GRAPHICAL PROGRAMMING AND THE USE OF SIMULATION FOR SPACE-BASED MANIPULATORS

Debra S. McGrath and James C. Reynolds
The MITRE Corporation
1120 NASA Road 1
Houston, TX 77058

ABSTRACT

Robotic manipulators are difficult to program even without the special requirements of a zero-gravity environment. While attention should be paid to investigating the usefulness of industrial application programming methods to space manipulators, new methods with potential application to both environments need to be invented. These methods should allow various levels of autonomy and human-in-the-loop interaction and simple, rapid switching among them. For all methods simulation must be integrated to provide reliability and safety. Graphical programming of manipulutors is a candidate for an effective robot programming method despite current limitations in input devices and displays. A research project in task-level robot programming has built an innovative interface to a state-of-the-art commercial simulation and robot programming platform. The prototype demonstrates simple augmented methods for graphical programming and simulation which may be of particular interest to those concerned with Space Station applications; its development has also raised important issues for the development of more sophisticated robot programming tools. This paper discusses both aspects of the project.

1. INTRODUCTION

1.1 Inherent Difficulty of Robot Programming

Programming robotic manipulators for safe and reliable execution in the face of inevitably uncertain conditions is difficult for a number of reasons [7]. Many robots today still provide only rudimentary control instructions that are roughly equivalent to machine language for computers. The only space-based robotic arm, the Remote Manipulator System (RMS), can only be programmed, as opposed to teleoperated, by setting the joint values. Obviously, the impossibility of envisioning the movement of the arm by mentally solving the forward kinematic problem makes this method unsatisfactory.

More advanced industrial manipulators provide sophisticated control languages like VAL II or Karel with point-level instructions and modern branching constructs for structured programming. Most often, these languages are not used because it is difficult for programmers to reliably envision spatial operations and exceptions even at the point level. A "bug" introduced off-line can have much more disastrous effects in a robot program than, for example, in a word processor.

Consequently, most robots are programmed with a teach pendant. This is an easy method and appropriate for simple, repetitive tasks. Its disadvantages are many: on the factory floor it requires down-time; complicated tasks like assembly are almost impossible to program; there is little if any branching capability, especially on complex feedback from machine vision systems or force/torque sensors. With regard to its use in space-based robotics, this last disadvantage is decisive. In addition, the experience of the RMS shows that space-based manipulators are more likely needed for complicated, "one-of-a-kind" tasks than for simple, repetitive operations, and on the Space Station Freedom, NASA plans to use the Flight Telerobotic Servicer (FTS) for assembly.

1.2 Task-level Robot Programming

1.2.1 Requirements for Task-level Robot Programming

The difficulties and limitations of current methods of programming robotic manipulators both on Earth and in space have motivated considerable research in developing new methods. One line of research beginning in 1976 [13] and continuing with impressive momentum during recent years has the goal of developing systems that allow manipulators to be programmed at the "task" level. An example of a space-oriented task command is PLACE ORU-1 IN PAYLOAD-BIN-2. A task-level robot programming system would translate this command into a sequence of motions and sensing operations that would reliably and safely accomplish the task. A necessary component of such a system is a model of the workspace and manipulator, including geometry with tolerances, kinematics, and dynamic attributes like mass and required forces and torques for assembly. The key challenge to building such a system is that any model is inaccurate to some extent, and therefore manipulator motion occurs in the context of uncertainty [8].

1.2.2 Programming Methods Must Allow Various Levels of Autonomy

One advantage of a task-level programming system for space-based robotics is that it provides the building blocks for various levels of autonomy. This is essential for the astronaut who uses the system to control a manipulator to gain confidence in its reliability and safety. At a lower level of autonomy the sequence of point-level motions and sensing operations generated by the task-level system can be examined, simulated, or executed one at a time under close monitoring; at a higher level of autonomy a number of task-level commands could be combined into a more complicated script. Using advanced ideas of augmented control [2], teleoperation could take over at any time.

