Proceedings of the 1996 IEEE
International Conference on Robotics and Automation
Minneapolis, Minnesota - April 1996

A SEPARABLE COMBINATION OF WHEELED ROVER AND ARM MECHANISM: $(DM)^2$

Yangsheng Xu, Christopher Lee, H. Benjamin Brown, Jr.
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We present a novel mobile manipulator concept called the Dual-Use Mobile Detachable Manipulator, or $(DM)^2$, for early construction and maintenance tasks in lunar stations. The robot consists of a wheeled rover, or mobile base and a detachable manipulator arm. The 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 center 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. The proposed concept of a separable combination of wheeled rover and arm mechanism provides not only both capabilities of locomotion and manipulation on structures and on wheeled bases, but also a high flexibility in connecting mobile bases to form various configurations of mobile robots. The paper discusses the concept and its advantages, the system under development, and its software architecture.

1 $(DM)^2$ concept

The process of construction and maintenance of a lunar base will be technologically challenging and expensive. 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, surface survey, and tool transport are just a few examples of these tasks. The tasks above can be classified into two groups: *exploration on the lunar terrain*, and *assistance at lunar manufacturing plants and construction sites* [10].

While various mobile robot concepts have been investigated for the first class of tasks, i.e., exploration, such as the rovers developed at JPL [1], CMU [4], and Tokyo Institute of Technology [3], less attention has been directed to the second class of task, i.e., maintenance and construction tasks on the lunar center. As the most expensive resource 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 in the early construction and maintenance of the lunar center.

Here we present a novel concept of mobile manipulator called *Dual-use Mobile Detachable Manipulator*, or $(DM)^2$, designed for providing required mobility and manipulation capabilities for both classes of tasks mentioned above. The robot consists of a wheeled rover (the *mobile base*) and a detachable manipulator (the *arm*). The 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 center 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.

The benefits of such a robot concept are numerous:

- *Providing manipulation and transport capability.* A conventional rover without an arm cannot perform manipulation tasks required in early construction and maintenance of lunar stations, such

0-7803-2988-4/96 $4.00 © 1996 IEEE

as taking soil samples, transporting tools, and assembling structures. Moreover, it is difficult for a fixed-base mobile manipulator to perform tasks on structures. $(DM)^2$ can detach from the mobile base and walk (or climb) on the structures for manipulation and tool transport tasks (see Figure 1 and Figure 2).

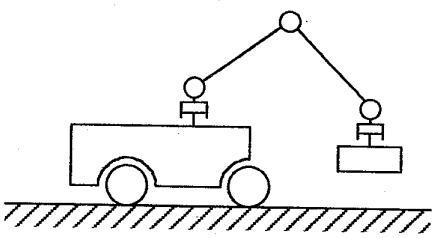


Figure 1: $(DM)^2$ can perform tool transport and manipulation tasks in lunar stations.

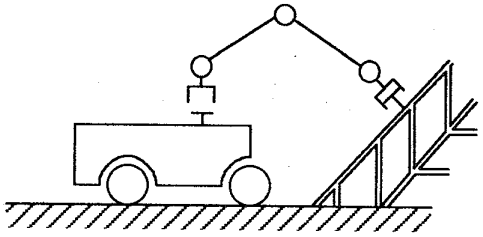


Figure 2: The arm can detach from the mobile base and climb on structures.

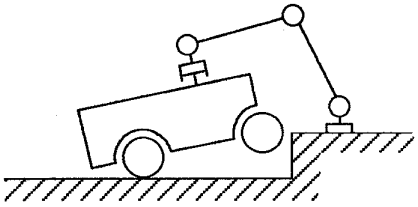


Figure 3: Using the arm as a leg allows $(DM)^2$ to lift the base for overcoming obstacles.

- *Assisting the mobile base for enhancing mobility.* $(DM)^2$ preserves the advantages of a wheeled mobile robot on smooth terrains, but also facilitates overcoming obstacles by using the arm to help lift the mobile base (see Figure 3). Hirose et al pointed out the advantage of using the combination of a fixed mobile base and arms [2]. In our concept, because the arm is detachable, with different handle locations on the mobile base, it is easy to adjust the posture of the arm relative to the ground and thus adapt to various sizes of obstacles.

- *Extending mobility on the structure.* The detachable arm provides not only manipulation capability, but also extends locomotion capability on the structures (see Figure 2). The mobility on the structures is important for various types of maintenance tasks in lunar stations, such as inspection of structures, fluid connectors, and electrical cables.

- *Potentially using arms as a part of structures permanently.* While the arm on $(DM)^2$ can assist in assembling structures, it might also be assembled as a permanent member of a structure. When a detachable arm has completed its mission, it could serve a final, permanent function and preclude the need for additional structural members (see Figure 4).

