Journal of Intelligent Manufacturing (1992) **3**, 59–73

Error management for robot programming

RICHARD SMITH* and MARIA GINI

Department of Computer Science, 4-192 EE/CSci Building, University of Minnesota, 200 Union St SE, Minneapolis, MN 55455, USA

Received August 1990 and accepted March 1991

Reliability is a serious problem in computer controlled robot systems. Although robots serve successfully in relatively simple applications such as painting and spot welding, their potential in areas such as automated assembly is hampered by the complexity of programming. A program for assembling parts may be logically correct, execute correctly on a simulator, and even execute correctly on a robot most of the time, yet still fail unexpectedly in the face of real world uncertainties. Recovery from such errors is far more complicated than recovery from simple controller errors, since even expected errors can manifest themselves in unexpected ways. In this paper we present a novel approach for improving robot reliability. Instead of anticipating errors, we use knowledge-based programming techniques so that the robot can autonomously exploit knowledge about its task and environment to detect and recover from failures. We describe a system that we have designed and constructed in our robotics laboratory.

Keywords: Robot programming, error detection, error recovery, reliability

1. Introduction

We want to make robots more dependable so that they can be trusted when left unattended. This paper describes the design and development of a robot system that continues to operate satisfactorily even after it encounters a serious error. Failures in achieving a task are the result of errors, but not every error produces an immediately detectable failure. Errors can occur at many levels, at the mechanical level (a joint becomes locked), at the hardware level (a sensor does not function properly so that the robot is driven to exceed its joint limits), at the controller level, in the computer controlling the robot (either at the hardware or the software level), and in the environment. We are mostly interested in errors in the environment because they tend to be more unpredictable and difficult to characterize with mathematical models. We are interested in errors in the component parts used for the assembly, and in errors in the work cell (loaders, feeders, conveyor belts, tools). Our goal is to detect automatically and correct problems caused by collisions, jammed parts, gripper slip, misorientation, alignment errors, and missing parts.

Robot systems that can recover from errors without human intervention do not exist today because robot control programs cannot handle the vast range of possible error conditions. Error-handling routines are usually produced through trial and error with specific robots, tasks, and work cell environments. It takes uncommon skill and experience to develop a reliable robot program, and the resulting program will then only apply to the specific robot task at hand. Minor changes in the robot, task, or environment can lead to major changes in the robot's program (Lozano-Perez, 1983). It is commonly believed among robotics engineers that less than 20% of the code produced is directly responsible for describing the robot's actual task. The rest of the code checks for errors and tries to recognize and prevent disasters (Fielding *et al.*, 1987).

A difficult problem in automatic error detection and recovery is detecting that something significant has occurred. Many events are usually reported to the robot controller but not all of them are significant. The same event may be important in some circumstances and almost irrelevant in others. Deciding when something is important is the first step in the error detection process. The second step involves detecting the cause of the error and its effect

*Present address: Secure Computing Technology Corporation, Arden Hills, MN, USA.

on the robot environment. Errors might appear a long time after what caused them happened making it more difficult to detect and correct them. Some errors do not affect the execution of the task so they could be left unrecovered. To recover from an error, a system needs to identify the differences between what has happened and what is wanted. The system must also be able to plan how to correct these differences itself.

By making autonomous error recovery an attribute of the robot programming and control system, we can improve the reliability of robots and at the same time simplify the programming task. The manufacturing engineer can concentrate on describing the task at hand without having to consider all likely errors and how to correct them. The task description only needs to describe the assembly task and does not need to specify error detection or recovery procedures. This saves engineering time as well as robot down-time (Smith and Gini, 1986b).

The system described here works with a fully functional IBM 7565 industrial manipulator. The system uses the task specification to generate a more detailed program describing the sequence of manipulator operations and the sensor activities necessary to monitor the task's successful operation. When an error is detected, the system automatically analyses the error and produces a sequence of operations to recover from the error. More details can be found in Smith (1987).

2. Related work

Reliability is a serious problem in robot programming and a difficult one as well. Parts slip, fall, jam, and get misplaced; surfaces become wet or slippery, and operations fail. The range of potential problems is so vast that there is no standard engineering procedure that addresses all of them (Gini, 1990).

Many ideas used to improve the reliability of robots have been taken from the fields of software reliability and safety (Green and Bourne, 1972; Randell *et al.*, 1978 and Leveson, 1986). Unfortunately, techniques for software reliability deal with states of information, not states of the world and are primarily directed towards restoring internal data states, not physical conditions. Robots operate in the real world and errors of interest are those manifested in the real world, not within the robot's software (Harmon, 1988). A common approach to failure analysis and diagnosis is to apply techniques based on fault trees, event trees, or cause–consequence diagrams. It has been proposed to combine fault-tree-based failure analysis with rule-oriented reasoning (Williams *et al.*, 1986 and Narayanan and Viswanadham, 1987), to use fuzzy logic (Tsukanoto and Terano, 1977), or to use Petri nets (Zhou and DiCesare, 1989).

Our approach is very different from the customary techniques of robot programmers. The typical approach is to use the robot's sensors to check for errors and to perform a pre-programmed recovery procedure in response (Cox and Gehani, 1989). As noted earlier, this leads to robot programs that consist predominantly of error detection and recovery code. In large applications the robot might not even be able to fit all possible error-handling codes into its control memory. Another serious problem is that error interpretation is inseparably tied to sensor readings. Specific sensor readings do not always refer to individual and unique errors.

Since it is difficult to consider all possible errors, many of which might never happen, a method proposed in artificial intelligence is to generate, from the task level description, a program that is guaranteed to be correctly executed even in the presence of uncertainties in the environment. This requires models of robot kinematics and dynamics, and models of physical properties of objects such as friction. This approach, so far, has been applied only to fine motions for specific tasks such as insertion operations (Lozano-Perez *et al.*, 1984). Modeling uncertainties and taking into account errors in the model (Donald, 1990) helps but the real world is so complex that we do not really know how to model it. It has been shown (Fielding *et al.*, 1988) that the problem of generating a complete set of automatic error recovery routines is undecidable.

Much previous research in artificial intelligence has centered on detection and correction of errors in simulated robot systems (Wilkins, 1985). Most AI research makes a number of assumptions: knowledge about events is correct, each action produces precisely defined post-conditions, there are no uncertain data, correct predicates are generated from sensor data every time they are needed, and sufficient knowledge is provided to take into account all the possible states of the environment. These assumptions are too strict to be realistic. A few exceptions exist. The most notable is STRIPS (Fikes and Nilsson, 1971) the system used to control the mobile robot, Shakey. More recently, work has been done on monitoring the execution of programs with real robots for manufacturing applications (Lee *et al.*, 1983 and Nof *et al.*, 1987). There is a growing interest in modeling sensors (Henderson and Shilcrat, 1984) and planning for their use (Doyle *et al.*, 1986) that will benefit work on error detection and recovery.

