NASA-TM-86288 #

NASA Technical Memorandum 86288

NASA-TM-86288 19850002350

A RELATIONAL APPROACH TO THE DEVELOPMENT OF EXPERT DIAGNOSTIC SYSTEMS

KATHY R. AMES

OCTOBER 1984

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

Table of Contents

1 Introduction 1
2 The RKBM Model 3
2.1 Implementation 3
2.2 The End Effector System 3
2.3 The Format of the End Effector Model 6
2.4 Some Details About the Model 8
2.4.1 Representing the Jaw Position 8
2.4.2 The Step Size of the Simulation 9
2.4.3 Representing Error Conditions in the Model 11
2.4.4 An Order-Dependent versus an Order-Independent Model .. 11
2.4.5 Some Example Simulations 14
3 Error Diagnosis Using the RKBM Model 19
3.1 The RKBM Model and the Operational System 19
3.2 An Algorithm for Error Diagnosis 20
3.2.1 Method I 21
3.2.2 Method II 24
4 Conclusions 26
5 References 28
Appendix A: Prime LISP versus UTLISP 29
Appendix B: RKBM Lisp Routines 32
Appendix C: The RKBM End Effector Model 36
Appendix D: "Trouble-shooting" Flowchart 49
Appendix E: Normal Simulation 52
Appendix F: Abnormal Simulation 54
Appendix G: Method I Trace for Bad Tachometer Example 56
Appendix H: Method II Trace for Bad Tachometer Example 60

N85-10658#

1. Introduction

Recently, there has been considerable interest in the development of expert systems that use causal reasoning - reasoning based on an understanding of the structure or function of the device or system they are examining [1]. A proposal under current investigation is that, given a representation of the functional and/or structural interrelationships among the components of a system, an expert system could be developed to analyze it. For example, such an expert system could be used for diagnostic problem solving, in which the normal states of the components of a system under analysis are known and an abnormal state and its cause can be identified.

A previous research project has examined this approach to developing expert systems [2]. A scheme for representing any real or abstract system has been developed along with a set of routines capable of executing a simulation of that system . The knowledge representation format chosen is similar to that of a relational data base - a system is modelled as a set of relations describing its structure and function. This knowledge representation along with the machinery to drive it is termed a Relational Knowledge-Base Machine (RKBM).

Given this RKBM modelling scheme, there are two goals of the research described in this paper. The first goal is to use the RKBM approach to model a microprocessor-controlled end effector/sensor system currently under development in the Intelligent Systems Research Laboratory of the Automation Technology Branch (ATB) at Langley Research Center. The second goal is, by studying the end effector model, to examine the possibility of extending the RKBM mechanisms to include the functions of an expert diagnostic system. This second goal can be stated in the form of two questions. First, can the RKBM representation be used as the basis of an expert system that can answer such questions as, "What is the state of a component of the system?" and "Why is the component in that state?" Second, if the RKBM approach is found to be suitable, then what is a reasonable algorithm for performing such analyses?

2. The RKBM Model

2.1. Implementation

The RKBM used for this research is programmed in UTLISP on the CDC Network Operating System. Appendix A is a discussion of the conversion from the original Prime LISP implementation to this implementation. The UTLISP code for the RKBM driver routines and end effector model is found in Appendices B and C, respectively.

2.2. The End Effector System

The end effector system used as a basis for this project is diagrammed in figure 1. The end effector components expanded in some detail in the RKBM model are the microprocessor controller and the servo power loop. The only mechanical action represented is the movement of the jaws - details about the motion of specific gears are not included. Also, no sensory information is included since little sensory information was included in the laboratory system at the time development began on the RKBM model. Following is a brief description of the operation of the end effector system.

The system is controlled by an 8031 microprocessor. This microprocessor has a position register which keeps track of the actual position of the jaws. There is also a memory location to hold a commanded jaw position, which may be changed at any time by a user. Both of these positions are represented in terms of octal encoder counts, where 0 represents the fully open jaw position and -26135 represents the completely closed position. When the system is on, the microprocessor pro-

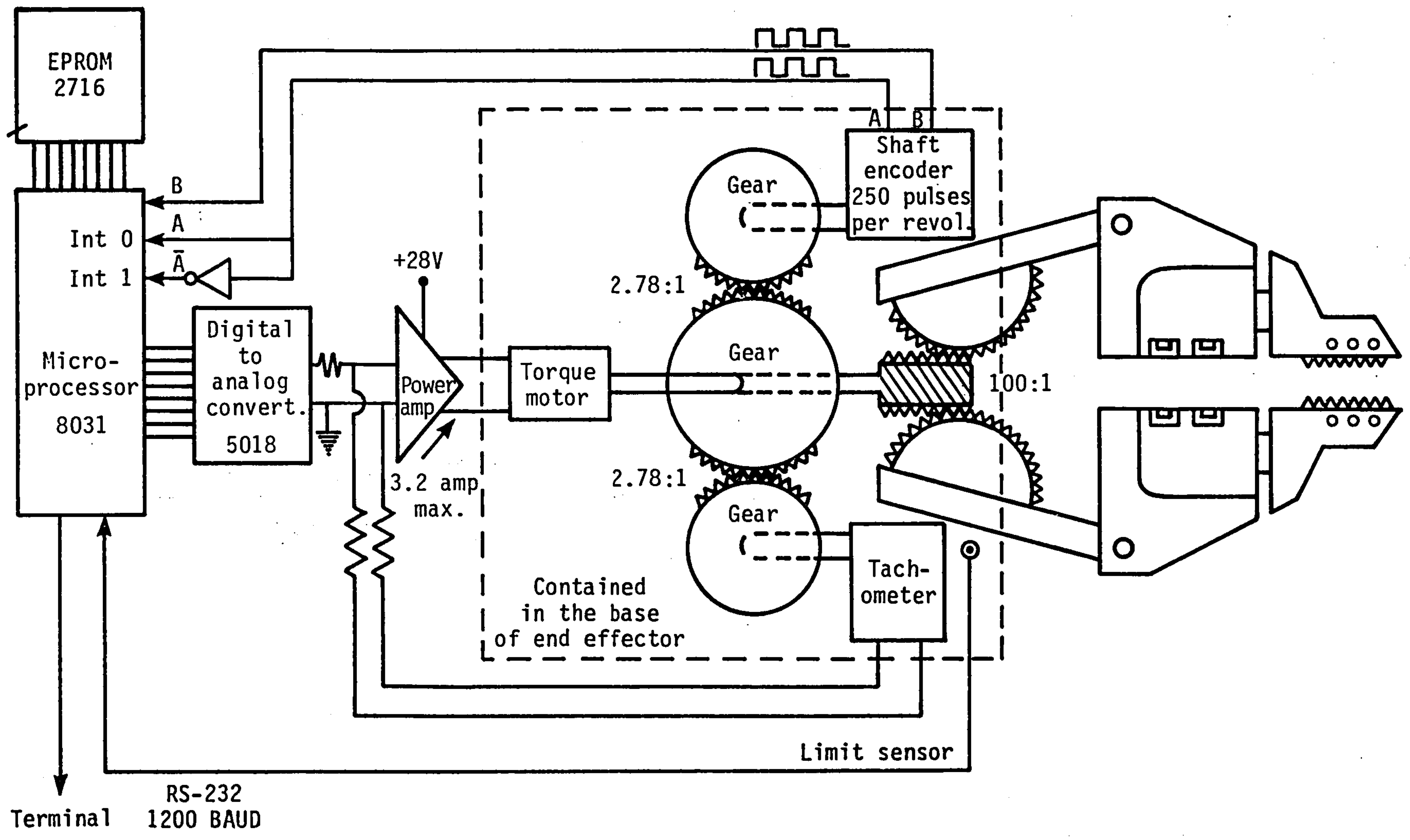


Figure 1. Diagram of the End Effector System

gram continually loops, computing the difference, or error, between this commanded position and the actual jaw position in the position register. The resulting error signal is transmitted to a digital/analog converter (DAC) which, in turn, transmits the signal to a servo power amplifier. From here, the amplified signal is passed to a DC torque motor. When the error signal is sufficiently small, there is not enough power to overcome friction in the motor. (In the model, however, a frictionless environment is assumed.) Consequently, there is no jaw movement and the system is in an equilibrium state. When the signal is strong enough, the motor shaft drives a worm gear which, in turn, drives two sector gears, each sector gear symmetrically controlling one of the jaw arms.

As the motor turns, an incremental shaft encoder geared to the motor provides feedback to the microprocessor program. A tachometer is also geared to the motor in the same manner as the shaft encoder and provides feedback from the motor to the servo amplifier. The tachometer outputs rate information about the motor shaft movement which is combined with the microprocessor error signal output to determine the input to the servo amplifier. The purpose of this is to prevent the motor shaft from rotating too quickly for the shaft encoder to encode the movement.

Each shaft encoder count interrupts the microprocessor program and this count is then used to increment or decrement the position register, depending on the direction of jaw movement. The change in the position register is then reflected in the error computed by the program which, in turn, is reflected in the error signal transmitted to the DAC. This cycle of jaw movement and corresponding position register updates continues until the system once again reaches an equilibrium state.

2.3. The Format of the End Effector Model

The RKBM format used to model the end effector is similar to that of Blanks' gas furnace system [2]. Table 1 illustrates the main relation in the RKBM end effector model. Using relational data base termi-

END EFFECTOR RELATION

NAME	VARIABLE	PARENT	VALUE_BY	VALUE*	VALUE
microprocessor	condition	system	default	ok	ok
commanded_ position	location	micro- processor	default	0	0
actual_ position	location	micro- processor	ACTUAL	0	0
distance	amount	system	DISTANCE_AMT	0	0
error_signal	direction	micro- processor	ERR_SIG_DIR	off	off
error_signal	magnitude	micro- processor	ERR_SIG_MAG	0	0
dac	condition	system	default	ok	ok
dac_voltage	direction	dac	DAC_VOLT	off	off
servo_amp	condition	system	default	ok	ok
servo_amp_ voltage	direction	servo_amp	SERVO_VOLT	off	off
motor_leads	condition	system	default	ok	ok
motor_circuit	condition	motor	default	closed	closed
motor_voltage	direction	motor	MOTOR_VOLT	off	off
motor_current	condition	motor	CURRENT	off	off
motor_switch	condition	motor	default	on	on
jaws_to_move	condition	system	MOVE_JAWS	no	no
gears	condition	system	default	ok	ok
power_supply	condition	system	default	on	on
short_circuit	condition	system	default	no	no
tachometer	condition	system	default	ok	ok
shaft_encoder	condition	system	default	ok	ok

Table 1. The End Effector Relation

nology, each column heading is an attribute and each row is six-tuple containing an entry for each of the six attributes. Each tuple represents a component of the system.

The attribute NAME is used to identify a particular component. Notice that each component does not necessarily correspond to a component in the actual end effector system. "Distance" and "jaws_to_move" are two such model components. These two components are discussed in more detail in later sections of this paper. The attribute VARIABLE indicates the particular quality of the component that is being described, e. g. the direction of the motor_voltage. The attribute PARENT is intended to convey the structural organization of the system. (The structural relationships between the components are neither incorporated in the proposed error detection algorithm, nor are they incorporated in running a simulation of this particular model.) The VALUE and VALUE* attributes indicate the state of each component, e.g. the direction of the motor_voltage may have a VALUE of positive, negative, or off. The necessity of these two apparently equivalent attributes is explained later. Throughout the remainder of this paper, references to the value of a component in the model refer to the entry for the VALUE attribute of that component. The VALUE_BY attribute indicates how new VALUE and VALUE* attributes are to be computed for each component, i. e. how the value of one component depends upon the values of other components. This is how the functional relationships between the components are represented and, as will be shown later, is the key to the error diagnosis.

Any entry other than "default" for the VALUE_BY attribute is the name of another relation in which this functional relationship is stored. Default indicates that this component is not affected by the other components during a simulation. For example, one tuple whose VALUE_BY attribute is default is "commanded_position". This is because the value of commanded_position is entered by the user before a simulation and is not changed until the user enters a new position. The rest of the tuples whose VALUE_BY attribute is default are included to indicate the status of the system components, either normal or defective in some way. The state of such components must be determined before executing a simulation and remain that way for duration of the simulation. These concepts will become clearer in the simulation examples.

2.4. Some Details About the Model

2.4.1. Representing the Jaw Position

There are three components associated with the position of the jaws. Two of these were mentioned in the description of the end effector system. They are "actual_position", which corresponds to the position register in the microprocessor, and "commanded_position", which corresponds to the user's input commanding the jaws to move to a specific location. The third component is "distance", which represents the observed distance between the jaws. This component is necessary to simulate failures in the system that result in a discrepancy between this observation and the value stored in the position register of the microprocessor. Although there is an equation available to convert a given encoder count position to a measurement of distance in inches, it was decided to eliminate this computation and to represent the observed distance between the jaws in terms of encoder counts. Also it was decided to represent all encoder counts as decimal rather than octal values. (To give the reader an idea of the scale of the laboratory system, at the fully open position, the jaws are approximately 3.25 inches apart.)

