N88-17266

SYSTEM INTEGRATION of a TELEROBOTIC DEMONSTRATION SYSTEM (TDS) TESTBED

John K. Myers
SRI International, Robotics Laboratory
Menlo Park, California

Abstract

This paper describes the concept for and status of a Telerobotic Demonstration System testbed that integrates teleoperation and robotics. The system is being developed by the Jet Propulsion Laboratories with technical assistance from SRI International. The components of the telerobotic system are described and the projects performed by SRI International are discussed. The system can be divided into two sections: the autonomous subsystems, and the additional interface and support subsystems including teleoperations. The autonomous subsystems are scheduled to be demonstrated separately at the end of 1987, and the entire, integrated telerobotic system is scheduled to be demonstrated at the end of 1988.

Acknowledgments: Other members of the consulting team at SRI International include Ron Cain, Cregg Cowan, Jim Herson, Tony Sword, and Doug Ruth. The project was supervised by Jan Kremers and David Nitzan. I would like to thank the contract monitor, Wayne Zimmerman of JPL, for excellent management and for significant contribution to most of the ideas expressed in this paper.

1 Overview

The Jet Propulsion Laboratories is constructing a telerobotic demonstration testbed system, known as the TDS, where the current state-of-the-art in robotic and teleoperation concepts can be integrated, tested and explored. Although building a space-worthy robot is not an immediate goal, the system concepts and experience obtained will be useful and may eventually be applied to a deployable system.

In order to demonstrate a realistic application, the task of servicing a defective satellite has been selected. The satellite has a defective electronics module contained in a backplane-type slot. As currently envisioned, the repair operation will proceed as follows: The operator will use teleoperation to fold back a flexible foil space blanket and attach its corners to holding tabs. The operator will then place the system in autonomous mode, and instruct the system to replace the electronics module. To accomplish this task, the system will plan and execute motions to unscrew and remove the door, unplug electrical connectors from the electronics module, remove and replace the module, replug the connectors, and replace and rescrew the door.

Figure 1 shows the layout of the TDS setup. Two Unimation PUMA robots perform the actual work on a satellite mockup. A third robot holds a stereo camera pair for the vision-system and for operator feedback. Three additional cameras are positioned on the walls and ceiling of the room. A rack for quick-change tools and spare parts is positioned in front of the robots. The robots are mounted on lathe beds to provide one additional degree of freedom. Figure 2 shows an early version of the actual testbed. Figure 3 shows a solid-model simulation of the testbed, which will be discussed further in a following section.

SRI International is consulting for JPL on the integration of the different parts of the system. The telerobotic system itself is constructed of four subsystems that comprise the main, autonomous part of the robotic system, and three additional subsystems that provide for operator interaction and support the system. The original layout and most of the actual coding of the system was determined and continues to be developed by JPL. SRI is helping JPL to specify the functions of each of these subsystems, and to develop the communication between them Besides this general consulting, SRI currently has four deliverable tasks.

This paper will first discuss the structure of the TDS. Each of the four autonomous subsystems will be defined and examined, followed by the three additional subsystems. After this, the four deliverable projects that SRI is performing will be examined and discussed.

[1]This paper and the consulting work described herein were supported by Jet Propulsion Laboratories under Contract 957908.

PRECEDING PAGE BLANK NOT FILMED

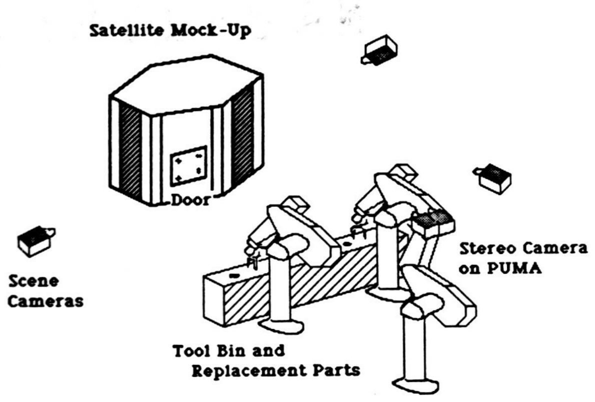

Figure 1: The Layout of the Testbed Setup

2 The Autonomous Subsystems

The autonomous portion of the TDS consists of the following four subsystems: Artificial Intelligence Planning (AIP), Run-Time Control (RTC), Manipulation and Controls Mechanization (MCM), and Sensing and Perception (S&P). See Figure 4. Each of these subsystems runs on its own separate computer. The AIP uses a Symbolics Lisp Machine, while the other subsystems use MicroVax II computers.