1.3 The Need for Simulation

As mentioned above, it is absolutely imperative that a robot programming system provides simulation capabilities. The use of simulation is as vital to programming manipulators as the use of a symbolic debugger is vital to programming large data manipulation applications for a conventional computer system. Despite a long-time and often world-class emphasis on simulation for training, NASA has not baselined simulation capabilities for the FTS. Real-time, three-dimensional (3D) simulation of manipulators in space is necessary for astronauts to gain confidence in any form of autonomous control.

1.4 Graphical Programming

All of the research at Stanford, MIT, and Carnegie Mellon in task-level robot programming has assumed a keyboard-based textual interface with the manipulator. Until recently this was justified by the relatively low resolution and slow speed of graphics workstations, and the resulting difficulty in specifying robot motions and effects. The last few years have demonstrated that high resolution, three-dimensional graphics displayed in real-time can be a practical component of a desktop-based robot programming system, while even more revolutionary simulation capabilities like stereoscopic displays and three-dimensional input devices are realizable in the near future.

For space the need for non-keyboard devices to input manipulator commands that can be simulated in real-time 3D is even more critical. Keyboard input is simply too awkward in zero gravity, and robot programming requires too much knowledge of the workspace not to be facilitated by contemporary interface components like menus and mice.

1.5 The Task-level Robot Programming Prototype

For the last year and a half the MITRE Corporation, with initial funding from NASA, has been conducting research in approaches to building a task-level robot programming system (TLRPS). A prototype has been built that includes an innovative interface to a state-of-the-art commercial simulation and robot programming software platform (Deneb Robotics' IGRIP) running on an advanced graphics workstation (Silicon Graphics' IRIS 4D/70GT). The interface is used to control a Microbot Alpha manipulator performing pick-and-place and bin-filling operations in an appropriately simple plastic blocks world. Feedback is provided by a 2D vision system capable of recognizing circles, triangles, and squares.

The prototype demonstrates simple augmented methods for graphical task specification and the use of simulation for testing commands that may be of interest to those concerned with Space Station Freedom robotic applications. In the course of building the prototype, limitations in today's 3D display and non-textual input devices became apparent, suggesting new requirements for tomorrow's applications.

2. SIMULATION

2.1 Need for Simulation on board the Space Station Freedom

Simulation of planned RMS teleoperation is conducted rigorously prior to Shuttle missions using world-class high fidelity simulation facilities. Experience has shown that many uses of the RMS were not predicted and yet were critical to mission success. On the Space Station Freedom, over much longer duty cycles, this may be even more true for its manipulators. If this is the case and control of space-based manipulators advances from continual man-in-the-loop teleoperation to some degree of autonomy, then simulation capabilities on board will be a necessity.

2.2 Uses of Simulation

2.2.1 Complete Simulation before Execution

This type of simulation is used today for the off-line programming of industrial manipulators and could be used for ground-based programming of complicated planned manipulator operations on the Space Station Freedom. The programmer describes the manipulator actions desired, either at a point-level or a task-level, and then views the complete simulation that was specified. If there are problems like joint limits exceeded or collisions detected, the program is altered and simulated again. From this loop of program, simulate, re-program, will develop a correct program that can with confidence be downloaded and executed by the actual manipulator. This use of simulation has been implemented in MITRE's task-level robot programming system prototype, building on top of IGRIP's simulation and robot-specific translation capabilities.

2.2.2 Simulation Simultaneous with Execution

The task-level robot programming system prototype uses the simulation capability of IGRIP simultaneously with execution by the Microbot manipulator to implement collision detection, reachability checks, and simple grasp planning. The simulation is run ahead of the execution by the robot so that any actions that result in undesirable effects are not physically carried out. This type of operation would allow continual supervision of the manipulator without teleoperation, and it is believed that the strain on the astronaut in a space-based implementation would be significantly reduced. It is important that the world model and the graphical display be updated to reflect changes in the workspace after manipulator actions. Ideally, this would be in real-time; in MITRE's task-level robot programming system prototype, updates can occur only after the task or collection of tasks is completed, and the Microbot returns to home position.

3. GRAPHICAL PROGRAMMING

3.1 Commercial programs

The ability to graphically program a robot is limited by the current state-of-the-art in input devices and displays. Although true 3D displays and pointing devices are in development, they are not widely available. We are therefore constrained to showing a projection of a 3D scene on a 2D graphics screen. The latest generation of graphics workstations lets us show perspective views of 3D scenes and even translate and rotate them in real time, but they are still only 2D projections.