Our early work (Gini and Gini, 1983) was inspired by early research at Jet Propulsion Laboratory (JPL). Srinivas (1977) developed a formlization of techniques to explain robot failures and to generate expectations of the effects of failures. The first of these techniques, failure reason analysis, used knowledge about why robot actions fail to generate a fault tree identifying possible reasons for a particular failure. The second technique, multiple outcome analysis, would generate a set of possible outcomes from a given failure along with an indication of sensor information that would disambiguate among the possibilities. Unfortu-

nately, Srinivas' system was only partially implemented on the JPL Robot.

The group at the National Bureau of Standards (Simpson *et al.*, 1983) has developed a hierarchical control architecture for a small-batch metal machining shop. The architecture has recently been adapted to control the NASA robot for the space station (Lumia *et al.*, 1989). The system is divided into three hierarchies; task decomposition, world model, and sensory processing. At each level, goals are decomposed into simpler goals for the next lower level. The sensory system updates the world model as rapidly as possible to keep the model consistent with the physical world.

The group at the University of Toulouse (Lopez-Mellado and Alami, 1986) has designed and implemented a system, NNS, to control a manufacturing cell that includes planning and some error recovery. They update the work cell model after each failure, which is computationally expensive except for trivial cells. The system concentrates on using assembly knowledge during the assembly task itself. Tasks are specified as a sequence of changes in the state of the work cell. The structure of the system makes the intentions of the robot's actions clear during task execution, making it possible to detect automatically and recover from errors. NNS does not address the problem of task specification; tasks are apparently coded directly in terms of work cell state descriptions.

The group of Lee and Hardy (Lee *et al.*, 1983 and Hardy *et al.*, 1989) at the University College of Wales has studied the problem of error recovery in conjunction with industrial robot tasks. They are working on an experimental system, called AFFIRM, for representing knowledge about the robot's task, work cell, and sensors. Tasks are encoded using the frame-based knowledge structure shared by the rest of the system. It appears that the work on AFFIRM concentrates on error diagnosis more than on error recovery.

Trevelyan and his group (Trevelyan *et al.*, 1988) has done similar work in sensor monitoring involving a radically different domain. In their problem domain, sheep shearing, they have succeeded in recognizing a number of specific failures through specific sensor readings. Each failure triggers a specific recovery procedure. The system does not attempt to represent knowledge about errors and recoveries in a flexible fashion or to support inferencing. While the technique of directly coupling sensor readings to error handling greatly simplifies the system, it also restricts the system to handling errors that can be explicitly recognized by sensors.

Lyons (Lyons *et al.*, 1989) proposes a representation for robot task plans that allows robots to react to suit the current environment. He has developed a formal model of distributed computations that represents reactive plans as networks of distributed processes and that allows to represent the environment in which the robot acts. This representation can be used to monitor the execution of plans.

3. Architecture for robot error recovery

The autonomous error recovery system consists of three components: the pre-processor, which accepts robot task specifications, the AP executive, which operates the robot and monitors its performance of the task, and the recoverer, which generates error recovery procedures in response to errors detected during execution. Figure 1 illustrates the architecture and the flow of information in the system. In the figure ovals represent data and rectangular boxes represent processes. Subsequent sections of the paper describe each component.

The architecture presented here explicitly divides the system into two parts: one that must operate in real time and one that does not. The activities associated with operating the robot and verifying its performance are assigned to a dedicated processor so that they may operate at the highest possible speed. Activities associated with preparing tasks for execution by the robot or with error analysis and recovery are handled by a separate processor that does not have serious time constraints.

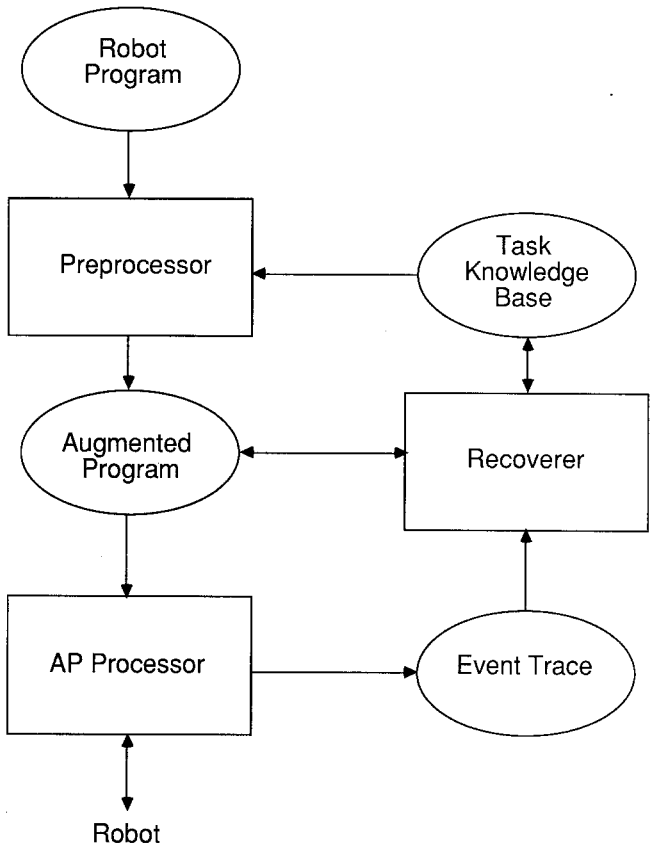

Fig. 1. Architecture of the system.

An important feature of the system is that it is designed to incorporate existing commercial robot controllers. It does not depend on custom-designed robot control software, instead it operates the robot through the robot's own controller using its own control protocol. The error recovery system simply needs to be adapted to a specific controller so that it can send motion commands to the robot's controller and use the robot's sensors.

The system cannot handle all possible errors. From the beginning, our research has concentrated on the problems posed by physical errors and uncertainties in the robot's task and not on verification of the program's correctness. The robot program given to the system is taken as the specification of the desired task. Errors that could be found by a robot simulation system must be eliminated before execution.

The present configuration of the error recovery testbed manages an IBM 7565 manufacturing manipulator. The IBM 7565 is a hydraulic gantry robot with six degrees of freedom and a closed loop controller. The robot's gripper contains strain gauges on each jaw that measure strain along three axes. Motion control and sensor filtering software resides on an IBM Series/1 minicomputer and allows manipulator and sensor programming in the AML language (IBM, 1982 and Taylor *et al.*, 1982). The AP executive is implemented in C on an MC68000-based personal computer system, an Apple Macintosh. The pre-processor and the recoverer are written in Franz Lisp on a Unix time-sharing system that communicates with the Macintosh.

4. The pre-processor

To operate the system, the user provides the pre-processor with a task specification in the system's robot programming language. The pre-processor converts this program into a specially designed intermediate form called the augmented program (AP). This expanded form is then used by the AP executive to monitor and control the robot's operation in real time.

Task specifications are written in a simple manipulator level programming language called woktalk. The name is taken from Project Woksape, the organization that provided the IBM robot used in the testbed. The woktalk language is equivalent in expressive power to typical robot languages even though it lacks the features of some languages. The woktalk primitives are adequate for describing manipulator and object motions in pick and place tasks.

