Jeanine Meyer

An Emulation System for Programmable Sensory Robots

This paper describes EMULA, an experimental interactive system for the emulation of sensory robots. EMULA was constructed as a bridge between an existing language for driving actual robots and an existing geometric modeling system. The modeling system was extended to handle mechanisms, such as robots, and an emulation language was introduced to indicate certain specific physical effects, including sensory feedback, grasping and releasing of parts, and gravity. EMULA allows manipulation programs to be tested by users in interactive terminal sessions or in batch mode. Monitoring functions are provided to record actions, store selected views, and check for collisions.

Introduction

During the past 20 years many languages have been developed for programming industrial robots [1–4]. Although some of these languages allow off-line programming without tying up production equipment, they do not in general provide facilities for off-line debugging of these programs. A few systems have been described that do allow off-line simulation of robot motions [5–9], but none allows off-line simulation of any sensory feedback to the robot. Since the typical robot program contains conditional branches that depend upon execution-time sensing, the inability to simulate sensors is a significant limitation to off-line program preparation. EMULA is an experimental software system for emulating assembly robots that provides a general method for emulating some forms of sensory feedback [10].

EMULA consists of four basic subsystems: a modeling system (an enhanced version of the geometric modeling facility GDP [11]); a programming language, AML (the language used to run the robot at this laboratory); an emulation system; and a graphics subsystem.

After an initial phase in which robot models are created, EMULA accepts statements in AML, through which the robot joints and sensors can be referred to directly. Any statements involving movement cause the corresponding changes to occur in the model of the robot. Specifically, EMULA evaluates motor and sensor values according to a basic sampling cycle much like the real-time control system that controls the actual manipulator. Transformations of the geometric models as well as monitoring functions take place during each basic cycle. The time granularity represented by the basic cycle can be changed by the user. A journal can be maintained of robot actions and monitored collisions.

The model can be displayed at intervals defined as a multiple of the basic cycle. For example, sets of pictures for an animated film were produced by setting the display rate to be 24 frames/second. The screen can be erased between displays, either from the terminal or automatically. If the display is not erased, we have the multiple exposures shown in some of the figures in this paper. Output from EMULA is normally displayed on a video terminal and on a graphical display device (IBM/3277 graphics attachment, TEKTRONIX display).

The unique aspect of EMULA to the robot user is that sensory events can be simulated. This is done through emulation commands that associate functions with sensors. Functions provided for sensor simulation include geometric interference and probabilistic tests. With the former, the user can specify that a sensor is to trip if the geometric models for objects are in contact or intersect. With the latter, the user can simulate some situations in which the positions of objects are somewhat uncertain. In addition, EMULA is extensible; user-provided sensor simulation functions can be added.

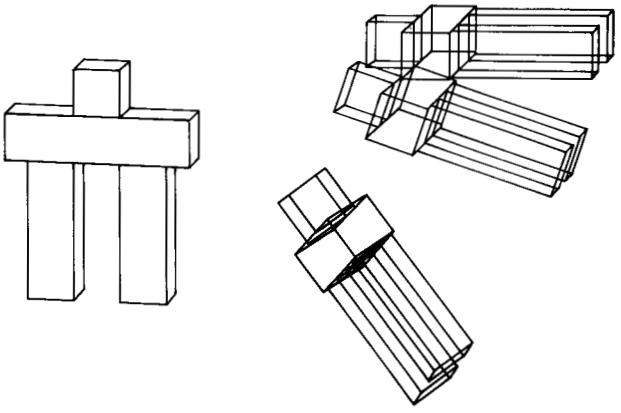

Figure 1 Disembodied hand.

In this paper, we next describe the extension of the modeling system to mechanisms and the simulation of sensors. We then provide examples of the use of EMULA, including copies of actual displays produced on a graphics terminal during sessions. The examples show the use of emulation statements concerning real-world physical effects, such as sensory feedback, grasping, and gravity.

