Researchers have developed systems that mimic biology to achieve robust robotic control. (Fleischer, 1999, Moller, 1998, Sharpe, 1998) Researchers have looked at multiple methods for representing the environment the robot will operate in and, from these representations, multiples of differing control architectures have been developed. Among these are the subsumption architecture (Jones, 1993, Brooks, 1987), spreading activation network (Maes, 1990), the PDL robot architecture (Steels, 1994), the behavior network architecture (Cherian, 1994), three layer architectures (Gat, 1998) and the animate agent architecture (Firby, 1998). All of these architectures are different, but their goals are the same, making an autonomous robot more robust. Subsumption tries to remain simple, with a direct connection between the sensors and actuators and a hard-wired arbitration scheme to determine final control. Other methods try to add interconnection between the behaviors to achieve higher level goals. Some methods even add on additional processors to perform high level tasks, such as planning.

The one thing all of these control architectures are missing is the ability to self detect error and to handle that error. Most roboticists desire to eliminate error entirely by

planning for all possible eventualities and making error proof sensors. This is simply not possible. Others try to reduce error by insuring that their control architecture will not remain in an error state for very long. They do not attempt to detect error or mitigate error, only that they have some form of timeout to prevent the robot from remaining in any one state for extended periods of time. This, too, is not realistic. Others add multiple redundant sensors and some form of external observer to decide which are functional and which sensors are dysfunctional.

The behavior-based approaches are very robust as long as their sensors are not in error. The behavior-based systems that do attempt to detect error all do it in the same manner. They add an external observer and use multiple redundant sensors to compare sensor outputs. This system, better known as Built-In-Test (BIT), is effective. The government requires all new systems to be designed with BIT. BIT, however, violates the definition of system detectable error because the BIT is a system that is added on to the system that it is supposed to monitor. It is not actually part of the system. BIT will slow the system down while it is performing the error detection housekeeping.

1.1 Research Goals

There are several goals for this research. One, to develop a control architecture that will not try to explicitly represent the environment, will have the ability to detect when it is in error and, most importantly, will be able to move out of this error state to achieve high level goals. In order to accomplish the first goal, a useable definition of error is required. Two, since computer simulations are intrinsically error proof (sensors are perfect and never fail), the developed architecture will be applied to a minimalist autonomous robot to verify that the architecture can perform in the real world, instead of

just in a simulated world. During the development of the minimalist robot, design guidelines will also be produced to make it possible for future researchers to know what steps are required in the design of a robot. The minimalist robot and the control architecture should allow for the development of an agent of interest. Agents of interest are ones that have the potential for intelligence. They must possess the following characteristics: they must be autonomous, not needing support from humans; they must be self-sufficient, able to sustain themselves for extended periods of time; they must be embodied, which means that they must be physical systems interacting in the real world; they must be situated, which means that any information that the agent has about the environment came from the interaction between the agent and the environment. (Pfeifer, 1997) There is no such thing as a general-purpose machine, just as there is no such thing as general-purpose living being, therefore, the environment for the robot will have to be considered along with the task and robot. (Nehmzow, p10)

1.2 Dissertation Organization

The research is broken down into several sections. Section 2 details the philosophy behind real robots. The third section discusses architectures, specifically subsumption. Including what it is and what are its limitations. The fourth section details the epistemological theory of Interactive Representation. This form of representation possesses system detectable error without producing a circular argument based on an external observer or the comparison of one sensor value to another sensor value. The fifth section is a very detailed description of error. This section diverges from the typical descriptions of error based upon sensor failure, but instead bases error on system failure. This section also gives examples of biologics that possess this type of error detection and

correction. The sixth section develops a control architecture based on the Interactive Representation with system detectable error. This section also uses a simulated animat in a simulated world to demonstrate how the system detects error and how the system moves away from the error. The seventh section details how the IRA could be representative of an autopoietic system. This is a very important section; it shows how the IRA based animat has nearly all of the components necessary to be considered an autopoietic machine. The eighth and ninth sections are the IRA simulations. The simulations start with simple single behavior animats and finally end up with three behavior animats. All simulations compare and contrast the IRA animat with an identical subsumption animat. The tenth section applies this new architecture to a minimalist robot in the process of developing some design guidelines. The eleventh section applies this behavior-based definition of error to a simplified cantilever beam system. As Einstein said, “Everything should be made as simple as possible, but no simpler.” (Kurzweil, 1999) The final three sections detail what has come from the research and where the research will lead.

1.3 Definitions

Agent: “An entity that produces an effect.” (Nehmzow, 2000)

Architecture: An abstraction of a control system showing the connections and interconnections of the system that remain unchanged during the life of the system. (Cherian, 1994)

Autonomy: According to Webster, there are two possible definitions of autonomy, a) undertaken without outside control, and b) having the power of self-government. For this work, the second definition will be utilized. (Nehmzow, 2000)

The real test of any control architecture/machine is its ability to autonomously navigate in an undetermined dynamic environment. Almost any architecture can operate well in a computer simulation; the relatively small number of truly autonomous systems that can operate in a dynamic environment demonstrates the difficulty of real navigation. “It has proven a lot more difficult to build robots that move around without getting into trouble than it has to build chess playing machines!” (Nehmzow, 2000) There are several successful examples of autonomous robots but they are confined to “well-structured, corridor-type environments such as hospitals, energy producing and manufacturing plants, and/or warehouses.” (Pin, 1999) The aim of this research is to extend this to unstructured, dynamic environments.

Autopoiesis: “An autopoietic machine is a machine organized as a network of processes of production (transformation and destruction) of components that produces the components which: (i) through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them; and (ii) constitute it (the machine) as a concrete entity in the space in which they (the components) exist by specifying the topological domain of its realization as such a network.” (Maturana, 1980)

Behavior: Behaviors are at a higher level than Reactions. Behaviors can be seen as a collection of Reactions designed to achieve higher level goals. An example of a Behavior would be “Explore”. “Explore” would combine Avoid, Escape, and another Reaction to allow the robot to keep moving.

Correspondence: An internal representation of the environment, whether the representation is explicit or implicit.

Error (previously): “An undesired state within a system that may lead to improper operation.” (Lockner, 1990)

Error (this research): An error is anything that breaks the circularity of the organization of a system. This can be the result of a sensory fault, but is not restricted to sensory faults. This definition of error is implicit and confined to the internal operation of the machine.

Explicit: Something that is fully and clearly expressed.

Fault: “A fault is a known or hypothesized cause of error.” (Lockner, 1980)

Failure: “A failure occurs when the system loses its expected service or the inability to perform its desired function.” (Lockner, 1980)

Function: “The activity for which one is specifically fitted or employed.” (Webster’s, 1994) Function and task are only relevant to the observer and belong to the domain of his description. (Maturana, 1980)

Functional Redundancy: The use of different types of elements to perform the same task. This is most desirable form of redundancy, because “this precludes an environmental condition injurious to one redundant element from affecting the other.” (Landers, 1963)

Implementation: The development of a control architecture to an actual system, whether in computer simulation or for an actual autonomous agent.

Implicit: Something that is contained in the nature of the machine but is not readily apparent or directly expressed.

Organization: “The relations that define a machine as a unity, and determine the dynamics of interactions and transformations which it may undergo as such a unity, constitute the organization of the machine.” (Maturana, 1980)

Planning: The robot is preprogrammed with all possible environmental interactions. The robots of classical systems use planning to achieve high level goals. The robot will perceive its environment, determine a course of action and then execute the plan, Sense-Plan-Act. The plan is based upon a symbol manipulation representing the robot environment.

Reactions: Reactions are the direct coupling of what the environment provides and how the robot will respond. These are observable interactions between the robot and its environment. Examples of a Reaction are Avoid Obstacle or Escape Obstacle. These are better known as behaviors to classic behavior-based roboticists.

Representation: Classical roboticists attempt to fully represent the environment that the robot will operate in, explicitly. This requires vast (unattainable) processing power and infallible sensors. Behavior-based control attempts to achieve autonomy without an explicit mapping from the real world into the perceived world of the robot. Since there is no mapping, there is no actual representation of things, only potentials to be exploited or avoided.

Skills: The capabilities of an autonomous agent. These may appear as Move Forward or Move Back, etc. A skill is simply one of the functions that an agent can perform.

Structure: “The actual relations which hold among the components which integrate a concrete machine in a given space constitute the structure.” (Maturana, 1980)

System: A system must be an entity that can be separated from the rest of the universe with some form of boundary. Also, a system must be composed of interactive parts. (Karnopp, 1975) The system consists of the machine (living or mechanical), the environment in which it operates, and the task it performs. (Maturana, 1980)

System Detectable Error: For something to possess system detectable error, it must be able to determine when it is not achieving its primary function without the use of an external observer.

2 Philosophy

There are several definitions available for what a robot is supposed to be, but none of these definitions really seem to adequately describe the essential character of what a robot “is”. One of the earliest definitions of robots dealt with task completion and the ability to be reprogrammed. According to the Robot Institute of America, “A robot is a re-programmable, multi-functional manipulator (or device) designed to move material, parts, tools, or specialised (sic) devices through variable programmed motions for the performance of a variety of tasks.” (Nehmzow, 2000) Why does a robot need to be reprogrammed? Is it important, “for toasters today and televisions tomorrow?” Back in 1978, in Robots On Your Doorstep the lack of an adequate definition was discussed. “What is a robot? Well . . . the difficulty of answering that characterizes the whole discussion. Nobody quite knows, because nothing can be accepted as a complete, real robot has yet been produced and recognized.” (Winkless, 1978) Here is an MIT definition, “of a robot as an intelligent connection of perception to action”. (Jones, 1993) This implementation can take place with mechanical logic, microprocessors or neural networks. There is no mention of task accomplishment, just the linking of sensing and action. Is this really a robot? Dr. Wade Troxell puts forth the idea that a robot is “something” that performs Real Tasks in the Real World. (Troxell, 1999a) There is a significant link between the Robot, the Task and the Environment. These three all must

have serious consideration to achieve a robust robotic design, see Figure 2.1. (Ford, 1996)

The word “real” is being widely used, for instance, “. . . an increasing number of people agree that there is something ‘real’ about robotics, . . .” (Winkless, 1978) What is meant by “real”? A “Real Robot” is a thinking machine. (Winkless, 1978) Currently there are no thinking machines that are self-aware and self-replicating. Winkless (1978) does give a definition of intelligence that may bring “real” to possible fruition. Intelligence is “the ability to do something ‘appropriate’ under unpredictable conditions”. This is by far the most succinct definition of intelligence in the literature. By using this definition of intelligence, then a Real Robot will be able to do something appropriate – Real Task – under unpredictable conditions – Real World.

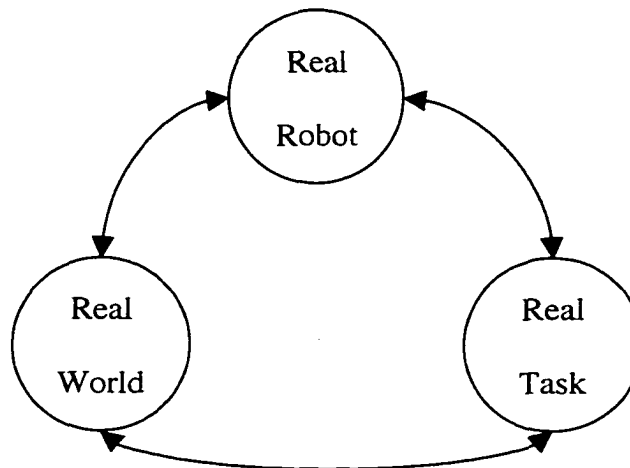


Figure 2.1, Linkages

Now that there exists a working definition of what a robot is, what does a robot need to be to accomplish this? The Theory of Affordances describes task accomplishment by animals as the result of a match between the animal and what the environment provides the animal. (Ford, 1976) Most eloquently stated by Thelen and

Smith (1994), “Seeking food and mates requires an accurate match between those properties in the environment necessary for the realization of those goals and the perceptual and motor apparatus of the animal. Development progresses toward such an adaptive match.” There should not be an attempt to explicitly map the environment to the robot’s specific action. (Ford, 1996) “It does not represent the external world (that is, the environment) in the sense of isomorphically mirroring it.” (Stry, 1995) This is extremely difficult, if not impossible, since a Real Robot will have a difficult time fully comprehending the Real World. Isomorphic maps of the environment require the use of sensors, which are unreliable and subject to occlusions. (Gat, 1998) Not only do sensors give incomplete information, a model is just a model, it can not perfectly represent what it is modeled after. (Ford, 1996) It is impossible to explicitly represent the complete environment. We have to look at not only what the environment affords the robot, but how the robot will “know” if what is in front of it is something to be avoided or something to be used. How will the robot “know” that what it perceives as an obstacle is truly an obstacle? There may be a mismatch between reality and the robots’ understanding of reality. System detectable error will allow the robot to know if it is in error. System detectable error is essential to building truly autonomous systems, but this type of error is impossible with contemporary approaches to representation. (Brickhard, 1998, Hamel, 1999) “No system, animal or machine can compare what an occurrent internal correspondence representation is supposed to be representing with what it is actually representing to determine if it is in error.” (Brickhard, 1999c) This makes fundamental sense. Just because a human thinks they see a COW, does that truly make it a COW? How can the human be sure? If a human cannot be sure, imagine how difficult

it will be for a electro-mechanical system programmed by a human to determine.
Interactive Representation will help to solve this dilemma.

3 Architectures

There are countless behavior-based control architectures being used in modern robotics. (Kortenkamp, 1998, Arkin, 1998) The key difference between behavior-based robotics and typical control architectures is the switch from explicit planning to environmental reaction. The behavior-based control system will consist of tightly coupled reactive systems, called behaviors. These behaviors are represented as finite state machines, loosely connected in a bottoms-up approach. A bottoms-up approach implies that all behaviors are active, with the higher level behaviors given the power to overwrite or subsume the lower level behaviors when sensory inputs demand higher level intervention. Some systems have hard-wired arbitration while others use a voting technique to select the active behavior. (Arkin, 1998)

The bottoms-up approach allows for extensibility, starting with the simplest system and then adding to the base system. Emerging from this bottoms-up approach can be a very complex system capable of dealing with dynamic environments.

The following section will detail a typical behavior-based approach, Subsumption, and show its advantages and limitations. Subsumption was chosen because it was the first behavior-based approach and all other behavior-based approaches have their roots firmly grounded in subsumption. (Brooks, 1987)

3.1 Subsumption

The Subsumption Approach developed by Professor Rodney Brooks at MIT entails “combining distributed real-time control with sensor triggered behaviors”. (Jones,

1993) Behaviors are the layers of control for the robots. A possible behavior for subsumption will take the form of Avoid or Escape. A direct link between sensor data and motor actuation is required. Conflicting sensor data is then arbitrated at the behavior level through a hard-wired arbitration scheme. The Subsumption Architecture used by MIT utilizes a flow chart hierarchy.

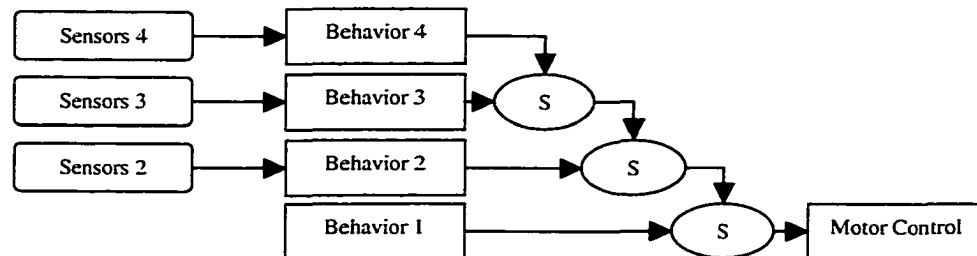


Figure 3.1, Generic Subsumption Flow Chart

The implementation of this flow chart is relatively simple. The microcontroller will first read all of the sensors. Each behavior will then calculate what the robot should do to satisfy the behavior. Finally, an arbitration subroutine will resolve all conflicts. A Bottom Up approach achieves conflict resolution. All behaviors are active, but the higher levels can subsume control from the lower levels. This is achieved by giving motor control to each behavior from lowest level first to the highest level. The lowest level behavior is always active; the higher levels may or may not be active depending on sensor input. For example, the arbitrate subroutine for Rug Warrior:

```
Void arbitrate()
{
    while(1) {
        If (cruise_output_flag ==1)
            {motor_input = cruise_output;}
        If (follow_output_flag ==1)
```

```

        {motor_input = follow_output;}
    If (avoid_output_flag == 1)
        {motor_input = avoid_output;}
    If (escape_output_flag == 1)
        {motor_input = escape_output;}
    sleep (tick);
}} (Jones, 1993)

```

All four behaviors will try to control the robot for every loop of the program. The base behavior will have control first and then in ascending order of precedence. If a higher level behavior is active, the higher behavior will overwrite the lower level behavior subsuming command of the robot.

This method is effective. There are numerous examples of this architecture effectively controlling mobile autonomous robots: Rug Warrior (Jones, 1993), Ants (McLurkin, 1995), Toto (Stein, 1995), Tom, Jerry, Genghis, Attila, Seymour, Tito, Polly, Cog (Arkin, 1998) and Squirt (Flynn, 1989), to name a few. In order to do something appropriate under unpredictable circumstances a system must be highly ordered. Order is defined as information that fits a purpose, and the purpose will be to survive. (Kurzweil, 1999) Survival is the robot's ability to autonomously navigate in an underdetermined complex environment. (Cherian, 1994) The solution to any problem should be as simple as possible, highly ordered, and may well require less data than that being input. (Kurzweil, 1999) This is the solution sought, as simple as possible to allow execution on a minimalist robot, but robust enough to navigate in underdetermined dynamic environments.

The subsumption architecture is effective for low levels of tasking. It works very well for obstacle avoidance or just cruising around.

“The most important problem we found with the subsumption architecture is that it is not sufficiently modular. Because upper layers interfere with the internal functions of lower level behaviors, they can not be designed independently and become increasingly complex. Subsumption of low-level behaviors by high-level behaviors is not always appropriate. Sometimes the low-level should override the higher levels.” (Gat, 1998)

A specific example for when a lower level behavior should supercede a higher level behavior is demonstrated in Section 9. In this instance, the robot is in the high level behavior of recharge, and one of the side station detectors is active causing the robot to turn toward the station. The robot, however, is too close to the station and cannot achieve an appropriate approach vector, so it continues to circle the station without ever recharging. If it were possible for the search behavior to gain control, just for a short while, then the robot would be able to move away from the station and hopefully gain an appropriate approach vector.

Subsumption does not have a mechanism to support high-level task achievement due to its requirement for hard-wired behavior selection. Also, Subsumption assumes a linear flow of control. Each behavior is layered on top of the next behavior. The Real World is not linear. Imposing a linear system on our robots will get them into situations that they can not extricate themselves from, see Figure 3.1. The hard-wiring of behaviors becomes very complicated and difficult for the programmer to use in real world applications as the tasks become more intricate. (Rao, 1998) We, therefore, need to use a slightly different architecture that allows for non-linear linkages and for high-level task achievement. Subsumption also lacks the ability to execute plans or to make actions based on previous actions since there is no internal state characterization or memory.

(Nehmzow, 2000) McLurkin (1995) gets around this by linking two subsumption architectures. As an example, he has the “it” and “not it” high-level behaviors and below them the lower level competencies required for the two behaviors. “It” and “not it” refer to the robots playing a game of tag. It is up to the programmer to cleverly arrange the behaviors for successful operation of the robot.

To overcome the apparent shortcomings of subsumption, five (Connell, Gat, Bonasso, Revel and Mahadevan) research groups developed control architectures to “fix” subsumption. (Kortenkamp, 1998) Their results were to add a task-based “planner”. They stacked a planner on top of subsumption to act as the mediator. The planner runs on a different clock than the subsumption control. The planner allows for rearranging subsumption to achieve high-level goals. This planning mechanism was successful but at what cost? Now two microcontrollers were needed and the environment cannot be explicitly represented, so planning which is necessarily based on a model of the environment will always be at a disadvantage. There will always be a situation that was not “anticipated” during our clever engineering practice. The additional processor also adds to the complexity of the system. The problem is that real world environments are dynamic, and so the plan has to be dynamic, and planning also requires that the sensors that perceive the environment are not fallible. Planning requires a representation of the environment derived from the sensors. From this, the planner generates a world model through symbol grounding. Any system that attempts to generate a world model based upon fallible and incomplete data is doomed to failure. Designers are finally realizing that an explicit representation of the environment will not ever be able to develop into

truly intelligent systems. (Nehmzow, 2000) There is a need to avoid planning, while maintaining simplicity and overcoming the limitations of subsumption.

In this research, a Behavior is a High Level Task. The next lower level are the Reactions, which are equivalent to things that were called behaviors by subsumption: Avoid, Escape, etc. Finally, at the lowest level are the Skills. A skill is what the robot can do at the most basic level. For example, a small autonomous robot can have the following Skills: Forward, Back, Right, and Left. These Skills can then be combined to form Reactions and Reactions combined to achieve Behaviors. A system very similar to the Chicago Robot Language (CRL) system will work nicely, but with the elimination of the planning that they felt was required. (Firby, 1998) “Planning is just a way of avoiding figuring out what to do next.” (Brooks, 1987) Behavior-Based Robotics can achieve high level goals without resorting to the doomed commitment of explicit planning and explicit environmental representation. A system that will allow for higher, more complex task completion, while still being modular and still being fast along the lines of the Behavioral Network Architecture (BNA), is required. (Cherian, 1994) The coupling of the sensor and actuator must be fast enough to remain stable, should be designed to know when it fails, and internal state should be avoided. (Gat, 1998) Gat, however, never discusses how “to know when it fails”, only to limit the amount of time that the robot will be in the “failed” state. There is also no reason to avoid internal state; history is a key ingredient to system detectable error and is a key ingredient in living systems. (Brickhard, 1999a, Maturana, 1980) The advantage to the BNA is that there is interconnectivity between behaviors. Each behavior can send Excitatory, Inhibitory or No influence feedback to the other behaviors. Each behavior is given the ability to

determine how to satisfy global goals. (Cherian, 1995) There does not have to be a linear behavior accomplishment. By allowing for inhibition and excitation, lower level behaviors may supercede higher level behaviors. This Inter-behavior connectivity will also allow a reaction to be shut off. Very complicated behavior is possible even for simple systems as long as the number of interconnections between subsystems is very large. (Casti, 1989) Although there is a great deal of attention paid to the idea of interconnection, there is very little information on how to make the interconnections in an intuitive manner.

The problem with subsumption is that there is no interconnectivity between the behaviors and consequently, no way for the system to detect functional errors. Each subsystem will perform its predetermined function, every time, whether the sensors are in a fault state or not. Error can take many forms, whether it is the response of the system due to a failed sensor or a correct response to a sensor that results in the inability of the system to achieve its higher level goals. These can result in “trapped” behavior. To demonstrate this inability to detect error due to the lack of interconnection, a *Mathematica* program that simulates the motion of a robot was written. Rug Warrior was the subsumption model for this simulation. There are three distance sensors and three bumpers. The robot is programmed to Escape if the bumpers are activated. To Avoid if the bumpers are not activated and distance sensors are activated. Finally, to Wander if neither the bumpers or the distance sensors are activated. The program uses a body-centered coordinate system for the robot. If the sensors become defective, or just consistently give the same input, what happens to the robot? For subsumption, the robot

will become “trapped”. This is clearly demonstrated by the following “path” diagram. The robot will be trapped in one of six paths.

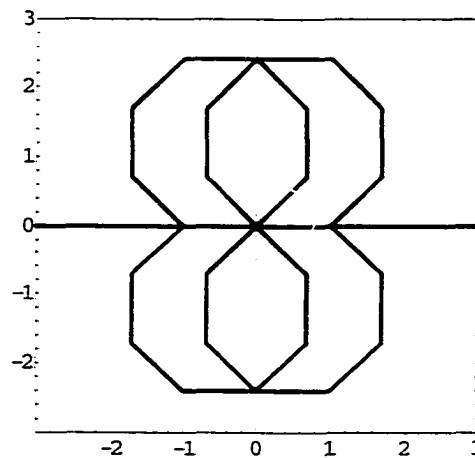


Figure 3.2, Trapped Trajectories

Figure 3.2 demonstrates that if for any reason the sensors become fixed, or cease to function, the robot will enter into stable trapped trajectories. There is no way in subsumption for the robot to detect that it is in error and it will continue to execute the same routine, thereby trapping itself. The six paths shown are: continue straight-ahead or straight back, four loops (two forward, two reverse), or to just remain at the origin. This is a very simple look at the types of errors that can occur.

3.2 AnSim Error Investigation

AnSim is a 2-D model of world that can be populated with animats. The code is written in C++ and will run on any desktop computer. The simulation allows for the testing and rapid prototyping of control architectures. For this error investigation, two simulations were run plus a baseline. The baseline simulation uses the subsumption architecture. The animat is equipped with two whiskers, three infrared sensors, eight

light detectors and four nugget detectors. The baseline is presented in Figure 3.3; it shows the animat navigating its world.

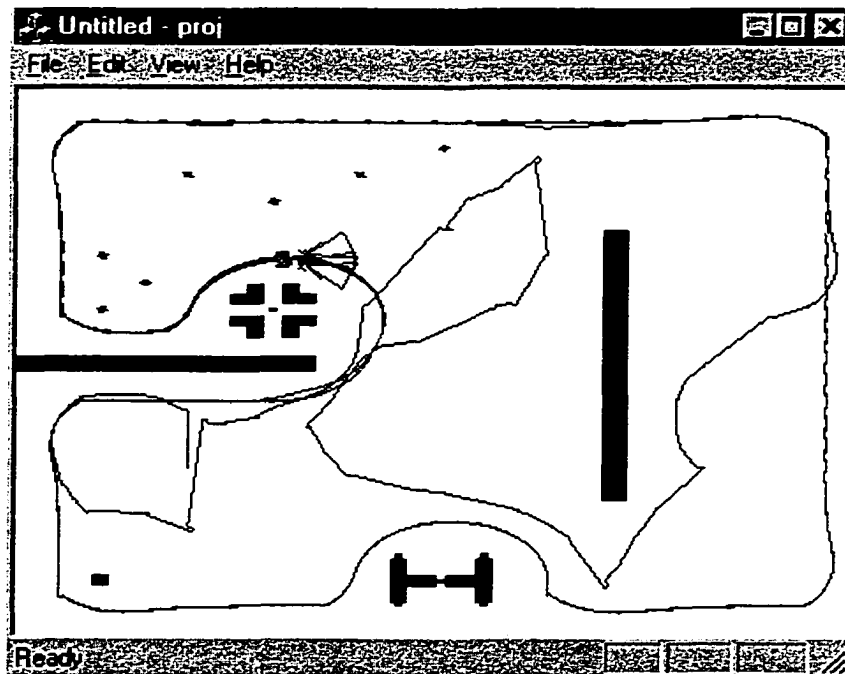


Figure 3.3, Animat Baseline

For the first error demonstration, the right whisker was fixed in the ON position. Using subsumption, or any current architecture, the robot will not be able to move out of this error condition. The reactions are hard-wired and the system is required to obey the correspondences it has been provided. This is shown in Figure 3.4.

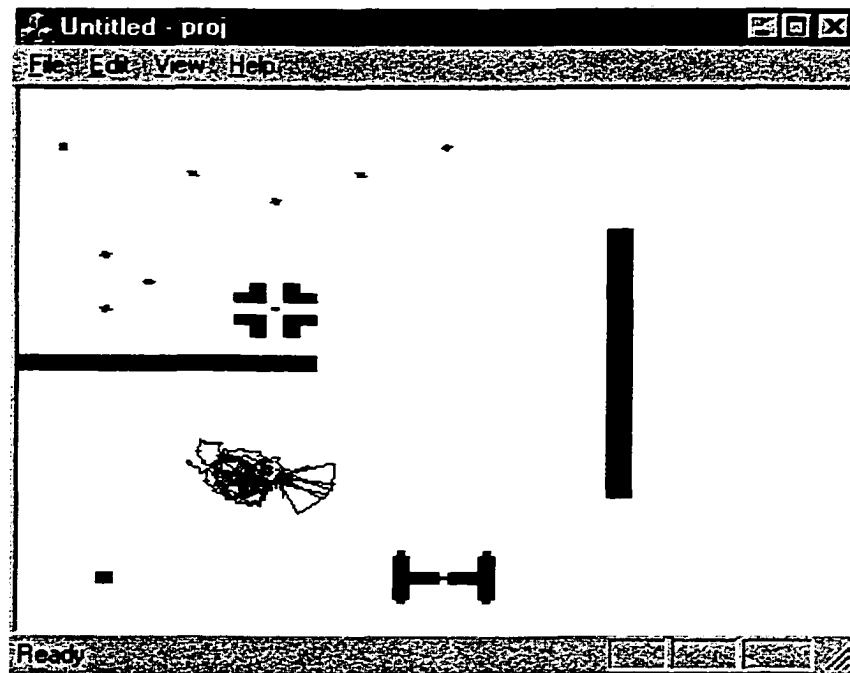


Figure 3.4, Whisker Error

The robot will basically remain in the same place until the maximum number of iterations is reached. The robot backs and turns until the maximum number of iterations is reached. The robot does not show the specific “trapped trajectories” that the *Mathematica* model showed. This is because the robot backs and turns a different amount each time. The best example of a “trapped trajectory” in AnSim is the case of fixing one of the infrared sensors in the ON position. This is demonstrated in Figure 3.5.

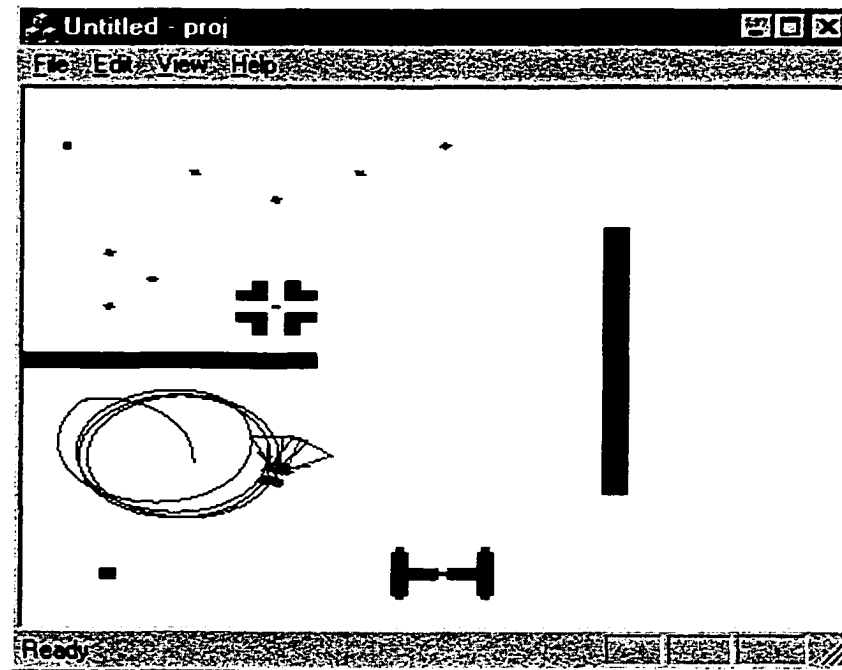


Figure 3.5, IR Error

This demonstration shows a “trapped trajectory”. The robot goes into an ellipse caused by the correspondence associated with an active right IR detector. It must veer left.

These are just examples of the kinds of errors that can be manifested in robots and how to describe them. Error will be investigated more deeply in the following sections. These examples demonstrate what could happen when error is ignored. A simple time-out would not fix this problem. The animat would move out of the trap for one iteration of the control loop and then right back into the same trap. There would be no perceptible change.

4 Interactive Representation

In the interest of developing a control architecture that is epistemologically sound as well as robust, extensible and fault tolerant, a relatively new theory of representation, Interactive Representation, is used. Interactive Representation has the claim of System Detectable Error.

Interactive Representation is a fundamentally different way of determining interaction between a system and its environment. It has already been shown that any system that attempts to explicitly represent its environment and all players in its environment is doomed to failure. There is no way for the programmer to anticipate all possible scenarios or all possible players, therefore, the system will be at a disadvantage, unable to reliably “do something appropriate”. Added to this is the inability of any current system to represent error. There is no way for any system, mechanical or living, to know if what it is representing is in fact what it thinks it is representing. Such a statement would require an outside observer. Take the typical form of error seen in modern robotics, the typical error is a location issue represented by:

$$\varepsilon = q_a - q$$

Where q_a is the robots actual location and q is where the robot is *supposed* to be. (Lin, 1995) The robot has no way of externally verifying its location, as far as the robot “knows” it is where it “thinks” it is. Therefore, any systems based on the current methods of representation are doomed to failure. There is no way to explicitly represent the environment and no way to detect error.

Interactive Representation will be described in three sections: Interactive Possibilities, Motivation and Error. These are the three main premises comprising Interactive Representation.

4.1 Interactive Possibilities

Interactive Representation does not attempt to explicitly represent the environment, per se; it only tries to represent interactive possibilities with the environment. The frog does not explicitly represent a fly as a food source. The frog only represents the presence of the fly as the possibility of eating. Interactive Representations only lead to the possibility of further interactions for the system.

Buzzing Fly □ Tongue Flicking □ Eating

There is never an explicit representation of the fly, as a fly. It is not possible for a frog to possess this capability. Interactive potentiality is defined by function, not by tokens picked up from the environment. Pointers are then required to determine which subsystems and which interactions are available. (Brickhard, 1999a) Subsumption and other behavior-based architectures already have the necessary pointers. Each reaction is active and points to the next reaction for each iteration of the control loop. The desire for each behavior is to accomplish its base reaction. The highest level reactions keep the system out of trouble. The indication of interaction outcomes will come from these pointers. These pointers represent internal states for the system. Internal is key because there is no need for the external observer. “The error or lack of error of such and outcome indication is constituted by the system either entering that internal function state or not entering it.” (Brickhard, 1999a) There is no representation of the environment.

No circularity is developed in the argument. There is no comparison of what one sensor thinks it is representing with what another sensor thinks it is representing.

4.2 Motivation

Very simple systems do not need a method to determine which interactive possibility to pursue. A simple reactive system will suffice. This is the case of systems that only possess a few Behaviors. The system then must only choose one or the other. As the complexity of the system grows, there will be many different interactive possibilities to pursue. A frog may see a fly and at the same time see the shadow of a hawk. Which interaction should the frog choose? Hide from the hawk? Flick the tongue to possibly eat? Motivation is what will make a system choose one activity over another activity. “Anticipation of what’s possible – representation – serves the function of selecting what among those possibilities to select next – motivation.” (Brickhard, 1999b) Added to this is the need for Positive and Negative Emotions. Positive Emotions are uncertainty situations where there is a strong possibility of resolving. A negative emotion is one that may cause an uncertainty loop. The anticipated outcome from the uncertainty is uncertainty, etc. These situations will be avoided. (Brickhard, 1999b)

Somatic markers are special instances of body feelings (visceral and non-visceral sensations) generated by emotions, that are acquired by experience based on internal preference systems and external events and which help to predict future outcomes of certain scenarios. They will force attention on the negative or positive outcome of certain options, that can be immediately defeated leaving fewer alternatives or can be immediately followed. (Gadanh, 1998)

Therefore, the robotic system will determine which behavior/reaction to use based upon which behavior/reaction will have a positive emotion for resolution of the problem. Those behaviors/reactions that will cause a negative emotion will be inhibited/avoided.

For minimalist systems there is no need to actually calculate motivation. The systems are very simple and their behaviors will have a definite order of execution. One example is a simple autonomous robotic system that has three main behaviors. The system Searches for food nuggets, the system takes nuggets to Home, and the system seeks Charge when it has a low charge level. For this system, with three behaviors, there is no need for motivation. If the system is running low on charge, it must seek a charge station to recharge or die. There is no need to seek home if the system does not possess a nugget of food. If it does possess a nugget of food, there is no need to look for more, since it can only carry one nugget at a time. Motivation is not required for minimalist applications.

4.3 Interactive Error

To properly define and address system detectable error, a good definition of *system* is required. There are two basic assumptions critical to the defining of a system.

1. A system must be an entity that can be separated from the rest of the universe with some form of boundary.
2. A system is composed of interactive parts. (Karnopp, 1975)

This understanding is critical to being able to understand system detectable error. Most researchers add something to their system to detect and recover from error. (Ferrel, 1993, de Benito, 1990a, O'Connor, 1993, Mosterman, 1997, Lockner, 1990, Baker, 1999, Gujar, 1994) No reference was discovered that claims to have the ability to possess system detectable error without the addition of another processor or another system. By

adding a system to their basic system, they violate the definition of system detectable error and have produced a circular argument. These methods are successful, but they do not possess system detectable error and they increase complexity and reduce speed. For example, the Government requires BITE on all of their equipment. By saying “on”, BITE is not part of the system. An external observer is being used to detect and recover from error.

Since Interactive Representation only represents the possibility for further interactions, it is possible for the system to self detect error. (Brickhard, 1999a) Error in this case, simply stated, is the failure to anticipate future interactions. (Brickhard, 1999d) If the system is not creating the possibility for future interactions, it is in error. ‘Creating the possibility’ may be as simple as not ‘wandering’ or not ‘eating’. The indication that the system is not anticipating or creating future interactive potentialities will be shown by the pointers not getting to the base reaction, not entering the internal function states. The system becomes “trapped” in one of the higher level reactions. These indications of not creating future interactions will be seen as patterns that will emerge from interactions with the environment. These patterns that develop can be manifest as shown in previous sections, i.e. Figures 3.4 and 3.5. Nowhere in this discussion was the mention of representational error. There is no mention of sensor fault. These patterns that develop are indications of functional error that can be detected by the system.

5 Error

Error is a very hard issue to describe and even more difficult to predict. Other researchers do not downplay the importance of error detection. “A system must be able to realize when it cannot achieve automated task objectives.” (Hamel, 1999) Hamel goes on to say that this is not possible. This research will enable a system to realize when it cannot achieve automated tasks. Some robotocists describe error in terms of position, (Lin, 1995) while others have described error in terms of misinformation coming from the robotic sensors, i.e. distortion in size and shape of object, specular reflections or cross talk between sensors. (Baker, 1999, Beckerman, 1990, Ferrel, 1993, Kortenkamp, 1998, Nehmzow, 2000) Still other researchers try to minimize error by trying to predict an uncertainty bound for their sensors. (Lee, 1999) In most current systems, error is only detectable from the perspective of some observer, whether the observer is a human or an additional system. (Brickhard, 1999c) All of these researchers have one thing in common to their error detection technique. They are based upon explicit sensor error using an external observer.

What is an error? What constitutes an error? How is an error detected? These are all critical questions that must be answered in the context of autonomous robotics. Error is described as a failure to detect a feature in the environment, or the failure of a sensor, or the failure of an actuator. (Baker, 1999, Ferrel, 1993, Lockner, 1990, Mosterman, 1997, Pin, 1999, Umeda, 1991) All solutions to error detection and correction of the sensory data are based upon sensor fusion. Comparing the data from

two or more sensors to “eliminate” the erroneous data. This leads to a circular argument. The system does not have the ability to rule out one sensor, in favor of another sensor, without the use of some outside observer. Does the addition or omission of a feature that is not present result in failure? Does a sensor failure result in a system failure? Does an actuator failure result in a system failure? These failures should only constitute an error for a system if the failures result in the inability of the system to perform its designated function. The main goal of any system is to stay viable, achievement of other goals are secondary. (Atkins, 1997) This is the premise behind robotics: Performing Real Tasks in the Real World.

5.1 Function

There is a great deal of work being performed in the literature on failure diagnosis and correction, but the methods are for design strategies and equipment failures. This research merges with contemporary robotics to come up with a system failure detection scheme based upon function, not upon sensor data.

To begin, there must be a definition of function. Yoshikawa in his General Design Theory describes the definition of function as “a set of a sub-class of abstract concepts.” (Chakrabarti, 1992) The “*visible function* as a behavior exhibited under a given circumstance; the total of the behaviors the design could potentially exhibit under various possible circumstances are called it *latent functions*. Problem-to-solution transition in real-world situations is seen as a sequence of patching operations on a promising provisional solutions, until all of the specification is met.” (Chakrabarti, 1992) Chakrabarti is talking about a design strategy for any system. This definition works very well with the work performed in this research. Simply replace “behavior” with

“reaction” and the *visible functions* become the Reactions and the *latent functions* become the Behaviors. Visible functions are the reactions exhibited under a given circumstance; the total of the reactions the design could potentially exhibit under various possible circumstances are called the latent functions – behaviors. The solution to problems is also very similar to this work. Solutions will be tried until the specification is met. Detecting the error will determine when the specification is not being met and then “some form of search” (Chakrabarti, 1992) through the reaction space will be required. Understanding that *function* is the critical point of interest is new to the fault tolerant control methodologies for autonomous robotics. Function is what the designer will have to look at to determine error. “Living systems, as physical autopoietic machines, are purposeless machines.” (Maturana, 1980) Function, goals, purposes, and aims are all things that can only be described by an observer. They have no meaning to the system.

5.2 Definitions and Theory

Most researchers have attempted to explicitly define error, failure and fault. They have defined error as improper operation caused by an undesired state within the system, and the instance which caused the error, whether known or hypothesized, is what constitutes a fault. Improper operation, as noted in the definition above, can be considered a failure of the system in that the system is not longer able to perform its desired function. (Lockner, 1990) A more precise definition of failure would be an event which instigates a change from an in-commission operational state to an out-of-commission operational state, while an event which simply impairs or degrades the ability of the system to accomplish its desired function is considered a degradation.

(Landers, 1963, Pau, 1981) If a researcher can predict or induce a consistent manner in which a failure, omission of an expected action/occurrence, or non-performance of a task will occur, this prediction is known as a failure mode. (Pau, 1981) In many cases, if a failure mode is known, the system can be designed so that it contains certain provisions to avoid failures after a fault has occurred. This is known as a fault tolerant system. A fault tolerant system utilizes the concept of coverage, which is a systems ability to avoid failure after the occurrence of a fault(s); however, a system can never have 100% coverage. (Lockner, 1990) To expound on the above definitions, any system containing sensors can have a sensor failure/fault such that the signal detected is systematically different from the actual value. There are four main types of sensor faults: 1) Disconnection; 2) Sticking; 3) Bias; and 4) Increased measurement noise. Disconnection and Sticking are the most common sensor faults. (deBenito, 1990b) Bias is often the result of improper installation or not zeroing the sensor prior to putting the sensor into service. Increased measurement noise is caused by drift in the sensor.

Redundancy is the key element of all fault tolerant designs. (Lockner, 1990) The problem with redundancy is that all systems that totally rely on redundancy must have some sort of voting system to determine which redundant system is operational and which redundant system is not operational. (Ferrel, 1993, Kortenkamp, 1998, vonWichert, 1997) This is accomplished in all systems discussed by an external observer. These external observers are given titles, such as, “observer mechanism”, “executive”, “extra software” or “diagnostic system”. (Mosterman, 1997, Kortenkamp, 1998, Lockner, 1990, Pau, 1981) All of these statements preclude the possibility of system detectable error, because they all require a diagnostic system that is separate from

the system being monitored. “The (external) observer beholds simultaneously the entity that he considers and the universe in which it lies.” (Maturana, 1980) Therefore, the external observer is also a system; a system that possesses more information than the system that it observes.

The above definitions can be applied to an examination of the subsumption architecture. One of the claims of subsumption is that the failure of one layer has only a minor influence on the performance of the entire system. (Nehmzow, 2000) The problem here is the definition of *failure*. As was shown in previous sections, if one of the sensors were to fail in the current behavior architectures, the submachine associated with the sensor has not *failed*. It still functions according to its program, but the system has *failed* because the system no longer provides its desired service. Here is where this work will diverge from current behavior architectures, by looking at system failure and not at sensor failure or submachine failure. The subsumption model has zero percent coverage for a sticking sensor. If any one of the sensors is stuck in the “ON” position, the system will fail, it becomes “trapped”. This is clearly demonstrated in Figures 3.4 and 3.5. This is because the submachines of subsumption are independent and are not connected. The system is required to respond to the input string it is provided. The correspondence exists according to Interactive Representation. The constructivists also attest to this statement. “Because the uniform transmission of neural information (in the form of spikes and spreading activations), no difference can be found between the interactions within the system and “peripheral” influences.” (Stary, 1995) The “peripheral” influences are the environment. Once the sensor reading is made, the internal mechanisms cannot distinguish between a true reading and an errant reading. The system will respond to an

errant reading as if it were a true reading. *Something* internal to the system is required so that it can determine if the sensor readings will cause failure. If a sensor is stuck in the “OFF” position or disconnected, the built-in functional redundancy should allow the system to continue its expected service. There is no way for the system to know that the sensor input is incorrect. There is no valid way for the system to ignore the input, based on the input alone.

There are two main classes of faultfinding mechanisms. The first is *Symptomatic*. “In symptomatic strategies possible failures corresponding with the symptoms are accessed via a search through a library of abnormal system state patterns.” (Toms, 1989) This method cannot handle “previously unencountered fault-symptom combinations.” (Toms, 1989) The designer is forced to determine all possible fault-symptom combinations and devise a plan for dealing with these faults. The other class is *Topological*. “Topological methods characterize behavior relations as directed graphs constructed from system models under normal operating conditions and faulty conditions.” (Mosterman, 1997)

5.3 Current Error Detection Methodologies

Current research deals with detecting sensor failure. This is done by the comparison of a sensor output with other redundant sensor output or by comparison of a sensor output to a known input. (Baker, 1999, Beulah, 1997, deBenito, 1990a, Ferrel, 1993, Godambe, 1998, Gujar, 1994, Kortenkamp, 1998, Lockner, 1990, Mosterman, 1997, O’Connor, 1993, vonWichert, 1997) These solutions are often referred to as BIT (Built-In-Test) and BITE (Built-In-Test-Equipment). These additional systems increase complexity and can actually reduce the reliability. (Ruff, 1993) The BITE is added to

the system to monitor the sensors to determine when and if a sensor failure occurs. Even though the BITE is resident with the system, it is not part of the system, rather it is an external observer. It compares the sensor output to a preprogrammed pattern of possible outputs. If the outputs do not match, it has a preprogrammed plan to “fix” the system. The combination of the base system and the BITE forms an aggregate system that could be self-referring, but again, the BITE must rely on what it perceives from the base system and BITE’s universe. BITE is required to compare like sensors and by using an arbitration scheme, activate one and deactivate the others to keep the base system functional. BITE has more information than the base system. It must then make a choice, but there is no way for a system to know that what one sensor is representing is correct while what the other sensor is representing is incorrect. There is no system, animal or mechanical, that can determine whether what it thinks it is representing is actually what it is representing. (Brickhard, 1999c) To do this, BITE would also need an external observer and that external observer would need an external observer . . .

One example of this built-in ability is the ECM (Engine Control Module) in late model automobiles. For electronic fuel injection to work properly, the ECM requires a great deal of sensory data: throttle position, mass air flow, coolant temperature and exhaust gas oxygen content to name a few. If one of these sensors fails, the ECM cannot correctly determine air/fuel ratio for proper operation. The ECM then defaults to a “safe mode”. Performance is greatly reduced but the automobile will still run. This is the same method used by Ferrel and Baker. In order to resolve a sensor fault, the system must be able to detect the presence of the fault and have a method to correct the fault. (Baker, 1999) The problem is that these measures require an external observer, BITE, to

determine that an error most likely exists and then to perform the “fix”. Therefore, these systems do not possess system detectable error.

Still other researchers steadfastly cling to the notion of generating plans. “Complex plans are required to safely operate an autonomous system in a dynamic environment.” (Atkins, 1997) Atkins goes on to state; the only way to avoid failure is by preplanned responses and time-bounded planning. This is definitely not a plausible solution.

5.3.1 Examples of Fault Detection

Ferrel (1993) compared multiple redundant sensors to determine which sensor was in a fault state. If the output from a sensor is plausible, when compared to the output of the other sensors, then the sensor is deemed to be functional. If the output is not plausible, then the sensor is deemed not functional. Once deemed not functional the sensor is ignored and the other sensors drive the behavior of the robot. Redundancy is desirable, but the reliance on redundancy does not necessarily constitute self-detected error. Redundancy basically relies again on the external observer. Ferrel’s example for how a system operates clearly shows the use of an external observer.

To illustrate this idea more clearly, imagine the following: you wake up in the morning and you want to know what the temperature is outside. You look at the thermometer outside your window and it reads 90 degrees Fahrenheit. However, you feel the window and find it’s ice cold. The temperature sensors in your hand complement the thermometer reading. (Ferrel, 1993)

In this example, the external observer, “you”, determines that the thermometer is incorrect and ignores its output, the thermometer is not capable of determining that it is in a fault state. The comparison of sensors on her robot was accomplished by using an additional microprocessor. The robot used by Ferrel had seventeen microprocessors.

Maybe the sensor is in a fault state maybe it is not in a fault state. Ferrel admits it is possible to incorrectly classify a sensor as broken. This again is not a plausible solution.

There is also the use of Direct Indicators. Direct Indicators are things like wear bars built into tire treads. These wear bars become exposed when the tread is 3/32" in thickness. An external observer, the owner, is required to notice the wear bars and know what they mean. Some disc brakes come with indicators that make a squealing noise when the pads become thin. Again an external observer is required to hear the sound and know what it means. Direct indicators require a knowledgeable diagnostician that knows *a priori* what the direct indicators mean. (Ruff, 1993) Direct Indicators are not error proof and they require the external observer. First, the diagnostician must correctly identify the direct indicator, then determine if the direct indicator is correct, and then perform the preventive maintenance or repair. None of these solutions allow for system detectable error. There is no way for the system to "know" whether the direct indicator is correct or incorrect. This relates back to the correspondence issue. If the correspondence exists, then the representation exists and it is correct. The system is forced to rely on the direct indicator, an external observer to the system, and then to execute a preplanned "fix".