Figure 2 presents a simple robot task written in woktalk. The robot moves about an object, picks it up, lifts it, puts it down, and releases it. Woktalk programs are written as Lisp forms with individual statements comprised of lists of atoms. The first statement, name, defines the relevant

```
((name (aboveslot (8.44 -15.92 7 -47 0 0))
       (slot (8.44 -15.92 .1 -47 0 0)))
 (move aboveslot)
 (open 1.5)
 (grab cp1 slot)
 (carry cp1 aboveslot)
 (release cp1 1.5 slot))
```

Fig. 2. A simple program in woktalk.

positions. The first three coordinates are cartesian x, y, and z in the work cell; the last three coordinates specify pitch, roll, and yaw angles in degrees. The move statement moves the gripper to the specified location aboveslot. The open statement commands the gripper to open 3.75 cm. The next three instructions describe manipulator actions that involve objects. The grab statement tells the robot to move to the location, slot and grasp the object, cp1. The carry statement tells the manipulator to move to the location aboveslot while carrying the object cp1. The release statement tells the manipulator to move to the location slot and open the gripper to 3.75 cm causing it to cease carrying an object. These statements name the particular object being affected so that the system can keep track of actions performed on parts in the work cell.

4.1. *The task knowledge base*

The task knowledge base (TKB) describes objects and locations that are relevant to the robot's task. The TKB serves as an initial model of the state of the robot's work cell when it begins its task. The TKB identifies all objects in the work cell that could possibly be used in the robot's task. Each object is identified by location and object type. Important locations in the work cell, such as pallet slots and fixture positions, are also identified in the model. If an error occurs, the recoverer updates the work cell model to reflect its probable state using information in the event trace. The TKB is constructed by the robot programmer using appropriate declaration statements. The TKB used for the sample program and other similar programs is shown in Table 1.

Our experiments used only one type of part, so there is only one object class. The object is of type coupler and its grasp width is declared to be 2.5 cm. The grasp width specifies a nominal value for the gripper's width when it is holding that type of object. There are three couplers: part cp1 stored in location slot1, part cp2 stored in slot2, and part cp3 stored in slot3. The coupler in slot3 is declared as a spare so it may be used to replace one of the other couplers if an error occurs.

Table 1. The task knowledge base for the sample program and for the experiments.

Object classes			*Parts*				*Location of parts*						
Identifier	*Descriptive name*	*Grasp width*	*Name*	*Class*	*Location*	*Usage*	*Name*	*Location*					*Usage*
coupler	'coupler'	1.0	cp1	coupler	slot1	task	slot1	8.44	−15.92	0.1	−47	0 0	coupler
			cp2	coupler	slot2	task	slot2	6.08	−15.94	0.1	−47	0 0	coupler
			cp3	coupler	slot3	spare	slot3	3.73	−15.96	0.1	−47	0 0	coupler
							fixture	−7.77	−14.41	4.43	−47	0 0	holder
							park	0	−15	4	−47	0 0	park
							discard	6.44	−5	3.5	−47	0 0	discard

The three slots that initially contain couplers are all named and their locations specified. The usage declares how a location can be used. The park location identifies the manipulator's park position in the work cell. The discard location identifies a location above a trash bin into which the robot may drop unwanted parts during error recovery.

The TKB could be generated automatically from a CAD data base, but we haven't done so in our current implementation. This would require enrichment of the semantics of typical CAD data files to indicate the usage of objects and locations. Incorporation of such CAD data would support the geometric modeling of objects, which would allow more sophisticated analysis and recovery techniques.

4.2. *Translating a woktalk program into an AP*

The augmented program (AP) is a representation of the robot's program that provides the interface between the pre-processor and the AP executive. In addition to the command, sensory, and procedural information necessary to control the robot, the AP contains information about how the robot's actions affect the objects in the work cell. The AP represents the sequence of actions as a finite automaton. The pre-processor generates the AP by translating the original woktalk program. The sequencing of instructions in the woktalk program is replaced by transitions in the AP. Crucial sensor readings preceding some action in the woktalk program will correspond to the pre-conditions of the corresponding state transition in the AP. The pre-conditions on transition leading out of a given state will correspond to the set of sensor readings to be monitored by the sensor handler.

The AP incorporates two kinds of information besides motion commands: sensor information and information about objects in the work cell. The sensor information in the AP is used to guide the real time system in its use of sensors. The AP specifies which sensors need to be monitored during each step of the robot's task and often identifies specific sensor values that are meaningful. All unspecified sensor readings can be ignored by the system, thus saving computation time and expense. The information about objects is not used to control the robot, but it is saved in the event trace and used only if an error occurs.

The automaton representation provides a natural way to map events to actions. Significant sensor readings are treated as tokens that may cause state transitions and actions related thereto. The AP structure is designed to be interpreted efficiently in real time. Another requirement is that the AP must be in a form that can be modified dynamically by the recoverer. When a failure has been analysed and a recovery plan devised, the recoverer needs to add new instructions to the AP. The AP representation allows incremental compilation and reliable patching.

The diagram in Fig. 3 shows the AP for the sample problem in a format that is easy to read. The actual internal representation of an AP is a Lisp expression. A state may contain any number of imply and expect forms and no more than one robot-do operation. For example, the operation 'robot-do: move wok (8.44 −15.92 7 −47 0 0)' instructs the robot named wok to move to the specified location. The imply form specifies facts about the work cell that are true when the state is entered, the expect form specifies the expected result of the current state if all goes correctly. For instance, if the program reaches state 5, the system assumes that the object cp1 has been grasped. If the destination of the move operation in state 5 is reached without any error, the system expects that the robot is still carrying the object cp1.

Significant sensor readings are indicated by predicates. For example, the predicate reach becomes true when wok reaches its assigned destination. A transition from one state to the next occurs when a continuously tested predicate such as that becomes true. All predicates in a state are tested concurrently. The AP executive chooses the transition whose predicate is satisfied first. The event predicates are described in Table 2.

Information about object manipulation is not important for operating the robot, but it is vital to error detection and

```
1   robot-do:   move wok (8.44 -15.92 7 -47 0 0)
    when        reach                          go to state 2
                hit                            go to state 9
                joint-error                    go to state 9
2   robot-do:   open wok 1.5
    when        open                           go to state 3
                hand-error                     go to state 9
3   robot-do:   move wok (8.44 -15.92 0.1 -47 0 0)
    expect:     grasp cp1 (8.44 -15.92 0.1 -47 0 0)
    when        reach                          go to state 4
                hit                            go to state 9
                joint-error                    go to state 9
4   robot-do:   center wok
    expect:     grasp cp1 (8.44 -15.92 0.1 -47 0 0))
    when        center                         go to state 5
                missed                         go to state 9
                crush                          go to state 9
                hand-error                     go to state 9
5   robot-do:   move wok (8.44 -15.92 7 -47 0 0)))
    imply:      grasp cp1 (8.44 -15.92 0.1 -47 0 0))
    expect:     carry cp1 (8.44 -15.92 7 -47 0 0))
    when        reach                          go to state 6
                hit                            go to state 9
                untouch                        go to state 9
                joint-error                    go to state 9
6   robot-do:   move wok (8.44 -15.92 0.1 -47 0 0)))
    imply:      carry cp1 (8.44 -15.92 7 -47 0 0))
    expect:     carry cp1 (8.44 -15.92 0.1 -47 0 0))
    when        reach                          go to state 7
                hit                            go to state 9
                untouch                        go to state 9
                joint-error                    go to state 9
7   robot-do:   open wok 1.5))
    imply:      carry cp1 (8.44 -15.92 0.1 -47 0 0))
    expect:     release cp1 (8.44 -15.92 0.1 -47 0 0))
    when        open                           go to state 8
                crush                          go to state 9
                hand-error                     go to state 9
8   imply:      done
9   imply:      error
```

Fig. 3. The AP for the given program.

recovery. At the level of robot control, a carry command is almost identical to a move, except that special sensors may be active to verify that the object is not dropped. The presence or absence of an object in the gripper, for example, would determine whether or not the hitobj event is activated. In the AP itself, states that represent actions on objects will contain special clauses (i.e. the expect and imply clauses) that are inserted into the event trace when the state is entered, as explained in the next section. If an error occurs, the clauses help the recoverer determine

Table 2. Event predicates.