Extension of modeling

Our robot emulation system is an extension of the Geometric Design Processor system, GDP [5, 10]. In fact, any geometric modeling facility which can represent complex objects as a collection of parts and, as in the case of a robot, a collection of parts with a specified articulation, would be satisfactory. In GDP objects are modeled as a hierarchy of sub-objects that are constructed from geometric primitives. Sub-objects can be defined as "solids" or "holes." The user can cause sub-objects to be "merged" to create new objects—polyhedra representing the sum of the descendant "solid" and "hole" parts. Once the elements of a subtree are merged, the resulting object is indivisible; its original sub-parts cannot move with respect to one another.

One can cause the model to be displayed in various ways (*i.e.*, projections, views, etc.). One can also cause parts of the model to undergo a geometric transformation. In the interactive GDP system, the user may indicate that a node is to undergo a linear transformation along a single axis or along all three axes or a rotational transformation is to take place with respect to one of the three axes.

The EMULA system models a robot in two distinct ways: In the geometric subsystem (GDP), a robot corresponds to a sub-object of the geometric model. In the emulation subsystem, the robot state is represented. This state consists of a set of values representing motor positions and various system control variables, such as motor speeds and joint limits. The geometry and the state are brought together by specifying that certain sub-objects of a geometric model are to undergo a certain set of transformations, these transformations being functions of the motor values, the changes in motor positions, the motor speeds, limits, etc. The definition of an emulation robot model, therefore, is this specification of state variables, along with a set of transformations of specified sub-objects. A certain amount of implicit information exists, most notably the actual initial positions of nodes of the geometric model. In contrast, the initial values of the motors are given explicitly. A related issue is that we make no attempt to model or validate the "physics" of a model and its associated mechanics. For example, we do not check whether the transformations leave a part of the model dangling unsupported in space.

We use the term "family" to include all models intended to represent the same configuration of robot. The family classification system allows the programmer to use different models, of varying complexity, during a single emulation session. The basic EMULA system accepts several 7-axis robot models, ranging from the simple, disembodied hand shown in Fig. 1, to the stylized, articulated box-frame robot shown in Figs. 2 and 3, to the more detailed robot shown only in part in Fig. 4. EMULA can also work for different manipulators as well as different models of the same (*i.e.*, 7-axis) manipulator. New models and/or new families can be made acceptable to the emulation system through an interactive or batch program as proposed in [9]. This program accepts data specifying the number of motors and sensors, motor limits, a required set of nodes, and details of the transformations of these nodes.

Thus, in EMULA it is possible to produce an emulation system for a variety of robots, together with arbitrary fixtures, tooling, and end-effectors, and to use a range of models for these robots, ranging from simple to complex. We achieve this flexibility by keeping the geometric modeling distinct from the other subsystems, that is, the robot language, the mechanics of simulation, and the graphic display.

Simulation of sensors

Advanced industrial robots are equipped with sensors in order to obtain information (feedback) about the "real world" and to act on that information. For the purposes of our emulation, we postulate that a sensor registers a single real value at any time and may be subject to monitoring. A sensory event occurs when a sensor's value exceeds some pre-established threshold. The event