Plug-In test equipment is available for many different types of systems. The automotive industry is one of the main contributors to this area. A service technician can directly plug his diagnostic equipment into a port on the automobile, as well as additional sensors. This equipment will then run diagnostics on all elements of the automobile. Most of the time, this diagnostic equipment will possess much more information than the ECM on the automobile, allowing the equipment to have a better understanding of why

the automobile is not functioning properly. This equipment also has the added advantage of a well-trained technician.

Baker (1999) generates a “confidence map” for his sensors leading to edge detection for a visual system. He is not looking for a functional error; he is looking for a specific sensor fault. The “confidence map” allows for edge detection by merging multiple sensors together to determine the probability of a feature in the environment. With a high enough probability, Baker assumes that edge exists. This allows for the generation of a map for planning purposes. For the visual based systems, the problems the authors allude to all involve the detection of edges that are not present, the false positive. (Baker, 1999, Pin, 1999) This is similar to the case of the case of a stuck sensor.

Stein (1995) combined subsumption with what is basically a planner, but in this case the planner was designed to mimic imagination. The planner would be given a map of its world and then would “imagine” how the sensors would respond on its journey to a designated point. It would then set out on the journey and compare the “imagined” path to the actual path. If there was a discrepancy, the world was assumed to have changed and the actual map would update the “imagined” map. By continually redeciding what to do the robot will avoid major errors. (Stein, 1995) Stein is still using the imagined path as a comparator, back to the external observer, and since her default is to the interactive control module, there is no way for the system to get out of the faults presented earlier. The correspondence exists; the system will obey its preplanned responses, trapping Toto.

5.4 Behavior Based Error Detection

This work develops a behavior-based approach to error detection and recovery that avoids the need for an external observer. The system can not know that one of its sensors has failed. If the sensor produces input, the system must respond to the input since there is no way for the system to know whether that input is correct or incorrect. A system is aware of what it is doing and this awareness could prevent the system from becoming trapped in one of its reactions.

This is a switch from the detection of a sensory fault to the detection of a behavior error. There is no way to know that if what the system is detecting is correct or incorrect, but there is a way to determine that the system is not performing its designated task or function. There is a way to know that the pointers are not moving from one Reaction to the next Reaction. The interactive possibilities are not occurring; the system is in error, not the sensor is in a false state.

Robotic systems can not self detect error because these systems are based upon correspondences. "If the correspondence exists, then the representation exists and it is correct, while, if the correspondence does not exist, then the representation does not exist, and so cannot be incorrect." (Bickhard, 1998) A more useful behavior definition comes from Bickhard (1999d) who simply states that an error is the failure to anticipate/predict future interactions. "The circularity of their organization continuously brings them back to the same internal state. Each internal state requires that certain conditions (interaction with the environment) be satisfied in order to proceed to the next state." (Maturana, 1980) The system will have certain goals to achieve, not achieving these goals, not returning to the base internal state, will cause the destruction of the system.

There are three major elements in the prediction of error: 1. The nature of the task and the environmental consequences of the task; 2. The mechanisms governing performance; and 3. The nature of the system. (Reason, 1990) These three elements must be addressed to develop an adequate theory of error, one that allows for the prediction of the conditions under which an error can occur and the form the error can take. (Reason, 1990) An error occurs when anticipated interactions are not occurring, (see Section 4.3), now all that is required is a method to determine what form that error will take. Beckerman (1990) is proceeding in this direction by looking for predictable patterns of conflict between sensors. In this case, the patterns arise from unknown position coordinates due to interference from objects in the robot's line of sight. The system is capable of "knowing" what it is doing. This knowledge can lead to the determination that it is not functioning properly. The system "knows" what will cause it to become "trapped", it can then move away from what is trapping it and complete its function.

The system will not specifically represent the environment, only whether the environment is providing the necessary conditions for further interaction. Errors then occur when the interactive possibilities are not being fulfilled or the interactive possibilities are not occurring. For very simple cases, this is not a difficult concept. If the main goal of the robot is to explore the environment, an error will occur when the robot is not exploring. These errors will be represented by patterns as seen in Figures 3.4 and 3.5. This is not a new concept. Pin describes the avoidance of what he calls limit cycles. In the purest definition of the term, these "trapped" trajectories are not limit cycles. In conversations with Dr. Gerhard Danglemayr (1999) of the Colorado State

University Department of Mathematics, it was resolved to call these trajectories “attractors”. The idea for error detection or limit cycle detection, however, is the same. Pin (1999) had the robot identify limit cycles by identifying that it was in approximately the same position and the same direction as it was at a previous time. The difference is that Pin does not call these errors, only cycles to be discerned and avoided. In fact, the example he shows is a case where the robot is trying to achieve a location, but a wall is in the way. The robot then tracks back and forth along the wall trying to reach its goal. A similar situation is displayed for the simulated animat in Figure 9.14. There are no sensory errors for the robot, only that it is not reaching its goal, interactive possibilities are not being met. The system is not returning to the base internal state. If the correspondence traps the system in an internal state, then the system has lost its integrity: the system has functionally failed. These attractor trajectories are also displayed in Nehmzow (2000).

The system will have to possess memory to know that it has been in one of these patterns and that it is still in the pattern. Maturana (1980) claims that living systems are historical systems, requiring memory to function. The systems designed here are not living systems, but if living systems require history to function, then it is logical to assume that all systems require history to function. Memory can serve one of three purposes: “1. It can send commands to the actuators. 2. It can modify the data stored in memory or 3. It can modify the processed sensor data that serves as input to the basic reactive system.” (Pin, 1999) Once error is detected, the interconnections between behaviors and reactions will allow the system to attempt to extricate itself by implementing Option Three. There is never a plan developed, by the programmer or by

the system, for how to extricate it from whatever is trapping it. The system is forced to explore its reaction space until it becomes free. Below are biologic examples that could be representative of systems that use the Interactive Representation to avoid behavior error.

5.5 The Medicinal Leech

In 1988 Charles Lent and Michael Dickinson published their results on the study of *hirudo medicinalis*, the medicinal leech. They chose this segmented worm because it has very few neurons and the neurons are relatively large. The leech basically possesses two behaviors, Hunger and Satiation.

Lent and Dickinson discovered that the leeches feeding behavior was dependent on the state of distension of the body wall. If the leech is full, it will not feed. If the leech is not full, “ingestion is stereotyped and compulsive: strong stimuli, such as forceful pulling or even cutting into the body, do not deter this tenacious worm once it begins to feed.” (Lent, 1988) These studies showed that there is a direct link between the leech’s behaviors and the motivation/inhibition of serotonin neurons. A stimulus that Motivates feeding excites serotonin neurons to fire. A stimulus that terminates feeding inhibits serotonin neurons from firing. (Lent, 1988) Let’s take a closer look at what the leech’s feeding behaviors are and how they are stimulated. Coupled to Hunger are several reactions: Rest at the edge of a pond, Orient toward a ripple in the water, Swim toward wave source, Explore object, Bite a warm region, If blood is drawn, feed, If blood is not drawn continue the search. There are two reactions associated with Satiation: Seek deep water and Hide. The leech does not swim towards objects and will not feed if the body wall is distended. The leech’s behavior switches between these two behaviors,

Satiation and Hunger, based on distension of the body wall. (Lent, 1988) The behavior network for the leech could look something like this:

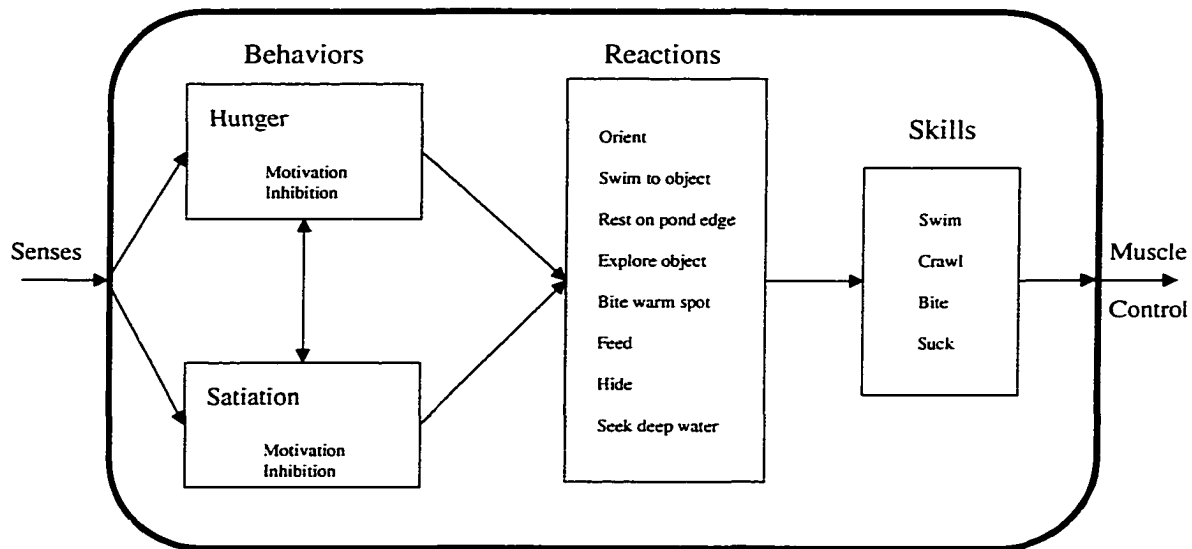


Figure 5.1, Leech Behavior

The leech does not specifically represent its environment. When hunger is motivated, the leech does not “look” for a dog, a human, a cow, etc. to feed upon. The leech uses an “interactive representation”. “Interactive representation is not built up out of particulars such as sense data, but as a process of differentiating the environment in ways that are relevant to further interactive possibilities for the system. Differentiation’s are intrinsically open and under specify what they differentiate.” (Brickhard, 1999c) The leech swims toward a disturbance (not a cow), explores the disturbance (not a leg), if the disturbance is warm enough the leech will bite, if blood flows the leech will feed. Never does the leech represent the animal it wants to feed upon; in fact, there is never a representation of food source at all. Only, the conditions for feeding may be being met, the leech is differentiating what the environment provides in such a way to produce the necessary conditions for further interactions in the hope of behavior satisfaction. The

leech bases all of its behaviors on body distension. If the body is not distended, the leech will try to feed. If the body is distended, the leech will not feed. There is no specific hierarchy for application of reactions the leech could use to achieve distension. All of the reactions are there and ready to be executed if the environment presents the necessary conditions. They differentiate what the environment provides in order to achieve global goal satisfaction. The leech desires to have its body distended, it does not desire to swim or desire to bite unless the environment provides the leech with the possibility of wall distension if it swims or if it bites. This kind of differentiation is what I have built into a robotic system.

The leech is also equipped with its own error detecting equipment. The leech's most interesting behavior is Hunger. The leech is motivated for the positive interactive possibility of wall distension. All of its reactions try to achieve this state in a circular fashion. It swims toward what it hopes is something that may allow for the possibility of wall distension. If it does not strike anything while swimming, indicating no interactive possibility and therefore in error so it will return to the edge of the pond and wait for the next ripple. If it does strike something, then it explores for a warm spot. If no warm spot, indicating no interactive possibility, therefore in error so it will return to the edge of the pond and wait for the next ripple. If while exploring it finds a warm spot, then it will bite. If it fails to draw blood, indicating no interactive possibility and therefore in error, it may then continue to explore or it may return to the edge of the pond and wait for the next ripple. (Lent, 1988) Each error was caused by not reaching wall distension. The circularity of the organization requires that it follow the progression of its reactions until wall distension is achieved. If while following a reaction the required circularity does not

bring it to wall distension, the leech “knows” it is in error and then the entire progression is allowed to begin again.

The leech responds very well to its environment, but the leech can be “tricked” when removed from its natural environment. This was shown by how Lent and Dickinson would cause wall distension by filling the crop with a saline solution, the leech would not feed when full, whether with saline or with blood. They would also empty the crop of the leech while feeding (effectively simulating a lack of wall distension) and the feeding behavior would go on for hours. (Lent, 1988) This shows that there must be a tight coupling between the system and the environment. Living systems cannot be understood when removed from the environment with which they are designed to interact. (Maturana, 1980)

5.6 The Human Model

The human model, like the leech, is also capable of being fooled. In the following example, the human model detects that there is an error, and moves away from the error in the same method proposed for a simple robotic system.

All living creatures have some method of keeping their bodies aligned with gravity. Some are very complicated, for example the human, using many sensors in a redundant fashion to accomplish posture control.

Imagine the following situation: You are standing upright in front of a striped display that occupies the greater portion of your visual field. The stripes now begin to move at a steady velocity in the downward direction while your eyes remain fixed on a target light in the center of the display. After a few seconds, you will begin to lean forward quite involuntarily, only maintaining your balance (if at all) by fairly strenuous work on the part of your ankle and leg muscles. After about a minute, the ‘force’ that caused your body to lean forward is removed. However, instead of drifting back to the upright position, you will now find yourself

leaning backwards – to such an extent that you may be forced to hold on to something solid to avoid falling over. But, if instead of the visual motion being stopped, you simply close your eyes, then there is no tilt-back effect. Your body merely returns to the upright position and stays there. (Reason, 1990)

The input from the eyes is not in error, but it is causing a functional error for the postural system. The input from the eyes is the dominant input for the postural system. Therefore, this input is ignored (closing the eyes) in an attempt to let the other functionally redundant systems, the ears, take over the control the body.

5.7 Fault Detection

Fault detection is based upon the failure of the system to perform its designated task as viewed by an observer, (i.e. in an error state). Function and task are only relevant to the observer and belong to the domain of his description. (Maturana, 1980) The system itself is not aware of function or of task, only of maintaining the circularity of its organization. The Reactions in a behavior-based robot all depend upon the sensor input. Transition diagrams as described in the Interactive Representation Architecture section can describe these reactions. To determine what constitutes a failure, the designer will perform basically a Failure Modes Effects Analysis, FMEA, for the robotic system. This will allow the designer to know what will cause the system to become “trapped” or what will cause the system to be incapable of performing its designated task. “A self-controlling learning machine needs task-specific definitions of *errant behavior*.” (Khan, 1995) The system developed here is not quite a learning machine, but it is self-controlling and if it does to possess system detectable error, the system must “know” what constitutes *errant behavior*. There are two main ways of performing this analysis:

1) the designer can use a detailed transition matrix for the system or 2) causing the failure of each sensor and observing the resultant behavior.

5.7.1 Transition Matrix

The creation of a transition matrix will allow the designer to be able to fully comprehend the behavior of the system for all possible input strings. (Gujar, 1994) The development process for the transition matrix allows for the determination of what will “trap” the system and what the “trap” will resemble. The development of the transition matrix will also serve the purpose of fully investigating the input and the output space for the system. This will prevent the possibility of an oversight, failure to program for a specific input string, which will also trap the system. If the system has no output for a given input string, the system will not respond to the input string.

Instead of attempting to explain the methods behind the development of a transition matrix, an example will be used. For a detailed description of how to create a transition matrix, see Gill (1962). To demonstrate this method, a very simple autonomous robotic system was developed. The robot is equipped with three infrared sensors, two whiskers and a random number generator. The system has only one behavior, Explore. The system has five reactions, Avoid Bumping, Avoid Obstacles, Seek Nugget, Follow Wall and Search. This system would use the following subsumption flow chart.

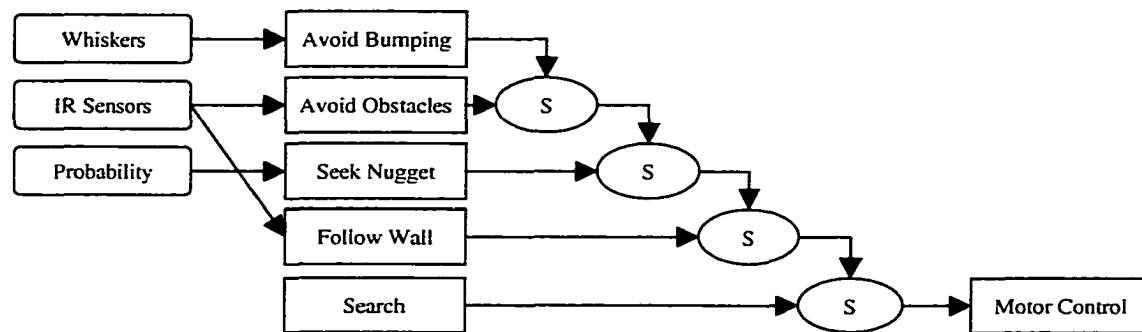


Figure 5.2, Explore Flow Chart

This and all behavior-based systems rely on functional redundancy. The bottoms-up approach emphasizes this point, although it is rarely stated. Avoid Bumping is the highest level reaction, but if Avoid Obstacles works perfectly, then Avoid Bumping would never be activated. This is the most desirable form of redundancy, because it prevents the loss of functionality caused by a conflict with the environment. What causes one type of sensor to fail should not cause the other type of sensor to fail. (Landers, 1963) The system for this demonstration has three sensor arrays, for a total of six possible inputs. The system has thirty-five possible input strings, $\text{Sensors}^2 - 1$, but the subsumption architecture simplifies the possible inputs to only twelve input strings that need to be investigated. Avoid Bumping is the highest level reaction, so there are three input strings for the whiskers to investigate, all other inputs, infrared and a probability will have no effect on the system when the whiskers are active. For Avoid Obstacles there are seven input strings that need to be investigated, again, there is no need to investigate the probability input as it will have no effect. Finally, there is the random number generator, two possible inputs, greater than or less than some predetermined number.

String	RW	LW	RIR	MIR	LIR	RN	Output
1	1	1	X	X	X	X	MB-RnT
2	1	0	X	X	X	X	TL
3	0	1	X	X	X	X	TR
4	0	0	1	1	1	X	RT
5	0	0	1	1	0	X	TL
6	0	0	1	0	1	X	RnT
7	0	0	1	0	0	X	VL
8	0	0	0	1	1	X	TR
9	0	0	0	1	0	X	RnT
10	0	0	0	0	1	X	VR
11	0	0	0	0	0	1	SE
12	0	0	0	0	0	0	FW-SE

Table 5.1, Transition Matrix

This transition matrix gives all of the possible outputs for the possible inputs. “X” means that it does not matter what that value takes, the output will be the same. “RnT” means random turn. For String Twelve, if Follow Wall was active, it is still active. If not, then Search will be active. The other outputs are as described in previous sections.

Redundancy is used to avoid the case of the false negative. That is the purpose behind having infrared sensors and having whiskers. If the infrared sensors miss an object, the whiskers will “find” the object and avoid the object. The case of the false positive is where the IRA will come into play. By looking at all possible input strings and the resulting output, the designer can determine what a “trapped” trajectory will resemble. These “trapped” trajectories can be self detected and then the system may move away from them, attempting something else. The system will not determine that a sensor is in a fault state, only that it is currently trapped and the reaction it has chosen is not working.

The transition matrix will allow the designer to locate and determine the symptomatic causes of error. Of course this method will not catch all of the possibilities that the system may encounter. This method is required to insure that the designer has accounted for all known causes of error and that the designer has provided for all possible sensory input patterns. Failure to complete the transition matrix could result in the system coming across an input for which it has no output. If this occurs the system will do nothing.

5.7.2 Observation

The other method to determine what causes a system function error and how that error appears is to design the robot using whatever architecture is preferred. Then induce faults to determine the behavior. Some faults will not cause system failure. Some will cause system failure. This method was demonstrated in Section 3, where the animat sensors were deliberately turned on to observe the resulting behavior. Casti detailed the observation method by generating an input and then allowing the system to run a few thousand iterations to determine the behavior. For these finite state machines, FSM, that are actually simple input-output devices, the “trapped” trajectories become very evident. This is the same method proposed for fault isolation and detection by Godambe (1998) for phase-locked loops. This method is effective, but a combination of the transition matrix and the observation will allow the designer to fully explore the input output space for their system.

Observation is using the topological method of faultfinding. The system will be run in both normal operating conditions and faulty situations to examine the behavior of

the system. Once the behavior is decoded, the system can be taught how to deal with the situations it encounters.

5.7.3 Behavior Error Detection

A combination of both a Transition Matrix and Observation will be required in the design of an Interactive Representation Architecture based autonomous system. The transition matrix will insure that the known “trapped” trajectories will be exposed. In this case, there will not be a plan to handle the “trapped” trajectory. The system will detect that it is in a “trapped” trajectory based upon the sensor-input string. It will then ignore the sensor that is causing the system to be “trapped”. It will not declare the sensor to be a fault. Only that the sensor is causing a system failure, the loss of circularity of its organization, and the system needs to try something else. It will then rearrange its reactions to accomplish its function.

The transition matrix will not catch all possible input strings that will “trap” the system. There is no way to determine all possible strings. Observation using a simulator will then be required. The simulator will allow the designer to encounter more, dynamic input strings and possibly detect other input strings or patterns of input strings that will “trap” the system. It is not possible to determine all trapping input strings. This point needs emphasis. All systems can be “trapped”. The human model clearly demonstrated this point. Humans are considered the most capable system on this planet, yet the human model can be tricked, can be fooled, can be “trapped”. This is a search for a more robust system. One that uses redundancy to achieve its function, one that uses system detectable error to determine that it is not performing its function and then is able to move away

from what is trapping it to achieve system level goals through a bottoms-up approach to both the behaviors and the error detection.

The advantage of this method of error detection and error recovery is that this method does possess the quality of system detectable error. The system does have the ability to detect that it is trapped. It does know what will cause the “trapped” pattern and it can then rely on its functional redundancy to complete its tasks. The behavior error detection will also have better performance than the systems using an external observer. “Naturally any decision to switch to the next node is delayed by this voting procedure.” (vonWichert, 1997) Adding software to perform error detection and recovery often decreases system performance. (Lockner, 1990) This is due to the fact that the external observer is outside the system and must decide what to do. The external observer compares the sensors. Is there a failure? How to resolve the failure? All of these tasks take time away from performing the function of the system. For system detectable error, comparison of sensors is not allowed, so there is no significant delay. This idea is demonstrated in the following sections concerning the Interactive Representation Architecture through the use of a simulated animat.

5.8 Error Detection Conclusions

The discussion above gave many examples of error and fault detection. The following table summarizes the results. All methods are being applied to the same system.

Method	Observer	Sensor	World	Function
Human	External	Low	High	High
BITE	External	High	Med	Low
Plug-In	External	High	Med	Med
Dir-Ind	External	Low	Low	Low
Inter-Rep	Internal	Low	Low	High

Table 5.2, System Level of Awareness

The human is used as an external observer to diagnose many system faults. The human has a very limited knowledge of the readings from the sensors. The human, however, has a very high amount of knowledge of the world the system is in and the function the system is performing. The function of a machine is only describable by an observer. An example of this is the auto technician. A well-trained auto technician can diagnose an engine fault without using any of the engine sensors. This is often accomplished with simply a well-trained ear or a well-trained nose. Injector number three is not firing. The engine is running rich. BITE is also an external observer. It has a much greater knowledge of the sensors than the human does. It has more information about the world than the base system, which comes from additional sensors. It still has only limited knowledge of function. Plug-in test equipment has excellent knowledge of the sensors and medium knowledge of both the world and the function. This is because most plug-in equipment utilizes more sensors to give a better idea of the world and because it is used when there is a problem, plug-in equipment is usually supplied with information regarding function. Direct Indicators are very effective when coupled with another external observer to diagnose them and fix the problem. Interactive representation is the only one that is internal. It has excellent knowledge of the system sensors. It only knows the world as a result of the interactions between the system and the sensors. It does have excellent knowledge of the function. It is specifically attempting to maintain the

circularity of the organization. The circularity ensures function when the system and the environment are well matched.

6 Interactive Representation Architecture

The discussion in Section Four outlined Interactive Representation. Interactive Representation is based in epistemology and philosophy; it needs further development to be applied to autonomous robots. Interactive Representation is the highest level of abstraction, in this section will be the Interactive Representation Architecture, which will be followed by implementation.

This control architecture will try to mimic the work done by Lent and Dickinson on the *Hirudo Medicinalis*, the medicinal leech, combined with the work of Brickhard, Interactive Representation and the work of Maturana for a way to self detect error. Lent and Dickinson give a biological example of how motivation and inhibition may be applied to a control architecture and Brickhard gives Interactive Representation which gives a method of system detectable error, coupled with a non-explicit representation of the system-environment interface. “System” is used here because there is no requirement that the system be a robot. The system could be a leech or a computer program.

A schematic of a system very similar to the Behavior Network Architecture is given in Figure 6.1.

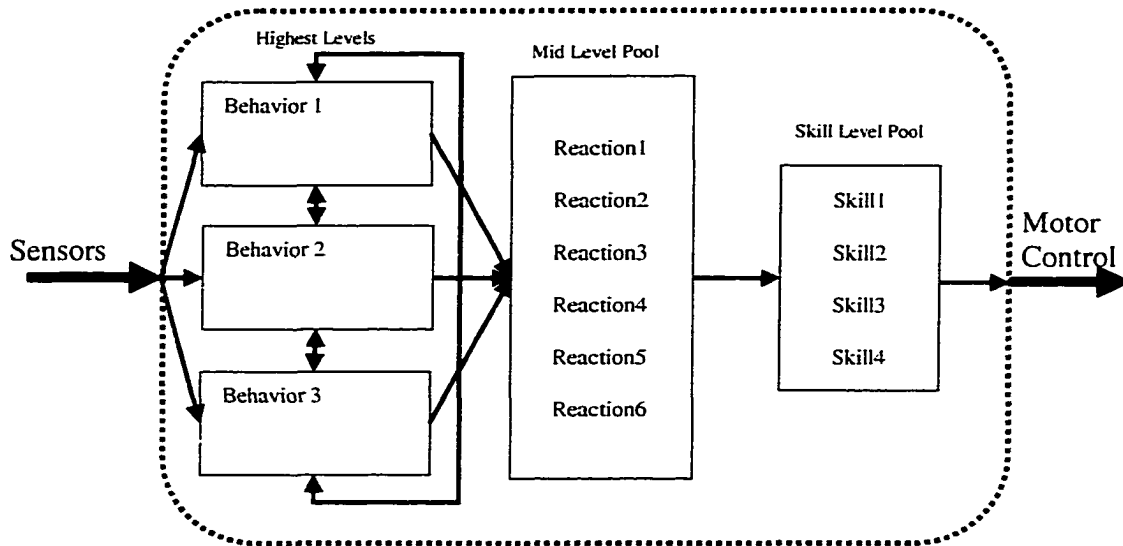


Figure 6.1, Interactive Representation Architecture

This modified BNA, Interactive Representation Architecture (IRA), possesses system detectable error. There is no need for Motivation and Inhibition because the systems being designed here are very simple and the tasks are very simple; therefore, a reactive system will suffice. All behaviors are active, and activation will be based upon the sensory data. There is not a linear order for selection of the active behavior. For the example given previously, the robot will search for a nugget if it does not have a nugget and its charge is not low. It will search for home if it has a nugget and its charge is not low and finally, it will search for a charging station if its charge is low. There are three high level behaviors, but there is no need to determine motivation. All reactions contained in the mid level pool are active, but the behaviors have the ability to rearrange the reactions to achieve their goals. For example, a behavior designed to search for food nuggets does not use the reaction, Seek Charge, but Seek Charge is still active; however, it is never given control of the system as long as Search is in control. So far, this is

nothing new. The IRA up to this point is a reactive interactive system much like all of the ones developed by other researchers.

The difference is the IRA possesses system detectable error. This will require the system to have memory. To determine whether the robot is in a trapped trajectory, memory will require past knowledge of what the robot has done. Interactive possibilities are what are desired. The system must have enough history and path information to know that there are no interactive possibilities occurring or that negative possibilities are occurring. Error is detected by the fact that interactive systems must possess a circular organization. The circularity of the organization must bring the system back to its base internal state. (Maturana, 1980) Error is detected by not returning to the base internal state. In other words, if Seek Charge is active, then the desire of the system is to reach the reaction, Dock Charge to recharge the system. If this does not occur, the system will lose its identity, it will die. How much memory and how much path information will depend on the system and upon the desired outcomes. This is a bottoms-up approach to error detection, by placing the error detection internal to the reaction activation. The advantage is that because error is detected by the system itself, is very fast. There is no time spent trying to detect error and there is no time spent planning for what to do about the error. Using BITE, “if a fault is detected, there can be significant delays in system response while the system decides what should be done.” (Lockner, 1980) The IRA is not so constrained.

The difference between the IRA and all other control architectures is that when the system becomes stuck, or finds itself in a “trapped” trajectory, the IRA does not tell the system what to do to get out of the situation that it has found itself in. Instead, it tells

the system to try some other interactive possibility, thus rearranging the order of the reactions until the system is free to continue its primary function, maintaining the circularity of its organization.

Below is a typical subsumption architecture as seen in Figure 2.1, but the generic case has been replaced with specific case to develop a scenario.

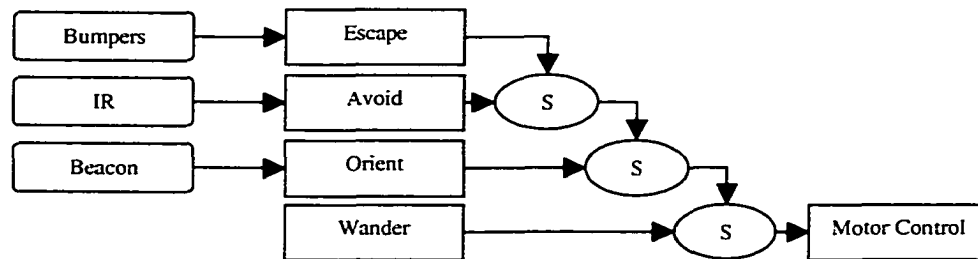


Figure 6.2, Subsumption Flow Chart

Within Escape and Avoid is a certain condition, a reaction, called Freeze. If all of the bumpers are tripped (the robot is being held) or all of the IR sensors are tripped (brightly-lit room), then the robot will freeze. This seems like a good idea. If the IR sensors cannot read, then the robot will not move. It will then be incapable of differentiating its environment with the linear order of subsumption. This, however, is not very interesting. In pure subsumption, there is no way to get the robot moving again without user intervention. This was demonstrated in Figure 3.1. Figures 2.1 and 6.2 are not very easy to analyze. Fortunately, there is a method already available for analyzing these types of systems, called finite state machines.

6.1 Finite State Machines

Subsumption defines each behavior as a finite state machine that is connected in a bottoms-up architecture. Analyzing the system as a finite state machine will allow

visualization and a better understanding of the function of each architectural type. A finite state machine requires the following:

$$z_v = f_z(x_v, s_v)$$

$$s_{v+1} = f_s(x_v, s_v)$$

z is the output, x is the input, s is the state and v is the time increment. The problem with subsumption is that there is no dependence on current or past state for the determination of current state. Transition Diagrams allow for visualization of the Finite State Machines. Transition Diagrams show how the finite state machines work, how they are interconnected, and how they move from one state to the next. The basic definition of how a finite state machines function is given in Gill (1962). The transition diagram for subsumption shows the basic functions of the system. Shown are the two possible states for the example theorized in Figure 6.2. Within the Behavior, Wander, are the Reactions, Escape, Avoid and Random. There are six inputs $\downarrow$ three bumpers and three distance sensors. Above each state is the input string to activate that state, and under the input string is the appropriate member from the output space.

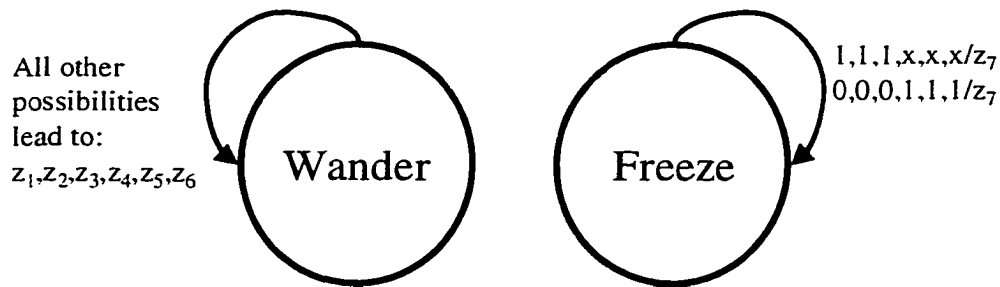


Figure 6.3, Transition Diagram Subsumption

There is an output space $z = \{F, FR, FL, R, RR, RL, S\}$. The definitions are, F = Forward, FR = Forward Right, etc. There is no connection between Freeze and Wander. If Freeze

is activated, Freeze will control the robot. If Wander is activated, Wander will control the robot. Also, there is no connection between the sub machines in Wander, i.e. Escape, Avoid and Random. This finite state machine is independent of previous state. It is a simple input-output device. It is not a dynamic system, unless coupled with its environment.

$$\begin{aligned} z_v &= f_z(x_z) \\ s_{v+1} &= f_s(x_z) \end{aligned}$$

The output depends only on the input and the state depends only on the input. This is the main problem with subsumption. Without interconnection, there is no possibility of achieving higher level goals or having the capability for system detectable error. Interconnection is required to move from one state to the next, whether the inputs call for the transition or not. Interconnection also needs to be based on some physical result. How is the interconnection made?

History is required to verify that the system is in one of the “trapped” trajectories. The history will allow the system to know that it is in a “trapped” trajectory and the interconnection will allow the system to leave the “trapped” trajectory. This is the basic idea behind the hormone system described by Gadanho. “The hormone values can be (positively or negatively) high enough to totally hide the real sensations from the robot’s perception of its body.” (Gadanho, 1998)

For an architecture to possess system detectable error, it must be able to “sense” that the IR is not working and turn Avoid off for the input that is trapping the system. It must also be able to “sense” that the robot is being held for too long and then try to get away from what is holding it by turning Freeze off. The desire is for a control system that is reactive, non-representational, and yet capable of achieving high level goals. “The

Norman-Shallice model achieves this by the provision of two kinds of control structures: (a) horizontal *threads*, each one comprising a self sufficient strand of specialized processing structures; and (b) *vertical threads*, which interact with the horizontal threads to provide the means by which motivational factors can modulate the schema activation values.” (Reason, 1990) This can all be done with the mechanism presented in Figure 6.1. The Reactions are pure subsumption (horizontal threads), but the Behaviors (vertical threads) allow for rearranging the linear progression required of subsumption without adding an explicit planner. These vertical threads are the interconnections required to achieve high level goals. The vertical threads are based on system detectable error and emotion. Using history and the “trapped” trajectories, the system will be able to rearrange the reactions in such a manner, that the system can achieve high level goals and more robustly deal with dynamic environments.

The advantage to this architecture is that the new program will not calculate what the robot “needs” to do to satisfy all of the behaviors with every pass of the control loop. This applies to all of the behaviors below the highest active behavior, as well as for each reaction. This will greatly reduce processor time. This system requires only one microprocessor to achieve high level goals. This system is capable of system detectable error without the use of an external observer, a separate piece of software code or the use of an additional processor. More importantly, it will not compare one sensor to another sensor to determine which one is not functioning. It will not require a programmer to plan for every contingency, but it does require the programmer to know what constitutes an error for the system in design. When the system gets stuck, it will try multiple interactive possibilities until it has freed itself. This implementation still has all of the

advantages of the Subsumption architecture. It is still completely modular. Adding a new behavior entails adding it to the program and determining how it should be activated with regard to the other Behaviors. There is still a direct link between the Sensory Data and the Motor Actuation. The difference is that we now have a Finite State Machine with memory. The new implementation still offers “real-time robustness”. (Jones, p267) This system will avoid “clever engineering” and planning. It will be very simple. There is no need for the omniscient programmer to determine which reaction supercedes which reaction, under all conditions. The system will allow for lower level reactions to supercede higher level reactions. This is demonstrated with a similar *Mathematica* program. In this case interconnection was made between Freeze and Wander. This allows the robot to escape when held or to venture when the distance sensors are not functioning.

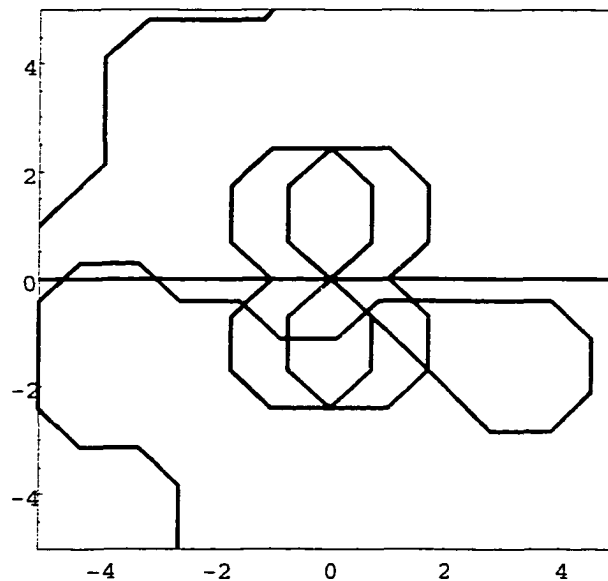


Figure 6.4, Trapped Trajectories with Interconnection

There is only one interconnection made, but this allows the robot to escape from one of the possible trapping trajectories. Figure 6.5 shows the Transition Diagram for the new system.

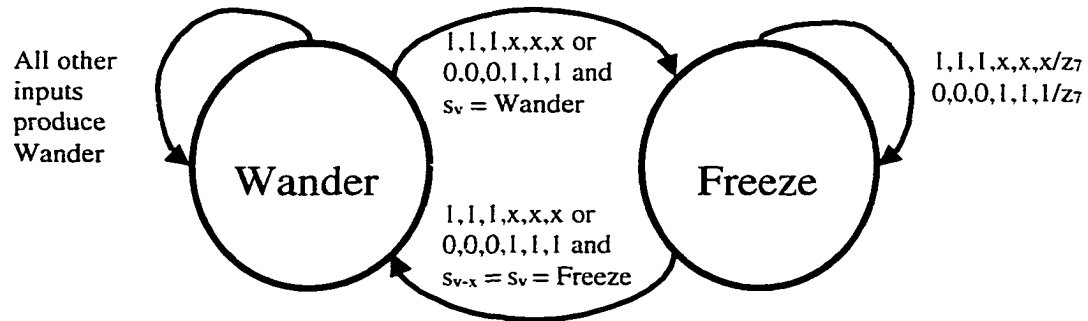


Figure 6.5, Transition Diagram, IRA

By increasing this interconnection, it is conceivable that the robot will be able to avoid all of the “trapped” trajectories, or at least be able to escape from them. System detectable error requires more than just knowledge of history. History, however, has strong and important connections to function. Function is what error is based upon. (Brickhard, 1999c, Stary, 1995) History is required to detect the “trapped” trajectory, but it does not and cannot plan for how the robot will extricate itself. The above implementation uses the fact that the robot is in a “trapped” trajectory along with history to “decide” when the system is dysfunctional. In essence, if a sensor has been actuated for too long the system becomes desensitized to that sensor and ignores that sensor. The main goal of the robot is to explore its environment. It must be able to detect when it is not exploring, not reaching the base internal state. The interactive possibility of exploring can only be manifested when the robot is not in one of the “trapped” trajectories. The robot can therefore use these “trapped” trajectories to determine if it is not achieving the goal. This is shown by the moving from the Freeze state to the Wander

state over many iterations. A *Mathematica* program was written to simulate a random environment and a bar chart was created showing the number of times the robot was in each output. The bar chart shows pure subsumption and the interactive representation with one layer of interconnection.

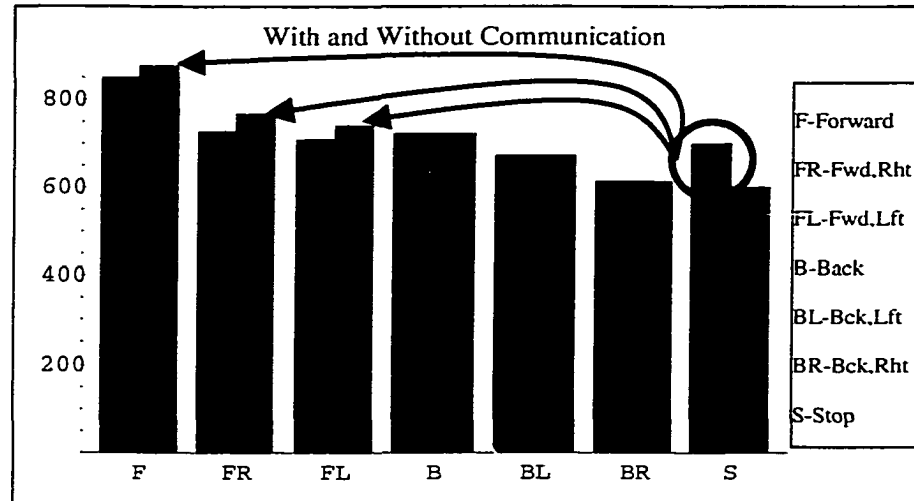


Figure 6.6, Distribution of States

Pure subsumption is depicted as the left bar and interactive representation as the right bar. The one layer of interconnection leads to the robot being trapped at Freeze less of the time and the number of Wander increasing. This allows the robot to be able to better perform its main function – exploration. See Appendix A for complete results.

6.2 Performance Demonstration

The above discussion shows what happens when the IRA detects an error and then escapes from the “trapped” trajectories. Robots are mechanical and electronic machines used in an environment; they will not perform the same in every experiment. To make comparisons with any meaning, a statistical database must be obtained, but since no two situations will be identical, computer simulations are used to produce consistent results. (Nehmzow, 2000, Brooks, 1995b) One performance test involving

simulation is goal directed behavior. Instead of having Wander as one of the state machines, it was replaced in a *Mathematica* simulation with a behavior called Goal. The main difference is that Goal, instead of randomly wandering around, has a desired location to achieve. The simulation involved the number of time steps required for pure subsumption and the IRA to reach the desired location. Both models will be faced with an open environment, but to make the simulation more interesting, occasionally *Mathematica* will generate a random set of input variables.

First look at the full transition diagram for a subsumption model with this goal directed behavior. Figure 6.7 shows that there is no interconnection between the behaviors or between the reactions. The lack of interconnection will allow this model to be trapped for extended periods of time due to certain sensory inputs.

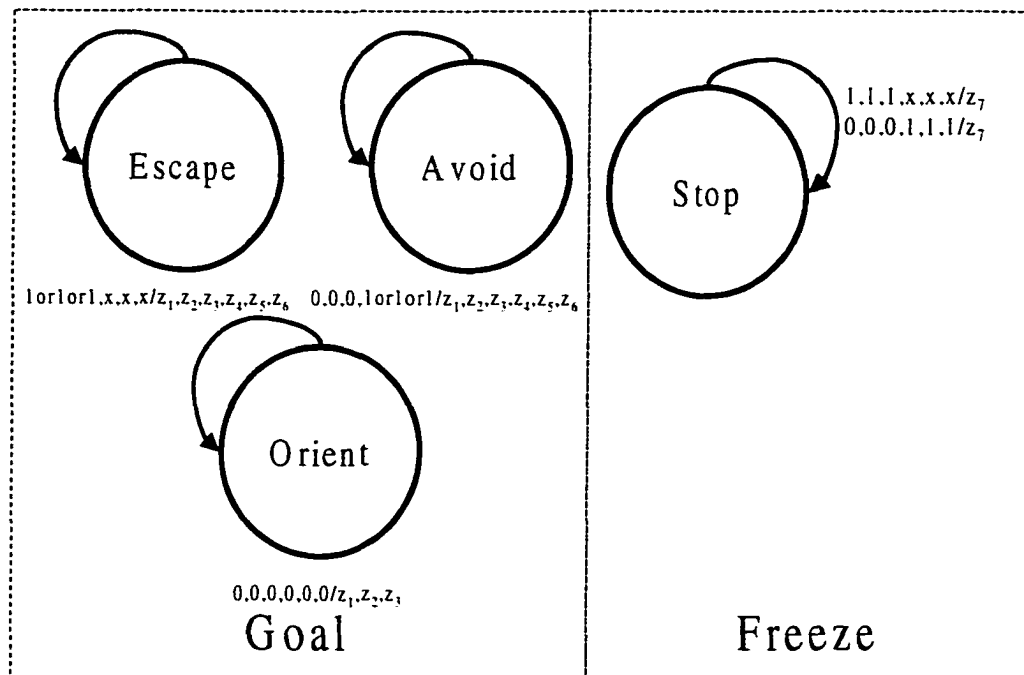


Figure 6.7, Subsumption Goal Directed

For the IRA model made with only one level of interconnection, a connection is made between the two Behaviors, Freeze and Goal, and more specifically between the Reactions, Stop and Orient. The two models are nearly identical except for one set of connections, shown in Figure 6.8. The interconnection allows for the system to escape from the trapped trajectories that it encounters when the randomly generated inputs cause the system to go off of track.

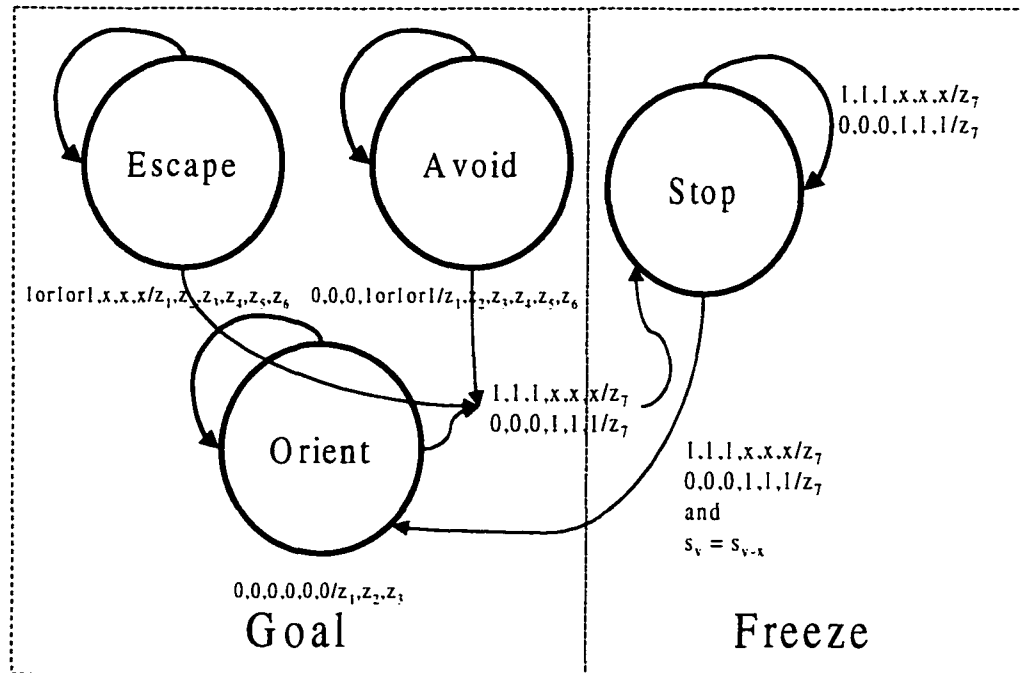


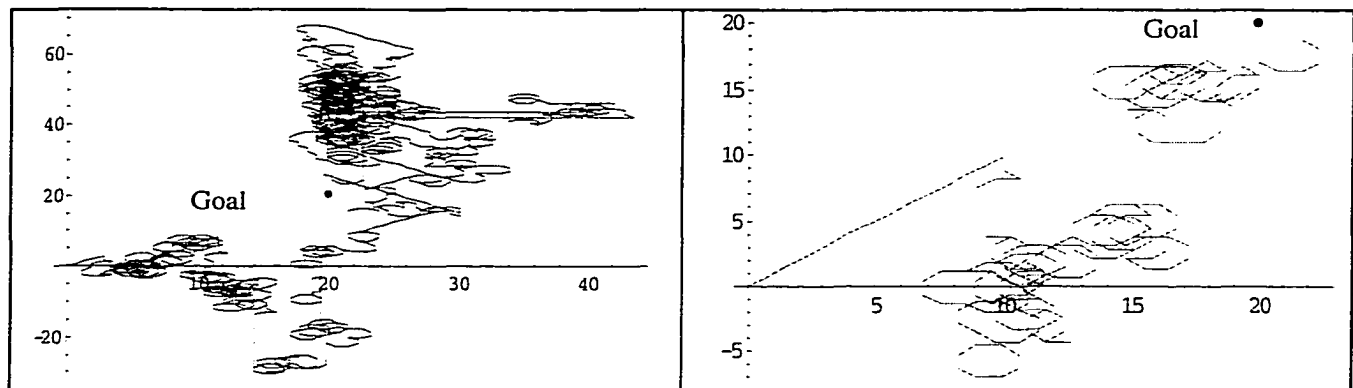
Figure 6.8, IRA Goal Directed

The simulations were then run for forty iterations and a statistical analysis was performed on the data sets. Table 6.1, shows the results of the two simulations. The mean is the average number of steps required for the robot to reach the “goal”.

	IRA	Subsumption
Mean	273	548
Standard Deviation	215	1002
Variance	46150	1.004 10 ⁶
Population Size	40	40

Table 6.1, Performance Test

Using the above data, a Mean Difference Test was performed with a significance level of 0.05. The result was that the difference in means was significant. In other words, the IRA got to the desired location quicker than pure subsumption. In fact, the IRA achieved the desired location (20,20) twice as fast as subsumption alone. See Figure 6.9 for a comparison of a typical IRA path and a subsumption path.



Subsumption

IRA

Figure 6.9, IRA vs Subsumption

It is apparent that the IRA model (right) achieved the point (20,20) much more quickly than did the subsumption model (left). The program was set up to come out of the while loop when the system was within a radius of two from the desired point. See Appendix B for complete results.

Simulations alone are not valid for developing real control systems. Simulations are only valid for rapid prototyping and performing multiple tests. The real world is the

real test. The real world will allow for what is truly an error to emerge. The real world has the dynamic interactions required to manifest the errors that are pertinent.

