Pergamon

0892-9270(95)00052-6

Space Technol. Vol. 16, No. 1, pp. 49–58, 1996

0892-9270/96 $15.00 + 0.00

(DM)^2: A MODULAR SOLUTION FOR ROBOTIC LUNAR MISSIONS

CHRISTOPHER LEE and YANGSHENG XU

The Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, U.S.A.

Abstract—This paper describes the Dual-Use Mobile Detachable Manipulator, (DM)^2, which is a mobile manipulator designed to operate in a lunar station scenario. (DM)^2 is designed to perform two very different kinds of tasks: exploration on the lunar terrain, and maintenance work in lunar manufacturing plants. Both tasks are essential during the early construction of a lunar station. In order to be able to competently perform both tasks, (DM)^2 embodies a modular hardware design—namely a mobile base, and a detachable, symmetric manipulator arm with exchangeable grippers at each end. (DM)^2 can work with a number of possibly different arms, each of which may use several kinds of specialized detachable end-effectors. This flexible hardware configuration enables the robot to be useful for many different kinds of operations on a lunar base. In turn, this flexibility of hardware configuration necessitates a software control architecture that is equally flexible—allowing for on-the-fly reconfiguration, and independence of high-level functionality from the details of the current hardware configuration. (DM)^2 is designed to perform its tasks either autonomously based on a task model and realtime vision system, or under the supervision of a human operator through a custom realtime teleoperation interface. Copyright © 1996 Elsevier Science Ltd.

1. (DM)^2 SYSTEM DEVELOPMENT

1.1. Concept

The process of construction and maintenance of a lunar base will be technologically challenging and expensive. One of the most expensive resources in a lunar base will be the astronauts themselves. It is thus important to make sure that the efforts of the astronauts be directed towards the high-level tasks which require their full abilities, while more menial work is performed through automation. Teleoperated robots could also allow astronauts to remotely perform tasks outside of the climate-controlled base structure without the complications of going through airlocks and unnecessary exposure to cosmic radiation on the lunar surface.

The number of tasks which could be automated in a lunar base is substantial. Light construction work on base structures, structure inspection, maintenance of oxygen plants and other structures, collection of soil samples, light excavation, exploration, and tool transport are just a few examples of these tasks [1–3]. The variety of tasks required for this kind of mission is such that highly flexible robot systems should in most cases be more cost-effective than specialized hardware.

We are developing a robot system, (DM)^2, which is a mobile manipulator designed to perform two very different kinds of tasks in a lunar station scenario: exploration on the lunar terrain, and assistance in lunar manufacturing plants and construction sites [4]. Although robots are often designed to perform either one of these tasks, particularly exploration [5, 6], (DM)^2 is able to perform both kinds of tasks because of its modular hardware configuration. The robot consists of a three-wheel mobile base and a detachable manipulator arm. This 5-DOF arm is symmetric, with a gripper at each end. When the arm attaches to the mobile base by grasping a handle with one of its grippers, the robot becomes a mobile manipulator and can perform exploration tasks such as collecting soil samples, surveying the lunar surface, and transporting tools and supplies. When the robot nears a lunar-station structure such as a manufacturing center or a fuel tank, the manipulator arm can detach from the base and walk hand-over-hand, by grasping a series of handles on the structure, to perform tasks such as structure inspection, parts delivery, and simple assembly tasks. Computer simulations of the robot (Fig. 1) demonstrate the suitability of the robot's configuration and mobility for shovelling, turning, gripping and climbing.

The ultimate goal of the (DM)^2's modularity is to allow a group of different bases and different detachable arms to cooperate and combine to generate configurations which are optimal for specific tasks. For instance, the robot could be specialized for a surface excavation task by providing it with a stronger, lower-DOF arm and a scooper end-effector, or it could be specialized for a dextrous manipulation task using a high-DOF arm and a hand-like gripper. A very strong low-DOF arm could even be used as a leg to enhance base mobility in very difficult terrain. The base would be chosen to have the appropriate range and mobility for the mission. For some jobs, it may be possible to send a group of bases and arms to a site to work together on a task, or to use multiple arms with a single base to enhance its ability to manipulate objects.

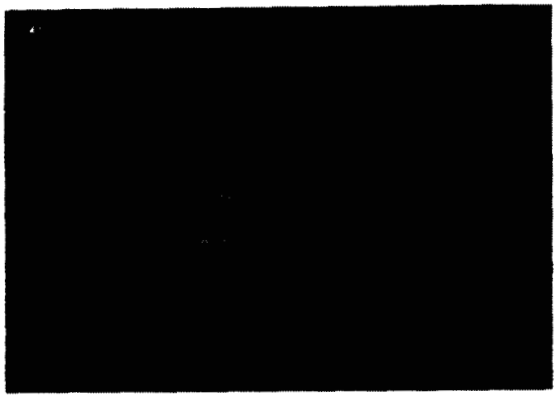


Fig. 1. Computer simulation of $(DM)^2$.