Event	*Description*	*Operations*		
	Manipulator position events	*move*	*open*	*center*
reach	arm reached its destination	X		
joint-error	arm motion error	X		
	Gripper Width Events			
open	hand reached specified opening size		X	
hand-error	gripper error		X	X
	Strain Gauge Events			
touch	gripper touched something		X	
untouch	gripper ceased to touch something	X		
center	force controlled grasp operation completed			X
missed	nothing found to grasp			X
crush	excessive grasp pressure		X	X
hit	unexpected touch on strain gauge	X		
hitobj	unexpected touch on carried part	X		
	LED Events			
detect	gripper LED detected an object		X	X
lost	gripper LED stopped detecting object	X		

where objects really are in the work cell and what the robot was trying to do when the failure occurred. The woktalk language is designed so that the appropriate clauses for a given state are inferred directly from the statement type and from statement parameters.

5. The AP executive

The AP executive is responsible for maintaining an accurate picture of what the robot does. The AP executive does more than simply observe and report on the robot's actions. It takes responsibility for issuing commands to move the robot. When the AP says that a robot action is to occur, the AP executive sends the command to the robot. The AP executive tracks the robot's activities by monitoring data from the robot's sensors. The sequence of sensor data yields an event trace from which we get the robot's recent history.

The testbed utilizes separate processors for executing the reactive, or real time, software components and for executing the reflective, or symbolic reasoning, components of the system. Providing separate processors for the real time and the automated reasoning components of the system prevents time-critical software components from having to compete for computation time. Since the AP executive is the only component that interacts with the robot continuously, a large-scale system would probably consist of several independent work cells, each with its own AP executive, sharing a central server that provides pre-processor and recoverer resources.

5.1. *The event trace*

As the AP is executed the AP executive produces the event trace. The trace tells when state transitions occur, the sensor readings that triggered them, and information about the progress of the robot's task. When a new state is entered, the AP executive adds a new-state entry to the trace. While processing the initial actions, the AP executive writes imply and expect entries into the trace. Sense entries are written when sensor events occur. Event trace entries are transmitted to the recoverer as they occur so that the recoverer may track the robot's activities and handle error recovery.

Figure 4 shows a portion of an event trace during the execution of the task shown in Fig. 2. The Fig. shows trace entries generated during an execution of states 4 and 5. Each entry begins with a timestamp. The new-state entries identify when state transitions occur. The expect and imply entries are generated from the current AP state. The sense entries identify the result of the commanded operation and provide precise numerical feed-back of the result of the operation. Gripper operations return the resulting jaw

```
(228055 new-state task 4 23)
(228055 expect grasp cp1 (8.44 -15.92 0.1 -47 0 0))
(228125 sense center wok 0.963099)
(228126 sense center wok 0.963099)
(228127 new-state task 5 24)
(228127 imply grasp cp1 (8.44 -15.92 0.1 -47 0 0))
(228127 expect carry cp1 (8.44 -15.92 7 -47 0 0))
(228213 sense reach wok (8.44114 -15.9155 6.99949 -46.9778 0.0310315 -0.0207186))
(228214 new-state task 6 25)
```

Fig. 4. Portion of the event trace.

width and motions return a six element vector identifying the resulting manipulator coordinates.

5.2. *Robot task execution*

The AP executive naturally follows an event-driven software structure. Sensory stimuli produce sensory events which in turn invoke procedures to perform state-related actions. The automaton structure permits a simple form of multi-programming. The AP processor implements this through nonpreemptive scheduling of active processes. Each time a process does a state transition the process executes the transition actions to completion before another transition may occur. Once the process completes its transition the scheduler is invoked which then seeks another event ready to process and handles it.

An AP state transition consists of several steps beginning with a sensor event being placed on the scheduler queue. When the scheduler dequeues the sensor event, it checks to see if the active AP is waiting for that event. If so, the scheduler generates a transition action, which is also placed on the queue. When the scheduler dequeues the transiton action, it performs the state transition. This is the point at which a new-state entry is placed in the event trace. The transition action itself involves three steps. First, the scheduler queues look-for actions that correspond to the event predicates in the new state. Next, the scheduler queues the robot-do action, if any, for execution. Finally, the scheduler extracts imply and expect clauses from the state and writes corresponding entries into the event trace.

The transition is complete once the scheduler performs the queued actions. It first dequeues the look-for actions and passes them to the appropriate sensor handlers. These actions direct the sensor handlers to look for appropriate sensory information. If a particular sensor handler does not need such prompting then the look-for action is ignored. Following the look-for actions on the queue is the robot-do action. This action is passed to the software process that controls the robot which in turn instructs the robot to do the action. Following the robot-do action the queue will probably be empty until the next sensory event occurs, probably triggering another state transition.

5.3. *Interfacing to the robot's controller*

As mentioned previously, the system is designed to work with existing, off-the-shelf robot systems. The system simply requires that robots are able to position themselves reliably with control systems provided by the manufacturer. The error recovery system operates the robots in terms of point-to-point manipulator positioning commands; the robots must be able to handle such commands reliably and accurately. Clearly, more functionality is better: built-in sensor systems such as the gripper strain gauges on the IBM 7565 system can be exploited to enhance system performance. Additional sensors such as cameras may be incorporated into the system to analyse the work cell state after an error is detected; such sensors may bypass the AP executive and connect directly to the recoverer. However, sensors such as gripper strain gauges that can verify the success of a robot action, must report to the AP executive.

To operate the robot, the AP executive must take the basic robot-do functions of move, open, and center and generate appropriate commands to operate the robot connected to it. This is usually handled by a robot driver process; a separate driver must be implemented for every type of robot used by the error recovery system. In the testbed system the driver is a process that receives commands to start a robot operation or to read messages sent back by the robot. The driver starts a robot operation by sending a command to the robot. The driver is then instructed to pull the robot's output port. The robot will send a message when it finishes the commanded operation; the driver interprets the message and generates a sensor event indicating the result. The robot's controller must provide sufficient functionality to implement three basic manipulator operations, as shown in Table 3.

Table 3. Operations required from the robot controller.