“Don’t use simulation as your primary test bed. In the long run you will be wasting your time and your sponsor’s money.” (Brooks, 1987) A minimalist robot is required to allow for the emergence of error without complicating the interaction space with too many variables. As Einstein said, “Everything should be as simple as possible, but no simpler.” (Kurzweil, 1999) The minimalist robot will provide a platform with sensors, actuators and structure. Section 7 details how the IRA is very close to being representative of an Autopoietic system. The only required element it lacks is self-replication. Section 8 develops the IRA for use in a simulated world. This allows for multiple runs with the same initial conditions. Section 9 will then apply the IRA to a minimalist robot to insure that the IRA applied to an embodied autonomous system can become self-sufficient and situated as well as developing design guidelines for both the IRA and a minimalist robot.

7 Autopoiesis

The above architecture closely resembles an autopoietic system. In fact it has many of the requirements associated with an autopoietic system. In the following discussion Maturana uses machine, unity and system interchangeably.

“An autopoietic machine is a machine organized as a network of processes of production (transformation and destruction) of components that produces the components which: (i) through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them; and (ii) constitute it (the machine) as a concrete entity in the space in which they (the components) exist by specifying the topological domain of its realization as such a network.” (Maturana, 1980)

These autopoietic systems are required to have circularity built into their organization; this is the requirement of systems to be autonomous. This circularity requires that the system subordinate all changes in the structure to maintaining the circularity of the organization. The organization is what details what transformations and interactions with the environment that the machine can undergo. The structure is the physical embodiment of the organization. The organization of the machine allows the machine to exist. If the organization becomes broken, the machine will cease to exist it will die. The organization of a machine is independent of the physical components and the structure that make up the machine. There are reactions and behaviors that define the machine and constitute the organization for the machine. These reactions and behaviors are independent of the physical entities that perform the reactions and behaviors.

The organization of the machine can be manifested by many different structures. (Maturana, 1980) Take for example the reaction of avoiding obstacles. Many different

types of sensor suites can accomplish this reaction. The designer could use infrared detectors, or photoresistors or opto-schmitt triggers or cameras or . . .

The key is that an autopoietic machine maintains its organization as invariant. The organization is what defines the machine as a unity. An autopoietic system must maintain its organization to remain viable. The system has no comprehension of function, but function is apparent to the observer. “As a consequence, all changes must occur in each living system without interference with its functioning as a unity in a history of structural change in which the autopoietic organization remains invariant.” (Maturana, 1980) So again, function is only apparent to an observer. The observer can define the function of the system, and relate inputs to outputs, but the system can only see whether the organization is remaining invariant. The system will then need to make changes to insure that the organization remains invariant.

“The compensatory changes that an autopoietic system may undergo which retains its identity, may be of two possible kinds according to how its structure is affected by the perturbations: they may be (a) conservative changes in which only the relations between the components change; or they may be (b) innovative changes in which the components themselves change.” (Maturana, 1980)

The autopoietic system can either rearrange the relations between the components of its structure or it can change the components themselves in order to maintain its organization.

The IRA nearly meets all of the requirements for an autopoietic system. The IRA attempts to maintain the circularity of its organization. If there are no errors or no trapped trajectories, the IRA control will continually check all of the pertinent reactions until it reaches the base reaction. The base reaction is the desired reaction for the active behavior: Seek Nugget, Seek Charge, or Seek Home. If the circularity has become

broken, the IRA is trapped in one of the higher level reactions; the IRA chooses Option (a) from above. The IRA as conceived here does not have the capability to form “innovative changes”, but it does have the capacity to affect the relations between reactions. It does this by detecting trapped trajectories and then rearranging the reactions to maintain the circularity of its organization. This allows low level reactions to supercede high level reactions. It does not detect a fault, only that the circularity is broken constituting an error.

The IRA does not represent its environment, only the interactions with the environment. “The continuous correspondence between conduct and ambience revealed during ontogeny is the result of the homeostatic nature of the autopoietic organization, and not of the existence of any representation of the ambience in it.” (Maturana, 1980)

The only item missing, albeit an important item, is that the components do not produce themselves in the IRA, they only maintain the organization detailed by the designer. The IRA is not a living system, but it is autonomous, it does possess system detectable error, and it does this without the use of an external observer.

The following section demonstrates the robustness of the IRA in a simulated world. The IRA is directly compared to a subsumption model for various errors and no error conditions.

8 IRA AnSim Simplified Simulation

The Mathematica simulations are very simple worlds with no real interaction. The goal of this research is to design a simple robot and control system that can function in dynamic environments. The problem is that even the best sensors tend to be unreliable. (Brooks, 1995b) This unreliable data can then trap the robot. “If the correspondence exists, then the representation exists and it is correct, while, if the correspondence does not exist, then the representation does not exist, and so cannot be incorrect.” (Brickhard, 1998) This is key to all previous architectures. In order to avoid these problems researchers focused on improving the reliability of the sensors, or providing multiple redundant sensors. The robot would be forced into following the progression of its control, even if the sensor it is obeying were in a fault state. This does not provide the required robustness sought in this research. Minimalist robots do not have the privilege of multiple redundant sensors and minimalist robots do not have the space for the highly reliable sensors.

The IRA looks for cases of error, but not specifically of the sensory kind. It looks for cases where it is not able to predict future interactions or have future interactions. The cases where there are no future interactions, which result in the “trapped” trajectories, were shown in previous sections. Once the IRA discovers these “trapped” trajectories, it rearranges the order of the reactions in a hope to escape.

8.1 IRA Whisker Error

AnSim was modified to look for the “trapped trajectories” and to move out of them. The first demonstration is for the case of one of the whiskers (bump sensors) being on all of the time, effectively stuck. Having worked with small robots, this is very feasible. A bump sensor is simply a wire in a loop. If the wire touches the loop, the sensor is on. If the robot were to hit an object too hard or the wire were to get caught on some object, the sensor could easily be turned on continuously. What happens to most architectures was demonstrated in Figures 3.4 and 3.5. The robot basically cannot move. Now with the addition of the IRA, the robot is capable of moving out of the trapped trajectory and to continue exploring its environment. The performance is not degraded. The system detects that it is in a trapped trajectory, it then moves out, away from what is trapping it and continues to explore its environment. This clear difference is demonstrated in the following figure.

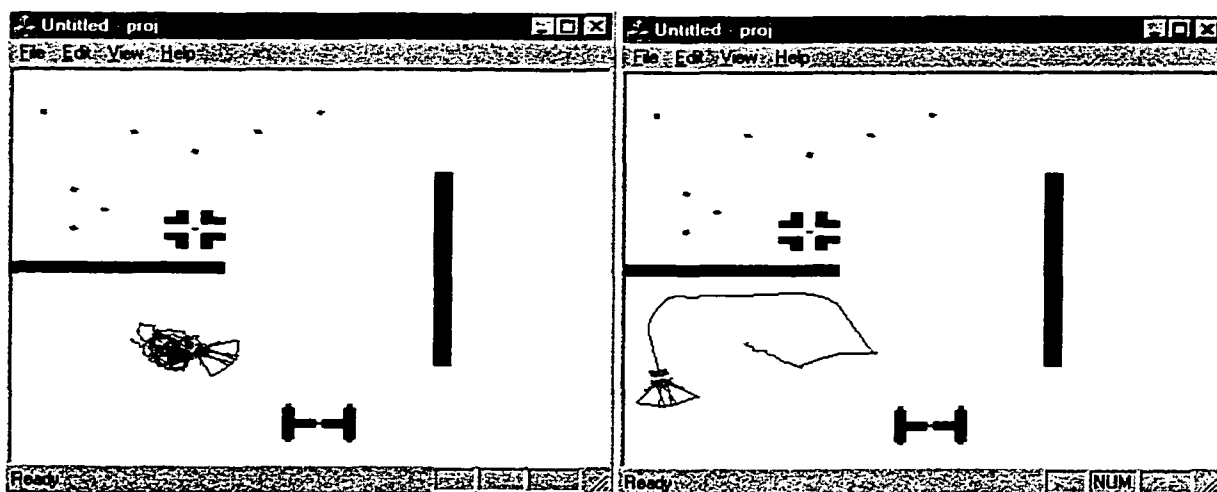


Figure 8.1, IRA Whisker Error

The above figure clearly shows the advantage of the IRA. The animat is stuck for a short time, but then breaks out and returns to exploring the environment. No other form of

error detection could have handled this situation. There are no multiple redundant sensors for comparison purposes. If there were a time-out for Avoid Obstacles, the robot would move out for one iteration and then right back into the pattern. The use of a time-out on Avoid Obstacles would also prevent the animat from successfully navigating the environment.

8.2 IRA Distance Sensor Error

The case of an infrared sensor failure is also addressed. This could easily happen if the room was too bright. This type of error will be much more prevalent for the minimalist robot. The distance sensor planned will be a photoresistor. This type of sensor is very sensitive to ambient light. Figure 3.5 showed the “trapped” trajectory. The addition of the IRA to the control architecture allows the animat to detect that it is trapped and then to break out of the “trapped” trajectory and to continue exploration of its environment. This is demonstrated in the figure below.

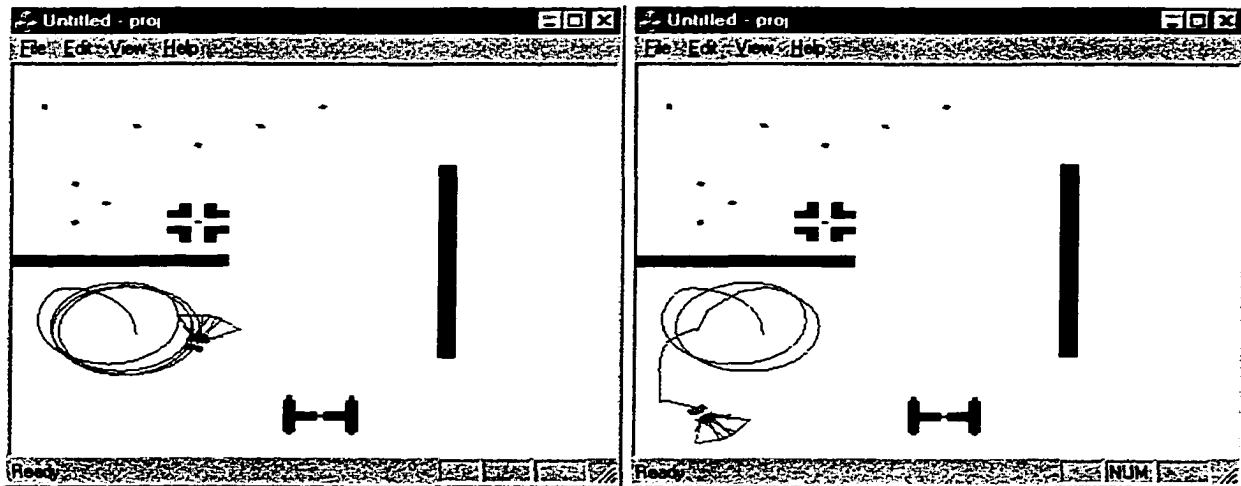


Figure 8.2, IRA IR Error

The IRA is not looking specifically for sensory errors; it is only looking at the fact that future interactions are not occurring. It does not tell the animat what to do when it

senses that the animat is trapped. It only tells the animat that the progression of its reactions are not reaching the base internal and to try something else. This is demonstrated in the following example. Under normal operation, the animat becomes trapped in a corner. This is caused by the fact that the Avoid Obstacles command, for all three-distance sensors being tripped, is to turn left. In this case, it cannot turn left, so it just sits there. For the IRA based animat, it notices that it cannot escape using Avoid Obstacles, so it tries something else until it escapes. This is shown in the following figure. On the left is subsumption. The animat becomes stuck and cannot clear itself. On the right is the IRA where the animat attempts different reactions until it escapes from the corner.

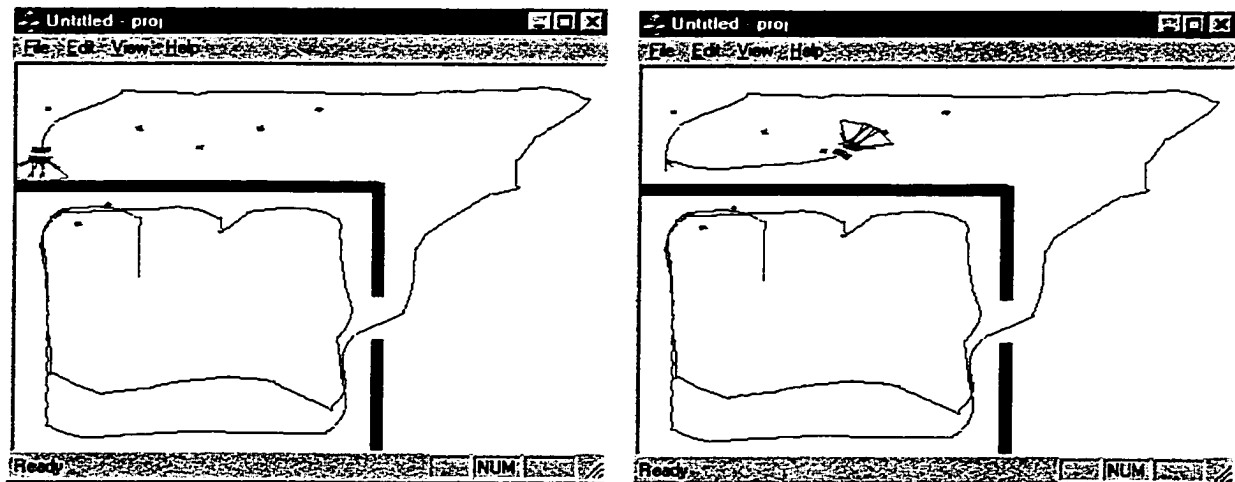


Figure 8.3, IRA Escape from a Corner

This demonstration clearly shows the advantages of the IRA. The IRA in this case is only for a system with one behavior, to explore. It has only five sensors, three IR and two whiskers. The IRA was then applied to the simulated animat for multiple behaviors, first two and then three. Finally, it was placed on a minimalist robot that was designed for this research. The minimalist robot is the true test. A simulated world is

“perfect”; the sensors are never in error. They do not trip without reason. They do not give false readings. Their performance does not change over time. The robot responds exactly the same way every time. From the beginning, this research was meant to be applied to a real system operating in a real world.

8.3 The Byzantine Generals Problem

The Byzantine Generals Problem is often used to model and solve the problem of determining a faulty process in redundant systems. “In order to avoid a system’s being vulnerable to the failure of one component, it is reasonable to use several sensors redundantly.” (Brooks, 1995a) Then there must be a way to determine which ones are “good” and which ones are “faulty”. The typical way to determine which redundant sensors are loyal and which redundant sensors are disloyal is by voting. (Gupta, 1999) This again requires an external observer, but the solution to this problem yields quantitative results that can be used in this research.

The Byzantine Generals Problem is posed in the following scenario. A city is under siege by the Byzantine Army. The Commanding General has deployed his army in units around the city that are each commanded by one of his Lieutenant Generals. The problem is that any one of the generals could be a traitor, including the Commanding General. When messages are passed, either to Attack or Retreat (boolean), the loyal generals must all perform the same action. The problem is determining who is loyal and who is a traitor. The solution to this problem has been solved in a variety of ways, but there is one thing in common with all solutions. If the number of generals is “N”, and the number of traitors is “t”, then a solution is possible if $t < N/3$. (Brooks, 1995a, Gupta,

1999, Kesteloot, 1995) These solutions all require an external observer to perform the voting and the solution comes from iterations.

Now examine the IRA case for one faulty whisker and one faulty IR sensor from above. This scenario involves redundant sensors, the whiskers and the IR sensors are both there to allow the animat to explore dynamic, unknown environments by keeping the animat from becoming trapped in a corner or up against an unknown object. There are five redundant sensors, $N = 5$. There are two faulty sensors, $t = 2$. The animat is capable of fully exploring its environment. It does not become trapped, either by the environment or by the finite state machines response to the faulty inputs. Therefore, for the case of redundant sensors, the IRA is able to tolerate $N/2.5$ faulty sensors. This is a substantial improvement over the solutions to the Byzantine Generals Problem. A solution requiring the use of an external observer to perform a voting scheme would only be able to determine that one sensor is faulty in a suite of five sensors. The IRA does not determine that a sensor is faulty, there is no voting scheme. The IRA determines that it is trapped by one of the inputs and then tries its other reactions. By not determining “faulty”, the IRA is able to handle one more sensor error and still perform its main task.

9 Full IRA AnSim Simulation

The above discussion showed that the IRA has the possibility of being more robust than other architectures. It has the capability to sense when it is in error, system detectable error. Because system detectable error is based upon the truth-value of producing interactive possibilities and not upon an explicit definition of a sensory fault, the IRA does not formulate or plan a solution to escape from the error situations. It only determines that its current actions are not producing positive interactive possibilities, the circularity is broken, and it must try something else. This is a bottoms-up approach to error detection. The addition of new behaviors only required the addition of what constitutes an error for the new behaviors.

9.1 IRA AnSim Induced IR Error

For this simulation, AnSim was set up with five Reactions: Search, Avoid Obstacles, Follow Wall, Avoid Bumping and Seek Nugget. These five Reactions fit into the following subsumption flow chart:

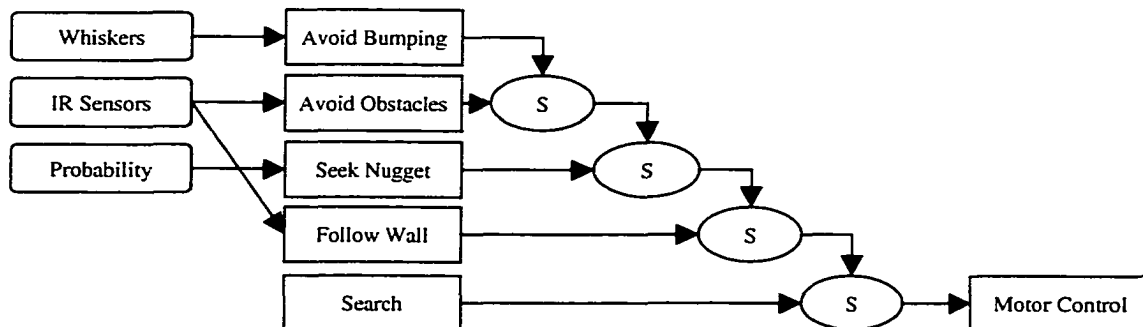


Figure 9.1, AnSim Subsumption

Seek Nugget has only one real function, it will shut off the Follow Wall Reaction. This makes the animat very good at exploring the environment. Without the Seek Nugget, the animat is attracted to walls and then just makes a path around its environment. The Subsumption flow chart provides the necessary pointers for the IRA to work on. Each reaction points to the next and finally down to the lowest reaction. If the reaction that is currently working is not delivering the system back to its lowest reaction, there is an error, no circularity. The memory required to detect that the organization has lost its circularity is contained within each of the reactions – internal to the control. The loss of circularity is based upon an internal criterion of error, not upon external influences. Figure 9.2 shows the no induced error situation for both the IRA and Subsumption.

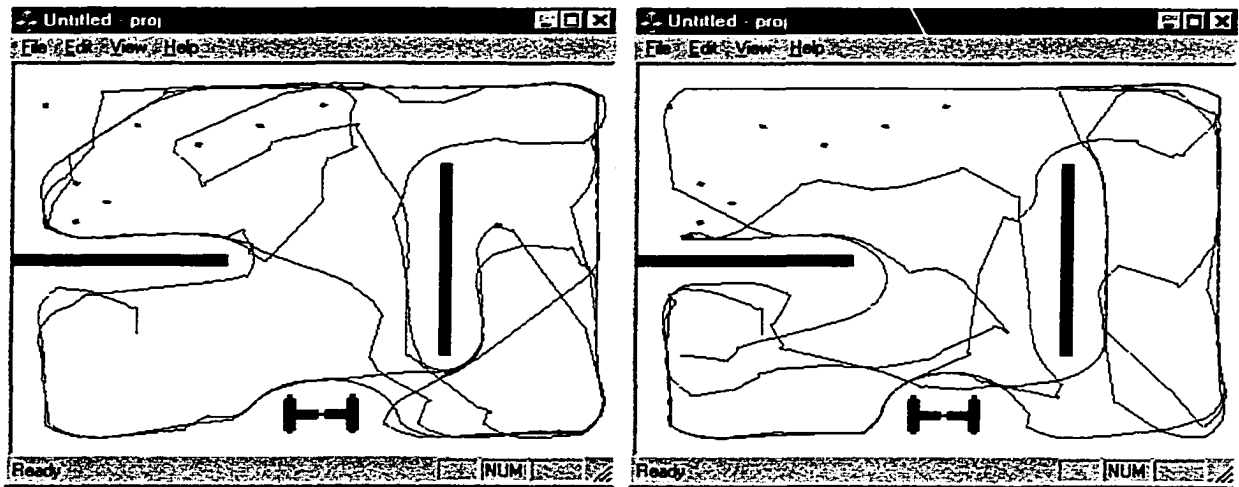


Figure 9.2, IRA & Subsumption No Errors

Both animats explore the environment equally well. The IRA animat is on the left and the Subsumption animat is on the right. Both illustrations have been run for the same number of iterations with the animat starting at the same location. The animats do not track the exact same path, but the paths are very similar. The Subsumption animat uses Seek Nugget to move off a wall. This reaction is based upon a randomly generated

number to whether Seek Nugget will subsume Follow Wall. The IRA animat has the Seek Nugget reaction as well as detecting that it is in an attractor, Follow Wall, and to move away from the attractor.

As was discussed in previous sections, the simulated world does not possess the necessary dynamics to produce faults. There is no real interaction between a dynamic real world and a dynamic real animat. Instead, in the simulated world, everything is constrained, essentially fault proof. Everything will give the same result every time. To more closely represent a dynamic environment or a dynamic animat, a small line of code was inserted into the Sensor.cpp code. This Sensor.cpp portion of AnSim determines the interaction between the environment and the animat. To represent an error situation, the following bit of code was added to Sensor.cpp:

```
if ( MyRand () > .05 )
{
    if ( Intersection( winPtr(), this ) || _id == 3 )
    {
        _status = TRUE;
        return;
    }
}
```

This portion of code will automatically trip the right infrared sensor if the randomly generated number is greater than 0.05. This is representative of an unreliable sensor, which is quite possible when using minimalist robots. The IRA will not determine that the sensor is in error and then plan for a contingency. It will only determine that the interactive possibilities are not being met and that the animat should try to use another one of its reactions. This is demonstrated in the following figure.

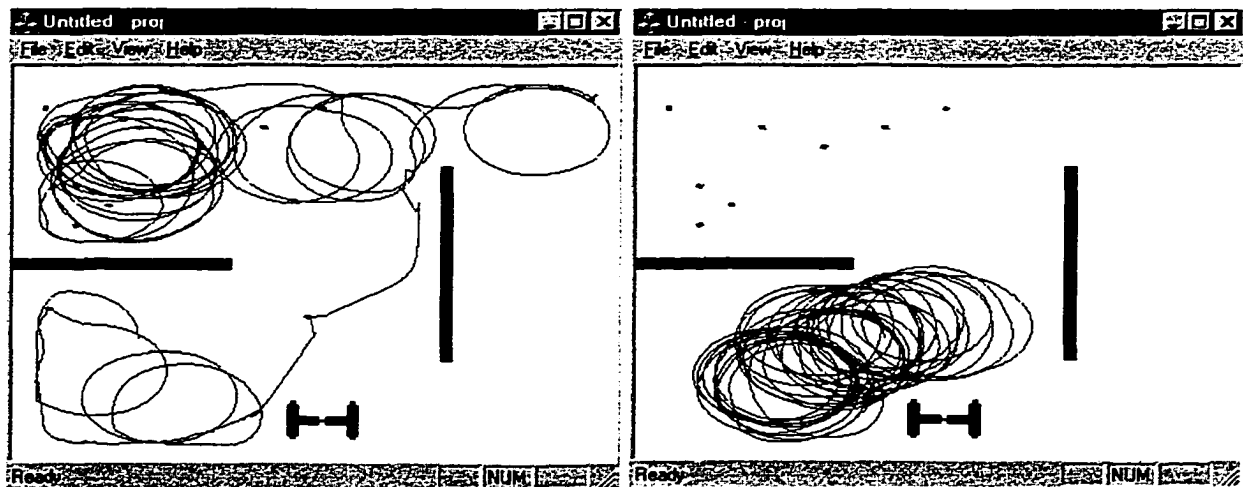


Figure 9.3, IRA & Subsumption with Induced IR Error

The IRA animat is on the left and the Subsumption animat is on the right. The Subsumption animat becomes “trapped” and cannot get out, while the IRA animat is able to determine that there is a problem and by using other reactions it is able to explore the environment. Both illustrations were run from the same starting point for the same number of iterations. The lower right portion of the environment became a trapping region for the Subsumption animat, while the IRA animat was able to enter that region and escape from that region. The IRA animat was able to explore nearly three quarters of the environment while the subsumption animat was only able to explore a little over one quarter.

9.2 IRA AnSim Induced IR and Whisker Error

The above simulation was very useful. It showed the ability of an IRA based animat to detect that there was an error, interactive possibilities not occurring. To fully test the capabilities of the IRA, both types of sensors will now have an induced error, based upon probability. The same lines of code for the IR error were used in this

simulation, while the following lines of code were added to Sensor.cpp for the whisker sensor:

```
if ( MyRand () > .01 )
{
    if ( Intersection( winPtr(), this ) || _id == 1 )
    {
        _status = TRUE;
        return;
    }
}
```

For this sensor, the whisker is more likely to have a problem. It is simply a wire in a wire loop. If it becomes stuck, it will stay stuck for a lot longer. Because of the low value for the random number, the whisker will trip more often and will stay tripped for longer. This kind of unreliable behavior is very prevalent among minimalist robots using whisker-type sensors. The following figure demonstrates the IRA animat and the Subsumption animat.

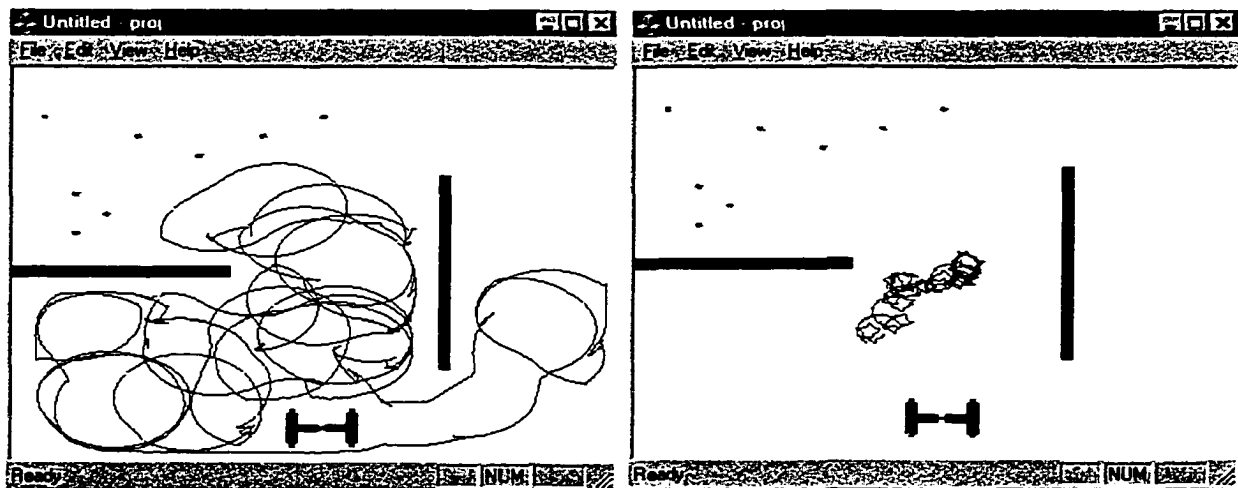


Figure 9.4, IRA & Subsumption with IR and Whisker Error

The IRA animat is on the left and the Subsumption animat is on the right. The Subsumption animat is basically stuck, unable to move out of this attractor. Both animats were run for the same number of iterations, starting from the same point. The IRA animat, on the other hand, is able to detect that it has an error and then to move away from the error to explore its environment. Figure 9.4 clearly demonstrates the advantages of system detectable error and what that means to a mobile autonomous robot. In this case, the IRA animat was able to explore more than half of its environment. The subsumption animat was basically unable to explore its environment, almost completely trapped in its devotion to its correspondences. There is one more case, the case of a total sensor failure. This is demonstrated in Figure 9.5. Both the right infrared and the left whisker have failed in the on position.

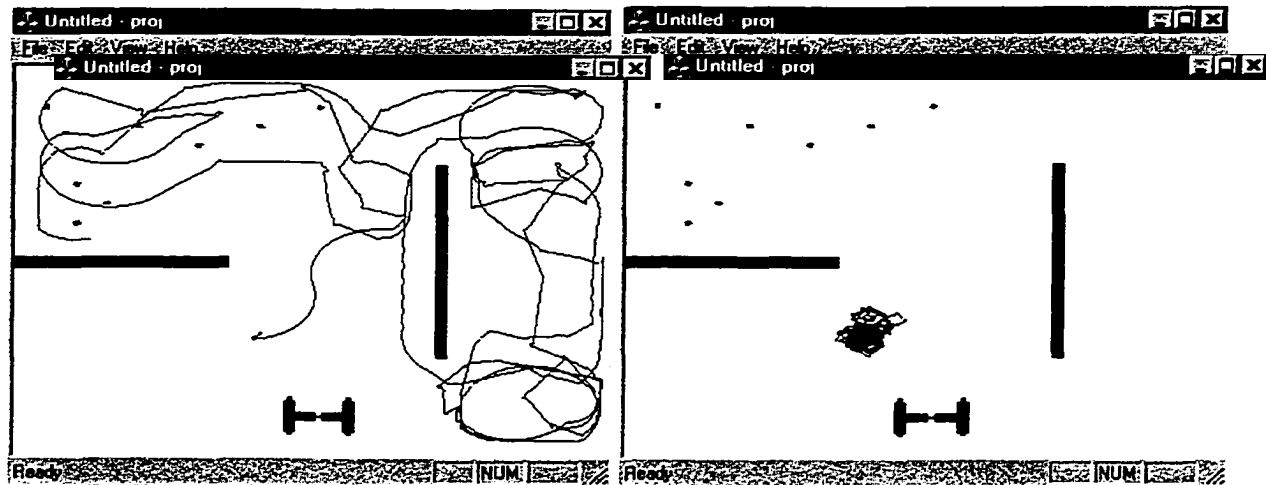


Figure 9.5, IRA & Subsumption with IR and Whisker Failure

If a system has a total failure, the IRA demonstrates how robustly it can handle the loss of a sensor and how it detects that the sensor is not functioning. The IRA does not plan what to do when a failure occurs, only that the current plan of action is not working and the system needs to try something else. It shuts off the reaction from the sensor that is

causing a “trapped” trajectory and then allows the regular progression of reactions to take over. As each reaction fails to remove the system from the trajectory it will be shut off until the system can free itself. The IRA is able to explore its environment very well, exploring nearly three-quarters of the environment. The subsumption animat was again completely trapped by its correspondences. The speed of the system was not effected by the errors. For this simple model, with only one behavior, the IRA has a coverage of 40%, while subsumption has a coverage of 0%. In other words, the IRA is able to lose 40% of its sensory data and still accomplish its main task, without any substantial penalty in speed or capability.

As stated in the Philosophy section, simulations are not sufficient. This AnSim simulation proves the feasibility of the IRA for one behavior.

9.3 IRA AnSim Two Behavior Simulation

The above discussion clearly showed how the IRA was able to detect that it was not achieving its base level reactions, and then to move out of what was trapping it. For one behavior, the IRA had a coverage of 40%. This section will demonstrate the capability of the IRA for two behaviors. In this section the animat has two behaviors: Seek Nugget and Seek Charge. This demonstration uses the following subsumption flow chart coupled with the system detectable error detection and correction of interactive representation.

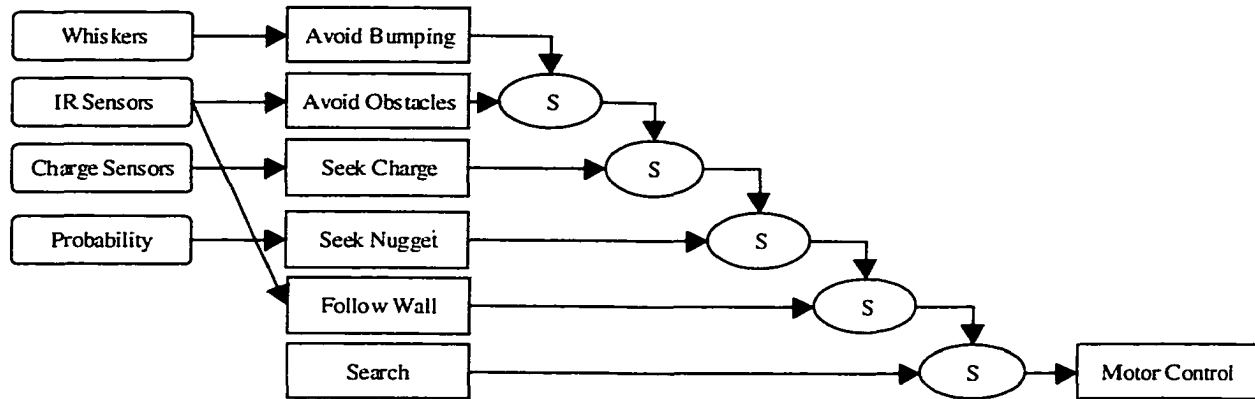


Figure 9.6, Two Behavior Subsumption Flow Chart

9.3.1 Two Behaviors – No Induced Errors

For the first demonstration, there will be no induced errors. Both the IRA and subsumption will be allowed to run in the same simulated world. These two behaviors are bringing the animat closer to an interesting system. The animats are both autonomous, they are both situated and they should both be self-sufficient. Of course, neither animat is embodied; this is still a simulated world. The following figure demonstrates what happens to both animats when started at the same location, in the same environment.

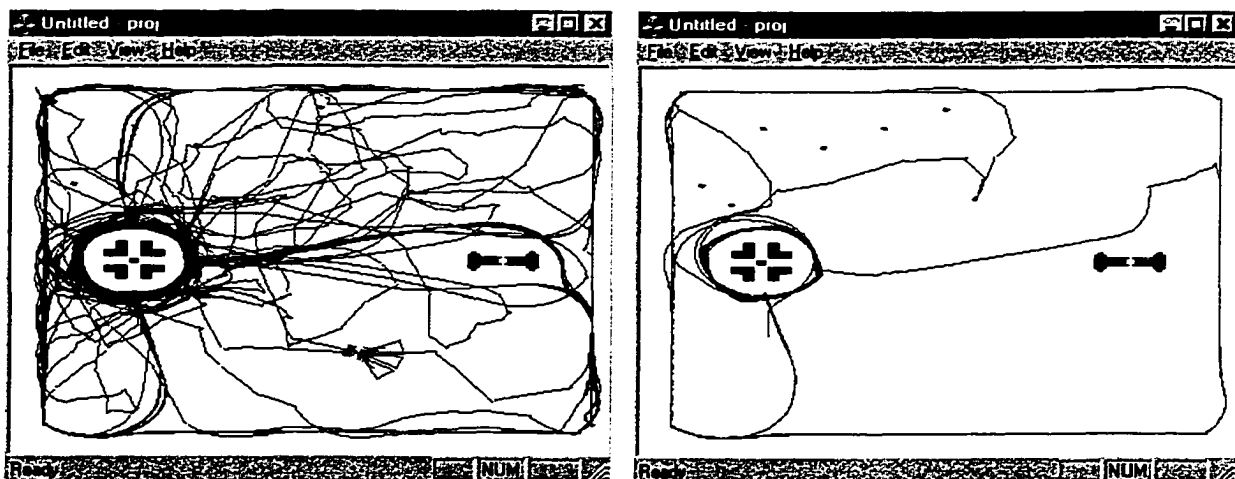


Figure 9.7, Two Behaviors No Errors

This demonstration exposed an unexpected “trapped” trajectory. The behavior, Seek Charge, uses five sensors to locate the charging station. Three of the sensors are located in the front of the animat that resemble the IR sensors in location and shape. The other two are on opposite sides of the animat. If Seek Charge is activated, when the animat is next to the charging station, the animat becomes trapped. It will just circle the charging station. Only the side sensor is activated, the command is for the animat to turn toward the sensor, but Avoid Objects will not let the animat make a sharp enough turn to activate the front sensors, so the animat becomes trapped. **There are no sensory faults.** The sensor is correct in what it is sensing and the animat is correct in its reaction to the sensor. The problem is that the animat has lost its intended purpose. It is not able to charge. It does not return to the Dock Charge reaction. The circularity of the reactions has been broken and the animat dies. The subsumption animat was able to recharge three times but it ran for only three minutes before it became trapped and died.

The IRA animat on the other hand, was able to detect that it was in a “trapped” trajectory and to move away from what was trapping it. There was no plan developed, in fact, the designer did not and could not anticipate this particular “trapped” trajectory. The IRA animat was able to recharge countless times, on the order of twenty times. The demonstration was terminated at twenty plus minutes with the animate continuing to explore the environment and recharging when necessary. A simple time-out on seek charge would not have fixed this problem either. The time-out would cause the animat to move out for one time step and then right back into the same trapped trajectory.

9.3.2 Two Behaviors – With Induced Faults

For this demonstration, the animat has lost one of its IR sensors, one of its whiskers and one of its charge detectors. This results in a thirty-percent reduction in sensory capability. In this case the sensors are tripped in the “ON” position. There is no functional redundancy built into the Seek Charge reaction. All five sensors are required to reliably return to the charging station. Since there is no redundancy with these sensors, the Byzantine Generals Problem is not really applicable. The following figure demonstrates what happens to the IRA and the subsumption model for the same sensor faults.

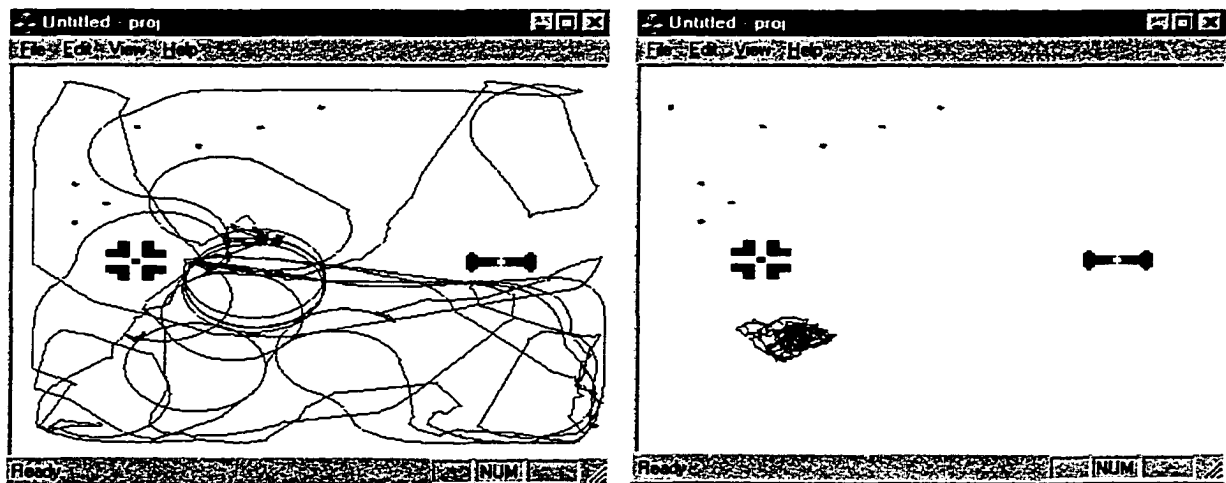


Figure 9.8, Two Behaviors – 30% Coverage

The figure, again, clearly shows what will happen to a system that cannot determine that it has lost its functional ability. The subsumption model is required to respond to Avoid Bumping. It continues to respond to this input string until it dies. The circularity of its organization is broken and the animat dies. The IRA animat, on the other hand detects all of the trapped trajectories and explores its environment until it needs to recharge, it then locates the recharging station and recharges.

9.4 IRA AnSim Three Behavior Simulation

This section will demonstrate the capability of the IRA for multiple behavior operation. This demonstration has three behaviors: Seek Nugget, Seek Home and Seek Charge. This allows the system to be self-sufficient; it will actively seek a recharging station if the charge is low. It is situated; there is no internal map or *a priori* knowledge about the environment provided to the system. The system is autonomous. The only missing element is embodiment. This is still just a simulation.

This simulation will utilize the following subsumption flow chart with the addition of system detectable error.

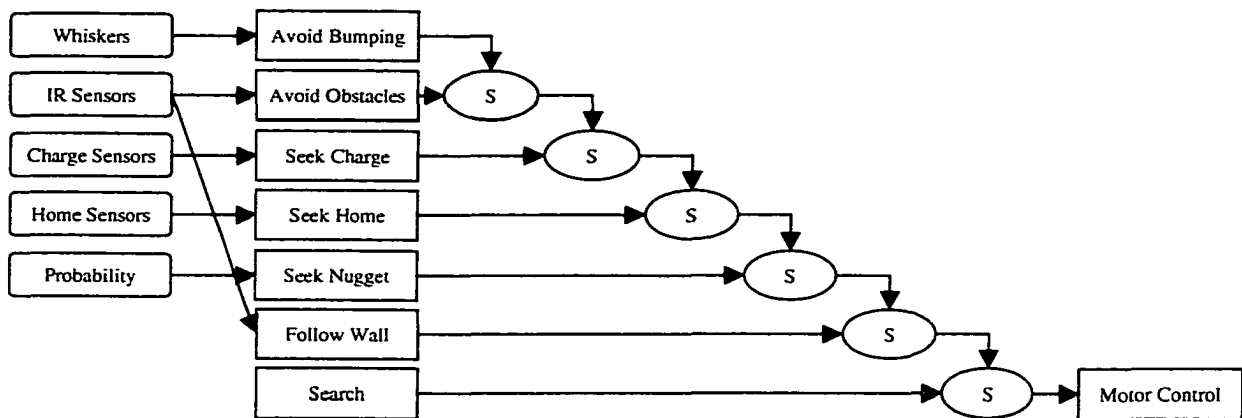


Figure 9.9, Three Behavior Subsumption Flow Chart

9.4.1 Three Behaviors – No Induced Faults

Notice that for all behaviors, Avoid Bumping, Avoid Obstacles and Follow Wall are active. For three behaviors, the IRA animat was again able to detect the trapped trajectory associated with circling the recharge station. This is shown in the figure below.

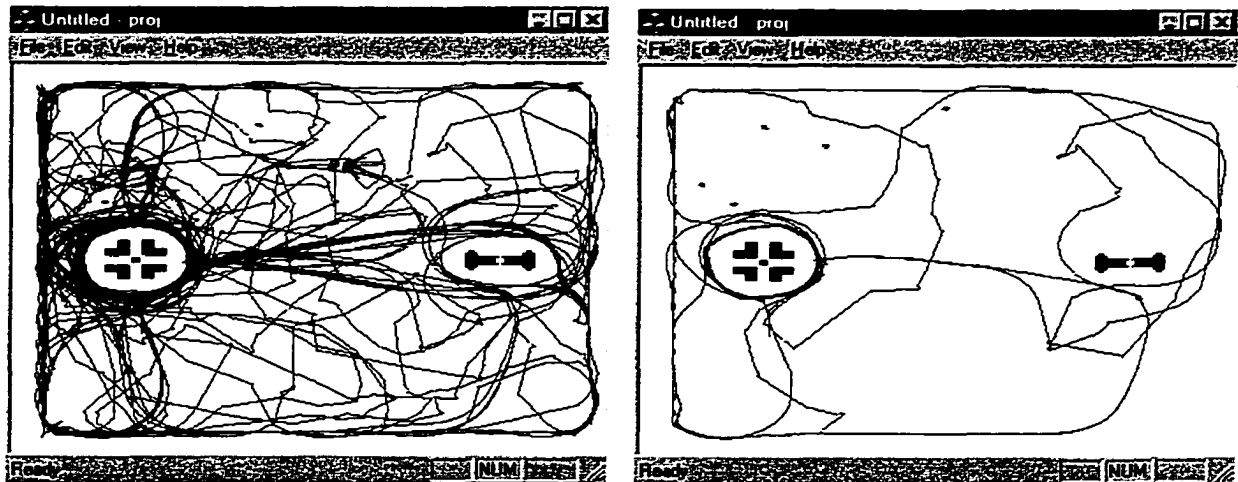


Figure 9.10, Three Behaviors, No Induced Errors

Again, the subsumption animat becomes trapped circling the charging station. **There are no sensory faults.** The subsumption animat is correctly responding to the correct sensory input. The result, however, is that the animat has lost its circular organization, it does not return to the Dock Charge reaction and it dies. The subsumption animat recharged three times and returned two nuggets before it got trapped and died. The IRA animat, on the other hand, recharged countless times, returned four nuggets and was still running at twenty plus minutes into the simulation when the simulation was halted. The following two spreadsheet plots show the trap that the subsumption animat became trapped in. It also shows the same trap, but in this case the IRA animat was able to get out and then reach the recharge station.

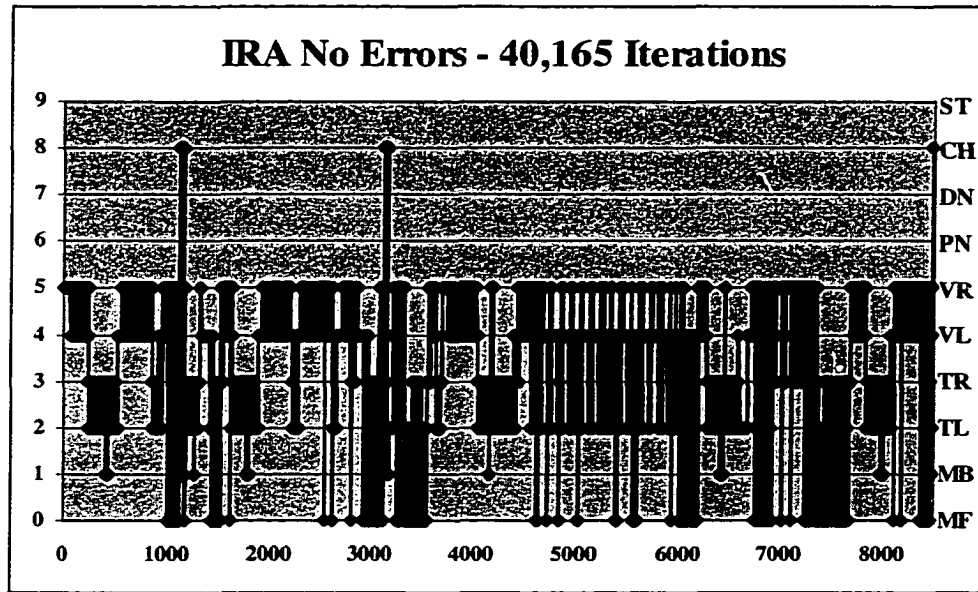


Figure 9.11, IRA Animat No Errors

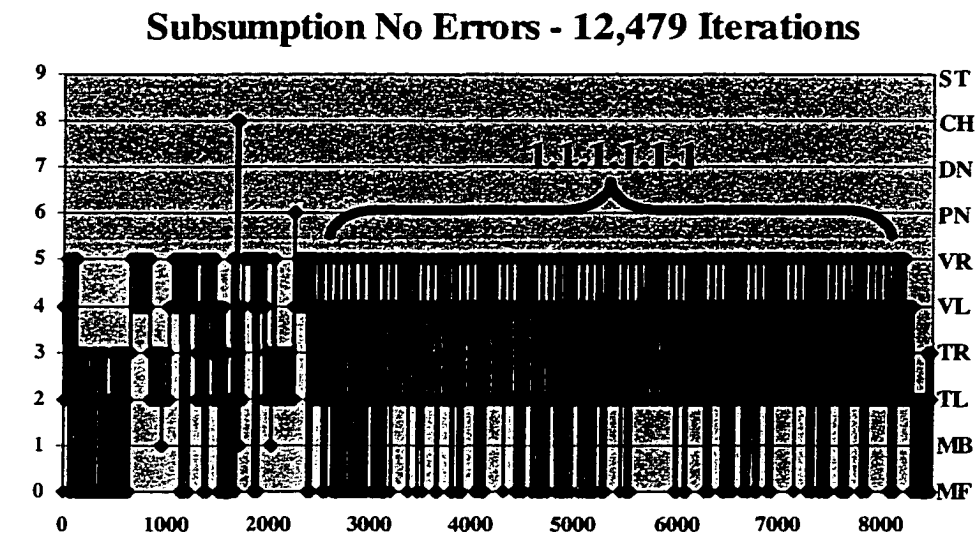


Figure 9.12, Subsumption No Errors

The IRA animat was still running at 40,165 iterations when the simulation was halted. The subsumption animat, with the same initial conditions and the same reactions ran for only 12,479 iterations when it died. The IRA animat gets caught in the same trap as the subsumption animat, it then rearranged its reactions to escape and recharge. Seek Charge

was never shut off, only the reaction, veer left due to left station sensor which was trapping the animat. A closer look at the trap is seen in the following figure.

