Proceedings of the 1991 IEEE
International Conference on Robotics and Automation
Sacramento, California - April 1991

DEMO 89 - THE INITIAL EXPERIMENT WITH THE HERMIES-III ROBOT

D. B. Reister, J. P. Jones, P. L. Butler, M. Beckerman, and F. J. Sweeney

Center for Engineering Systems Advanced Research
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6364

Abstract

HERMIES-III is a large mobile robot designed for human scale experiments. The initial experiment with the robot (DEMO 89) was the cleanup of a simulated chemical spill. To perform the experiment, the robot was required to plan a path through an a priori known world, navigate along the path (avoiding unexpected obstacles) and locate and remove debris from a target area. This paper describes the software system that was developed to perform the experiment. The software system consisted of 19 processes that operated on a distributed set of hetrogeneous computers.

1. Introduction

The U.S. Department of Energy has provided support to four universities (Florida, Michigan, Tennessee, and Texas) and to the Oak Ridge National Laboratory (ORNL) to pursue research leading to the development and deployment of advanced robotic systems. The long-term goal is to develop systems that can perform surveillance, maintenance, and repair tasks in nuclear energy facilities or in other hazardous environments [1].

The program team has divided the research effort among the four universities and ORNL. In addition to its research effort, ORNL has the responsibility of systems integration; to integrate the robotic modules developed by the research teams into operational prototypes. The third generation Hostile Environment Robotic Machine Intelligence Experiment Series (HERMIES-III) robot is the primary testbed for systems integration.

In 1989, the objective of the systems integration experiment was to autonomously clean up a simulated chemical spill (see Fig. 1). Initially, the approximate location of the spill is known. The robot uses its knowledge of the environment to plan a path from its current location to a location close to the spill. The robot then follows the path to the spill, automatically sensing and avoiding unexpected obstacles while on route. Once it has arrived at the goal, the robot senses the debris and uses a vacuum cleaner mounted on a manipulator to remove it.

The next section describes the hardware used in the experiment. The third section discusses the shared memory communications system. The fourth section outlines the software that was used in the experiment. The fifth section details the lessons that were learned during the successful performance of the experiment.

2. Hardware

A robotic system consists of effectors, sensors, and computers. For our robotic system, we can subdivide the hardware into two categories: onboard and offboard. HERMIES-III is a large mobile robot designed for human scale experiments[2]. The chassis (1.6m X 1.3m X 1.9m) has two drive wheels and four corner caster wheels. Currently, the robot is tethered and weighs 820 kg (when battery powered, the weight will be 1230 kg). The manipulation system consists of the CESAR manipulator (CESARm) a 7-DOF compliant arm with all revolute joints, a spherical wrist, and a low friction back driveable drive train[3]. The CESARm can reach 1.4m, has a load capacity of approximately 14 kg, and has an unloaded tip speed of 3.0 m/s.

The sensor suite for HERMIES-III includes: two CCD cameras on pan/tilt platforms, a third CCD camera mounted on the CESARm, a ring of 32 sonar transceivers around the chassis, an Odetics laser range camera on a pan platform, and encoders on all motor driven shafts.

HERMIES-III has a dual computer system with both an NCUBE hypercube computer and a VME bus based system. The NCUBE system can have 16 nodes on an IBM 7532 host. The host has 1.5 MB of RAM and each node has 0.5 MB of RAM. Mass storage for the NCUBE system is provided by a 40 MB hard disk, a 1.2 MB floppy disk, and a 0.4 MB floppy disk. The VME

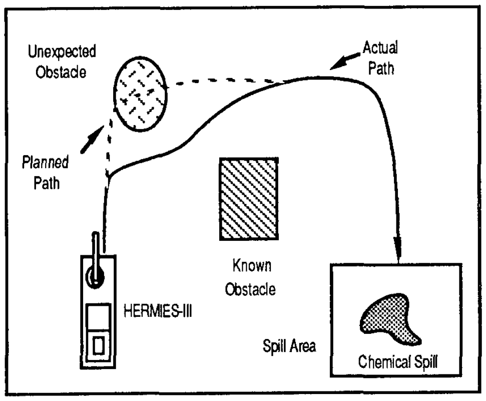

Fig. 1. Floor plan for DEMO 89, a robot experiment to autonomously clean up a simulated chemical spill.

system has 5 Motorola 68020 processors and 11 MB of RAM. Storage is provided by a 40 MB hard disk and by a 0.8 MB floppy disk.