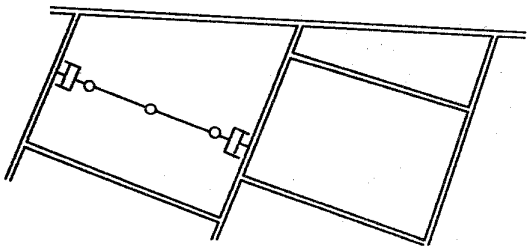


Figure 4: $(DM)^2$ may disassemble itself, becoming a permanent member of the structure.

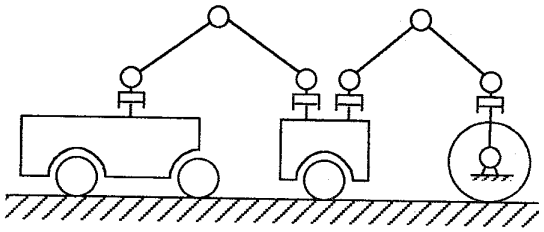


Figure 5: $(DM)^2$ can connect wheeled rovers to form various configurations of mobile robot.

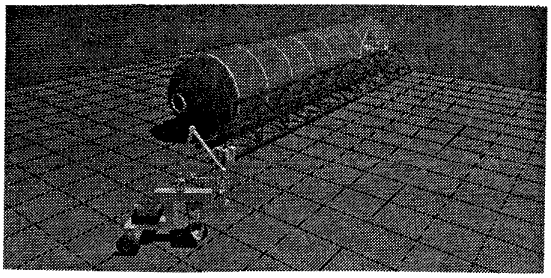


Figure 6: Computer simulation shows multiple $(DM)^2$ with or without the mobile base (or arm) operated simultaneously.

- *Connecting multiple mobile bases in various con-*

figurations. The arm on $(DM)^2$ can be used not only for attaching on the structure, but also for connecting to another mobile base (see Figure 5). Using a variety of arm and base modules with a common interface (handle), $(DM)^2$ becomes a broad concept for mobile robots. The $(DM)^2$'s modularity allows a group of different bases and different detachable arms to cooperate and combine to generate configurations which are optimal for specific tasks. The connected configurations could be train-like, snake-like, or spider-like.

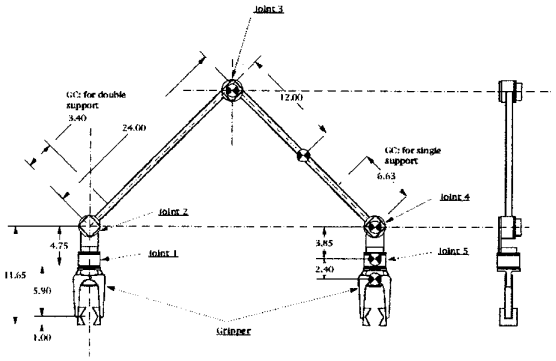


Figure 7: The detachable manipulator arm of $(DM)^2$.

- *Allowing multiple arms and mobile bases to work simultaneously.* Because the modularity and separability of the $(DM)^2$ concept, it provides the flexibility of organizing a group of robots in an optimal way. For example, the simulation of the $(DM)^2$ in Figure 6 demonstrates that $(DM)^2$ with the arm can perform exploration tasks, while the detachable arm simultaneously performs maintenance tasks on the structure. This provides a high degree of flexibility for the most efficient execution of various tasks.

2 $(DM)^2$ system

$(DM)^2$'s main hardware components are its mobile base, its detachable manipulator, and its grippers. The grippers are currently 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 Figure 7. It is symmetric and has five degrees of freedom for walking on lunar structures, in a similar manner to the configuration of $(SM)^2$ robot that was developed by the authors [11].

The mobile base currently used is the Neptune base, previously developed by the Field Robotics Center at CMU [7]. This base, along with the arm and an onboard gravity compensation system currently under development, are shown in Figure 8. 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 [9].

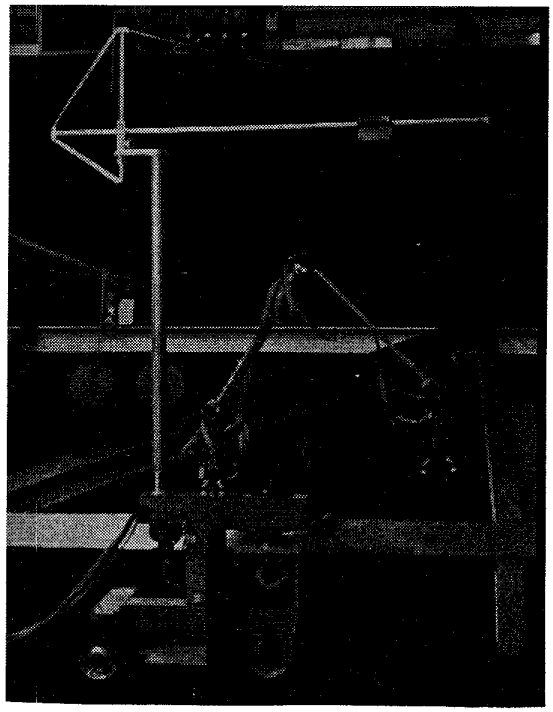


Figure 8: $(DM)^2$'s manipulator, mobile base, and onboard gravity compensation system.

