An Integrated System for Dextrous Manipulation

Peter K. Allen
Paul Michelman
Kenneth S. Roberts

Department of Computer Science
Columbia University
New York, New York 10027

ABSTRACT

This paper describes an integrated system for dextrous manipulation using a Utah-MIT hand that allows one to look at the higher levels of control in a number of grasping and manipulation tasks. The system consists of a number of low-level system primitives for grasping, integrated hand and robotic arm movement, tactile sensors mounted on the fingertips, sensing primitives to utilize joint position, tendon force and tactile array feedback, and a high-level programming environment that allows task level scripts to be created for grasping and manipulation tasks. A number of grasping and manipulation tasks are described that have been implemented with this system.

1. INTRODUCTION

Great strides have been made in developing multi-fingered hands such as the Salisbury hand [9], the Utah-MIT hand [8], and the Belgrade hand [16]. The focus of the early work on these hands was at the lower levels of control. This is entirely reasonable since a bottom up effort to build a usable robotic hand must necessarily precede higher level uses of the hand. But the state of affairs has changed somewhat; the hands mentioned above have made their way into labs and researchers are starting to exploit their potential. This paper details our own experience in building an integrated system for dextrous manipulation using a Utah-MIT hand that allows one to look at the higher levels of control in a number of grasping and manipulation tasks.

We wish to acknowledge the work of others who have attempted to build *systems* to perform manipulation and grasping tasks. Among these are Geschke's early system to perform integrated robotic manipulation tasks [5], the work of Takase et al [14] in building an integrated robotic teaching and learning system, Salisbury's integrated hand/tactile system [13], Fearing's work with a tactile sensor mounted on a dextrous hand [3], the work of the group at USC integrating the Belgrade hand into an active sensing environment[16], and the original Utah-MIT hand researchers [12] who developed a low-level control system for the hand and a software environment to utilize the low-level control functions.

The system we have built contains the following components:

- A set of low-level system primitives that serve as the basis for the control of the hand.
- A true hand/arm system with many degrees of freedom formed by mounting the hand on a robotic manipulator (a PUMA 560).
- Integrated tactile sensors on the fingertips that provide force-sensitive responses and Cartesian position information.
- Sensing primitives that make use of joint position, tendon force, and tactile array sensing in a number of grasping and manipulation tasks.
- A high-level programming front end that allows task-level scripts for common grasping and manipulation tasks to be written easily and cogently, allowing the natural concurrency of such a hand/arm system to be captured at the programming level.

In building this system, we have been surprised by the ease with which a number of reasonably complex grasping and manipulation tasks (described below) have been implemented. We are encouraged by this result to believe that integrating a hand into an existing robotics environment can be done simply and robustly once the right set of primitives and structures are developed. The remainder of this paper is a description of each of the modules in our system and a description of some tasks that have been performed by the system to date. In the conclusion, we offer some of our ideas on future directions of this research, including adding new sensors to this environment.

Figure 1: Hand/Arm system.

2. SYSTEM OVERVIEW

The system we have built consists of a Utah-MIT hand attached to a PUMA 560 manipulator. The hand contains four fingers, each with four degrees of freedom. It resembles the human hand in size and shape, but lacks a number of features that humans find very useful. In particular, it has no palmar degree of freedom (closing of the palm) and the thumb is placed directly opposite the other three fingers, with all fingers identical in size (see figure 1). The hand has joint position sensors that yield joint angle data and tendon force sensors that measure forces on each of the two tendons (extensor and flexor) that control a joint. The PUMA adds 6 degrees of freedom to the system (3 translation parameters to move the hand in space and 3 rotational parameters to orient the hand), yielding a 22 degree of freedom system. Clearly, such a system is a nightmare to control at the servo-level in real-time. Our approach is to use the embedded controllers in each of these systems, controlling and communicating with them through an intelligent, high-level controller that links together the movements of arm, hand and fingers with the feedback sensing of joint positions, tendon forces, and tactile responses on the fingers.

The hardware structure of the system is shown in Figure 2. The high-level control resides in a SUN-3 processor. The SUN serves as the central controller, and has access to a full UNIX based system for program development and debugging as well as a set of window-based utilities to allow graphical output and display of the system's various states. The hand is controlled by an analog controller that is commanded through D/A boards from a dedicated 68020 system. The SUN is capable of downloading and executing code on the 68020 and can communicate with it through a shared memory interface [11]. The tactile sensing system is controlled by another dedicated 68020 that monitors the forces on each of the sensor pads. The connection from the SUN to the PUMA is via the VAL-II host control option over a serial interface.