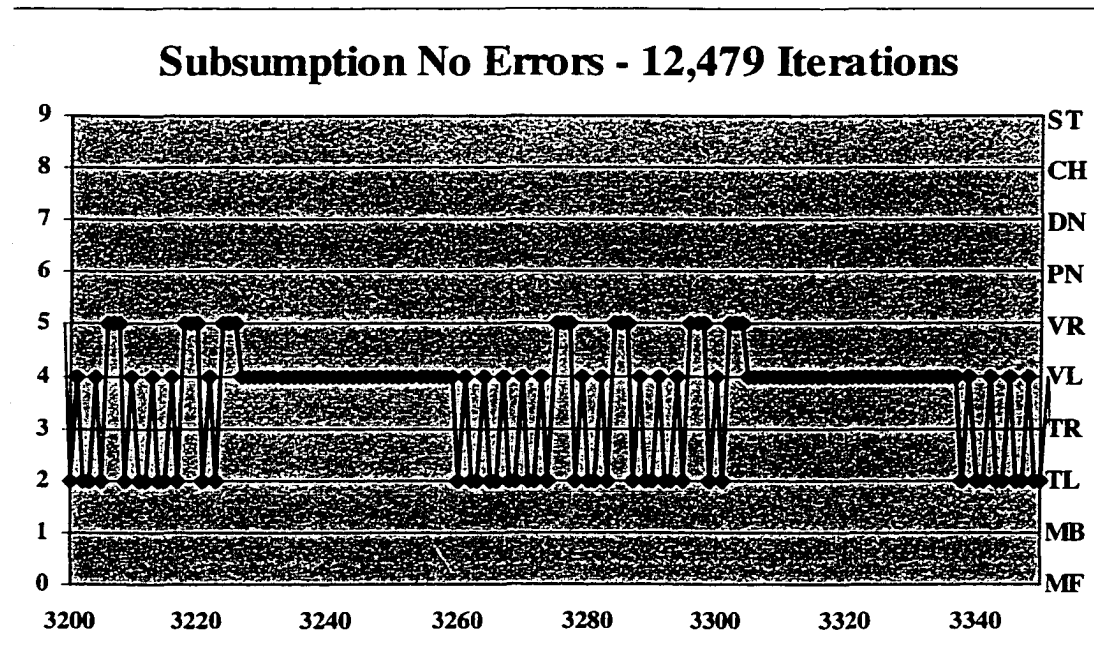


Figure 9.13, Close-up of Trap

In the above figure it is clear that all reactions are still active. The reaction inducing the trap is the left station sensor. The response to this sensor is to veer left. Other reactions cause the turn left and turn right. The repetitive nature of the response is picked up by the organization of the IRA animat and the response to veer left is suppressed. This allows the animat to try something else, move forward, etc. In Figure 9.10, the animat becomes snared in the same trap. It then attempts to escape somewhere between 6000 and 7000 iterations. It becomes trapped again, but then finally escapes and reaches the recharge station. No other form of error detection could possibly have picked up this error. There were no sensor errors to detect.

In an attempt to trap the animat, the world from above was modified to have concave trapping regions. In order to force the animats into the trapping regions, holes were cut in the blocks to allow the charge detectors to see the charging station. In effect, mimicking the case of an object between the animat and the goal. The figure below illustrates what happens to both animats.

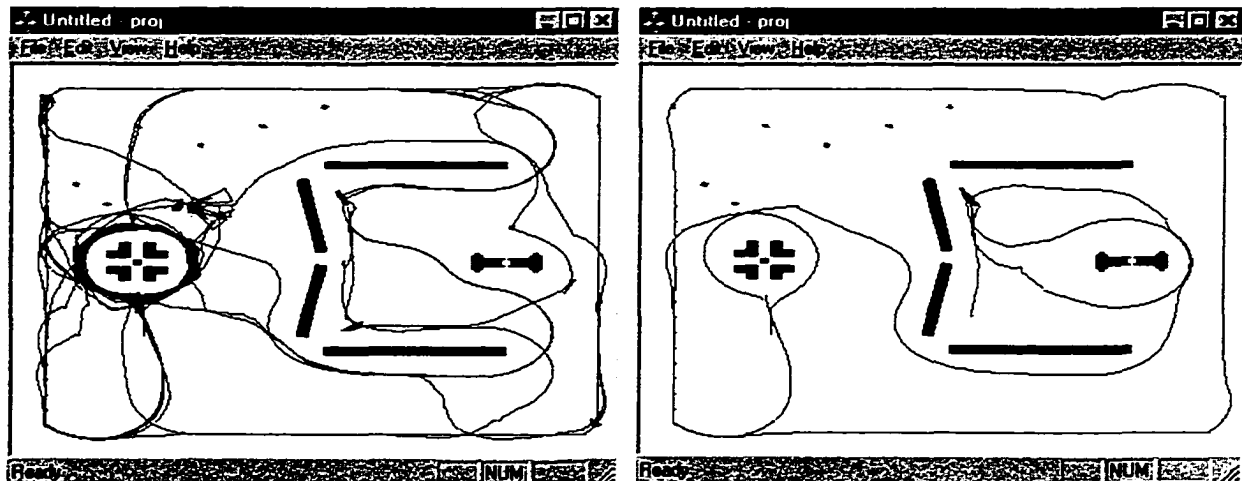


Figure 9.14, Three Behaviors – Trapping Region

The subsumption animat on the right was unable to escape from the trapping region. It became fatally stuck in the lower left-hand portion of the sigma shaped region. The animat was able to retrieve one nugget, but it was not able to return it to the home station. The animat was able to recharge once before it became trapped and died. The IRA animat was able to escape from the trapping region. It recharged better than ten times, found two nuggets and returned one to the home station. The simulation was halted at five plus minutes when it was clear that the trapping region would not trap the IRA animat. **There were no sensory faults.** The subsumption animats sensors were not in error and the animats reactions to the sensors were not in error, but it still lost the circularity of its organization and died.

9.4.2 Three Behaviors – With Induced Errors

For this demonstration, there are four induced faults: one whisker, one IR, one charge and one home detector are tripped in the “ON” position. Again, there is no functional redundancy built into Seek Charge or Seek Home. The Byzantine Generals Problem is not applicable. The following figure demonstrates what happens to both the IRA and subsumption animats.

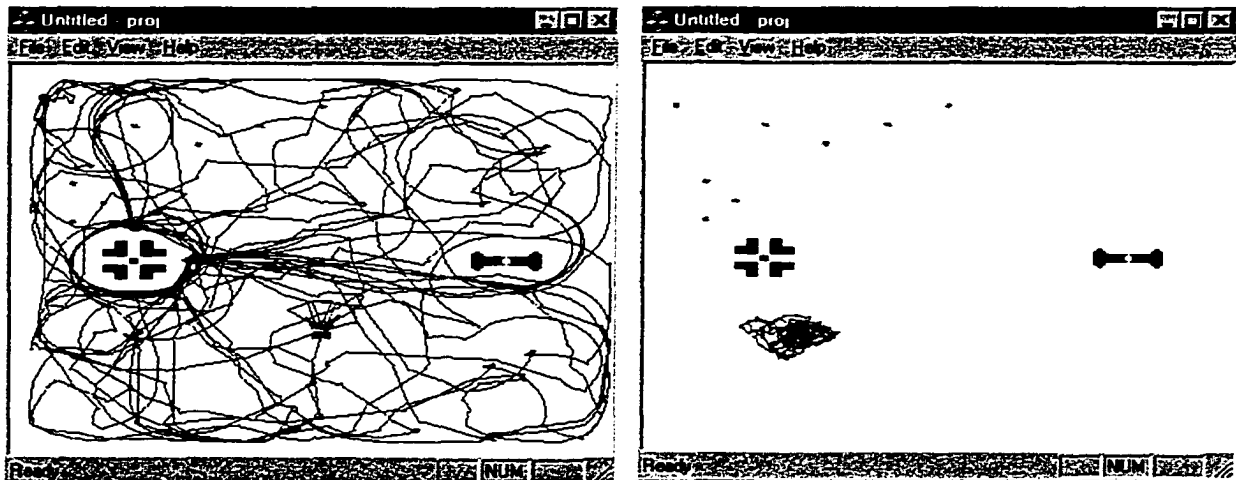


Figure 9.15, Three Behaviors – 31% Coverage

The subsumption animat is required to respond to the given input string. Avoid Bumping controls the animat until the animat dies. This figure clearly demonstrates the capabilities of the IRA to detect and tolerate faults. The animat is encumbered with a 31% reduction in sensory capability, yet it is able to stay alive. Seek Home utilizes only three sensors, by losing 33% of the sensory capability for this reaction, the IRA animat was unable to return the nugget that it found. This is not surprising. The Byzantine Generals Problem does not have a solution for three generals. The IRA animat had a great deal of trouble locating the home, and when it did, it was unable to approach in the required orientation to dock. The IRA animat, however, did not die. Therefore, the

animat is still autonomous, situated and self-sufficient. The primary goal of any system should be survival. (Atkins, 1997)

10 Design Guidelines

There has been a great deal of work presented that covers control architectures of robots. However, there has been no work that actually describes the necessary processes to designing and fabricating one of these systems. This research makes an initial contribution toward design support.

In the world of autonomous robotics there is a lot of *inferential knowledge* (Nowack, 1997), personal experience. There are groups of designers creating autonomous robots using many different control paradigms and mobilization techniques. There are the BEAM robots of Mark Tilden. There are the behavior-based systems developed by Brooks and others. There are countless planning-based systems. The actual systems are all produced by the inferential knowledge of the designers. There is almost no *static knowledge* (Nowack, 1997), that is text book data that details a methodology for the development of a robotic system, whether behavior-based or planning-based. There is *static knowledge* for the structure of the system to be designed, but none for the organization of the system. There are no guidelines telling the designer what needs to be done to produce a functional system. The following discussion will detail a design methodology in chronological/hierarchical order. “Design principles and guidelines are most valuable when making the transition from the functions to be fulfilled to the chosen physical embodiment.” (Nowack, 1997) In the case of IRA-based minimalist robots, such transitions exist in the progression from basis objectives to the

IRA organization and from the completed organization to the physically embodied robot. Emphasis here will be on the former situation.

10.1 The Organization

Nehmzow (2000) notes there is no such thing as a general purpose being, so there cannot be a general purpose robot. Maturana also agrees with this statement. He talked about how the structure of a system can be realized in many different ways, but the organization of the system is what describes the functionality of the system. These statements also hold true in the design world. A design is derived from the requirements, not the reverse, since a design is a mapping from one to many. (Nowack, 1997) To this end, the designer must know what the system is to be; he must have the requirements. The requirements will form the organization. This is the first and most important step. The organization is independent of the structure. The organization is independent of the control methodology. Maturana speaks of a living system, but it is directly applicable to this work. “The closed nature of the functional organization of the nervous system is a consequence of the self-referring domain of interactions of the living organization; every change of state of the organism must bring forth another change of state, and so on recursively, always maintaining its basic circularity.” (Maturana, 1980) The behavior-based approaches have the best opportunity to realize this required circularity; therefore, the focus of this work will be on behavior-based systems.

As was stated previously, the organization of the system determines the dynamics of the interactions and transformations that the machine can undergo with the environment. (Maturana, 1980) This is where the designer must spend a great deal of time. What then are the dynamics of interactions and transformations for the system

being designed? The function of a machine is not a function of the organization, it is, however, dependent on the environment the machine is placed in. (Maturana, 1980) Machines are not general purpose. Simply maintaining the circularity of the organization will not necessarily make the machine useful. The machine will only be useful if the organization and the environment are tightly matched. The designer must know how, where and under what circumstances the machine/system will be used.

Each component of the organization can be represented as a finite state machine with several sub machines. There could be just one finite state machine, representing a system with only one behavior. There could be several finite state machines, representing a system that possesses multiple behaviors. In this step, the designer must formulate the organization of the system in order to match the organization; the environment and the task.

The system is no longer a separate entity. The system is the machine, the environment and task. All three things have to be present to have meaning. Therefore, the design process must include the organization, the environment and the task to adequately capture the system. This coupling can be represented by the following figure.

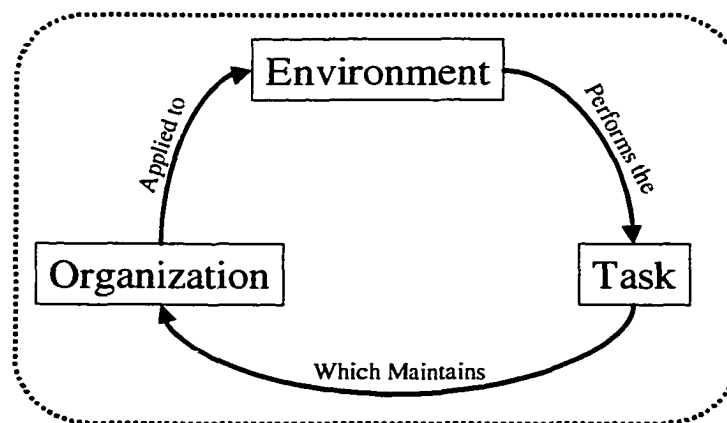


Figure 10.1, The System

10.2 The Structure

Once it is determined what the organization for the machine will be, the designer can then focus on the structure. The design of the structure will also depend upon the environment where the machine is to be put to use and upon the *inferential and static knowledge* of the designer. The structure will evolve from the components of the organization. Physically, how will the designer produce the interactions of the system with the environment that will perform a task while maintaining the organization? There are countless ways that this matching can take place, it will be up to the designer. Here is where the designer will determine the number of sensors to be used for each reaction and the total number of skills that the system will possess. This will lead to the embodiment of the system. How many sensors does each submachine require to perform its task? How will control of the machine be executed? How many actuators does the machine require? These questions will be answered here. One method is to use the Action-Centered Design Model illustrated below. (Nowack, 1997)

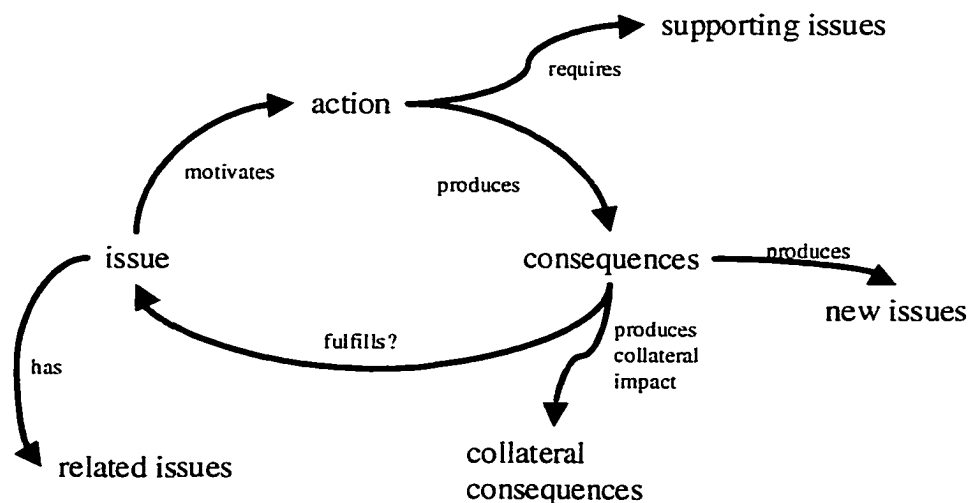


Figure 10.2, Action-Centered Design Model

Each submachine will constitute an issue that the designer will have to satisfy with some action. The actions may or may not satisfy the issue, but will definitely lead to consequences. These consequences will produce new issues and possibly collateral consequences.

The structure requires the designer to perform the task of matching input strings to output strings. This is critical. If the designer were to omit a possible input string, the system would not have a matching output; it would not do anything for the given input. For a dynamic environment, the omission of an input string may not be debilitating to the system, but in a moderately static environment; most systems would become stuck. The best way to avoid the omission of an input-output match is to create the transition matrix for the system. See Gill (1962) for a full development of the methods to forming transition matrices. Studying the transition matrix will allow the designer to determine what are the trapping trajectories. A trapped trajectory will break the circularity of the organization; this must not be allowed to happen. As was discussed in the organization section, the structure must be subordinated to the maintenance of the organization. This subordination is the key **new issue** produced by each submachine. Each reaction must be empowered to detect the trapped trajectories in order to maintain the organization. Once detected, the reaction that is causing the trapped trajectory will shut itself off, there will be no external observer. There is no plan for how to deal with the trapped trajectory, just to try another reaction and ignore the one that is creating the trap. This allows the system to possess system detectable error, and closely resemble an autopoietic system. This new issue is also based only on an internal criterion of loss of circularity, it is not based upon

an external representation of the environment. The design guidelines are summarized below.

1. Begin the Design by determining the requirements
 - 1.1 The first requirement to determine is the fundamental purpose of the robot.
 - 1.2 After the fundamental purpose is determined, identify the secondary requirements.
 - 1.3 Determine the intended environment for the robot.
2. Form the organization from the requirements
 - 2.1 To ensure circularity in the system, use a behavior-based organization
3. To ensure a useful machine, take great care to match the machine and the environment.
 - 3.1 To match the machine to the environment, consider each functional component of the organization as a finite state machine.
 - 3.2 Due to the dynamic nature of the environment, functional redundancy should be used to the greatest extent possible.
 - 3.3 Performance of a task must maintain the circularity of the organization.
4. Determine the type of sensor that will complement the task and the environment for each of the finite state machines.
 - 4.1 Determine what constitutes “ON” for the sensor.
 - 4.2 “ON” is a match between the environment and the sensor.
 - 4.3 Determine the number of sensors required to perform the task of the finite state machine.
5. Determine the type of actuators that will complement the task and the environment for each of the finite state machines.
 - 5.1 Power requirements are extremely important when dealing with autonomous mobile robots.
6. To maintain the circularity of the organization, all input strings must have an output string.
 - 6.1 To ensure an input-output match, construct a transition matrix.
 - 6.2 The transition matrix will identify possible trapped trajectories for sticking sensors.
7. The IRA is a bottoms-up approach, begin programming the finite state machines at the lowest level.
 - 7.1 Build each finite state machine by stacking each level on top of the lower level.
8. Determine what constitutes a trap for each of the reactions.
 - 8.1 Error is determined internally and is based on the operation of the control loop.
 - 8.2 Error is determined at each of the reaction activation levels.
 - 8.3 Expect error to be different for every reaction.
9. The robot must be tested in its environment.
 - 9.1 Ensure sensor “ON” determination is correct.
 - 9.2 Ensure error “ON” determination is correct.
 - 9.3 Ensure actuators complement the environment.

10.3 Implementation

The discussion above laid out a method to design a system using finite state machines that possesses system detectable error. The best way to fully describe the method is with a robotic example. This research requires the use of a minimalist robotic system. There are several available for purchase, but use of the design process is more useful. To enable comparison, the minimalist system will be designed to mimic the animats of the AnSim simulations as closely as possible. The system will have the same number of sensors and the sensors will be placed in the same locations as on the simulations.

10.3.1 Organization

In the interest of minimalism and simplicity, the robot will have a very simple organization. The system will avoid bumping into objects, it will try to avoid obstacles and it will explore its environment. Basically, the machine will survive. The circularity of its organization will point from avoid bumping to avoid obstacles to explore. Every control loop will go through this process in a circular fashion. Avoid bumping, avoid obstacles and explore all represent submachines of the single behavior, explore while surviving. This organization will only be self-maintaining in a flat environment. There is nothing in the organization to detect, predict, or cope with pits. The organization has no concept of a hole. At this point, there has been no discussion of structure, skills, sensors or actuators. The machine, however, has been limited to relatively flat environments with no pits, and the machine will possess three submachines.

10.3.2 Structure

See Appendix G for a discussion of current minimalist micro-robots. The structure will be the physical implementation of each of these submachines. For avoid bumping, two whisker type sensors will be used. These whiskers will also have a bumper function for things that could fit directly between the two whiskers. This will give the system full frontal coverage for bumping into objects. For avoid obstacles, a distance type sensor is required. For this demonstration, photoresistors will be used. This type of sensor was chosen for several reasons. Photoresistors are very sensitive to the environment, allowing for a good opportunity to produce and observe errors. This system will use a Basic Stamp II microcontroller. It does not come with an A/D converter, but it can read photoresistors without the use of A/D conversion. For full frontal, obstacle avoidance the system will require three photoresistors. Explore will be random forward motion biased toward straight ahead. These are the connections between the submachines and the environment.

The submachines must also connect to the physical robot. The robot is a platform that has two motors, one driving each of the rear wheels. This is the most common method of directional control used in minimalist robotics. In Appendix G you will note all of the robots use two motors, except for Squirt which can turn right by backing up. The robot has a pin in the front, so the environment will have to be restricted to hard surfaces. Control is obtained with four inputs. There is a forward and reverse input and there are two inputs to stop each of the motors. These inputs can be combined in any fashion for the robot to possess the following skills: forward, back, forward right, forward left, back right, back left and stop. The figure below is a picture of the minimalist robot.

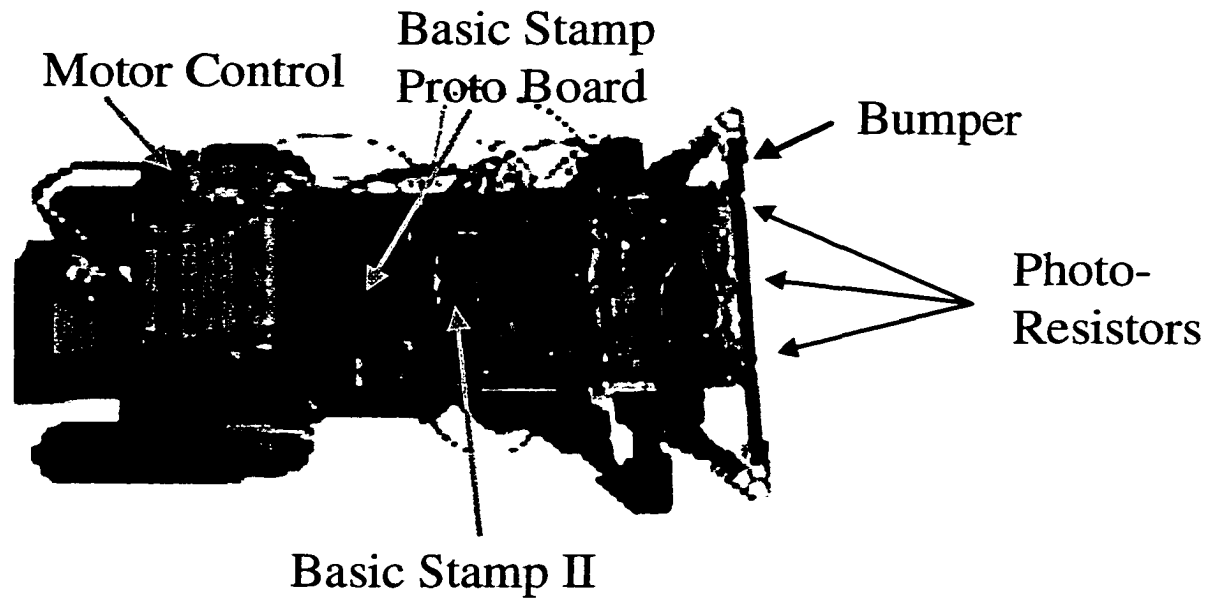


Figure 10.3, Minimalist Robot

The submachines must be linked to the skills of the robot. This is accomplished by creating the transition matrix for the robot. This robot will have six entries for the input string. There are the two whiskers/bumpers; there are three photoresistors and the pseudo-random number generator. The transition matrix is given below. For this demonstration, the system will rely on the mismatch between the two motors. This will cause the robot to not track a perfectly straight line. It will also cause the robot to back at some angle, hopefully allowing the robot to clear the obstacle. Of course, when actually implemented this may not be enough, causing one of the **collateral consequences**. The minimalist system is programmed with the same reactions as the similar AnSim simulation. In the table, F stands for forward, B for back, R for right and L for left.

String	RW	LW	RIR	MIR	LIR	RN	Output
1	1	1	X	X	X	X	B
2	1	0	X	X	X	X	BL
3	0	1	X	X	X	X	BR
4	0	0	1	1	1	X	B
5	0	0	1	1	0	X	BL
6	0	0	1	0	1	X	B
7	0	0	1	0	0	X	FL
8	0	0	0	1	1	X	BR
9	0	0	0	1	0	X	B
10	0	0	0	0	1	X	FR
11	0	0	0	0	0	1	F
12	0	0	0	0	0	0	FR
13	0	0	0	0	0	-1	FL

Table 10.1, Demo Transition Matrix

The basic organization of this machine will be subsumption. Input strings 1 – 3 refer to avoid bumping. Input strings 4 – 10 refer to avoid obstacles and input strings 11 – 13 refer to explore. Avoid bumping is the highest level of submachine, followed by avoid obstacles and finally explore. This machine has functional redundancy provided by the whiskers and the photoresistors. Functional redundancy will allow the machine to handle the errors associated with a false negative, i.e. not detecting an object. The false positives will trap the machine in one of the skills. In order for the organization to be invariant, the machine must detect that it is trapped in one of the skills and then escape from that skill. Each submachine will be equipped with memory to detect how many times it has been called consecutively. If an individual reaction has been called too many times, resulting in a trap, that individual reaction will shut itself off and allow the other reactions to take over. This will insure that the organization remains circular. An entire submachine is never shut off. The only way for an entire submachine to be shut off is for each reaction within the submachine to be considered trapping the system. How many times, will be dependent on the execution of the control loop and will have to be determined

experimentally. How many times will also vary from submachine to submachine. See Appendix C for the Basic Stamp II control code for this demonstration. You will note that the time required to deactivate a reaction is different for every reaction. This is a fusion between the sensor the environment and the task. The sensors, especially the photoresistors are all different, therefore, the amount of time to represent a trap is also different. The photoresistors and the bumpers are used to insure that the robot keeps moving. It allows for avoid obstacles and getting around the ones that it hits. The fusion is a process of determining how many control steps are considered a trap. This is an internal representation of error; it is not based on an explicit representation of the environment or of the task. It is internal because error is based on behavior, and the behavior is based on the control process. Some reactions require more physical time to run than other reactions. In this case, the bumpers are simply on or off. There is no testing required. The photoresistors on the other hand, must time the decay rate of the capacitor. This takes much more time than simply checking to see if the bumper is high or low. For the bumper, it is a very short time, just five control steps. For the photoresistors it is much more time, between 30 and 40 control steps.

Also note in Appendix C that there is a slight divergence from the classical subsumption architecture. The arbitration scheme is performed before calculating what each reaction should do to control the robot. The order of execution is to check for reaction activation, perform the arbitration and then determine what the animat action the animat should take. Classical subsumption would check for activation, calculate what each reaction should do and then perform the arbitration. This reordering eliminates the

need to determine what each reaction should do for each step of the control loop. It only determines what the reaction should do that is in control which saves processor time.

10.3.3 Lessons Learned

The purpose in building this minimalist robot rather than simply purchasing an off-the-shelf platform was the learning process. The first step was the development of the mobile platform. In the interest of saving time and money a “full function” remote control car was purchased. The car cost \$10 at KayBee Toys. The car functioned by the use of two switches. One switch did forward and reverse and the other switch would stop either the right or left motor. The microcontroller would then have to replace these two switches. The Basic Stamp II could not source enough power to actually run the motors so transistors were used as the switches connecting the microcontroller to the platform. The total cost of the platform was not more than \$15 for all parts.

The platform then needed to be coupled with sensor suites. The key point to photoresistors is that they are all not created equal. Sensitivity is what the system needs, to this end each photoresistor was tested to insure maximum sensitivity. Cadmium-sulfide photo resistors from RadioShack were used. They came in an assortment of five. The largest (most sensitive) photoresistor was chosen. In four packages, the ambient resistant ranged from $\approx 300\ \Omega$ to $\approx 900\ \Omega$. The three with the most sensitivity were used on the robot. For the Basic Stamp II to read the photo resistors without an A/D converter, the photo resistors needed an additional resistor and a capacitor. A $249\ \Omega$ precision resistor and a $0.1\ \mu\text{F}$ capacitor were used. The basic stamp can then read a photoresistor by checking the charge or discharge time for the capacitor.

The bumper required two switches, one right and one left. The Basic Stamp II has an interesting quirk. If there is no input to one of the I/O pins, the state of the pin will arbitrarily bounce from high to low. Therefore, a SPDT switch is required. In the interest of keeping things as small as possible a sub-miniature SPST switch was used and the I/O pins were grounded when the switch was in the open position, creating the second throw. See the table below for a complete parts list.

Part	Part Number	Quantity	Unit Cost (\$)	Cost (\$)
Toy Car		1	9.95	9.95
5 Vdc Relays	900-2346	4	1.72	6.88
Resistors – Misc	900-0xxx	6	0.07	0.42
Capacitors – .1 μ F	900-2165	3	0.20	0.60
Transistors	900-5428	4	0.29	1.16
Photocell Asst	276-1657	3	2.29	6.87
SPST switch	900-7144	2	1.29	2.58
Basic Stamp II	900-3268	1	49.00	49.00
BSII Proto Board	900-3269	1	20.00	20.00
Circuit Boards	910-3803	3	1.49	4.47
			Total	101.93

Table 10.2, Minimalist Robot Parts List

The part numbers listed are all Radio Shack part numbers; similar parts could be obtained from other sources. The Photocell Assortment was the only way Radio Shack sold the photocells; they could not be purchased individually.

10.3.4 Performance Results

The goal of this research is define error and a way of recovering from that error. To avoid all extraneous sources of possible error the minimalist system will be placed in a constrained environment. The environment will mimic the simulated world of AnSim to enable a comparison of results. The robot was programmed to mimic the animat in AnSim. It was not identical, since the minimalist robot uses a bumper and not whiskers. There is also a discrepancy due to the fact that the minimalist robot does not possess the

capability to veer. Three experiments were performed on the minimalist robot for both the IRA and Subsumption. The animats were both programmed for five hundred iterations of their respective control loops, approximately three minutes of exploration. The first was no induced errors, the second was the case of a stuck photoresistor and the third was the case of the stuck bumper. The results are presented below. Videotape of all experiments was taken. This videotape was then imported into the computer through the use of a Studio PCTV video card for further analysis.

In order to insure that all of the submachines were functioning properly, an LED was added to the front of the robot. The LED would only become lit if the IRA had detected a possible trapped state. This gave a visual key to the designer to insure that the photoresistors were not continuously tripped, or the bumper was not functioning for the no error situations. It also insured that the system was performing properly for the induced error situations.

10.3.4.1 NO INDUCED FAULTS

The subsumption animat was able to explore most of the environment, but became trapped in a dark corner of the environment. The corner although not drastically darker than the rest of the environment was dark enough to trip all of the photoresistors. When this happened, the subsumption animat is programmed to back up and turn. It would then back and turn itself into the corner to where it could not back and turn anymore, effectively trapping the animat in the corner. When the animat's starting location was moved the animat would not enter the darker region, it remained in the upper portion of its environment. During another run of the subsumption animat the bumper was knocked off of its base, causing the bumper to produce the false positive.

This caused the animat to back and turn until it reached maximum iterations, effectively resulting in the death of the animat. It was no longer able search the environment. The circularity of its organization was broken. The exact same situation as was shown in Figure 3.4, with the simulated subsumption animat. Due to the faults caused in the bumper and photoresistors the subsumption animat was never able to explore the entire environment without becoming trapped.

The IRA animat performed very well when there were no induced faults. The animat was able to explore the entire environment. It also preferred the lighter region, but eventually ventured into the dark region where it was still able to perform its main task. There were several instances of a bumper fault as noted for the subsumption animat. When this occurred, the animat would naturally respond to the input string. It would begin to back and turn as if there was something in front of it. Once it realized that it was trapped, the order of the reactions was changed due to one reaction being turned off and the animat would then continue to explore its environment. Without the use of this sensor, the animat would usually hit the wall of the environment, which would reseal the bumper, and normal operation would again continue. The animat never became trapped in any situation, either with the false positive from the bumper or from the false positives from the photoresistors.

10.3.4.2 PHOTORESISTOR FAULT

For this demonstration, the right photoresistor was tripped in the “ON” position. This was accomplished by placing a black object in front of the right photoresistor. The subsumption animat is required to respond to the input string it is provided. The animat immediately went into a left-hand turn and stayed in the left-hand turn until maximum

iterations was reached, exactly the same as in Figure 3.5. It was clear that the other reactions were still active. During the left-hand turn, the animat approached the wall; this caused the middle photoresistor to also become tripped. When this happened the animat moved away from the wall and then back into the left-hand turn. Again, this is exactly as in Figure 3.5 when the animat got too close to the recharge station. The simulated world and the real world had the same results, which provides proof that the use of the simulated world and the results from the simulated world are applicable to the real world.

The IRA animat performed with no noticeable impairment due to the tripped photoresistor. The middle photoresistor, as mentioned previously, is much more sensitive and has a much greater effective range than the right and left photoresistors, this allowed the animat to pretty much run on the middle photoresistor alone. The only thing the animat could not do was following the wall on the left side. It did not become trapped in a left-hand turn. It did not become trapped in a corner or against a wall. **There was virtually no perceivable difference between the case of no induced errors and the case of a tripped photoresistor.**

10.3.4.3 BUMPER FAULT

For this demonstration, the left bumper was tripped in the “ON” position. The results were again exactly as predicted by the simulated animat in Figure 3.4. The real animat immediately began to back and turn to the left; it did this until the maximum iterations were reached.

The IRA animat performed well under the tripped bumper as well. It entered into the trapped trajectory and then rearranged the reactions to move away from this trapped trajectory. It then went about exploring its environment. The loss of the left bumper did

present some troubles for the animat. As long as it did not hit a wall with the left bumper, the animat had no trouble exploring its environment and to keep moving. If it did hit with the left bumper, the animat may then become stuck. Depending on the inputs from the photoresistors. If all of the photoresistors were also tripped, the animat would back away from the wall. If only one or two were tripped, the animat would try to turn away from the wall. The motors were not strong enough to slide the animat off of the wall. Better traction or more powerful motors would have easily fixed this problem.

10.3.4.4 OVERALL PERFORMANCE RESULT

The simulated animat and the real animat performed qualitatively exactly the same under the same situations for induced faults. This means that any information gathered or conclusion drawn from the simulated world is applicable to the real world. The bumper becoming knocked off of its base manifested the advantage of the real world. The real subsumption animat became trapped in the back and turn trajectory and was not able to extricate itself. It stayed in the trapped trajectory until the maximum iterations were reached. The real IRA animat also was able to knock the bumper off of its base. It would then enter the trapped trajectory, but was able to move away from it. The loss of this sensor would then cause the animat to strike the wall. This would reseal the bumper and normal operation would continue. This demonstrates the power of the IRA. There is no attempt to declare the sensor in error and therefore dysfunctional, only that the system is trapped and to try something else. When the bumper began to work properly, it was again incorporated into the control loop. This intermittent failure was then “diagnosed” and “handled” without the use of an external observer or by resorting to complex plans.

The simulated world and the real world showed that the IRA could handle the case of a stuck or disconnected sensor. The reason for using the real world was shown by the ability of the real IRA animat to detect and handle an intermittent stuck or disconnected sensor. Appendix C shows that the error detection is an integral part of reaction activation which is internal to the control program. There is no external observer, there is no plan, there is no attempt to represent the environment and there is no reduction in speed.

11 Redundant Measurement of a Vibrating Beam

The techniques presented in the previous chapters are not only applicable to minimalist robotics. The same procedures can be used for other types of redundant sensing devices. The behavior-based approach allows for extremely rapid determination of the sensed quantity and the use of interactive representation allows for the rapid determination of error allowing for the rearranging of reactions. Applying the behavior-based approach of direct links between the sensor and action and the inclusion of the behavior-based error detection allows the designer to select the sensors that are behaving appropriately and which sensors are not behaving appropriately. This decision can then enable the system to choose the appropriate reaction for the given situation.

To demonstrate the applicability of the techniques developed in this research a simple experiment was performed, both in a computer simulation and through an actual experimental procedure.

The tip deflection of a simple cantilever beam can be determined through the use of a single strain gauge placed virtually anywhere along the length of the beam. Through calibration of the transducer, an extremely accurate measurement device can be created. To improve the accuracy of the transducer, multiple gauges in multiple locations along the length of the beam can be used. This results in a transducer with redundant sensors. Any one of the gauges can be used to determine the tip deflection, but the more gauges available for the calculation, the more accurate the overall result. The problem with this method is that strain gauges are susceptible to error sources ranging from temperature to

magnetic fields. Strain gauges are also susceptible to failure due to poor soldering techniques, improper handling and disconnected wires. These types of failures almost always manifest themselves as a stuck or disconnected sensor. Using the same techniques discussed for the minimalist robot, this system will be able to determine that it is trapped and use one of its other reactions to calculate the tip deflection. The first demonstration uses a *Mathematica* simulation and the second uses an aluminum beam and *LabView*. For both demonstrations, the modeled beam will resemble the figure below.

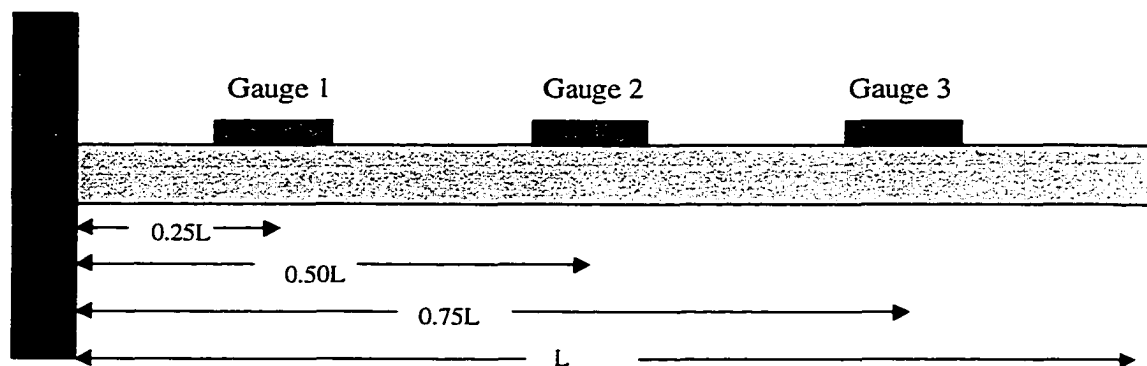


Figure 11.1, Beam Model

11.1 *Mathematica* Beam Deflection Simulation

Before actual experimentation, it is frequently more efficient to run a simulation to ensure that the method proposed will function for the application in question. The use of three strain gauges in conjunction with the fact that the root of the beam is not allowed to displace provides four points on the beam. These four points can then be used to determine a cubic curve for the deflection of the beam. The resulting equation can then be used to extrapolate the tip deflection of the beam. This is simple structural mechanics; this paper will not reproduce the development here. For the full development, refer to the

complete *Mathematica* simulation in Appendix E. To more accurately represent the effects of a strain gauge, each simulated measurement was incorporated with a random 0.01% error. The tip deflection of the beam is driven by a sinusoidal function. If all of the gauges are working properly, the result is shown below.

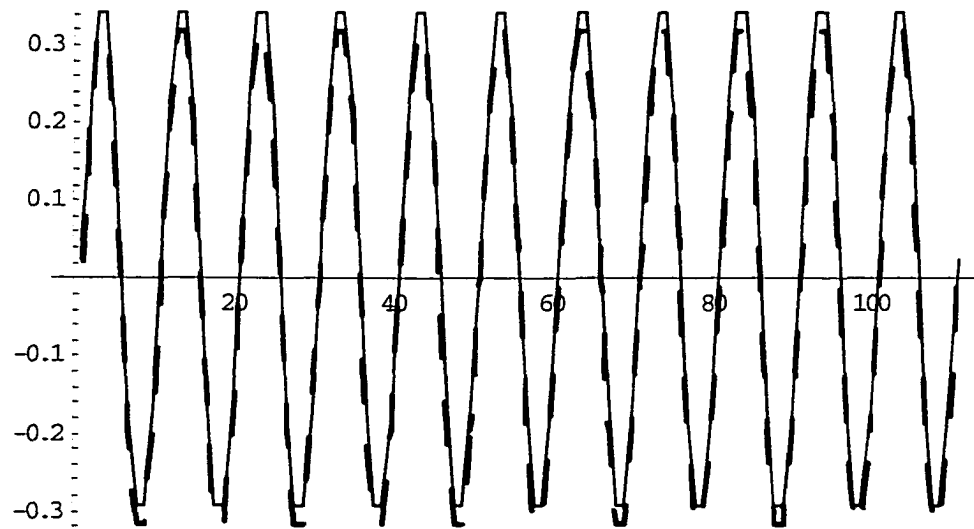


Figure 11.2, Three Good Gauges

The dashed line represents the actual tip deflection as determined by the input and the solid line represents the calculated tip deflection. This is very nearly a perfect fit. The random noise inherent in the strain measurements accounts for the slight variations.

What would happen if one of the strain gauges were to stop functioning? This would be the case of the gauge sticking, which is very possible for a broken connection, the wheatstone bridge would go to a full-scale reading and remain there. It is also the case for the gauge becoming detached from the surface of the beam. For the instance demonstrated by Figure 11.3, the first gauge works half of the time and then becomes disconnected; at that point it goes to full scale, resulting in a reading of 0.08.

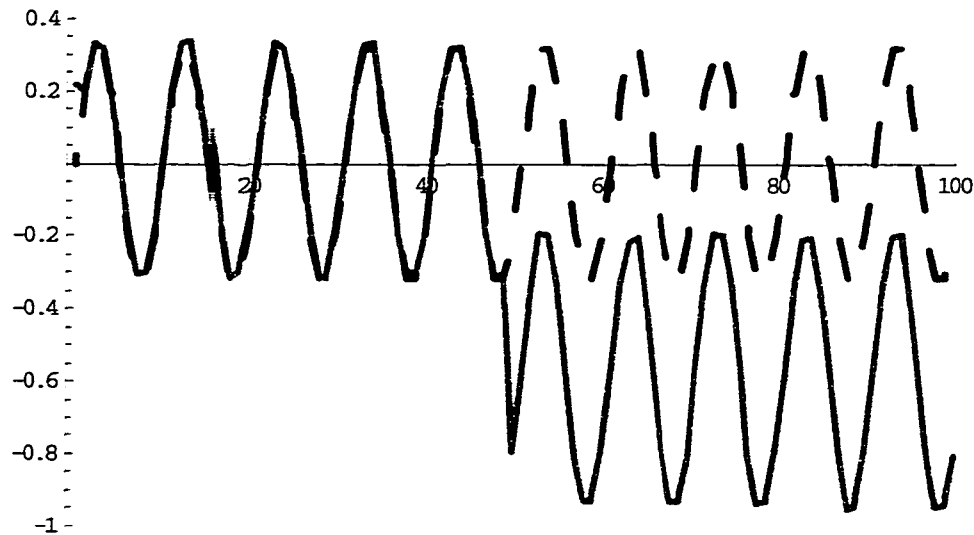


Figure 11.3, Gauge One Fails

The dashed line is the actual tip deflection and the solid line is the result of using the output of three gauges with one faulty to determine the tip deflection. This would be disastrous if the result of the tip deflection were to be used in a control algorithm. Past solutions would involve comparing each of the sensors together to ensure that their readings coincided with each other. The problem with this method was demonstrated with the Byzantine Generals Problem. There must be at least three gauges and only one can be faulty. (Gupta, 1999) What if two are faulty and one is correct? The process of comparing and voting would not work. The comparison process would throw out the good gauge and retain the two bad gauges.

To solve this problem, the tip deflection of the beam is the high level behavior for the simulation. Achievement of this behavior can be accomplished by any of seven different reactions: all good, 1bad, 2bad, 3bad, 1&2bad, 1&3bad and 2&3bad. Determination of which reaction to use is performed exactly as it was for the minimalist robot. Sensors are never declared to be in error, only that their output is trapping the

system, so the reaction that is using the trapping sensor will be ignored and another reaction will be chosen to calculate the tip deflection. The designer must be aware of what constitutes a trap for the system. For the demonstration illustrated in Figure 11.4, the beam will be vibrating or stationary. Then all of the gauges should be changing or they all should be stationary, but they are never compared to each other. The behavior of the sensor will determine its usage. If the sensor is changing, then that sensor is good. If the sensor is not changing, then that sensor will not be used unless the progression leads to the base reaction that uses all of the strain gauges in a no change situation. The result of one bad gauge is depicted below. Circularity will be maintained by forcing the calculation to run through all of the reactions before it declares a 'no change' situation.

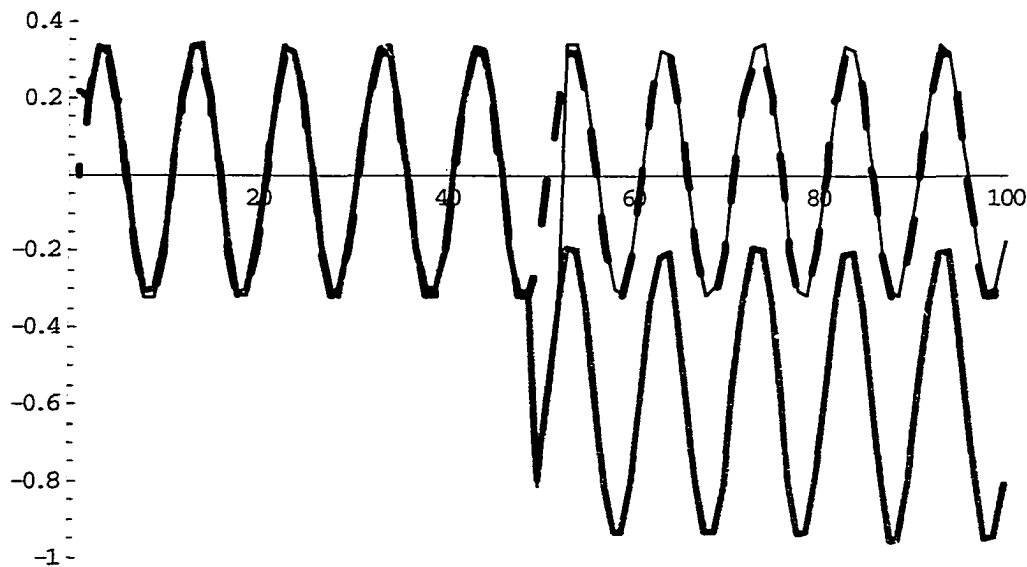


Figure 11.4, One Faulty, With and Without Error Correction

The thin solid line is the result of determining that there is likely a problem with strain Gauge Number One, the system then ignores the reaction that uses all three strain gauges and uses only the reaction that uses Gauges Two and Three for determining the tip deflection. The system determines that it is trapped and then rearranges the reactions to

determine tip deflection. This result is a near perfect fit, except for the blip (trap) that caused it to rearrange its reactions.

Any of the other error detection and correction techniques discussed previously could have handled this situation. They would have compared the output of Gauge One with the outputs from Gauges Two and Three. It would have been apparent that the output from Gauge One was not plausible when compared to Gauges Two and Three, Gauge One would then be ignored and Gauges Two and Three would be used. However, the comparison method could not handle the loss of two gauges as shown below. For this demonstration, Gauges One and Two will function half of the time and then they will both give a full-scale reading.

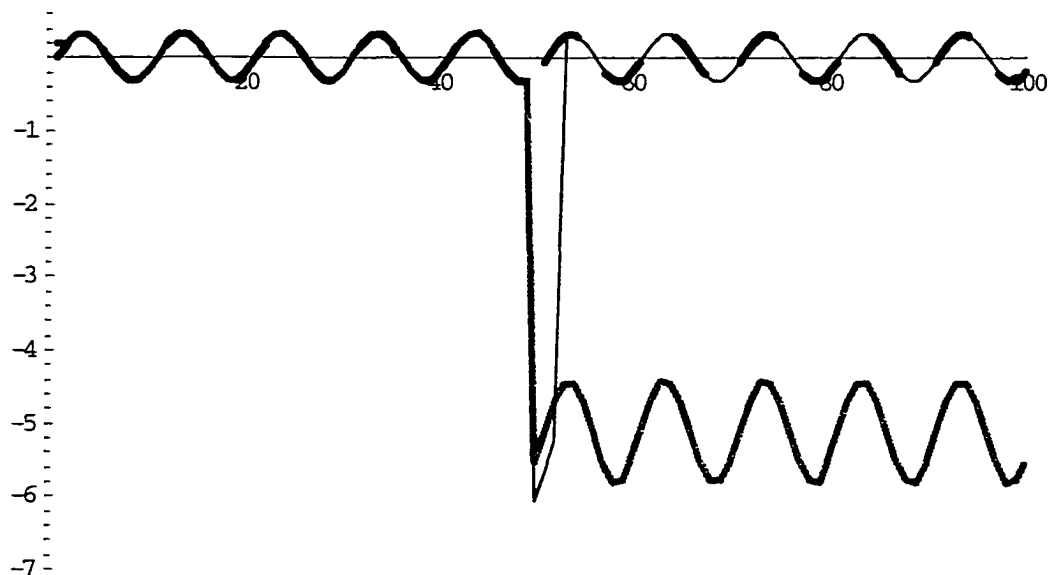


Figure 11.5, Two Faulty, With and Without Error Correction

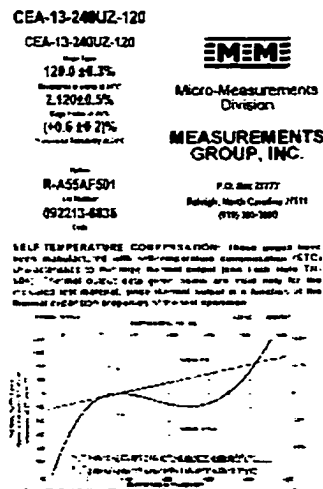
The plots are the same as in the previous charts. There is the blip associated with the two gauges going full-scale, the system detects this and then rearranges the reactions to determine the tip deflection using only Gauge Three. Gauge One and Gauge Two are

never declared to be in error, they are just determined to not be behaving as they should. This causes the control program to select the reaction that only uses Gauge Three to determine the tip deflection.

This is just a simulation. To verify the functionality of this algorithm, an actual experiment is required. Real world applications allow for fine tuning the algorithm and ensuring functionality. Just because something works in a simulated world, does not necessarily mean it will work in real world.

11.2 LabView Beam Deflection Experiment

To verify the algorithm, a LabView experiment was performed at the United States Air Force Academy. A test fixture similar to that depicted in Figure 39 was used. The beam was made of 6061-T6511 aluminum, 1.508" W x .256" T x 22.375" L. Three Micro Measurement linear aluminum strain gauges (CEA-13-240UZ-120) were installed on the beam in accordance with Figure 11.1. The strain gauges have the following characteristics:



The experimental setup consisted of a complete National Instruments experimentation and control station consisting of the equipment listed Table 11.1 below.

Item Number	Equipment Name	Description
1	Compaq P1650	P-266 Laptop Computer
2	SCXI-1000DC	Chassis and Power Supply
3	SCXI-1200	12 bit DAC Module
4	SCXI-1121	4 Channel Isolation Amp
5	SCXI-1321	Terminal Block
6	Wavetek Model 132	Frequency Generator
7	MB Dynamics SS250	Amplifier
9	LDS Model 203	2 pound Shaker Table
10	Bridge Completion Block	4 Channel Wheatstone Bridge

Table 11.1, Experimental Equipment List

An image of the equipment and setup is shown in the figure below.