The modularity of this robot hardware concept has motivated the development of a software control architecture which is equally modular and reconfigurable. This software architecture relies on a careful stratification of responsibility for the robot control task into low-level modules for control of particular hardware components in a given robot configuration, and high-level modules which can manage the execution of tasks with minimal dependence on the current hardware configuration.

For a full demonstration of the robot concept, a teleoperation interface has been developed and a robot vision system is under development. The goal of the project is to demonstrate the entire system as a reliable mechanism for remote operation of a wide range of tasks which will need to be performed in a lunar base scenario.

1.2. $(DM)^2$ system

Successful operation of a robot system involves the integration of many different subsystems. Flexible and reliable mechanical hardware is needed so that the robot may interact with its environment and perform its work. Sensing is necessary for monitoring the state of the robot and its progress in performing tasks. Electronic subsystems are necessary for power, control, and support of sensors. A software control system running on a realtime computer is necessary for controlling and coordinating the operation of the hardware, and for cooperating with a human teleoperator.

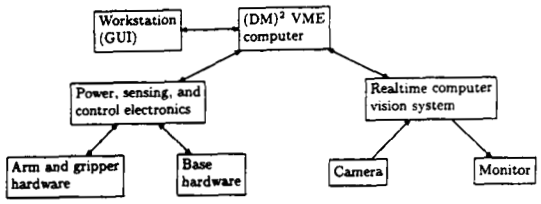


Fig. 2. Overall system architecture of $(DM)^2$.

Figure 2 shows the overall system architecture of $(DM)^2$. The teleoperation interface (Section 4) runs on a separate workstation and communicates with $(DM)^2$'s realtime computer over an Ethernet connection. The realtime computer runs the robot's internal software (Section 2), which controls the robot hardware through communication with a custom array of power, control, and sensing electronics (Section 1.3). This set of electronics is interfaced to the arm and base components of the robot, which include actuators and encoders at each joint, and sensors on the grippers. A vision system (Section 2.3) is used for sensing in the course of performing tasks, and communicates with the robot's realtime computer via a memory-mapped VME to VME communications card.

1.3. $(DM)^2$ hardware

$(DM)^2$ was designed and built within the Space Robotics Lab at CMU. Its main hardware components are its mobile base, its detachable manipulator, and its grippers. The grippers are currently

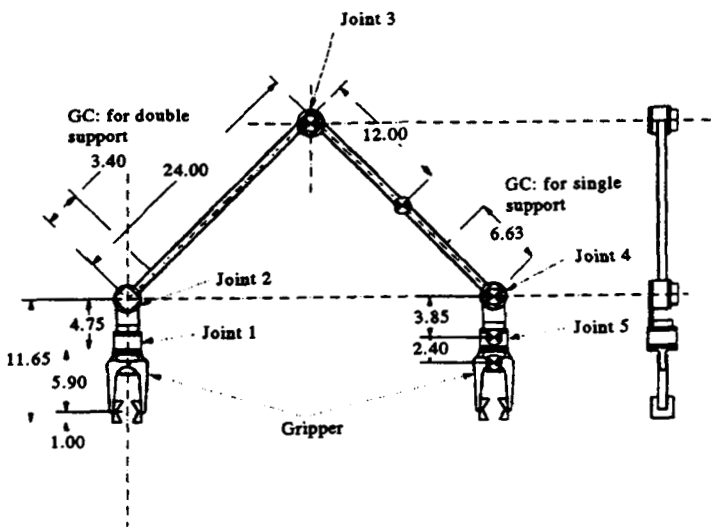


Fig. 3. Current manipulator design.

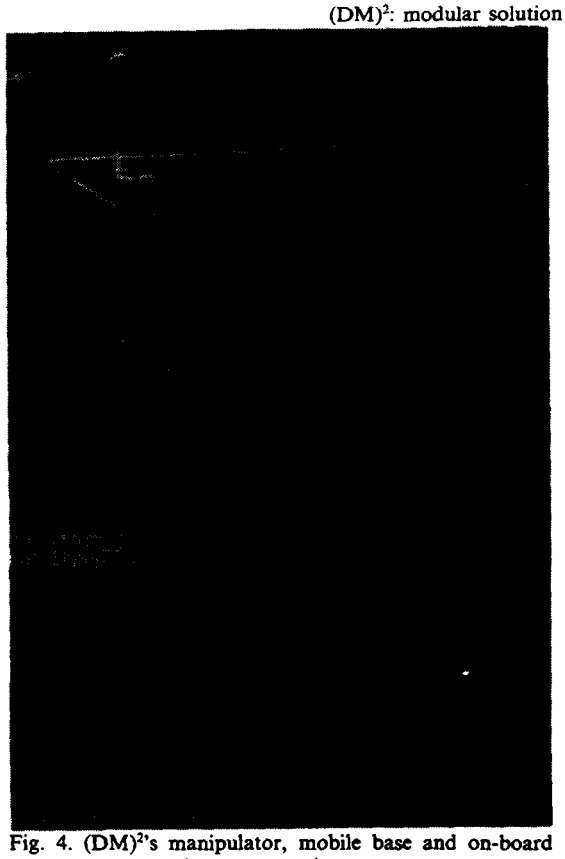


Fig. 4. (DM)²'s manipulator, mobile base and on-board gravity compensation system.

designed to grasp handles on the base or a lunar structure, and are detachable from the arm. The current design of the manipulator arm is shown in Fig. 3. It is symmetric and has five degrees of freedom for walking on truss-like lunar structures and for performing tasks as a mobile manipulator.