The software structure is shown in Figure 3. The high-level control program is called DIAL and was originally developed by Steven Feiner[4] for use as a graphical animation language. DIAL is implemented on the SUN-3 and is able to communicate via system level primitives to the hand system over the shared VME bus and via a serial line to the PUMA manipulator. The following sections describe the system in detail.

2.1. Low-Level System Primitives

The low-level system primitives described in Table 1 are organized along two dimensions, *type* and *domain*. *Type* refers to the type of sensing or actuation (or both) which the primitive implements. Continuous sensing implies a monitoring mode while one shot sensing implies a static sensor reading. Continuous action implies a synchronized control loop while one shot action implies an imperative command. *Domain* refers to the coordinate frame or sensor domain that the primitive operates in. The primitives are:

- GET JOINTS: Reads the joint angles on the hand.
- GET FORCES: Reads the forces (measured at the wrist of the hand) on the flexor and extensor tendons that control each joint.
- GET TACTILE: Reads a 16x16 tactile array on each finger.
- GET WRIST: Reads the Cartesian position of the PUMA wrist.

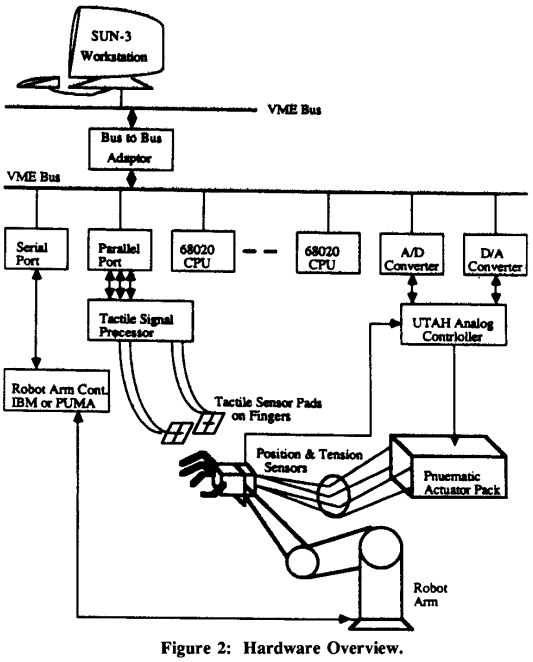

Figure 2: Hardware Overview.

- GOTO MOVE: A one shot move that is done atomically. Implemented by setting the desired joint position in the analog control system directly to the final position value of the move for each joint. Move commands have the ability to use symbolic names for specified poses. There is a set of standard pose names available for use by the higher level programming system, particularly for hand pre-shaping operations.
- TRAJECTORY MOVE: Allows a single joint or a number of joints to be moved along an interpolated trajectory from a starting joint space vector to an ending joint space vector. There is also an analogous command that interpolates in Cartesian space for fingertip motion.
- POSITION CONTACT: This is a continuous sensing primitive that is implemented in the Cartesian position domain. When the difference between actual and desired Cartesian fingertip position exceeds a threshold, contact is signaled.
- FORCE CONTACT: Same as above but tendon forces are monitored for changes. A change in force differentials implies a finger contact.
- TACTILE CONTACT: The tactile arrays are monitored for readings above a prespecified threshold.

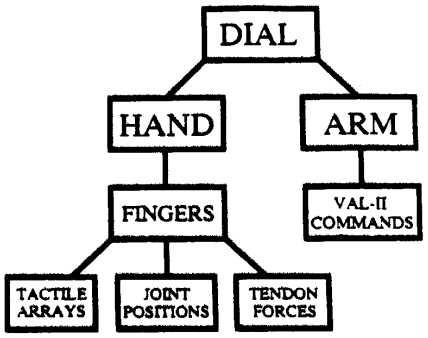

Figure 3: Software Overview.

