Proceedings of the IEEE Conference on Control Applications, pp.508-513, Albany, NY, 1995

Real-Time Control Interface for $(DM)^2$

Donald D. Nelson and Yangsheng Xu
Carnegie Mellon University
The Robotics Institute
School of Computer Science

Abstract

The Dual-Use Mobile Detachable Manipulator, or $(DM)^2$, is a modular robot with reconfigurable hardware and real-time controls for lunar exploration and maintenance. Since the controls software is used directly only at a very low level, the purpose of the User Interface to create an abstraction through which the robot is controlled. This is accomplished with the use of interactive graphics to automatically generate and communicate lower level robotic commands, making the robot more usable.

1. Introduction

Our goal is to develop a lunar robot which can perform a variety of tasks that would be valuable in lunar base operations. We developed $(DM)^2$, a modular, reconfigurable robot that is capable of performing multiple functions in a lunar environment. The categories of tasks that be performed by the robot are the lunar terrain exploration and excavation tasks with the attached arm and base configuration, and the climbing, construction and plant operation tasks with the free arm and independent base configuration.

The idea behind the Dual-Use Mobile Detachable Manipulator is that organizing a robot into modules would enable the hardware to have a large, versatile workspace, have the accuracy and large manipulability within that workspace, and for coordination of multiple robots in a workspace. Because the arm module can attach and ride on the base, the arm can reach a large area in which to detach, grasp an excavation tool, or grasp an object. The detachability of the arm allows it to climb on various lunar structures that have graspable surfaces. The detached arm could also cooperate with other arms in a large number of configurations that are discussed in the following.

FIGURE 1. $(DM)^2$ Robot

The research into practical operations for the robot was done through the simulation of several configurations of the robot [Figures 2,3]. The configurations included a shoveling attachment for the hand, the detached arm configuration for structure walking, and the grasping and manipulating capability of the hand on various lunar structures.

2. Global Motion Control

The first level of breaking down the calls to the real-time robotic control functions and state variables is the use of a higher-level robot command language.

0-7803-2550-8/95$4.00©1995 IEEE

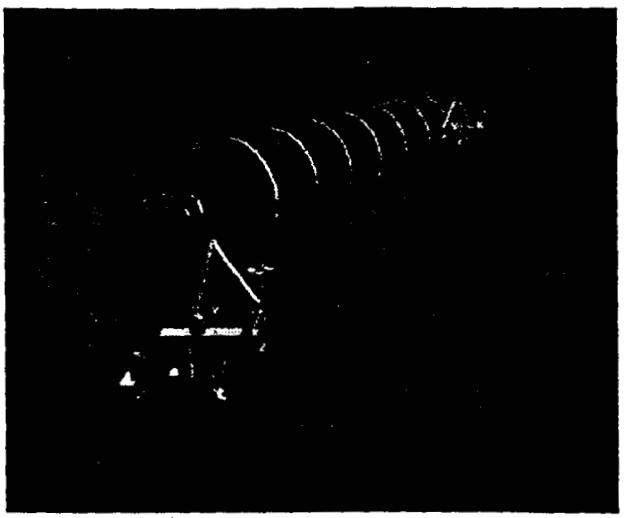

FIGURE 2. (DM)^2 Operation in Lunar Environment

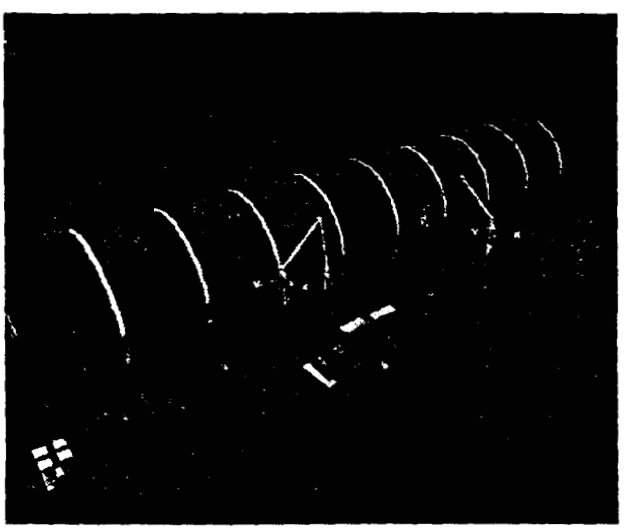

FIGURE 3. (DM)^2 Detached Arm Configuration

2.1. Extensible Robot Command Language

The relative ease of generating text commands is one of the motivations for the creation of a lexical analyzer and parser which interpret the commands for the rest of the control system. Another motivation lies in the extensibility and modularity of the command interface since the parser can be easily extended to allow for additional controller or other real-time subsystem additions.