Instruction	*Operation*	*Robot controller operation*
move	move gripper to absolute location	manipulator movement
open	change gripper's opening to specific width	gripper movement
center	close gripper until object grasped	feed-back from gripper's fingers

The AP executive makes no assumptions about the actual form of robot controller commands and responses or the nature of the robot's interface. Robots we used in our experiments all use standard RS-232 or RS-422 serial interfaces; the commands and responses are all in the form of ASCII text strings. However, this is visible only in the robot drivers themselves. A robot interface could just as easily be through a complicated custom controller con-

nected to the AP executive or through a series of parallel interfaces.

In our implementation all manipulator motions are performed by AMLs MOVE statement. The arguments to MOVE are a list of joints to move, a matching list of destinations, and an optional list of AML sensor monitors that can terminate the motion. Joint destinations are given in cartesian coordinates for x, y, and z positions, degrees for pitch, roll, and yaw angles, and centimeters for the gripper width. The AP move function is performed by a MOVE statement that specifies an absolute destination for the x, y, z, roll, pitch, and yaw joints. The open function is performed by a MOVE statement that affects the gripper opening only. The center operation is performed by a MOVE statement to close the gripper combined with an embedded MONITOR statement to terminate the motion when the strain gauges detect an object being held.

5.4. *Sensor management through filtration*

Sensor information is filtered in several ways. The AP specifies sensor information that is significant to the execution of that task. This specification is given in terms of sensory events that can cause state transitions in the AP. The specification is used both to identify potential state transition events and to identify sensory information significant for the event trace. This specification is also passed to sensor filter tasks that activate appropriate sensors and map sensor values into events.

AP transitions are caused by discrete events, so a robot's progress at its task depends on the occurrence of events that cause appropriate transitions. Significant sensory readings must be mapped into events that cause state transitions. This mapping provides one form of sensory filtration; sensory readings are reported only when the value is significant to the progress of the robot's task. In some cases the identity of the event is the only specific sensory information returned and in other cases numerical data is included as well.

Each AP state contains event predicates identifying sensor readings that would be significant to the successful execution of that state. The AP executive passes the information in the event predicates to the appropriate sensor filters before initiating robot motion. The sensor filters activate appropriate procedures so that necessary sensor readings will take place.

All sensor filtering on the IBM 7565 is implemented using the monitor facility of the AML language. Monitors are used to define ranges of sensor values that can activate user-defined procedures or terminate robot motions. When initializing the IBM 7565, the AP executive defines a set of monitors for classifying gripping forces and associates each monitor with an AP event type. The numerical values used for classifying gripping force depend on the objects being used in the robot's task and the actions performed on them, so these values may be adjusted when a task begins. More details on how to obtain these values are in the next section. When the AP executive gives the IBM 7565 a motion command, it also specifies a set of monitors to activate. The AML system collects the appropriate sensor readings for each active monitor and trips the appropriate monitor if its sensor enters the monitor's defined range. This terminates the motion in progress and generates a message to the AP executive identifying the qualitative value of the sensor reading, as determined from the monitor that was tripped. If no monitor terminates the active motion, a similar message indicating uninterrupted completion is sent instead. The AP executive then generates a sensor event and, if necessary, updates the event trace and performs an AP state transition.

Figure 5 shows an example of the transformation of a move operation in an AP state into the corresponding AML commands executed by the robot. The desired destination and the desired AML monitoring sets to be activated (**E_HIT** and **E_UNTOUCH**) are passed to the APM procedure. This procedure, written in AML and executing on the IBM Series/1, performs the MOVE operation and the related filtering for the 7565's sensors. The procedure activates the appropriate monitors and performs the motion subject to the selected monitors.

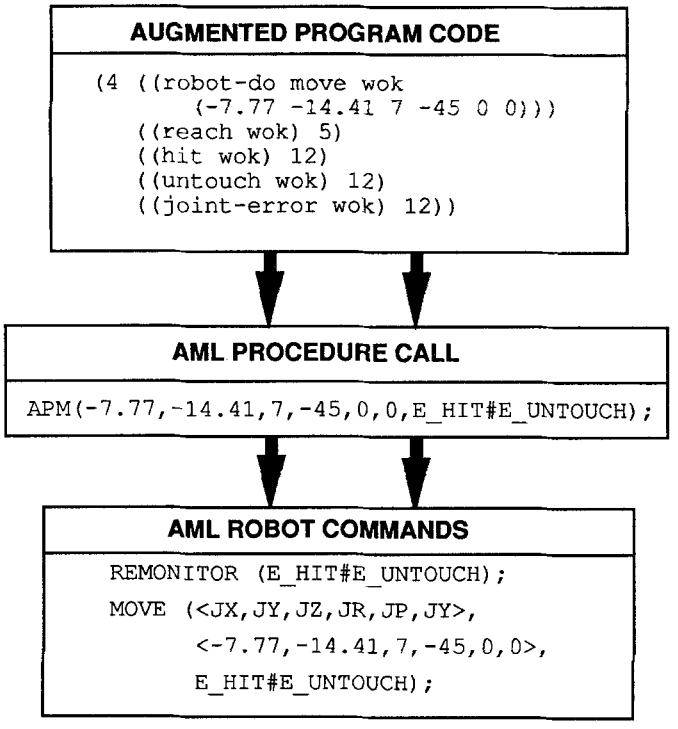

Fig. 5. Converting AP statements into AML statements.

5.5. *Qualitative sensor interpretation*

Although the event trace often provides numerical sensor data, such information is not of primary importance when reasoning about the robot's activities. To meet this need, the error recovery system assigns symbolic meanings to numerical sensor values in a number of ways. Spatial locations and critical dimensions are assigned symbolic names. Gripping forces are assigned qualitative values according to the range in which a force value falls.

Qualitative classification of sensor data serves a second purpose as well. When executing an AP, the AP executive responds to events in terms of symbolic classifications. Upon successful completion of a motion command the AP responds to a reach sensor event instead of examining and matching the robot's reported destination. If the gripper drops an object and the gripping force drops to a small value, the AP responds to an untouch sensor event instead of testing the specific force value. The classification of sensor values into different types of AP events is performed by a sensor filter procedure that operates on the behalf of the AP executive, as described above.

Gripper forces, when they are significant, determine whether the gripper is touching an object and holding with an adequate force. Identification of appropriate touching and grasping forces must be communicated to the AP executive so that appropriate AP state transitions occur depending on the gripping forces encountered. The sensor filter classifies gripping forces into specific ranges according to the robot's current action. Each range corresponds to a type of sensing event that can be produced by the gripping force sensor.

Ranges for strain gauge forces were determined experimentally by measuring side, pinch, and tip forces while the robot manipulated objects. Figures 6 and 7 summarize the range of pinch and tip forces encountered while grasping a coupler held in a slot. The experiments identified the normal force values and the variations thereof encountered during normal operation. This information defined an envelope for acceptable sensor readings. We used the experimental information to assign values to symbolic ranges, as shown in Fig. 8. Each symbolic sensor event corresponds to a range tested by a MONITOR statement in the AML sensor filtering code. The appropriate monitors are activated at the beginning of the robot's action. If a tested strain gauge reading enters the range marked by the dashed line the appropriate monitor is tripped. When a monitor is tripped it terminates the robot's action and the corresponding sensory event is passed to the AP executive. The actual boundary values may be changed when initializing and calibrating the testbed system.