2.4.2. The Step Size of the Simulation

As stated in the introduction, machinery has been previously developed to drive a simulation of a system modelled using the RKBM format. It is useful to check the validity of a model by executing such simulations and comparing the results to the behavior of the physical system. It is also necessary to execute a simulation where one or more of the system components is in an error state in order to test and/or illustrate the usefulness of the error diagnosis algorithm presented later in this paper. An important consideration when developing the model was the determination of the step size of a simulation. To understand what is meant by this it becomes necessary to explain how the RKBM model is used to perform a simulation.

A simulation is executed by a routine that cycles through the end effector relation until a point is reached in which no changes are made to the data in the relation or until some maximum number of cycles are executed. A cycle consists of evaluating, for each tuple in the relation, the VALUE_BY attribute and storing the result in the VALUE attribute. At the end of a cycle, if, for every tuple, VALUE* (the previous VALUE) is equal to VALUE, then the simulation is ended and the system is said to be in an equilibrium state. If the system is not at equilibrium, the VALUE* attributes are updated with the entries in the VALUE attributes and another cycle is executed.

The question that needs to be answered is, "How much should the jaws move on each cycle through the relation?" One possibility is to have one simulation cycle be equivalent to one loop in the physical system, i. e. move the jaws one encoder count per cycle. However, given that there are 11357 encoder counts between the open and closed jaw positions, moving the jaws any significant distance would require too many simulation cycles. Another possibility is simulating the entire jaw movement in one simulation cycle, but this is not a very natural solution. The end effector is naturally an incremental system, where one component takes some input, processes it, and passes along its output as input to the next component. To determine the output of one component, it is only necessary to examine its input and to consider the states of a subset of the other components. Simulating the entire movement at once would require that the state of the entire system be considered to update each component.

The approach selected was a compromise of these two extreme solutions. It was decided to move the jaw in increments of 100 encoder counts. This step size is large enough to prevent extremely long simulations, yet small enough to realistically simulate the operation of the jaws under both normal and abnormal conditions. There is a slight trade-off for selecting this step size in order to have simulations of reasonable length. An error tolerance of 100 must be introduced, which limits the distance that the jaws can be requested to move to values over 100. Under normal conditions, if the jaws are commanded to move to a position less than 100 counts away from their current position, the current position immediately satisfies the error tolerance and the jaws will not move at all. However, it was decided that this is not a severe restriction for the purposes of this model. (There is one exception to this restriction. If there is a defective component in the system that will cause the jaws to be driven to the fully open or fully closed position, whether or not this is what was requested, the model will move the jaws all the way to position 0 or position -11357.)

2.4.3. Representing Error Conditions in the Model

In order to simulate the operation of the system when one or more components are defective, the effects of these defective conditions, or error states, had to be built into the VALUE_BY relations. Therefore, another decision that had to be made in setting up the model was which error states to attempt to simulate. Through several discussions with personnel in ATB and through the use of the "trouble-shooting" flowchart in Appendix D, a set of possible error states to include in the model was defined along with the effects these error states should have on the operation of the model. A list of the components that may be defective and their possible states, both normal and abnormal (error), is given in table 2.

2.4.4. An Order-Dependent versus an Order-Independent Model

By referencing the VALUE attribute, the evaluation of a component can incorporate new component values that have been computed earlier in the current simulation cycle. Referencing the VALUE* attribute prevents

Component States

Component Name	Normal State	Possible Error States
microprocessor	ok	bad constant_positive constant_negative
dac	ok	bad reverse constant_positive constant_negative
servo_amp	ok	bad reverse constant_positive constant_negative
motor_leads	ok	reverse
motor_circuit	closed	open
motor_switch	on	off
gears	ok	jammed
power_supply	on	off
short_circuit	no	yes
tachometer	ok	bad
shaft_encoder	ok	bad constant_positive constant_negative

Table 2. Possible Error States.

evaluations from accessing these new values until the next cycle, when VALUE* has been updated to VALUE. Therefore, referencing VALUE* will require more simulation cycles to accomplish a given amount of action than will referencing VALUE. The amount of difference depends on the order of the components in the relation when referencing VALUE. The most efficient ordering would place each component after components that it references and before components that reference it.

At first, it seemed better to leave any order dependence out of the system. Although this is less efficient, it has the benefit of preventing order dependence from being built into the model such that rearranging the order of the components would not only change the number of simulation cycles required, but would change the results of a simulation. Also, Blanks' models [2] were designed without order dependence so this did not appear to be an unnatural restriction.

However, the end effector proved to be very difficult, if not impossible, to model without referencing VALUE and therefore, having an order-dependent model. The problem involves the number of step by step computations involved in starting and stopping the incremental movement of the jaws. The model requires several steps for the signal from the microprocessor to reach the motor and, in turn, set the VALUE of jaws_to_move to "yes". Jaws_to_move was introduced as an intermediate computation between determining that the jaws should move and the actual movement. Without this intermediate calculation, the VALUE_BY relations for distance and actual_position would be so complicated as to be almost incomprehensible.

Once it is determined that the jaws should move (the value of jaws_to_move is "yes"), they move a fixed amount each time distance is re-evaluated. This occurs on every cycle until the value of jaws_to_move is changed to "no". If several cycles are required from determining that the jaws should not move until jaws_to_move is changed to "no", they will move several more increments than desired before the movement is actually stopped. After several unsuccessful attempts to alter the model in order to solve this problem without resorting to order dependence, the desire for having an order-independent model was re-examined. It was decided that the operation of the end effector system itself is naturally order-dependent and, therefore, there is no reason to force a model of the system to operate without order dependence. This aspect of the RKBM system should be studied further using several different models before drawing any definite conclusions about order dependence versus order independence.

2.4.5. Some Example Simulations

As stated earlier, a simulation is executed by cycling through the data base, evaluating the VALUE_BY attribute for each tuple and storing the result in the VALUE attribute. The VALUE and VALUE* attributes are then compared and if, for any tuple, VALUE is not equal to VALUE*, the the VALUE* attributes are updated with the entries in the VALUE attributes and another cycle is executed. If, for all tuples, VALUE is equal to VALUE*, then the system is said to be in a state of equilibrium and the simulation is ended.

Function "gotoit" controls the execution of a simulation and it is invoked along with a parameter "max" which indicates the upper bound on the number of cycles to perform. If equilibrium is not reached in max cycles, the simulation is ended, indicating that the number of cycles that have been executed has reached this upper bound. To inform the user about the progress of the simulation, the NAME, VARIABLE, and VALUE attributes of any tuple for which VALUE is not equal to VALUE* is printed for each cycle. Thus, by knowing the status of all components before the simulation begins, the user can determine the status of all components when it ends.

Figures 2 and 3 are examples of two simulation executions. Some extraneous output has been eliminated from the examples to conserve space. The complete output for each example can be seen in Appendices E and F, respectively.

In figure 2, the end effector begins at equilibrium with the jaws fully open (at position 0, as shown in table 1.) The commanded_position is changed to -433 and the simulation is begun with the call (gotoit 8). In the first cycle, the change in commanded_position is noted as well as the effects of this change on the other effector components. The signal from the microprocessor can be seen as it passes through the system. The cycle ends with jaws_to_move equal to "yes". On the next cycle, the jaws begin moving. This is reflected in the new actual_position location, distance amount, and error_signal direction. The next 2 cycles are similar in that the jaws move 100 more counts. On the fifth cycle, the jaws move to position -400, which is within 100 counts of the commanded_position. Since this satisfies the error tolerance, the error_signal from the microprocessor is turned off and the effect of this change on the rest of the components can be seen, ending with jaws_to_move equal to "no". Cycle 6 then detects that the system is at equilibrium and the simulation is ended.

```
? (update 'EFFECTOR '(equal (# NAME) 'commanded_position)
                    '(VALUE -433))
? (gotoit 8)
(CYCLE_NUMBER - 1)
(COMMANDED_POSITION LOCATION -433)
(ERROR_SIGNAL DIRECTION NEGATIVE)
(ERROR_SIGNAL MAGNITUDE 433)
(DAC_VOLTAGE DIRECTION NEGATIVE)
(MOTOR_VOLTAGE DIRECTION NEGATIVE)
(MOTOR_CURRENT CONDITION ON)
(JAWS_TO_MOVE CONDITION YES)
(CYCLE_NUMBER - 2)
(ACTUAL_POSITION LOCATION -100)
(DISTANCE AMOUNT -100)
(ERROR_SIGNAL MAGNITUDE 333)
(CYCLE_NUMBER - 3)
(ACTUAL_POSITION LOCATION -200)
(DISTANCE AMOUNT -200)
(ERROR_SIGNAL MAGNITUDE 233)
(CYCLE_NUMBER - 4)
(ACTUAL_POSITION LOCATION -300)
(DISTANCE AMOUNT -300)
(ERROR_SIGNAL MAGNITUDE 133)
(CYCLE_NUMBER - 5)
(ACTUAL_POSITION LOCATION -400)
(DISTANCE AMOUNT -400)
(ERROR_SIGNAL DIRECTION OFF)
(ERROR_SIGNAL MAGNITUDE 0)
(DAC_VOLTAGE DIRECTION OFF)
(SERVO_AMP_VOLTAGE OFF)
(MOTOR_VOLTAGE DIRECTION OFF)
(MOTOR_CURRENT CONDITION OFF)
(JAWS_TO_MOVE CONDITION NO)
(CYCLE_NUMBER - 6)
(-- AT EQUILIBRIUM --)
```

Figure 2. Normal Simulation

```
? (update ´EFFECTOR ´(equal (# NAME) ´commanded_position)
                     ´(VALUE -120))
? (update ´EFFECTOR ´(equal (# NAME) ´tachometer)
                     ´(VALUE ´bad))
? (gotoit 6)
(CYCLE NUMBER - 1)
(COMMANDED_POSITION LOCATION -120)
(ACTUAL_POSITION LOCATION -400)
(DISTANCE AMOUNT -400)
(ERROR_SIGNAL DIRECTION POSITIVE)
(ERROR_SIGNAL MAGNITUDE 280)
(DAC_VOLTAGE DIRECTION POSITIVE)
(SERVO_AMP_VOLTAGE DIRECTION POSITIVE)
(MOTOR_VOLTAGE DIRECTION POSITIVE)
(MOTOR_CURRENT CONDITION ON)
(JAWS_TO_MOVE CONDITION YES)
(TACHOMETER CONDITION BAD)
(CYCLE NUMBER - 2)
(ACTUAL_POSITION LOCATION -380)
(DISTANCE AMOUNT -300)
(ERROR_SIGNAL MAGNITUDE 260)
(CYCLE NUMBER - 3)
(ACTUAL_POSITION LOCATION -360)
(DISTANCE AMOUNT -200)
(ERROR_SIGNAL MAGNITUDE 240)
(CYCLE NUMBER - 4)
(ACTUAL_POSITION LOCATION -340)
(DISTANCE AMOUNT -100)
(ERROR_SIGNAL MAGNITUDE 220)
(CYCLE NUMBER - 5)
(ACTUAL_POSITION LOCATION -320)
(DISTANCE AMOUNT 0)
(ERROR_SIGNAL MAGNITUDE 200)
(CYCLE NUMBER - 6)
(-- AT EQUILIBRIUM --)
```

Figure 3. Abnormal Simulation

In figure 3, the end effector begins at equilibrium with the jaws at position -400. This time, however, the condition of the tachometer component is changed to "bad" before beginning the simulation. When the tachometer is not working properly, the jaws move too quickly for the shaft encoder to encode the movement. Therefore, the position register in the microprocessor is not updated properly and the jaws move too far, usually either all the way to the fully open or fully closed position, depending on the direction of travel. To simulate the bad tachometer, it was decided to assume that the position register is only updated 20 counts for every 100 counts of jaw movement. (The number 20 was chosen arbitrarily - the goal is to show that the jaws will end up in the wrong position and the exact location of this position is not important.) The commanded_position is set to -120. On a normal simulation, where none of the components is defective, the jaws should move to position -200 and the system should return to a normal state of equilibrium, i.e. the jaws are not moving and there is no voltage signal in any of the components. However, as the simulation indicates, the jaws move past position -200, all the way to position 0, where they can move no farther. At this point the system is in equilibrium - no VALUE attribute will change no matter how many cycles are executed. However, the system is in an abnormal state. There is still a positive error signal which occurs all the way through the system because the microprocessor program thinks the jaws are at position -320. The system will remain in this state until something or someone intervenes to correct it.

3. Error Diagnosis Using the RKBM Model

After completing the RKBM end effector model, the remaining task was to determine how to use the information contained in the model to answer questions about the state of the system. The question of particular interest in error diagnosis is, "Why does component "x" have value "y" when it should have value "z"?" The answer might be something like, "because component "q" is defective." Since, in the RKBM model, nothing is defective unless it is specified as such, the reader may question the need for an algorithm to detect a defective component. However, an RKBM-based expert system working in cooperation with an operational system would require such an algorithm.