The control hardware of $(DM)^2$ consists of the following components: (1) 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; (2) VME-based Sun workstation with a VME to VME communications board; (3) quadrature decoding hardware, linear transconductance motor amplifier hardware, an H-bridge motor controller, and power supplies; and (4) VME-based Innovision computer vision system, with specialized hardware for image digitization, frame-buffering, specialized image analysis computations, and VME to VME communications.

3 Software architecture

The modularity of this robot hardware concept has motivated the development of a software control archi-

tecture which is equally modular and reconfigurable. 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 [8]). Thus, the choice of controllers, trajectory generators, etc, are specifiable and fully tunable (e.g. controller gains) at run-time 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.

The software on the robot consists of realtime code and high-level code. The realtime software modules are responsible for arm and base control, trajectory generation, gripper control, and servicing actuators and encoders. The high-level code 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 via Ethernet sockets. A separate, specialized computer runs $(DM)^2$'s vision system and communicates with the realtime software through a memory-mapped VME-to-VME communications card.

3.1 Realtime system

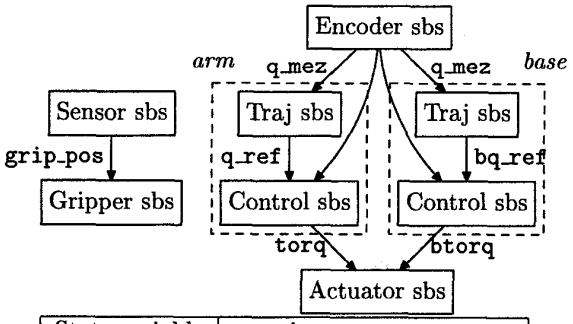
$(DM)^2$ runs the the Chimera 3.2 operating system, a multiprocessor realtime operating system which supports software written as a group of independent modules [8]. These modules each run as one (or more) separate threads, cycle at independent frequencies, and can be allocated at run-time to any of 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 the 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 9 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.



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

Figure 9: Data-flow of realtime software.

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.

3.2 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 discussed in our paper [5]. 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 command.

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.

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.

Acknowledgements

This project is partially supported by the Space Project Office of Shimizu Corporation. Funding is also provided from DOE, NSF, and Pennsylvania Space Grant graduate fellowships. Many people have provided technical contributions and other valuable helps, including Jeffrey Yang, Donald Nelson, Shigeru Aoki, Charles Su, Alongkrit Chutinan, Michael Nechyba, and Tetsuji Yoshida.

References

[1] D.B. Bickler, "The new family of JPL planetary surface vehicles", Proceedings of International Conference on Missions, Technologies, and Design of Planetary Mobile Vehicles, Toulouse, 1992.

[2] Shgeo Hirose, et al., "Fundamental considerations for the design of a planetary rover", Proceedings of IEEE International Conference on Robotics and Automation, pp. 1939-1944, 1995.

[3] Shgeo Hirose and N. Ootsukasa, "Design and development of a quadra-rhomb rover for Mars exploration", Proceedings of International Conference on Advanced Robotics, pp. 109-112, 1993.

[4] E.P. Krotkov, R.G. Simmons, and W.L. Whittaker, "Ambler: performance of a six-legged planetary rover", Acta Astronautica, Vol.35, No.1, 1995.

[5] Chris Lee and Yangsheng Xu, " $(DM)^2$: A modular solution for robotic lunar missions", (to appear in the International Journal on Space Technology).

[6] Donald Nelson and Yangsheng Xu, "Real-time control interface for $(DM)^2$", Proceedings of IEEE Conference on Control Applications, 1995.

[7] G. Podnar and K. Dowling and M. Blackwell, "A functional vehicle for autonomous mobile robot research", Robotics Institute, Carnegie Mellon University, CMU-RI-TR-84-28, 1984.

[8] D. Stewart and R. Volpe and P. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects", Robotics Institute, Carnegie Mellon University, CMU-RI-TR-93-11, 1993.

[9] G. White and Y. Xu, "An active vertical-direction gravity compensation system", IEEE Trans. on Instruments and Measurement, Vol.43, No.6, pp.786-792, 1994.

[10] Y. Xu and J. Yang and S. Aoki and B. Brown, "Dual-Use Mobile Detachable Manipulator - $(DM)^2$", Proceedings of IEEE International Symposium on Intelligent Control, pp. 255-260, 1994.

[11] Yangsheng Xu, Ben Brown, Mark Friedman, and Takeo Kanade, "Control systems of Self-Mobile Space Manipulator", IEEE Trans. on Control Systems Technology, Vol.2, No.3, pp.207-219, 1994.