LOW-LEVEL SYSTEM PRIMITIVES		
NAME	TYPE	DOMAIN
GET JOINTS	One Shot Sensing	Joint Position
GET FORCES	One Shot Sensing	Tendon Force
GET TACTILE	One Shot Sensing	Tactile Array
GET WRIST	One Shot Sensing	Cartesian Position
GOTO MOVE	One Shot Action	Joint Position Cartesian Position
TRAJECTORY MOVE	Continuous Action	Joint Position Cartesian Position
POSITION CONTACT	Continuous Sensing	Cartesian Position
FORCE CONTACT	Continuous Sensing	Tendon Force
TACTILE CONTACT	Continuous Sensing	Tactile Array

Table 1: Low-Level System Primitives

2.2. Composite Functions

Table 2 lists a number of composite functions that have been built out of the low-level system primitives. Each composite function is described below:

COMPOSITE FUNCTIONS		
NAME	TYPE	DOMAIN
GRASP WITH FORCE	Continuous Sensing, Continuous Action	Tendon Force Joint Position
GUARDED MOVE	Continuous Sensing, Continuous Action	Tendon Force Cartesian Position Tactile Array
LIMP	Continuous Sensing, Continuous Action	Joint Position

Table 2: Composite Functions

- GRASP WITH FORCE: Used to grasp objects with a desired grip strength. The command closes all joints specified by a joint mask, incrementally while monitoring the tendon forces controlling the joints in the tendon mask. When the difference between the flexor and extensor tendon forces on a joint exceeds the specified threshold, movement of that joint stops. This process continues until the forces on all of the tendons in the tendon mask have surpassed the threshold. This primitive is useful for grasping objects when their precise dimensions are unknown.
- GUARDED MOVE: This composite function combines one or more of the three low-level contact primitives (POSITION CONTACT, FORCE CONTACT, or TACTILE CONTACT) with either finger motion or motion of the PUMA arm. When a contact is detected, the relevant motion ceases.
- LIMP: This primitive is very useful in establishing grasps. It allows a human to interact with the hand and position it manually. It is implemented by comparing the actual joint positions with the previous joint positions, and updating the current desired position to reflect the new actual joint positions. For example, a hand can be placed on an object in a known position and the position can then be recorded for future use. It has a feature to allow the masking out of joints that need to be fixed (not LIMP) and those which should be under LIMP control.

2.3. Tactile Primitives

The tactile sensors we are using [7, 15] consist of 16 x 16 grids of piezo-resistive polymeric material that are conformable to the finger's shape. They are manufactured by sandwiching the polymeric material between two pliable sheets of Kapton that contain electrical etching. The application of forces on the pads provides an increased electrical flow channel between the two sheets as the material within is compressed. Results with this sensor have been good, particularly with respect to signal isolation. The sensors are monitored by a separate 68020 that is responsible for low-level tactile processing, including A/D conversion, and thresholding and normalization of signals. Some of the low-level tactile primitives that have been implemented are:

- TACTILE FILTERS: A number of useful digital filters have been implemented including averaging and median filters which are very useful in processing noisy tactile data [10].
- TACTILE MOMENTS: A useful technique for quickly getting contact information is central moment analysis [6]. The contact area and centroid of the contact can be determined using moments. The second moments are useful for determining the eccentricity of the contact region and the principal axes of the contact.
- EDGE DETECTION: A number of edge detectors have been developed and used for feature extraction from tactile images.
- LINE DETECTION: Lines are detected by using the output of the edge detection procedure in a Hough transform [2]. We have successfully analyzed tactile image data and obtained the equations of straight lines in a single impression [1].

2.4. PUMA Arm Primitives

The arm primitives are already embedded in VAL-II, the programming environment of the PUMA. They include movement primitives, asynchronous interrupt capability, and the ability to establish arbitrary coordinate frames such as the hand coordinate system. It also provides a global coordinate system in which the tasks can take place.

3. DIAL: A HIGH-LEVEL CONTROL SYSTEM

DIAL is a diagrammatic language that allows parallel processes to be represented in a compact graphical "time line". It has been used for animation of graphical displays but it has been transported by us to the robotics domain so we may exploit its ability to express parallel operation of robotic devices. It also provides a convenient way to implement task-level scripts which can then be bound to particular sensors, actuators and methods for accomplishing a generic grasping or manipulation task. The instruction set that DIAL supports is specified at run-time in a user-supplied backend. We have created a DIAL backend to work in our environment which has the need to express parallel actions in the performance of simple grasping and manipulation tasks. Given that we need to control the hand and arm movement concurrently, as well as integrating information from three different sensor systems (joint position, tendon force, tactile array), we found it natural to program tasks in DIAL scripts.

```
/* Curl a finger, then extend it, then curl it again
        more slowly than the first time */

/* DEFINITION LINES */
% curl          move_finger     1 0 2000 2000 2000
% extend        move_finger     1 0 -2000 -2000 -2000

/* EXECUTION LINES */
/*      12345678901234567890123.... */
curl    #=====    #==========
extend        #===
```

Figure 4: A simple DIAL script to move fingers.