3.1. The RKBM Model and the Operational System

The ultimate goal of this research is to have an RKBM model residing on a microprocessor which is physically connected to the operational system represented by the model. As the system operates, the model will be updated according to the values of the system components. For example, the microprocessor on which the end effector model resides will have a direct connection to the output of the end effector controller microprocessor to detect the error signal output by the controller microprocessor. Similar connections will exist for all components of the RKBM model whose VALUE_BY attribute is not default. Consequently, these components of the model will be updated by the operation of the actual system, rather than the functional relationships in the model that are used to drive a simulation. There will be no such connections for components whose VALUE_BY attribute is default and, therefore, these components will not be updated.

When the operational system malfunctions, the error diagnosis algorithm will be invoked to trace backwards through the functional description of the model to find the cause of the malfunction. As the trace through the derivation of a component is performed and a component whose VALUE attribute is derived by default is encountered, its current value will not be found in the model. Human intervention will be required to check that component and inform the expert system of the state of the component. In performing these checks, the human user will eventually discover the component responsible for the malfunction. Because of the required human intervention, the algorithm proposed here is one for error diagnosis guided by an expert system rather than a completely automatic error diagnosis scheme.

3.2. An Algorithm for Error Diagnosis

The algorithm presented is defined by the recursive function "examine_origin". Examine_origin performs a backwards trace on the derivation of the VALUE attribute of its component argument and returns the entry in this VALUE attribute. During this trace, the states of all components that directly or indirectly (through the recursion) determine the value of the component under examination are revealed. Therefore, if there is a defective component that could have caused the incorrect value of the component under examination, it will be discovered during the trace.

Two slightly different versions of examine_origin are presented here, although both give the same results for the end effector example upon which they are demonstrated. Method I represents the algorithm originally tested and method II is an adaptation. The difference between the two methods is noted as each is described. It is believed that this difference would not be significant for any example involving the end effector model, however, this has not been proven. It is also unknown as to whether significantly different results would be produced by the two methods if they were tested on a different RKBM model.

3.2.1. Method I

The algorithm for method I is shown in figure 4. The first step of examine_origin is to add its argument to the global examined list. (This list prevents a component from being examined more than once and thus prevents the trace from endlessly looping.) Next, if the VALUE attribute of the component is derived by default, its value is returned and examine_origin is finished. However, if the VALUE attribute of the component is not derived by default, the relation used to derive VALUE must be examined. Each relation consists of a set of boolean expression (b_expr) - value expression (v_expr) pairs. At the time the relation is used to derive the value of a component, only one b_expr is true and its corresponding v_expr is used to derive the value. Examine_origin uses "evaluate_b_expr" on each b_expr in the relation until the true b_expr is found. Evaluate_b_expr uses "is" to check as many subconditions of the b_expr as necessary to determine its truth or falsehood. Examine_origin is then invoked for each unexamined component in the true b_expr and corresponding v_expr. As the value of a component is returned by examine_origin, it is checked against a set of acceptable values for that component by "in_range". Table 3 shows the acceptable value range

```
examine_origin (component);  (* returns value of component *)
begin
   add component to examined list;
   if component value is derived by default then
      return value
   else begin  (* component value is derived by relation *)
      repeat
         evaluate_b_expr (b_expr_number)
      until a true b_expr is found;
      for each component in the true b_expr and corresponding
                                                      v_expr do
         if component not already examined then
            if not inrange (component,examine_origin(component))
                                                              then
               add component to possible malfunction list;
      return value;
      end
end; (* examine_origin *)

evaluate_b_expr (b_expr_number);  (* returns true if b_expr   *)
                                  (* is true, false otherwise *)
begin
   repeat
      is (condition of b_expr)
   until b_expr can be determined true or false;
   return true or false
end; (* evaluate_b_expr *)

is (condition of b_expr);  (* returns true if condition *)
                           (* is true, false otherwise  *)
begin
   look in database for value(s) of component(s) in condition
                                                   of b_expr;
   use value(s) to determine if condition is true or false;
   return true or false
end; (* is *)

inrange (component,value);  (* returns true if value is valid *)
                            (* range for component, false     *)
                            (* otherwise                      *)
```

Figure 4. Method I - Algorithm for Error Detection.

for each component of the end effector. If a component's value is not in this acceptable range, the component is added to a possible

Value Ranges for End Effector Components

commanded_position	0 to -11357
actual_position	0 to -11357
distance	0 to -11357
error_signal direction	positive, negative, off
error_signal magnitude	-11357 to +11357
dac	ok
dac_voltage	positive, negative, off
servo_amp	ok
servo_amp_voltage	positive, negative, off
motor_leads	ok
motor_circuit	closed
motor_current	on, off
motor_switch	on
motor_voltage	positive, negative, off
jaws_to_move	yes, no
gears	ok
power_supply	on
short_circuit	no
tachometer	ok
shaft_encoder	ok

Table 3. Value Ranges for End Effector Components.

malfunction list. When the original invocation of examine_origin is finished, the possible malfunction list should contain any components whose out-of-range values could have caused the problem in the component under examination.

Appendix G contains a trace of the recursive calls to examine_origin when examine_origin(distance) is invoked after the previously discussed simulation with the bad tachometer in figure 3. Examine_origin(distance) is invoked because the problem here is that distance does not have the value that was commanded.

3.2.2. Method II

The algorithm using method II is shown in figure 5. Method II differs from method I in that, as each b_expr is evaluated, examine_origin is invoked for the components in the b_expr. This seems more logical than simply inquiring about the value of a component at one point and later examining it. However, it could also cause unnecessary examination of some of the components in a false b_expr since the components examined before determining that the b_expr is false may not have any influence on the derivation of the value, i.e. the true b_expr and corresponding v_expr may not involve these components.

Appendix H contains a trace of examine_origin using method II for the same bad tachometer example used to illustrate method I.

```
examine_origin (component);  (* returns value of component *)
begin
   add component to examined list;
   if component value is derived by default then
      return value
   else begin  (* component value is derived by relation *)
      repeat
         evaluate_b_expr (b_expr_number)
      until a true b_expr is found;
      for each component in the corresponding v_expr do
         if component not already examined then
            if not inrange (component,examine_origin(component))
                                                            then
               add component to possible malfunction list;
      return value;
      end
end; (* examine_origin *)

evaluate_b_expr (b_expr_number);  (* returns true if b_expr   *)
begin                             (* is true, false otherwise *)
   repeat
      is (condition of b_expr)
   until b_expr can be determined true or false;
   return true or false
end; (* evaluate_b_expr *)

is (condition of b_expr);  (* returns true if condition *)
begin                      (* is true, false otherwise  *)
   for each component in condition of b_expr do
      if component of condition of b_expr not already examined
                                                    then begin
         use examine_origin(component) to get value of component;
         if not inrange(component,value) then
            add component to possible malfunction list
         end
      else
         look up value of component in database;
   use value(s) to determine if condition of b_expr is true or
                                                      false;
   return true or false
end; (* is *)

inrange (component,value);   (* returns true if value is valid *)
                             (* range for component, false     *)
                             (* otherwise                      *)
```

Figure 5. Method II - Algorithm for Error Detection.

4. Conclusions

The RKBM knowledge representation format has proven to be usable as the basis of an expert diagnostic system. However, the method for using an RKBM description in connection with an operational system has been only loosely defined. More research is required in several areas before such an expert system becomes a reality.

First, the RKBM approach to modelling a system requires further examination. Problems that have been encountered in modelling the end effector may be non-existent when modelling other systems. In particular, the question of an order-dependent versus an order-independent model may not be a significant issue for other systems. More experimentation should be done to define classes of systems and corresponding methods for best describing them using the RKBM approach.

Another area requiring further research concerns the two versions of the error diagnosis algorithm presented here. Several other RKBM models should be examined using both versions of the algorithm. Although the difference between the two versions is insignificant for the end effector model, error diagnosis using another RKBM model may reveal that this difference has some significance.

Finally, further examination of the relationships between the model, the operational system, the diagnosis algorithm, and the human user is required. As stated earlier, the expert system proposed here is not a completely automated error diagnosis system. Human intervention is a critical component of the error diagnosis scheme. The way in which the diagnostic algorithm and the model are connected to the operational system and the way in which information is requested and received from the user must be defined more specifically. This area of research necessarily includes the development of an implementation of the error diagnosis system.

5. References

[1] Davis, R., "Diagnosis via causal reasoning: Paths of Interaction and the Locality Principle", Proceedings of the National Conference on Artificial Intelligence, August 22-26, 1983, pp. 88-94.

[2] Blanks, M., "Relational Knowledge-Base Machines - A New Approach to Computer Problem Solving", Honors thesis for B. S. Computer Science, College of William and Mary, Williamsburg, Va., April 1983.

Appendix A: Prime LISP versus UTLISP

Before beginning development of the end effector model, the RKBM LISP functions had to be rewritten from Prime LISP to UTLISP for execution on the CDC Network Operating System. This appendix is intended to explain the differences between Blanks' implementation for Prime LISP and the UTLISP implementation presented here. It is also intended to aid the reader interested in the implementation of an RKBM system in porting the LISP routines provided here to another LISP implementation.

Minor changes to Blanks' implementation include:

(1) The SELECT function has been renamed SELEKT to avoid conflicts with the UTLISP SELECT function.

(2) The Prime LISP functions SDEFUN and SNDEFUN are equivalent to the UTLISP functions DEF and DEFF, respectively. The later functions in each pair are used to define other functions whose arguments will not be evaluated upon invocation of the function.

(3) The purpose of the fourth argument in Blanks' UPDATE function could not be determined and this argument has therefore been removed.

More substantial changes to Blanks' implementation include:

(1) The functions have been changed from `pure' LISP to include the use of iteration for the sake of clarity.

(2) The Prime LISP GET and PUT functions for property lists are simulated by GETV and PUTV in the UTLISP implementation. Prime LISP

allows any list with an even number of elements, regardless of how it is generated, to be operated upon by GET and PUT as a property list. UTLISP has a stricter implementation of property lists - only true property lists can be operated upon as property lists and a property list does not have the same structure as an ordinary list.

(3) The inability to manipulate property lists as in Prime LISP required a change in the UPDATE function so that when a component of the system is updated, the entire relation is replaced. This is necessary so that components evaluated later in the database relation will have access to values that have been updated earlier in the same cycle through the database. The Prime LISP property list functions evidently manipulate the internal structure of the list, thus making a replacement of the entire list unnecessary. Without this change to the UTLISP implementation, an order-dependent model as described in section 2.3.4 would have been impossible .

(4) Since property list GET and PUT functions are no longer being used, it is no longer necessary for each tuple in the representation of the relation to contain attribute - value pairs. Instead, the first tuple in a relation is a list of attributes and each tuple contains a list of values corresponding to the attributes.

(5) A function GOTOIT has been added that accepts an argument `max'. As described in section 2.3.5, GOTOIT drives a simulation of the RKBM end effector system.

(6) Finally, the attribute DERIVED has been eliminated because the information it contains can be found in the VALUE_BY attribute. A function VALUEFUNCTION has been added to be used by UPDATE within GOTOIT to determine how to evaluate a new value for a component, depending on whether value_by is equal to `default´, a relation name, or an expression. (By representing expressions as relations with one boolean expression - value expression pair, there are only two possible entries for value_by - default or a relation name.)