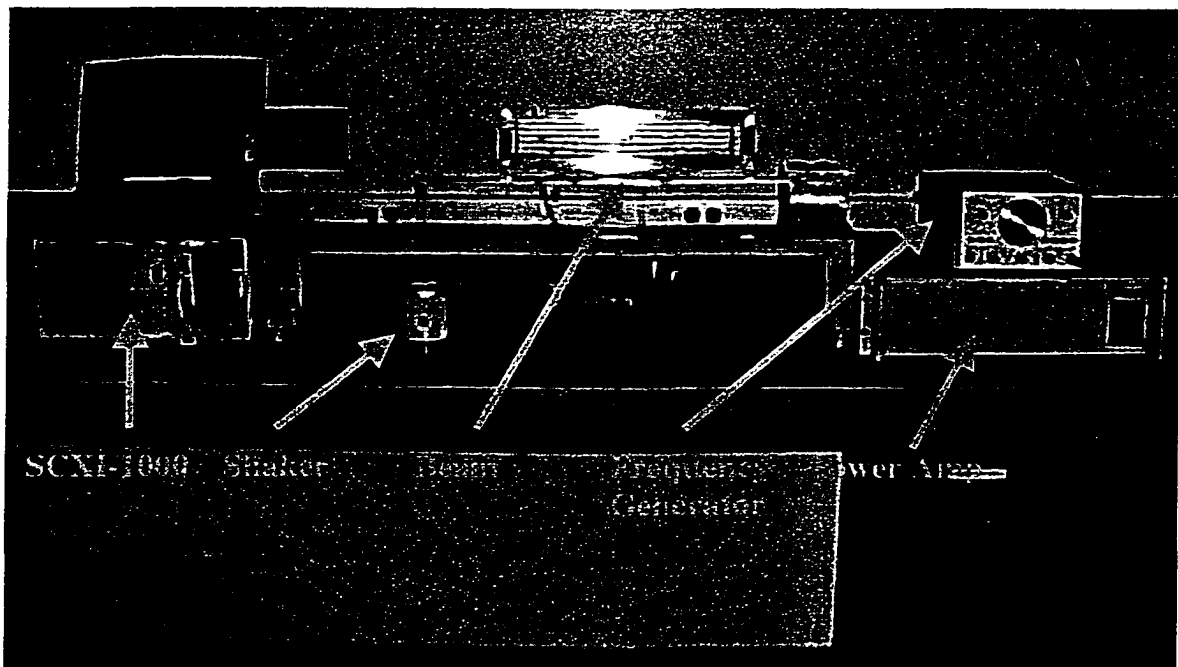


Figure 11.7, Experimentation Setup

The Model 203 shaker table did not produce a large amount of deflection. A load of two pounds on the above beam would create a maximum strain of approximately 350

$\mu\epsilon$ at the gauge closest to the root and a maximum tip deflection of approximately 0.05 inches. The other two gauges had less strain, Gauge Two was two thirds of Gauge One and Gauge Three was one third of Gauge One. The goal of the experiment is to demonstrate the ability to detect and correct errors, so this miniscule amount of strain would increase the difficulty of error detection. The SCXI-1121 isolation amplifier was set up with a gain of 2000 to make the strain readings more accurate.

There were three LabView virtual instruments (vi's) written for this experiment. The first without error correction (dynamicbeam.vi), the second with behavior-based error correction (errorcorrect.vi), and the third with typical error correction of comparison and voting (typical.vi). All of the vi's use a buffer to acquire data before processing and then continue to acquire and process data in near real time. The vi's also have identical front panels, represented in the figure below.

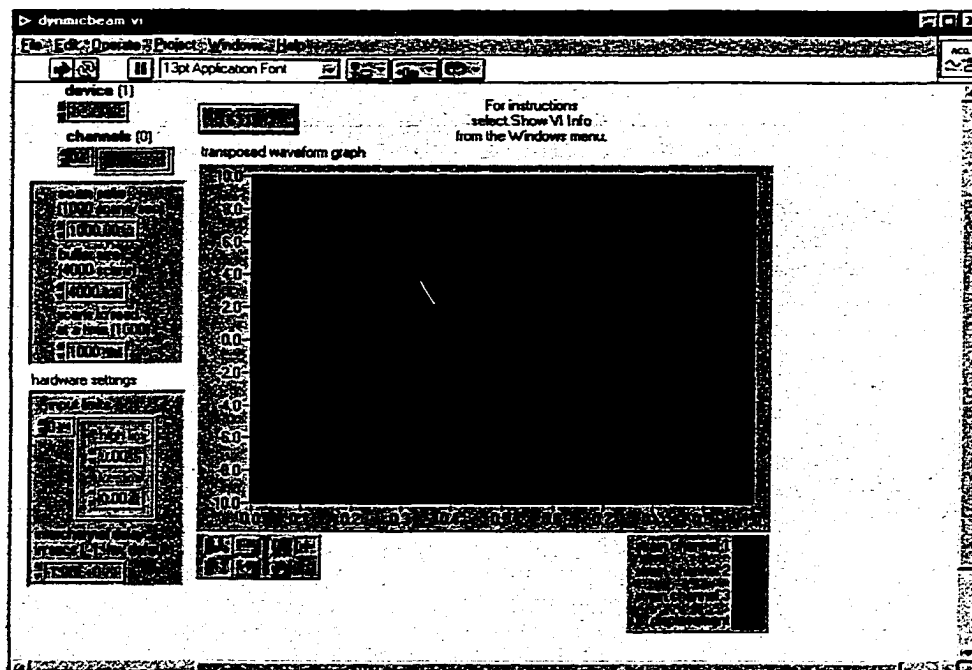


Figure 11.8, VI Front Panel

The front panel will display the calculated tip deflection of the beam as well as the deflection readings from each of the three gauges. The vi's will store all strain and deflection data into a spreadsheet file when the "STOP" button is depressed for further analysis and data reduction using Microsoft Excel. The pertinent wiring diagrams are presented in Appendix F.

11.2.1 Experiment 1: All three gauges good

In the first experiment, all three gauges are functional. The shaker table was driven with a 1 Hz sine wave and the virtual instruments were setup to sample at 10 Hz. A 1 Hz sine wave was chosen to insure that the driving frequency of the beam was well below the first natural frequency. This will ensure that the shape of the beam remains in the first mode shape only. The calculations used for tip deflection are not valid for the higher mode shapes. The natural frequency for a cantilever beam uses the following equation from Roark's (Young, 1989):

$$f_1 = \frac{K_n}{2\pi} \sqrt{\frac{EI}{wl^4}}$$

The first natural frequency for the beam used in this demonstration is 16.9 Hz. This was experimentally verified.

For this first experiment, all three virtual instruments should provide identical results. Figure 11.9 presents all three outputs. Data was sampled for thirty seconds, but only the first ten seconds are displayed for clarity.

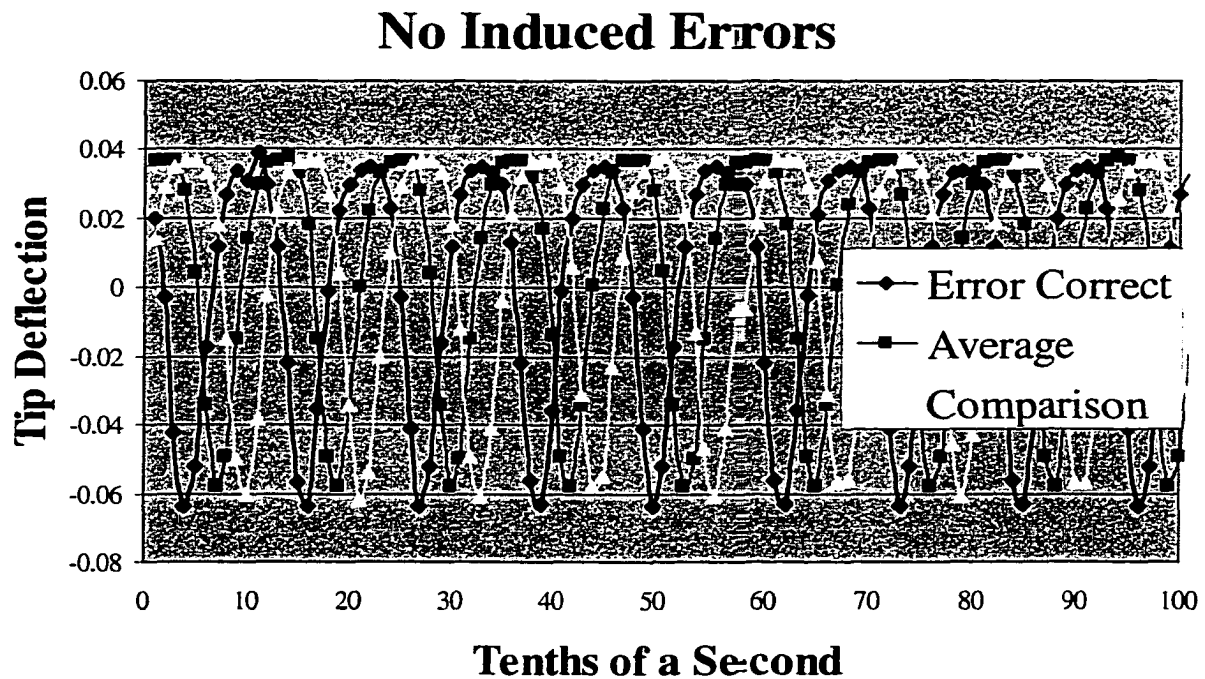


Figure 11.9, Tip Deflection, No Induced Errors

The Error Correct vi utilizes the behavior-based error correction technique. The Average vi simply takes the average of the three strain gauges for the tip deflection, while the Comparison vi , compares tip deflections for each gauge to determine which gauges are good and which gauges are bad. All three plots are virtually identical. There is some slight offset due to the initiation of sampling, but the waveforms are identical. A DFT was performed on this data to verify the input frequency. Using the DFT in Microsoft Excel, the input frequency is 0.9 Hz. The Wavetek frequency generator is analog, so this value is acceptable.

11.2.2 Experiment 2: One bad gauge

In this experiment, all three gauges are functional for part of the time. At approximately ten seconds into the experiment, Gauge Three (the one furthest from the

root) is disconnected from the signal conditioner, simulating a broken wire. This will cause the output for Gauge Three to go full scale. The result is presented in Figure 11.10.

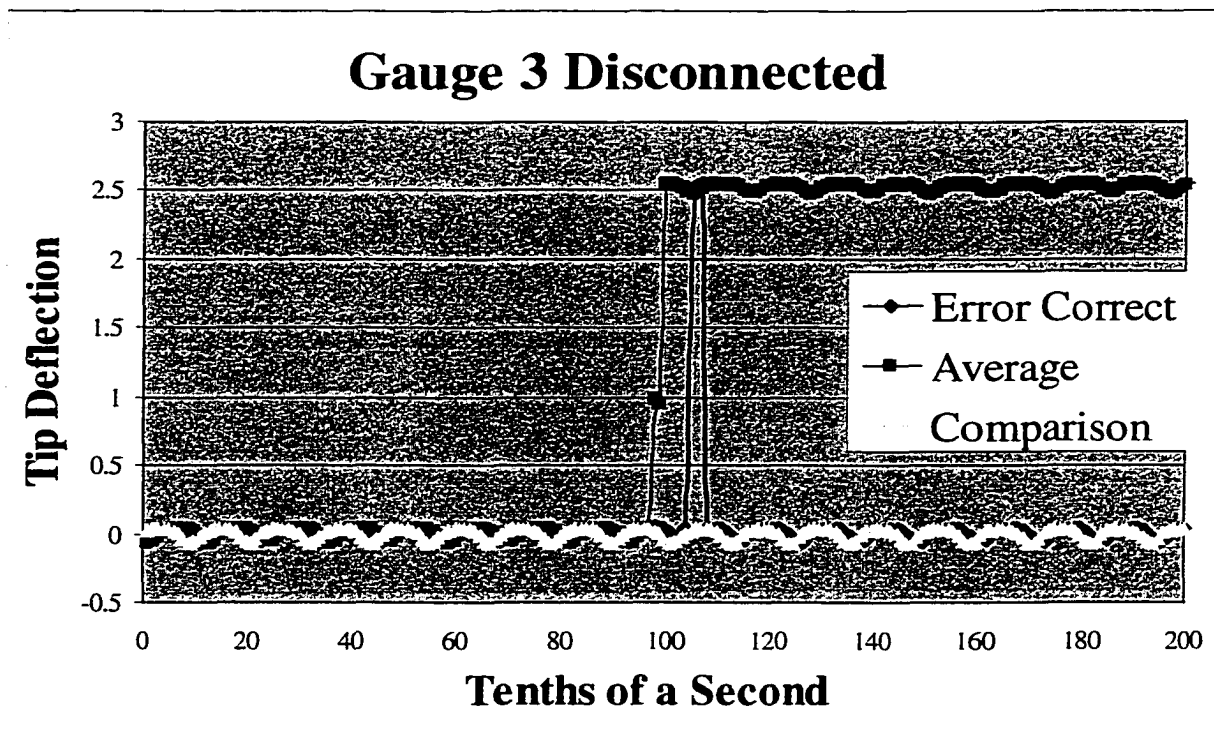


Figure 11.10, Tip Deflection, One Bad Gauge

Only the first twenty seconds of data is presented here. When the gauge becomes disconnected, the Average calculation continues to determine tip deflection, but with one gauge reading full scale. This causes the resulting large errant reading. The Comparison technique works extremely well for this experiment. This technique immediately detects the errant reading from Gauge Three and uses only Gauges One and Two to determine tip deflection. The behavior-based method also works extremely well. This method requires memory so there is the blip where the fact that Gauge Three is not responding as it should is recognized. This method then makes the switch to determining tip deflection using

only Gauges One and Two. The behavior-based and comparison method both have a coverage of 33%.

11.2.3 Experiment 3: Two bad gauges

The third experiment demonstrates the situation that other error detection techniques cannot support, the case of more faulty gauges than reliable gauges. At approximately ten seconds into the demonstration, Gauge Three will be disconnected from the signal conditioner, and then at approximately twenty seconds into the demonstration Gauge Two will also be disconnected from the signal conditioner. The result is presented in Figure 11.11.

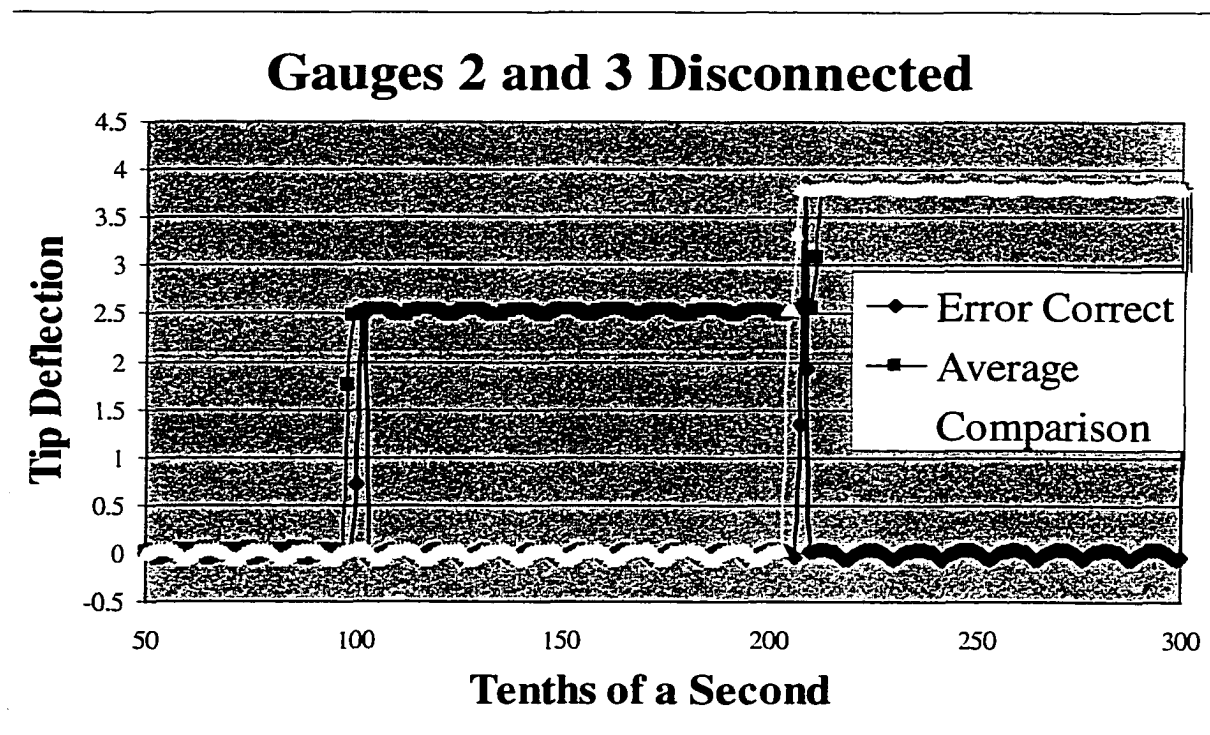


Figure 11.11, Tip Deflection, Two Bad Gauges

For this experiment, only the last twenty-five seconds are displayed. At approximately the ten-second point, both the behavior-based approach and the comparison approach

performed as expected. The behavior-based approach has the small blip where it determines the system is trapped. The comparison technique performs admirably. The comparison technique; however, cannot handle the loss of two gauges at twenty seconds. At approximately twenty seconds, this method determines that Gauges Two and Three are the good gauges and uses the result from these gauges, giving a flat line output. There is no way for this method to know that Gauges Two and Three are incorrect. A full-scale reading is possible, and with two gauges reading full-scale, that must be the correct reading. The behavior-based approach; however, shows its flexibility. When Gauge Two is also removed the behavior-based approach, detects that the behavior of the sensor is not what it should be and consequently it ignores that sensor as well and only uses Gauge One to determine the tip deflection. That gives the behavior-based approach a coverage of 67%, while the comparison method still only has a coverage of 33%. In a completely redundant system, the behavior-based method requires only one sensor to correctly perform the task. If the designer is aware of a useable indicator of the correct sensor behavior, the behavior-based method can accommodate any number of sensor failures as long as one sensor is still behaving appropriately. The reactions are setup to accomplish the task of tip deflection. Error detection allows for the rearranging of the reactions to accomplish this task. The primary reaction is the use of all three gauges, but error detection $\downarrow$ a sensor not behaving appropriately $\downarrow$ will allow a lower level reaction, one that uses only one or two gauges, to subsume the higher reaction. Circularity imposed by the error detection scheme forces the behavior to check all of the reactions before using all three gauges to determine tip deflection.

11.3 Vibrating Beam Conclusions

The use of redundant sensors is extremely valuable. The problem lies in deciding which sensors are reliable and which sensors have become unreliable. The typical method compares similar outputs and determines a majority rule through voting. This method is very effective as long as the number of unreliable sensors is, at most, one-third of the total number of sensors. The capability of this type of error correction was clearly demonstrated by Figure 11.10. The weakness of this system was also clearly demonstrated by Figure 11.11. When the number of unreliable sensors was greater than one-third the total number of sensors, the system chose the bad sensors as reliable and greatly over-estimated the tip deflection.

The behavior-based method does not compare sensors to each other. Such comparison is not valid. There is no way to determine that one sensor is correct while another sensor is incorrect without the use of an external observer. An external observer is one that possesses more information than the system requiring the sensors. Therefore, the behavior-based method assesses the behavior of the sensors individually. For this experiment, it was known that the sensors would be measuring a vibrating beam, so the sensors should have a varying output. If the output is not varying, then the sensor should not be declared to be in error, only that the behavior of the sensor is not as expected so the use of other sensors/reactions to determine the tip deflection is recommended. This is very similar to what Paasch and Agogino (1993) accomplished when attempting to detect errors in the Time-of-Flight Scintillation Array. "So, if only one channel is changing as expected, the diagnostic system should indicate that the photomultiplier tube is the most likely cause of failure; if multiple channels are changing, it should indicate the high-

voltage module as the most likely cause.” (Paasch, 1993) They were not trying to decide among redundant sensors, only which component was the most likely cause of failure; however, they are still examining the system behavior to determine the most likely cause. Their effort was directed at quickly repairing the system (i.e. detecting the faulty component) not upon functioning in a degraded fashion. However, they were still using an “expert diagnostician” which would compare current values to expected values given a known input to determine that the behavior was not as expected and probably in error. This system contained large files of data corresponding to component failures used to determine the functional error. So although they are looking for the same types of failures, they are doing it in a different fashion. An external observer is still used to compare values to expected values.

This was a real world example, albeit a very uncomplicated experiment. There are countless examples in the real world for the use of multiple sensors to determine a single output. The only difficult part is that the designer must realize what the behavior of the sensor is supposed to be. This can be accomplished, as it was in the robotics example, by looking for trapped trajectories or by looking specifically at the sensor behavior as demonstrated here and forcing the behavior to check all of the reactions – circularity. This method will require more time in the development process, but the rewards of a more effective system are clearly worth the investment. The key concept here is that the designer does not have to know *a priori* what the *value* for the sensor is supposed to be only the behavior for the sensor. There is no need for large data files for expected outputs given certain inputs. There is no comparison of one sensor with another

sensor. There is no need for an external observer equipped with additional sensors. The behavior-based system is very simple and very fast.

12 Dynamic Control of a Vibrating Beam

The demonstration depicted in the previous section did not possess any control of the vibrating beam; it only showed how the measurement of the displacement could be in error. This demonstration will now add dynamic control to a similarly vibrating beam. This demonstration is similar in that there is only a single behavior, actively damp out a vibrating beam. For this demonstration, the goal of the behavior is to damp out the vibration of the beam and reach the reaction that applies no force. The first type of reaction applies a damping force to stop the vibration of the beam; the second type of reaction is to apply no damping force. Again circularity must be maintained. Activation of each reaction will be dependent upon sensor behavior, not upon sensor value. If the behavior of a sensor is not as expected, the reactions using that sensor will be inhibited causing the rearranging of reactions.

The United States Air Force and many other organizations are interested in smart structures. Smart structures are specially designed for their intended use, whether they can passively or actively damp out vibration or that their behavior under load is more conducive to the operation. Composite materials often offer the best opportunity to achieve these functions. The X-29 with its forward swept wings is an example of a smart structure, the wings are designed to decrease the angle of attack under load; thus preventing the wings from being torn from the aircraft. Composites also offer the advantage of allowing the user to design a structure to passively damp out vibrations or to embed force actuators for the purpose of actively damping out vibrations.

12.1 Experimental Set up

This experiment uses a 6061-T651-aluminum beam, 6.50mm T x 38.30mm W x 567mm L (0.256" T x 1.508" W x 22.4" L). Displacement is measured using two Micro Measurement strain gauges, CEA-13-240UZ-120, refer to the previous section for the details on these strain gauges. To actively damp the beam, some kind of force actuator is required. Piezoelectric actuators were chosen due to their small size and force capability. Active Control Experts, QP20N force actuators were used. The specifications on these force actuators are given below.

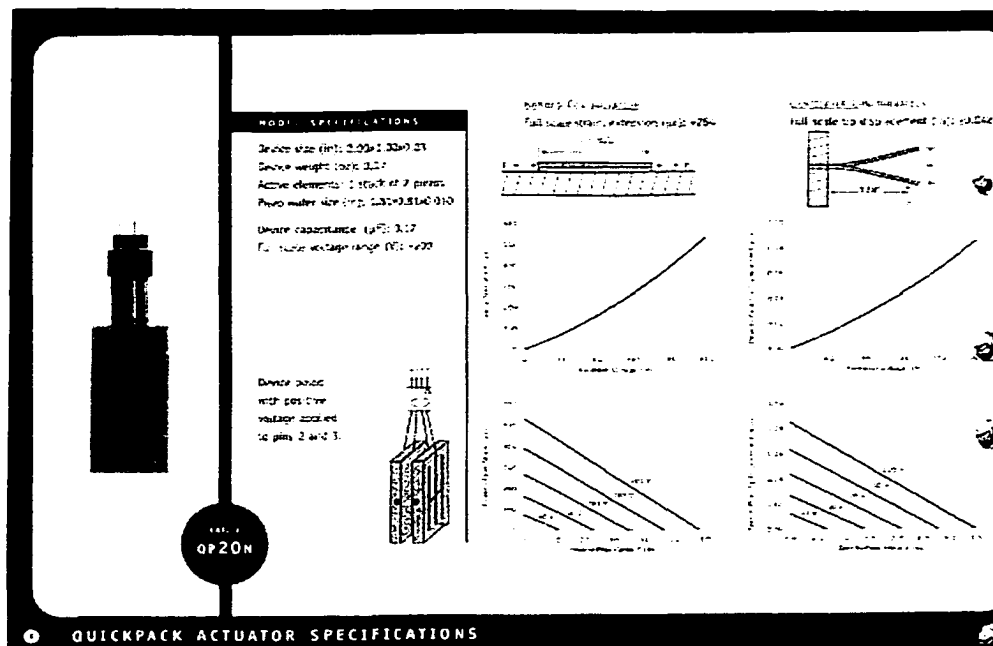


Figure 12.1, QP20N Characteristics

The beam with strain gauges and force actuators is depicted in Figure 12.2.

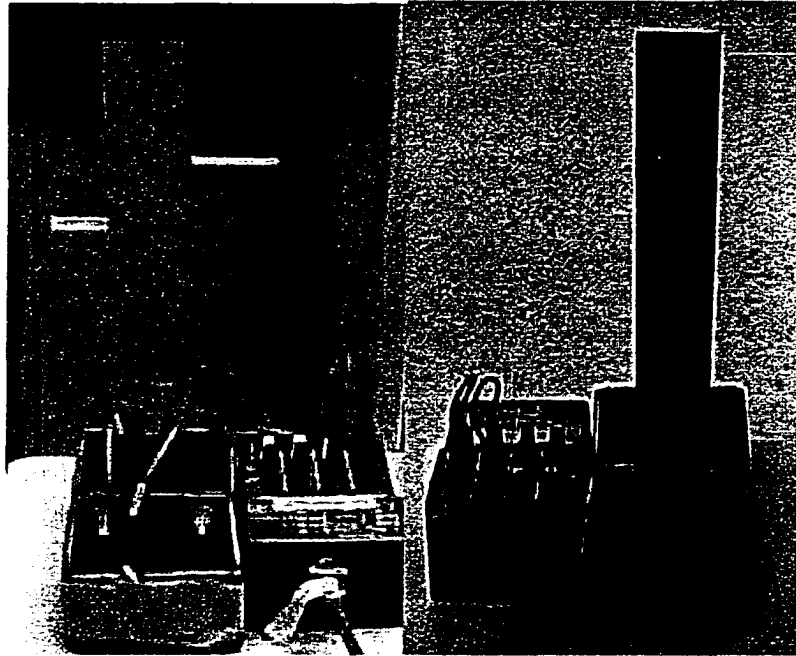


Figure 12.2, Actively Controlled Beam

The experimental set up was very similar to the Vibrating Beam experiment in the previous section. This experiment; however, required greater speed than could be utilized with a parallel port connection. Therefore, this experiment used an internal DAC board. The table below lists all of the equipment used.

Item Number	Equipment Name	Description
1	SCXI-1000	Chassis
2	SCXI-1121	4 Channel Isolation Amp
3	SCXI-1321	Terminal Block
4	BNC-2090	BNC Connector
5	AT-MIO-16X	16 Bit Internal DAC Board
6	Wheastone Bridge	4 Channel Bridge Completion
7	Micon PII-266	Desktop Computer
8	EL-1224/5	Power Amplifiers

Table 12.1, Active Damping Equipment List

The experimental setup is illustrated in Figure 12.3.

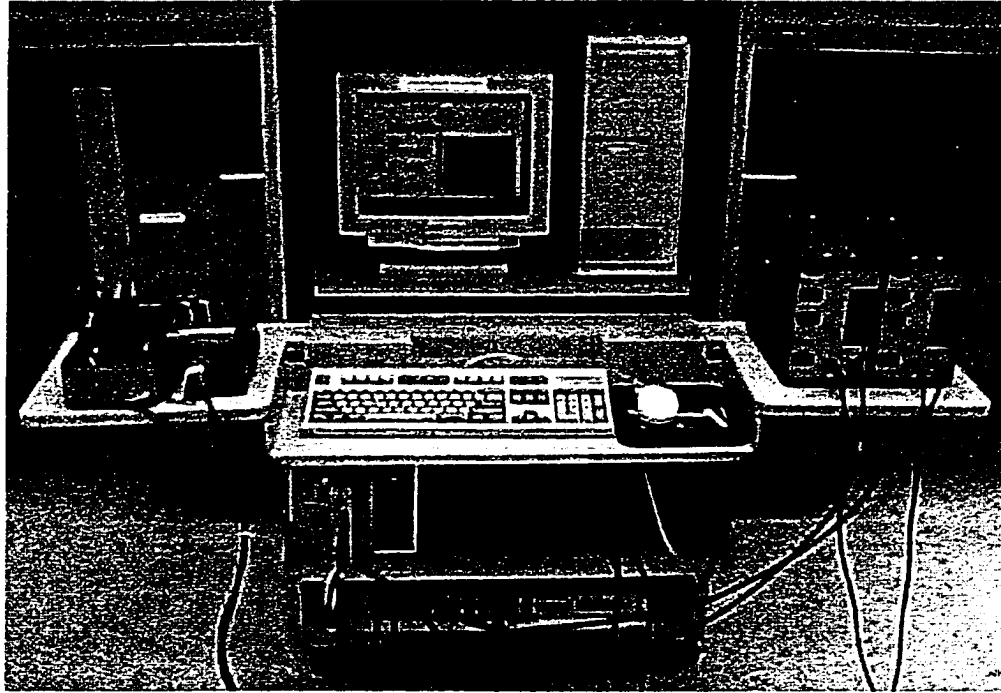


Figure 12.3, Active Damping Experimental Setup

12.2 Experimental Algorithm

Dr. Robin Redfield at the United States Air Force Academy provided the algorithm used to actively damp the beam. (Redfield, 2000) It is a very simple algorithm that allows the DAC system to run as fast as possible. The approximate velocity of the beam is determined by taking the difference of the last two strain measurements and dividing that difference by the time between measurements. This velocity is then multiplied by a gain to determine the output voltage according to the following equation.

$$v_o = -g \left(\frac{\epsilon_n - \epsilon_{n-1}}{\Delta t} \right)$$

The negative sign is required to damp out the vibrations; a positive sign would have caused the beam to be driven instead of damped. When both strain gauges are functioning, the strain is the average reading of the two gauges for sequential time steps.

This very simple algorithm was programmed into LabView for control of the beam. The front panel is shown below.

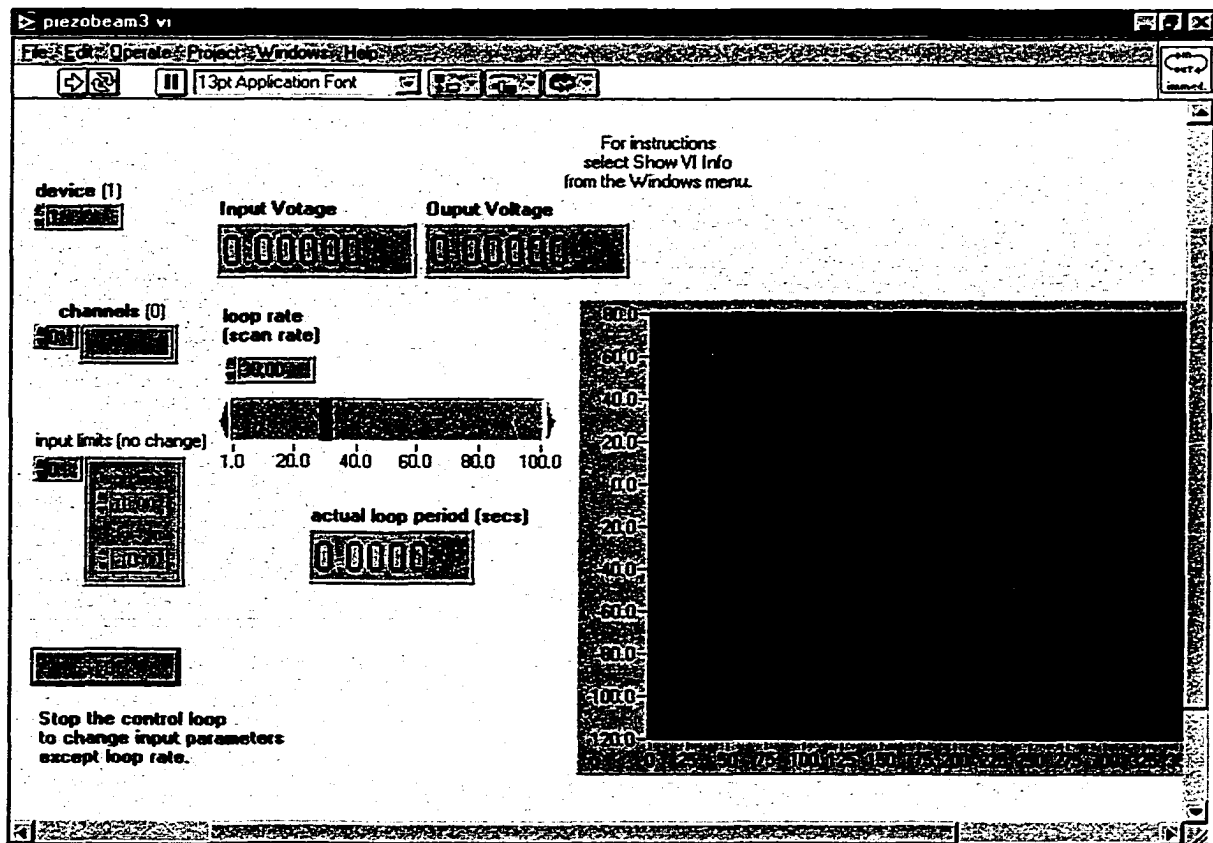


Figure 12.4, Active Damping Front Panel

Both virtual instruments, error correction and no error correction, use the same front panel. The virtual instruments read the strain from two channels, calculate an output voltage and then send the output voltage to the power amplifiers. When the “STOP” button is depressed, the virtual instrument will display the time-displacement plot of the beam. The goal of the error correction behavior is to apply no force to the beam; the beam has stopped vibrating. The activation of the base reaction requires that both strain gauges have a velocity less than $2 \mu\text{ε}$ over a sample period. Otherwise, there are three reactions to determine the appropriate damping force. One that uses both strain

gauges, one that use the right strain gauge and one that uses the left strain gauge. Circularity is maintained by requiring the control loop to check each of these reactions before allowing for a zero force to be applied. Error detection is what determines which reaction is active. This control program is setup exactly as if it were an IRA robot. See Appendix H for the full details on the virtual instruments including transition diagram for both virtual instruments.

The maximum scan rate achievable by this equipment was 40 scans per second. For this demonstration, the best damping rate achievable was with a scan rate of 40 scans per second and a gain of 0.015.

12.3 Experimental Results

The experimental results are divided into two sections. The first section is without error detection and the second section is with error detection.

12.3.1 Active Damping with No Error Detection

The experimental procedure involves turning on the virtual instrument and striking the beam. The virtual instrument is shut off when the beam has damped out. The figure below presents the results without error detection.

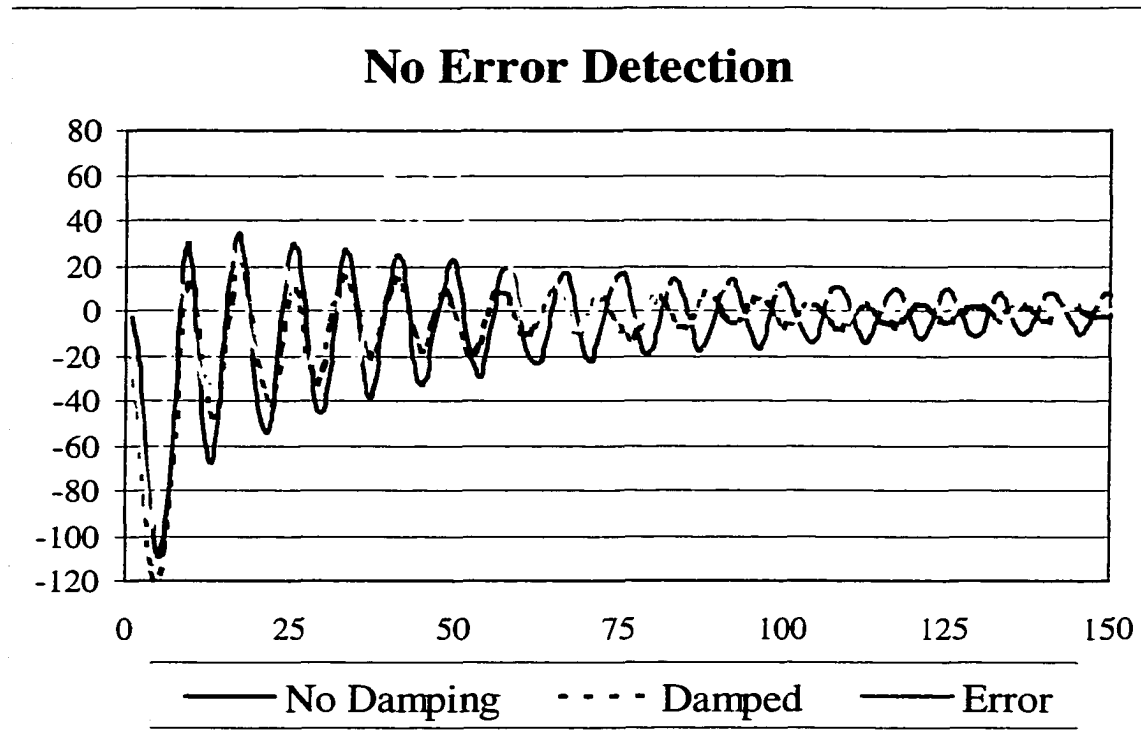


Figure 12.5, Active Damping with No Error Detection

The solid line represents the case of no damping. The gain was set to zero and the beam was allowed to vibrate until it stopped. The dashed line represents the case of active damping with both strain gauges functional. The patterned line represents the case where both strain gauges were functional for part of the time, but then at approximately the thirtieth time step one of the gauges was disconnected. This caused the virtual instrument to calculate a huge velocity difference and, therefore, a large force was required to counter the velocity. The amount of time for the beam vibration to decay is large because in effect, the velocity detected is only one-half of the actual velocity.

Frequency analysis was performed on the active damping of the aluminum beam to determine the damping ratio (ζ) for the beam both with and without the active

damping. The Half Power Method is the most common method used for determining the damping ratio. (Jones, 1990) This method uses the following equation.

$$\zeta = \frac{\Delta\omega}{2\omega_d}$$

This equation was applied to the three curves in Figure 12.5 to determine the damping ratio for the aluminum beam. The damped natural frequency of the beam was calculated to be 4.6 Hz, the experimental results varied from 4.9 – 5.1 Hz. That is only a 4% variation over the forty runs of the beam. The results are presented in the following table.

Case	Iterations	Mean (ζ)	Variance (ζ)
Undamped	2	0.012	2.32×10^{-7}
Active Damping	10	0.0331	3.40×10^{-5}
Active Damping w/Error	10	0.0294	1.45×10^{-5}

Table 12.2, Damping Ratio with No Error Detection

By removing one of the strain gauges during the experiment, the damping ratio was reduced from 0.0331 to 0.0294, that is a reduction of 11%. Since the virtual instrument works by determining the velocity, when the one gauge is disconnected it incorrectly determines there to be a very large velocity, so it induces a very large force. Then, since one gauge is disconnected, the velocity resulting from that gauge is zero. The force to damp the beam is then roughly cut in half causing the reduction in damping ratio. Figure 12.5 shows that there is a noticeable difference in the damping of the beam, both with the large deflection when the gauge is disconnected and in the trailing portions of the diagram. If both gauges were removed the apparent velocity would be zero and there would be no damping force applied to the beam.

12.3.2 Active Damping with Error Detection

The same procedure as above was then performed on the virtual instrument that incorporated error detection. The results are depicted in Figure 12.6.

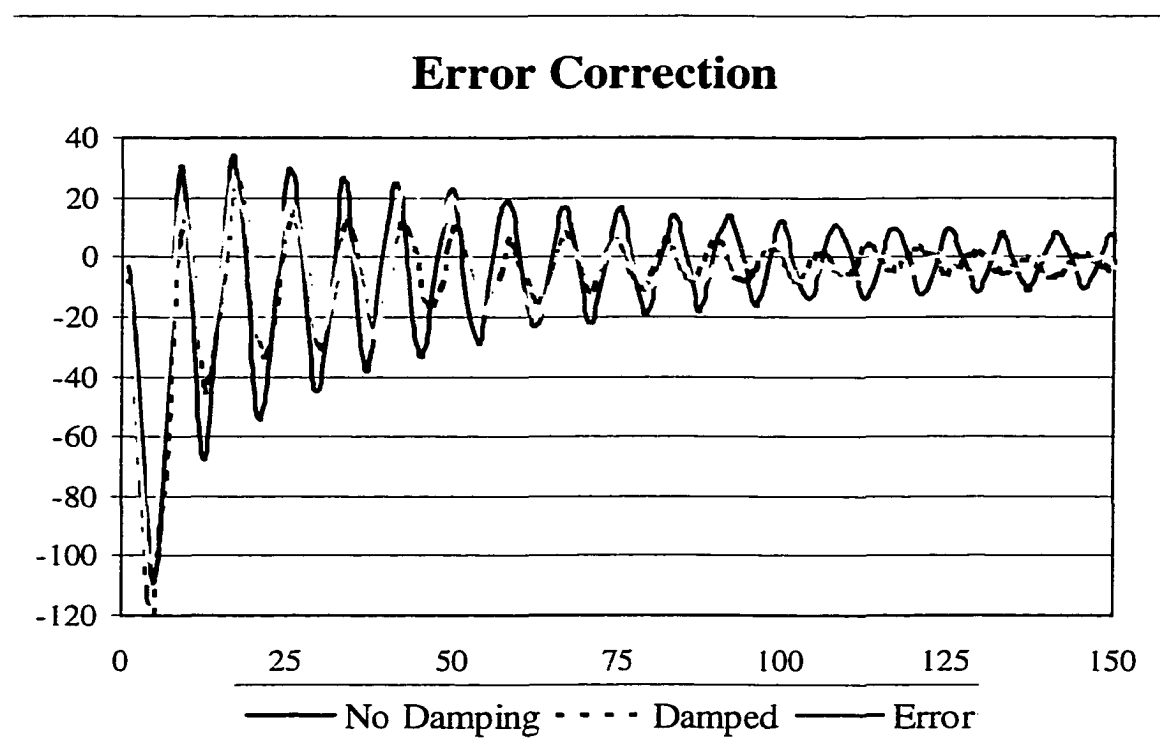


Figure 12.6, Active Damping with Error Detection

The plots are the same as in the first experiment. When one gauge is disconnected, there is virtually no effect on the overall damping of the beam. Since memory of past performance is required, there is a slight blip, but then the beam quickly returns to the same damping as in the no error situation.

Frequency analysis was also performed on the beam that incorporated error correction. The undamped case remained unchanged. The same analysis techniques presented in the previous section were used for this section. The results are presented in the table below.

Case	Iterations	Mean (ζ)	Variance (ζ)
Undamped	2	0.012	2.32×10^{-7}
Active Damping	10	0.0425	7.87×10^{-5}
Active Damping w/Error	10	0.0422	5.57×10^{-5}

Table 12.3, Active Damping with Error Correction

The induced error of removing one of the strain gauges had no significant effect on the damping ratio of the beam. This is obvious in Figure 12.6. This is important; there was no significant difference to the control when one of the gauges was removed. If both gauges were removed the control program would not apply a damping force. As was mentioned in the Interactive Representation Section, the false negative is system undetectable.

12.4 Statistical Analysis

Since one of the goals of this research was to be able to say those systems which possess system detectable error work better than those systems which do not possess system detectable error, a statistical analysis of this control problem was performed. The full details of the statistical analysis are contained in Appendix J. Stevens (1990) recommends the relaxation of the confidence level from the typical 0.05 to 0.1 or 0.15 in order to increase the power of the test. Mathematica provides the p-value for all tests performed. The p-value is the α level where the selection or rejection of the hypothesis is possible. (Rosner, 1990) It should be noted in Appendix J, that the confidence for most tests could have been increased well above 90% to get the same result, this of course would have lowered the power of the test.

12.4.1 Statistical Analysis of Results with No Error Detection

The first analysis was done on the tests performed with no error detection. The goal of the analysis was to determine what effect, if any, the induced error had on the damping ratio of the vibrating beam. Since the population size is relatively small, a comparison of means tests utilizing the t-distribution was used. This requires that the variance of the two tests be considered equal. Using the F-ratio comparison of the variances can be made. With a confidence of 90%, the variances are equal. The comparison of means showed, with a confidence of 90%, that the means were in fact different. One other test should be performed to insure that the hypothesis test is valid, the calculation of the power of the test. The power the probability that the sample point is actually in the critical region of the test. (Hogg, 1978) If the power is high, then the result of rejecting the hypothesis is concluded to be a good decision. If the power is low, then rejecting the hypothesis, may introduce a Type II error. For this test the power was found to be 62%. That is, there is a 62% chance that rejecting this point is correct or a 38% chance that the means are the same. The hypothesis is therefore rejected; there is a significant difference in the means of these two populations. The induced error had a significant effect on the damping ration of the vibrating beam.

12.4.2 Statistical Analysis of Results with Error Detection

The second statistical analysis was performed on the behavior-based error detection to determine whether the induced error had an effect on the damping ratio of the vibrating beam. The same procedures discussed above were conducted for this comparison. The F-ratio test verified that there was no significant difference in the variances of the tests, to a confidence of 90%. The comparison of means failed to reject

the null hypothesis; in other words, it is accepted that there is no significant difference in the means. There is no test to verify that two things are the same, only that we are not able to reject the hypothesis that they are the same. Finally, the power of the test was determined to be only 12%. That is, there is a 12% chance of rejecting the hypothesis at this point. The hypothesis is therefore accepted that there is no significant difference in the means of these two populations. The induced error had no effect on the damping ratio of the vibrating beam when the behavior-based error detection scheme was incorporated into the control.

12.4.3 Detection to No Error Detection – No Error

The third comparison was made between the error detection scheme and the no error detection scheme with no induced error. The same analysis was again performed. The F-ratio test verified that there was no significant difference in the variances with a confidence of 90%. The comparison of means rejected the null hypothesis. In other words, there is a significant difference in the means with a confidence of 90%. The power of the test was determined to be 95%. The difference in the means is significant with no induced error. The system that possesses system detectable error is more effective than the system that does not possess system detectable error, even when there are no induced errors. One reason for the superior performance has to do with the behavior activation. With no error detection, activation is based upon the average velocity calculated by the two strain gauges. The resolution of the strain gauges was 1.38 $\mu\epsilon$ and activation was based on a change of only 2 $\mu\epsilon$. The chance of the average velocity being zero was quite high, resulting in zero output. The error detection scheme; however, looked first at the average, but then it also looked at each gauge, so it was

correcting the beam more times than the no error detection scheme resulting in superior performance even where there are no errors.

12.4.4 Error Detection to No Error Detection – Induced Error

The fourth comparison was made between the error detection scheme and the no error detection scheme with an induced error. The F-ratio test was performed on the variances. There was no significant difference in the variances of these two tests with a confidence of 85%. The variances between these two tests were the largest, but an 85% confidence is acceptable according to Stevens. The comparison of means rejected the null hypothesis; there is a significant difference in the means, with a confidence of 90%. The power was calculated to be 99.7%. The induced error resulted in a significant difference between the error detection and the no error detection schemes.

12.5 Active Damping Conclusions

This demonstration allowed for the use of behavior-based error detection and correction within a control loop. The results were far better than expected. The error detection and correction scheme actually outperformed the no error detection when there were no induced errors, similar to the case of the animat with no induced errors. Both cases did an efficient job of damping out the vibrations in the beam, but the error detection method performed 28% better than the case with no error detection. With an induced error the results were even more dramatic. The error detection method outperformed the non-error detection method by 44%. A comparison of means test was also performed on these two cases. There is a 90% confidence that the means are different for comparing both error detection and no error detection, both with and without an error. Stevens recommends the use of lower α to allow for higher power. The power

of these tests were also calculated to determine the confidence of not making a Type II error. For the no induced error situation, the power is 99.7%. For the induced error case, the power is 94.95%. The full details are contained in Appendix J. The results contained in Appendix J are summarized in Table 12.4.

Comparison	Hypothesis	P-Value	Power of Test
EDER – EDNE	Fail to Reject – 90%	0.468	12%
NEDE – NEDN	Reject – 90%	5.53×10^{-2}	62%
EDER – NEDE	Reject – 90%	2.29×10^{-4}	99.7%
EDNE – NEDN	Reject – 90%	3.24×10^{-3}	94.95%

Table 12.4, Hypothesis Test Results

EDER – Error Detection with an induced error.

EDNE – Error Detection with no induced error.

NEDE – No Error Detection with an induced error.

NEDN – No Error Detection with no induced error.

The induced error had no effect on the damping ratio when the behavior-based error detection was incorporated. The induced error did effect the case with no error detection. As expected the effect was not enormous. The force to damp the beam is the average velocity calculated from the two gauges. When one gauge is removed the velocity is divided by two, and subsequently the required damping force gets divided by two.

The behavior-based error detection method has proven itself to be a very effective way of adding redundancy to control algorithms without necessitating an external observer or adding to the complexity of the algorithm. There was no reduction in processing speed for the two virtual instruments. They both had the same maximum effective speed of 40 samples/sec. Therefore, the speed is a hardware issue as opposed to

a software issue. When the workload of the virtual instrument was increased, outputting more data or displaying more data, the sample rate would decrease. As the sample rate decreased, the performance of the damping also decreased. Dr. Redfield considered 40 samples/sec to be really too slow, but acceptable. The point of this experiment was not to develop the best active damping of an aluminum beam, but to display the power of system detectable error and the ability to incorporate this definition of system detectable error into other types of systems.