The mobile base for the robot is currently the Neptune base, previously developed by the Mobile Robot Laboratory at CMU [7]. This base, along with the arm and an on-board gravity compensation system currently under development, are shown in Fig. 4. When the manipulator arm is not attached to the base, it walks on a "lunar structure" mockup with an actively-controlled counterweight-based overhead gravity compensation system [8].

The control hardware of (DM)² consists of the following components:

- 6U Versa Module Eurocard (VME) cage containing two 68020 CPU boards, ADC and DAC boards, a parallel I/O board, and a VME to VME communications board;
- VME-based Sun workstation with a VME to VME communications board;
- quadrature decoding hardware, linear transconductance motor amplifier hardware, an H-bridge motor controller, and power supplies;
- VME-based Innovision computer vision system, with specialized hardware for image digitization, frame-buffering, specialized image analysis computations, and VME to VME communications.

2. SOFTWARE ARCHITECTURE

The creation of a software architecture for controlling such a highly flexible system is a challenging problem. Our approach is the development of an architecture which is as inherently modular and reconfigurable as the hardware, and to carefully stratify the architecture into low-level modules for control of particular hardware in the configuration, and a high-level system which can manage the execution of tasks with minimal regard for the current hardware configuration.

Modular software allows for easy analysis, understanding, and modification of the code, as well as forming the foundation of support for hardware modularity and reconfigurability. It also simplifies the addition of new robot capabilities.

Reconfigurability is largely supported though the configuration files which accompany each independent module of the low-level system (i.e. each Chimera subsystem module [9]). Thus, the choice of controllers, trajectory generators, etc . . . , are specifiable and fully tunable (e.g. controller gains) at runtime rather than link-time. This allows for faster development cycles in the long-term and "freer" experimentation in the course of research.

A high-level control system takes text-based descriptions of robot tasks from the operator, and then supervises the execution of the low-level modules to perform the specified tasks. The motivation of this high-level control system is to provide a framework for teleoperation and more intelligent performance of tasks. The descriptions of tasks may be specified graphically through a custom graphical user interface (Section 4).

The overall architecture of (DM)²'s software is shown in Fig. 5. The software on the robot consists of realtime code and high-level code. The realtime software modules (see Section 2.1) are responsible for arm and base control, trajectory generation, gripper control, and servicing actuators and encoders. The high-level code (Section 2.2) is responsible for getting task-specifications from the operator and coordinating the realtime system during the execution of these tasks. Both realtime and high-level software can communicate with a graphical user interface (Section 4) via Ethernet sockets. A separate, specialized computer runs (DM)²'s vision system (Section 2.3) and communicates with the realtime software through a memory-mapped VME-to-VME communications card.

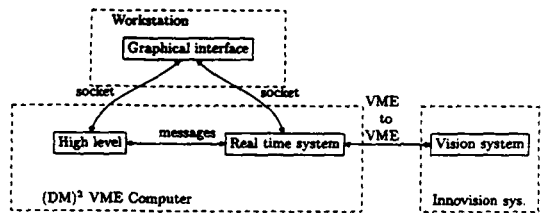
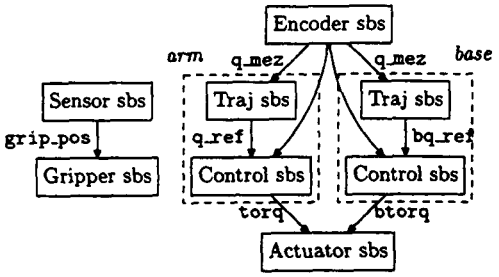


Fig. 5. Overall software system and communication architecture.



State variable	meaning
grip_pos	gripper position
q_mez	measured joint angles
q_ref	reference arm joint angles
bq_ref	reference base joint angles
torq	arm joint torques
btorq	base joint torques

Fig. 6. (Simplified) data-flow of realtime software.

2.1. *Realtime system*

$(DM)^2$ runs the Chimera 3.2 operating system, a multiprocessor realtime operating system which supports software written as a group of independent modules [9, 10]. These modules each run as one (or more) separate threads, cycle at independent frequencies, and can be allocated at runtime to any of the processors on the system without modification. Modules can communicate through a variety of means, but realtime control modules generally exchange data at the beginning and end of each execution cycle through Chimera's state variable table mechanism.