Appendix B: RKBM Lisp Routines

This appendix contains the lisp routines that execute the RKBM system.

```
(DEF (GETV (LISS ATTLIST KEY)
     % LOCATES KEY IN ATTLIST, THEN RETURNS VALUE THAT %
     % OCCUPIES CORRESPONDING POSITION IN LISS        %
     % - INTENDED TO SIMULATE PRIME LISP GET FUNCTION  %
     % FOR PROPERTY LIST VALUES                        %
     (COND ((EQUAL (CAR ATTLIST) KEY) (CAR LISS))
           (T (GETV (CDR LISS) (CDR ATTLIST) KEY))
))   )
(DEF (PUTV (LISS ATTLIST KEY VALUE)
     % LOCATES KEY IN ATTLIST, THEN REPLACES VALUE IN %
     % CORRESPONDING POSITION IN LISS - INTENDED TO   %
     % SIMULATE PRIME LISP PUT FUNCTION FOR PROPERTY  %
     % LIST VALUES                                     %
     (COND ((EQUAL (CAR ATTLIST) KEY)
             (CONS VALUE (CDR LISS))
           )
           (T (CONS (CAR LISS) (PUTV (CDR LISS)
                                     (CDR ATTLIST)
                                     KEY
                                     VALUE
))   )   )    )                 )
(DEFF (# (X)
      (GETV (CAR TUPLES) ATTRIBUTES (CAR X))
))
(DEF (SELEKT (RELATION SCRIT PCRIT)
     % DATABASE SELECT QUERY - FOR EVERY TUPLE IN RELATION %
     % FOR WHICH SCRIT IS TRUE, A TUPLE IS CREATED         %
     % CONTAINING THE VALUES SPECIFIED IN PCRIT.  THESE    %
     % TUPLES ARE GATHERED INTO A NEW RELATION THAT IS     %
     % RETURNED BY SELEKT.                                 %
     (PROG (ATTRIBUTES TUPLES SELEKTION)
           (SETQ ATTRIBUTES (CAR RELATION))
           (SETQ TUPLES (CDR RELATION))
           LOOP
           (COND % END OF RELATION - RETURN NEW RELATION TUPLES %
                 ((NULL TUPLES) (RETURN (REVERSE SELEKTION)))
                 % SCRIT TRUE - ADD NEW TUPLE TO NEW RELATION %
                 ((EVAL SCRIT) (SETQ SELEKTION
                                     (CONS (MAPCAR PCRIT 'EVAL) SELEKTION)
                               )
                 )
           )
           (SETQ TUPLES (CDR TUPLES)) % NEXT TUPLE %
           (GO LOOP)
))   )
(DEF (SELECT1 (RELATION SCRIT PCRIT)
     % SIMILAR TO SELEKT EXCEPT ONLY RETURNS     %
```

```
            % ONE TUPLE - FOR FIRST TUPLE FOUND FOR     %
            % WHICH SCRIT IS TRUE.                     %
            (PROG (ATTRIBUTES TUPLES SELEKTION)
                  (SETQ ATTRIBUTES (CAR RELATION))
                  (SETQ TUPLES (CDR RELATION))
                  LOOP
                  (COND % END OF RELATION - RETURN %
                        ((NULL TUPLES) (RETURN NIL))
                        % SCRIT TRUE - RETURN NEW TUPLE %
                        ((EVAL SCRIT) (RETURN (EVAL (CAR PCRIT))))
                  )
                  (SETQ TUPLES (CDR TUPLES)) % NEXT TUPLE %
                  (GO LOOP)
      ))    )
      (DEF (PROJECT (RELATION SCRIT PCRIT)
            % DATABASE PROJECT QUERY - DISPLAYS NEW RELATION AS %
            % WOULD BE CONSTRUCTED AND RETURNED BY SELEKT BUT   %
            % RETURNS NIL.                                      %
            (PROG (ATTRIBUTES TUPLES)
                  (SETQ ATTRIBUTES (CAR RELATION))
                  (SETQ TUPLES (CDR RELATION))
                  LOOP
                  (COND % END OF RELATION - RETURN %
                        ((NULL TUPLES) (PRINT ´(--END PROJECT))
                                       (RETURN NIL)
                        )
                        % SCRIT TRUE - DISPLAY NEW TUPLE %
                        ((EVAL SCRIT) (PRINT (MAPCAR PCRIT ´EVAL)))
                  )
                  (SETQ TUPLES (CDR TUPLES)) % NEXT TUPLE %
                  (GO LOOP)
      ))    )
      (DEF (UPDATE (RELATION SCRIT APAIR)
            % DATABASE UPDATE QUERY - FOR EACH TUPLE IN RELATION %
            % FOR WHICH SCRIT IS TRUE, REPLACE THE VALUE OF      %
            % ATTRIBUTE (CAR APAIR) WITH EVALUATION OF THE       %
            % SECOND ITEM IN APAIR (CADR APAIR).                 %
            (PROG (ATTRIBUTES TUPLES UPDATED REL)
                  (SETQ REL (EVAL RELATION))
                  (SETQ ATTRIBUTES (CAR REL))
                  (SETQ TUPLES (CDR REL))
                  (SETQ UPDATED (LIST ATTRIBUTES))
                  LOOP
                  (COND % END OF RELATION - RETURN %
                        ((NULL TUPLES) (PRINT ´(--END UPDATE))
                                       (RETURN NIL)
                        )
                        % SCRIT TRUE - UPDATE ATTRIBUTE %
                        ((EVAL SCRIT) (SETQ UPDATED (CONS (PUTV (CAR TUPLES)
                                                                ATTRIBUTES
                                                                (CAR APAIR)
                                                                (EVAL (CADR APAIR))
                                                          )
```

```
                                            UPDATED
                              )          )
                              % REPLACE ENTIRE RELATION - NECESSARY SO %
                              % LATER TUPLES DURING THIS UPDATE CAN    %
                              % ACCESS NEW ATTRIBUTE OF THIS TUPLE.    %
                              (SET RELATION (APPEND (REVERSE UPDATED)
                                             (CDR TUPLES)
                     )        )          )
                     % SCRIT FALSE - GATHER OLD TUPLE INTO UPDATED LIST %
                     (T (SETQ UPDATED (CONS (CAR TUPLES) UPDATED)))
                )
                (SETQ TUPLES (CDR TUPLES)) % NEXT TUPLE %
                (GO LOOP)
))    )
(DEF (NOTEQ (X Y)
     (COND ((EQUAL X Y) NIL)
           (T T)
     )
))
(DEF (GTREQ (X Y)
     (COND ((LESSP X Y) NIL)
           (T T)
     )
))
(DEF (LSSEQ (X Y)
     (COND ((GREATERP X Y) NIL)
           (T T)
     )
))
(DEF (ABS (X)
     (COND ((GTREQ X 0) X)
           (T (MINUS X))
     )
))
(DEF (EVLX (X)
     (COND ((ATOM X) X)
           (T (EVAL X))
     )
))
(DEF (GOTOIT (MAX)
     % CYCLES THROUGH END_EFFECTOR DATABASE UPDATING VALUE %
     % ATTRIBUTES.  STOPS WHEN 2 CONSECUTIVE CYCLES        %
     % PRODUCE EXACTLY THE SAME VALUES (I.E. VALUE =       %
     % VALUE* SO EQUILIBRIUM IS REACHED) OR MAX IS REACHED %
     (PROG (KNT)
           (SETQ KNT 1)
           LOOP
           (PRINT (APPEND ´(CYCLE NUMBER -) (LIST KNT)))
           (UPDATE ´EFFECTOR ´T ´(VALUE (VALUEFUNCTION)))
           (COND % ALL VALUES=VALUE*S - STOP %
                 ((EQUAL (SELEKT EFFECTOR ´T ´((# VALUE)))
                         (SELEKT EFFECTOR ´T ´((# VALUE*)))
                  )
                  (PRINT ´(-- AT EQUILIBRIUM --))
                  (RETURN NIL)
            )     )
            % DISPLAY ALL VALUES CHANGED ON THIS CYCLE %
            (PROJECT EFFECTOR ´(NOTEQ (# VALUE*) (# VALUE))
                              ´((# NAME) (# VARIABLE) (# VALUE))
            )
            % UPDATE VALUE* FIELDS = VALUE FIELDS %
            (UPDATE ´EFFECTOR ´T ´(VALUE* (# VALUE)))
            (SETQ KNT (PLUS KNT 1))
            (COND % MAX IS REACHED - STOP %
                  ((GREATERP KNT MAX) (PRINT ´(-- MAX CYCLE REACHED --))
                                      (RETURN NIL)
            )     )
            (GO LOOP)
))  )
(DEF (VALUEFUNCTION ()
     % FUNCTION TO DETERMINE HOW TO DERIVE NEW VALUE %
     % ATTRIBUTE IN RELATION                         %
     (COND % VALUE DERIVED BY DEFAULT %
            ((EQUAL (# VALUE_BY) ´DEFAULT) (# VALUE))
            % VALUE DERIVED BY RELATION %
            ((ATOM (# VALUE_BY)) (SELECT1 (EVAL (# VALUE_BY))
                                          ´(EQUAL (EVLX (# B_EXPR)) ´T)
                                          ´((EVLX (# V_EXPR)))
            )                       )
            % VALUE DERIVED BY SINGLE EXPRESSION %
            (T (EVAL (# VALUE_BY)))
))  )
```