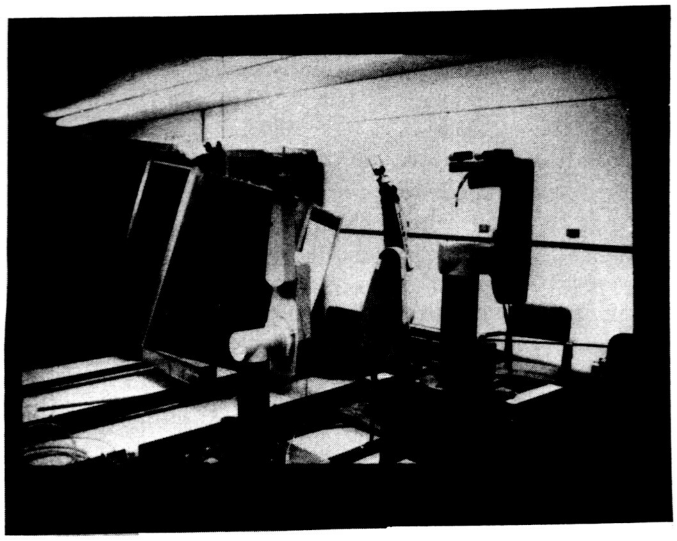

Figure 2: The JPL Testbed

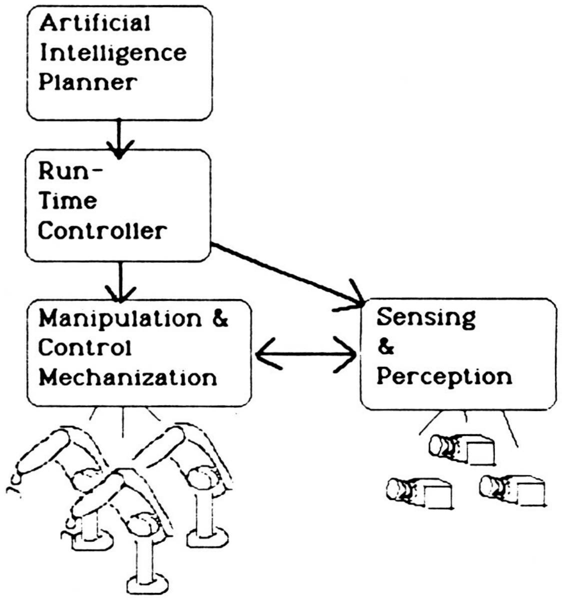
Artificial Intelligence Planner
Run-Time Controller
Manipulation & Control Mechanization
Sensing & Perception

Figure 4: The Autonomous Subsystems

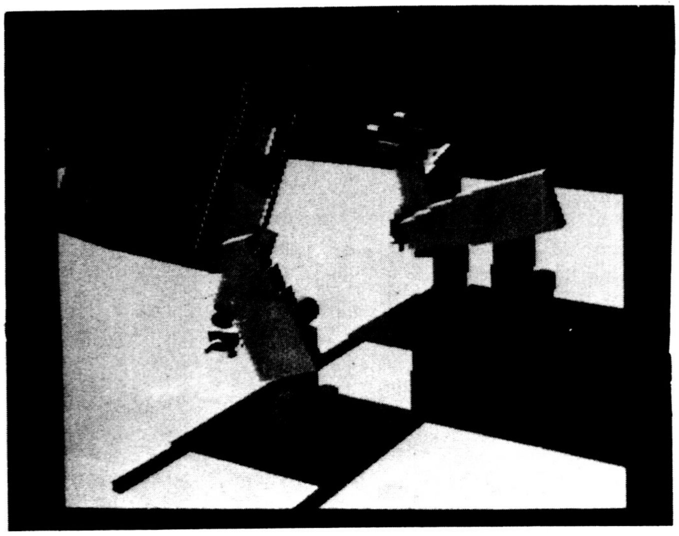

Figure 3: A Simulation of the Testbed

2.1 The Artificial Intelligence Planning Subsystem

The AIP is responsible for planning the actions to perform when the system is under autonomous control. The input to the AIP subsystem is a description of the problem, and a statement of the goal to be accomplished. For instance, in the demonstration task, the input is a description of the state of the satellite, (including the fact that the electronics module is defective), and the goal that the satellite be healthy. The AIP plans the actions to take, and directs the Run Time Control to execute those actions.