3.1. DIAL Scripts

Figure 4 shows a simple DIAL script to move a finger of the hand. The horizontal dimension of DIAL's 2-D language represents time, with each column corresponding to what is called a *tick*. Each event starts at a particular tick and lasts for some integral number of ticks. DIAL handles all flow of control by determining which events are to be executed during each tick. Entries in each column of execution lines (described below) specify the events that are to be performed during that column's tick. The first lines are *comment lines*, and they have the same format as comments in C.

3.2. Definition Lines

Following the comments are two *definition lines* for the events named *curl* and *extend* that perform a finger movement instruction, *move_finger*, when the event is executed. Definition lines begin with a "%" in the first column, followed by an event name, an instruction name, and any parameters. Note that this line defines the event, but does not cause it to be executed. The *move_finger* instruction takes 5 parameters. The first is the finger number (0 for thumb, 1 for index finger, and so on). The last four parameters correspond to the finger's joint angles at the completion of the event.

3.3. Execution Lines

The last two lines in the example are *execution lines* that specify a sequence of successively executed events. Execution lines begin with an event name. Each column in an execution line corresponds to a tick during which an event might be executed. A time line is shown in a comment above the execution lines to aid in discussing the timing of operations.

In the example, the appearance of the *execution character* "#" in the first column causes *curl* to begin execution during the first tick. The "#" character is followed by a "=". This is a *continuation character*. The appearance of a continuation character after the execution character means that the event's execution is to be extended into the next tick. The first occurrence of *curl* is extended into the sixth tick. In other words, by the sixth tick, the index finger will have moved from its initial position to a completed curled position, and the duration of movement will have been six ticks. During the seventh tick, the event *extend* causes the finger to begin straightening. This operation is completed during the tenth tick. In the 11th tick, the finger starts to curl again, only this time, the curling takes ten ticks to be completed and the script will be completed in the 20th tick. The speed of execution of a script, that is, the length of each tick, can be modified via a system parameter. This enables tasks to be sped up or slowed down with a single parameter.

```
/* Script showing parallelism and guarded move:
        Two fingers are moved.
        If a finger senses tactile contact,
           it halts.                          */

/* DEFINITION LINES */
% stop0          guarded_tact     0 0 2000 2000 2000
% stop1          guarded_tact     1 0 2000 2000 2000


/* EXECUTION LINES */
/*           12345678901234567890...      */
stop0       #=========
stop1            #=========
```

Figure 5: DIAL script for guarded finger moves.

3.4. Expressing Parallelism

The events in the script of Figure 4 are executed sequentially. Figure 5 shows how parallelism is notated. The task to be executed is a guarded move with tactile sensing: two fingers are moved simultaneously, and each stops if its tactile sensors detect contact. The events are named *stop0* and *stop1*, and the guarded move with tactile sensing instruction is called *guarded_tact*. The parameters in the definition lines have the same meaning as those in the previous script, that is, move finger N to the location defined in the last four parameter values. The difference between the *move_finger* and *guarded_tact* instructions is that with the guarded move, the finger stops its motion when contact is sensed -- whether it has reached the final location or not. If no contact is sensed, then the finger reaches its final location, and it uses the number of ticks defined in the execution lines to do so. In the example, the thumb begins its movement (event *stop0*) at tick 1. While the thumb is moving, the index finger also begins to move (event *stop1*, beginning at tick 5). The script shows that both events also terminate at different times. If only one of the fingers makes contact, then the other finger continues in its motion until it reaches its destination.