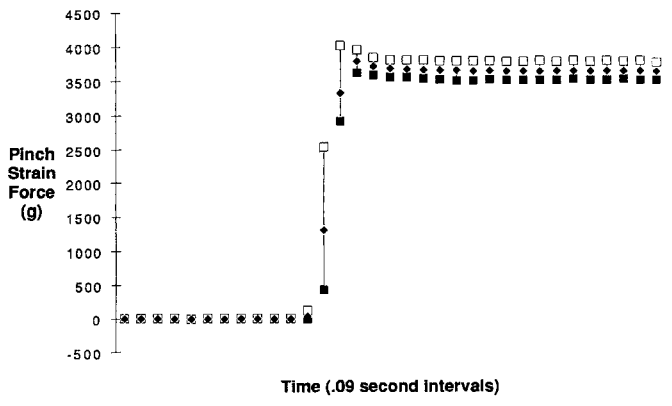

Fig. 6. Range of pinch forces during a grasp.

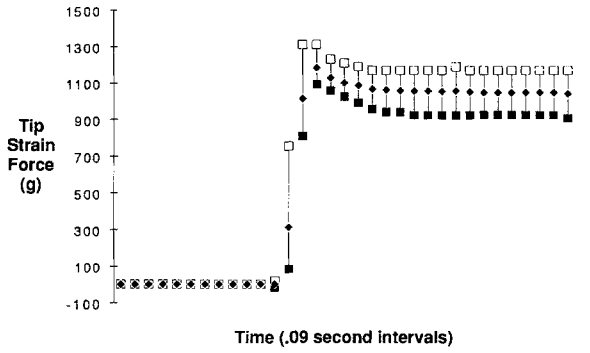

Fig. 7. Range of tip forces during a grasp.

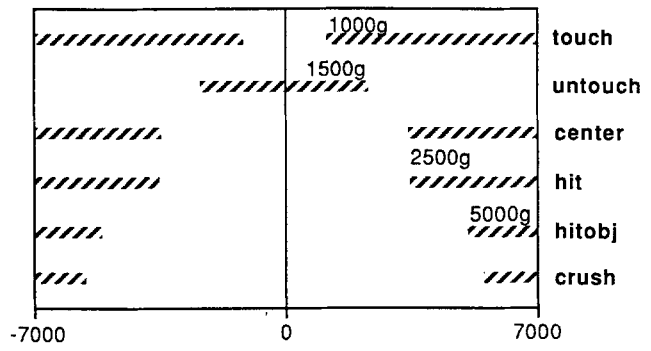

Fig. 8. Interpretation of grasping forces.

6. The recoverer

If an AP state transition leads to an error state, a message to that effect is appended to the event trace and the trace is passed to the recoverer. The recoverer generates a model of the current work cell's state and of its desired state using information in the event trace (Smith and Gini, 1986a). Locations visited by the robot or by objects in the work cell

are assigned symbolic identifiers, and a history is produced of visits for each location, object, and robot gripper in the work cell. This model is used to produce a recovery plan in the form of a sequence of AP states to be appended to the task's existing AP. The recoverer passes these additional states back to the AP executive where they are executed. If the new states each execute successfully, they will lead the task back to a state in the original AP. To effect recoveries in this manner successfully, the recoverer requires a copy of the task's AP and the information in the event trace.

6.1. *Error recovery planning*

The kinds of recovery activity of which the system is capable depends on the level of abstraction at which the system understands the task. This is related to the notion of levels of abstraction in robot programming languages. If the system understands nothing about the task beyond the manipulator level, then effective recovery is seldom possible. On the other hand, if the system understands the task in terms of factory inventories and mechanism design, the system could conceivably design and build a new part in response to an error. For assembly tasks in this research, however, the highest level at which a task is understood is in terms of object motions. Task goals, failures, and recoveries are understood in terms of the desired positions of parts in the assembly. If errors are understood primarily as situations in which parts are misplaced, then recovery consists primarily of putting parts back where they should be. This is often referred to as the pick and place synthesis problem (Lozano-Perez, 1983).

Classically, a pick and place motion planner requires three types of input data. First, it requires a description of the manipulator, available parts, and of the work cell. In the error recovery testbed, this is provided by the task knowledge base. Second, it requires the current configuration of the manipulator and parts in the work cell. This information is derived from the event trace. Finally, the planner needs to know the desired destinations of parts to be moved. The error recovery testbed derives this information by comparing the expected and actual results of object motions after an error is detected. The pick and place planning problem is often decomposed into two subproblems: grasp and path planning. Grasp planning consists of selecting the appropriate sequence of motions to grasp an object rigidly. This is simple if the object is always grasped while being held in a fixed position, but it is quite difficult if the part's position is unconstrained. Our experiments relied on simplifying assumptions for grasp and path planning as described in the next section.

Although pick and place problems are the most complex problems addressed here, the system encounters other types of errors as well. For example, robots may encounter controller errors, hydraulic or power failures, and operator intervention *via* safety switches. Errors such as these prevent the robot from performing the attempted manipulator motion. Unlike errors involving objects, these failures prevent the robot from performing its task at the manipulator level. When such an error is detected it can be recovered from by retrying the manipulator level operation that failed. Manipulator level errors may be classified as soft or hard errors. A soft error is a situation in which the robot is disabled only momentarily and can recover without operator assistance or repair. For example, some robot controllers experience momentary instabilities or wobbles which infrequently prevent them from completing a motion. Such a failure is intermittent and is unlikely to occur if the operation is retried. Recovery consists of simply retrying the failed operation. On the other hand, some manipulator level errors are serious enough to require operator intervention or even repair. For example, the IBM 7565 might lose its counterbalancing air pressure and cease to operate until the pressure is restored. This would constitute a hard error since the robot can not resume operation without operator assistance. However, once the problem has been resolved the recovery strategy is the same as for soft errors: simply retry the failed operation. A recovery plan for hard errors, then, must ask for operator assistance and, once the problem is resolved, should retry the operation that failed.

6.2. *The woktalk planner*

In order to simplify grasp and path planning, the error recovery planner exploits some simplifying aspects of the testbed's work cell. The testbed assembly tasks all use plastic parts that are large and easily sensed by the strain gauges on the IBM 7565's gripper. Assembly parts are provided with individual slots to hold them in a precise position for grasping. Parts, slots, and fixtures are spread about the work cell surface so that all may be reached through straight vertical motion without the risk of colliding with other work cell obstacles.

Grasp planning requires information about the part's position and the locations of appropriate grasp points on the object. This information is extracted from the TKB. The recovery planner always plans to grasp parts that have been installed in slots designed for the purpose. The appropriate gripper location for grasping a particular part is given in the TKB as an attribute of the slot containing the part. The grasping width for approaching the part is derived from information on the part contained in the TKB. This provides sufficient information for grasping the part.

Path planning exploits the flat surface nature of the testbed's work cell lay-out. Typically, a path needs to be planned from one location near a slot or fixture to another. These motions are planned in three steps: up, over, then down. Collision avoidance is achieved by moving up from the surface of the work cell and into a parking plane during

the up step. The parking plane is a region high enough above the work cell that the manipulator can move anywhere in it without hitting anything in the work cell. Its location is inferred from the park location defined in the TKB. The over step places the manipulator on top of its destination and the down step approaches the desired destination.