Appendix C: The RKBM End Effector Model

This appendix contains the lisp code that defines the RKBM end effector model.

```
(SETQ EFFECTOR '(
  ( NAME
    VARIABLE
    PARENT
    VALUE_BY
    VALUE*
    VALUE
  )
  ( MICROPROCESSOR
    CONDITION
    SYSTEM
    DEFAULT
    OK
    OK
  )
  ( COMMANDED_POSITION
    LOCATION
    MICROPROCESSOR
    DEFAULT
    0
    0
  )
  ( ACTUAL_POSITION
    LOCATION
    MICROPROCESSOR
    ACTUAL
    0
    0
  )
  ( DISTANCE
    AMOUNT
    SYSTEM
    DISTANCE_AMT
    0
    0
  )
  ( ERROR_SIGNAL
    DIRECTION
    MICROPROCESSOR
    ERR_SIG_DIR
    OFF
    OFF
  )
  ( ERROR_SIGNAL
    MAGNITUDE
    MICROPROCESSOR
  ERR_SIG_MAG
  0
  0
)
( DAC
  CONDITION
  SYSTEM
  DEFAULT
  OK
  OK
)
( DAC_VOLTAGE
  DIRECTION
  DAC
  DAC_VOLT
  OFF
  OFF
)
( SERVO_AMP
  CONDITION
  SYSTEM
  DEFAULT
  OK
  OK
)
( SERVO_AMP_VOLTAGE
  DIRECTION
  SERVO_AMP
  SERVO_VOLT
  OFF
  OFF
)
( MOTOR_LEADS
  CONDITION
  SYSTEM
  DEFAULT
  OK
  OK
)
( MOTOR_CIRCUIT
  CONDITION
  MOTOR
  DEFAULT
  CLOSED
  CLOSED
)
( MOTOR_SWITCH
  CONDITION
  MOTOR
  DEFAULT
  ON
  ON
)
```

```
( MOTOR_VOLTAGE
  DIRECTION
  MOTOR
  MOTOR_VOLT
  OFF
  OFF
)
( MOTOR_CURRENT
  CONDITION
  MOTOR
  CURRENT
  OFF
  OFF
)
( JAWS_TO_MOVE
  CONDITION
  SYSTEM
  MOVE_JAWS
  NO
  NO
)
( GEARS
  CONDITION
  SYSTEM
  DEFAULT
  OK
  OK
)
( POWER_SUPPLY
  CONDITION
  SYSTEM
  DEFAULT
  ON
  ON
)
( SHORT_CIRCUIT
  CONDITION
  SYSTEM
  DEFAULT
  NO
  NO
)
( TACHOMETER
  CONDITION
  SYSTEM
  DEFAULT
  OK
  OK
)
( SHAFT_ENCODER
  CONDITION
  SYSTEM
  DEFAULT
```

```
      OK
      OK
    )
  ))
  (SETQ MOTOR_VOLT ´(
    (V_EXPR B_EXPR)
    ((SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP_VOLTAGE) ´((# VALUE)))
     (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´POWER_SUPPLY) ´((# VALUE)))
                 ´ON
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_LEADS) ´((# VALUE)))
                 ´OK
    ))    )
    (OFF
     (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´POWER_SUPPLY) ´((# VALUE)))
                ´OFF
         )
         (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP_VOLTAGE)
                                   ´((# VALUE))
                )
                ´OFF
    ))   )
    (POSITIVE
     (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´POWER_SUPPLY) ´((# VALUE)))
                 ´ON
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_LEADS) ´((# VALUE)))
                 ´REVERSE
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP_VOLTAGE)
                                    ´((# VALUE))
                 )
                 ´NEGATIVE
    ))    )
    (NEGATIVE
     (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´POWER_SUPPLY) ´((# VALUE)))
                 ´ON
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_LEADS) ´((# VALUE)))
                 ´REVERSE
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP_VOLTAGE)
                                    ´((# VALUE))
                 )
                 ´POSITIVE
    ))    )
  ))
  (SETQ SERVO_VOLT ´(
    (V_EXPR B_EXPR)
    ((SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC_VOLTAGE) ´((# VALUE)))
     (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP) ´((# VALUE)))
            ´OK
    ))
    (OFF
     (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP) ´((# VALUE)))
            ´BAD
    ))
    (POSITIVE
      (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP) ´((# VALUE)))
                 ´CONSTANT_POSITIVE
          )
          (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP) ´((# VALUE)))
                      ´REVERSE
               )
               (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC_VOLTAGE)
                                        ´((# VALUE))
                      )
                      ´NEGATIVE
    ))    )    )
    (NEGATIVE
      (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP) ´((# VALUE)))
                 ´CONSTANT_NEGATIVE
          )
          (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SERVO_AMP) ´((# VALUE)))
                      ´REVERSE
               )
               (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC_VOLTAGE)
                                        ´((# VALUE))
                      )
                      ´POSITIVE
    ))    )    )
 ))
 (SETQ DAC_VOLT ´(
   (V_EXPR B_EXPR)
   ((SELECT1 EFFECTOR ´(AND (EQUAL (# NAME) ´ERROR_SIGNAL)
                            (EQUAL (# VARIABLE) ´DIRECTION)
                       )
                      ´((# VALUE))
    )
    (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC) ´((# VALUE)))
           ´OK
   ))
   (OFF
    (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC) ´((# VALUE)))
               ´BAD
        )
        (EQUAL (SELECT1 EFFECTOR ´(AND (EQUAL (# NAME) ´ERROR_SIGNAL)
                                       (EQUAL (# VARIABLE) ´DIRECTION)
                                  )
                                  ´((# VALUE))
               )
               ´NIL
   ))   )
   (POSITIVE
    (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC) ´((# VALUE)))
               ´CONSTANT_POSITIVE
        )
        (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC) ´((# VALUE)))
                    ´REVERSE
             )
             (EQUAL (SELECT1 EFFECTOR ´(AND (EQUAL (# NAME) ´ERROR_SIGNAL)
                                            (EQUAL (# VARIABLE) ´DIRECTION)
                                       )
                                      ´((# VALUE))
                    )
                    ´NEGATIVE
    ))  )   )
    (NEGATIVE
     (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC) ´((# VALUE)))
                ´CONSTANT_NEGATIVE
         )
         (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DAC) ´((# VALUE)))
                     ´REVERSE
              )
              (EQUAL (SELECT1 EFFECTOR ´(AND (EQUAL (# NAME) ´ERROR_SIGNAL)
                                             (EQUAL (# VARIABLE) ´DIRECTION)
                                        )
                                       ´((# VALUE))
                     )
                     ´POSITIVE
    ))  )   )
  ))
  (SETQ CURRENT ´(
    (V_EXPR B_EXPR)
    (ON
     (AND (NOTEQ (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
                 ´OFF
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_CIRCUIT) ´((# VALUE)))
                 ´CLOSED
    ))   )
    (OFF
     (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
                ´OFF
         )
         (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_CIRCUIT) ´((# VALUE)))
                ´OPEN
    ))   )
  ))
  (SETQ ERR_SIG_MAG ´(
    (V_EXPR B_EXPR)
    (0
     (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MICROPROCESSOR) ´((# VALUE)))
                ´BAD
         )
         (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MICROPROCESSOR)
                                       ´((# VALUE))
                     )
                     ´OK
```

```
            )
            (LESSP (ABS (DIFFERENCE (SELECT1 EFFECTOR
                                             '(EQUAL (# NAME) 'ACTUAL_POSITION)
                                             '((# VALUE))
                                    )
                                    (SELECT1 EFFECTOR
                                             '(EQUAL (# NAME)
                                                     'COMMANDED_POSITION
                                              )
                                             '((# VALUE))
                   )     )          )
                   100
  ))   )    )
  (1200
   (OR (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR) '((# VALUE)))
              'CONSTANT_POSITIVE
       )
       (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR) '((# VALUE)))
              'CONSTANT_NEGATIVE
  ))   )
  ((ABS (DIFFERENCE (SELECT1 EFFECTOR '(EQUAL (# NAME) 'ACTUAL_POSITION)
                                      '((# VALUE))
                    )
                    (SELECT1 EFFECTOR '(EQUAL (# NAME) 'COMMANDED_POSITION)
                                      '((# VALUE))
   )    )           )
   (AND (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR) '((# VALUE)))
               'OK
        )
        (GTREQ (ABS (DIFFERENCE (SELECT1 EFFECTOR
                                            '(EQUAL (# NAME) 'ACTUAL_POSITION)
                                            '((# VALUE))
                                   )
                                   (SELECT1 EFFECTOR
                                            '(EQUAL (# NAME)
                                                    'COMMANDED_POSITION
                                             )
                                            '((# VALUE))
                  )    )           )
                  100
  ))      )
))
(SETQ ERR_SIG_DIR '(
  (V_EXPR B_EXPR)
  (OFF
   (OR (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR) '((# VALUE)))
              'BAD
       )
       (AND (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR)
                                     '((# VALUE))
                   )
                   'OK
            )
            (LESSP (ABS (DIFFERENCE (SELECT1 EFFECTOR
                                             '(EQUAL (# NAME) 'ACTUAL_POSITION)
                                             '((# VALUE))
                                    )
                                    (SELECT1 EFFECTOR
                                             '(EQUAL (# NAME)
                                                     'COMMANDED_POSITION
                                              )
                                             '((# VALUE))
                   )    )           )
                   100
  ))   )    )
  (POSITIVE
   (OR (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR) '((# VALUE)))
              'CONSTANT_POSITIVE
       )
       (AND (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR)
                                     '((# VALUE))
                   )
                   'OK
            )
            (GTREQ (DIFFERENCE (SELECT1 EFFECTOR
                                        '(EQUAL (# NAME) 'COMMANDED_POSITION)
                                        '((# VALUE))
                               )
                               (SELECT1 EFFECTOR
                                        '(EQUAL (# NAME) 'ACTUAL_POSITION)
                                        '((# VALUE))
                   )           )
                   100
  ))   )    )
  (NEGATIVE
   (OR (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR) '((# VALUE)))
              'CONSTANT_NEGATIVE
       )
       (AND (EQUAL (SELECT1 EFFECTOR '(EQUAL (# NAME) 'MICROPROCESSOR)
                                     '((# VALUE))
                   )
                   'OK
            )
            (GTREQ (DIFFERENCE (SELECT1 EFFECTOR
                                        '(EQUAL (# NAME) 'ACTUAL_POSITION)
                                        '((# VALUE))
                               )
                               (SELECT1 EFFECTOR
                                        '(EQUAL (# NAME) 'COMMANDED_POSITION)
                                        '((# VALUE))
                   )           )
                   100
  ))   )    )
))
(SETQ DISTANCE_AMT '(
  (V_EXPR B_EXPR)
```

```
   ((DIFFERENCE (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
                100
    )
    (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
                ´YES
         )
         (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
                ´NEGATIVE
         )
         (GREATERP (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
                   -11258
   ))    )
   ((PLUS (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
          100
    )
    (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
                ´YES
         )
         (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
                ´POSITIVE
         )
         (LESSP (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))

                -100
   ))    )
   (-11357
    (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
                ´YES
         )
         (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
                ´NEGATIVE
         )
         (LSSEQ (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
                -11258
   ))    )
   (0
    (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
                ´YES
         )
         (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
                ´POSITIVE
         )
         (GTREQ (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
                -100
   ))    )
   ((SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
    (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
           ´NO
   ))
  ))
  (SETQ ACTUAL ´(
   (V_EXPR B_EXPR)
   ((DIFFERENCE (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´ACTUAL_POSITION)
                                        ´((# VALUE))
                 )
                 100
     )
     (OR (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE)
                                        ´((# VALUE))
                     )
                     ´YES
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´TACHOMETER) ´((# VALUE)))
                     ´OK
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER)
                                        ´((# VALUE))
                     )
                     ´OK
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE)
                                        ´((# VALUE))
                     )
                     ´NEGATIVE
              )
              (GREATERP (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE)
                                           ´((# VALUE))
                        )
                        -11258
         )    )
         (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER) ´((# VALUE)))
                ´CONSTANT_NEGATIVE
    ))   )
    ((PLUS (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´ACTUAL_POSITION) ´((# VALUE)))
           100
     )
     (OR (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE)
                                        ´((# VALUE))
                     )
                     ´YES
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´TACHOMETER) ´((# VALUE)))
                     ´OK
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER)
                                        ´((# VALUE))
                     )
                     ´OK
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE)
                                        ´((# VALUE))
                     )
                     ´POSITIVE
              )
              (LESSP (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE)
                                        ´((# VALUE))
```

```
                   )
                   -100
      )    )
      (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER) ´((# VALUE)))
             ´CONSTANT_POSITIVE
 ))   )
 ((DIFFERENCE (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´ACTUAL_POSITION)
                                ´((# VALUE))
              )
              20
  )
  (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
              ´YES
       )
       (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´TACHOMETER) ´((# VALUE)))
              ´BAD
       )
       (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER) ´((# VALUE)))
              ´OK
       )
       (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
              ´NEGATIVE
       )
       (NOTEQ (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
              -11357
 ))    )
 ((PLUS (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´ACTUAL_POSITION) ´((# VALUE)))
              20
  )
  (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
              ´YES
       )
       (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´TACHOMETER) ´((# VALUE)))
              ´BAD
       )
       (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER) ´((# VALUE)))
              ´OK
       )
       (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
              ´POSITIVE
       )
       (NOTEQ (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
              0
 ))    )
 (-11357
  (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
              ´YES
       )
       (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´TACHOMETER) ´((# VALUE)))
              ´OK
       )
       (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER) ´((# VALUE)))
              ´OK
```

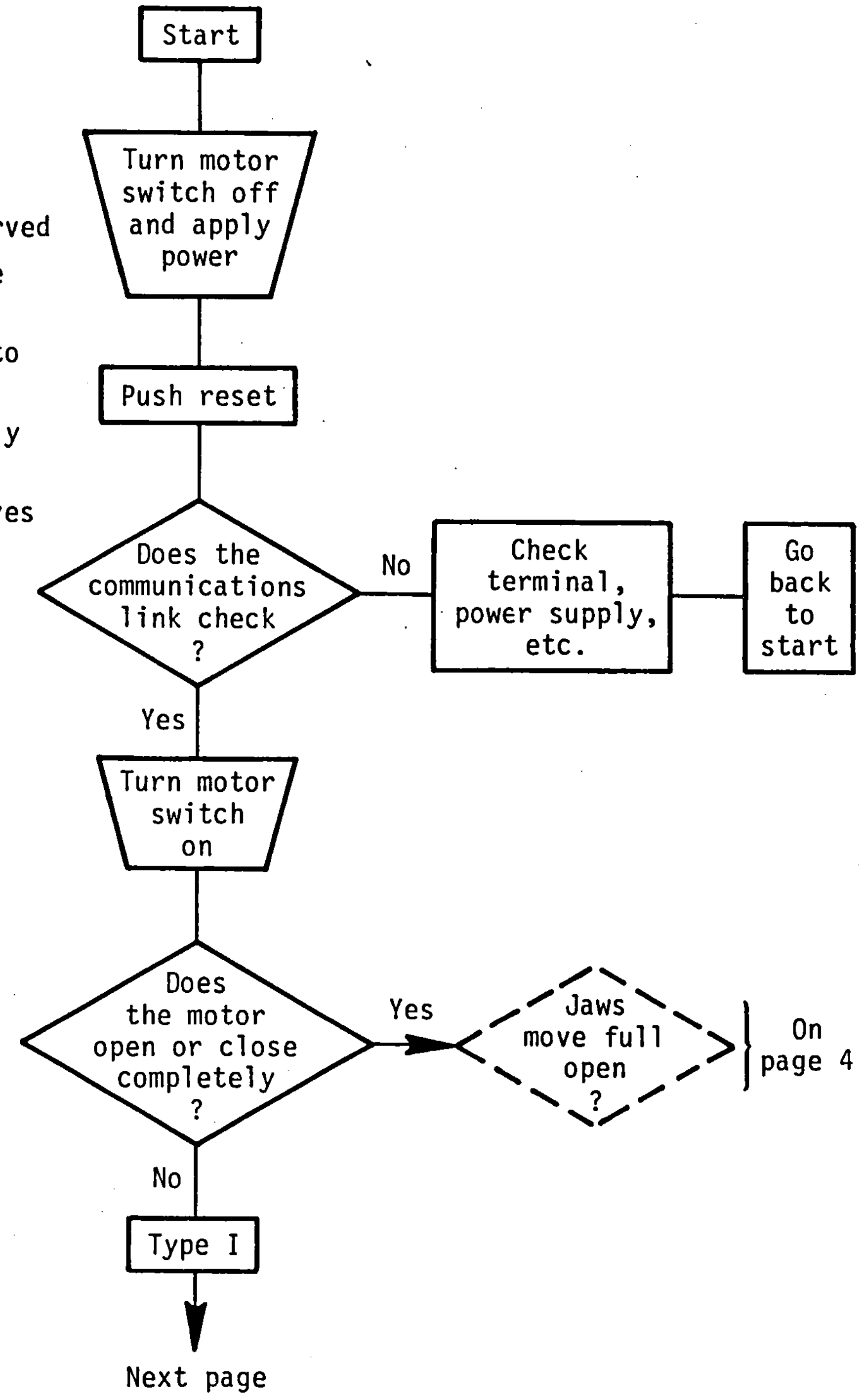
```
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
                 ´NEGATIVE
          )
          (LSSEQ (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
                 -11258
    ))    )
    (0
     (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
                 ´YES
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´TACHOMETER) ´((# VALUE)))
                 ´OK
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER) ´((# VALUE)))
                 ´OK
          )
          (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
                 ´POSITIVE
          )
          (GTREQ (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
                 -100
    ))    )
    ((SELECT1 EFFECTOR ´(EQUAL (# NAME) ´ACTUAL_POSITION) ´((# VALUE)))
     (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER) ´((# VALUE)))
                ´BAD
         )
         (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE) ´((# VALUE)))
                ´NO
         )
         (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE)
                                       ´((# VALUE))
                     )
                     ´YES
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´TACHOMETER) ´((# VALUE)))
                     ´BAD
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER)
                                       ´((# VALUE))
                     )
                     ´OK
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE)
                                       ´((# VALUE))
                     )
                     ´NEGATIVE
              )
              (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
                     -11357
         )    )
         (AND (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´JAWS_TO_MOVE)
                                       ´((# VALUE))
```