Trying to point to an arbitrary location in 3 dimensions on a 2D screen poses problems. It is difficult to determine the dimension corresponding to depth into the scene, or distance from the user (see Figure 1). Several methods are used to help remedy this problem. The S-GEOMETRY 3D graphics software from Symbolics attempts to alleviate the problem by allowing optional cursors for selecting points [12]. These cursors have extra lines drawn to help the user determine the current 3D location. One cursor, the arm cursor (see Figure 2), has lines drawn from the cursor point to the intersection of

each of the x, y, and z planes. A second cursor, the box cursor (see Figure 3), has lines drawn from the cursor point to the intersection of the three planes (as in the arm cursor) and lines drawn to the coordinate axes. Cursor motion in combination with mouse buttons determine the 3D motion of the point. Even with these methods, accurately specifying a 3D location requires concentration and can be difficult if the screen is cluttered with objects.

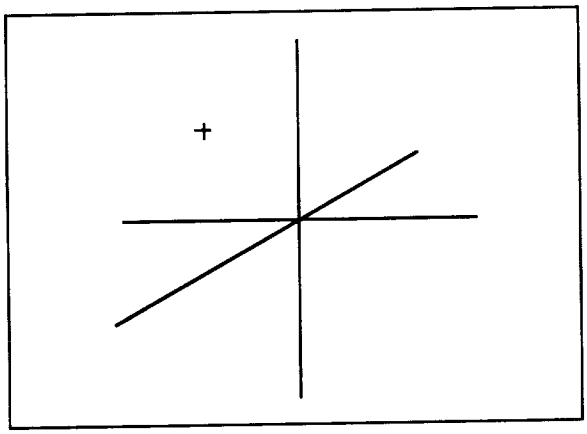

Figure 1 - Crosshair Cursor

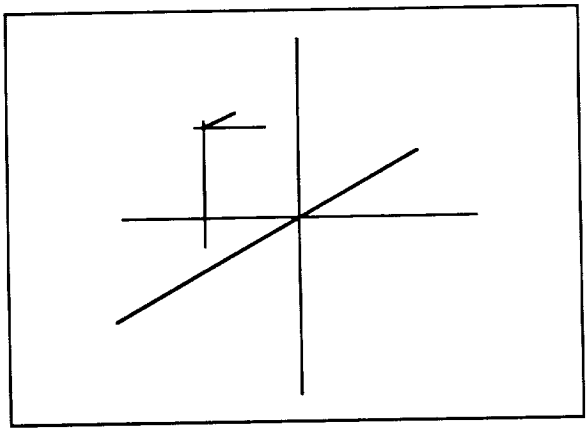

Figure 2 - Arm Cursor

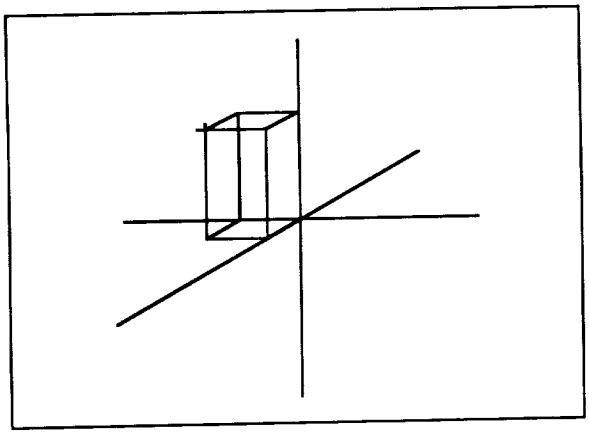

Figure 3 - Box Cursor

Commercially available robot programming and simulation packages typically allow the user to program the robot by specifying a desired position and orientation for the manipulator's toolpoint. The toolpoint is most often the tip of the robot gripper and its position and orientation in space is called a "pose". A pose can be specified by giving the x, y, and z coordinates and the yaw, pitch, and roll angles or by selecting predefined locations in the robot's workspace. These predefined, named location are called tagpoints or reference frames by the various robot programming packages [3, 4, 9]. To avoid the 3D pointing problems mentioned above, the tagpoints are defined in advance by specifying coordinates or by aligning a new coordinate system with components of objects in the workspace. Thus a tagpoint can be coincident with the local origin of an object or aligned with a vertex, for example. The robot programming package can perform the inverse kinematics to translate a pose into a set of joint angles, or a configuration, for the specific robot. Motion between configurations can be constrained to be a straight line in cartesian space or can be joint-interpolated, meaning the joint angles will change linearly between configurations but the resulting path of the toolpoint will be curved.

Robot programming and simulation packages generally provide a Pascal-like language in which robot programs can be written and simulated. Figure 4 is an example of a program written in GSL, the programming language in IGRIP from Deneb Robotics. These languages are robot-independent and are therefore ideal for prototyping and simulation, where different manipulators may be tested. If a translator is available, the programs in these language can be translated into the language understood by the robot controller hardware or software. For example, the program in Figure 4 can be converted to a form that a Microbot Alpha I robot can use, as in Figure 5. It is obvious that programs written in the high-level languages are much easier to write, test, and debug than are robot-specific programs.