There is no current error detection and recovery scheme that could have handled this error situation. There were only two gauges. There is no way to determine one gauge is incorrect and the other gauge is correct through the use of comparison. A full-scale reading is valid. The only method that would work would entail the addition of another sensor and as well as an external observer. These additions would increase processor time and violate the definition of system detectable error. The behavior-based definition of error allows for system detectable error without resorting to an external observer, without requiring multiple redundant sensors for comparison purposes, and without slowing down the processing speed.

13 Discussion

The previous sections have detailed and given examples of how to develop systems that possess system detectable error. The key point is that it is possible for a system to be able to determine that it is in error and to mitigate that error without the use of an external observer. The reliance on an external observer is often effective, the case of one bad strain gauge on the vibrating beam is one example. The problem with these types of systems is that they all require some kind of comparison and voting scheme. Comparing and voting then requires that the number of faulty sensors be one-third the total number of sensors, majority rule. This means that the comparison method can have a maximum coverage of 33%. It cannot achieve better than that.

The behavior-based approach however, can have an unlimited amount of coverage as long as at least one sensor is still behaving appropriately. There is no voting scheme and more importantly there is no comparison of one sensor with another sensor. This is simply not valid in the context of system detectable error. Without providing an external observer that simultaneously views the system and the universe the system operates in, there is no way to say one sensor is correct while another sensor is incorrect. More information is required than just the values of the sensors. This is why the comparison method requires that the number of faulty sensors be one-third the total number of sensors. This method simply looks for the majority. The behavior-based method does not look at the *value* of the sensory output, only the behavior of the sensory output. A system is not capable of knowing if the *values* of its sensors are correct. This

was shown by Reason and how the sensory input from the eyes could cause a human to lean forward. It was also shown in Ferrel's (1993) example for a thermometer giving the wrong temperature reading. Neither the human nor the thermometer was able to determine that the inputs were incorrect. An external observer was required. Someone to say, "Hey, you are leaning forward", or "The window is ice cold, there is no way its ninety degrees out there". It is possible for the system to recognize that the behavior resulting from the sensor or the behavior of the sensor is not correct.

For the robotic systems, the recognition of faulty behavior due to a sensory input is manifested by a lack of circularity in the organization of the system. The finest example of this was in the simulated world of the animat with no induced errors. The simulated animat lost its circularity and died when there were no sensory errors (see Section 9). No current error detection scheme would be able to have prevented the death of these animats. There were no sensory faults. Multiple redundant sensors would not have helped. Pin's path similarity would not have worked. Determining that there was a lack of circularity in the organization was able to free the animat from its trapping trajectory to either recharge or escape from the trapping region. All of this was done without enlisting the aid of an external observer, without adopting a preprogrammed plan of action to cope with situation, without providing an explicit definition of a sensory fault, without attempting to represent the external environment, and without providing the system a list of possible outputs given possible inputs. Detecting the lack of circularity is performed internally to the reaction activation. It is seamless with no perceptible degradation in speed in the simulated world or in the real world.

For simplified systems, like the vibrating beam, identifying improper sensory behavior is also capable of determining errors without the use of an external observer, without comparing one sensor to another, without supplying the system with vast sums of data and without generating plans. This is also seamless with no perceptible degradation in speed.

The only drawback to detecting a lack of circularity or improper behavior is that the designer must now determine what constitutes a lack of circularity. In the simulated world where all of the sensors were identical, this was not a difficult task. What constituted a trap was the same for every sensor type. In the real world where every sensor was different, the task was somewhat more daunting. What constituted a lack of circularity, a trap, was different for each individual sensor on the real robot. This required a great deal of time in the final programming stages to make a match between the task and the robot. The amount of time is based on the internal operation of the control loop. Once the match was made, there was near flawless operation of the system. It avoided the walls, explored the entire environment and was able to handle the occasional bumper failure.

A system must consist of the machine, the environment and the task. More importantly, maintaining the circularity of the organization will not necessarily make the machine useful. This is because all systems are capable of becoming fooled ¹ the human model that leans forward due to the input from the eyes or the leech that feeds for hours because its crop is being emptied. In both of these cases, the circularity of the organization was being maintained. The human was not allowed to close his or her eyes. This prevented error recovery from taking place. The human was trapped in a forward

lean. In the case of the leech that was being fed while its crop was being emptied, the leech was maintaining the circularity of its organization. It was doing everything in its power to reach the state of wall distension, but it was not occurring. The leech was trapped in ingestion. There was no way for the leech to detect that what it was doing was wrong. That would have required the use of an additional sensor that the leech does not possess. Either one to tell the leech that it has been feeding for hours (i.e. time) or one that tells the leech that it is not feeding on blood (i.e. taste). The minimalist robot also fell into one of these traps when it had an induced bumper error. The bumper was not functional, the system detected this and it then relied on the photoresistors for both avoiding obstacles and avoiding bumping. During one of the runs it hit the wall with the non-functional bumper and became stuck. The motors were not powerful enough to get the robot off of the wall. The circularity of the organization was being maintained. The controller was attempting to explore, the motors were told to move the robot forward, but it could not slip off of the wall. The robot would require another sensor to “know” that it was not moving. More powerful motor, or perhaps better frictions between the drive wheels and the environment floor, would have made escape possible. This would be the best solution. This research has already stated that the comparison of one sensor with another sensor is not valid. How could the robot ‘know’ that the ‘non-moving’ sensor was correct while the ‘maintaining the organization’ sensors were not correct? The best solution is always to have the best possible match between the environment and the machine.

The point is that there are no perfect systems. Flawless operation is not possible without considering the entire system: machine, environment and task. An excellent

match between these three will make for the most robust system. With an excellent match made, maintaining the circularity of the organization will insure proper operation, with or without errors. Again, the three behaviors with no errors AnSim simulation showed this the best. The animat did not die and it was able to perform its designated tasks running for more than twenty minutes.

14 Conclusion

Behavior-based robotics have made great strides in recent years. By avoiding complicated plans, the behavior-based systems are capable of surviving in very complicated and dynamic environments. The main drawback and the drawback of all current representational methods is that they cannot possess system detectable error. Those systems that possess error detection all utilize the addition of an external observer that performs the task of polling the sensors and determining which are faulty and which are loyal. This type of system requires that the number of faulty sensors be no more than one-third the total number of sensors. They also require this “omniscient” external observer that possesses more information than the base system. There is simply no way to rule for one sensor in favor of another sensor unless more information is provided to the observer. This will always require the addition of more and more sensors. One of the goals of this research was to whittle down what constitutes an error and a method to detect and correct that error. Adding complexity was not the solution sought. The addition of more and more sensors and the addition of more and more processors are effective, but if the same result can be achieved with less, this is the desired solution.

This research presented a fundamentally different method for detecting an error. Instead of basing error on the value of a sensor, the error is based on the resultant system level behavior – implicit error. A behavior-based definition of error as opposed to a sensor-based definition of error. By not focusing on the value of a sensor, the system can determine if it is maintaining the circularity of its organization. The lack of circularity

indicates error. This is a very powerful definition of error. It is based solely upon the internal operations of the machine. There are no external influences, no plans generated, no voting, and no comparisons made.

The strength of this definition was clearly demonstrated in the simulated world when there were no sensory errors. For the simulated animat, it became trapped attempting to dock with the recharge station. The sensors were reading correctly and the reactions were performing the designated tasks, but the animat was never able to reach the recharge station. It could not dock charge. The circularity of its organization was broken. The result was that the animat used up all of its charge before reaching dock charge and subsequently died. No previous error detection system would have identified this problem.

There were no sensory errors, but the circularity of the organization was broken. The system was not reaching dock charge. The IRA animat detected that it was not reaching dock charge, it then shut off the reaction that was causing it to remain in the trapped trajectory around the charging station; however, it did not shut off the behavior Seek Charge. This allowed the animat to venture away from the recharge station and develop a more appropriate approach trajectory allowing the animat to recharge. There were no sensory errors, but the subsumption animat died. Any current control architecture would also fail this test. There were no errors. The IRA animat, without the use of an external observer and without resorting to some preprogrammed plan, was able to detect that it was not reaching dock charge and to simply try something else in its repertoire. The results were outstanding in the simulated world. The animat could not be trapped, with or without induced errors. The simulation could be run for more than

twenty minutes with no failures. The subsumption animat never survived for more than three minutes without becoming trapped. Keep in mind that the subsumption animat and the IRA animat were programmed identically, except that the IRA animat incorporated a few lines of code to detect when the circularity of the organization had become broken. The error detection code was not a stand-alone module; it was an integral part of each of the reaction activation modules. The operation was seamless with no degradation in speed with or without errors.

This type of error detection and correction is not limited to simple cantilever beams or to behavior-based robotics. All systems can incorporate a behavior-based definition of error see Paasch and Agogino (1993). All systems are capable of knowing what they are doing and determining whether that is accomplishing the intended task. A Sense-Plan-Act system could easily incorporate this behavior-based definition of error. All systems have one thing in common; they have purpose, whether that purpose is easy to identify like a robot welder or more complex to identify like a frog. The organization of the system defines the purpose when the system is placed in the correct environment. The system is then capable of monitoring its own organization to insure circularity. When the circularity is broken, the system is capable of determining why. It can then ignore the input that is causing the circularity to be broken and then hopefully complete its intended purpose.

This research has detailed a behavior-based definition of error and then incorporated it into simulated and real systems. The error detection and correction technique is able to handle the case of a stuck sensor, the false positive. Functional redundancy allows the system to maintain its organization despite the false negative,

which is system undetectable. There really is no way for a system to possess system detectable error for a false negative. There is no way for an internal system to know that something is there when its sensors say that there is nothing there without the use of additional sensors and comparison. This is what this research was trying to avoid. Functional redundancy makes this possible by using two different types of sensors to perform the same task. By using different types of sensors, what causes the failure of one will presumably not cause the failure of the other. This has made for systems that are much more robust without adding complexity, without adding more sensors, and more importantly without resorting to complex plans.

This work has also furthered the design process for these types of systems by detailing a methodology for designing behavior-based robots. This design guidance was also used to show that these behavior-based methods are not restricted to the world of robotics, but can be easily applied to other types of control systems.

Current designers are always seeking more reliable, more accurate, more precise, and more capable sensors. Why? Most of the sensors on the human body are not that exceptional. Humans are practically blind at night. Humans can only hear a very small range. Human touch sensors can be easily fooled. Yet researchers continually strive try to build robots with perfect senses, accurate to 1/1000 of a millimeter, able to discern the slightest shades of gray, able to detect a wall to within 1/1000 of millimeter. Why? Humans cannot do that, yet get around quite well. The method proposed in this paper has allowed a very unsophisticated, very noisy mobile robot to maintain the circularity of its organization to continue exploring its environment with no trouble. The same robot with a subsumption architecture was not able to explore its environment, quickly becoming

trapped by the environment or by a false positive. The IRA animat explored the entire environment and was able to move away from the false positives.

The best answer is always the simplest answer. This is the simplest answer. A behavior-based definition of error incorporated into a control architecture will provide a system with system detectable error. This does not add to the complexity of the system. For the minimalist robot, the code required to add system detectable error to a subsumption model was only twenty-five lines of code for the single behavior. Deactivation of the error detection only required the removal of only five lines of code. This has made for a much more robust system without adding additional sensors, without adding additional processors, and without generating complex plans.

An error is not an incorrect sensor reading. An error is not a mismatch between what the environment provides and what the system detects. An error is not a disconnected or stuck sensor. These are all things that could cause an error, but are not an error in themselves. **AN ERROR IS ANYTHING THAT CAUSES THE CIRCULARITY OF THE ORGANIZATION OF A SYSTEM TO BECOME BROKEN.** None of the classical errors occurred when the simulated robot was attempting to recharge, but the circularity was broken and without the addition of the IRA the subsumption animat died.

15 Future Work

The IRA has been fully developed and tested in the simulated world. A minimalist robot was fabricated and tested with only one behavior. The results of the real robot with one behavior directly coincided with the results of the simulation with one behavior. The real robot also demonstrated the capability of the IRA to handle intermittent failures. The vibrating beam demonstration showed how this behavior-based definition of error might be applied to other systems.

The above demonstrations all demonstrated the capability of the IRA to detect and correct the stuck or disconnected sensor. Will the IRA be able to detect and correct the case of a floating sensor or a bias error on the sensor? It is presumed that as long as the system has functional redundancy, it will detect the float of the sensor and then ignore it, allowing the other reactions to control the system. Bias error is avoidable by the designer, but again, if the bias error causes the system to lose its circularity, as long as there is functional redundancy the system will be able to maintain its circularity.

References

1. Alice 95 and 98, from http://dmtwww.epfl.ch/isr/asl/projects/alice_pj.html
2. Arkin, Ronald C., 1998, Behavior-Based Robotics, The MIT Press, Cambridge, MA.
3. Atkins, Ella M.; Durfee, Edmund H. and Kang G. Shin, 1997, *Avoiding Failure via Pre-Planned Responses and Time-Bounded Planning*. Proceedings of the 1997 14th National Conference on Artificial Intelligence, July 27 – 31, 1997, Providence, RI, p821.
4. Basic Stamp Microcontrollers, www.parallax.com
5. Baker, J. E., 1999, *Automated Integration of Multiple Sensors*, edited by Ghosh, Bijoy K., Xi, Ning and Tarn, T. J. Control in Robotics and Automation, Sensor-Based Integration, Academic Press, 525 B Street, Suite 1900, San Diego, CA, p 313 – 345.
6. Beaulah, S. A., and Z. S. Chalabi, 1997, *Intelligent Real-Time Fault Diagnosis of Greenhouse Sensors*, Control Engineering Practice, volume 5, number 11, p 1573 – 1580.
7. Beckerman, Martin and Obrow, E. M., 1990, *Treatment of Systematic Errors in the Processing of Wide-Angle Sonar Sensor Data for Robotic Navigation*, IEEE Transactions on Robotics and Automation, volume 6, number 2, 2 April 1990, p 137 – 145.
8. Brickhard, Mark H, 1999a, *Critical Principles: On the Negative Side of Rationality*, In Herfel, W. Hooker, C. A. (eds) Beyond Ruling Reason: Non-formal Approaches to Rationality. Available at: <http://www.lehigh.edu/~mhb0/mhb0.html>.
9. Brickhard, Mark H., 1999b, *Motivation and Emotion: An Interactive Process Model*, (in work). Available at: <http://www.lehigh.edu/~mhb0/mhb0.html>.
10. Brickhard, Mark, H., 1998, *Robots and Representation*, in Animals to Animats 5, edited by Rolf Pfeifer, Bruce Blumberg, Jean-Arcady Meyer and Stuart Wilson, Bradford Books, MIT Press, Cambridge, MA, p 58 – 63.
11. Brickhard, Mark H., 1999c, *Interaction and Representation*, Theory and Psychology, volume 9, number 4, p 435 – 458.
12. Brickhard, Mark H., 1999d, e-mail interview, 17 Oct 1999.

13. Brooks, Rodney A, 1987, *Planning is Just a Way of Avoiding Figuring Out What To Do Next*, Working Paper 303, Massachusetts Institute of Technology.
14. Brooks, R. R. and S. S. Iyengar, 1995a, *Methods of Approximate Agreement for Multisensor Fusion*. Proceedings of SPIE, Signal Processing, Sensor Fusion, and Target Acquisition IV, 17 – 19 Apr 1995, volume 2484, p37 - 44.
15. Brooks, R. R. and S. S. Iyengar., 1995b, *Robot Algorithm Evaluation by Simulating Sensor Faults*. Proceedings of SPIE, Signal Processing, Sensor Fusion, and Target Acquisition IV, 17 – 19 Apr 1995, volume 2484, p394 – 401.
16. Cambridge Engineering Database
17. Chakrabarti, Amaresh, 1992, *Functional Reasoning in Design: Function as a Common Representation for Design Problem Solving*. Understanding Function and Function-To-Form Evolution. Cambridge University Engineering Department, p 79 - 88.
18. Casti, John L, Alternate Realities, 1989, Mathematical Models of Nature and Man, John Wiley and Sons, New York, NY.
19. Cherian, Sunil, 1995a, Design and Dynamics of Behavior Networks in Autonomous Robotic Systems, Ph.D. Thesis, Colorado State University, Fort Collins, CO.
20. Cherian, Sunil, 1995b, *Intelligent Behavior in Machines Emerging from a Collection of Interactive Control Structures*, Computational Intelligence, volume 11, number 4, Blackwell Publishers, p 565 - 592.
21. Danglemayr, Gerhard, 1999, Personal Interview, 9 December 1999, professor of Mathematics, Colorado State University.
22. De Benito, C.D. and Eckert, S.J., 1990a, *Control of Active Suspension System Subject to Random Component Failures*. Journal of Dynamic Systems, Measurement, and Control, volume 112, number 4, p 94 – 99.
23. De Benito, C.D. and Eckert, S.J., 1990b, *On-Board Real-Time Failure Detection and Diagnosis of Automotive Systems*. Journal of Dynamic Systems, Measurement, and Control, volume 112, number 4, p 769 - 773.
24. Ferrel, Cynthia, 1993, Robust Agent Control of an Autonomous Robot with Many Sensors and Actuators, Master of Science Thesis, Massachusetts Institute of Technology.
25. Firby, R. J., Prokopowicz, P. N. and Swain, M. J., 1998, *The Animate Agent Architecture*, in Kortenkamp, David, R. Peter Bonasso and Robin Murphy, Artificial Intelligence and Mobile Robots, Case Studies of Successful Robot Systems, AAAI Press, 445 Burgesss Dr, Menlo Park, CA, 94025, p 243 – 275.

26. Fliescher, Jason G. and Troxell, Wade O., 1999, *Bio-Mimicry as a Tool in the Design of Robotic Systems*, in press.
27. Flynn, Anita M., Brooks, Rodney A, et al, 1989, *Intelligence for Miniature Robots, Sensors and Actuators*, volume 20, pages 187 – 196.
28. Ford, Paul H, 1996, Description of a Robot, Task and Environment Using the Theory of Affordances, Master's Thesis, Colorado State University, Fort Collins, CO.
29. Ford, Kenneth; Glymour, Clark and Patrick Hayes, 1995, *Introduction*, In Android Epistemology, edited by Ford, Kenneth; Glymour, Clark and Patrick Hayes, AAAI press, Menlo Park, CA, p xi – xvii.
30. French, Michael J, 1994, Invention and Evolution, Design in Nature and Engineering, 2ed, Cambridge University Press, Cambridge, England.
31. Gadanho, Sandra Clara and John Hallam, 1998, *Emotion-Driven Learning for Animat Control*, in Animals to Animats 5, edited by Rolf Pfeifer, Bruce Blumberg, Jean-Arcady Meyer and Stuart Wilson, Bradford Books, MIT Press, Cambridge, MA, p 354 – 359.
32. Gat, Erann, 1998, *Three-Layer Architectures*, in Kortenkamp, David, R. Peter Bonasso and Robin Murphy, Artificial Intelligence and Mobile Robots, Case Studies of Successful Robot Systems, AAAI Press, Menlo Park, CA, p 193 – 210.
33. Gill, Arthur, 1962, Introduction to the Theory of Finite State Machines, McGraw-Hill Book Company, Inc, New York.
34. Godambe, Nihal J. and C. J. Shi, 1998, *Behavioral Level Noise Modeling and Jitter Simulation of Phase-Locked Loops with Faults Using VHDL-AMS*, Journal of Electronic Testing: Theory and Applications, volume 13, number 1, p 7 – 17.
35. Gujar, R.K. and J.L. Sanders, 1994, *Analytical Models to Study the Impact of Error Detection and Recovery on the Performance of a Robotic Assembly Cell*, International Journal of Production Research, volume 32, number 4, p 769 – 785.
36. Gupta, Indranil, 1999, Notes on The Byzantine Generals Problem. 8 March 1999. <http://www.cs.cornell.edu/cs614-sp99/notes99/byzantine.html/>
37. Hamel, William R., 1999, *Sensor-Based Planning and Control in Telerobotics*, edited by Ghosh, Bijoy K., Xi, Ning and Tarn, T. J. Control in Robotics and Automation, Sensor-Based Integration, Academic Press, San Diego, CA, p 285 – 309.
38. Hogg, Robert V. and Allen T. Craig, 1978, Introduction to Mathematical Statistics, 4th Edition, Macmillan Publishing Co., Inc., New York.

39. Hoppenot, P.; and E. Colle, 1998, *Real-time Localization of a Low-cost Mobile Robot with Poor Ultrasonic Data*, Control Engineering Practice, volume 6, number 8, p 925 – 934.
40. Jemmy and Inchy, from <http://diwww.epfl.ch/lami/mirobots/lcubes.html>
41. Jones, Joseph L. and Anita M. Flynn, 1993, Mobile Robots Inspiration to Implementation, A. K. Peters, Ltd., Wellesley, MA.
42. Jones, William J, Capt, 1990, USAF, Structures Flight Test Handbook, AFFTC-TIH-90-001, November 1990.
43. Kahn, A. F. Ulrich, 1995, *The Ethics of Autonomous Learning Systems*, In Android Epistemology, edited by Ford, Kenneth; Glymour, Clark and Patrick Hayes, AAAI press, Menlo Park, CA, p 253 – 266.
44. Karnopp, Dean and Ronald Rosenberg, 1975, System Dynamics: A Unified Approach, John Wiley and Sons, New York.
45. Kesteloot, Lawrence, 1995, Byzantine Failures in Synchronous Systems. 20 Jan 1995, [http://www.alt.net/~lk/290 paper/node7.html](http://www.alt.net/~lk/290%20paper/node7.html).
46. Kortenkamp, David, R.; Huber, Marcus; Cohen, Charles; Raschke, Ulrich; Koss, Frank and Clare Congdon, 1998, *Integrating High-speed Obstacle Avoidance, Global Planning, and Vision Sensing on a Mobile Robot*. in Kortenkamp, David, R. Peter Bonasso and Robin Murphy, Artificial Intelligence and Mobile Robots, Case Studies of Successful Robot Systems, AAAI Press, Menlo Park, CA, p 53 – 72.
47. Kurzweil, Ray, 1999, The Age of Spiritual Machines, Viking Penguin a member of Penguin Books Ltd, Hammondsworth, Middlesex, England.
48. Landers, Richard R., 1963, Reliability and Product Assurance, A Manual for Engineering and Management, Prentice-Hall, Inc., Englewood Cliffs, NJ.
49. Lee, Sukhan and Sookwang Ro, 1999, *Robotics with Perception and Action Nets*, in edited by Ghosh, Bijoy K., Xi, Ning and Tarn, T. J. Control in Robotics and Automation, Sensor-Based Integration, Academic Press, San Diego, CA, p 347 – 380.
50. Lent, Charles M. and Michael H. Dickinson, 1988, *The Neurobiology of Feeding in Leeches*, Scientific American, June 1988, p 78 – 83.
51. Lin, Shield B, and Wang, Sheng-Guo, 1995, *Robust Control Design for an Uncertain Nonlinear Robotic System*, Proceedings of the IEEE Conference On Decision and Control 2, 13 – 15 Dec 1995, IEEE Press, p 1014 – 1015.
52. Lockner P. and Hancock, P., 1990, *Redundancy in Fault-Tolerant Systems*, Mechanical Engineering, volume 112, number 5, p 76 – 83.

53. Mahadevan, Sridhar, Theocharous, Georgios and Nikfar Khaleeli, 1998, *Rapid Concept Learning for Mobile Robots*, Machine Learning, volume 31, numbers 1 - 3, p7 - 27.
54. Mathematica 3.0, Wolfram Research
55. Maes, Pattie, 1990, *Situated Agents can have Goals*, edited by Pattie Maes, Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, MIT Press, p 49 -70.
56. Maturana,, Humberto R. and Francisco J. Valera, 1980, Autopoiesis and Cognition, The Realization of the Living, D. Reidel Publishing Company, Boston.
57. Moller, Ralf, Labrinos, Dimitrios, Pfeifer, Rolf, Labhart, Thomas and Rudiger Wehner, 1998, *Modeling Ant Navigation with an Autonomous Agent*, in Animals to Animats 5, edited by Rolf Pfeifer, Bruce Blumberg, Jean-Arcady Meyer and Stuart Wilson, Bradford Books, MIT Press, Cambridge, MA, p 186 – 194.
58. Mosterman, P. and Gautman Biswas, 1997, *Monitoring, Predicting, and Fault Isolation in Dynamic Physical Systems*, AAAI '97, 14th National Conference on Artificial Intelligence, July 27 – 31, 1997, Providence, RI, Proceedings sponsored by AAAI, p 100 - 105.
59. Micro motors, www.micromo.com
60. McLurkin, James, 1999, *Using Cooperative Robots for Explosive Ordinance Disposal*, MIT Press, (in work).
61. Nehmzow, Ulrich, 2000, Mobile Robotics: A Practical Introduction, Springer-Verlag London Limited.
62. Nowack, Mark L., 1997, Design Guideline Support for Manufacturability. Ph.D. Thesis, Cambridge University Engineering Department, Cambridge, England.
63. O'Connor, R.F.; Baber, C.; Musri, M. and H. Ekerol, 1993, *Identification, classification and management of errors in automated component assembly*. International Journal of Production Research, volume 31, number 8, p 1853 – 1863.
64. Paasch, Robert K. and Alice M Agogino, 1993, *A Structural and Behavioral Reasoning System for Diagnosing Large-Scale Systems*, IEEE Expert, p 31 – 36.
65. Pau, L.F., 1981, Failure Diagnosis and Performance Monitoring. Marcel Dekker, Inc. New York.
66. Pfiefer, Rolf and Christain Scheier, 1997, *Implications for Embodiment for Robot Learning*, Proceedings of the Second Euromicor Wrokshop on Advanced Mobile Robots, Euorbot '97, 22 – 24 Oct 1997, IEEE Computer Society, p 38 – 43.

67. Pin, Francois G., 1999, *A Fuzzy Behaviorist Approach to Sensor-Based Reasoning and Robot Navigation*, edited by Ghosh, Bijoy K., Xi, Ning and Tarn, T. J., Control in Robotics and Automation, Sensor-Based Integration, Academic Press, San Diego, CA, p 381 – 418.
68. Rao, Rajeesh P. N. and Olac Fuentes, 1998, *Hierarchical Learning of Navigation Behaviors in an Autonomous Robot using a Predictive Sparse Distribution Memory*, Machine Learning, volume 31, numbers 1 - 3, p87 - 113.
69. Reason, James, 1990, Human Error, Cambridge University Press, New York.
70. Redfield, Robin C, 2000, Associate Professor United States Air Force Academy, Personal Interview, 5 July 2000.
71. Revel, A., Gaussier, P., Lepretre, S. and J.P. Banquet, 1998, *Planification versus Sensory-Motor Conditioning: What are the Issues*, in Animals to Animats 5, edited by Rolf Pfeifer, Bruce Blumberg, Jean-Arcady Meyer and Stuart Wilson, Bradford Books, MIT Press, Cambridge, MA, p 129 - 138.
72. Rosner, Bernard, 1990, Fundamentals of Biostatistics, 3rd Edition, PWS-Kent Publishing Company, Boston, Massachusetts.
73. Ruff, David N. and Robert K. Paasch, 1993, *Consideration of Failure Diagnosis in Conceptual Design of Mechanical Systems*, Design Theory and Methodology, ASME, p 175 – 187.
74. Sharpe, Titus and Babara Webb, 1998, *Simulated and Situated Models of Chemical Trail Following in Ants*, in Animals to Animats 5, edited by Rolf Pfeifer, Bruce Blumberg, Jean-Arcady Meyer and Stuart Wilson, Bradford Books, MIT Press, Cambridge, MA, p 195 – 204.
75. Smoovy Motors, www.smoovy.com
76. Stary, Chris and Marcus F Peschl, 1995, *Towards Constructivist Unification of Machine Learning and Parallel Distributed Processing*, In Android Epistemology, edited by Ford, Kenneth; Glymour, Clark and Patrick Hayes, AAAI press, Menlo Park, CA, p 183 – 215.
77. Steels, Luc, 1994, *Emergent Functionality in Robotic Systems Through On-Line Evolution*, edited by Brooks, Rodney A. and Maes, Pattie, Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, MIT Press/Bradford Books, p 8 – 14.
78. Stein, Lynn Andrea, 1995, *Imagination and Situated Cognition*, In Android Epistemology, edited by Ford, Kenneth; Glymour, Clark and Patrick Hayes, AAAI press, Menlo Park, CA, p 167 – 182.

79. Stevens, James, 1990, Intermediate Statistics A Modern Approach, Lawrence Erlbaum Associates, Hillsdale, NJ.
80. TechOptimizer 3.0
81. Thelen, Esther and Linda B. Smith, 1994, A Dynamic Systems Approach to the Development of Cognition and Action, Bradford Books, MIT Press, Cambridge, MA.
82. Toms, M. and Patrick, J., 1989, *Components of Fault Finding: Symptom Interpretation*, Human Factors, volume 31, number 4, p 465 – 483.
83. Troxell, Wade O., 1999a, ME 514 Lesson Notes, Colorado State University, Spring 1999.
84. Troxell, Wade O., 1999b, NP-Completeness, ME 514 Handout, Colorado State University, Spring 1999.
85. Umeda, Y.; Tomiyama, T. and Yoshikawa, H., 1991, *A Design Methodology for a Self-Maintenance Machine*. Design Theory and Methodology – DTM '91, DE-Vol 31, p 143 – 150, American Society of Mechanical Engineers, New York.
86. von Wichert, Georg, 1997, *Mobile Robot Localization Using a Self-organized Visual Environment Representation*. Euromicro Workshop on Advanced Mobile Robots (Eurobot '97), Oct 22 – 24, 1997, IEEE Computer Society, p 29 – 36.
87. Webster's II, New Riverside University Dictionary, The Riverside Publishing Company, Boston, 1994.
88. Winkless, Nels and Iben Browning, 1978, Robots On Your Doorstep (A Book About Thinking Machines), Robotics Press, Portland, OR.
89. Young, Warren C., 1989, Roark's Formulas for Stress and Strain, 6th edition, McGraw-Hill, Inc., NY.

Appendix A: IRA vs Subsumption

Systems Dynamics Model for Subsumption and the IRA

Pure Subsumption

Every iteration will generate a new set of random inputs for the system. This will allow for testing of all possible inputs. What will the system do? Where will the system go? This routine will save the inputs in a vector that can then be input into the IRA, so that both systems are getting the same inputs.

```
yspace = {{1, 0}, { $\frac{\sqrt{2}}{2}$ ,  $\frac{\sqrt{2}}{2}$ }, { $\frac{\sqrt{2}}{2}$ ,  $-\frac{\sqrt{2}}{2}$ },  
          {-1, 0}, {- $\frac{\sqrt{2}}{2}$ ,  $\frac{\sqrt{2}}{2}$ }, {- $\frac{\sqrt{2}}{2}$ ,  $-\frac{\sqrt{2}}{2}$ }, {0, 0}};  
path = {{0, 0}};  
ClearAll[a, b1, b2, b3, d1, d2, d3, B1, B2, B3, D1, D2, D3];  
dirx = 1;  
diry = 0;  
mag = 1;  
B1 = {};  
B2 = {};  
B3 = {};  
D1 = {};  
D2 = {};  
D3 = {};  
  
Do[  
  b1 = Random[Integer];  
  b2 = Random[Integer];  
  b3 = Random[Integer];  
  d1 = Random[Integer];  
  d2 = Random[Integer];  
  d3 = Random[Integer];  
  B1 = Flatten[{B1, b1}];  
  B2 = Flatten[{B2, b2}];  
  B3 = Flatten[{B3, b3}];  
  D1 = Flatten[{D1, d1}];  
  D2 = Flatten[{D2, d2}];
```

```

D3 = Flatten[{D3, d3}];

If[a == yspace[[4]], Goto[calc]];

If[path[[i]] != path[[i - 1]] && i != 1,
  mag =  $\sqrt{(\text{path}[[i, 1]] - \text{path}[[i - 1, 1]])^2 +$ 
     $(\text{path}[[i, 2]] - \text{path}[[i - 1, 2]])^2}$ ;
  dirx = (path[[i, 1]] - path[[i - 1, 1]]) / mag;
  diry = (path[[i, 2]] - path[[i - 1, 2]]) / mag];

If[a == yspace[[5]] || a == yspace[[6]], dirx = -dirx; diry = -diry];
Label[calc];

If[b1 == 1 || b2 == 1 || b3 == 1, Goto[escape],
  If[d1 == 1 || d2 == 1 || d3 == 1, Goto[avoid], Goto[wander]]];

Label[escape];
If[b1 == 1, a = yspace[[4]]];
If[b2 == 1, a = yspace[[3]]];
If[b3 == 1, a = yspace[[2]]];
If[b1 == 1 && b2 == 1, a = yspace[[6]]];
If[b1 == 1 && b3 == 1, a = yspace[[5]]];
If[b2 == 1 && b3 == 1, a = yspace[[1]]];
If[b1 == 1 && b2 == 1 && b3 == 1, a = yspace[[7]]];
Goto[drive];

Label[avoid];
If[d1 == 1, a = yspace[[3]]];
If[d2 == 1, a = yspace[[2]]];
If[d3 == 1, a = yspace[[1]]];
If[d1 == 1 && d2 == 1, a = yspace[[4]]];
If[d1 == 1 && d2 == 1 && d3 == 1, a = yspace[[7]]];
Goto[drive];

Label[wander];
a = yspace[[1]];
Goto[drive];

Label[drive];

c =
Partition[
  Flatten[{a[[1]] dirx - a[[2]] diry, a[[1]] diry + a[[2]] dirx}], 2];

d = Partition[Flatten[path[[i]]], 2] + c;
path = N[Partition[Flatten[{path, d}], 2]]
  , {i, 1, 5000}]

```

Here is 1000 iterations pure subsumption

Here is 5000 iteration pure subsumption

Now the IRA with only one connection between Freeze and Wander

The IRA will have the same inputs as the subsumption model of above.

```

ClearAll[a];
step = Length[B1];
path2 = {{0, 0}};
cnt = 0;
dirx = 1;
diry = 0;
mag = 1;

Do[
  If[a == yspace[[4]], Goto[calc]];

  If[path2[[i]] != path2[[i - 1]] && i != 1,
    mag =  $\sqrt{(\text{path2}[[i, 1]] - \text{path2}[[i - 1, 1]])^2 +$ 
       $(\text{path2}[[i, 2]] - \text{path2}[[i - 1, 2]])^2}$ ;
    dirx = (path2[[i, 1]] - path2[[i - 1, 1]]) / mag;
    diry = (path2[[i, 2]] - path2[[i - 1, 2]]) / mag];

  If[a == yspace[[5]] || a == yspace[[6]], dirx = -dirx; diry = -diry];

Label[calc];

If[B1[[i]] == 1 || B2[[i]] == 1 || B3[[i]] == 1,
  Goto[escape], If[D1[[i]] == 1 || D2[[i]] == 1 || D3[[i]] == 1,
    Goto[avoid], Goto[wander]]];

Label[escape];
If[B1[[i]] == 1, a = yspace[[4]]];
If[B2[[i]] == 1, a = yspace[[3]]];
If[B3[[i]] == 1, a = yspace[[2]]];
If[B1[[i]] == 1 && B2[[i]] == 1, a = yspace[[6]]];
If[B1[[i]] == 1 && B3[[i]] == 1, a = yspace[[5]]];
If[B2[[i]] == 1 && B3[[i]] == 1, a = yspace[[1]]];
If[B1[[i]] == 1 && B2[[i]] == 1 && B3[[i]] == 1,
  a = yspace[[7]]; cnt += 1];
If[a != yspace[[7]], cnt = 0];
If[cnt >= 2, Goto[wander]];
Goto[drive];

```

```

Label[avoid];
If[D1[[i]] == 1, a = yspace[[3]]];
If[D2[[i]] == 1, a = yspace[[2]]];
If[D3[[i]] == 1, a = yspace[[1]]];
If[D1[[i]] == 1 && D2[[i]] == 1, a = yspace[[4]]];
If[D1[[i]] == 1 && D2[[i]] == 1 && D3[[i]] == 1,
  a = yspace[[7]]; cnt += 1];
If[a != yspace[[7]], cnt = 0];
If[cnt >= 2, Goto[wander]];
Goto[drive];

Label[wander];
a = yspace[[Random[Integer, {1, 3}]]];
Goto[drive];

Label[drive];

c =
  Partition[
    Flatten[{a[[1]] dirx - a[[2]] diry, a[[1]] diry + a[[2]] dirx}], 2];

d = Partition[Flatten[path2[[i]]], 2] + c;
path2 = N[Partition[Flatten[{path2, d}], 2]]
, {i, 1, step}]

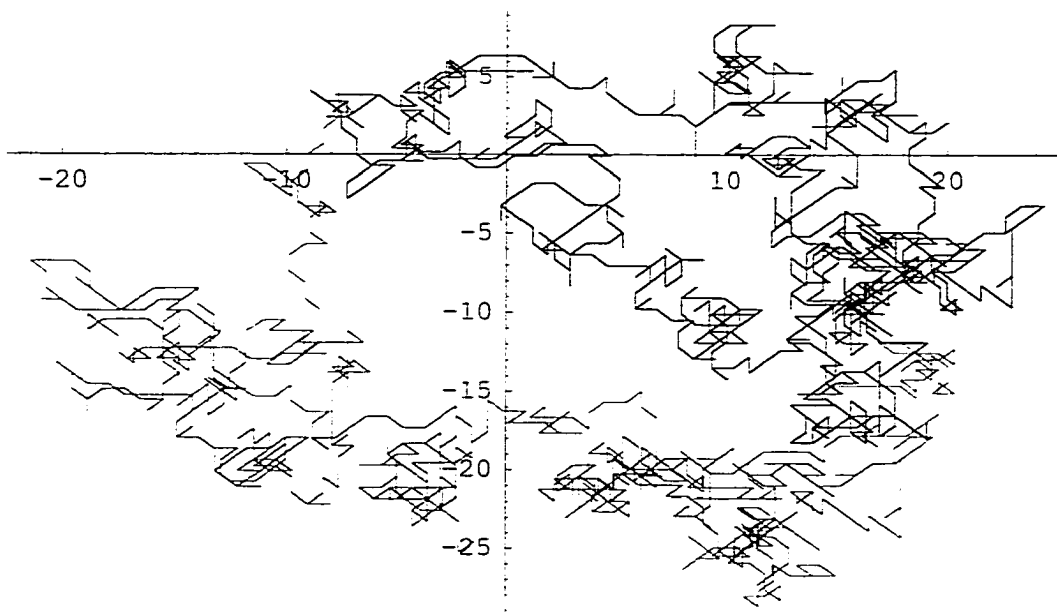
```

Here is 1000 Iterations of IRA

Here is 5000 iterations of IRA

The IRA travels further than subsumption for the same 1000 inputs.

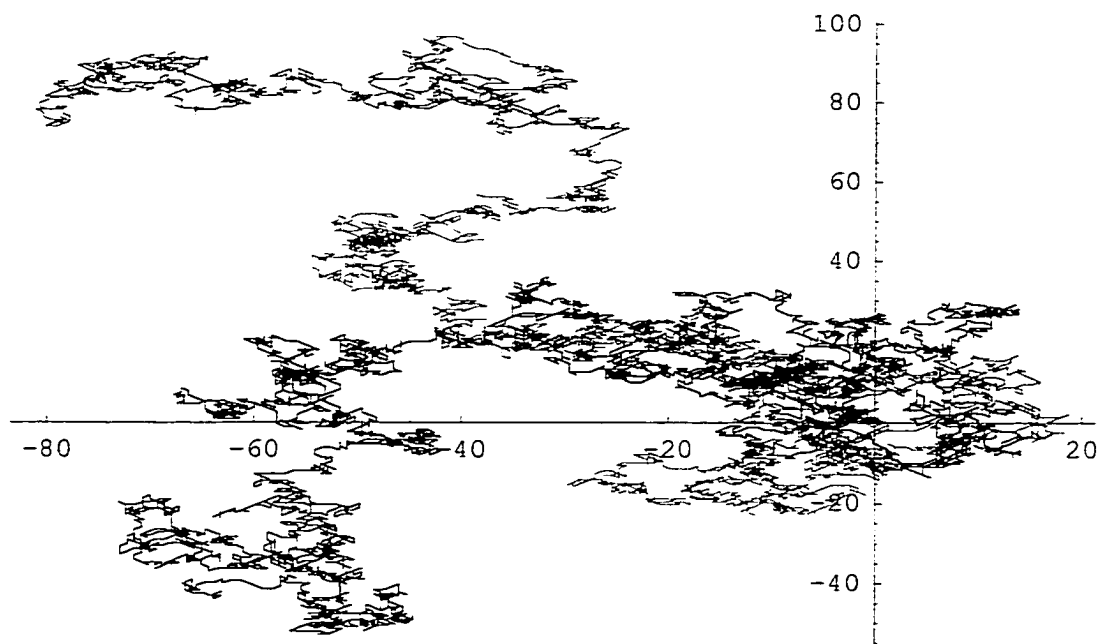
```
Show[%17, %15]
```



- Graphics -

5000 Iterations, again the IRA travels further for the same inputs.

Show[%22, %20]



- Graphics -

Appendix B: Goal Oriented Results

Systems Dynamics Model for Subsumption and the IRA for a goal oriented system.

Pure Subsumption

This routine uses a goal directed behavior. The system is supposed to reach "desired" with an allowable delta of two. The number of steps required to reach "desired" is then saved. This routine is run multiple times to generate a statistically significant data base.

```
yspace = {{1, 0}, { $\frac{\sqrt{2}}{2}$ ,  $\frac{\sqrt{2}}{2}$ }, { $\frac{\sqrt{2}}{2}$ ,  $-\frac{\sqrt{2}}{2}$ },  
          {-1, 0}, {- $\frac{\sqrt{2}}{2}$ ,  $\frac{\sqrt{2}}{2}$ }, {- $\frac{\sqrt{2}}{2}$ ,  $-\frac{\sqrt{2}}{2}$ }, {0, 0}};  
desired = {20, 20};  
path = {{0, 0}};  
ClearAll[a, b1, b2, b3, d1, d2, d3, B1, B2, B3, D1, D2, D3];  
dirx = 1;  
diry = 0;  
mag = 1;  
  
i = 1;  
delta = 2;  
j = 0;  
  
While[  
  (Abs[path[[i, 1]] - desired[[1]]] > delta ||  
    Abs[path[[i, 2]] - desired[[2]]] > delta),  
  
  If[Mod[i, 5] >= 3,  
    b1 = Random[Integer];  
    b2 = Random[Integer];  
    b3 = Random[Integer];  
    d1 = Random[Integer];  
    d2 = Random[Integer];  
    d3 = Random[Integer], b1 = 0; b2 = 0; b3 = 0; d1 = 0; d2 = 0; d3 = 0];  
  
  If[a == yspace[[4]], Goto[calc]];
```

```

If[path[[i]] != path[[i - 1]] && i != 1,
  mag =  $\sqrt{(\text{path}[[i, 1]] - \text{path}[[i - 1, 1]])^2 +$ 
     $(\text{path}[[i, 2]] - \text{path}[[i - 1, 2]])^2}$ ;
  dirx = (path[[i, 1]] - path[[i - 1, 1]]) / mag;
  diry = (path[[i, 2]] - path[[i - 1, 2]]) / mag];

If[a == yspace[[5]] || a == yspace[[6]], dirx = -dirx; diry = -diry];
Label[calc];

If[b1 == 1 || b2 == 1 || b3 == 1, Goto[escape],
  If[d1 == 1 || d2 == 1 || d3 == 1, Goto[avoid], Goto[goal]]];

Label[escape];
If[b1 == 1, a = yspace[[4]]];
If[b2 == 1, a = yspace[[3]]];
If[b3 == 1, a = yspace[[2]]];
If[b1 == 1 && b2 == 1, a = yspace[[6]]];
If[b1 == 1 && b3 == 1, a = yspace[[5]]];
If[b2 == 1 && b3 == 1, a = yspace[[1]]];
If[b1 == 1 && b2 == 1 && b3 == 1, a = yspace[[7]]];
Goto[drive];

Label[avoid];
If[d1 == 1, a = yspace[[3]]];
If[d2 == 1, a = yspace[[2]]];
If[d3 == 1, a = yspace[[1]]];
If[d1 == 1 && d2 == 1, a = yspace[[4]]];
If[d1 == 1 && d2 == 1 && d3 == 1, a = yspace[[7]]];
Goto[drive];

Label[goal];
j += 1;
Partition[Flatten[path[[i]]], 2];
dist = N[ $\sqrt{(\text{desired}[[1]] - \text{path}[[i, 1]])^2 +$ 
   $(\text{desired}[[2]] - \text{path}[[i, 2]])^2}$ ];
xterm = N[(desired[[1]] - path[[i, 1]]) / dist];
yterm = N[(desired[[2]] - path[[i, 2]]) / dist];

If[dirx > 0 && xterm > 0, If[Abs[yterm - diry] < .1, a = yspace[[1]],
  If[yterm > 0, a = yspace[[2]], a = yspace[[3]]]]];

If[dirx < 0 && xterm > 0, a = yspace[[2]]];

If[dirx > 0 && xterm < 0, a = yspace[[2]]];

```

```
Goto[drive];

Label[drive];

c =
Partition[
  Flatten[{a[[1]] dirx - a[[2]] diry, a[[1]] diry + a[[2]] dirx}], 2];

d = Partition[Flatten[path[[i]]], 2] + c;
path = N[Partition[Flatten[{path, d}], 2]];
i += 1;]
subs = Flatten[{subs, i}]

{437, 257, 122, 92, 298, 758, 129, 598, 149, 471, 414, 193, 188, 384,
  663, 344, 88, 363, 6260, 321, 837, 174, 582, 128, 1281, 162,
  329, 508, 283, 263, 854, 79, 52, 91, 38, 306, 1951, 168, 204, 1122}

subs = {};

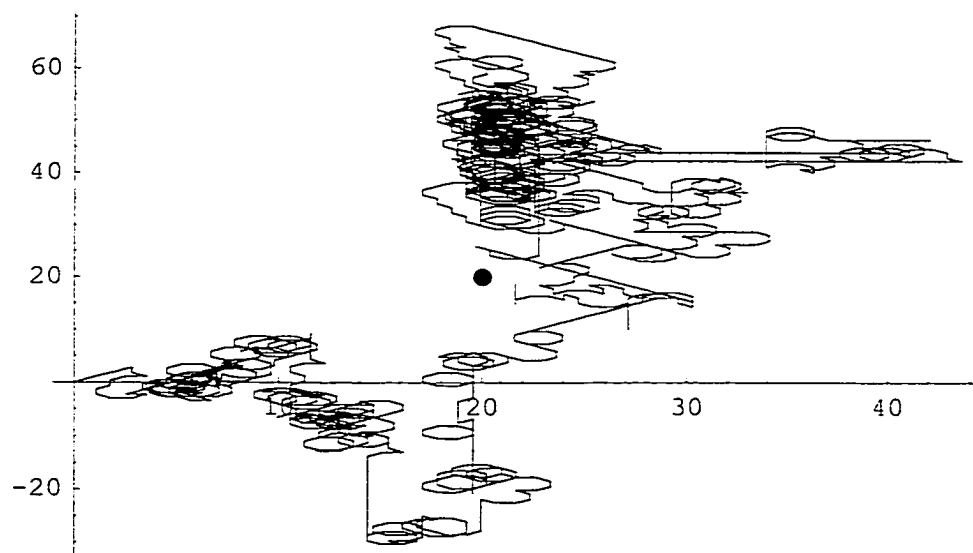
j

222

ListPlot[path, PlotJoined -> True, PlotStyle -> RGBColor[0, 0, 1]]
ListPlot[{desired}, PlotStyle -> PointSize[0.02]]
```

This is one example of subsumption. It took 1147 steps to come within a delta of 2 of "desired".