4. TASK 1: PICK AND PLACE

Figure 6 shows an example of a DIAL script that performs a common grasping task: picking up an object with a three-fingered grasp and putting it down in a different location. This script illustrates the use of several continuous sensing, continuous action primitives, as well as coordinated use of the hand and the arm. The events *preshape* and *to_object* are executed in parallel in ticks 1-8, preshaping the hand for a three-fingered grasp and simultaneously moving the arm to a predetermined location. The preshaping is concluded at tick 8 while the arm motion continues until tick 14. After the hand reaches the object in tick 14 (the object's position is assumed known in this script), the event *grasp* executes the composite function *grasp with force*. *Grasp with force* instructs the hand to grasp an object and to terminate when the tendon forces exceed a certain threshold. The parameters in the definition line specify which joints to move, which tendons to monitor, and what the value of the force is. The instruction differs from the previously discussed instructions in that there is no way to determine ahead of time how long its execution will take. Therefore, while it is shown as a single-tick operation in the script, the grasping operation continues until the parameter values specified in the definition line are satisfied. In this way, the essential non-determinism of a robotic task can be captured in DIAL. Commands of indeterminate length are implemented as single tick events, but the completion of the event actually determines the tick's length.

```
/* Pick and place script
        (1) Move hand/arm to a predefined location.
        (2) Grasp an object using tendon force control
              (grasp with force primitive)
        (3) Pick up hand/arm, move object to new location.
        (4) Use PUMA move-until-touch primitive with ten-
             don force sensing to locate table height.
        (5) Put down the object.                    */

/* DEFINITION LINES */
% preshape       hand_pose        "3_fing_preshape"
% open           hand_pose        "open_hand"
% grasp          grasp_force      0x0662 0x0222 400
% to_object      puma_command     "ex move_to_object"
% relocate       puma_command     "ex relocate_object"
% retract_arm    move_arm         "final_arm_pos"
% findtable      puma_mut_force    /* move puma until hand's
                                      tendon force sensors
                                      detect table */

/* EXECUTION LINES */
/*                  1         2         3         4
/*          12345678901234567890123456789012345678901.. */
preshape    #=======
to_object   #==========
grasp                #
relocate              #=========
findtable                      #
open                            #=======
retract_arm                                  #=========
```

Figure 6: Pick and Place Script.

After the hand has formed a grasp of desired strength around the object, the hand/arm picks up the object and moves it to a new location above the table (the event *relocate* that begins in tick 16 and ends at tick 25). The exact position above the table is not known, only that the object is above the table. The next event, *findtable*, is used to move the arm toward the table and to stop when the *hand* detects contact with the table. Like the event *grasp*, the event is shown as a single tick because it is not know how long it will take to make contact. When the tendon force sensors determine that the object has contacted the table, the hand releases the object (the event *open*, beginning at tick 27), and then retracts the arm (the event *retract_arm* in ticks 36-45). Thus, we can embed sensory feedback in DIAL scripts.

5. TASK 2: POURING FROM A PITCHER

Another example script is to have our hand/arm system pick the top off of a pitcher on a table, place the top on the table, pick up the pitcher and pour the liquid out, replacing the pitcher on the table. This particular task is implemented with joint space position control, joint space force control, and Cartesian space arm control. The DIAL script for such a task is shown in figure 7. The primitives used here are the same as those introduced in the previous scripts. During ticks 1-13, the arm and hand are moved toward the pitcher (event *get_top*), and the hand is preshaped to grasp the top (event *pittop*). In tick 14, the event *grab_top* is used to grasp the pitcher top securely. During ticks 15-24, the hand/arm lifts the top off the pitcher and moves the top to a position at which it will release the top (event *lift_top*, Figure 1). Starting at tick 25, the event *pittop* (which uses the low-level system primitive *hand_pose*) is used to move the hand back to its preshape position. This movement has the effect of releasing the top, because in the preshaping pose the hand is not in contact with the top. From tick 29 to 35, the arm moves to a position to grasp the pitcher (event *to_grasp*), simultaneously preshapes the hand to grasp the pitcher (event *prepit*). The event *grabpit* (tick 36) grasps the pitcher; event *pourpit* (ticks 37-44) lifts the