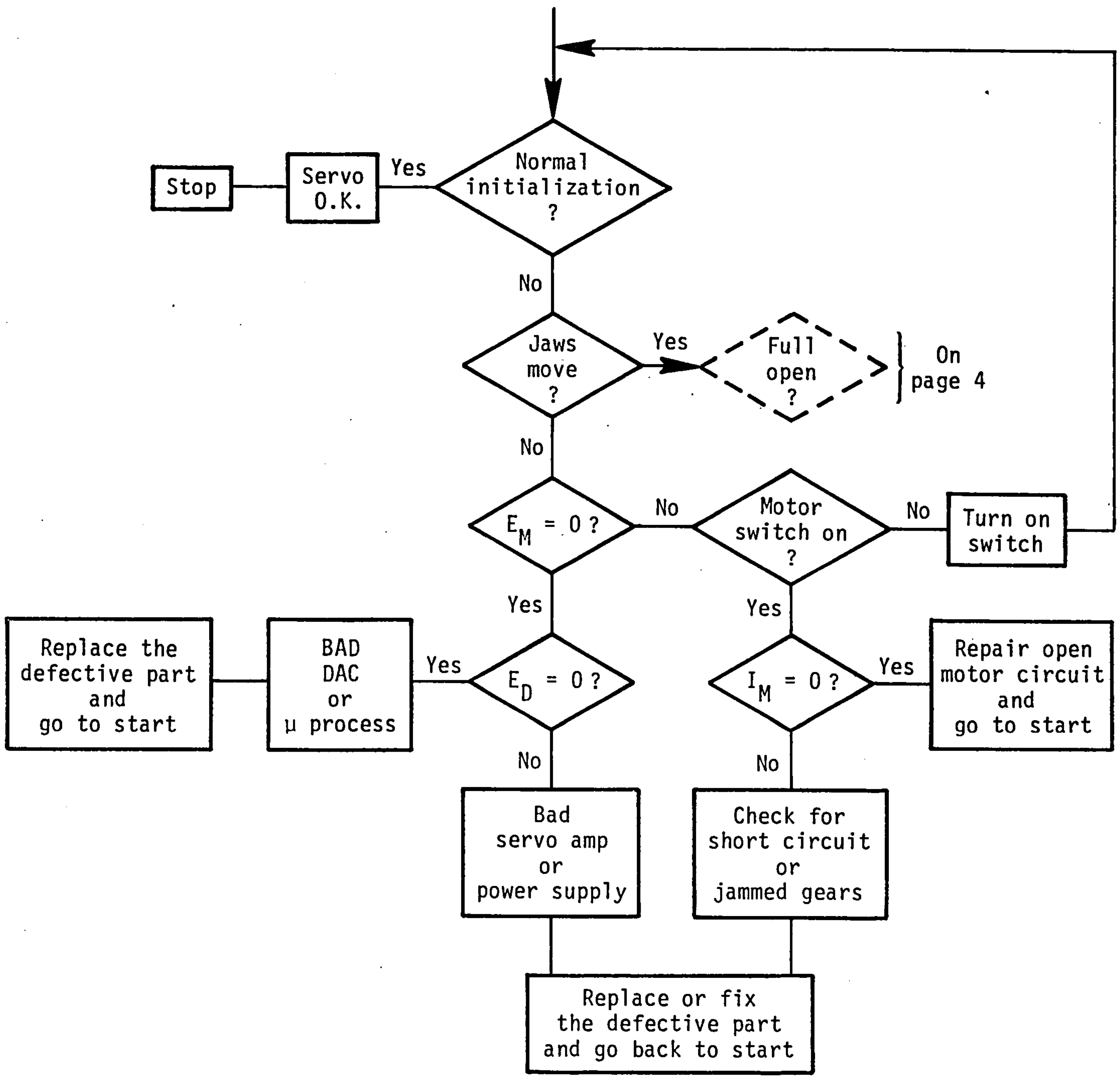
```
                    )
                    ´YES
             )
             (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´TACHOMETER) ´((# VALUE)))
                    ´BAD
             )
             (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHAFT_ENCODER)
                                      ´((# VALUE))
                    )
                    ´OK
             )
             (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE)
                                      ´((# VALUE))
                    )
                    ´POSITIVE
             )
             (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´DISTANCE) ´((# VALUE)))
                    0
  ))    )    )
))
(SETQ MOVE_JAWS ´(
  (V_EXPR B_EXPR)
  (YES
   (AND (NOTEQ (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
               ´OFF
        )
        (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_SWITCH) ´((# VALUE)))
               ´ON
        )
        (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_CURRENT) ´((# VALUE)))
               ´ON
        )
        (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHORT_CIRCUIT) ´((# VALUE)))
               ´NO
        )
        (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´GEARS) ´((# VALUE)))
               ´OK
  ))    )
  (NO
   (OR (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_VOLTAGE) ´((# VALUE)))
               ´OFF
        )
        (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_SWITCH) ´((# VALUE)))
               ´OFF
        )
        (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´MOTOR_CURRENT) ´((# VALUE)))
               ´OFF
        )
        (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´SHORT_CIRCUIT) ´((# VALUE)))
               ´YES
        )
        (EQUAL (SELECT1 EFFECTOR ´(EQUAL (# NAME) ´GEARS) ´((# VALUE)))
               ´JAMMED
  ))    )
))
```

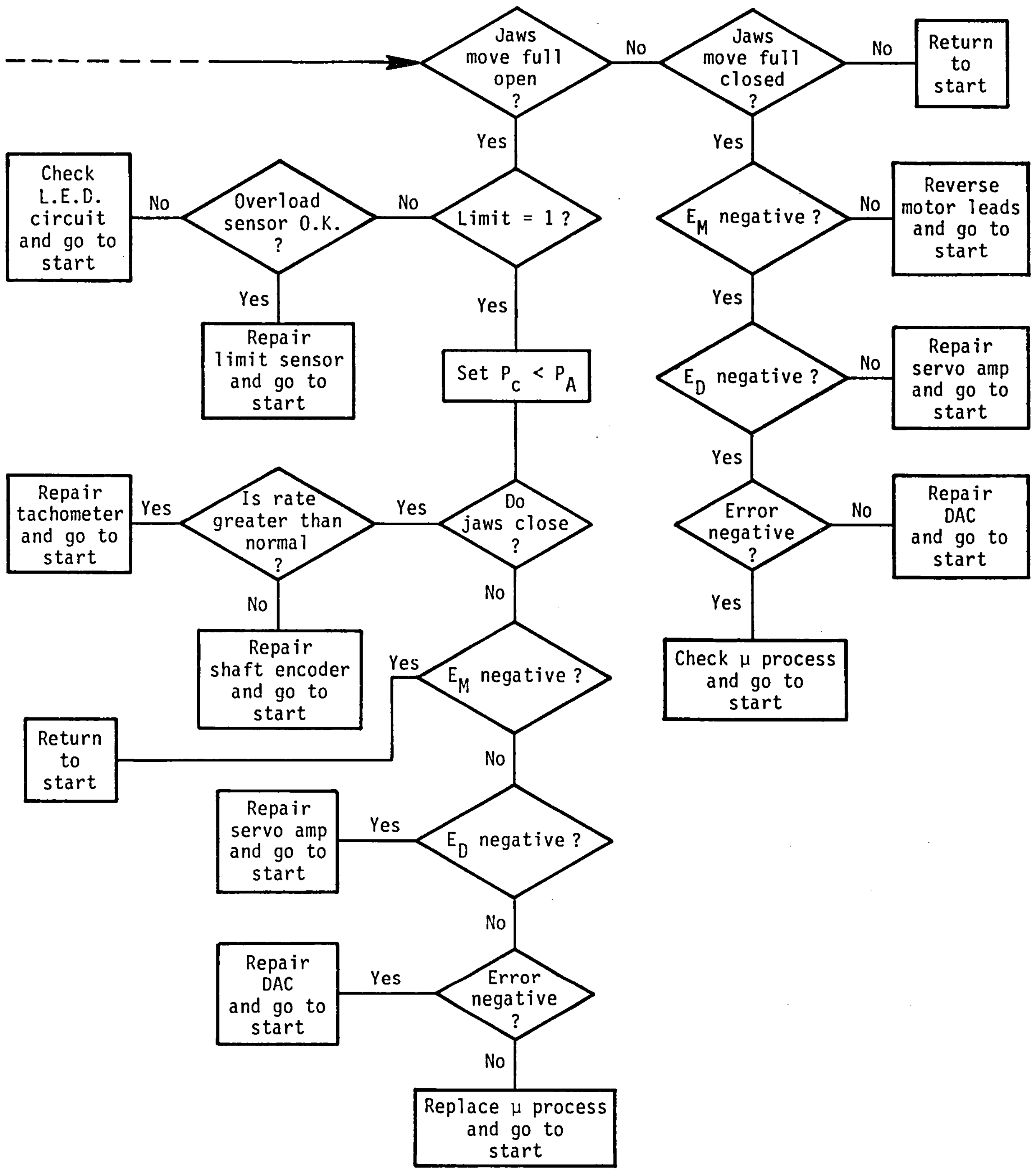
Appendix D: "Trouble-shooting" Flowchart

* Assumptions:

1. Actual jaw position can be visually observed
2. No obstruction in the jaws
3. Voltmeter connected to DAC output (E_D)
4. Move jaws mechanically to mid travel
5. A positive error drives the jaws open


Start
Turn motor switch off and apply power
Push reset
Does the communications link check ?
No
Check terminal, power supply, etc.
Go back to start
Yes
Turn motor switch on
Does the motor open or close completely ?
Yes
Jaws move full open ?
On page 4
No
Type I
Next page


Stop
Servo O.K.
Yes
Normal initialization ?
No
Jaws move ?
Yes
Full open ?
On page 4
No
$E_M = 0$?
No
Motor switch on ?
No
Turn on switch
Yes
Yes
Replace the defective part and go to start
BAD DAC or μ process
Yes
$E_D = 0$?
$I_M = 0$?
Yes
Repair open motor circuit and go to start
No
No
Bad servo amp or power supply
Check for short circuit or jammed gears
Replace or fix the defective part and go back to start


Jaws move full open ?
No
Jaws move full closed ?
No
Return to start
Yes
Yes
Check L.E.D. circuit and go to start
No
Overload sensor O.K. ?
No
Limit = 1 ?
E_M negative ?
No
Reverse motor leads and go to start
Yes
Repair limit sensor and go to start
Yes
Yes
Set $P_C < P_A$
E_D negative ?
No
Repair servo amp and go to start
Yes
Repair tachometer and go to start
Yes
Is rate greater than normal ?
Yes
Do jaws close ?
Error negative ?
No
Repair DAC and go to start
No
No
Yes
Repair shaft encoder and go to start
Yes
E_M negative ?
Check μ process and go to start
Return to start
No
Repair servo amp and go to start
Yes
E_D negative ?
No
Repair DAC and go to start
Yes
Error negative ?
No
Replace μ process and go to start

Appendix E: Normal Simulation

Following is the complete output for the normal simulation discussed in section 2.3.5 and presented in figure 2.

```
? (update 'effector '(equal (# name) 'commanded_position) '(value -433))
(--END UPDATE)
? (gotoit 8)
(CYCLE NUMBER - 1)
(--END UPDATE)
(COMMANDED_POSITION LOCATION -433)
(ERROR_SIGNAL DIRECTION NEGATIVE)
(ERROR_SIGNAL MAGNITUDE 433)
(DAC_VOLTAGE DIRECTION NEGATIVE)
(MOTOR_VOLTAGE DIRECTION NEGATIVE)
(MOTOR_CURRENT CONDITION ON)
(JAWS_TO_MOVE CONDITION YES)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 2)
(--END UPDATE)
(ACTUAL_POSITION LOCATION -100)
(DISTANCE AMOUNT -100)
(ERROR_SIGNAL MAGNITUDE 333)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 3)
(--END UPDATE)
(ACTUAL_POSITION LOCATION -200)
(DISTANCE AMOUNT -200)
(ERROR_SIGNAL MAGNITUDE 233)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 4)
(--END UPDATE)
(ACTUAL_POSITION LOCATION -300)
(DISTANCE AMOUNT -300)
(ERROR_SIGNAL MAGNITUDE 133)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 5)
(--END UPDATE)
(ACTUAL_POSITION LOCATION -400)
(DISTANCE AMOUNT -400)
(ERROR_SIGNAL DIRECTION OFF)
(ERROR_SIGNAL MAGNITUDE 0)
(DAC_VOLTAGE DIRECTION OFF)
(SERVO_AMP_VOLTAGE OFF)
(MOTOR_VOLTAGE DIRECTION OFF)
(MOTOR_CURRENT CONDITION OFF)
```

```
(JAWS_TO_MOVE CONDITION NO)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 6)
(--END UPDATE)
(-- AT EQUILIBRIUM --)
```