2.2. Simple Specification with Side, Top, and 3D Views

To provide a very high level of instruction to the (DM)^2 robot, a mechanism of automatically generating the robotic commands for the parser has been implemented. The method is the interactive graphical description of robotic commands for a variety of robotic motions and a networking communications algorithm that provides the resulting commands from an unconstrained system to the real-time system.

The networked communications system is necessary since the real-time hardware and operating system do not have graphics or any user-output facilities. The graphical interactions are point-and-click mouse specifications and also object selections through a menu "database" containing information about object positions.

2.3. Graphical Goal Specification

The variety of graphically-modeled robotic motions include a free arm movement option, a specially coordinated base and arm reaching and grasping movement, and independent base movement. The free arm and free base movement refer to the state when the arm is not attached to the base.

Free arm and base movements can be run in parallel through a synchronization command language construction. This language construction is also formulated automatically in the graphical user interface.

The robotic motions are generally specified by moving a database goal object to a place and orientation in cartesian space and then selecting that object (see Figure 4). This directing method is much more intuitive that an interface of specifying an array of joint values, even if the joint values are graphically specified. Multiple moves and base switches are also more feasible. Cartesian space direction has the advantage that the goal is immediately realized. Joint calculation and a approach-deproach sequence of robotic commands are automatically generated.

As another option, for the free arm movement, the point-and-click mechanism is used to define the goal coordinates in three dimensional space. The points in the x-z plane (Figure 9) are specified in the side viewing area, and the points in the x-y plane are specified by click selection in the top viewing area.

In the point-and-click detailed view (Figure 10), the interface to the wrist and hand orientations that has worked the best is having the orientations specified with sliders relative to the ground or goal workspace, not the local robot workspace. Grasping of objects or walking along handles is usually visualized at a common angle with respect to the ground, or world coordinate frame, rather than the robot's end-effector coordinate frame.

2.4. Switch Base

The inverse kinematics mechanism used in converting cartesian specification into joint instructions takes into

account an arbitrary base orientation. The base orientation is in the orientation that was previously specified as the hand goal position before the "Switch Base" command was given.

"Switch Base" allows the free arm to grasp onto a handle, and then let the working base of the robot be that handle. The other gripper is opened and moves to subsequent goal position specifications. Comments in the parsing code are also generated to keep track of which gripper is active after a "Switch Base."