```
program moveit
----------------------------------------

VAR
 round_cap_1_1, pos_1:  POSITION
----------------------------------------
begin
  UNITS = ENGLISH
  $motype = JOINT
  move link  6  by  50 relative nosimul
  move near round_cap_1_1 by 3
  move to round_cap_1_1
  move link  6  by  -30 relative nosimul
  grab round_cap_1_1 at link 6
  move away  3
  move near pos_1 by 3
  move to pos_1
  release round_cap_1_1
  move link  6  by  30 relative nosimul
  move away  3
  move home
  move link  1  by  60 relative nosimul

----------------------------------------
end moveit
```

Figure 4. GSL Program

IGRIP also has a menu-driven graphical robot programming capability, which provides an interactive method of producing GSL programs [4]. Menus provide the ability to set up parameters and choose manipulator motion commands. When a motion "move to tagpoint" type command is chosen, the user has the option of pointing to tagpoints on the screen or choosing the named point from a list, as well as keying in the name. Motion commands are carried out by the simulated robot as they are specified. An entire robot program can be generated in this manner and then tested, translated, and downloaded to the actual manipulator for execution. However, because this programming is at the point level, the use of menus can become tedious compared to simply writing the program directly in GSL.

```
@STEP 199,0,0,0,0,0,1136
@STEP 199,-1345,815,503,1025,623,503
@STEP 199,0,390,176,0,0,176
@STEP 199,0,0,0,0,0,-681
@STEP 199,0,-390,-176,0,0,-176
@STEP 199,-690,693,-408,-431,431,-408
@STEP 199,0,415,50,0,0,50
@STEP 199,0,0,0,0,0,681
@STEP 199,0,-415,-50,0,0,-50
@STEP 199,690,-1515,-98,-595,-1057,-1234
@STEP 199,1351,0,0,0,0,0
```

Figure 5. Microbot Alpha Program

3.2 TLRPS prototype and extensions

3.2.1 Specifying parts

An important step up from manually creating a robot program by specifying individual points on the manipulator's path is the ability to specify the objects to be manipulated and their goal positions, with an appropriate path automatically generated. This is the capability that the TLRPS adds to the commercial programming package. For a simple pick and place task, the object to be moved and its goal location must be specified. In the TLRPS prototype an object to be manipulated is chosen from a menu of all known objects in the workspace. The object is then highlighted on the graphics screen to allow the user to confirm the choice. A goal location can then be chosen from a menu of predefined locations and the tagpoint is highlighted for confirmation. Optionally, the user can specify an x-y location, in inches from the origin, for the goal, with z and rotations defaulted to reasonable values for a pick-and-place operation. A tagpoint for the specified goal is created and highlighted for confirmation.

Once an object and a goal have been chosen a sequence of motions is automatically generated to move the object from its original location to the goal. Checks are made along the way to ensure that all locations are reachable by the robot and neither the robot nor the object being moved will collide with other objects in the workspace. The user needs only to specify the object and the goal; the TLRPS generates the intermediate steps needed to safely move the object.