Internally, data used by the AIP consists of a rule base describing different conditions and the order in which different procedures must be accomplished. For example: in order to have a healthy satellite, the module must be replaced; in order to replace the module, first the old module must be removed and then the new module must be inserted; in order to remove the old module, the door must be open; in order to open the door, the retaining bolts must be unscrewed; in order to unscrew the bolts, the arm must pick up a nutdriver and then perform the unscrewing action. These rules are represented and stored in a commercial expert system, ART, which is used to reason about what robot system actions must be taken.

The output from the AIP is a series of symbolic actions, such as "Move to above tool bin," "Pick up nutdriver," "Move to above bolt #1," and "Unscrew bolt." These are passed to the Run-Time Control.

2.2 The Run-Time Control Subsystem

The RTC is responsible for instantiating symbolic actions into robotic motions for the system to execute. It takes commands from the AIP, and coordinates the functions of both the MCM and the S&P subsystems. The RTC uses a collision-detection spatial simulator to verify motions. In the future, it will use a collision avoidance module to plan paths around obstructions.

For example, given the symbolic command "Move to above bolt #1", the RTC might perform the following. First, the RTC accesses the location of bolt #1 in a database to instantiate it into a "[4×4]" homogeneous coordinate transformation matrix. If the precise location is not known, or could have changed, the RTC directs the S&P subsystem to verify (or determine) the current location using vision. Next, the RTC uses a predefined "above" distance for that particular bolt to compute the actual location to move to. After this, a "move to above" program is accessed, which may actually contain several individual arm motions, depending on where the robot is at the present time. The RTC executes a predictive collision detection simulation of the proposed motion, to ensure that the robot will not collide with anything when it moves. Finally, the actual instantiated robot system commands are scheduled and sent to the MCM subsystem, and the S&P subsystem if required (e.g., in the previous example).

2.3 The Manipulation and Controls Mechanization Subsystem

The MCM subsystem directs the robots. It takes commands from the RTC, and executes the robot motions on the hardware. The MCM uses force feedback, if required, to modify the motions of the robots. Currently the MCM interfaces with the VAL controllers of the PUMA robots; in the future, the MCM may control the robots directly.

The MCM is responsible for performing atomic (basic, low-level) motions. However, it does have some "reflex macro" motions that are considered to be atomic, but are actually composed of a series of motions. For instance, the robot's nutdriver might be positioned right above the bolt, and the MCM might be instructed to "execute unscrewing motion." The robot lowers the nutdriver onto the bolt head, and "feels around" until the nutdriver is seated. Then, the robot rotates the nutdriver, maintaining appropriate pressure in the direction of the bolt shaft, until the bolt is unscrewed. This "macro" is actually a series of motions. Since this action sequence will always be the same, and is only varied on the fine motion scale based on force sensor readings, the action is considered to be an atomic motion.

2.4 The Sensing and Perception Subsystem

The S&P subsystem is responsible for verifying the locations of objects by using visual feedback. The S&P subsystem has three-dimensional models of all of the viewable parts in the testbed. It uses these models, and an edge image of the scene extracted from the gray-scale image, to perform verification of the position and orientation of parts. The vision system can track moving objects (using a Kalman filter to predict the location of moving objects based on time), use information from multiple camera sources taken at different times, and verify the locations of partially occluded objects. In addition, randomly positioned objects can be searched for and visually acquired; however, this takes significantly longer. See [1] for further details.

The S&P reports its results to the RTC, and also sends results directly to the MCM when requested. An example of a command might be to "verify the location of bolt #1 at approximate location X." The S&P decides the most appropriate camera for viewing that location, takes a picture and computes the edge image, verifies the bolt's location using the visual model of the bolt, and returns the refined location to the RTC.

3 The Additional Subsystems

Besides the autonomous subsystems, there are three additional subsystems that support the TDS and provide important functions. These are: Teleoperations (TELEOP), the Operator's Control Station (OCS), and the System Executive (EXEC). See Figure 5.