The recovery planner uses a state-oriented model of parts and locations in the work cell. Error interpretation for purposes of recovery attempts to achieve the expected occupancy states of parts and locations in the assembly task. Parts may be in use or spare or lost, depending on the progress of the task and the effects of errors and recovery plans. Locations are usually occupied or empty and are usually manipulated symbolically though each location's geometric position is also known. A typical recovery may be to replace the missing part, x, with a new part, x, currently in location, a, and move the part to its expected position of location, b.

When the recoverer is invoked it uses a four-step process to produce the recovery plan. First, it uses the event trace and TKB to update the work cell model. Next, it chooses its recovery plan depending on the type of error that occurred. Third, it generates assertions about the expected and desired state of the work cell using the event trace and the work cell model. Finally, the recoverer selects a recovery plan according to the type of error; the specific plan steps are generated by applying the assertions to the individual steps of the recovery plan. Plan steps are generated in woktalk and converted to AP form before being sent to the AP executive.

The recoverer generates a pick and place recovery whenever a sense event occurs that indicates trouble with the part the manipulator is carrying. The primary purpose of gripper strain gauge readings in the testbed system is to verify the proper transportation of a gripped part, so the recoverer usually generates a pick and place recovery in response to strain gauge sense events. The generalized pick and place recovery plan treats it as a problem of discarding an invalid part and acquiring a new one. The generalized pick and place recovery plan consists of six steps. The first step is to move to the parking plane in preparation for the next step. The next step is to discard the part currently held in the manipulator, if any. The TKB defines a discard location that is used for this purpose. The third step is generated if the recoverer believes there is an obstruction in the part's destination: a message is generated that asks the operator to remove any obstruction from the specified location. The fourth step generated if there are no spare parts of the required type available: an operator message is generated to ask for the desired part to be provided. The fifth step picks up the replacement part from its initial location and moves it to its expected destination, fulfilling the forward recovery. The sixth step generates a state transition into the forward recovery state.

6.3. *Recovery from multiple errors*

If an error occurs, the recoverer passes additional AP states to the AP executive. These additional states do not replace existing states in the AP; they are appended to them. The AP executive resumes task execution with the first of the recovery states passed to it. Once the recovery execution begins, the AP executive treats the recovery states identically to the states in the original AP. If another failure occurs, whether during the recovery or after completing the recovery, the AP executive again reports the failure to the recoverer and resumes execution when it receives a set of recovery states. For example, if the robot loses a part, it can attempt a recovery by opening the gripper, moving to the work cell surface, and trying to grab the part. If the part is there, the recovery can proceed. If the grasp fails, the AP executive simply informs the recoverer which can then produce another recovery plan and try again.

The ability to do multiple recoveries allows the recoverer to profit from mistakes in a recovery plan. When faced with multiple recovery choices, the recoverer can choose the one that is most likely to reduce uncertainty about the state of the work cell. The recoverer can also produce recovery plans with the sole purpose of taking sensor readings in the robot work cell. If the recoverer needs to probe a specific spatial location it can produce a recovery plan that performs the desired sensor reading and then immediately fails. The resulting event traces will increase the amount of information in the work cell model and the unsuccessful recovery will not prevent a subsequent recovery from being attempted.

Another useful feature during error recovery is the AP executive's ability to display messages for the robot's operator. These messages are produced by statements in the AP and thus may be generated by the recoverer. This facility allows the recoverer to request specific operator intervention when necessary.

6.4. *Experimental results*

We have performed a variety of simple experiments involving repetitive tasks, all of them using the workcell lay-out described in Table 1. Multiple parts were alternately grasped, moved, and released so that all basic operations occur repeatedly.

We have collected results on experiments with two task procedures, called pkplc and pkpair. Pkplc is a repetitive task that moves a coupler from its slot to the fixture and back. Pkpair is more complex; it moves two couplers around the work cell. The couplers to be moved are placed in slots 1 and 2. The task starts by using the coupler in slot1. The gripper picks up the coupler from its slot, moves the coupler to the fixture, inserts it into the fixture, then puts the coupler back into its original slot, releasing it. The gripper then moves to the coupler in slot2 and repeats the same sequence of actions on it. The task then repeats.

Table 4. Summary of experiments.

Task	*Successful failure/ recovery cycles*	*# Runs*	*Activity*	*Successful failure/ recovery cycles*	*Error*	*Recovery*	*# Recoveries*
pickslot	9	3	move coupler	9	part lost/removed	get another part	6
pickpair	5	1	insert coupler	3	collision/part dislodged	discard part and get another	3
			grasp coupler	2	destination obstructed	discard part and get another	2
					part missing	get another part	3

Table 4 summarizes the results of a series of video-taped experiments that used the error recovery testbed. These experiments relied entirely on the gripper strain gauges for sensing. For each task run, we started the robot and repeatedly induced failures of various kinds. We induced the errors manually by pulling parts out of the gripper, striking it, putting objects in its path, stealing parts that were about to be used, and placing foreign objects in a part's destination. In Table 4, the column listing Successful failure/recovery cycles indicates the number of times that we induced a failure and the system recovered, resuming its previous task. Two of these recovery cycles took place while the robot was already executing steps to recover from a previous error, demonstrating the system's ability to recover from nested errors. In other experimental runs the system also demonstrated its ability to recover from errors during nested recoveries as well.

Each run continued until encountering an error the system could not handle. In two cases, the part collided with a foreign object with insufficient force to trigger the error event. In another case, the robot collided with an obstructed part and did not have a recovery strategy that ignored the resulting overforce measurement; this caused subsequent motion commands also to fail with the same overforce error. In another case, an eager operator provided extra spare parts that were not explicitly requested by the system and thus were not reflected in its workcell model or in its behavior.

7. Conclusions

We have presented an architecture for building programmable industrial robots that automatically detect and recover from errors encountered while manipulating parts. The architecture separates the real time robot control components from the task analysis, programming, error interpretation, and error recovery planning components. The system uses one computer to run the AP executive, providing real time robot control, and a separate computer for the symbolic computation and recovery tasks. The architecture relies on the augmented program form, a special representation of the robot's task, to provide information needed for robot operation and automatic error recovery.

An important feature of this system is that it is designed to work with conventional robot programming languages. If a task-level programming language does become available the error recovery system could work with it. The statements of the woktalk language used in the error recovery testbed are extremely similar in form and content to the primitive robot operations generated by typical task-level motion planners. Thus the error recovery system could serve as a back end to a task planner with minor modifications.

A basic criterion of the system design has always been language independence, both for the input language that specifies the robot's task and the output language that controls the robot. New input languages may be incorporated by implementing a pre-processor for them. Pre-processors have been designed to work with AL and woktalk; a pre-processor for AML or VAL is expected to present no problems. New robots may be incorporated by implementing AP executive driver processes for them. We have implemented drivers for the IBM 7565 and for the Microbot TeachMover.