With manual robot programming or the menu-driven programming provided by IGRIP, a complete robot program must be written, tested, translated, and downloaded before the actual manipulator can be used. The TLRPS prototype allow more interactive control of the manipulator. The user may choose to test the task to be complete by simulation only. However, the TLRPS provides the ability to simultaneously simulate the task and execute it with the actual robot. This was discussed in greater detail in Section 2.

3.2.2 Dragging objects

Although specifying operations by naming objects or choosing them from a menu is certainly an advance from point-by-point programming, a more intuitive interface would be to allow the user to point to an object to be moved. Pointing to an object in 3D is not a problem. An object can be chosen by placing the cursor over any portion of the object facing the user. The function providing selection of objects with a mouse under program control is not available in the simulation and robot programming package currently being used for the TLRPS prototype, so this capability has not been implemented. Another desirable option, which is not implemented in the current prototype, is to allow the user to drag an object on the screen from its original location to a new location and have the TLRPS generate the equivalent manipulator motions to move the object without collisions. With this option there still is the problem of visualizing the 3D motion on a 2D screen.

3.3 Input Devices for 3D Manipulation and New Display Technology

Even with a more functional and open software platform to use in building a graphical interface to the control of a manipulator, the inherently two dimensional input and display technology of today's workstations would be severely limiting. There are, however, laboratory efforts and even a few advanced commercial products that demonstrate this will not be true in the near future. With respect to space-based robotics and Space Station Freedom in particular, it is of utmost importance that plans should be made now to provide hooks and scars so that these rapidly developing technologies can eventually be used.

Complex interaction in zero gravity with the computer control of dynamic physical devices such as a manipulator requires a large bandwidth of information that would be difficult if not impossible to communicate using a keyboard. Current NASA thinking foresees voice recognition technology as an alternative. The use of this technology will be an important advance, but for robotic control it will have to be implemented together with to the interactive display concepts (menus and highlighting) described above, or else too much workspace knowledge (names, dimensions, dynamic attributes) will be required of the astronaut operator.

The most natural control and programming of a manipulator, though, can only be expressed with three-dimensional input devices. If an object needs to be grasped, the operator should only have to move his hand appropriately and the effect should be displayed on the screen. This is, in fact, possible today. There are two commercial products, a glove-like device (DataGlove™ by VPL Research) and an optical gesture sensing device (by Sensor Frame Corporation) that could be used to build an interface to a manipulator. Ames Research Center is already using the first device in conjunction with demonstrations of their head-mounted display technology. An alternate device for manipulator programming that is within the realm of today's technology, although not commercially available, is a small manipulator replica that could provide true three-dimensional input for graphical display.

Equally important, especially to NASA, is the development of true three-dimensional displays. This is necessary for remote

teleoperation as well as supervised autonomy. It has been demonstrated that the remote control of robots in response to ordinary video feedback is extremely difficult. True 3D displays may be built using stereoscopic or holographic technology; there are many laboratory efforts currently in progress with the objective of developing these displays.

4. ISSUES

4.1 Uncertainty

Most robot programming and control systems assume a perfect world: error-free sensors, perfect object models, and robots that can be positioned precisely where desired. The real world, unfortunately, falls far short of perfection. The robot and environment can be engineered to minimize these errors and uncertainties, but they will never be perfect. Robot programs need to handle differences between models and actual objects and handle errors in sensor data and manipulator control. Practical methods for dealing with these uncertainties need to be developed. Brooks [1], Erdmann [6], Durrant-Whyte [5], and Volz, Xiao, and Desai [14], among others, have published work upon which a practical system might be based.

4.2 Path planning

The current version of the TLRPS prototype does not have any true path planning capabilities. To move an object from one position to another the system follows a fixed set of steps with a few parameters for initial object location and goal. The manipulator first opens the gripper and moves to a pose directly above the object to be moved, then moves straight down over the object and closes the gripper to grasp the object. Next the manipulator moves straight up, then to a pose directly above the object's goal location, then straight down. Then the robot opens the gripper and moves straight up again. This sequence of motions is sufficient to handle any simple pick-and-place operation in two dimensions, where the objects are all roughly the same height and can be grasped from the top.