The modularity of the robot's low-level software mirrors the modularity of the robot's hardware. When the robot changes physical configuration (the arm detaching from the base, for example), the robot can autonomously turn modules on and off to rearrange the software and its data-flow to fit the robot's new configuration. This architecture also facilitates rapid support for new hardware. Hardware is added by writing new (or adapting old) hardware-interface and control modules, and interfacing these modules to the state variable table. Debugging is simplified because individual modules are generally simple and connected through well-defined interfaces. Processor loads and controller performance can be optimized at runtime by modifying cycle frequencies of periodic modules and moving modules between processors.

Figure 6 shows the basic outline of the relationship between the realtime modules of the robot system (there are more variables and modules than this in practice). The modules cycle at different frequencies and exchange information through the state variable table, potentially across different CPUs on the system. These modules are turned-on and turned-off by the high level system as needed, receive commands in the form of messages from the high-level system (e.g. "open the gripper"), and use messages to send information (e.g. "trajectory-generation finished"). Note that the arm and base control systems actually use different instances of the same modules. Different configuration files tell these instances which state variables to read and write and how to behave so that they can perform more than one task.

The modules corresponding to the controller and trajectory generator for the arm and the mobile base make use of the "controlobject" library. This library allows for the creation of controllers or trajectory generators from different "controlobjects" by reading a configuration file. For example, a configuration file might specify that a joint controller be built from the combination of a PID controller object, a feed-forward gravity-compensation object, a friction-compensation object, and a joint-torque limiter object. In addition, the library allows for the definition of several different controllers, and allows the system to select among them at runtime. Thus, $(DM)^2$ can select trajectory generators for performing a motion either in joint space or Cartesian space, or a generator which follows commands from the vision system for visual servoing. It is also possible to change arm and base controllers to allow for performing various tasks or for tuning the controllers.

Table 1. Control objects

Control object	Description
	Controller control objects
PDPI2	Combination PD and PID controller
tq_limit	Torque limiter
gravcmp	Basic gravity compensation
gravcomp2	Experimental gravity compensation system
friction	Joint friction compensation
enet_torq	Realtime torque adjustment through Ethernet to GUI
enet_PID	Realtime PID gain adjustment through Ethernet to GUI
	Trajectory generation control objects
linj	Linear joint interpolation
blendj	Linear joint interpolation with parabolic blends
blendj_t	Linear joint interpolation with parabolic blends; generates its own time of motion
blendx_t	Linear Cartesian space interpolation with parabolic blends; generates its own time of motion
stay	Generates trajectory to stay in place some amount of time
vservo	Gets direction of motion from vision system

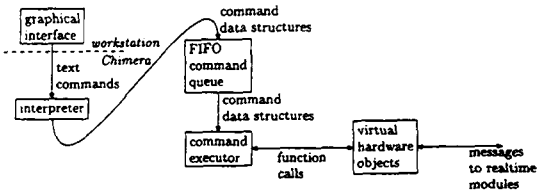


Fig. 7. High level system architecture.

Current control-objects for joint control and trajectory generation are listed in Table 1.

Creation of new control-objects is facilitated by a code-generation tool which reads information about the controller from the developer and then uses this information to create a source-code skeleton from a template file. Filling-in the code skeleton to produce a functional control-object takes just a couple of minutes. Two lines of code are then needed to add the object to the usable library of control-objects for a given module.

2.2. High-level control system

In order to understand and supervise the execution of tasks which are specified by an operator, (DM)² has a high-level control system which communicates with and coordinates the operation of the low-level realtime modules.

The basic organization of the high-level system is shown in Fig. 7. Commands, in text form, are sent from the interface (or from the host workstation console) to the robot. Here they are interpreted, and placed into a first-in–first-out (FIFO) command queue. When the command executor is finished with its current command, it gets the next command from this queue, and begins the execution of the command. The most basic commands interact with software representations of the robot hardware called "virtual hardware objects." These, in turn, communicate with the low-level modules via a Chimera messaging system to coordinate the performance of the robot hardware.

The text-based commands which are sent to the robot from the host workstation may be of two kinds. The first kind of command is a predefined low-level command, called an "atomic command." Examples of this kind of command are those which open and close the grippers, move the arm or base to a given position, and change the realtime controllers or the trajectory generation algorithm. The second kind of command is a user-defined command, which is written in the text-based language as a combination of the atomic commands and previously defined user-commands. The specification of a user-defined command resembles a command to execute a certain task, but the robot is told to name the task command and assign it a parameter list rather than execute it. The text-based language has a straightforward syntax which supports concurrent execution of commands and synchronization points for these concurrent commands.

In order to simplify the development and testing of the software which parses and executes these commands, the data structures and supporting code are developed as a hierarchy of C++ classes. This approach not only simplifies issues such as building the parse trees, memory management, and command execution, but also allows specialized commands to be derived from more general commands and thereby inherit code from the general commands.

