LANGLEY GRANT
IN-63-CR
93195
P. 10

A Software Toolbox For Robotics

Final Report

October 14, 1985

NASA GRANT NAG-1-533

J. C. SANWAL

Principal Investigator

Department of Mathematics

The College of William and Mary

Williamsburg, Virginia 23185

Project Monitor

Mr. F. Wallace Harrison, Jr.

Mail Stop 152D

NASA Langley Research Center

Hampton, Virginia 23665

Abstract

We give a method for programming cooperating manipulators, which is guided by a geometric description of the task to be carried out. For this we must have a suitable language and a method for describing the workplace and the objects in it in geometric terms. A task level command language and its implementation for concurrently driven multiple robot arms is described. The language is suitable for driving a cell in which manipulators, end effectors and sensors are controlled by their own dedicated processors and these processors can communicate with each other through a communication network. A mechanism for keeping track of the history of the commands already executed allows the command language for the manipulators to be event driven. A frame based world modeling system is utilised to describe objects in the work environment and any relationships that hold between these objects. This system provides a versatile tool for managing information about the world model. Default actions normally needed are invoked when the data base is updated or accessed. Most of the first level error recovery is also invoked by the database by utilising the concepts of demons. The package can be utilised to generate task level commands in a problem solver or a planner.

1.0 SUMMARY OF RESULTS OBTAINED:

1. I examined the existing Puma interface and found that it will have to be modified so that it can be used in a cell with multiple arms.

2. Therefore I implemented a simulator for a system with multiple arms and changeble end effectors for each arm. A list is maintained of all available resources for the arm. Dry runs can be made for a task to see whether there will be any special obstacle in carrying it out.

3. Implemented a scheme which keeps the history of all the commands given to any unit of the system. The history is kept in two places in different forms. It is stored in a disc file where it can be examined later for debugging and optimization. It is also kept in memory as a stack to be used by the program for control of the system.

4. I implemented a data base using the concept of frames.

5. This data base is interfaced to the system and is used in writing task level commands. When the task level commands are issued the world model is automatically updated and a snap shot can be taken.

6. A function can be defined to determine the working envelope of the Puma arm. This function can be called to determine if a point lies outside the reach of the arm. If the end effector is commanded to move to a point outside the envelope of the manipulator the program can detect it and thus can take corrective action without issuing the command.

In addition the effective envelope can be modified, by changing some constants in the function, and thus it can be used to give to the arm some collision avoidance capability with itself and other fixed obstacles such as the waist column of the arm, the base table etc.

2.0 INTRODUCTION

In designing a controller for a manipulator the manufacturer uses diverse schemes to control the motion of the manipulator arm. These can be thought of as machine instructions for the manipulator. We give a standard instruction set which can be constructed from the manufacturer's set. Thus different manipulators can be integrated in a system in a uniform manner. This set is adequate to implement task level conditional commands for a manipulator when we interface them with a data base.

3.0 STANDARD SET

For a manipulator we can implement the following basic commands ([4] Orlando):

```
(INITIALIZE ARMNUMBER)                             --- (1)
```

which puts the indicated manipulator arm in a predetermined state.

```
(MOVE       ARMNUMBER XYZOAT-LIST)                 --- (2)
```

moves the origin of the orthogonal coordinate frame associated with the end effector of the arm to the location with cartesian coordinates (x, y, z) and orients this frame with respect to a fixed reference coordinate frame with angular coordinates (phi1, phi2, phi3). In other words, XYZOAT-LIST is a point in the semidirect product of E(3) (the 3-dimensional euclidean space) and SO(3) (the group of rigid motions of E(3)). The end effector is not forced to follow any predetermined path.

```
(MOVE-ALONG ARMNUMBER DIRECTION-DISTANCE-LIST)  --- (3)
```

moves the origin of the end effector for a distance d along a vector (x, y, z) from its present position. The orientation of the end effector is not modified by this instruction.

(SET-ANGLES ARMNUMBER ANGLE-LIST) --- (4)

sets the links of the manipulator at the ANGLE-LIST.

(POSITION ARMNUMBER) --- (5)

returns the current XYZOAT-LIST coordinates of the end effector.

(STATUS ARMNUMBER IDENTIFICATION-NUMBER) --- (6)

can be used to find out whether the command with the given identification number has been completed or is still pending.

(RESUME ARMNUMBER) --- (7)

enables the arm and

(SUSPEND ARMNUMBER) --- (8)

disables the arm. The last two commands cancel each other out.

(CANCEL ARMNUMBER) --- (9)

deletes all commands to the arm that are on the command queue.

4.0 HISTORY MANAGEMENT

If the receiving unit is enabled it puts on its command queue commands of type (1)-(4) and carries them out sequentially. Whenever a command has been completed the central processing unit moves it to its history stack along with the identification number given to it by the issuing unit. Notice that the issuer's event count is used even though the device may maintain its own time stamp. Normally the commanding unit does not wait after giving these commands although the receiving unit does issue a completion report when it transfers the command to its history stack. If the status command is received the central processing unit searches this stack to give its response. Commands of the type (5)-(9) are executed immediately even if it is command is stacked in front of the command queue. For commands of type (5)-(9) the commanding unit waits for the response.

In a work cell system, as shown in figure 1, the connecting lines indicate the communication path between the central processing units.