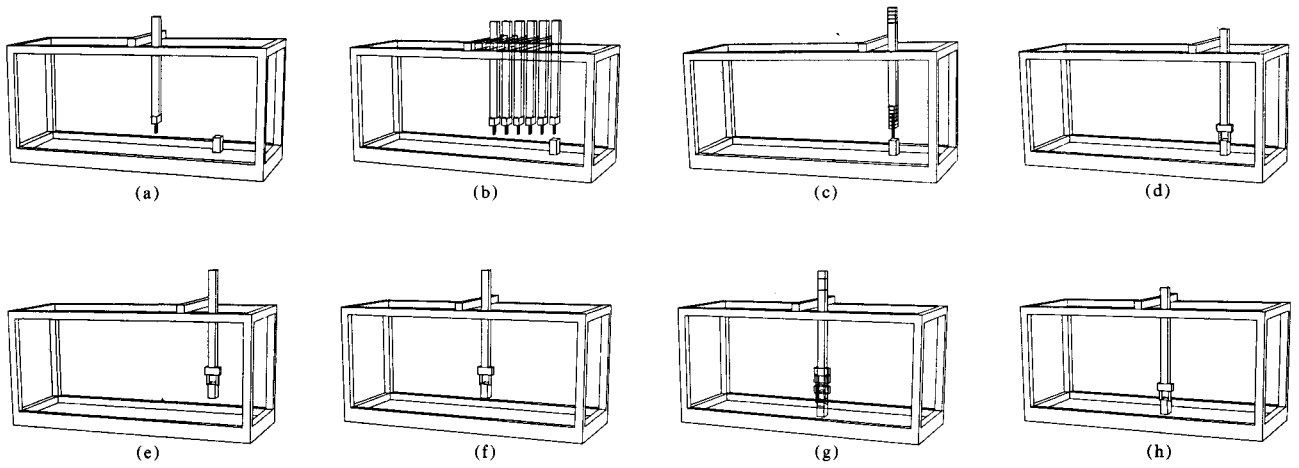

Figure 2 Sequence using CONTACT.

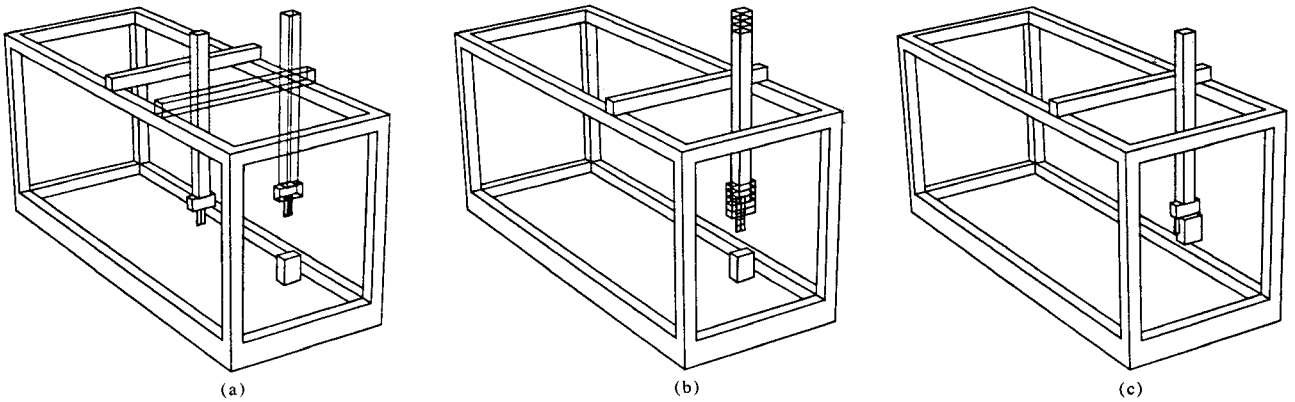

Figure 3 Sequence using RANDOM and adjustable option on GRASP.

causes motion to stop. The prototypical sensor is a force/torque sensor that permits the robot to feel for objects in its work place.

EMULA provides an extensible facility for simulating sensors so that a variety of sensor functions can be used. The emulation command ATTACH is used to associate a sensor with a function. The association of a function with a sensor is dynamic and can be changed. At each cycle, the pseudo-real-time system invokes all functions associated with sensors and sets each sensor with the value returned by its function. The subsequent interpretation of these values is left to the execution of the manipulator level language statements. Some of the emulation functions to be described are essentially predicates concerning events, and the emulation function arbitrarily picks the current upper threshold defined for the sensor to return as the sensor value. This causes the desired triggering of a sensory event. We now describe functions for simulation of sensors which we have implemented.

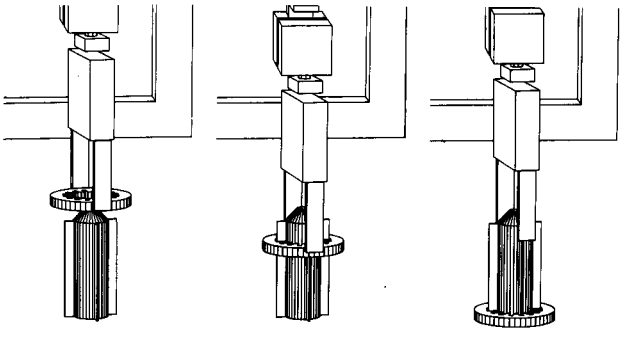

Figure 4 Sequence using SUPPORT showing a part dropping.