Each atomic command and user-defined function receives a list of parameters as input. These parameters are type-checked as the tree of nested command-lists is built. Parameter lists in user-defined commands allow for definition of generic commands that can represent tasks such as "take a soil sample at location *X*" or "move the arm to *Y* using control-object *Z*." Individual parameters may be character strings, integers, floating-point numbers, or vectors.

Command-atoms that are meant to perform hardware actions operate by communicating with the appropriate realtime controller-module. It is, however, undesirable for each command-macro to worry about interprocess communication with the relevant controller modules and all of the complexities, such as preserving state-information and avoiding race conditions, that this communication implies. It is also undesirable for two or more commands running synchronously to be allowed to give potentially conflicting commands to the same controller-module.

To avoid these problems, C++ objects representing *virtual hardware* were created to represent each set of hardware control-modules. Each virtual hardware object provides a limited number of handle-objects (typically one) to command-atoms that can be used to perform operations with the hardware. To open gripper 1, for example, a command-atom can acquire a handle for the gripper object, then call the "open" member function on that handle. The gripper object is responsible for passing this command to the gripper controller-module, and then monitoring the state of the gripper through interprocess communication with the gripper module. Any command-atom may thus query the state of the gripper through the virtual hardware object rather than the controller-module.

This approach lets authors of command-atoms concentrate on functionality rather than interprocess communication. It also centralizes data about the entire robot within the high-level system and provides a consistent interface for accessing that information. Virtual hardware objects interface with controller-modules by sending messages to these modules. Limiting access to virtual objects through handles avoids situations such as two conflicting commands being sent to the base-trajectory generator at once.

If the realtime modules are reconfigured to reflect changes in the hardware configuration of the robot, the virtual hardware objects are responsible for ensuring that problems of redirecting communication are handled, and for providing information about the reconfiguration to the command-atoms.

```
# Turn the base 90 degrees to the right
command base_turn_right() = async {  # async means do the following as a sequence:
    base_move_joint(1, -1.571, 3.0); # Move steering wheel to -pi/2 radians
    base_move_rel  (0,  6.75,  7.0); # Move drive wheel so base rotates pi rad.
    base_move_joint(1,   0.0,  3.0); # Move steering wheel to 0 rad. (straight)
}                                    # The turn takes 3.0+7.0+3.0 = 13 seconds.
```

Fig. 8. Command for turning the robot base 90° to the right.

Similar object-based interfaces can be also constructed for using remote resources such as off-line planners on the host workstation, making the use of these resources as straightforward as using robot hardware.

The robot reads a file on initialization which defines a set of new compound commands for the user based on the simpler predefined atomic commands. This has allowed us to eliminate some of the more complicated atomic commands, and rewrite them as compound commands from simpler atomic commands. This has the benefit of making operation of the robot simpler, and allowing the atomic commands of the robot's command language to be fewer in number and more directly related to the fundamental capabilities of the hardware. It also allows for faster development of task descriptions for the robot, because less overall programming is involved, and new commands can be added to the robot's command-library without recompiling the robot's source code. Since much of the robot's programming is more easily done in the command language than in C or C++, the power of the command language will be gradually increased and more of the programmed robot functionality moved into compound macro-commands rather than compiled code.

Figure 8 shows an example of a command which previously was defined as an atomic command, and now is easily defined in the initialization file as a compound command. This command turns the robot's mobile base 90 degrees to the right.

Figure 9 shows the definition of the command which is currently used to tell the robot to perform the visual-servoing task.

2.3. Vision system

An Innovision image processing computer system is currently used as a basis for a vision system for $(DM)^2$. The computer has specialized vision hardware and software libraries for high-speed image analysis. We are currently using this system to test strategies for automatically identifying image features to track, and to test strategies for realtime tracking of these features for high-speed visual feedback in manipulation tasks. We have mounted a camera on one end-effector of the robot, so that it may be used to guide the end-effector during tasks such as attaching to the handle on the mobile base under initial uncertainty in the position of the base. Visual feedback is important for $(DM)^2$ during actual performance of tasks, because there will always be uncertainty in the positions of the mobile base, the positions of lunar

```
# Put the arm of the robot under the control of the vision system
command vservo() = async {
      # Use the visual servoing control object as the current
      #  trajectory generator for the robot arm.  This control object
      #  is responsible for communication with the vision system.
    arm_traj_set_cobj("vservo");
      # Now, start the arm controller and the trajectory generator,
      #  and perform the servoing task until either the trajectory
      #  generator or the vision system decides to end the servo task.
    arm_control_mode();
}
```

Fig. 9. Command for performing visual servoing control.

structures with respect to the mobile base, and the positions of tools and other objects the robot would like to grasp.