Our robotic system has two offboard computers: Silicon Graphics IRIS-4D work station and Micro-Vax-II computer. The onboard computers and the offboard computers are linked by an Ethernet. Another offboard hardware component of the system is a three degree of freedom force reflecting manual controller (joystick). The joystick can be used to control both HERMIES-III and CESARm.

3. Communications

All robotic systems consist of a collection of effectors and sensors, together with a number of processes that control them, and that control each other. These processes communicate with one another by sharing data. To transfer data between processes and between the three computers (HERMIES-III, Silicon Graphics, and Vax), we have developed a shared memory communications system that will be outlined in this section.

Each of the three computers has a copy of the shared memory. The shared memory was divided into a number of blocks. Using a concept from the C programming language, each block of shared memory is a structure. We adopted the policy that only one process is allowed to write data into each of the structures that comprise shared memory. All processes may read any block of shared memory.

Shared memory is communicated between machines on a structure-by-structure basis. The five routines that provide the complete interface to the communications system are: mem_attach, get_read_ok, read_done, get_write_ok, and write_done. The function mem_attach returns a pointer to a structure that contains pointers to all of the structures in shared memory. The functions get_read_ok and read_done are used for reading the contents of shared memory while get_write_ok and write_done are used for writing. The get_read_ok function waits until all writing has been completed while the get_write_ok function waits until all reading has been completed. After new information has been placed in a structure, the write_done function broadcasts the information to the other computers.

The entire system is controlled by a single variable (mode), and there is only one process in the system that determines the value of this variable (Process Monitor). Decisions on the change from one mode to the next are made by this process based on the current value of the mode variable, and a state-dependent inspection of the contents of shared memory. Individual processes inspect the value of the mode variable and respond in an appropriate manner. The modes and processes for DEMO 89 are discussed in the next section.

4. Software Design

The details of the modes and processes depend on the tasks that are to be performed. For DEMO 89, there were 53 modes and 19 processes. This section describes the modes and processes.

The tasks to be performed in the experiment are illustrated in Fig. 2. The six tasks are:

1. Database Query
2. Path Planning
3. Path Execution
4. Debris Removal
5. Manual Control of the CESARm
6. Manual Control of the Platform.

Performance of the tasks required the 19 processes listed in Table 1. For each of the processes, Table 1 specifies the number of states (control modes) for which the process is active and the members of the project team that developed the process. The following paragraphs describe the purpose of each process.

The arm controller moves the arm during the three arm motions required for the experiment: to home position, to snap position, and spill sweep. In addition, the arm controller reads the encoders and calculates the current position of the arm.

Carrot monitors the position of the platform. The path produced by the off line planner consists of setpoints and via points. The platform moves through setpoints and stops at via points. As the platform approaches a setpoint, carrot provides the next point on the plan to the local planner. As the platform approaches a via point, carrot does not provide the next point on the plan until the platform has attained the position and orientation specified by the via point.

The communications system replicates and distributes the contents of shared memory whenever one of the processes changes the values in a region of shared memory. The database provides information on the a priori knowledge available to HERMIES-III[4]. Displayer updates images from the CCD camera, the range camera, the sonars, and the floor map on a TV screen while the platform is moving.

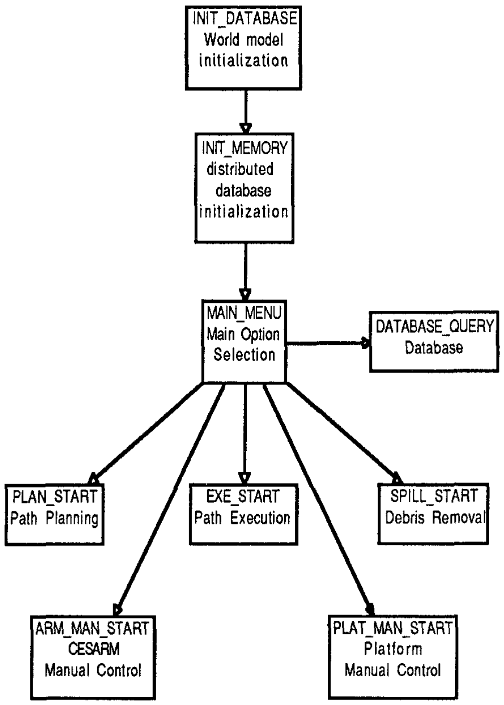

Fig. 2. Control modes for the main menu.