Appendix F: Abnormal Simulation

Following is the complete output for the abnormal simulation discussed in section 2.3.5 and presented in figure 3.

```
? (update ´effector ´(equal (# name) ´commanded_position) ´(value -120))
(--END UPDATE)
? (update ´effector ´(equal (# name) ´tachometer) ´(value ´bad))
(--END UPDATE)
? (gotoit 6)
(CYCLE NUMBER - 1)
(--END UPDATE)
(COMMANDED_POSITION LOCATION -120)
(ACTUAL_POSITION LOCATION -400)
(DISTANCE AMOUNT -400)
(ERROR_SIGNAL DIRECTION POSITIVE)
(ERROR_SIGNAL MAGNITUDE 280)
(DAC_VOLTAGE DIRECTION POSITIVE)
(SERVO_AMP_VOLTAGE DIRECTION POSITIVE)
(MOTOR_VOLTAGE DIRECTION POSITIVE)
(MOTOR_CURRENT CONDITION ON)
(JAWS_TO_MOVE CONDITION YES)
(TACHOMETER CONDITION BAD)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 2)
(--END UPDATE)
(ACTUAL_POSITION LOCATION -380)
(DISTANCE AMOUNT -300)
(ERROR_SIGNAL MAGNITUDE 260)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 3)
(--END UPDATE)
(ACTUAL_POSITION LOCATION -360)
(DISTANCE AMOUNT -200)
(ERROR_SIGNAL MAGNITUDE 240)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 4)
(--END UPDATE)
(ACTUAL_POSITION LOCATION -340)
(DISTANCE AMOUNT -100)
(ERROR_SIGNAL MAGNITUDE 220)
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 5)
(--END UPDATE)
(ACTUAL_POSITION LOCATION -320)
(DISTANCE AMOUNT 0)
(ERROR_SIGNAL MAGNITUDE 200)
```

```
(--END PROJECT)
(--END UPDATE)
(CYCLE NUMBER - 6)
(--END UPDATE)
(-- AT EQUILIBRIUM --)
```

Appendix G: Method I Trace for Bad Tachometer Example

Following is a trace of the recursive calls to examine_origin for method I generated by a call to examine_origin(distance). The end effector system is in the abnormal state resulting from the simulation with the bad tachometer discussed in section 2.3.5 and presented in figure 3 and Appendix F.

```
examine_origin(distance)
. examined = (distance)
. if default? - n
. else relation
.   eval_b_expr(1)
.     is(jaws_to_move = y) - t
.     is(motor_voltage = negative) - f
.   eval_b_expr(2)
.     is(jaws_to_move = y) - t
.     is(motor_voltage = positive) - t
.     is(distance <= -100) - f
.   eval_b_expr(3)
.     is(jaws_to_move = y) - t
.     is(motor_voltage = negative) - f
.   eval_b_expr(4)
.     is(jaws_to_move = y) - t
.     is(motor_voltage = positive) - t
.     is(distance >= -100) - t
.   examine_origin(jaws_to_move)
.   . examined = examined + jaws_to_move
.   . if default? - n
.   . else relation
.   .   eval_b_expr(1)
.   .     is(motor_voltage <> off) - t
.   .     is(switch = on) - t
.   .     is(motor_current = on) - t
.   .     is(short_circuit = n) - t
.   .     is(gears = ok) - t
.   .   examine_origin(motor_voltage)
.   .   . examined = examined + motor_voltage
.   .   . if default? - n
.   .   . else relation
.   .   .   eval_b_expr(1)
.   .   .     is(power_supply = on) - t
.   .   .     is(motor_leads = ok) - t
.   .   .   examine_origin(power_supply)
.   .   .   . examined = examined + power_supply
.   .   .   . if default? - y
.   .   .   ..return 'on' - power_supply
.   .   .   inrange(power_supply,on) - t
.   .   .   examine_origin(motor_leads)
.   .   .   . examined = examined + motor_leads
.   .   .   . if default? - y
.    .    .    ..return ´ok´ - motor_leads
.    .    .    inrange(motor_leads,ok) - t
.    .    .    examine_origin(servo_amp_voltage)
.    .    .    . examined = examined + servo_amp_voltage
.    .    .    . if default? - n
.    .    .    . else relation
.    .    .    .   eval_b_expr(1)
.    .    .    .     is(servo_amp = ok) - t
.    .    .    .   examine_origin(servo_amp)
.    .    .    .   . examine = examined + servo_amp
.    .    .    .   . if default? - y
.    .    .    .   ..return ´ok´ - servo_amp
.    .    .    .   inrange(servo_amp,ok)
.    .    .    .   examine_origin(dac_volt)
.    .    .    .   . examined = examined + dac_volt
.    .    .    .   . if default? - n
.    .    .    .   . else relation
.    .    .    .   .   eval_b_expr(1)
.    .    .    .   .     is(dac = ok) - t
.    .    .    .   .   examine_origin(dac)
.    .    .    .   .   . examined = examined + dac
.    .    .    .   .   . if default? - y
.    .    .    .   .   ..return ´ok´
.    .    .    .   .   inrange(dac,ok) - t
.    .    .    .   .   examine_origin(err_sig_dir)
.    .    .    .   .   . examined = examined + err_sig_dir
.    .    .    .   .   . if default? - n
.    .    .    .   .   . else relation
.    .    .    .   .   .   eval_b_expr(1)
.    .    .    .   .   .     is(microprocessor = bad) - f
.    .    .    .   .   .       or
.    .    .    .   .   .     is(microprocessor = ok) - t
.    .    .    .   .   .     is(abs(actual-commanded) < 100) - f
.    .    .    .   .   .   eval_b_expr(2)
.    .    .    .   .   .     is(microprocessor = constant_positive) - f
.    .    .    .   .   .       or
.    .    .    .   .   .     is(microprocessor = ok) - t
.    .    .    .   .   .     is(abs(actual-commanded) >= 100) - t
.    .    .    .   .   .   examine_origin(microprocessor)
.    .    .    .   .   .   . examined = examined + microprocessor
.    .    .    .   .   .   . if default? - y
.    .    .    .   .   .   ..return ´ok´ - microprocessor
.    .    .    .   .   .   inrange(microprocessor,ok) - t
.    .    .    .   .   .   examine_origin(commanded_position)
.    .    .    .   .   .   . examined = examined + commanded_position
.    .    .    .   .   .   . if default? - y
.    .    .    .   .   .   ..return ´-120´ - commanded_position
.    .    .    .   .   .   inrange(commanded_position) - t
.    .    .    .   .   .   examine_origin(actual_position)
.    .    .    .   .   .   . examined = examined + actual_position
.    .    .    .   .   .   . if default? - n
.    .    .    .   .   .   . else relation
.    .    .    .   .   .   .   eval_b_expr
. . . . . . .   is(jaws_to_move = yes) - t
. . . . . . .   is(tachometer = ok) - f
. . . . . . .     or
. . . . . . .   is(shaft_encoder = constant_negative) - f
. . . . . . . eval_b_expr(2)
. . . . . . .   is(jaws_to_move = yes) - t
. . . . . . .   is(tachometer = ok) - f
. . . . . . .     or
. . . . . . .   is(shaft_encoder = constant_positive) - f
. . . . . . . eval_b_expr(3)
. . . . . . .   is(jaws_to_move = yes) - t
. . . . . . .   is(tachometer = bad) - t
. . . . . . .   is(shaft_encoder = ok) - t
. . . . . . .   is(motor_voltage = negative) - f
. . . . . . . eval_b_expr(4)
. . . . . . .   is(jaws_to_move = yes) - t
. . . . . . .   is(tachometer = bad) - t
. . . . . . .   is(shaft_encoder = ok) - t
. . . . . . .   is(motor_voltage = positive) - t
. . . . . . .   is(distance <> 0) - f
. . . . . . . eval_b_expr(5)
. . . . . . .   is(jaws_to_move = yes) - t
. . . . . . .   is(tachometer = ok) - f
. . . . . . . eval_b_expr(6)
. . . . . . .   is(jaws_to_move = yes) - t
. . . . . . .   is(tachometer = ok) - f
. . . . . . . eval_b_expr(7)
. . . . . . .   is(shaft_encoder = bad) - f
. . . . . . .     or
. . . . . . .   is(jaws_to_move = no) - f
. . . . . . .     or
. . . . . . .   is(jaws_to_move = yes) - t
. . . . . . .   is(tachometer = bad) - t
. . . . . . .   is(shaft_encoder = bad) - t
. . . . . . .   is(motor_voltage = negative) - f
. . . . . . .     or
. . . . . . .   is(jaws_to_move = yes) - t
. . . . . . .   is(tachometer = bad) - t
. . . . . . .   is(shaft_encoder = ok) - t
. . . . . . .   is(motor_voltage = positive) - t
. . . . . . .   is(distance = 0) - t
. . . . . . . jaws_to_move already examined
. . . . . . . examine_origin(tachometer)
. . . . . . . . examined = examined + tachometer
. . . . . . . . if default? - y
. . . . . . . ..return 'bad' - tachometer
. . . . . . . inrange(tachometer,bad) - f
. . . . . . . poss_malfunctions = (tachometer = bad)
. . . . . . . examine_origin(shaft_encoder)
. . . . . . . . examined = examined + shaft_encoder
. . . . . . . . if default? - y
. . . . . . . ..return 'ok' - shaft_encoder
. . . . . . . inrange(shaft_encoder,ok) - t
.   .   .   .   .   .   .   motor_voltage already examined
.   .   .   .   .   .   .   distance already examined
.   .   .   .   .   .   ..return '-320' - actual_position
.   .   .   .   .   . inrange(actual_position,-320) - t
.   .   .   .   .   ..return 'positive' - err_sig_dir
.   .   .   .   . inrange(err_sig_dir,positive) - t
.   .   .   .   ..return 'positive' - dac_volt
.   .   .   . inrange(dac_volt,positive) - t
.   .   .   ..return 'positive' - servo_amp_volt
.   .   . inrange(servo_amp_volt,positive) - t
.   .   ..return 'positive' - motor_voltage
.   .   inrange(motor_voltage,positive) - t
.   .   examine_origin(motor_switch)
.   .   . examined = examined + motor_switch
.   .   . if default? - y
.   .   ..return 'on' - motor_switch
.   .   inrange(motor_switch,on) - t
.   .   examine_origin(motor_current)
.   .   . examined = examined + motor_current
.   .   . if default? - n
.   .   . else relation
.   .   .   eval_b_expr(1)
.   .   .     is(motor_voltage <> off) - t
.   .   .     is(motor_circuit = closed) - t
.   .   .   motor_voltage already examined
.   .   .   examine_origin(motor_circuit)
.   .   .   . examined = examined + motor_circuit
.   .   .   . if default? - y
.   .   .   ..return 'closed'
.   .   .   inrange(motor_circuit,closed) - t
.   .   ....return 'on' - motor_current
.   .   inrange(motor_current,on) - t
.   .   examine_origin(short_circuit)
.   .   . examined = examined + short_circuit
.   .   . if default? - y
.   .   ..return 'no' - short_circuit
.   .   inrange(short_circuit,no) - t
.   .   examine_origin(gears)
.   .   . examined = examined + gears
.   .   . if default? - y
.   .   ..return 'ok' - gears
.   .   inrange(gears,ok) - t
.   ....return 'yes' - jaws_to_move
.   inrange(jaws_to_move,yes) - t
.   motor_voltage already examined
.   distance already examined
.   distance already examined
....return '0' - distance
```