We have developed initial working versions of software for identifying trackable features in the camera image of targets which we want to track, and of software for realtime tracking of these features during visual servoing. We have also developed a preliminary communications protocol for high-speed communication between the robot's control computers and the vision system through a BIT3 VME-to-VME card, which allows the robot and vision system to work together to perform visual servoing tasks. The protocol makes sure that each system changes its operational state in a safe, predictable manner when the robot asks the vision system to help in a servoing task and when either the robot or the vision system decides to end the visual-servoing task. This protocol has been tested on a simple task of letting the vision computer send end-effector-relative motion commands to the robot and having the robot follow these commands.

Our next step in the development of the vision system is to create a mapping between the locations of feature points tracked by the vision software and desired motion of the robot. This problem is equivalent to the description of the visual servoing task. Our first goal is to create a controller which will allow the vision system to guide the end-effector in grasping a handle by tracking the coordinates of features in the image of a target attached to the handle. This experiment will allow us to tune both the control strategy and also the performance of the feature-extraction and feature-tracking algorithms. This problem is an exciting research challenge, for which we will develop, implement, and compare many different kinds of strategies. As we develop more capable visual control strategies, we will try to perform increasingly difficult tasks.

3. PLANNING

3.1. Arm/base coordination

The combination of manipulator arm and mobile base introduces redundancy in the process of manipulation for (DM)². Thus, there are virtually infinite ways to coordinate the position of the mobile base and the end-effector so that the end-effector is at a specified point in the robot's workspace. To take advantage of this redundancy, we have developed a coordination scheme for the base and manipulator arm which can optimize the use of both with respect to a set of performance criteria. These criteria are

(1) the manipulability of the manipulator arm,
(2) the distance the base must move,
(3) the static stability of the whole robot when it takes on a load.

The result of optimizing the robot's configuration for these criteria is that the robot is able to perform tasks with maximum dexterity and minimum energy consumption, while enabling it to handle loads with minimal danger of overturning. The coordination scheme we have developed allows these three criteria to be weighted with respect to one another depending on which factors are considered most important at a given time.

The first criterion, manipulability, is a common measure of a manipulator's ability to move its end-effector from a given configuration [11]. By maximizing this measure, we help to ensure that the manipulator is in a configuration that is well suited for generating the necessary motions for performing its tasks. To generate a manipulability index for (DM)², we first derive the Jacobian matrix of the manipulator from its forward kinematic equations. The scalar index of manipulability is the norm of this Jacobian matrix. For this measure, we consider only the effects of the three joints nearest the base of the manipulator so that the robot may use the remaining two joints to control the orientation of the gripper at the desired location.

In addition to maximizing manipulability, it is important that (DM)² conserve energy by minimizing unnecessary motion of its mobile base. Because the size and mass of the mobile base is much greater than that of manipulator arm, driving the base requires significantly more energy than moving the manipulator to generate the same motion at the end-effector. We thus consider the straight-line distance that the base must move as an index of the energy consumption required to achieve a given configuration, and attempt to minimize this index.

The last criterion we currently consider is static stability. It is important that (DM)² never try to lift a load from or move a load to any configuration for which it is in danger of overturning. We can minimize the possibility of destabilizing the robot during its work by keeping the center of the mass of the whole system as close as possible to where it was without the load. We thus use the horizontal distance of the center of mass of the loaded arm from the original center of mass as an index to be minimized.

We form an optimization function by multiplying the expressions for each of the three indices by weighting coefficients, and then subtracting the resulting energy-index term and the stability-index term from the manipulability-index term. The optimization function is

$$V = -k_1 D + k_2 w - k_3(\bar{x}^2 + \bar{y}^2),$$

where w is arm's manipulability (considering only the effect of the arm's first three joints), D is the distance the base must travel, and $\bar{x}$ and $\bar{y}$ measure the offset of the center of gravity that is generated by the load at the end-effector.

Assuming that the base will move in a straight line to its final configuration, it can be shown that for (DM)², w, $\bar{x}$, and $\bar{y}$ are functions of the desired location of the end-effector, the end-effector load,