Filter determines the setpoints for the arm controller and the wheel controller. The setpoint depends on the control mode; for example, during the mode PLAT_MAN_LIVE, the setpoint comes from the Joystick Reader; during the mode EXE_LIVE, the setpoint comes from the local planner; and during the mode SPILL_PLAT_LIVE, the setpoint comes from the sweep planner. In addition, filter insures that the setpoints are within the legal limits.

The joystick can be used to control either the CESARm or the platform[5]. The joystick reader reads analog values from the joystick and produces setpoints (in Cartesian coordinates) for the arm or the platform. In addition, the joystick reader provides force feedback commands for the joystick.

	Process	States	Source
1	Arm Controller	7	Texas/ORNL
2	Carrot	1	ORNL
3	Communications	53	ORNL
4	Database	2	Florida
5	Displayer	7	ORNL
6	Filter	10	ORNL
7	Image Snapper	2	ORNL
8	Joystick Reader	6	Texas
9	Local Planner	1	Michigan
10	Off Line Planner	1	Michigan
11	Process Monitor	53	ORNL
12	Range Analyzer	2	ORNL
13	Range Snapper	2	ORNL
14	Sonar Analyzer	1	ORNL
15	Sonar Scanner	1	ORNL
16	Spill Finder	2	Tennessee
17	Sweep Planner	1	ORNL
18	User Interface	22	Florida
19	Wheel Controller	3	ORNL

Table 1. Processes required by the experiment.

The local planner moves the platform, while avoiding obstacles detected by the sensors[6,7]. The inputs to the local planner are the next setpoint from the a priori path and a sensor based map of the environment of the platform.

The off line planner creates a detailed path through a set of via points (determined through the user interface). The off line planner uses an a priori world model that includes obstacles and "redzones", areas that are potentially dangerous and to be avoided.

The process monitor determines the mode of the system (the system has 53 modes). During each mode several processes are active. When all of the processes have been completed, the process monitor changes the mode to the next value in the state transition diagram.

Three sets of sensors can be used during the experiment: the laser range camera, sonar, and video. Each sensor had a process to acquire an image and a second process to analyze the image. Using the image acquired by the range snapper, the range analyzer updates the floor map. The sonar scanner acquires data from the sonar belt that are used by the sonar analyzer to update the floor map. The image snapper acquires an image from the arm mounted CCD camera that is used by the spill finder to map the location of the spill[8,9].

The sweep planner coordinates the spill cleanup. At the start of the cleanup the robot is near the spill. The first step is to move the arm, snap an image, and locate the spill. If the spill finder does not find a spill, the cleanup

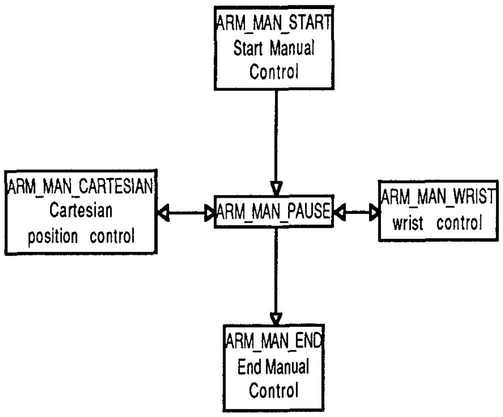

Fig. 3. Control modes for the manual control of the arm.

is completed. If there is a spill, the robot is moved to a corner of the spill and the arm sweeps up the spill. If the spill is large, the platform may be moved several times to clean up the spill. When the sweeping is completed, another image is acquired and analyzed. If any remaining debris is detected, the cleanup continues.

The user interface both allows the user to control the actions of the robot and displays the movements of the robot[10]. The user interface is the primary active process during the main menu (Fig. 2) and during planning. During all motions by the robot, the user interface displays the position of the platform and arm in the context of the current world model.

The wheel controller moves the platform to track the setpoints received from filter. In addition, the wheel controller reads the encoders and calculates the current position and orientation of the platform.

The main menu allows the user to choose one of the six tasks illustrated in Fig. 2. All of the tasks except database query require the sequential execution of several of the 53 control modes. The modes that are active for five of the six tasks are displayed in Figs. 3 to 7: manual control of the arm in Fig. 3; manual control of the platform in Fig. 4; path planning in Fig. 5; path execution in Fig. 6; and spill cleanup in Fig. 7.