The CONTACT function can be used to detect contact of the robot with known objects at known locations in the work space. This function establishes a check at each cycle for touching or intersection between the objects indicated as arguments. It is based on the primitive interference check provided by the geometric modeling system. Though we accept only a contact/no-contact determination from this function, the base operation computes the exact structure of the interference (one, two, or three dimensional objects). A more elaborate sensor function could return a value proportional to the volume of the intersection. Alternatively, CONTACT could be modified to distinguish surface contact from penetration. This may prove useful for force controlled insertion between tight fitting parts.

Certain kinds of noncontact sensors may be simulated using the CONTACT function. For example, the robot at this laboratory has a light beam sensor which can detect the presence or absence of an object between the fingers of the gripper. A model for the robot can be modified by the addition of a cylinder representing the light beam. Detection of contact of this cylinder with any object would correspond to the beam being broken. The cylinder would not be added as a sub-object of either of the fingers; thus, other simulation of sensors would still be valid.

If the exact position of objects in the work space is uncertain, or if it is important to test the robot program for unpredictable events, sensors can be simulated by functions that do not depend on the specification of geometric models.

The QUERY function allows the user to set the sensor value interactively at each cycle. Thus, if there is a particular, delicate interval in the assembly sequence at which a sensory event may occur, the QUERY function could be used to simulate a variety of test conditions. QUERY requests terminal input to set the sensor to an arbitrary value, to set the sensor to the current limit, or to make no change.

The RANDOM function allows the user to put some limited indeterminacy into the system by associating a sensory event with a procedure that accesses a pseudo-random number generator. RANDOM allows the user to establish that the sensor-triggering event will take place with linearly increasing probability, the exact parameters of the distribution being adjustable through the parameters supplied to RANDOM. For example, a typical situation in robotics is to assume that an object will probably be found at some point along a specific path and to program the robot to move along the path until a sensor (force, tactile, or LED) is triggered by the object. The RANDOM function provides a means for modeling such a search sequence. We note that for any single search operation the indeterminacy corresponds to one degree of freedom, *e.g.*, the position of the object along one dimension. That is, we are not simultaneously simulating indeterminacy in position and extents along multiple dimensions. Later we describe how the GRASP command with an adjustment option can be used to continue simulation following a probabilistically determined event.

We have implemented a slightly different sensor simulation function in conjunction with a primitive modeling system consisting of rectangular solids. The contact/intersect test is simplified, and a probabilistic function that simulates a sensory event sometime during each move is provided. Our experience with this mini-system shows that emulation of robots with simulation of sensors is possible even when the underlying modeling system is quite rudimentary and there is no graphical display. In fact, this exercise with a primitive modeling system led us to consider implementation of a more elaborate contact/intersect function that involves fuzzy objects. In this system, we define an object (block) of fixed extents and, whenever the user chooses, apply a probabilistic function to modify its boundaries. The sensor simulating function references the modified boundaries. For complex models, this would involve a nontrivial amount of computation. This function is an example of a further extension of geometric modeling. The definition of the (real world) model is made to be the product of an original, static, geometric model and emulation commands.

The functions described here cannot handle all types of sensory feedback. This is one reason why we chose not to integrate the emulation of sensors with the geometric modeling or the robot programming language and why we provide for user-supplied functions to be associated with sensors. However, the functions do fulfill some common requirements: The geometric interference check (CONTACT) can be used to do preliminary feasibility tests, perhaps make a movie of a new configuration of robot. The QUERY function can be used for unpredictable events, such as spurious sensor readings. The RANDOM function can be used, as shown below, to model a one dimensional search sequence, a common use of force/torque sensors. Repeated use of RANDOM or any other sensor function in conjunction with the appropriate AML code can be used to program the robot for more elaborate operations. By an artful use of more than one sensor simulation function, the user can perform some valuable off-line testing of an assembly sequence.