Show[%26, %30]



- Graphics -

The IRA

Same procedure as above

```

ClearAll[a, xterm, yterm];
path2 = {{0, 0}};
cnt = 0;
dirx = 1;
diry = 0;
mag = 1;
delta = 2;
i = 1;
j = 0;

While[
  (Abs[path2[[i, 1]] - desired[[1]]] > delta ||
   Abs[path2[[i, 2]] - desired[[2]]] > delta),

  If[Mod[i, 5] >= 3,
    b1 = Random[Integer];
    b2 = Random[Integer];
    b3 = Random[Integer];
    d1 = Random[Integer];
    d2 = Random[Integer];
    d3 = Random[Integer], b1 = 0; b2 = 0; b3 = 0; d1 = 0; d2 = 0; d3 = 0];

    If[a == yspace[[4]], Goto[calc]];

    If[path2[[i]] != path2[[i - 1]] && i != 1,
      mag =  $\sqrt{(\text{path2}[[i, 1]] - \text{path2}[[i - 1, 1]])^2 +$ 
         $(\text{path2}[[i, 2]] - \text{path2}[[i - 1, 2]])^2}$ ;
      dirx = (path2[[i, 1]] - path2[[i - 1, 1]]) / mag;
      diry = (path2[[i, 2]] - path2[[i - 1, 2]]) / mag];

    If[a == yspace[[5]] || a == yspace[[6]], dirx = -dirx; diry = -diry];

  Label[calc];

  If[b1 == 1 || b2 == 1 || b3 == 1, Goto[escape],
    If[d1 == 1 || d2 == 1 || d3 == 1, Goto[avoid], Goto[goal]]];

  .. Label[escape];
  If[b1 == 1, a = yspace[[4]]];

```

```

If[b2 == 1, a = yspace[[3]]];
If[b3 == 1, a = yspace[[2]]];
If[b1 == 1 && b2 == 1, a = yspace[[6]]];
If[b1 == 1 && b3 == 1, a = yspace[[5]]];
If[b2 == 1 && b3 == 1, a = yspace[[1]]];
If[b1 == 1 && b2 == 1 && b3 == 1, a = yspace[[7]]; cnt += 1];
If[a != yspace[[7]], cnt = 0];
If[cnt >= 2, Goto[goal]];
Goto[drive];

Label[avoid];
If[d1 == 1, a = yspace[[3]]];
If[d2 == 1, a = yspace[[2]]];
If[d3 == 1, a = yspace[[1]]];
If[d1 == 1 && d2 == 1, a = yspace[[4]]];
If[d1 == 1 && d2 == 1 && d3 == 1, a = yspace[[7]]; cnt += 1];
If[a != yspace[[7]], cnt = 0];
If[cnt >= 2, Goto[goal]];
Goto[drive];

Label[goal];
j += 1;
Partition[Flatten[path2[[i]]], 2];
dist = N[Sqrt[(desired[[1]] - path2[[i, 1]])^2 +
              (desired[[2]] - path2[[i, 2]])^2)];
xterm = N[(desired[[1]] - path2[[i, 1]]) / dist];
yterm = N[(desired[[2]] - path2[[i, 2]]) / dist];

If[dirx > 0 && xterm > 0, If[Abs[yterm - diry] < .1, a = yspace[[1]],
  If[yterm > 0, a = yspace[[2]], a = yspace[[3]]]]];

If[dirx < 0 && xterm > 0, a = yspace[[2]]];

If[dirx > 0 && xterm < 0, a = yspace[[2]]];

Goto[drive];

Label[drive];

c =
  Partition[
    Flatten[{a[[1]] dirx - a[[2]] diry, a[[1]] diry + a[[2]] dirx}], 2];

d = Partition[Flatten[path2[[i]]], 2] + c;
path2 = N[Partition[Flatten[{path2, d}], 2]];
i += 1;]
comm = Flatten[{comm, i}]

```

```
{96, 123, 153, 58, 663, 401, 193, 216, 163, 41, 128, 178, 557, 174,
 150, 383, 354, 300, 451, 82, 154, 156, 277, 208, 284, 525,
 894, 280, 267, 640, 150, 87, 142, 92, 134, 182, 247, 953, 223, 177}
```

```
comm = {};
```

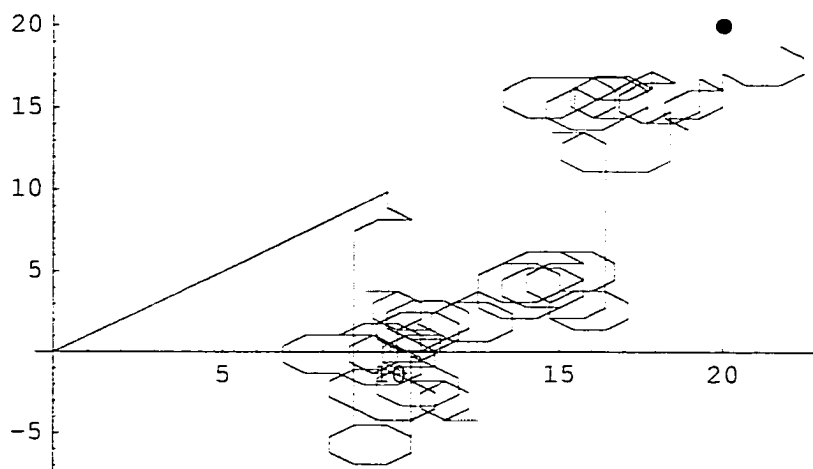
```
j
```

```
109
```

```
ListPlot[path2, PlotJoined -> True, PlotStyle -> RGBColor[1, 0, 0]]
```

The IRA achieved "desired" in only 177 steps here.

```
Show[%35, %30]
```



```
- Graphics -
```

Look at the statistics.

```
<< Statistics`DescriptiveStatistics`
```

```

Mean[comm] // N
StandardDeviation[comm] // N
Variance[comm] // N
Mean[subs] // N
StandardDeviation[subs] // N
Variance[subs] // N

273.4

214.827

46150.5

548.525

1001.94

 $1.00389 \times 10^6$ 

<< Statistics`HypothesisTests`

MeanDifferenceTest[comm,
  subs, 0, KnownVariance -> {46150.5,  $1.00389 \times 10^6$ },
  SignificanceLevel -> .05, FullReport -> True] //
N

{FullReport ->
  MeanDiff      TestStat      Distribution
  -275.125      -1.69808      NormalDistribution[]
  OneSidedPValue -> 0.0447467,
  Reject null hypothesis at significance level -> 0.05}

```

Appendix C: Minimalist Robot Control Code

```
lpres var byte
rpres var byte      'Photo resistor activation
mpres var byte

lset var byte
rset var byte      'Photo resistor ambient settings
mset var byte

rbumper var bit      'Bumper activation
lbumper var bit

myrand var word      'Random number generator

lpcnt var byte
rpcnt var byte      'Photo resistor counters
mpcnt var byte

rbcnt var nib        'Bumper counters
lbcnt var nib

rbflag var bit        'Bumper flags
lbflag var bit

lflag var bit        'Photo resistor flags
rflag var bit
mflag var bit

setter var nib        'Used to set the average light levels

momentum var nib      'Used for follow wall
direction var bit

maxsteps var word     'To control number of iterations

OUTPUT 0
OUTPUT 1
OUTPUT 2
OUTPUT 3
INPUT 7
INPUT 8
```

```

OUT0 = 0          'This sets the motor drivers as outputs
OUT1 = 0
OUT2 = 0
OUT3 = 0
OUT10 = 0

lpcnt = 0
rpcnt = 0
mpcnt = 0        'Initialize all counters to zero
rbcnt = 0
lbcnt = 0

myrand = 0       'Random number generator requires a start
                  point

direction = 0
momentum = 0

maxsteps = 500   'Number of iterations
out10 = 0
PHOTORESET:

for setter = 0 to 3

  high 4
  pause 1
  RCTIME 4,1,lset

  high 5
  pause 1
  RCTIME 5,1,rset   'This sets the ambient readings
                    'for the photoresistors

  high 6
  pause 1
  RCTIME 6,1,mset

  if not setter = 0 then average
  moveon:
  OUT0 = 0
  OUT1 = 1
  OUT2 = 1
  OUT3 = 0
  pause 500
  OUT1 = 0
  OUT2 = 0
  pause 500
NEXT

```

```

pause 2000          'Time to move away

start:              'Actual control loop start point

maxsteps = maxsteps - 1
if maxsteps = 0 then FINISHED
gosub getrandom

high 4
pause 1
RCTIME 4,1,lpres

high 5
pause 1
RCTIME 5,1,rpres    'This reads each of the photoresistors

high 6
pause 1
RCTIME 6,1,mpres

rbumper = in7       'This reads each of the bumpers
lbumper = in8

'This is AvoidBumping Activation

if rbumper then RBUMP
rbcnt = 0
rbflag = 0
if (OUT10 = 1) then RBCR
out10 = 0
RBCR:

if lbumper then LBUMP
lbcnt = 0
lbflag = 0
if (OUT10 = 1) then LBCR
out10 = 0
LBCR:
debug ? lbcnt
if (rbflag or lbflag) then AVOIDBUMPING

'This is AvoidObstacles Activation

if (lpres > (lset + 60)) then LCOUNT
lpcnt = 0
lflag = 0
if (OUT10 = 1) then LCR

```

```

low 10
LCR:

if (rpres > (rset + 60)) then RCOUNT
rpcnt = 0
rflag = 0
if (OUT10 = 1) then RCR
low 10
RCR:

if (mpres > (mset + 90)) then MCOUNT
mpcnt = 0
mflag = 0
if (OUT10 = 1) then MCR
low 10
MCR:

if (lflag or rflag or mflag) then AVOIDTASK

'This is Follow Wall Activation

if momentum = 0 then EXPLORE
momentum = momentum - 1
if momentum > 0 then FOLLOW

GOTO EXPLORE

AVOIDBUMPING:

if (rbflag and lbflag) then BACKTURN
if rbflag then BRTURN
if lbflag then BLTURN

AVOIDTASK:

if (((lflag=1) and (rflag=1) and (mflag=1)) and (myrand >
13107)) then BRTURN
if (((lflag=1) and (rflag=1) and (mflag=1)) and (myrand <
13106)) then BLTURN
if ((lflag=1) and (mflag=1)) then RTURN
if ((rflag=1) and (mflag=1)) then LTURN
if (((rflag=1) and (lflag=1)) or (mflag=1)) and myrand >
6554) then RTURN
if (((rflag=1) and (lflag=1)) or (mflag=1)) and myrand <
6553) then LTURN
if (lflag=1) then VRIGHT
if (rflag=1) then VLEFT

```

FOLLOW:

```
if direction = 0 then VLEFT
if direction = 1 then VRIGHT
```

EXPLORE:

```
if (myrand < 5000) then VRIGHT
if (myrand < 15000) then VLEFT
GOTO FWD
```

FWD:

```
OUT0 = 1
OUT1 = 0
OUT2 = 0           'Forward Skill
OUT3 = 0
GOTO start
```

RVS:

```
OUT0 = 0
OUT1 = 1
OUT2 = 0           'Reverse Skill
OUT3 = 0
GOTO start
```

RTURN:

```
OUT0 = 1
OUT1 = 0
OUT2 = 1           'Right Turn Skill
OUT3 = 0
GOTO start
```

LTURN:

```
OUT0 = 1
OUT1 = 0
OUT2 = 0           'Left Turn Skill
OUT3 = 1
GOTO start
```

VLEFT:

```
OUT0 = 1
OUT1 = 0
OUT2 = 0
OUT3 = 1
GOTO start
```

```
VRIGHT:
OUT0 = 1
OUT1 = 0
OUT2 = 1
OUT3 = 0
GOTO start
```

```
BRTURN:
OUT0 = 0
OUT1 = 1
OUT2 = 0           'Back Right Skill
OUT3 = 1
PAUSE 750
GOTO start
```

```
BLTURN:
OUT0 = 0
OUT1 = 1
OUT2 = 1           'Back Left Skill
OUT3 = 0
PAUSE 750
GOTO start
```

```
HALT:
OUT0 = 0
OUT1 = 0
OUT2 = 0           'Stop Skill
OUT3 = 0
RETURN
```

```
FWDS:
OUT0 = 1
OUT1 = 0
OUT2 = 0           'Forward Subroutine
OUT3 = 0
RETURN
```

```
BACKS:
OUT0 = 0
OUT1 = 1
OUT2 = 0           'Reverse Subroutine
OUT3 = 0
RETURN
```

```
RTURNS:
OUT0 = 1
OUT1 = 0
```

```

OUT2 = 1                'Right Turn Subroutine
OUT3 = 0
RETURN

LURNS:
OUT0 = 1
OUT1 = 0
OUT2 = 0                'Left Turn Subroutine
OUT3 = 1
RETURN

BACKTURN:
gosub backs
if myrand > 30000 then timer
PAUSE 200
GOTO what                'Back and Turn Skill
timer:
PAUSE 350
what:
if myrand > 20000 then turner
gosub LURNS
if myrand > 30000 then trnltime
PAUSE 200
GOTO finish
trnltime:
PAUSE 350
GOTO finish
turner:
gosub RTURNS
if myrand > 30000 then trntime
PAUSE 200
GOTO finish
trntime:
PAUSE 350
finish:
GOTO start

LCOUNT:
if lpcnt = 30 then lignore
lpcnt = lpcnt + 1
lflag = 1
momentum = 15
direction = 0
Low 10
goto LCR                'Counts Left Photo Resistor
lignore:
lflag = 0

```

```

High 10
goto LCR

RCOUNT:
if rpcnt = 30 then rignore
rpcnt = rpcnt + 1
rflag = 1
Low 10
momentum = 15
direction = 1
goto RCR          'Counts Right Photo Resistor
rignore:
rflag = 0
High 10
goto RCR

MCOUNT:
if mpcnt = 40 then mignore
mpcnt = mpcnt + 1
mflag = 1
Low 10
goto MCR          'Counts Middle Photo Resistor
mignore:
mflag = 0
High 10
goto MCR

RBUMP:
if rbcnt = 5 then rbignore
rbcnt = rbcnt + 1
rbflag = 1
goto RBCR         'Counts Right Bumper
rbignore:
rbflag = 0
High 10
goto RBCR

LBUMP:
if lbcnt = 5 then lbignore
lbcnt = lbcnt + 1
lbflag = 1
goto LBCR         'Counts Left Bumper
lbignore:
lbflag = 0
High 10
goto LBCR

```

```
GETRANDOM:
RANDOM myrand
RETURN
```

```
AVERAGE:
mset = (mset*setter + mpres)/(setter + 1)
lset = (lset*setter + lpres)/(setter + 1)
rset = (rset*setter + rpres)/(setter + 1)
GOTO moveon
```

```
FINISHED:
out0 = 0
out1 = 0
out2 = 0
out3 = 0
out10 = 1           'Signals end of control
pause 1000
out10 = 0
pause 1000
out10 = 1
```

Appendix D: AnSim Users Manual

The AnSim simulation was developed for rapid prototyping and testing of control architectures in a controlled environment. The code did not come with a users manual. This users manual will allow a designer to quickly modify the animat, behavior and environment. AnSim is written in C++ and will run on any IBM compatible PC. The only way to have the program fulfill its intended purpose is through the development of a users manual. The goal of this users manual is to explain the pertinent components of the code so that the code can be modified more easily.

Basics

When compiled the code generates an executable. The icon associated with the executable is the standard MFC icon. This can easily be changed, but there was no real interest in creating a clever icon. Once the icon is created the executable file can be moved to any directory and executed by double clicking on the icon. The necessary input files must be in the same directory as the executable file. The input files can be altered at any time without the need for recompiling the code. Animats can be added and removed, the environment can be changed and the behaviors can be changed. If the controller is changed, then the program will have to be recompiled.

The main components of interest are the input files, Animat.cpp, and Controller.cpp. Each of these programs and how to modify the input files will be briefly detailed. First the input files are discussed.

Input Files

The overall program requires a minimum of four input files: world.dat, animat.dat, controller.dat and behaviors.dat. Controller.dat may call more than one behavior. World.dat specifies which animat to use, animat.dat. Animat.dat specifies which controller will be used, controller.dat and controller.dat calls out the necessary behavior/reaction data files.

World.dat

When executed the program will first look to world.dat. This data file will supply all of the information required for the environment and it will also provide the program with the data file for an animat. The world.dat file tells the program which animat to use. A “%” tells the program not to read that line. Blocks are obstacles for animats. The ID is just a number for each block. The X-Y coordinates are the centers of the blocks. The length is the overall length of block. The orientation is the angle, where 0° is horizontal and 90° is vertical. Color is not important it is hard-wired in the Env.cpp code. The blocks are drawn in black. The next segment puts in the nuggets. They are drawn as small blocks, so there is no length. Color is not important here as well. The nuggets are drawn in brown. The next segment can be used to add a light source. The station segment draws a recharge station. Color is important here. The light detectors associated with the recharge station will look for “mid-color”. The shape of the station is hard-wired in the code. The same is true for the home station for shape and color. A typical world.dat file is provided.

```

% WINDOW NAME DIMENSION TRACE
      WI AnSim      400      0
% ANIMAT FILE
      AN animat.dat
% AN animat2.dat
% AN animat3.dat
%
% BLOCK ID X Y LENGTH WIDTH ORIENT CHARGE COLOR
      BL 1 48 40 19 2 80 G 0
      BL 2 65 75 30 2 0 G 0
      BL 3 65 27 30 2 0 G 0
      BL 4 48 62 19 2 -80 G 0
% BL 1 50 50 30 3 0 G 0
% BL 2 50 70 30 3 0 G 0
% BL 3 66.5 60 23 3 90 G 0
% BL 4 50 85.75 28.5 3 90 G 0
%
% NUGGET ID X Y ORIENT COLOR
      NU 1 10 60 20 0
      NU 2 10 70 -20 5
      NU 3 5 90 0 5
      NU 4 15 65 45 5
      NU 5 20 85 10 5
      NU 6 50 90 60 5
      NU 7 40 85 15 5
      NU 8 30 80 80 5
%
% LIGHT ID X Y RADIUS COLOR
% LI 1 10 10 1 0
%
% STATION ID X Y ORIENT COLOR
      ST 1 20 50 0 1
% ST 2 30 30 0 1
%
% HOME ID X Y ORIENT MID-COLOR OUT-COLOR
% HO 1 10 10 0 3 2
      HO 2 80 50 0 3 2

```

This is just an example. The figure below gives an example of all of these elements in one environment.

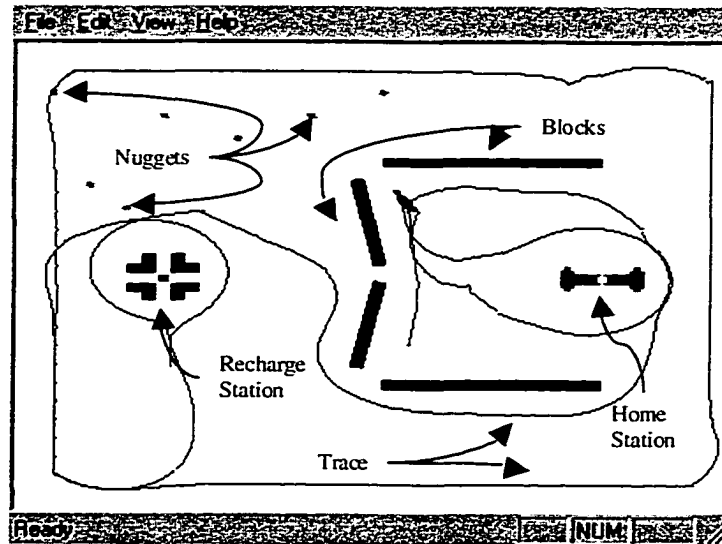


Figure D.1, Typical World

Animat.dat

World.dat calls out which animat file to use. The animat.dat file is setup for two whiskers, three IR sensors, eight light detectors, four nugget detectors, one charge detector and one stuck detector. The animat can be modified to have up to twenty-seven sensors, but only three types can be changed. There can be up to four whiskers, four IR sensors, and ten light detectors. Of course, if the animat is changed, then the user will have to change the controller code to let the program know what the animat is supposed to do for each additional sensor. All of the parameters associated with each sensor can be changed without requiring modification of the control code. A typical animat file is provided below.

```

% ANIMAT id x y minenergy maxenergy trail(boolean)
AN 1 21 30 4000 5000 1
% WHISKER id x y minorient maxorient step minrange maxrange step
WH 1 -1 2.5 -35 0 0 3 0 0
WH 2 1 2.5 35 0 0 3 0 0
% I-RED id x y minorient maxorient step minrange maxrange step spread
IR 1 0 2.5 0 0 0 6.25 0 0 30
IR 2 -1 2.5 -30 0 0 6 0 0 35
IR 3 1 2.5 30 0 0 6 0 0 35
% LIGHT D id x y color minorient maxorient step minsprd maxsprd step
LD 1 -1 2.0 1 -90 0 0 60 0 0
LD 2 1 2.0 1 90 0 0 60 0 0
%
LD 3 0 2.5 3 0 0 12 25 0 0
LD 4 -0.6 2.5 3 -30 0 12 80 0 0
LD 5 0.6 2.5 3 30 0 12 80 0 0
%
LD 6 0 2.5 1 0 0 12 25 0 0
LD 7 -1 2.5 1 -30 0 12 70 0 0
LD 8 1 2.5 1 30 0 12 70 0 0
% NUGGET D id x y
ND 1 -.4 1.6
ND 2 0.3 1.5
ND 3 0.3 1.9
ND 4 -0.3 1.9
% CHARGE D id x y leftWhId rightWhId
CD 1 0 2.5 1 2
% STUCK DETECT
SD 1 0 2 5
% DR controller
CO compoundbeh.dat
% CO charge2.dat

```

The first section defines the minimum energy and maximum energy for the animat as well as whether to draw a trace or not. The rest develops all of the sensors that the animat possesses where they are located, and what they are looking for. The whiskers, IR and nugget detectors all make sense in what they do. Light detectors 1, 2, 6, 7 and 8 are all used to locate the recharge station. They are associated with color number 1 and if you notice the recharge station is also color number 1. Light detectors 3, 4 and 5 are used

to locate the home station. The mid color, center light source, is color number 3 as is the color for these light detectors. Finally, this input file calls out the appropriate type of controller and the data file associated with the animat. This one is calling out compoundbeh.dat.

Behavior.dat

Behavior.dat is used by the controller to pick out the appropriate reactions and how they are subordinated to each other. For simplicity a single behavior will be used here. This behavior uses the reactions: search, avoid obstacles, follow wall, avoid bumping, seek nugget and get nugget. The subsumption matrix is a fourteen by fourteen matrix that tells the controller which reaction subsumes control over another reaction. There are zeros down the diagonal since a reaction should not subsume itself. The rows and columns are in the following order: SE, AO, FW, AB, SN, GN, INN, SH, DH, IH, SC, DC, IC and MA. These are defined in defs.h in the code. The first row is search; it is the base reaction and does not subsume over any of the other reactions. The second row is avoid obstacles, it subsumes search. The rest follow. Avoid bumping subsumes all of the other reactions except get nugget, which requires the animat to drive over the nugget. The manipulation of this matrix is very important to proper operation of the animat. The Nugget.dat is presented below.

```

% num of beh.
NB 14
% specify component interactive control structures and indices.
% SE 0 Search
% AO 1 Avoid Obstacles
% FW 2 Follow Wall
% AB 3 Avoid Bumping
% SN 4 Seek nugget
% GN 5 Get Nugget
%
SE 0 0.16
AO 1
FW 2
AB 3
SN 4 0.05
GN 5
% denote the start of the subsumption matrix
% specify matrix; num of beh. X num of beh.
SM
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
%

```

If multiple behaviors are used, it is imperative that the user builds the subsumption matrix correctly. Errant behavior will result if the reactions are not correctly placed in the subsumption matrix. It is also imperative that the user understand how each reaction performs its function. For example, the dock charge reaction requires that one whisker contacts the negative block and the other whisker contacts a positive block. Therefore,

dock charge must subsume avoid bumping. Failure to do this will result in the inability of the animat to dock with the charging station.

Animat.cpp

Animat.cpp is the code used to generate the animat. This is also responsible for the movement of the animat based upon the controller that is built when the program is compiled. As written now, the animat has eight possible skills. The animat can move forward, move back, turn left, turn right, veer left, veer right, pick nugget, drop nugget and charge. A typical animat is shown in the figure below:

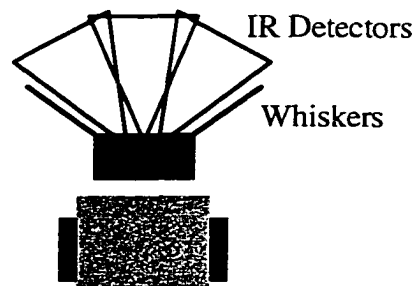


Figure D.2, Typical Animat

Controller.cpp

Controller.cpp builds up the control for the animat based upon the subsumption matrix provided in the input files. Each reaction has four main parts: activation, reset, task and execution. The ones that require modification, if desired, are activation and task. Activation determines if this reaction is active, the sensory input requires action on the part of this reaction. Task determines what the animat should do for the input string from the sensors.

Conclusion

The program is much easier to use if the users manual is consulted first. The code now has some comments built into it, but not nearly enough. The guidance from above should make it simple for someone relatively comfortable with C++ and versed in behavior-based robotics to modify for their needs.

Appendix E: *Mathematica* Beam Deflection Simulation

Dynamic Measurement of a deflecting beam

For this demonstration a simple cantilever beam will be used. Due to the variations in strain gauges, and the effects of temperature, installation and measurement, three gauges will be used to determine the maximum tip deflection.

To determine the maximum deflection, the tip deflection, interpolating functions need to be written for the strain given by each of the gauges. Determining the deflections for each position is very simple. For gauge 1, the deflection is:

$$v_1 = \frac{38.1944 \cdot 10^{-3} L^2}{y}$$

The other gauges are similar. It is also known that the root deflection is zero. Using these simple derivations, the required interpolating functions can be determined for all possible inputs; all gauges good, one gauge bad or two gauges bad. Three bad gauges will cause the program to fail.

The first interpolating function is for three good gauges, the most desired result.

```
l = 1; (* length of beam *)
y = .01; (* NA to outer fiber *)
x = {.25, .5, .75}; (* location of gauges *)
disp =  $\frac{x^2 (3 l - x)}{6 (1 - x) y}$ 
{3.81944, 20.8333, 84.375}

ClearAll[e1, e2, e3];
data = {{0, 0}, {.25, disp[[1]] e1}, {.5, disp[[2]] e2},
  {.75, disp[[3]] e3}}
{{0, 0}, {0.25, 3.81944 e1}, {0.5, 20.8333 e2}, {0.75, 84.375 e3}}
```

```

ClearAll[x];
allgood = Fit[data, {x2, x3}, x]

x2 (19.3494  $\epsilon$ 1 + 197.229  $\epsilon$ 2 - 252.506  $\epsilon$ 3) +
  x3 (-26.596  $\epsilon$ 1 - 260.799  $\epsilon$ 2 + 534.719  $\epsilon$ 3)

 $\epsilon$ 1 = .0075;
 $\epsilon$ 2 = .005;
 $\epsilon$ 3 = .0025;
x = 1;
allgood

0.333333

```

Now for gauge 1 being faulty:

```

ClearAll[ $\epsilon$ 1,  $\epsilon$ 2,  $\epsilon$ 3];
data = {{0, 0}, {.5, disp[[2]]  $\epsilon$ 2}, {.75, disp[[3]]  $\epsilon$ 3}}

{{0, 0}, {0.5, 20.8333  $\epsilon$ 2}, {0.75, 84.375  $\epsilon$ 3}}

ClearAll[x];
onebad = Fit[data, {x2, x3}, x]

x2 (250.  $\epsilon$ 2 - 300.  $\epsilon$ 3) + x3 (-333.333  $\epsilon$ 2 + 600.  $\epsilon$ 3)

 $\epsilon$ 2 = .005;
 $\epsilon$ 3 = .0025;
x = 1;
onebad

0.333333

```

Now for gauge 2 being faulty:

```

ClearAll[ $\epsilon$ 1,  $\epsilon$ 2,  $\epsilon$ 3];
data = {{0, 0}, {.25, disp[[1]]  $\epsilon$ 1}, {.75, disp[[3]]  $\epsilon$ 3}}

{{0, 0}, {0.25, 3.81944  $\epsilon$ 1}, {0.75, 84.375  $\epsilon$ 3}}

```

```

ClearAll[x];
twobad = Fit[data, {x2, x3}, x]

x2 (91.6667 ε1 - 75. ε3) + x3 (-122.222 ε1 + 300. ε3)

ε1 = .0075;
ε3 = .0025;
x = 1;
twobad

0.333333

```

Now for gauge 3 being faulty:

```

ClearAll[ε1, ε2, ε3];
data = {{0, 0}, {.25, disp[[1]] ε1}, {.5, disp[[2]] ε2}}

{{0, 0}, {0.25, 3.81944 ε1}, {0.5, 20.8333 ε2}}

ClearAll[x];
threebad = Fit[data, {x2, x3}, x]

x2 (122.222 ε1 - 83.3333 ε2) + x3 (-244.444 ε1 + 333.333 ε2)

ε1 = .0075;
ε2 = .005;
x = 1;
threebad

0.333333

```

For Just one good gauge:

```

ClearAll[ε1, ε2, ε3];
x = {.25, .5, .75};
maxdef =  $\frac{l^3}{3(1-x)y}$ 

{44.4444, 66.6667, 133.333}

```

Now for gauges 1 and 2 being faulty:

```
maxdef3 = maxdef[[3]] ε3
```

```
133.333 ε3
```

```
ε3 = .0025;
```

```
x = 1;
```

```
maxdef3
```

```
0.333333
```

Now for gauges 1 and 3 being faulty:

```
ClearAll[ε1, ε2, ε3];
```

```
maxdef2 = maxdef[[2]] ε2
```

```
66.6667 ε2
```

```
ε2 = .005;
```

```
x = 1;
```

```
maxdef2
```

```
0.333333
```

Now for gauges 2 and 3 being faulty:

```
ClearAll[ε1, ε2, ε3];
```

```
maxdef1 = maxdef[[1]] ε1
```

```
44.4444 ε1
```

```
ε1 = .0075;
```

```
x = 1;
```

```
maxdef1
```

```
0.333333
```

Now to perform the simulations. First the tip will have a known deflection. It will driven by a sinusoidal function.

```
tip = Table[ $\frac{1}{3}$  Sin[2  $\pi$  t], {t, 1, 12, .1}] // N;
ListPlot[tip, PlotJoined -> True, PlotStyle ->
  {Thickness[.008], RGBColor[1, 0, 0], Dashing[{.05, .05}]]]
```

Determine the strains for each location from the known deflection:

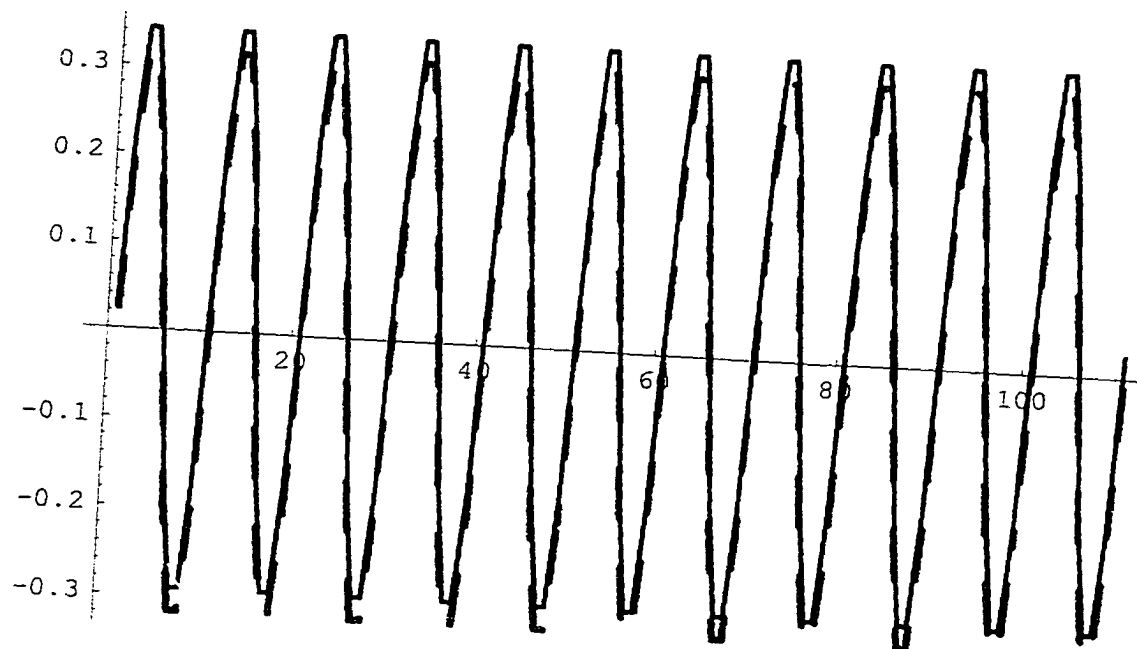
```
ClearAll[x, e1, e2, e3];
strain = p (1 - x) .01;
p = tip * 3;
x = .25;
e1 = strain + Random[] * .0001;
x = .5;
e2 = strain + Random[] * .0001;
x = .75;
e3 = strain + Random[] * .0001;
```

Now if all of the gauges are good:

```
x = 1;
ListPlot[allgood, PlotJoined -> True,
  PlotStyle -> {Thickness[.005], RGBColor[0, 0, 1]}]
```

Now overlay the exact tip and the calculated tip. With the slight deviation thrown in by the random number, the fit is still pretty good.

```
Show[%23, %25]
```



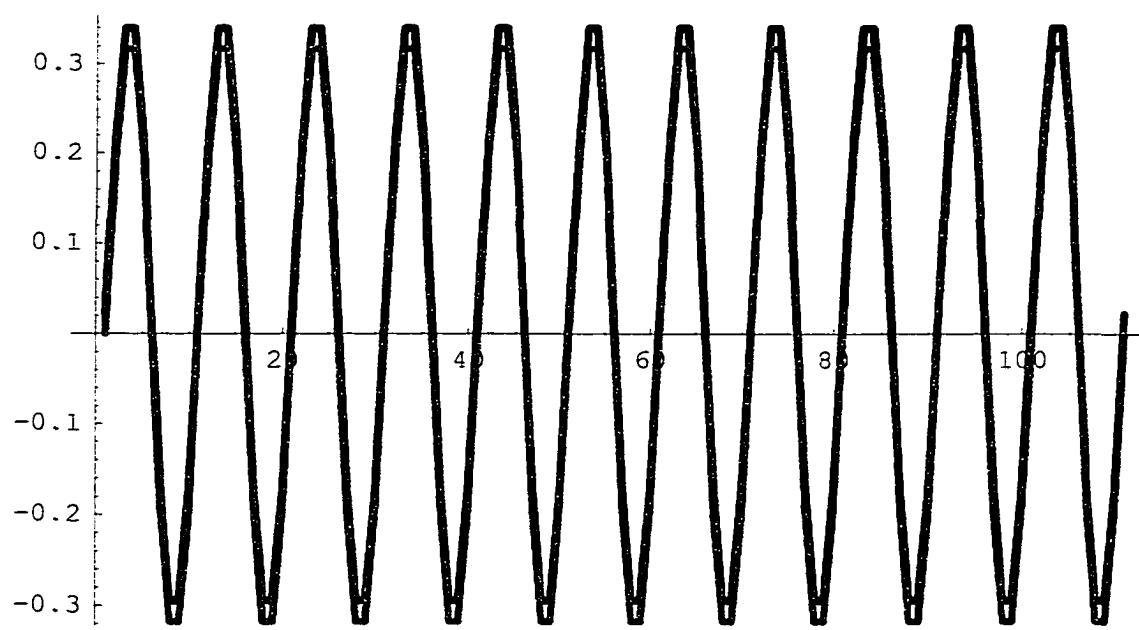
- Graphics -

Now for the first gauge being bad:

```
x = 1;
ListPlot[onebad, PlotJoined -> True,
PlotStyle -> {Thickness[.008], RGBColor[0, 0, 1]}]
```

The fit is still good for just one gauge bad and the random noise thrown in.

Show[%35, %31]



- Graphics -

Now for gauges 2 and 3 bad:

With just one gauge nearly an exact fit.

Behavior method to determine the tip deflection. This will have only one behavior, but it will have seven reactions to determine the tip deflection. It can use all of the gauges, two of the gauges or only one of the gauges. For this simulation, there will be the case of losing gauge 1 and the case of losing gauges 1 and 2.

Demonstration 1: All gauges are functional.

```

cnt1 = 0;
cnt2 = 0;
cnt3 = 0;
holder1 = 0;
holder2 = 0;
holder3 = 0;
deflection = Table[0, {i, 1, 101}];

Do[
ClearAll[x, e1, e2, e3];
strain = p (1 - x) .01;
p =  $\frac{1}{3}$  Sin[2  $\pi$  i (0.1)] * 3;
x = .25;
strain1 = strain + Random[] * .0001;
x = .5;
strain2 = strain + Random[] * .0001;
x = .75;
strain3 = strain + Random[] * .0001;

e1 = strain1;
e2 = strain2;
e3 = strain3;
x = 1;

if[i == 1, Goto[threegood]];

if[strain1 == holder1,
{cnt1++, flag1 = 1}, {cnt1 = 0, flag1 = 0, holder1 = strain1}];
if[strain2 == holder2,
{cnt2++, flag2 = 1}, {cnt2 = 0, flag2 = 0, holder2 = strain2}];
if[strain3 == holder3,
{cnt3++, flag3 = 1}, {cnt3 = 0, flag3 = 0, holder3 = strain3}];

```

```

if[
  cnt1 > 2 && cnt2 > 2 && cnt3 > 2, {defholder = lastdef, Goto[update]}}];
if[cnt1 > 2 && cnt2 > 2, Goto[lastone]];
if[cnt1 > 2 && cnt3 > 2, Goto[middleone]];
if[cnt2 > 2 && cnt3 > 2, Goto[firstone]];
if[cnt1 > 2, Goto[lasttwo]];
if[cnt2 > 2, Goto[firstandlast]];
if[cnt3 > 2, Goto[firsttwo]];

Label[threegood];
defholder = allgood;
Goto[update];

Label[lastone];
defholder = maxdef3;
Goto[update];

Label[middleone];
defholder = maxdef2;
Goto[update];

Label[firstone];
defholder = maxdef1;
Goto[update];

Label[lasttwo];
defholder = onebad;
Goto[update];

Label[firstandlast];
defholder = twobad;
Goto[update];

Label[firsttwo];
defholder = threebad;
Goto[update];

Label[update];

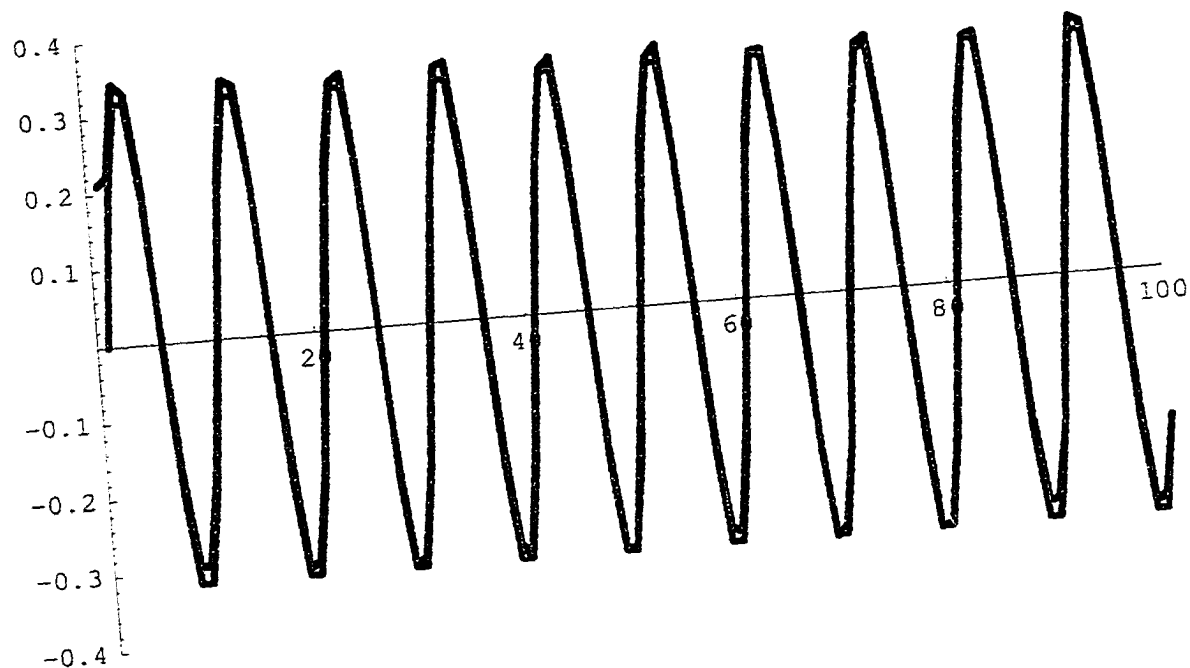
lastdef = defholder;
deflection[[i]] = defholder;
Flatten[deflection, {i, 1, 101}]

ListPlot[deflection, PlotJoined -> True,
  PlotStyle -> {Thickness[.008], RGBColor[0, 0, 1]}]

```

With the random noise, there is some error, but overall an excellent fit.

```
Show[%43, %31, PlotRange -> {{0, 100}, {- .4, .4}}]
```



- Graphics -

Demonstration 2: Gauge 1 works half of the time and then locks up for the rest of the time.

```

cnt1 = 0;
cnt2 = 0;
cnt3 = 0;
holder1 = 0;
holder2 = 0;
holder3 = 0;
deflection = Table[0, {i, 1, 101}];

Do[
ClearAll[x, e1, e2, e3];
strain = p (1 - x) .01;
p =  $\frac{1}{3} \sin[2 \pi i (0.1)] * 3$ ;
x = .25;
If[i ≥ 50, strain1 = .08,
strain1 = strain + Random[] * .0001];
x = .5;
strain2 = strain + Random[] * .0001;
x = .75;
strain3 = strain + Random[] * .0001;

e1 = strain1;
e2 = strain2;
e3 = strain3;
x = 1;

If[i == 1, Goto[threegood]];

If[strain1 == holder1, cnt1++;
flag1 = 1, cnt1 = 0; flag1 = 0; holder1 = strain1];
If[strain2 == holder2, cnt2++;
flag2 = 1, cnt2 = 0; flag2 = 0; holder2 = strain2];
If[strain3 == holder3, cnt3++;
flag3 = 1, cnt3 = 0; flag3 = 0; holder3 = strain3];

If[cnt1 > 2 && cnt2 > 2 && cnt3 > 2, defholder = lastdef; Goto[update]];
If[cnt1 > 2 && cnt2 > 2, Goto[lastone]];
If[cnt1 > 2 && cnt3 > 2, Goto[middleone]];
If[cnt2 > 2 && cnt3 > 2, Goto[firstone]];
If[cnt1 > 2, Goto[lasttwo]];
If[cnt2 > 2, Goto[firstandlast]];

```

```

If[cnt3 > 2, Goto[firsttwo]];

Label[threegood];
defholder = allgood;
Goto[update];

Label[lastone];
defholder = maxdef3;
Goto[update];

Label[middleone];
defholder = maxdef2;
Goto[update];

Label[firstone];
defholder = maxdef1;
Goto[update];

Label[lasttwo];
defholder = onebad;
Goto[update];

Label[firstandlast];
defholder = twobad;
Goto[update];

Label[firsttwo];
defholder = threebad;
Goto[update];

Label[update];

lastdef = defholder;
deflection[[i]] = defholder;
Flatten[deflection], {i, 1, 101}]

ListPlot[deflection, PlotJoined -> True,
  PlotStyle -> {Thickness[.005], RGBColor[0, 0, 1]}]

```

Now without the error detection

```

cnt1 = 0;
cnt2 = 0;
cnt3 = 0;
holder1 = 0;
holder2 = 0;
holder3 = 0;
deflection = Table[0, {i, 1, 101}];

Do[
  ClearAll[x, e1, e2, e3];
  strain = p (1 - x) .01;
  p =  $\frac{1}{3} \sin[2 \pi i (0.1)] * 3$ ;
  x = .25;
  strain1 = If[i ≥ 50, .08,
    strain + Random[] * .0001];
  x = .5;
  strain2 = strain + Random[] * .0001;
  x = .75;
  strain3 = strain + Random[] * .0001;

  e1 = strain1;
  e2 = strain2;
  e3 = strain3;
  x = 1;

  Label[threegood];
  defholder = allgood;
  Goto[update];

Label[update];

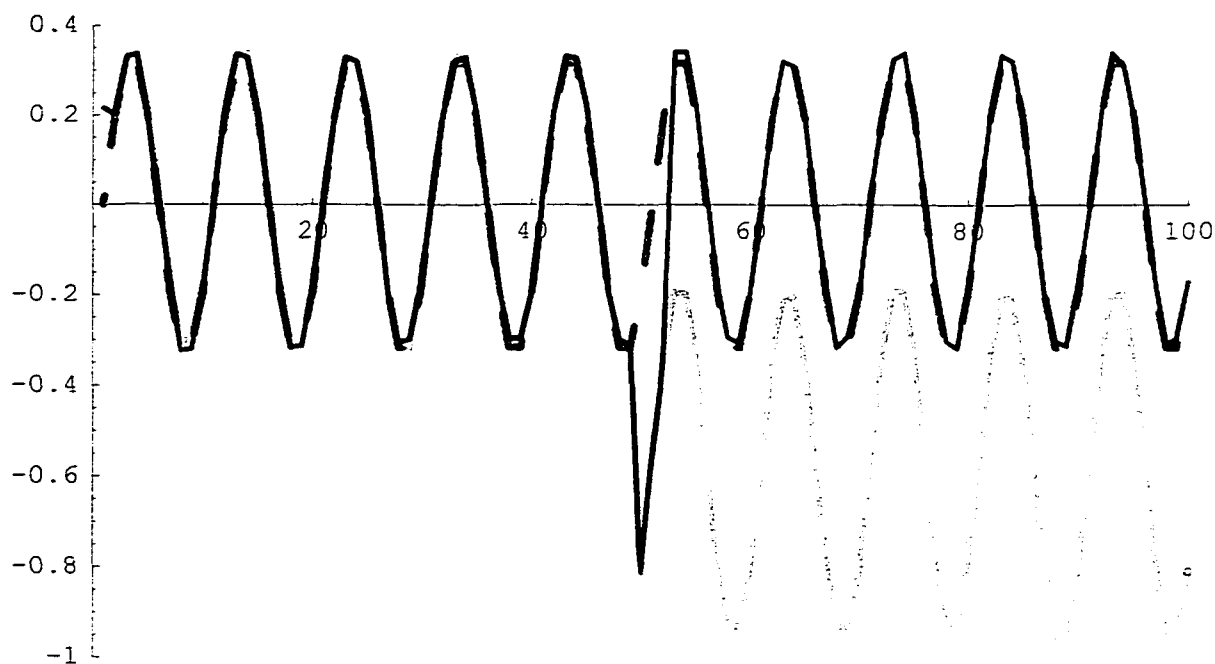
lastdef = defholder;
deflection[[i]] = defholder;
Flatten[deflection, {i, 1, 101}]

ListPlot[deflection, PlotJoined -> True,
  PlotStyle -> {Thickness[.008], RGBColor[0, 1, 0]}]

```

There was never an attempt to claim that the sensor was in error, only that it is not likely to give the same response repeatably, so ignore the reaction associated with it and move to the next reaction. The result is quite good.

```
Show[%36, %23, %54, PlotRange -> {{0, 100}, {-1, .4}}]
```



- Graphics -

Demonstration 3: Gauge 1 and Gauge 2 work half of the time and then lock up for the rest of the time.

```

cnt1 = 0;
cnt2 = 0;
cnt3 = 0;
holder1 = 0;
holder2 = 0;
holder3 = 0;
deflection = Table[0, {i, 1, 101}];

Do[
ClearAll[x, e1, e2, e3];
strain = p (1 - x) .01;
p =  $\frac{1}{3} \sin[2 \pi i (0.1)] * 3$ ;
x = .25;
If[i ≥ 50, strain1 = .08,
strain1 = strain + Random[] * .0001];
x = .5;
If[i ≥ 50, strain2 = .08, strain2 = strain + Random[] * .0001];
x = .75;
strain3 = strain + Random[] * .0001;

e1 = strain1;
e2 = strain2;
e3 = strain3;
x = 1;

If[i == 1, Goto[threegood]];

If[strain1 == holder1, cnt1++;
flag1 = 1, cnt1 = 0; flag1 = 0; holder1 = strain1];
If[strain2 == holder2, cnt2++;
flag2 = 1, cnt2 = 0; flag2 = 0; holder2 = strain2];
If[strain3 == holder3, cnt3++;
flag3 = 1, cnt3 = 0; flag3 = 0; holder3 = strain3];

If[cnt1 > 2 && cnt2 > 2 && cnt3 > 2, defholder = lastdef; Goto[update]];
If[cnt1 > 2 && cnt2 > 2, Goto[lastone]];
If[cnt1 > 2 && cnt3 > 2, Goto[middleone]];
If[cnt2 > 2 && cnt3 > 2, Goto[firstone]];
If[cnt1 > 2, Goto[lasttwo]];
If[cnt2 > 2, Goto[firstandlast]];

```

```

If[cnt3 > 2, Goto[firsttwo]];

Label[threegood];
defholder = allgood;
Goto[update];

Label[lastone];
defholder = maxdef3;
Goto[update];

Label[middleone];
defholder = maxdef2;
Goto[update];

Label[firstone];
defholder = maxdef1;
Goto[update];

Label[lasttwo];
defholder = onebad;
Goto[update];

Label[firstandlast];
defholder = twobad;
Goto[update];

Label[firsttwo];
defholder = threebad;
Goto[update];

Label[update];

lastdef = defholder;
deflection[[i]] = defholder;
Flatten[deflection], {i, 1, 101}]

ListPlot[deflection, PlotJoined -> True,
  PlotStyle -> {Thickness[.005], RGBColor[0, 0, 1]}]

```

Now without the error detection

```

cnt1 = 0;
cnt2 = 0;
cnt3 = 0;
holder1 = 0;
holder2 = 0;
holder3 = 0;
deflection = Table[0, {i, 1, 101}];

Do[
ClearAll[x, e1, e2, e3];
strain = p (1 - x) .01;
p =  $\frac{1}{3}$  Sin[2  $\pi$  i (0.1)] * 3;
x = .25;
If[i ≥ 50, strain1 = .08,
strain1 = strain + Random[] * .0001];
x = .5;
If[i ≥ 50, strain2 = .08, strain2 = strain + Random[] * .0001];
x = .75;
strain3 = strain + Random[] * .0001;

e1 = strain1;
e2 = strain2;
e3 = strain3;
x = 1;

Label[threegood];
defholder = allgood;
Goto[update];

Label[update];

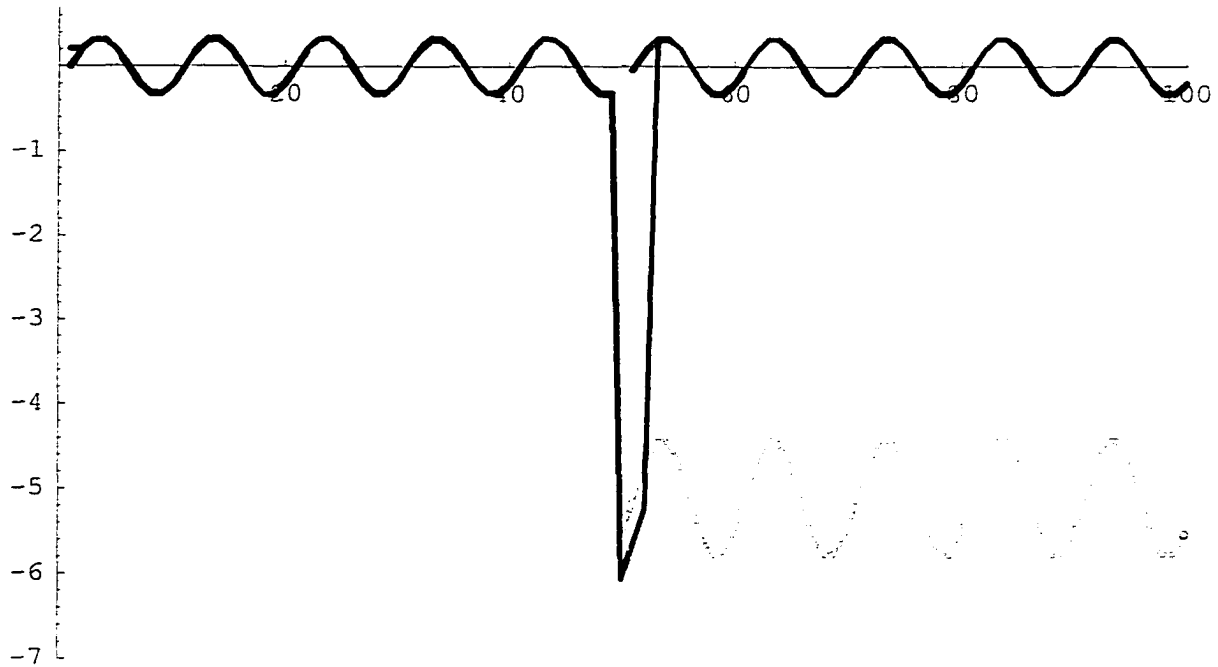
deflection[[i]] = defholder;
Flatten[deflection], {i, 1, 101}]

ListPlot[deflection, PlotJoined -> True,
PlotStyle -> {Thickness[.008], RGBColor[0, 1, 0]}]