Appendix H: Method II Trace for Bad Tachometer Example

Following is a trace of the recursive calls to examine_origin for method II generated by a call to examine_origin(distance). The end effector system is in the abnormal state resulting from the simulation with the bad tachometer discussed in section 2.3.5 and presented in figure 3 and Appendix F.

```
examine_origin(distance)
. examined = (distance)
. if default? -no
. else relation
.   eval_b_expr(1)
.     is(jaws_to_move = yes) - t
.       examine_origin(jaws_to_move)
.       . examined = examined + jaws_to_move
.       . if default? - n
.       . else relation
.       .   eval_b_expr(1)
.       .     is(motor_voltage <> off) - t
.       .       examine_origin(motor_voltage)
.       .       . examined = examined + motor_voltage
.       .       . if default? - n
.       .       . else relation
.       .       .   eval_b_expr(1)
.       .       .     is(power_supply = on) - t
.       .       .       examine_origin(power_supply)
.       .       .       . examined = examined + power_supply
.       .       .       . if default? - y
.       .       .       ..return 'on' - power_supply
.       .       .       inrange(power_supply,on) - t
.       .       .     is(motor_leads = ok) - t
.       .       .       examine_origin(motor_leads)
.       .       .       . examined = examined + motor_leads
.       .       .       . if default? - y
.       .       .       ..return 'ok' - motor_leads
.       .       .       inrange(motor_leads,ok) - t
.       .       .   (* b_expr 1 true - motor_voltage *)
.       .       .   examine_origin(servo_amp_voltage)
.       .       .   . examined = examined + servo_amp_voltage
.       .       .   . if default? - n
.       .       .   . else relation
.       .       .   .   eval_b_expr(1)
.       .       .   .     is(servo_amp = ok) - t
.       .       .   .       examine_origin(servo_amp)
.       .       .   .       . examined = examined + servo_amp
.       .       .   .       . if default? - y
.       .       .   .       ..return 'ok' - servo_amp
.       .       .   .       inrange(servo_amp,ok) - t
.       .       .   .   (* b_expr 1 true - servo_amp_voltage *)
.       .       .   .   examine_origin(dac_voltage)
.   .   . . . examined = examined + dac_voltage
.   .   . . . if default? - n
.   .   . . . else relation
.   .   . . .   eval_b_expr(1)
.   .   . . .    is(dac = ok) - t
.   .   . . .     examine_origin(dac)
.   .   . . .     . examined = examined + dac
.   .   . . .     . if default? - y
.   .   . . .     ..return 'ok' - dac
.   .   . . .     inrange(dac,ok) - t
.   .   . . .   (* b_expr 1 true - dac_voltage *)
.   .   . . .   examine_origin(err_sig_dir)
.   .   . . .   . examined = examined + err_sig_dir
.   .   . . .   . if default? - n
.   .   . . .   . else relation
.   .   . . .  .  eval_b_expr(1)
.   .   . . .  .   is(microprocessor = bad) - f
.   .   . . .  .    examine_origin(microprocessor)
.   .   . . .  .    . examined = examined +
.   .   . . .  .    .                   microprocessor
.   .   . . .  .    . if default? - y
.   .   . . .  .    ..return 'ok' - microprocessor
.   .   . . .  .    inrange(microprocessor,ok) - t
.   .   . . .  .   or
.   .   . . .  .   is(abs(actual_position-
.   .   . . .  .      commanded_position) < 100) - f
.   .   . . .  .    examine_origin(actual_position)
.   .   . . .  .    . examined = examined +
.   .   . . .  .    .                  actual_position
.   .   . . .  .    . if default? - n
.   .   . . .  .    . else relation
.   .   . . .  .    .  eval_b_expr(1)
.   .   . . .  .    .   is(jaws_to_move = yes) - t
.   .   . . .  .    .    jaws_to_move already
.   .   . . .  .    .                       examined
.   .   . . .  .    .   is(tachometer = ok) - f
.   .   . . .  .    .    examine_origin(tachometer)
.   .   . . .  .    .    . examined = examined +
.   .   . . .  .    .    .                  tachometer
.   .   . . .  .    .    . if default? - y
.   .   . . .  .    .    ..return 'bad' - tachometer
.   .   . . .  .    .    inrange(tachometer,bad) - f
.   .   . . .  .    .    poss_malfunctions =
.   .   . . .  .    .            (tachometer = bad)
.   .   . . .  .    .  or
.   .   . . .  .    .   is(shaft_encoder =
.   .   . . .  .    .       constant_negative) - f
.   .   . . .  .    .    examine_origin(shaft_
.   .   . . .  .    .    .                 encoder)
.   .   . . .  .    .    . examined = examined +
.   .   . . .  .    .    .               shaft_encoder
.   .   . . .  .    .    . if default? - y
.   .   . . .  .    .    ..return 'ok' - shaft_
.       .       .   .   .   .       .                         encoder
.       .       .   .   .   .       .         inrange(shaft_encoder,
.       .       .   .   .   .       .                          ok) - t
.       .       .   .   .   .       .     eval_b_expr(2)
.       .       .   .   .   .       .       is(jaws_to_move = yes) - t
.       .       .   .   .   .       .         jaws_to_move already
.       .       .   .   .   .       .                         examined
.       .       .   .   .   .       .       is(tachometer = ok) - f
.       .       .   .   .   .       .         tachometer already examined
.       .       .   .   .   .       .      or
.       .       .   .   .   .       .       is(shaft_encoder =
.       .       .   .   .   .       .              constant_negative) - f
.       .       .   .   .   .       .     eval_b_expr(3)
.       .       .   .   .   .       .       is(jaws_to_move = yes) - t
.       .       .   .   .   .       .         jaws_to_move already
.       .       .   .   .   .       .                         examined
.       .       .   .   .   .       .       is(tachometer = bad) - t
.       .       .   .   .   .       .         tachometer already examined
.       .       .   .   .   .       .       is(shaft_encoder = ok) - t
.       .       .   .   .   .       .         shaft_encoder already
.       .       .   .   .   .       .                          examined
.       .       .   .   .   .       .       is(motor_voltage =
.       .       .   .   .   .       .                     negative) - f
.       .       .   .   .   .       .         motor_voltage already
.       .       .   .   .   .       .                          examined
.       .       .   .   .   .       .     eval_b_expr(4)
.       .       .   .   .   .       .       is(jaws_to_move = yes) - t
.       .       .   .   .   .       .         jaws_to_move already
.       .       .   .   .   .       .                         examined
.       .       .   .   .   .       .       is(tachometer = bad) - t
.       .       .   .   .   .       .         tachometer already examined
.       .       .   .   .   .       .       is(shaft_encoder = ok) - t
.       .       .   .   .   .       .         shaft_encoder already
.       .       .   .   .   .       .                          examined
.       .       .   .   .   .       .       is(motor_voltage =
.       .       .   .   .   .       .                     positive) - t
.       .       .   .   .   .       .         motor_voltage already
.       .       .   .   .   .       .                          examined
.       .       .   .   .   .       .       is(distance <> 0) - f
.       .       .   .   .   .       .         distance already examined
.       .       .   .   .   .       .     eval_b_expr(5)
.       .       .   .   .   .       .       is(jaws_to_move = yes) - t
.       .       .   .   .   .       .         jaws_to_move already
.       .       .   .   .   .       .                         examined
.       .       .   .   .   .       .       is(tachometer = ok) - f
.       .       .   .   .   .       .         tachometer already examined
.       .       .   .   .   .       .     eval_b_expr(6)
.       .       .   .   .   .       .       is(jaws_to_move = yes) - t
.       .       .   .   .   .       .         jaws_to_move already
.       .       .   .   .   .       .                         examined
.       .       .   .   .   .       .       is(tachometer = ok) - f
.       .       .   .   .   .       .         tachometer already examined
.       .       .   .   .   .       .     eval_b_expr(7)
.       .       .   .   .   .       .    is(shaft_encoder = bad) - f
.       .       .   .   .   .       .      shaft_encoder already
.       .       .   .   .   .       .                        examined
.       .       .   .   .   .       .   or
.       .       .   .   .   .       .    is(jaws_to_move = no) - f
.       .       .   .   .   .       .      jaws_to_move already
.       .       .   .   .   .       .                       examined
.       .       .   .   .   .       .   or
.       .       .   .   .   .       .    is(jaws_to_move = yes) - t
.       .       .   .   .   .       .      jaws_to_move already
.       .       .   .   .   .       .                       examined
.       .       .   .   .   .       .    is(tachometer = bad) - t
.       .       .   .   .   .       .      tachometer already examined
.       .       .   .   .   .       .    is(shaft_encoder = ok) - t
.       .       .   .   .   .       .      shaft_encoder already
.       .       .   .   .   .       .                        examined
.       .       .   .   .   .       .    is(motor_voltage =
.       .       .   .   .   .       .                   negative) - f
.       .       .   .   .   .       .      motor_voltage already
.       .       .   .   .   .       .                        examined
.       .       .   .   .   .       .   or
.       .       .   .   .   .       .    is(jaws_to_move = yes) - t
.       .       .   .   .   .       .      jaws_to_move already
.       .       .   .   .   .       .                       examined
.       .       .   .   .   .       .    is(tachometer = bad) - t
.       .       .   .   .   .       .      tachometer already examined
.       .       .   .   .   .       .    is(shaft_encoder = ok) - t
.       .       .   .   .   .       .      shaft_encoder already
.       .       .   .   .   .       .                        examined
.       .       .   .   .   .       .    is(motor_voltage =
.       .       .   .   .   .       .                   positive) - t
.       .       .   .   .   .       .      motor_voltage already
.       .       .   .   .   .       .                        examined
.       .       .   .   .   .       .    is(distance = 0) - t
.       .       .   .   .   .       .      distance already examined
.       .       .   .   .   .       ..return '-320' - actual_position
.       .       .   .   .   .       inrange(actual_position,-320) - t
.       .       .   .   .   .       examine_origin(commanded_position)
.       .       .   .   .   .       . examined = examined +
.       .       .   .   .   .       .              commanded_position
.       .       .   .   .   .       . if default? - y
.       .       .   .   .   .       ..return '-120' - commanded_position
.       .       .   .   .   .       inrange(commanded_position,-120) - t
.       .       .   .   .   .   (* b_expr 1 false - err_sig_dir *)
.       .       .   .   .   .   eval_b_expr(2)
.       .       .   .   .   .     is(microprocessor =
.       .       .   .   .   .                  constant_positive) - f
.       .       .   .   .   .       microprocessor already examined
.       .       .   .   .   .   or
.       .       .   .   .   .     is(microprocessor = ok) - t
.       .       .   .   .   .       microprocessor already examined
.       .       .   .   .   .     is(abs(commanded_position -
.       .       .   .   .   .          actual_position) >= 100) - t
.       .       .   .   .   ..return 'positive' - err_sig_dir
.       .       .   .   .   inrange(err_sig_dir,positive) - t
.       .       .   .   ..return 'positive' - dac_volt
.       .       .   .   inrange(dac_volt,positive) - t
.       .       .   ..return 'positive' - servo_amp_volt
.       .       .   inrange(servo_amp_volt,positive) - t
.       .       ..return 'positive' - motor_voltage
.       .       inrange(motor_voltage,positive) - t
.       .     is(motor_switch = on) - t
.       .       examine_origin(motor_switch)
.       .       . examined = examined + motor_switch
.       .       . if default? - y
.       .       ..return 'on' - motor_switch
.       .       inrange(motor_switch,on) - t
.       .     is(motor_current = on)
.       .       examine_origin(motor_current)
.       .       . examined = examined + motor_current
.       .       . if default? - n
.       .       . else relation
.       .       .   eval_b_expr(1)
.       .       .     is(motor_voltage <> off) - t
.       .       .       motor_voltage already examined
.       .       .     is(motor_circuit = closed)
.       .       .       examine_origin(motor_circuit)
.       .       .       . examined = examined + motor_circuit
.       .       .       . if default? - y
.       .       .       ..return 'closed' - motor_circuit
.       .       .       inrange(motor_circuit,closed) - t
.       .       .   (* b_expr 1 true - motor_current *)
.       .       ..return 'on' - motor_current
.       .       inrange(motor_current,on) - t
.       .     is(short_circuit = no)
.       .       examine_origin(short_circuit)
.       .       . examined = examined + short_circuit
.       .       . if default? - y
.       .       ..return 'no' - short_circuit
.       .       inrange(short_circuit,no) - t
.       .     is(gears = ok)
.       .       examine_origin(gears)
.       .       . examined = examined + gears
.       .       . if default? - y
.       .       ..return 'ok' - gears
.       .       inrange(gears,ok)
.       .   (* b_expr 1 true - jaws_to_move *)
.       ..return 'yes' - jaws_to_move
.       inrange(jaws_to_move,yes) - t
.     is(motor_voltage = negative) - f
.       motor_voltage already examined
.   eval_b_expr(2)
.     is(jaws_to_move = yes) - t
.       jaws_to_move already examined
.     is(motor_voltage = positive) - t
.       motor_voltage already examined
.     is(distance <= -100) - f
.       distance already examined
.   eval_b_expr(3)
.     is(jaws_to_move = yes) - t
.       jaws_to_move already examined
.     is(motor_voltage = negative - f
.       motor_voltage already examined
.   eval_b_expr(4)
.     is(jaws_to_move = yes) - t
.     jaws_to_move already examined
.       is(motor_voltage = positive) - t
.       motor_voltage already examined
.     is(distance >= -100) - t
.       distance already examined
.   (* b_expr 4 true - distance *)
.   distance already examined
..return '0' - distance
```

1. Report No. NASA TM-86288	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A Relational Approach to the Development of Expert Diagnostic Systems		5. Report Date October 1984
		6. Performing Organization Code 505-37-13-04
7. Author(s) Kathy R. Ames		8. Performing Organization Report No.
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665		10. Work Unit No.
		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Adminstration Washington, DC 20546		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

The proposition that, given a structural and/or functional description of any real or abstract system, an expert system can be built based on this description is examined. First, a model is developed for a microprocessor-controlled end effector/sensor system using a modeling approach called a Relational Knowledge-Base Machine (RKBM). Next, an explanation of how the end effector model could be used for the error diagnosis on the operational end effector is given and two versions of an error diagnosis algorithm based on the model are presented. Finally, areas of further research are described that are necessary before an expert system using this approach becomes a reality.

17. Key Words (Suggested by Author(s)) expert system, relational data base, simulation, system modeling, error diagnosis	18. Distribution Statement Unclassified - Unlimited. Subject Category 59

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 67	22. Price A04

N-305

For sale by the National Technical Information Service, Springfield, Virginia 22161