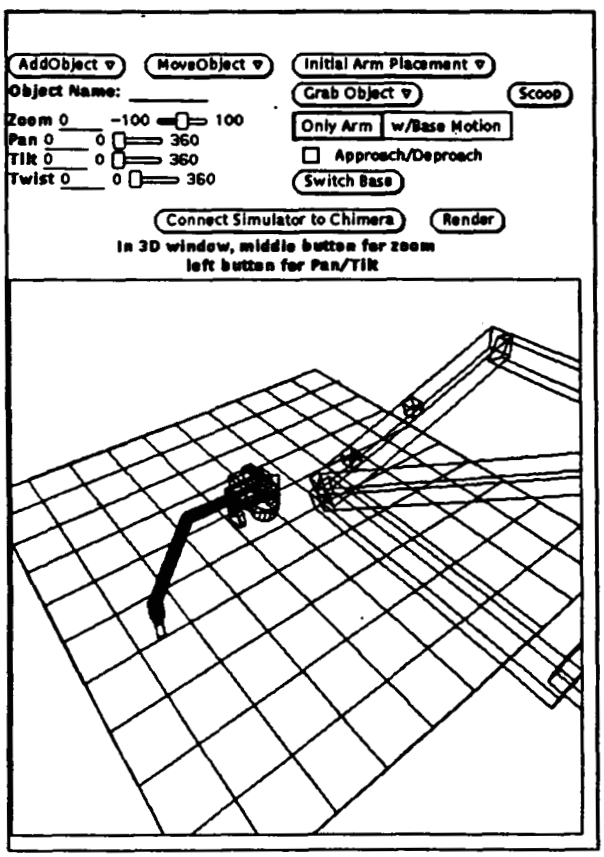

FIGURE 4. UI Scenario for Arm Goal Specification

3. Hand/Gripper Detailed View

3.1. Screen Size and GUI Windows

Some of the generally accepted results from research into usability and convenience of Graphical User Interfaces is that user preference is few windows, most importantly, and small window size, along with the need for a good design and layout of the interface widgets in a window. The trade-off in the $(DM)^2$ user interface is the need for a large window for entering detailed points in the robotic workspace or for multiple smaller windows for entering accurate points about the workspace. The best solution for the $(DM)^2$ interface is for an average size window that contains an overall view of the robot's workspace and a zooming option which can magnify a view in the small window that provides a picture of the robot's end-effector.

An important capability of the interactive graphical specification is the detailed specification of the end-effector's goal position. A detailed view that is continuously centered on the end-effector using a perspective projection representation has been employed to more easily specify the gripper's goal location [Figure 5]. Animation in real-time using a wireframe representation and viewpoint rotation about the end-effector helped in visualizing the relationship of the robot and its goal objects. A "Pan and Tilt" interface method found in other 3D modeling packages proved to be the most intuitive method of rotation about the workspace axes.

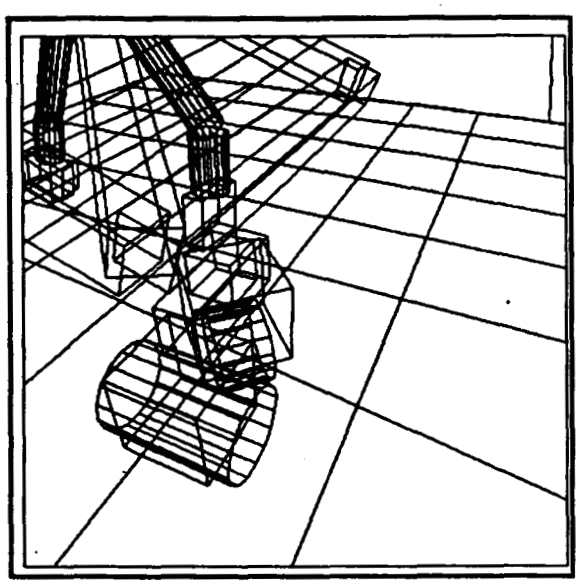

FIGURE 5. UI Detailed View Scenario

The practical matter of needing a large amount of view or window space to represent the gripper's exact location makes the detailed view necessary. Thus, an approximate goal location is entered into the larger view of the robot in the Top and Side windows. Then the goal is adjusted in the detailed view area. The wrist and hand orientations are also more visible in the detail view. The numerical cartesian position of the goal object is also written to the display so that a user may precisely enter an object position.

3.2. Perspective Projection

To give the robot programmer a more realistic look-and-feel of working in world space to add objects, switch the

robot's base, and enter commands, a perspective rather than orthographic projection was used throughout the wireframe rendering and animation interface. A wide field of view was used to enhance the "vanishing in the distance" effect. Camera twist about the viewing axis and the Pan and Tilt rotations instead of the more cumbersome to use "Look From" and "Look At" were also part of the projection model.

4. Incorporation of Configuration Optimizations

Because $(DM)^2$ is an overactuated manipulator, with a 5 DOF arm and a 3 DOF base, the motion of the arm-base configuration needs to be solved in a way that makes the best use of the redundant manipulator. The optimal choice among the space of solutions is determined according to the solution manipulability about the goal, the solution stability with respect to the extension of the arm from the base, and the solution energy requirements for performance of the solution. The algorithm that supplies the solution is supplied in [3].