Some minor modifications of this same plan would enable the system to handle a more extensive collection of objects and limited three dimensional placement. To cover a larger set of tasks, however, these heuristics should be replaced by algorithmic path planning. This would add the capability, given the initial and goal poses, to compute a path for the manipulator through the free space or unoccupied volume of the workspace avoiding collisions. Some of the commercial robot programming package vendors are developing this kind of path planning capability.

4.3 Human-in-the-loop

Any method of robot programming must allow various levels of autonomy and human-in-the-loop interaction and simple, rapid switching among them. While it is desirable to automate as much as possible to free the astronaut from tedious and time-consuming chores, we still need to allow the human to immediately and safely assume control if it becomes necessary. Once a crisis situation has ended, the system should be able to resume autonomous operations with as little input from the human as possible and preferable without having to re-plan an entire task from scratch. Sheridan [10], Stark, Kim, and Tendick [11], and Conway, Volz, and Walker [2], among others, have all made some suggestions and progress along these lines, but there is still much work to be done.

5. CONCLUSIONS AND FUTURE WORK

Graphical programming of manipulators is an effective approach despite current limitations in input devices and displays. The prototype TLRPS described in this paper includes an interface that takes advantage of these graphical techniques. In the future we would like to investigate both the use of voice recognition technology with the interactive display methods described in this paper and true three-dimensional input devices that promise a more natural way of programming manipulators. We feel this is an especially important technology area for NASA to develop and exploit.

REFERENCES

1. Brooks, Rodney, "Symbolic Error Analysis and Robot Programming," *International Journal of Robotics Research*, Vol. 1, No. 4, Winter 1982, pp. 29-68.

2. Conway, Lynn, et al., "Tele-Autonomous Systems: Methods and Architectures for Intermingling Autonomous and Telerobotic Technology," in *Proceedings 1987 IEEE International Conference on Robotics and Automation*, Raleigh, NC, 1987, pp. 1121-1130.

3. Deneb Robotics, Inc., *GSL Graphics Simulation Language Reference Manual*, Version 1.5, Deneb Robotics, Inc., Troy, MI, July 1988.

4. Deneb Robotics, Inc., *IGRIP Simulation System User Manual*, Version 1.5, Deneb Robotics, Inc., Troy, MI, July 1988.

5. Durrant-Whyte, Hugh, "Uncertain Geometry in Robotics," *IEEE Journal of Robotics and Automation*, Vol 4, No. 1, February, 1988, pp. 23-31.

6. Erdmann, Michael, *On Motion Planning with Uncertainty*, Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1984.

7. Lozano-Perez, Tomas, "Robot Programming," *Proceedings of the IEEE*, Vol. 71, No. 7, July 1983.

8. Lozano-Perez, Tomas and Brooks, Rodney, "An Approach to Automatic Robot Programming," A.I. Memo No. 842, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, April 1985.

9. Silma, Inc., *CimStation User's Manual*, Revision 3.0, Silma, Inc., Los Altos, CA, 1986.

10. Sheridan, Thomas, "Human Supervisory Control of Robot Systems," in *Proceedings 1986 IEEE International Conference on Robotics and Automation*, San Francisco, CA, 1986, pp. 808-812.

11. Stark, Lawrence, et al., "Cooperative Control in Telerobotics," in *Proceedings 1988 IEEE International Conference on Robotics and Automation*, Philadelphia, PA, 1988, pp. 593-595.

12. Symbolics, Inc., *S-Geometry*, Graphics Division of Symbolics, Inc., Cambridge, MA, October 1986.

13. Taylor, Russell, "The Synthesis of Manipulator Control Programs from Task-Level Specification," AIM-382, Stanford Artificial Intelligence Laboratory, Palo Alto, CA, July 1976.

14. Volz, Richard, et al., "Contact Formations and Design Constraints: A New Basis for the Automatic Generation of Robot Programs," unpublished report, the University of Michigan and the Jet Propulsion Laboratory, 1988.