5.0 DATA BASE FOR WORLD MODELLING

To be able to implement high level commands for the manipulators we have implemented a data base to represent the working environment using frames. In the implementation all the default actions normally needed are invoked when the data base is updated or accessed. Frames ([2] Minsky) are structures to make declarative statments about objects and relationships between objects. We selected a frame based data base because of the ease with which the working environment can be described in geometric terms. Thus two typical frames in this world model which describes an arm and an object of the system would be:

```
(arml (type    (name   (puma)))
      (base    (radius (150.0)))
      (init    (value  ((1064.4 681.461 1092.4 0.0 0.0 0.0))))
      (located (value  ((1064.4 681.461 0.0 0.0 0.0 0.0))))
      (using   (hand   (handl))))
(a    (type    (value  (cube)))
      (height  (value  (35.0)))
      (base    (radius (24.75)))
      (on      (object (p)))
      (located (value  ((1480.0 895.0 35.0 3.14 0.0 0.0)))))
```

6.0 CONCURRENT PROGRAMMING LANGUAGE

Using the above history management scheme we can now implement a language to program a multiple arm cell. Thus in the command

(COND-PUT ARM OBJECT LOCATION SET-COND TEST-COND) --- (10)

all the information about the OBJECT is stored in a frame and suitable error recovery functions are invoked when the MOVE and PICK subcommands of the COND-PUT are issued. The COND-PUT waits for TEST-COND to be true before it starts executing and when completed sets the SET-COND to true. An interpreter for the language is implemented using the communication network, the time stamp and the history tables described above.

To illustrate some of the commands and how they can be used to describe a task in the language we give a program to build a seesaw of figure 2.

```
(CMOVE    ARM1 HAND1  BAR A      ON  C10 PV1)
(CPICK    ARM2 HAND2  A2  C21    PV2 )
;pv1, pv2 are condition codes returned by the previous task
;C10 will be set to T when CMOVE has placed the BAR on the block A
;and C21 will be set to T when CPICK has picked up the object A2
(CPICK    ARM1 HAND1  A1  C11    C10 )
(CMOVETO  ARM2 BAR-RT C22 C21)
(CMOVETO  ARM1 BAR-LT C12 C11)
(CRELEASE ARM2 HAND2  C23 (AND C12 C22))
;(AND C12 C22) guarantees that both objects A1 and A2 are immediately
;above the bar so that if one cube is placed before the other the bar
;would not fall
(CRELEASE ARM1 HAND1  C13 C12)
```

7.0 REFERENCES

[1] Brady, Michael, et al., [1982], Robot Motion: Planning and Control. MIT Press, Cambridge.

[2] Hopcroft, John, [1983], Robotics - A New Direction in Theoretical Computer Science. Third International Conference on the Foundation of Software and Theoretical Computer Science, Banglore.

[3] Lozano-Perez, Tomas, [1983], Robot Programming, Proceeding of the IEEE, vol 7.

[4] Minsky, M, [1975], A Framework for Representing Knowledge. The Psychology of Computer Vision (P. Winston, Ed.), McGraw Hill, New York.

[5] Orlando, Nancy E., [1984], An Intelligent Robotics Control Scheme. American Control Conference, San Diego, California.

[6] Orlando, Nancy E., [1985], Device Command Sets, ISRL internal memo. Langely Research Center, Hampton, Virginia.

[7] Sanwal, J. C., [1984], Commands for Cooperating Robot Arms. NASA report number 172409, Langley Research Center, Hampton, Virginia.

[8] Sanwal, J. C., [1985], Programming of Robot Arms by Geometric Description. NASA/ASEE report, Langley Research Center, Hampton, Virginia.

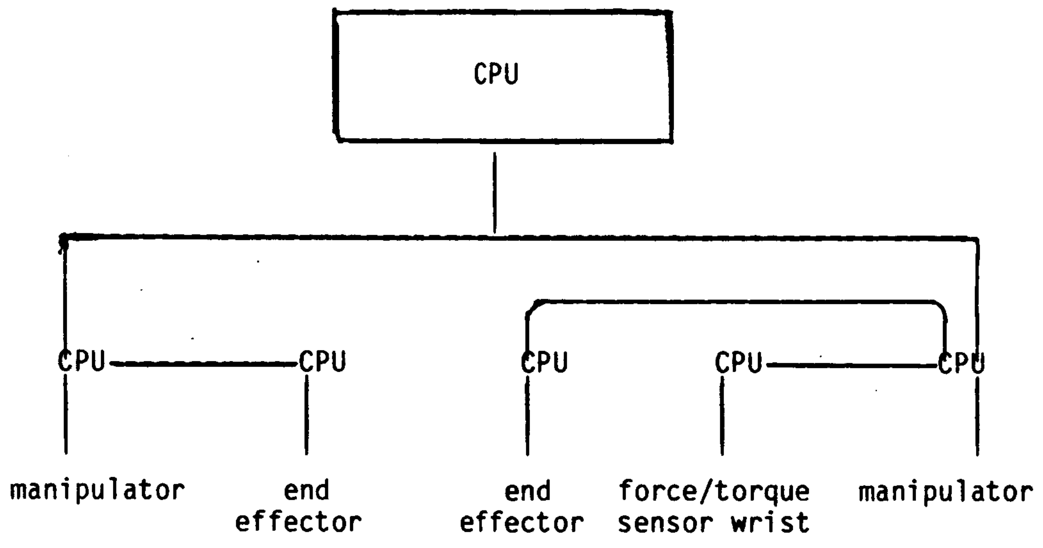

Figure 1: Multiple robot arms cell

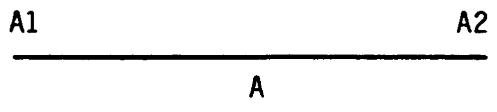

Figure 2: A seesaw