The $(DM)^2$ interface makes use of the optimization algorithm by simply specifying the start position of the base in the x-y plane and the three dimensional goal position and wrist orientation of the arm's gripper or end-effector. The orientation of the base is not specified since the current workspace is assumed to be the base orientation coordinate frame. The objects in the world coordinate frame are brought into the base's frame in higher level modules such as the path planner.

5. Incorporation of the Path Planner

5.1. Navigation Through an Obstacle Field and Concurrent Arm Movement

The mobility of the arm/base configuration over a large area in the x-y plane is provided by an automated path planning mechanism [Figure 6]. This motion is also specified using a start and goal position in the x-y plane as in the optimization movement option, but the movement is intended over a large area that contains obstacles that are not specified more precisely than a "grid cell" in the movement plane. Thus the grid cell is larger than the robot's mobile base, and the start and goal positions are given as particular grid cells. This movement option allows a path planner [4] to generate a sequence of base commands that will steer $(DM)^2$ through a workspace containing an arbitrary number of obstacles. Options in the path planner include a changing safety boundary around obstacles and changing the granularity of the grid cells.

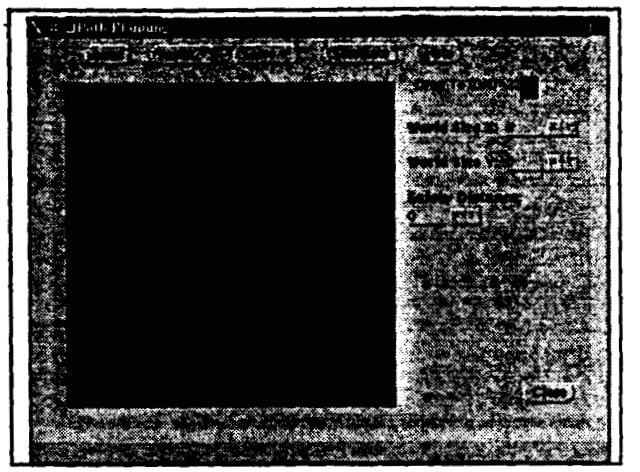

FIGURE 6. UI Path Planning Mechanism

5.2. Drag and Drop into Panel List

The arm/base optimization is then applied at the end of a generated path or at some point in the path to accurately grasp an object or to detach the arm onto a handle. The optimization algorithm is automatically given the orientation of the base at the end of the path.

The point at which the optimization algorithm is applied in the path is determined by where the arm commands are placed using a "drag and drop" windowing mechanism. The path commands are generated into a sequence of frames -- pages of commands instead of a long scrolling list -- in the $(DM)^2$ command-organizing window (see Figure 7).

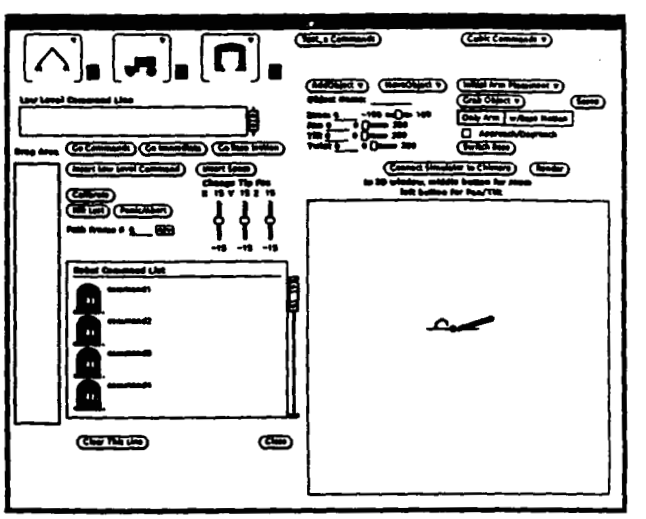

FIGURE 7. UI Command Organizing Scenario

5.3. Multiple Path Planning Solutions, Arm

Optimizations, and Approach/Deproach

At a particular frame, the position of the base is specified and made available to the other motion options. A free arm movement or optimization movement can then be specified and "dragged" graphically into the organizing area.

To ensure a smooth encounter between the gripper and the object, with respect to a given grasp orientation to the object, it became useful to develop a mechanism which would generate commands for the gripper to approach an object a certain distance away in a direction along the grasp vector, and then allow the gripper to move in (see Figure 8).