pitcher, turns it to pour the contents, and then replaces the pitcher in its original position. Event *prepit* is used to release the pitcher and the PUMA movement event *retract_arm* removes the hand/arm from the pitcher.

```
/* Pitcher script:
        (1)  Move arm/hand to above pitcher,grasp the top
        (2)  Lift top, move to new location,put it on table.
        (3)  Preshape hand to lift pitcher using pitcher's
                handle, move the arm/hand into the position.
        (4)  Grasp the pitcher, lift it,pour out the contents.
        (5)  Replace the pitcher to its original location.

/* DEFINITION LINES */
% prepit        hand_pose       "open_hand"
% pittop        hand_pose       "pitcher_top"  /* preshape */
% get_top       puma_command    "ex get_top"   /* arm to pitcher */
% lift_top      puma_command    "ex lift_top"  /* lift pitcher */
% to_grasp      puma_command    "ex to_grasp"
% pour_pit      puma_command    "ex pour_pit"
% retract_arm   puma_command    "ex retract_arm"
% grabtop       get_force       0x6666 0x0222 5
% grabpit       get_force       0xeeee 0x0222 2

/* EXECUTION LINES */
/*                   1         2         3         4         5
/*          123456789012345678901234567890123456789012345678*/
pittop      #------------            #---
get_top     #------------
grabtop                  #
lift_top                  #---------
to_grasp                                 #------
prepit                                   #------        #---
grabpit                                         #
pour_pit                                         #-------
retract_arm                                                  #-----
```

Figure 7: DIAL script for pouring pitcher task.

6. TASK 3: REMOVING A LIGHT BULB

Figure 8 is a last example of a DIAL script -- removing a light bulb from a socket (see figure 9). This is a task that requires tendon force feedback to determine the grasp strength and coordination of the fingers moving in parallel to unscrew the bulb. The script performs the task by moving the arm to the lightbulb while preshaping the hand, and then repeating the following sequence of events: grasp the lightbulb, turn the hand/arm counterclockwise, release the grasp, move back to the initial preshaped position by rotating the hand and arm clockwise. After the bulb becomes loose, the hand grasps the bulb and retracts it from the socket.

In the script, movement of the arm to the bulb occurs during ticks 1-10. At tick 6, the preshaping of the hand is begun and will execute in parallel with the arm motion, resulting in the arm reaching its destination above the bulb in tick 10 with the hand preshaped. The next set of event lines represents the repeated sequence of grasps and movements that unscrew the bulb from the socket. The event *grasp_bulb* in tick 11 causes the hand to develop a secure grasp around the bulb. The next events, *twist_hand* and *twist_armCCW*, rotate the fingers of the hand and the arm wrist, respectively, in a counter-clockwise motion (ticks 12-20). The hand is moved to the position *ungrasp* (ticks 21-23) to release the grasp of the hand, and then the hand and arm are returned to their initial positions by the events *preshape* and *twist_armCW* in ticks 24-30. This sequence must be repeated a number of times before the light bulb is unscrewed, as shown. At the completion of the script, the event *retract_arm* is executed to remove the lightbulb from its socket and lift it in air (notice that the hand is still grasping the bulb).

7. SUMMARY

The system we have built is an important step in building an integrated multi-sensor robotic perception system. Robotic systems need to have the ability to process sensor data from a number of sources and to be programmed in a simple and natural way to accomplish grasping and manipulation tasks. Our system has allowed us to build a number of task level scripts that embody parallel actuation of devices (hands, fingers, arm, wrist) with sensory feedback from a number of different sources (joint positions, tendon forces, tactile arrays). We are currently

```
/* Script for removing a lightbulb
   (1)  Move arm/hand to above lightbulb and preshape

   (2)  Repeat the following sequence until bulb is
        unscrewed:
          - Grasp the lightbulb tightly
          - Twist the fingers and joint 6 of PUMA
                counterclockwise
          - Release the grip on the lightbulb
          - Twist fingers and joint 6 of PUMA clockwise
                back to position at start of (2)

   (3)  Grasp lightbulb and retract arm to lift bulb out
        of socket.                 */

/* DEFINITION LINES */
% preshape      hand_pose       "prelight"
% to_bulb       puma_command    "ex to_bulb"
% grasp_bulb    grasp_force     0x0666 0x0222 400
% twist_hand    hand_pose       "unscrew"
% ungrasp       hand_pose       "ungrasp"
% twist_armCCW  puma_command    "ex twist_wrist"
% twist_armCW   puma_command    "ex twist_CW"
% retract_arm   puma_command    "ex retract_arm"


/* EXECUTION LINES */
/*                  1   */
/*         1234567890   */
to_bulb    #---------
preshape        #----

/* REPEAT THE FOLLOWING SECTION DESIRED
        NUMBER OF TIMES */
/*                1         2         3  */
/*                1234567890123456789012  */
grasp_bulb      #
twist_hand       #--------
twist_armCCW     #--------
twist_armCW                  #------
ungrasp                   #--
preshape                     #------


/* THE LAST ITERATION, PERFORM THE FOLLOWING
        SEQUENCE TO REMOVE
        THE BULB FROM THE SOCKET */
grasp_bulb      #
twist_hand       #--------
twist_armCCW     #--------
retract_arm               #---------
```

Figure 8: DIAL script for removing light bulb task.

adding vision sensors to this system, and these sensors will be integrated through the same set of software as the hand primitives.