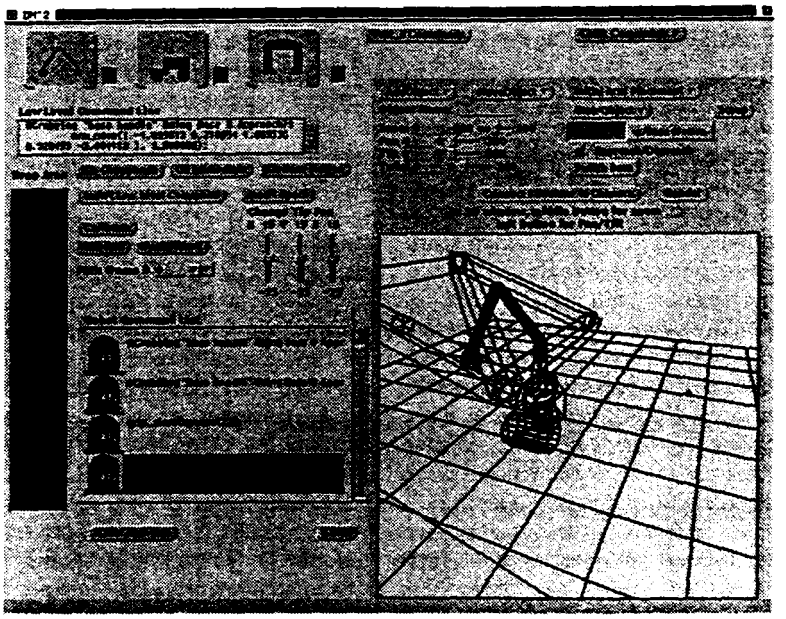


Fig. 10. $(DM)^2$ graphical teleoperation interface.

and the distance D that the base moves in the process of achieving the goal position. Thus, if the weight of the load and the desired location and orientation of the end-effector are given as inputs, we can easily perform a one-dimensional search for the value of D which maximizes V. This D-value gives the configuration of the base and manipulator that is optimal with respect to manipulability, energy consumption, and robot stability.

3.2. $(DM)^2$ mobile base path planning

The path planner developed for $(DM)^2$ is an extension of a wavefront expansion algorithm developed by Latombe [12–14]. The wavefront expansion method relies on the initialization of a grid-based model of the robot's workspace with respect to a particular goal location. Each location in the grid that is occupied by an object is designated as off-limits. The goal-region is assigned the value 1, any cell which neighbors the goal cell and is not an object is given the value 2, any cell which neighbors a cell with value 2 and has not already been assigned a value is given the value 3. This process is repeated until every cell in the grid has been assigned a value. A path from any given starting location to the goal location can be found by iteratively moving from the current location to the neighboring cell with the lowest value. This algorithm generates paths of minimum distances, with no possibility of generating local minima.

The major problem faced during the implementation of the wavefront path planner for $(DM)^2$ is the nonholonomic kinematic constraint on the motion of the wheeled base. This nonholonomic constraint restricts the geometry of the feasible free paths between the start and the goal configurations. To solve this problem, movement of the base is restricted to straight-line motion, and the grid resolution is set to 2′ by 2′. The mobile base is divided into a head and tail section, each of which occupy one cell in the workspace. The path planner uses the size of the workspace, the location of all of the objects, and the starting and ending locations for the head and the tail of the mobile base. Turns are accomplished by rotations using the center of the tail-section as a pivot point. In addition, turns are currently restricted to 90 or 180 degrees.

The process of motion planning for the base is divided into three parts: initial rotation, path following, and final rotation. This division of movement planning assures that the base will always travel forward and never backwards. Preference is given to long horizontal or vertical movements over diagonal movements so that robot motion is smooth and efficient.

4. REALTIME INTERFACE

We have developed an interface [15] for graphical teleoperation of the robot (see Fig. 10). The interface allows for scripting and execution of sequences of robot motions, and allows the scripting process to proceed through working with a graphical simulation of the robot. The view of the robot can be rapidly changed by realtime panning and tilting by dragging the mouse over the simulation window (Fig. 11). This allows the operator to make sure that a command makes sense before sending it to the robot. The operator may also specify that the simulation window update itself with the current robot position at regular intervals so that the robot's motion can be viewed in realtime through the simulation window.

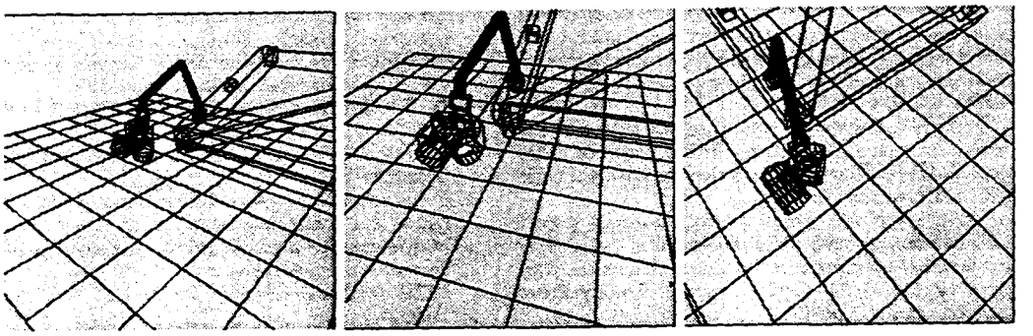


Fig. 11. Pan/tilt views of the robot through the interface.

Motion of the arm may be specified in two ways. In the first method, the operator can use the mouse to select destination points through windows which render the robot and its environment from the top-view, side-view, and "gripper-view". In the second method, the user predefines interesting-points in the robot's environment, and then instructs the robot-arm to move to these points by selecting them from a menu. The second method is often much easier than the first, because the same points are usually needed repeatedly during the course of a task. For example, the location of handles on a structure and tool locations will typically be used many times during an operation.