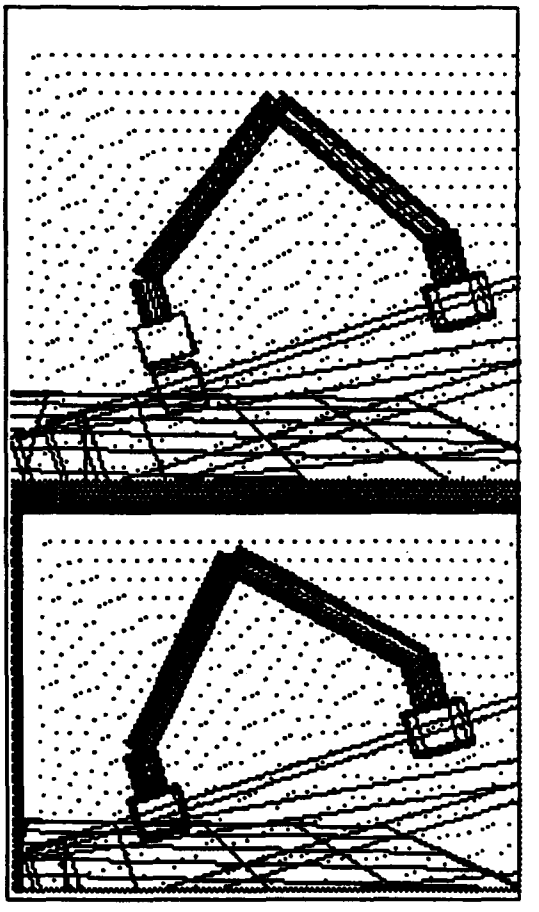

FIGURE 8. Approach/Deproach Scenario

6. Communications Algorithm

A real-time communications algorithm has been implemented to handle the problem of interfacing the GUI that allows user interaction and is not under real-time constraints with the robotic controls system.

The communications system consists of threads of execution that are spawned by the real-time controls software upon the request for GUI information from one of the controls subsystems. The command macro parsing subsystem makes extensive use of the threads when it asks for text information from the GUI. Other subsystems can simultaneously request GUI information that is only limited by the number of threads the system can spawn and still have enough cycles to serve the real-time modules.

The threads connect to an Internet socket that is a global GUI socket. The response from the GUI is a port number for a new socket connection that is then opened between a forked process on the GUI workstation side and real-time.system. Information that becomes available upon subsequent user action is then communicated over the new socket. The threads are blocked until communication is received. The communications real-time subsystem that is periodically polling the status of the threads' shared memory will get the information results and pass it along to the command macro subsystem.

However, for state information like encoder values that needs to be sent from the real-time system to the GUI, a dedicated socket that was permanent in nature was used along with a system of semaphores that would perform the blocking instead of the socket read performing the blocking. The real-time thread responsible for taking the state information from the module would block initially, would get unblocked by the cycling module after a set number of iterations, and then would block itself again after the information was relayed. In this way, the state information would be sent in a set interval to allow the GUI to update its knowledge about the configuration of the robot.

7. Demonstration

A real-time communications algorithm has been implemented to handle requests by the real-time module for information from the Graphical User Interface. A special communications channel using this algorithm was also dedicated to transmit joint data from the real-time encoder module to the GUI so that the GUI could use its animation capability to display the current configuration of the robot while it is moving or still.

Exact measurements in the physical workspace of the robot are needed when specifying a robotic task in the User Interface. Since the robot hardware does not yet have a vision system at this time, accuracy is especially

important in the case of the movement options where the arm gripper is grasping an object. The location given to the User Interface as the goal must be precisely measured. The detailed view of the gripper and the database of precise object coordinates are solutions to minimizing the error in specifying the goal location.

A sample demo of free arm movement along two handles that have been set up in the lab provided good results. The gripper handles were entered into the arm specification scenario. The demo was dependent upon placing the base in a predetermined position from which the arm could detach and grasp the first handle. The error from the generated commands is on the order of tenths of degrees in the joint angles in the arm. The error seems to be entirely dependent on measurements on the work environment as was previously explained.