Another important feature is how the system combines information about the robot's task. The AP is used to operate the robot in real time so it describes the sequence of manipulator motions the robot must perform. Associated with each motion there are statements of how the motions are intended to affect the robot's task. Each time the AP executive starts a robot motion, the statements associated with the motion are written to the event trace. When the motion is finished, sensor readings that detect its completion are also written to the trace. This provides a detailed record of events in the work cell, consisting both of sensed facts and of deductions about the effects of the robot's actions.

Sensor filtering is also an important feature of the AP executive. Filtering the sensor data reduces the overhead associated with analysing sensor data and may in some cases be performed on a separate processor.

While it is important to recognize the design features that make the system work, it is also important to recognize the features that make it applicable to other problems. The error recovery testbed demonstrates the system's feasibility for performing woktalk programs on an IBM 7565 robot, but the underlying architecture makes it possible to adapt the system to other robot languages or robot hardware.

The system also has the flexibility to incorporate improved automated reasoning, deductive, and planning systems. The architecture has a mechanism to relate symbolic information (e.g. assertions, deductions, even rules) with manipulator actions, but the content of this information is transparent to the underlying system. The contents are subject only to cooperation between the pre-processor and recoverer. There is no architectural impediment to implementing a task-level programming system to serve as the pre-processor as long as it generates APs. Similarly, the recoverer could be implemented using a high-performance spatial reasoning and robot motion planning system. Many techniques exist that could be adapted to this system; the architecture is designed to make this possible.

Acknowledgements

This work was funded in part by the NSF under grants NSF/DMC–8518735 and NSF/CCR–8715220, and by the Productivity Center of the University of Minnesota.

References

Cox, I. J. and Gehani, N. H. (1989) Exception handling in robotics. *IEEE Computer*, **22**, 43–9.

Donald, B. (1990) Planning multi-step error detection and recovery strategies. *International Journal of Robotics Research*, **9**, 3–60.

Doyle, R. J., Atkinson, D. J. and Doshi, R. S. (1986) Generating perception requests and expectations to verify the execution of plans, in *Proceedings of American Association for Artificial Intelligence, 86*, Philadelphia, pp. 81–7.

Fielding, P. J., DiCesare, F. and Goldbogen, G. (1988) Error recovery in automated manufacturing through the augmentation of programmed processes. *Journal of Robotic Systems*, Philadelphia, PA, **5**, 337–62.

Fielding, P. J., DiCesare, F., Goldbogen, G. and Desrochers, A. (1987) Intelligent automated error recovery in manufacturing workstations, in *Proceedings of the IEEE Inter Symposium on Intelligent Control*, pp. 280–5.

Fikes, R. E. and Nilsson, N. J. (1971) STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**, 189–208.

Gini, M. (1990) Automatic error detection and recovery, in *Robot Technology and Applications*, Rembold, U. (ed.), M. Dekker, New York, 445–483.

Gini, M. and Gini, G. (1983) Towards automatic error recovery in robot programs, in *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, pp. 821–3.

Green, A. E. and Bourne, A. J. (1972) *Reliability Technology*, Wiley, London.

Hardy, N. W., Barnes, D. P. and Lee, M. H. (1989) Automatic diagnosis of tasks faults in flexible manufacturing. *Robotica*, **7**, 25–35.

Harmon, S. Y. (1988) Dynamic task allocation and execution monitoring in teams of cooperating humans and robots, in *Proceedings of the 1988 Workshop on Human-Machine Symbiotic Systems*, Oak Ridge.

Henderson, T. and Shilcrat, E. (1984) Logical sensor systems. *Journal of Robotics*, **1**, 169–93.

IBM Corporation (1982) A Manufacturing Language Reference, Publication 8509015, IBM Corporation, Boca Raton, FL.

Lee, M. H., Barnes, D. P. and Hardy, N. W. (1983) Knowledge based error recovery in industrial robots, in *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pp. 824–6.

Leveson, N. G. (1986) Software safety: what, why, and how. *Computing Surveys*, **18**, 125–63.

Lopez-Mellado, E. and Alami, R. (1986) An execution monitoring system for a flexible assembly workcell, in *Proceedings of the 16th ISIR*, pp. 955–62.

Lozano-Perez, T. (1983) Robot programming. *Proceedings of the IEEE*, **71**, 821–41.

Lozano-Perez, T., Mason, M. T. and Taylor, R. H. (1984) Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, **3**, 3–24.

Lumia, R., Fiala, J. and Wavering, A. (1989) The NASREM robot control system standard. *Robotics & Computer-Integrated Manufacturing*, **6**, 303–8.

Lyons, D. M., Vijaykumar, R. and Venkataraman, S. T. (1989) A representation for error detection and recovery in robot task plans, in *Proceedings of the 1989 SPIE Symposium on Advances in Intelligent Robotics Systems*, Philadelphia, PA, Vol. 1196.

Narayanan, N. H. and Viswanadham, N. (1987) A methodology for knowledge acquisition and reasoning failure analysis of systems. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-17**, 274–88.

Nof, S. Y., Maimon, O. Z. and Wilhelm, R. G. (1987) Experiments for planning error_recovery programs in robotic work, in *Proceedings of the ASME International Conference on Computers in Engineering*, New York.

Randell, B., Lee, P. A. and Treleaven, P. C. (1978) Reliability issues in computer system design. *ACM Computing Surveys*, **10**, 123–65.

Simpson, J. A., Hocken, R. J. and Albus, J. S. (1983) The automated manufacturing research facility of the national bureau of standards. *Journal of Manufacturing Systems*, **1**, 17.

Smith, R. (1987) An autonomous system for recovery from object manipulation errors in industrial robot tasks, PhD Thesis, University of Minnesota.

Smith, R. and Gini, M. (1986a) Robot tracking and control issues in an intelligent recovery system, in *Proceedings of the 1986 IEEE Conference on Robotics and Automation*, San Francisco, (A, pp. 1070–5.

Smith, R. and Gini, M. (1986b) Reliable real-time robot operation employing intelligent forward recovery. *Journal of Robotic Systems*, **Fall**, 281–300.

Srinivas, S. (1977) Error recovery in robot systems, PhD Thesis, CIT.

Taylor, R. H., Summers, P. D. and Meyer, J. M. (1982) AML: a manufacturing language. *International Journal of Robotics Research*, **1**, pp. 19–41.

Trevelyan, J. P., Nelson, M. and Kovesi, P. (1988) Adaptive motion sequencing for process robots, in *Proceedings of Robotics Research, the 4th International Symposium*, Bolles, R. and Roth, B. (eds), The MIT Press, pp. 445–53.

Tsukanoto, Y. and Terano, T. (1977) Failure diagnosis by using fuzzy logic. *IEEE Proceedings Decision and Control*, **2**, 1390–5.

Wilkins, D. (1985) Monitoring the execution of plans in SIPE. *Computational Intelligence*, **1**, 33–45.

Williams, D. J., Rogers, P. and Upton, D. M. (1986) Programming and recovery in cells for factory automation. *International Journal of Advanced Manufacturing Technology*, **1**, 37–47.

Zhou, M. C. and DiCesare, F. (1989) Adaptive design of Petri net controllers for error recovery in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics*, **19**, 963–73.