Examples of emulation

The use of EMULA is first illustrated for a sample program in which a 7-axis, box frame manipulator searches for a part, picks it up, moves with it, and places it down again on the work table. The program to perform this task would use three guarded moves, that is, moves preceded by establishment of limits on one or more sensors. Such a move is stopped if a sensor limit is exceeded. One guarded move is required to find the part, another to grasp the part, and a third to locate the table top when placing the part. An example of what the robot program would look like in a somewhat simplified pseudo-code follows:

```
move (−2,10,8)   /* moves over and above part */
movedown (10) until sensors (1,2) exceed (10) /* guarded
move */
open /* open fingers */
rotate (90)
movedown (1)
close until sensors (1,2) exceed (10) /* guarded move to
grasp part */
moveup (10)
move (2,−10,0)
movedown (10) until sensors (1,2,3) exceed (10)
/* guarded move using wrist sensors to detect part
touching table top */
```

We now present two different examples of emulation sequences for this same robot program. The difference is in the function used to simulate sensor activity. In the first example we use CONTACT and in the second, RANDOM.

The sequences of displays in Figs. 2 and 3 are from these two emulation sessions. We note that these figures were copied from what actually appears on the display. Among other features of EMULA, they demonstrate switching back and forth between the two basic modes governing the emulation of robot movements: the normal, "incremental" mode and the "complete move" mode. In the incremental mode, the basic cycle is somewhat, though not completely, analogous to a real-time cycle of the actual manipulator control system. A move is divided up into sub-moves corresponding to the sampling time of the control system. The model is transformed at every sub-move. In the complete move mode, the entire move, as given in the manipulator command, is done during a single cycle.

We now describe our sample emulation sessions. Figure 2(a) shows the manipulator in its initial position. A command is then given to move the arm to a position over the part. Figure 2(b) shows a multiple exposure of a move. (The number of displays showing intermediate positions during a move is governed by emulation commands.) Note that only the moving part of the model has been re-drawn. The user sees the lines being drawn on top of the original display. Before making the next move, we erase the screen. The next move is a guarded move down towards the part. We use emulation commands to associate functions for simulation of sensory events with real sensors. Specifically, during every sampling cycle we check whether the fingers of the manipulator intersect or contact the part. Figure 2(c) shows the resulting multiple exposure of the guarded move down. Note that the arm has stopped, "resting" on the part.

We now skip ahead in the program. Figure 2(d) shows the arm after it has been rotated and with the fingers touching the outside of the part. The next move would cause the part to be picked up. Since the system cannot determine whether the robot is actually grasping an object, an emulation command is used to assert this relationship between the robot and its physical environment. Figure 2(e) is a single display after a completed move of the grasped part. Figure 2(f) shows the arm after a move towards the center of the work station. The aim of the program now is to touch the part down gently on the table top. We achieve this by using the contact function again, with the gripper and the work table as operands. The previous emulation assertion command has established that the part is a sub-object of the gripper. Figure 2(g) is a multiple exposure of the incremental guarded move down. Figure 2(h) shows the final position.

The minimum modification of the code shown above for simulation of sensors would be the addition of an emulation command for associating a simulation function with a sensor and an assertion command representing the fact that the part has been grasped. Samples of such modifications of the pseudo-code given above are shown in Tables 1 and 2. Table 3 shows the original program modified by the addition of all the various commands required to produce the sequence in Fig. 2.

The next set of displays (Fig. 3) involves emulation of the first part of the same code fragment. However, this time we put some indeterminacy into our test by using a different function for simulation of sensory events. We assume that we do not know exactly how big the part is.

Figure 3(a) is a multiple exposure showing before and after a complete move of the manipulator arm to a position above the part. We now associate a randomizing function with a sensor. This function produces an event with probability linearly increasing over time. The exact linear function is governed by its arguments. Figure 3(b)

Table 1 Program with simulation of sensors by CONTACT.

```
attach (1,contact,finger1,parta)
attach (2,contact,finger2,parta)
  /* separate contact check for each finger and parta */

move (-2,10,8)      /* moves over and above part */
movedown (10) until sensors (1,2) exceed (10) /* guarded move */
open /* open fingers */
rotate (90)
movedown (1)
close until sensors (1,2) exceed (10) /* close fingers around part */

grasp (parta)

moveup (10)
move (2,-10, 0)

attach (3,contact,gripper,table)

movedown (10) until sensors (1,2,3) exceed (10) /* guarded
move */
```