The system of an arbitrary number of communication channels between the real-time system and the User Interface was shown to have fairly good performance. A large set of sample commands that were generated from the path planner were communicated on the order of seconds. The overhead of threads, additional state variables, and shared memory did not cause the real-time control system to miss cycles.

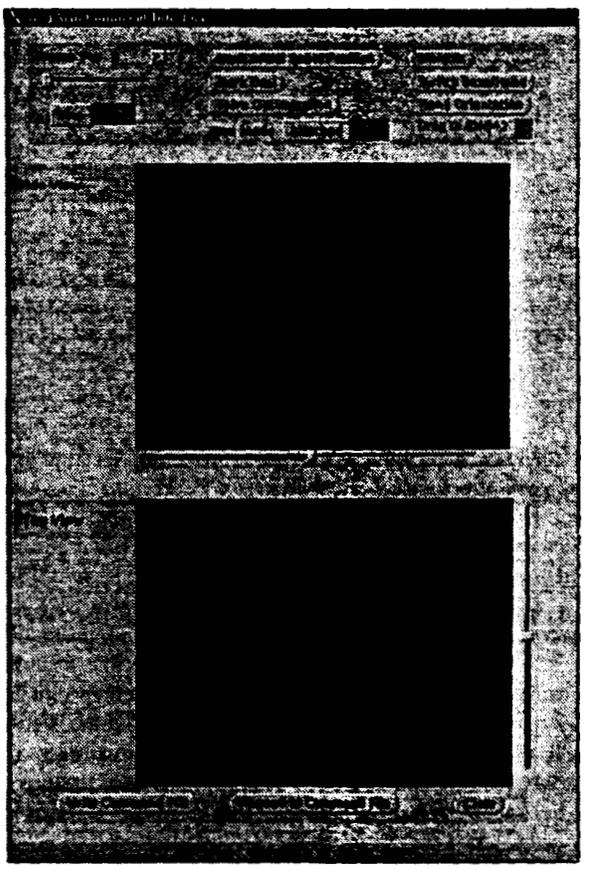

FIGURE 9. **Point-And-Click Side and Top View**

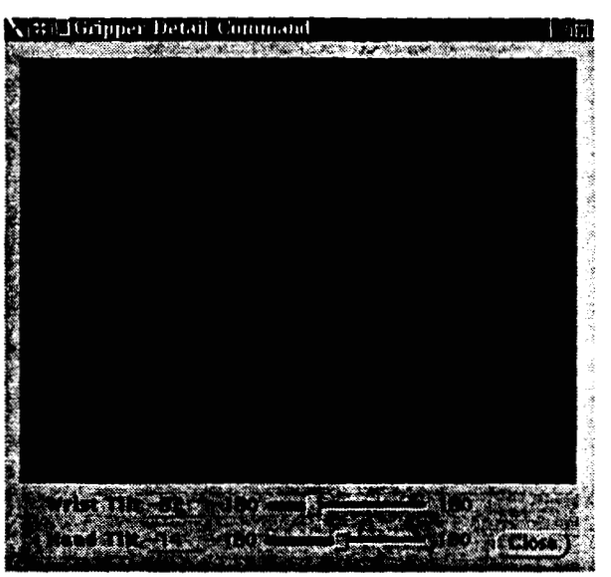

FIGURE 10. **Point-And-Click Detailed View**

8. Acknowledgments

The work was partially supported by the Space Project Office of Shimizu Corporation in Japan. We are grateful for the help in hardware and software development to Chris Lee, Ben Brown, Jeffrey Yang, Shigeru Aoki, Chuck Su, and Alongkrit Chutinan.

9. References

[1] Space-time Constraints, Andrew Witkin and Michael Kass, Proc. Siggraph '88.

[2] Christopher Lee and Yangsheng Xu, The Design of a Control Architecture for $(DM)^2$, Project Report, Carnegie Mellon University, 1995.

[3] Alongkrit Chutinan, Coordination Control of the Lunar Mobile Manipulator, Project Report, Carnegie Mellon University, 1994.

[4] Charles Su, Path Planning for the Dual-Use Detachable Mobile Manipulator, Project Report, Carnegie Mellon University, 1994.