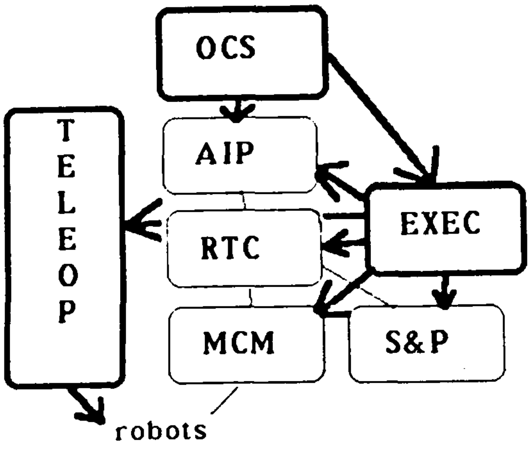

Figure 5: The Additional Subsystems

3.1 The Teleoperations Subsystem

The TELEOP permits the robots to be operated by a person. This allows greater flexibility in actions that can be accomplished, by permitting the operator to perform actions that cannot be done autonomously. The TELEOP controls two Salisbury "hand controllers," which each consist of a handle in a gimbaled cradle attached to the end of a pivoted telescoping shaft. Each of these controllers permits input in six degrees of freedom. The hand controllers' mechanical system are "counterweighted" and use bearings to allow fast and smooth motion. Besides passively providing location as an input to the system, the hand controllers also actively reflect force to the operator's handles as a feedback output. This is used for example to communicate contact at the end of the arm, as detected by the arm's force sensor, and to slow the hand controller down if the operator is slewing too rapidly and the robot arm is lagging too far in its tracking. In the future it will also be used to reflect virtual forces from imaginary "force fields" around modeled objects, and so assist the operator in avoiding collisions [2].

The TELEOP also supports the switchover from autonomous mode to teleoperational mode and back again. In the present design, the two system modes are essentially disjoint. The autonomous system runs by itself and drives the robot arms. At any time, the operator can request or demand a switchover, and the autonomous system either gracefully shuts down, or aborts all actions and returns control to the teleoperator. The teleoperation system then becomes active; the teleoperator can move the arms and remedy any anomalous conditions (such as dropped, wedged, bent or damaged parts), or can perform necessary actions that are beyond the dexterity of present-day autonomous robotics. When the teleoperator is finished, he or she returns control to the autonomous part of the system, and the TELEOP becomes inactive.

An open research issue is the best way of switching from teleoperator to autonomous control. The autonomous subsystems depend on knowing the approximate location of all objects in the system. In both the present-day TDS, and in the eventual space application, this is a reasonable assumption; once the satellite has been acquired and fixed relative to the robots, the robots, tools, and satellite parts become a closed system. After teleoperation, however, the information in the autonomous subsystems' databases may be invalid. Old parts may be unexpectedly moved or missing (e.g., dropped on the floor); the operator could conceivably introduce new parts or sufficiently unfamiliar configurations or modifications of old parts such that they would be unrecognizable by the system.

Assuming that the difficulties are restricted to relocations of old parts, at least four possibilities for solving this problem are being considered. The autonomous system could direct the teleoperator and give instructions as to which parts he or she is allowed to work on. Or, the teleoperator could pick from a menu of standard teleoperation procedures or states to inform the autonomous system of the status of the system. Alternatively, the teleoperator could explicitly tell the autonomous system about each object that was moved. An advanced autonomous system could reinitialize its view of the world by verifying the locations of all expected parts and recognizing the intruding locations of all relocated parts.

Perhaps the best solution would be to convert the system from one that is disjoint between the autonomous and teleoperation modes, to one where the two parts are cooperative and the distinction is blurred. In a futuristic system, the autonomous part of the system would remain on all the time, and "watch over the operator's shoulder" as the teleoperator works. It would observe where the operator is placing parts, so that even during teleoperation, the system would have a full, accurate model of the locations of objects. The autonomous system would also attempt to understand what actions the teleoperator would be performing, and guess what he or she would be trying to accomplish. The autonomous system could then direct additional arms to assist the teleoperator, or take over if a routine task (e.g., unscrewing the bolt) is being performed.