Four processes are active for all five of the tasks: communications, filter, process monitor, and user interface. The additional processes that are required for manual control of the arm are arm controller and joystick reader. For manual control of the platform, the additional processes are wheel controller and joystick reader. For path planning, the extra processes are off line planner and joystick reader.

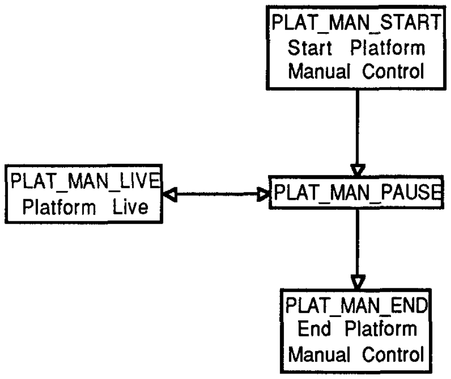

Fig. 4. Control modes for the manual control of the platform.

Path execution requires eight additional processes: carrot, displayer, local planner, range analyzer, range snapper, sonar analyzer, sonar scanner, and wheel controller. The spill cleanup requires six additional processes: arm controller, displayer, image snapper, spill finder, sweep planner, and wheel controller.

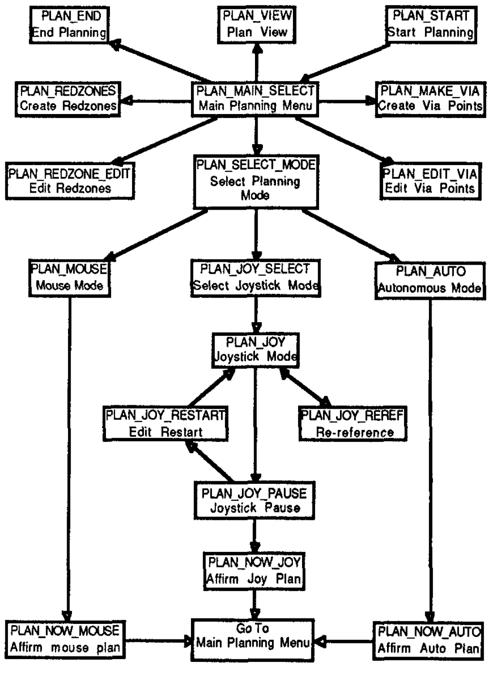

Fig. 5. Control modes for path planning.

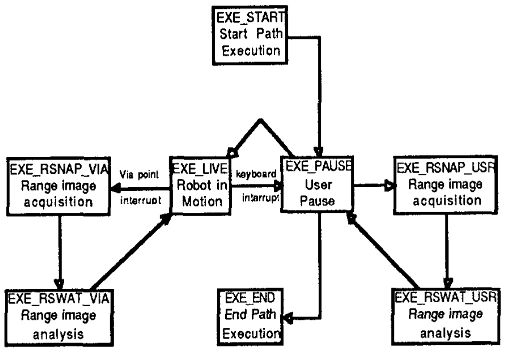

Fig. 6. Control modes for path execution.

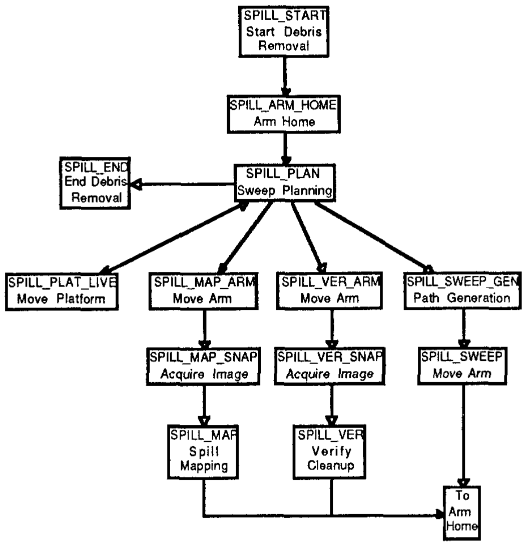

Fig. 7. Control modes for the spill cleanup.

5. Conclusions

The software system described in the previous section was constructed and successfully performed DEMO 89. This section discusses two lessons that we have learned and plan to apply to future experiments. The lessons are to avoid having a single mode variable and to minimize our use of heterogeneous computers.

In the present system, the behavior of the system depends upon a single variable, the mode variable. Almost all processes read this variable to determine the appropriate behavior based on its value. This policy has some advantages and some drawbacks. The primary advantage is that it is extremely simple. A programmer can easily write a program which behaves in a certain way when the mode assumes a certain value. Furthermore, it is straightforward to write a process monitor that determines the next value of the mode variable in all possible circumstances.