Table 2 Program using RANDOM function.

```
attach (1,random,6, .1, .8)
  /* probabilistic test, starting with probability .1 and reaching
     .8 in 6 cycles */

move (-2,10,8)      /* moves over and above part */
movedown (10) until sensors (1,2) exceed (10) /* guarded move */
open /* open fingers */
rotate (90)
movedown (1)
close until sensors (1,2) exceed (10) /* close fingers around
part */

grasp(parta) at (1,0,0) from (finger1)
  /* the position of block adjusted; it will move with finger1 */
```

Table 3 Program with various emulation commands.

```
draw
  /* draw manipulator in current position */
setdisplaytk(5)
  /* set drawing to 1 time per 5 basic cycles
     incremental mode is default */
attach (1,contact,finger1,parta)
attach (2,contact,finger2,parta)
  /* separate contact check for each finger and parta */

move (-2,10,8)      /* moves over and above part */

erase
  /* erase display */
setdrawall(1)
  /* set display to redraw whole model each time */
seterase(1)
  /* set display to erase between drawings */

movedown (10) until sensors (1,2) exceed (10) /* guarded move */

erase
draw
setdisplay(0)
  /* set display off */

open /* open fingers */
rotate (90)
movedown (1)
close until sensors (1,2) exceed (10) /* close fingers around part */

grasp(parta)
  /* inform system that parta is to move with manipulator */
setdisplay(1)
  /* reset display to on */
setincrement(0)
  /* set for complete moves */

moveup (10)
move (2,-10, 0)

attach (3,contact,gripper,table)
setincrement(1)
  /* reset for incremental motion */

movedown (10) until sensors (1,2,3) exceed (10) /* guarded
move */

draw
```

shows a multiple exposure of the incremental guarded move. Note that motion stops after the third sub-move. In order to continue our testing, we again assert that the part is grasped, in this case using a re-adjust option as part of the GRASP command to place the part in the fingers. Figure 3(c) shows the grasped position. (It is also possible to access GDP commands for manipulating objects if a more elaborate re-adjustment is required.)

Another example of EMULA involves use of the emulation assertion command concerning gravitation and support. In Fig. 4, we see the robot dropping a part. The assertion to release this object told the emulation system to stop moving the part along with the robot fingers. If no other assertion is made to the emulation system, the part will remain in the same position. The SUPPORT assertion checks whether an indicated pair of objects, say the table and the part, are in contact. If they are not, the second object is translated according to the calculated effects of gravity in a specified direction. We emphasize here the assumption that support of one object by another is indicated by a contact check between the models.

Figures 2, 3 and 4 were obtained from interactive emulation sessions with a graphics terminal. EMULA can be used in a batch mode during which pictures can be constructed and saved or when watching the display is not so critical. Two features useful for the batch mode are collision checking and journaling. For example, a monitoring function can be used to check for collisions during

execution of a program. We note that this use of interference checking is independent of any use of CONTACT for simulation of sensors. In fact, this monitoring may indicate to the programmer where sensor-based moves should be inserted in the program. An emulation command establishes monitoring of collisions between specified pairs of elements. A collision check can be established for any number of pairs of objects. A journal file can be established by an emulation command: all commands entered by the user, along with any responses from the system, are recorded in the journal file. We give a sample journal file in Table 4. In the case shown here, the objects monitored for collision are the fingers of the robot. This batch mode of operation can be used to detect which points of a robot program require delicate maneuvering around parts. Testing of a robot program does not depend on the user looking at the graphics display and judging that an operation is safe.

Table 4 Sample journal file.

```
***JOURNAL OPENED AT 10:00:00:000***
setdisplay (0);
startupmotors;
open (1); /* open fingers */
setcrashchk (1, 'FINGER1', 'FINGER2');
close (0); /* close fingers */
**CRASH DETECTED* FINGER1 & FINGER2
**CRASH DETECTED* FINGER1 & FINGER2
setjournal (0);
***JOURNAL CLOSED AT 10:04:50:000***
```

Future work

The issue of variability and defined tolerances of parts offers several possibilities for future research in geometric modeling and emulation. For example, we would like some automatic or semi-automatic method of generating models of the same family. The use of a particular member of a family would be based on a trade-off involving the computer time required for the display, transformation, and monitoring of the model *versus* a measure of the closeness of its fit to the real robot. One choice for this measure of closeness could be volume; another possibility is maximum linear variance along each of three principal axes.