The high-level control of the system through DIAL needs to be further modified to extend its capabilities in the robotics domain. One extension is to include the ability to dynamically alter script sequences. Currently, DIAL must continue on a single script thread. For example, in haptic exploration tasks, the feedback from the hand/arm system is used to generate hypotheses about an object's shape, which can drive new rounds of sensory exploration. This can be implemented by having multiple DIAL scripts available, and dynamically executing a particular script sequence based upon the outcome of the previous exploration.

8. ACKNOWLEDGEMENTS

This work was supported in part by DARPA contract N00039-84-C-0165, NSF grants DMC-86-05065, DCI-86-08845, CCR-86-12709, IRI-86-57151, North American Philips Laboratories, and the AT&T Foundation. Special thanks to Steve Feiner and Cliff Beshers for bringing DIAL to our attention and helping us port it to our environment.

Figure 9: Hand removing light bulb from socket.

References

1. Allen, Peter and Kenneth S. Roberts, "Haptic object recognition using a multi-fingered dextrous hand," *IEEE Conference on Robotics and Automation*, Scottsdale, AZ, May 1989.
2. Ballard, D. and C. Brown, *Computer vision*, Prentice-Hall, 1982.
3. Fearing, Ronald, "Simplified grasping and manipulation with dextrous robot hands," *IEEE Journal of Robotics and Automation*, vol. RA-2(4), pp. 188-195.
4. Feiner, Steven, David Salesin, and Thomas Banchoff, "DIAL: A diagrammatic animation language," *IEEE Computer Graphics and Animation*, vol. 2, no. 7, pp. 43-54, September 1982.
5. Geschke, Clifford C., "A system for programming and controlling sensor-based robot manipulators," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 1, pp. 1-7, January 1983.
6. Horn, B. K. P., *Robot vision*, M.I.T. Press, 1986.
7. Interlink,, *The force sensing resistor: A new tool in sensor technology*, 535 E. Montecito, Santa Barbara, CA, June 25, 1986.
8. Jacobsen, S. C., E. K. Iversen, D. F. Knutti, R. T. Johnson, and K. B. Biggers, "Design of the Utah/MIT dextrous hand," *Proceedings of the IEEE Conference on Robotics and Automation*, pp. 1520-1532, San Francisco, April 7-10, 1986.
9. Mason, Matthew T. and J. Kenneth Salisbury, *Robot hands and the mechanics of manipulation*, M. I. T. Press, Cambridge, 1985.
10. Muthukrishnan, C., D. Smith, D. Myers, J. Rebman, and A. Koivo, "Edge detection in tactile images," *Proc. of IEEE International Conference on Robotics and Automation*, pp. 1500-1505, Raleigh, 1987.
11. Narasimhan, Sundar, David M. Siegel, and John M. Hollerbach, "Condor: A revised architecture for controlling the Utah-MIT hand," *IEEE Conference on Robotics and Automation*, pp. 446-449, Philadelphia, April 24-29, 1988.
12. Narasimhan, Sundar, David M. Siegel, John M. Hollerbach, Klaus Biggers, and George E. Gerpheide, "Implementation of control methodologies on the computational architecture for the Utah/MIT hand," *Proc. IEEE International Conference on Robotics and Automation*, pp. 1884-1889, San Francisco, April 7-10, 1986.
13. Salisbury, Keneth, David Brock, and Stephen Chu, "Integrated language, sensing and control for a robot hand," *International Symposium on Robotics Research*, Gouvieax, France, October 1985.
14. Takase, K., R. Paul, and E. Berg, "A structured approach to robot programming and teaching," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-11, no. 4, pp. 274-289.
15. Tise, B., "A compact high resolution piezo-resistive digital tactile sensor," *IEEE Conference on Robotics and Automation*, pp. 760-764, Philadelphia, April 24-29, 1988.
16. Tomovic, Rajko, George Bekey, and Walter Karplus, "A strategy for grasp synthesis with multi-fingered robot hands," *Proc. IEEE International Conference on Robotics And Automation*, pp. 83-89, Raleigh, N. C., March 31-April 3 1987.