There are two disadvantages of relying upon a single mode variable. First, it compromises the reusability of the code for other purposes. In the event that the scenario changes, or that it grows more sophisticated in time, all appearances of the mode variable read in all processes must be guaranteed to be upwardly compatible. The impossibility of a omniscient migration path makes this element of the design ultimately unattractive, The second problem is that a single variable cannot assume more than one value at a time. Although this is exactly what is desired in the finished system, it creates an access conflict during the construction phase. That is, as the project nears completion, teams working on still separate branches require total control of the variable for extended periods of time.

To eliminate these problems, we plan in the next design experiment to replace the mode variable with a state vector. A state vector is a more fine-grained description of the behavior of the system,. Each process will inspect a single variable which describes its desired mode. The primary negative tradeoff for this modification is envisioned to be an increase in the complexity of the process monitor, which will now have to determine a number of variables simultaneously. However, our judgement is that this increase in complexity will be justified, since it impacts only one process.

Since the computers in this experiment involved 24 CPUs in computers delivered by four different manufacturers with five different operating systems, three different architectural styles (serial, shared memory and distributed memory), three different internal representations of numbers, and three different communications technologies, we have accumulated considerable experience with heterogeneous distributed systems during the execution of this project.

In general, our experience has been negative. Each component of the computer system that is different from the others introduces its own collection of idiosyncrasies and increases the time and effort required to build the system. For the solution to a particular problem to be generally available on all stations in a heterogeneous system, it must conform to the least common denominator over all systems. Thus, the use of heterogeneous systems reduces the sophistication of the solution.

Nevertheless, certain principles for adapting to heterogeneous environments are useful. It is desirable to minimize reliance upon operating systems services and other machine specific resources, such as compiler specialization. Any code which contains a call to an operating systems service is non-portable, and should be avoided, or, if it is necessary, isolated into a routine which bring the interface to the operating system out to a commonly available syntax. For example, the routine mem_attach in the present system makes use of operating systems services, but this usage is isolated from applications processes.

6. References

[1] F. J. Sweeney, et al., "DOE/NE University Program in Robotics for Advanced Reactors - Program Plan: FY 1990 - FY 1994," DOE/OR-884/R2, U. S. Department of Energy, Oak Ridge, Tennessee, 1990.

[2] C. R. Weisbin, et al., "HERMIES-III: A step toward autonomous mobility, manipulation, and perception," *Robotica* , vol. 8, pp. 7-12, 1990.

[3] R. V. Dubey, J. A. Euler, and S. M. Babcock, "Real-Time Implementation of a Kinematic Gradient Projection Optimization Scheme for Seven-Degree-of-Freedom Redundant Robots with Spherical Wrists," (submitted to *J. Robotics and Automation),* CESAR-88/36.

[4] G.R. Dalton, et al., "Concurrent Use of Database and Graphics Computer Workstations to Provide Graphic Access to Large, Complex Databases for Robotics Control", Trans. Am. Nucl. Soc., vol. 61, pp. 407, 1990.

[5] J. T. Lovett and P. J. Bevill, "A Universal Bilateral Manual Controller Utilizing a Unique Parallel Architecture," Trans. Am. Nucl. Soc., vol. 61, pp. 409, 1990.

[6] J. Borenstein and Y. Koren, "Real-time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments," 1990 IEEE Int. Conf. Robotics and Automation, pp. 572-577,Cincinnati, Ohio, May 13-18, 1990.

[7] J. Borenstein and Y. Koren, "The vector field histogram: fast obstacle avoidance for mobile robots." IEEE Journal of Robotics and Automation, in press.

[8] C. Chen, M.M. Trivedi, and C.R. Bidlack, "Design and Implementation of An Autonomous Spill Cleaning Robotic System," Proceedings of the Applications of Artificial Intelligence VIII Conference, Orlando, FL, April, 1990.

[9] M.M. Trivedi and C. Chen, "Sensor-Driven Intelligent Robotics," Advances in Computers (Editor: Marshall Yovitts), vol. 30, Academic Press, New York, 1990.

[10] C. D. Crane III, et al., "Faster Than Real Time Robot Simulation for Plan Development and Robot Safety," Trans. Am. Nucl. Soc., vol. 61, pp. 408, 1990.