Similarly, the idea of probabilistic functions for simulation of sensory events, which is represented here by the RANDOM function, may be adapted to specifying models with variability. This would involve another extension of the modeling subsystem to handle objects with variable extents. We would maintain the dynamic nature of the association of sensors with simulation functions, such as a combination of CONTACT and RANDOM, but some information would be incorporated in the model.

Conclusion

EMULA offers the user of industrial robots a facility for developing and testing programs, since it accepts the same instruction set as an actual robot along with an emulation language for specifying such things as sensory input and gravitational effects. As part of the work, an extension was made to the existing modeling system to handle mechanisms: We establish a procedure to define what would be an acceptable geometric model for a family of robots and define the relationship of changes in motor values to movement of parts of acceptable models. Various features are included to make the system usable: for example, definition of a robot family, programmer control of rate and nature of displays, and recording of display and/or monitor information in a journal. Of course, EMULA cannot be used for complete testing of robot programs. In addition to the general problem of modeling tolerances of parts, fixtures, and the robot itself, the robot application domain contains indeterminacies caused by hysteresis, wear, noise, etc. Unlike resolution, these cannot be adequately modeled by a geometric modeling/emulation system. The EMULA system, however, does ease the difficult task of designing and programming a computer controlled mechanical manipulator to perform assembly and/or testing jobs in a "real-life" production environment.

Acknowledgments

The author wishes to thank Peter Will for original encouragement in the work, Frederick Ris for solution of a numerical problem in the GDP system, Kenneth Davies and Norman Brenner for continuous help in PL/I and system problems, and Anna Bruss, Phillip Summers, and David Grossman for reading many drafts of this paper.

References and note

1. C. Cunningham, "Robot Flexibility through Software," *Proceedings of the 9th International Symposium on Industrial Robots*, Society for Manufacturing Engineers, Dearborn, MI, March 1979.
2. R. Finkel, R. Taylor, R. Bolles, R. Paul, and J. Feldman, "AL, A Programming System for Automation," *Stanford Artificial Intelligence Laboratory Memo AIM-243*, Stanford University, Stanford, CA, November 1974.
3. L. I. Lieberman and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM J. Res. Develop.* **21**, 321–333 (1977).
4. W. T. Park and D. J. Burnett, "An Interactive Incremental Compiler for More Productive Programming of Computer-Controlled Industrial Robots and Flexible Automation," *Proceedings of the 9th International Symposium on Industrial Robots*, Society for Manufacturing Engineers, Dearborn, MI, March 1979.
5. D. D. Grossman, "Procedural Representation of Three-Dimensional Objects," *Research Report RC5314*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1975.

6. W. B. Heginbotham, M. Dooner, and K. Case, "Robot Application Simulation," *The Industrial Robot* **6** (1979).
7. T. Kuno, F. Matsumari, H. Moribe, and T. Ikeda, "Robot: A Performance Simulator," *Proceedings of the 9th International Symposium on Industrial Robots*, Society for Manufacturing Engineers, Dearborn, MI, March 1979.
8. R. Paul, "Modelling, Trajectory Calculation and Servoing of a Computer-Controlled Arm," *Stanford Artificial Intelligence Laboratory Memo AIM-177*, Stanford University, Stanford, CA, November 1972.
9. B. I. Soroka, "Debugging Robot Programs with a Simulator," presented at CADCAM-8 Conference, Society for Manufacturing Engineers, Dearborn, MI, November 1980.
10. There is a lack of consensus on the distinction between simulation and emulation. In our view, simulators are concerned with the internal mechanism of a system or phenomenon, whereas emulators try to match a specific, external interface. It is a significant feature of the EMULA system that the manipulator commands are the same as those of an actual system for running manipulators. However, no attempt is made to predict physical effects, such as gravity, grasping of a part by the robot, etc. The design and implementation of EMULA is concerned primarily with making the behavior of the robot controllable, reasonable, and understandable by the user. For these reasons, we chose to call this an emulation system rather than a simulation system.
11. L. I. Lieberman, M. A. Wesley, and M. A. Lavin, "A Geometric Modelling System for Automated Mechanical Assembly, *Research Report RC7089*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1978.

Received December 22, 1980; revised April 24, 1981

The author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.