```

Without error detection there is an error of >100%, but with this very simple method, there is only the delay for detection and then practically no error at all.

```
Show[%50, %23, %44, PlotRange -> {{0, 100}, {-7, .7}}]
```



- Graphics -

Demonstration 4: Gauge 2 locks up, but then becomes active again.

```

cnt1 = 0;
cnt2 = 0;
cnt3 = 0;
holder1 = 0;
holder2 = 0;
holder3 = 0;
deflection = Table[0, {i, 1, 101}];

Do[
  ClearAll[x, e1, e2, e3];
  strain = p (1 - x) .01;
  p =  $\frac{1}{3} \sin[2 \pi i (0.1)] * 3$ ;
  x = .25;
  strain1 = strain + Random[] * .0001;
  x = .5;
  If[i ≥ 35 && i ≤ 70,
    strain2 = holder2, strain2 = strain + Random[] * .0001];
  x = .75;
  strain3 = strain + Random[] * .0001;

  e1 = strain1;
  e2 = strain2;
  e3 = strain3;
  x = 1;

  If[i == 1, Goto[threegood]];

  If[strain1 == holder1, cnt1++;
    flag1 = 1, cnt1 = 0; flag1 = 0; holder1 = strain1];
  If[strain2 == holder2, cnt2++;
    flag2 = 1, cnt2 = 0; flag2 = 0; holder2 = strain2];
  If[strain3 == holder3, cnt3++;
    flag3 = 1, cnt3 = 0; flag3 = 0; holder3 = strain3];

  If[cnt1 > 2 && cnt2 > 2 && cnt3 > 2, defholder = lastdef; Goto[update]];
  If[cnt1 > 2 && cnt2 > 2, Goto[lastone]];
  If[cnt1 > 2 && cnt3 > 2, Goto[middleone]];
  If[cnt2 > 2 && cnt3 > 2, Goto[firstone]];
  If[cnt1 > 2, Goto[lasttwo]];
  If[cnt2 > 2, Goto[firstandlast]];
  If[cnt3 > 2, Goto[firsttwo]];

```

```

Label[threegood];
defholder = allgood;
Goto[update];

Label[lastone];
defholder = maxdef3;
Goto[update];

Label[middleone];
defholder = maxdef2;
Goto[update];

Label[firstone];
defholder = maxdef1;
Goto[update];

Label[lasttwo];
defholder = onebad;
Goto[update];

Label[firstandlast];
defholder = twobad;
Goto[update];

Label[firsttwo];
defholder = threebad;
Goto[update];

Label[update];

lastdef = defholder;
deflection[[i]] = defholder;
Flatten[deflection], {i, 1, 101}]

ListPlot[deflection, PlotJoined -> True,
  PlotStyle -> {Thickness[.008], RGBColor[0, 0, 1]}]

```

Now with no error detection:

```

cnt1 = 0;
cnt2 = 0;
cnt3 = 0;
holder1 = 0;
holder2 = 0;
holder3 = 0;
deflection = Table[0, {i, 1, 101}];

Do[
ClearAll[x, e1, e2, e3];
strain = p (1 - x) .01;
p =  $\frac{1}{3} \sin[2 \pi i (0.1)] * 3$ ;
x = .25;
strain1 = strain + Random[] * .0001;
x = .5;
If[i ≥ 35 && i ≤ 70,
    strain2 = holder2, strain2 = strain + Random[] * .0001];
x = .75;
strain3 = strain + Random[] * .0001;

e1 = strain1;
e2 = strain2;
e3 = strain3;
x = 1;

Label[threegood];
defholder = allgood;
Goto[update];

Label[update];

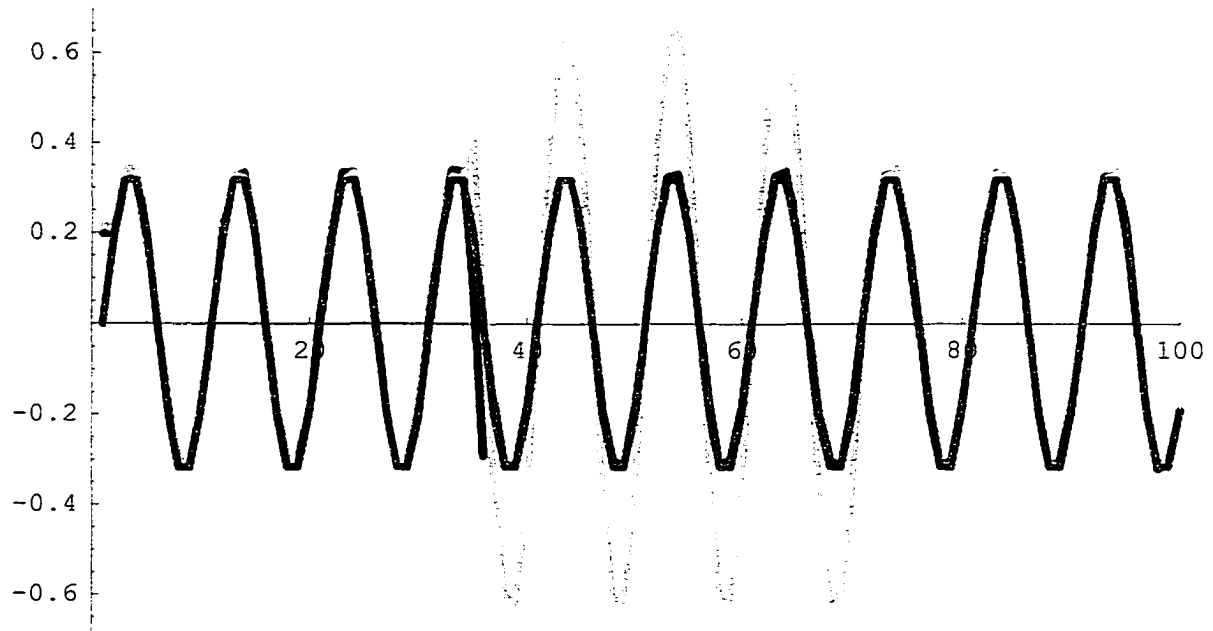
deflection[[i]] = defholder;
Flatten[deflection, {i, 1, 101}]

ListPlot[deflection, PlotJoined -> True,
    PlotStyle -> {Thickness[.008], RGBColor[0, 1, 0]}]

```

Almost no detectable difference between the known deflection and the predicted, even with the random noise and the loss of one of the sensors for part of the time.

```
Show[%56, %58, %31, PlotRange -> {{0, 100}, {- .7, .7}}]
```



- Graphics -

Appendix F, LabView Virtual Instruments

Each virtual instrument used the same front panel. The wiring diagram associated with the front panel is presented below.

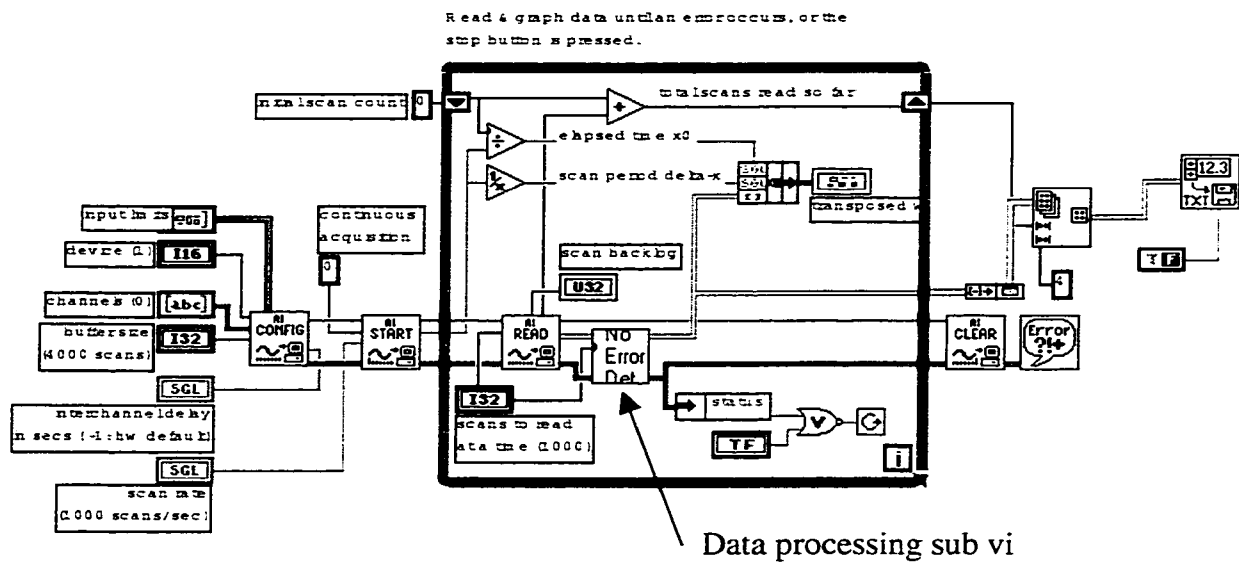


Figure F.1, Front Panel Wiring Diagram

The only thing that changes for each of the virtual instruments is the data processing sub vi labeled in the above wiring diagram. Each of the data processing sub vi's also use the same front panel, so only the wiring diagrams will be presented here.

No Error Correction

This sub vi, takes the strain reading from all three gauges, determines the tip deflection from each gauge, and then outputs the average tip deflection. The wiring diagram is presented below.

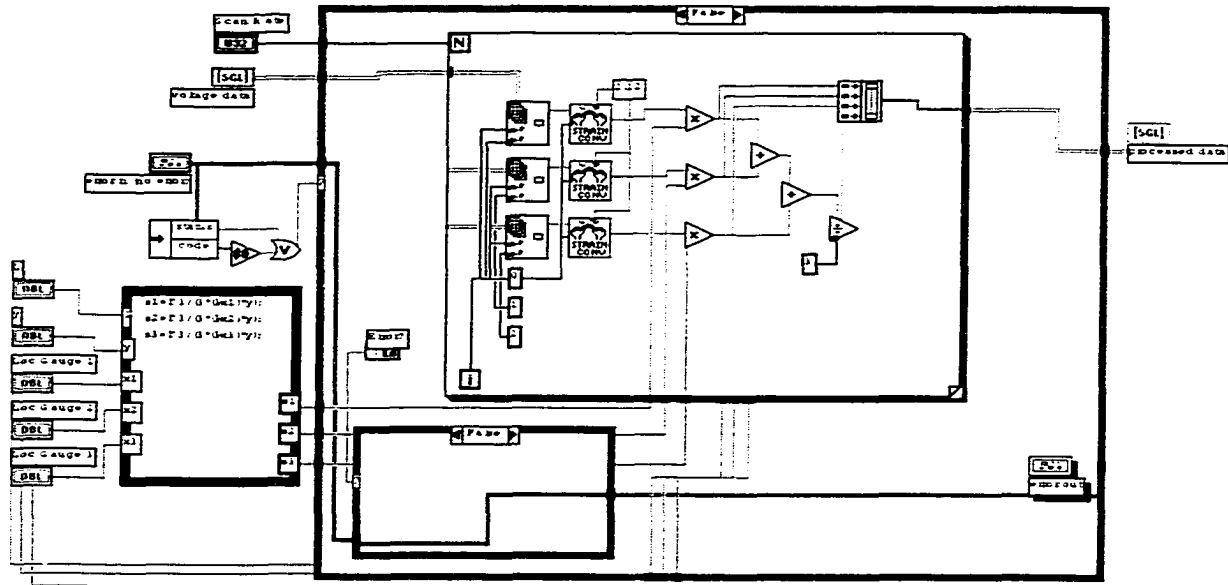


Figure F.2, Average Tip Deflection Sub vi

This vi has no cases, it does not attempt to determine if there is an error. It has one behavior and only one reaction to accomplish that behavior. This is typically what a designer would write. This vi does not address the idea of a possible error. It assumes all of the gauges are good all of the time.

Comparison Error Detection

This sub vi is very similar to the sub vi above. It has only one behavior, to determine the tip deflection of the beam, but it has four reactions to determine the tip deflection. It can use all three gauges or any two gauges by using the majority rule. The critical part of the sub vi is presented below.

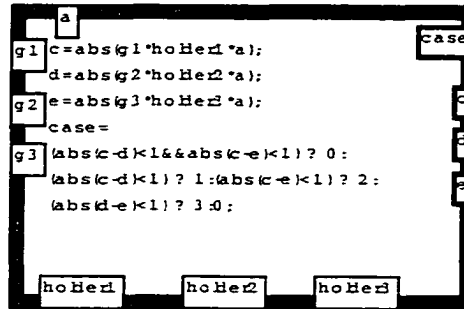


Figure F.3, Formula Node for Comparison

This formula node checks which two gauges agree or if all three gauges agree and then it chooses which case for the vi to calculate the tip deflection. If all three gauges are good, case 0, the vi will determine tip deflection by taking the average of all three. Cases 1, 2 and 3 cover the possibility of two gauges being good and one being bad. This method does not require memory, so it is very fast and requires little programming.

Behavior-Based Error Detection

The behavior-based method requires memory to determine if the behavior of the sensor is not as it should be. The formula node for this error detection is presented below.

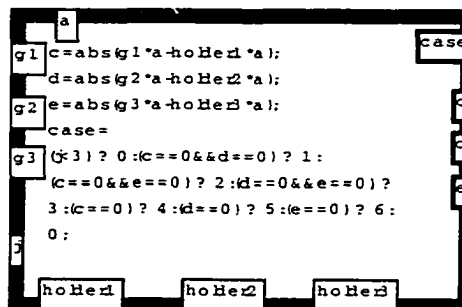


Figure F.4, Formula Node for Behavior-Based Error

This method still has only one behavior, to determine the tip deflection of the beam, but it has seven reactions to make this calculation. Case 0 is for all gauges being good; it is also the default. Case 1 is for only gauge three behaving appropriately, etc. There is no perceptible speed difference between the comparison, behavior-based or no error detection methods.

Appendix G, Minimalist Robots

Currently there is not much research being conducted in the area of micro-robots. Micro-robots will be defined on the millimeter scale. MIT has Ants, Swiss Federal Institute of Technology, LAMI has Jemmy and Inchy and Swiss Federal Institute of Technology, ELF has Alice. Jemmy, Inchy and Alice are all purpose built robots. They were designed and built to compete in the Micro-Mouse competition. These four robots can complete a maze. Ants is a much more capable robot. All of these robots have several design aspects in common. They all use two motors. This allows for steering by either stopping or reversing one of the motors. All of these robots use analog to digital conversion. There is only one other small successful robot, Squirt. Squirt uses only one motor and turns by backing up, only to the right.

By looking at the number of sensors and the volumetric size, all of these robots fit on one chart.

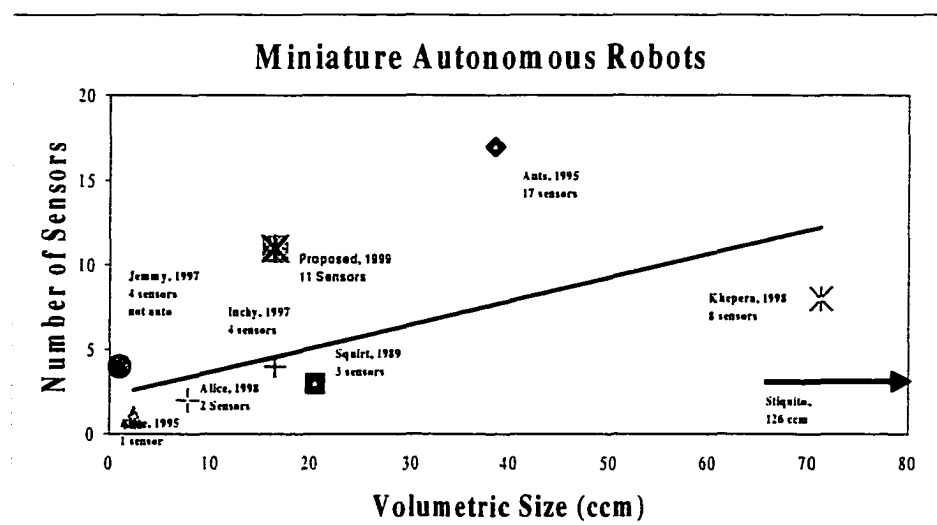


Figure G.1, Miniature Autonomous Robots

There is clearly a pattern here, most robots fall below the best fit line. Ants and Jemmy are the exceptions. This chart becomes much more interesting when behaviors are investigated instead of number of sensors. Jemmy, Inchy and Alice 98 were all designed to negotiate a maze, Alice 95 was designed to follow a line, all only one behavior. Jemmy, Inchy and Alice use the PIC16 microcontroller and two three phase motors. Each motor requires six I/O lines for operation, this means that these robots can not add another sensor to the four they are currently using. These robots can not be enhanced. All of their I/O lines are in use.

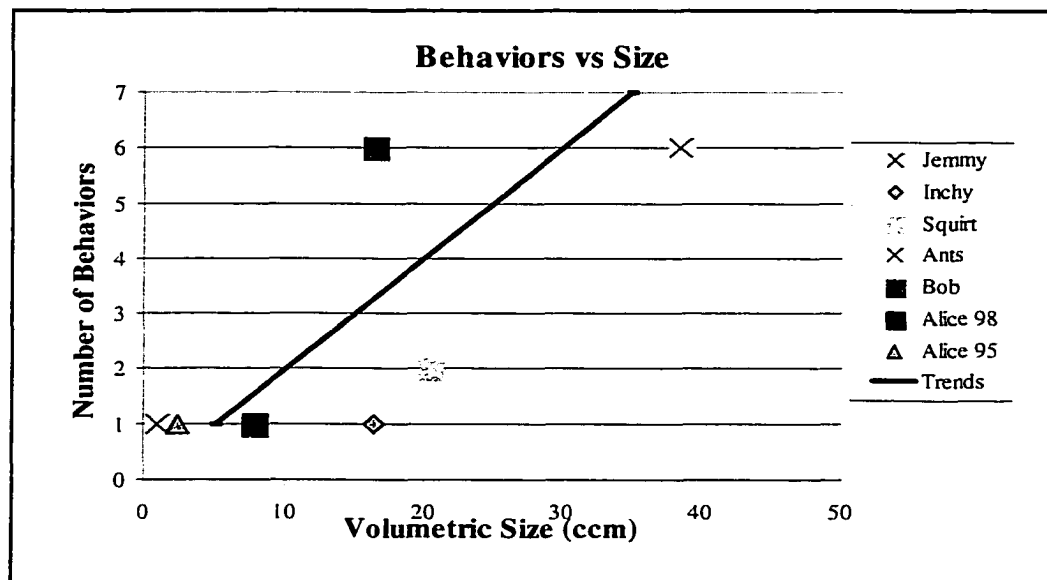


Figure G.2, Behaviors vs Size

Instead of best-fit line, a line with a slope of $0.2 \text{ behaviors/cm}^3$ is included. This shows that all micro-robots developed fall below this line. It can also be argued that a robot with only one behavior does not match our definition of what a "Real Robot" should be in section two. A robot can perform Real Tasks in the Real World. I could be argued that negotiating a maze or following a black line is not a Real World application. The Real World does not come equipped with closed path black lines to follow or single path

mazes to negotiate. Ants, Squirt and Bob are all capable of being placed in most any environment and surviving.

“A superior solution for a purpose may very well involve less data.” (Kurzweil, 1999) What information is actually required for the robots to function?

Squirt requires two bits of information to complete his purpose. When he hears noise he hides in the dark. So he needs one bit of information, sound and one bit of information, dark. Two bits to perform.

Inchy, Jemmy and Alice were all designed to compete in the Micro-Mouse competition. They complete a maze. The best way to complete a maze, assuming the entrance and the goal are on the same island, is the left-hand or right-hand rule. This is not the quickest, but it is the easiest to program. (Troxell, 1999b) Therefore, the only information these robots require is wall or no wall, essentially one bit of information. They achieve this one bit of information with four sensors for Inchy and Jemmy. Alice uses two sensors for that one bit of information.

Ants is equipped with seventeen sensors, however, all seventeen are not required for all “purposes”. One example of Ant programming is the Tag game. Each Ant has essentially two moods. A mood is a hierarchy of behaviors to handle more complex behaviors. (McLurkin, 1995) Each mood incorporates three reactions to play the game. For the game of Tag, each Ant requires one bit of information first, “It” or “Not It”. This will activate the appropriate mood. Then each mood requires two or three bits of information to operate. (McLurkin, 1995)

Minimalist robots use the principle of Avoiding Irreversibility’s, that can be broken down into two main propositions: “1. Use the cheapest that will do and 2.

Throw away nothing of value.” (French, 1994) TechOptimizer 3.0 was employed to come up with these design ideas. Using the principle of Separation, micro-robots should use only one of the motors, and completely remove the A/D board. According to the Cambridge Engineering Database, Guideline 3615: “To make a system as small as possible, every element of the system should be miniaturized to its technical or physical limit and all elements should be based on the same or compatible technology.”

Appendix H: Active Damping Virtual Instruments

There were two virtual instruments written for the active damping experimentation. The first, piezobeam3.vi, did not incorporate any form of error detection and correction. The second, errorpiezo.vi, incorporated the behavior-based error detection and correction outlined in this work. Both of the virtual instruments used the same front panel as shown the Active Damping section. The virtual instruments were both setup to read channels zero and one. Both control loops were run at 40 samples/second. The only difference in the two virtual instruments were in the sub-vi's associated with them. Each sub-vi will be discussed here.

Noerror.vi was the sub-vi that did not incorporate any form of error detection or correction. The front panel associated with this sub-vi is shown below.

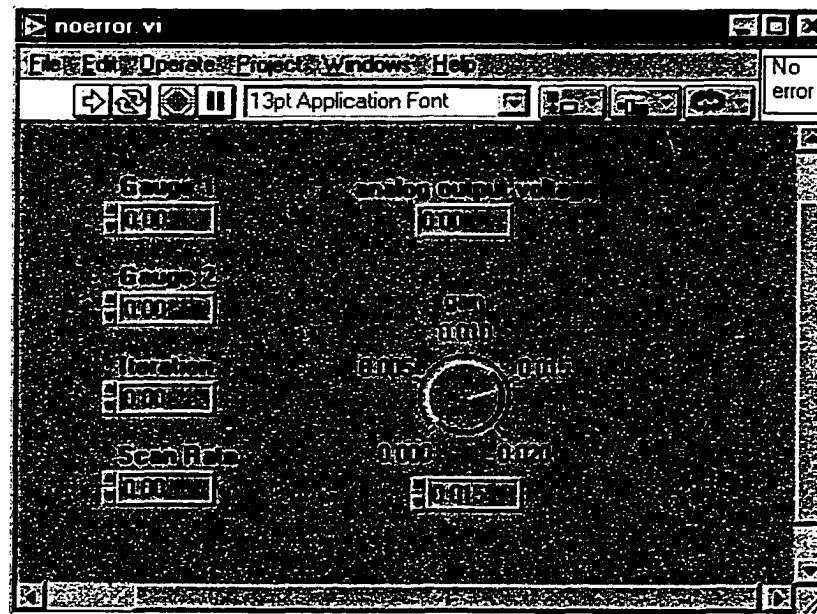


Figure H.1, No Error Sub-Vi Front Panel

The gain for the damping force is adjustable. Experimentation showed that the best damping was achieved with a gain of 0.015. The thick control wires on the piezoelectric force actuators as opposed to very small strain gauge wires caused the beam to have differing stiffnesses on each side. For this reason a gain of 0.017 was used on one side and 0.015 was used on the other side. The wiring diagram for this sub-vi is presented below.

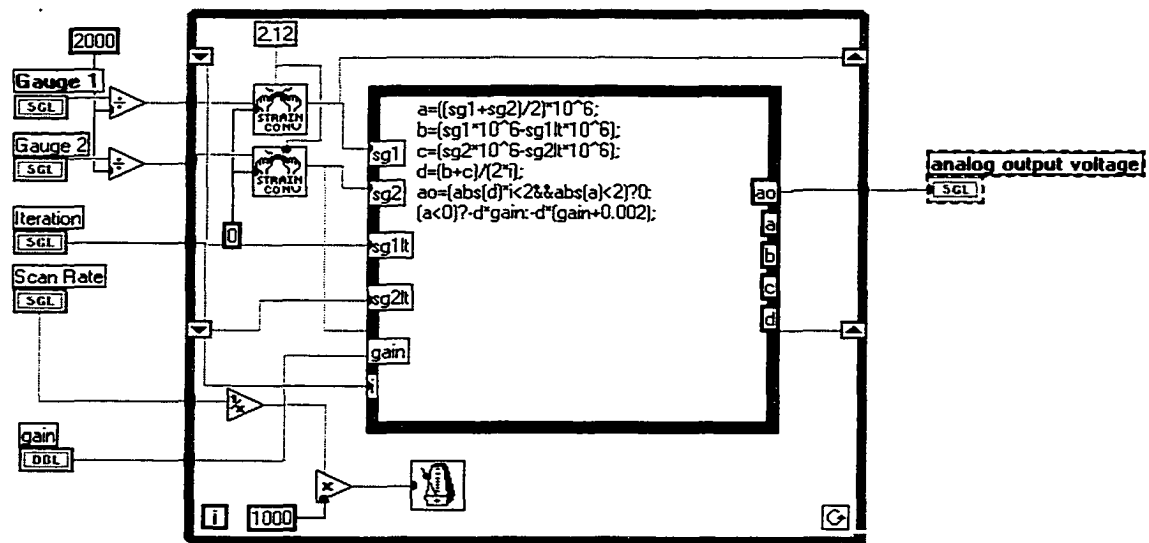


Figure H.2, No Error Sub-Vi Wiring Diagram

The output voltage for this sub-vi is zero whenever the velocity is less than two and the absolute value of the displacement is less than two. For all other situations, the output voltage is average velocity of the two strain gauges multiplied by the gain. The gain, again, is variable depending on whether the displacement is positive or negative. This gave the best results. I am sure that the damping could be improved further by continuing to tweak these parameters, but the goal of this research only dealt with error detection and recovery, this was just a system to test the technique.

The transition diagram for this sub-vi will display how the sub-vi performs its task. There are two submachines associated with the single behavior, damp the beam. One submachine actively applies a force and the other submachine applies zero force.

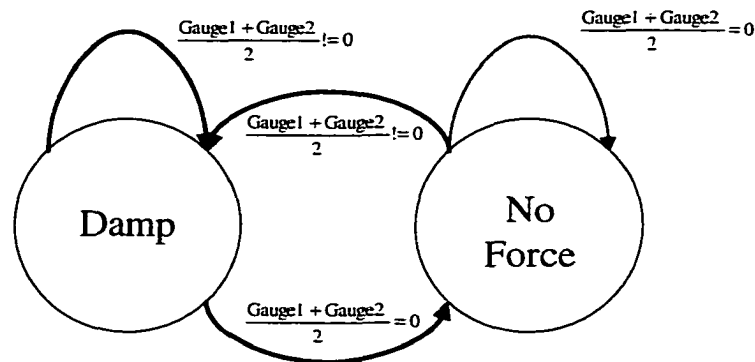


Figure H.3, Transition Diagram for No Error Detection

This submachine works only on the average of the two strain gauges. This can cause the situation where the submachine is applying no force when it should simply because the average of the two gauges is zero. The strain gauges only have a resolution of $1.38 \mu\epsilon$, so looking at a change of $2 \mu\epsilon$ strain will cause the average to be zero frequently, especially near zero micro-strain and at the displacement peaks. This behavior was seen during experimentation with the virtual instrument.

The other sub-vi was called piezocntrl.vi. The front panel for this sub-vi was identical to the noerror sub-vi. The wiring diagram, of course, was different. The wiring diagram is presented below.

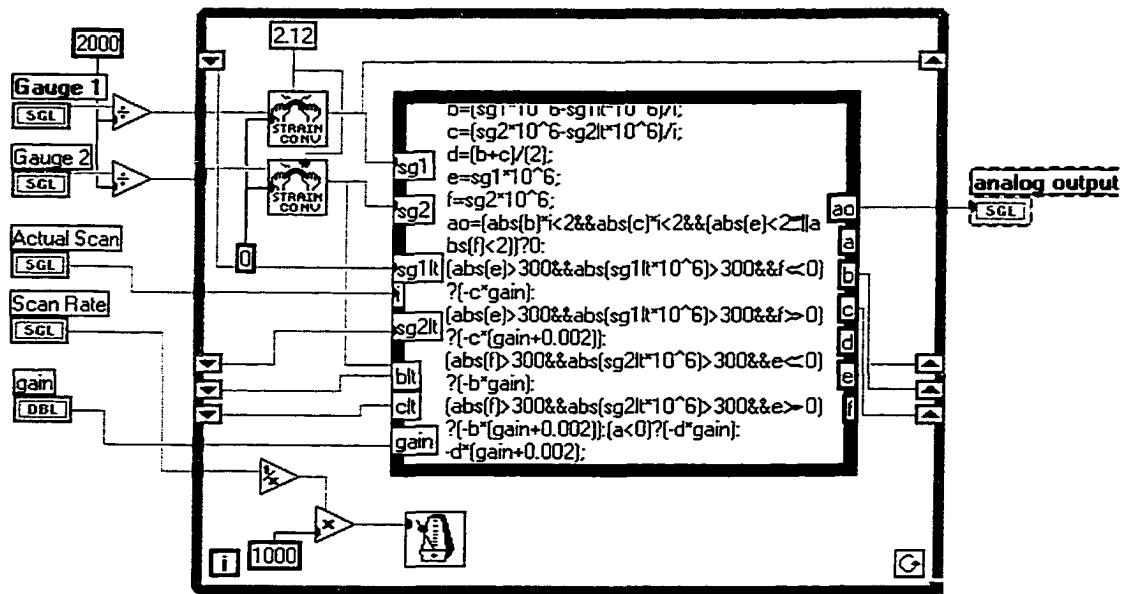


Figure H.4, Piezocntrl Sub-Vi Wiring Diagram

The main difference between this sub-vi and the previous sub-vi, is that this sub-vi has three ways to determine what the correct output voltage should be. It can use both strain gauges, the default, or it can use either of the two strain gauges. It does not compare strain gauges together to determine one is good and the other is bad, it simply looks for whether the strain readings are changing. If the readings are changing, it assumes that the gauge is good and it uses it.

Now to draw up the transition diagram for this sub-vi.

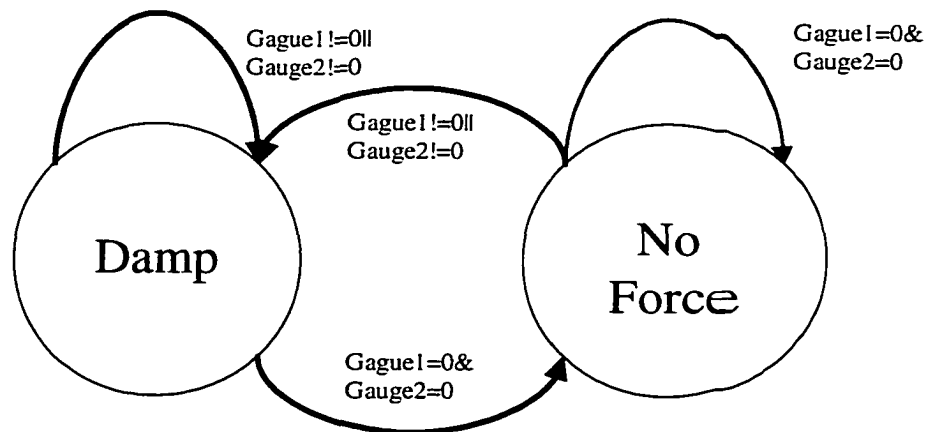


Figure H.5, Transition Diagram with Error Detection

This is the highest level transition diagram for this finite state machine. Within Damp there are three submachines used to determine the required damping force. These three submachines are associated with the reactions of using both strain gauges or only one of the two strain gauges. Error detection is how the submachine will determine which reaction is active. Damp uses the following the transition diagram.

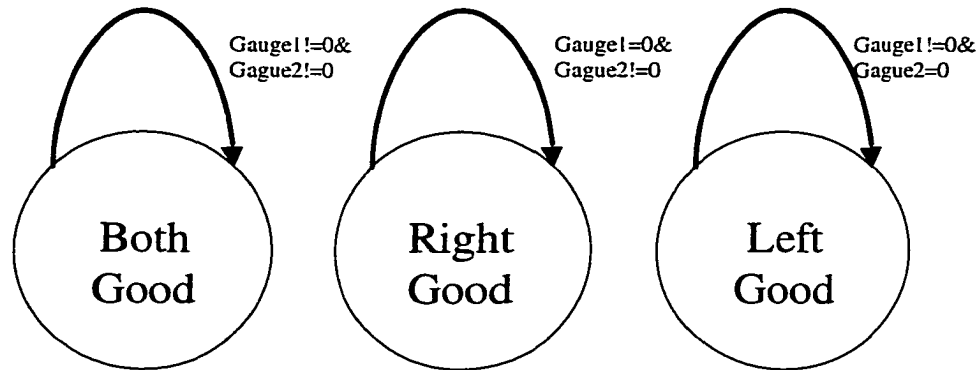


Figure H.6, Damp Transition Diagram

A velocity less than $2 \mu\epsilon$ is possible and is what is desired by the control algorithm. Therefore, there is no error detection scheme that could handle this situation. There are not enough sensors to have a majority rule. The values of the sensors are possible, therefore, no way to detect an error, but the behavior of the sensors is detectable. If there is a velocity, then the control should apply a countering force. Circularity is built in to the Damp submachine. First it checks for both gauges showing a velocity, then it checks for only the right gauge and then only the left gauge. If there is a velocity reading, then a force will be applied. Once it is certain there is no velocity, then the control will transition to the No Force submachine. This resulted in far superior performance to the case of only looking at the average from both gauges.

Appendix I: Determination of Damping Ratio

Determining Damping Ratio

Determine the approximate natural frequency of the beam.

```
mod = 10.6 * 106;
l = 15;
thick = .032;
width = 2.270;
i =  $\frac{1}{12}$  width thick3;
g = 32.2 * 12;
weight = 0.1 * width * thick;

natural =  $\frac{3.52}{2 \pi} \sqrt{\frac{\text{mod } i}{\text{weight} * l^4}}$ 

4.65491
```

First, without active damping

Now with Active Damping

Active Damping with Error Correction

```

adec = {-8.3, -34.582, -77.459, -114.801, -124.482, -91.29, -73.31,
-23.516, 6.917, 12.45, -8.3, -29.049, -45.647, -38.731, -15.216, 0,
24.9, 24.9, 9.683, -13.833, -29.049, -33.198, -22.132, -1.383,
9.683, 13.833, 6.917, -8.3, -23.516, -30.432, -24.899, -9.683,
4.15, 12.45, 6.917, -8.3, -19.366, -23.516, -20.749, -9.683, 5.533,
11.067, 9.683, 2.767, -8.3, -16.599, -16.599, -9.683, 4.15, 9.683,
8.3, 1.383, -6.917, -15.216, -17.983, -13.833, -5.533, 5.533, 4.15,
-2.767, -8.3, -13.833, -15.216, -12.45, -4.15, 4.15, 6.917, 5.533,
-4.15, -11.066, -11.066, -4.15, 2.767, 5.533, 4.15, 0, -6.917,
-9.683, -9.683, -5.533, 0, 5.533, 2.767, -2.767, -6.917, -8.3,
-5.533, -1.383, 4.15, 5.533, 2.767, -2.767, -6.917, -8.3, -6.917,
-4.15, 1.383, 2.767, 2.767, 0, -5.533, -8.3, -6.917, -2.767, 0,
1.383, 0, -4.15, -5.533, -5.533, -4.15, 1.383, 4.15, 2.767, -1.383,
-4.15, -4.15, -2.767, 0, 2.767, 2.767, 1.383, 0, -4.15, -5.533,
-4.15, -2.767, 0, 1.383, 1.383, -1.383, -4.15, -5.533, -5.533,
-2.767, -1.383, -1.383, -2.767, -4.15, -6.917, -6.917, -5.533,
-2.767, 0, 0, 0, -1.383, -2.767, -4.15, -2.767, 0, 1.383, 2.767,
1.383, -1.383, -2.767, -2.767, -2.767, -1.383, 0, 0, -1.383,
-2.767, -5.533, -5.533, -5.533, -4.15, -2.767, -1.383, -1.383,
-1.383, -2.767, -2.767, -2.767, -1.383, 0, 2.767, 2.767, 1.383,
1.383, 0, -1.383, -1.383, 0, 0, 0, 0, -1.383, -2.767, -4.15, -2.767,
-1.383, -1.383, 0, 0, -2.767, -4.15, -4.15, -4.15, -2.767, -1.383,
-1.383, -1.383, -1.383, -2.767, -4.15, -4.15, -2.767, -1.383, 0, 0,
0, -1.383, -1.383, -2.767, -1.383, -1.383, 0, 0, 0, -1.383,
-2.767, -2.767, -2.767, -1.383, 0, 0, 0, -1.383, -2.767,
-2.767, -2.767, -2.767, -1.383, 0, 0, -1.383, -1.383,
-2.767, -2.767, -2.767, -1.383, -1.383, 0, -1.383, -1.383,
-2.767, -2.767, -2.767, -1.383, -1.383, 0, -1.383, -2.767};

```

```

ListPlot[adec, PlotJoined -> True,
PlotStyle -> {RGBColor[0, 1, 1], Thickness[0.008]},
PlotRange -> All]

```

```

n = Length[adec];
fadec = Abs[Fourier[adec]];
fadec = ReplacePart[fadec, 30, 1];
peak = Flatten[Position[fadec, Max[fadec]]];

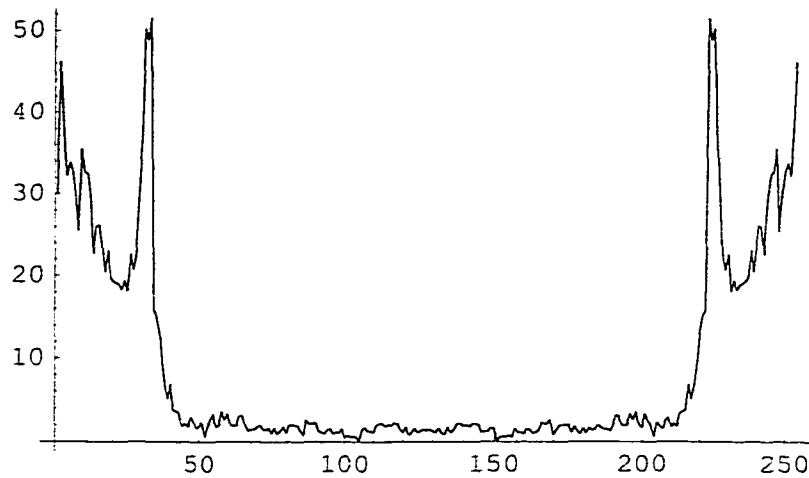
```

$$\Delta t = \frac{1}{40};$$

$$\omega_n = \frac{\text{peak}[[1]]}{n \Delta t} // N$$

```
5.19685
```

```
ListPlot[fadec, PlotJoined -> True, PlotRange -> All]
```



- Graphics -

$$\text{halfpower} = \frac{1}{\sqrt{2}} \text{Max}[\text{fadec}]$$

```
36.4583
```

```
ListPlot[fadec, PlotJoined -> True, PlotRange -> {{29, 34}, {0, 60}},  
PlotStyle -> RGBColor[1, 0, 0]]
```

```
ClearAll[x];
```

```
data = {{29, fadec[[29]]}, {30, fadec[[30]]},  
{31, fadec[[31]]}, {32, fadec[[32]]}, {33, fadec[[33]]}};
```

```
left = Fit[data, {1, x, x^2, x^3}, x];
```

```
Solve[halfpower == left, {x}]
```

```
{{x -> 6.27061}, {x -> 29.6522}, {x -> 35.0978}}
```

```
ClearAll[x];
```

```
data = {{33, fadec[[33]]}, {34, fadec[[34]]}};
```

```
right = Fit[data, {1, x}, x];
```

```
Solve[halfpower == right, {x}]
```

- General::spell1 : Possible spelling error: new symbol name "right" is similar to existing symbol "Right".

```
{{x -> 33.4228}}
```

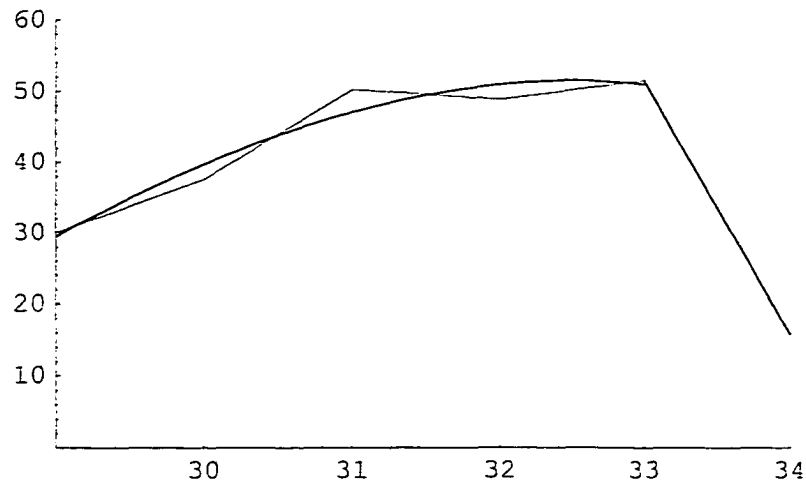
$$\text{dampingratio} = \frac{\frac{33.4228 - 29.6522}{n \Delta t}}{2 (\omega_n)}$$

0.0571303

Plot[left, {x, 29, 33}]

Plot[right, {x, 33, 34}]

Show[%12, %21, %24, PlotRange -> {{29, 34}, {0, 60}}]



- Graphics -

Now with the error and no error correction

Now with error correction

Appendix J: Hypothesis Tests and Power Determination

Statistical Analysis of Damping Ratios

Use Mathematica to perform the statistical analysis to verify whether the sample populations are the same or different.

The data:

```
ercrtnoerr = {0.057, 0.043,  
  0.032, 0.046, 0.044, 0.048, 0.033, 0.046, 0.048, 0.028};  
errcrterr = {0.055, 0.047,  
  0.047, 0.032, 0.040, 0.045, 0.039, 0.048, 0.032, 0.037};  
noerror = {0.033, 0.038, 0.028,  
  0.035, 0.037, 0.028, 0.036, 0.030, 0.036, 0.030};  
error = {0.023, 0.025, 0.030,  
  0.028, 0.028, 0.033, 0.030, 0.043, 0.023, 0.031};  
grandmean = {0.057, 0.043, 0.032, 0.046, 0.044, 0.048, 0.033,  
  0.046, 0.048, 0.0280, .055, 0.047, 0.047, 0.032, 0.040,  
  0.045, 0.039, 0.048, 0.032, 0.037};
```

The required analysis package:

```
<< Statistics`HypothesisTests`
```

The Means and Variances of the populations:

```
mu1 = Mean[ercrtnoerr]  
var1 = Variance[ercrtnoerr]  
  
0.0425  
  
0.0000787222
```

```
mu2 = Mean[errcrterr]
var2 = Variance[errcrterr]
```

```
0.0422
```

```
0.0000557333
```

```
mu3 = Mean[noerror]
var3 = Variance[noerror]
```

```
0.0331
```

```
0.0000145444
```

```
mu4 = Mean[error]
var4 = Variance[error]
```

```
0.0294
```

```
0.0000340444
```

```
mu5 = Mean[grandmean]
var5 = Variance[grandmean]
```

```
0.04235
```

```
0.0000637132
```

```
VarianceRatioTest[error, noerror, 1, SignificanceLevel -> 0.1,
  FullReport -> True]
```

```
{FullReport -> Ratio      TestStat      Distribution
  2.34072      2.34072      FRatioDistribution[9, 9]}
OneSidedPValue -> 0.110597,
Fail to reject null hypothesis at significance level -> 0.1}
```

```
VarianceRatioTest[erctrnoerr, errcrterr, 1,
  SignificanceLevel -> 0.1, FullReport -> True]
```

```
{FullReport -> Ratio      TestStat      Distribution
  1.41248      1.41248      FRatioDistribution[9, 9]}
OneSidedPValue -> 0.307605,
Fail to reject null hypothesis at significance level -> 0.1}
```

```
VarianceRatioTest[error, errcrterr, 1, SignificanceLevel -> 0.1,  
FullReport -> True]
```

```
{FullReport ->  
  Ratio      TestStat      Distribution  
  0.610845    0.610845    FRatioDistribution[9, 9]'  
OneSidedPValue -> 0.237099,  
Fail to reject null hypothesis at significance level -> 0.1}
```

```
VarianceRatioTest[ercrtnoerr, noerror, 1, SignificanceLevel -> 0.15,  
FullReport -> True]
```

```
{FullReport ->  
  Ratio      TestStat      Distribution  
  5.41253    5.41253    FRatioDistribution[9, 9]'  
OneSidedPValue -> 0.00962313,  
Reject null hypothesis at significance level -> 0.15}
```

The goal is to be able to say that there is no difference when the strain gauge is removed for behavior-based error detection. To make this statement, the power of the test is required.

Assume that damping ratios have a normal distribution. Then calculate the power of the test from the following relationship (Rosner, p276):

$$\text{Power} = \Phi\left(-z_{1-\alpha} \sqrt{n_1} \frac{\Delta}{\sqrt{\sigma_1^2 + \sigma_2^2/k}}\right)$$

$$\Delta = \mu_1 - \mu_2$$

σ_i^2 = variance of the respective population

$$k = n_2/n_1$$

$$\alpha = 0.1$$

Using this relationship the following powers are calculated:

For Comparison of Error Detection, with and without an induced error. The power was 12%. That is there is only a 12% chance of rejecting the hypothesis.

For Comparison of No Error Detection, with and without an induced error. The power was 62%. There is a 62% chance of detecting a significant difference at an α of 0.1.

For Comparison of Error Detection and No Error Detection, with an error. The power was 99.7%. There is a 99.7% chance of detecting a significant difference at an α of 0.1.

For Comparison of Error Detection and No Error Detection, without an error. The power was 99.95%. There is a 99.95% chance of detecting a significant difference at an α of 0.1.

Below is the Difference of Means tests. It uses the Students t distribution since the population size is small. The test is whether the difference in means is zero.

Compare both cases, with and without error detection, with no error.

```
MeanDifferenceTest[ercrtnoerr, noerror, 0,
EqualVariances -> True,
SignificanceLevel -> .10, FullReport -> True]

{FullReport ->
  MeanDiff      TestStat      Distribution
  0.0094        3.07797       StudentTDistribution[18]'
  OneSidedPValue -> 0.00324208,
  Reject null hypothesis at significance level -> 0.1}
```

90% confident that the means are different. The P value tells that the point where one can not make a decision is about 99.7%.

Compare both error cases, one with error detection and one without.

```
MeanDifferenceTest[errcrterr, error, 0,
EqualVariances -> True,
SignificanceLevel -> .10, FullReport -> True]

{FullReport ->
  MeanDiff      TestStat      Distribution
  0.0128        4.27194       StudentTDistribution[18]'
  OneSidedPValue -> 0.000229386,
  Reject null hypothesis at significance level -> 0.1}
```

90% confident that means are different. The P value tells that the point where one can not make a decision is about 99.98%.

Compare both cases with error correction to see if there is a difference in their means.

```
MeanDifferenceTest[ercrtnoerr, errcrterr, 0,
EqualVariances -> True,
SignificanceLevel -> .10, FullReport -> True]

{FullReport ->
  MeanDiff      TestStat      Distribution
  0.0003         0.0818148     StudentTDistribution[18]'
  OneSidedPValue -> 0.467848,
  Fail to reject null hypothesis at significance level -> 0.1}
```

There is no difference when a strain gauge is removed. The P value to make the statement would be 53%.

Compare both cases that did not use error detection to see if the difference in the means is significant.

```
MeanDifferenceTest[error, noerror, 0,
EqualVariances -> True,
SignificanceLevel -> .10, FullReport -> True]

{FullReport ->
  MeanDiff      TestStat      Distribution
  -0.0037        -1.67855     StudentTDistribution[18]'
  OneSidedPValue -> 0.0552583,
  Reject null hypothesis at significance level -> 0.1}
```

There was a difference when the strain gauge was removed from the non error detection scheme. The P value tells that the point where one can not make a decision is about 95%.