Robot motion may be specified in terms of free-arm motion, free-base motion, and coordinated arm/base motion (using the optimization function described in Section 3.1). In free-arm and free-base motion, the arm and base move independently. In the free-arm mode, one of the arm's grippers is considered the "base gripper" which secures it on a lunar structure or the mobile base, and the other is the "free gripper" which the arm can move to any point in its workspace. If both grippers are grasping handles at once, the "free-gripper" and "base-gripper" designations may be interchanged.

The arm walks hand-over hand by first grasping a handle with its "free gripper", then interchanging the arm's "free-gripper" and "base-gripper" designations, and finally releasing the handle held by the new "free gripper" so that it may take the next step. After an operator tells the robot arm to step onto the mobile base, the mode of the teleoperation interface can be changed so that it will generate coordinated arm/base commands.

The user interface can also be used to specify motion of the base via the path planning algorithm described in Section 3.2. A window is used to graphically specify an obstacle field around the robot and a goal position for the robot base, and the path-planner interface generates base-motion commands for reaching the specified goal.

The motion-commands specified by the teleoperator through the interface are used to generate commands in the text-based language for the robot's high-level system (Section 2.2). These commands may be edited by the user, copied into the robot's initialization file to predefine a high-level command, or sent to the robot for immediate execution.

5. CONCLUSION AND FUTURE WORK

(DM)² is a mobile manipulator designed to operate in a lunar station scenario. We have shown that with modular hardware and modular software, it is possible to perform tasks involving mobility and manipulation both on the planet surface, and on base structures. A working prototype of the robot concept has been developed, consisting of a mobile base, a detachable manipulator arm, a gravity compensation system, and a teleoperation interface.

Reliable execution of tasks will involve increasing use of vision to eliminate uncertainties and to help in item identification, and will require more intelligent high-level software for the difficult job of identifying and correcting problems which necessarily arise during the execution of complex tasks. Execution of tasks will also place a greater burden on low-level systems, and more specialized controllers will be built and integrated into the system to handle the needs of skills such as digging and transporting loads such as rock samples and tools. To make operating the system as easy as possible, we are developing technology for intelligent gesture-based teleoperation and programming-by-example, which will be integrated into the teleoperation interface.

REFERENCES

1. W. Sadeh, S. Sture and R. Miller (Eds), *Engineering, Construction, and Operation in Space III*, Vol. I and II. American Society of Civil Engineers, New York (1992).
2. G. Rodriguez (Ed), *Proceedings of NASA Conference on Space Telerobotics*. NASA, Washington, DC (1987).
3. G. Rodriguez (Ed), *Proceedings of NASA Conference on Space Telerobotics*. NASA, Washington, DC (1989).
4. Y. Xu, J. Yang, S. Aoki and B. Brown, Dual-use mobile detachable manipulator—(DM)². In *Proceedings of IEEE International Symposium on Intelligent Control*, pp. 255–260. IEEE Press, New York (1994).
5. D. B. Bickler, The new family of JPL planetary surface vehicles. In *Proceedings of International Conference on Missions, Technologies, and Design of Planetary Mobile Vehicles* (1992).

6. S. Hirose and N. Ootsukasa, Design and development of a quadra-rhomb rover for mars exploration. In *Proceedings of International Conference on Advanced Robotics*, pp. 109–112. Springer-Verlag, Berlin (1993).
7. G. Podnar, K. Dowling and M. Blackwell, A functional vehicle for autonomous mobile robot research. Tech. Rep. CMU-RI-TR-84-28, Robotics Institute, Carnegie Mellon University (1984).
8. G. White and Y. Xu, Active control z-direction gravity compensation system. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1181–1187. IEEE Computer Society Press, Los Alamitos, CA (1993).
9. D. Stewart, R. Volpe and P. Khosla, Integration of real-time software modules for reconfigurable sensor-based control systems. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 325–332. IEEE Computer Society Press, Los Alamitos, CA (1992).
10. D. Stewart, R. Volpe and P. Khosla, Design of dynamically reconfigurable real-time software using port-based objects. Tech. Rep. CMU-RI-TR-93-11, Department of Electrical and Computer Engineering, Carnegie Mellon University (July, 1993).
11. T. Yoshikawa, *Foundations of Robotics: Analysis and Control*. The MIT Press, Cambridge, MA (1990).
12. J. Latombe, *Robot Motion Planning*. Kluwer, Amsterdam (1991).
13. A Zelinsky, R. Jarvis, J. Byrne and S. Yuta, Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Processing of IEEE International Conference on Robotics and Automation*. IEEE Computer Society Press, Los Alamitos, CA (1993).
14. J. Barraquand, B. Langlois and J. Latombe, Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22 (March, 1992).
15. D. Nelson and Y. Xu, Real-time control interface for $(DM)^2$. In *IEEE Conference on Control Applications* (1995).