3.2 The Operator's Control Station Subsystem

The OSC subsystem consists of a number of display screens and a keyboard in an ergonomically designed layout. The operator can monitor the status of the system and enter commands for the system to execute. The commands are sent to the AIP, the TELEOP, or the System Executive (to be discussed). The different ordinary and emergency status messages are displayed for the operator; the operator can also obtain views of the work from any of the cameras, and displays of solid models of the testbed describing what the system believes the current location of robots and objects to be. The OCS will be equipped with discrete-word voice input, and voice output for status messages and alarms. In addition, a dual-screen superimposed polarized display will allow three-dimensional viewing to operators wearing polarized glasses. The input for this will probably be taken from the stereo camera pair mounted on the third arm, although it could be computer-generated from solid models of the scene.

3.3 The System Executive Subsystem

The EXEC subsystem is responsible for configuring the system, testing each individual subsystem and the integrated system as a whole, and managing the health of the system. It can suspend and resume the entire system or pieces of the system, and it is also responsible for maintaining an initialization database for the entire system. It manages all of the other subsystems.

The EXEC maintains a library of executable programs for the system. This ensures that the system's software is consistent, and that the different versions of executable files for the different subsystems are kept up-to-date. When the system is initially turned on, the EXEC is responsible for configuring the system. Executable files are downloaded and the system is "brought up" piece by piece. The EXEC also maintains an initialization database, which is downloaded to the different parts of the system once the system is running. These will be discussed in greater detail in following sections.

Once the system is running, the EXEC is responsible for testing each of the subsystems in turn, to ensure that each one is fully configured and capable of running. Each subsystem is also directed to test its hardware, if any, and report the results back to the EXEC. After this, the system as a whole is tested: the AIP is directed to plan a minute, single movement and take a picture. This command is watched as it filters down through the RTC to the MCM and S&P, and as the results are returned via the RTC to the AIP. If everything works properly, then the system as a whole is up and running. Future commands will similarly test the TELEOP and OCS subsystems' operation in the system as a whole. In addition, the EXEC will eventually be able to "watch over the shoulder" of the system as it executes tasks, detect when a computer has become "wedged" or has "crashed," and recover the system from this state.

The EXEC is also responsible for gradual and emergency shutdowns of the system, and the corresponding resumption of execution. There will be several grades of shutdown, depending upon whether the operator wants control when convenient to the system, "soon," or immediately; whether the arms are expected to retreat to a convenient safe position, finish the current process and then stop, or freeze "dead in their tracks;" and whether the computer processes are expected to be able to resume from where they left off, start over, or be completely deleted. System resumption will similarly have to take different forms, depending upon the state of the robots and computer processes.

4 The SRI Projects for the TDS

In addition to general consultation on the design and development of these subsystems, SRI International is providing four deliverables for the TDS. These include: the Network Interface Protocol (NIP), the Robotic Simulator with Collision Detection (RCODE), the System Configuration package, and the System-wide Initialization Database and Editor. The position of these packages in the TDS is shown in Figure 6.

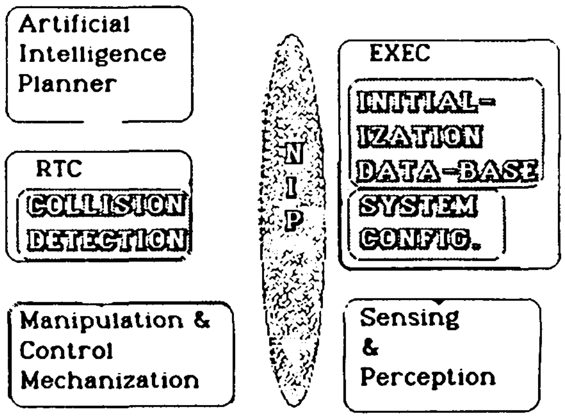
Artificial Intelligence Planner
RTC
COLLISION DETECTION
Manipulation & Control Mechanization
N I P
EXEC
INITIAL-IZATION DATA-BASE
SYSTEM CONFIG.
Sensing & Perception

Figure 6: SRI International Contributions

ORIGINAL PAGE IS OF POOR QUALITY

4.1 The Network Interface Protocol

The Network Interface Protocol (NIP) is a package of software that allows communications between different submodules, with the emphasis on robotic applications. It has been delivered and installed on MicroVax machines, and SRI is currently finishing development of a version for the Symbolics. The NIP is currently implemented on top of DECNET for the Ethernet. However, one of its goals is to separate the implementation of the communications channel from the actual communications themselves. So, if a new technology of communications channel is chosen, none of the subsystems have to be modified; only the implementation-specific levels of the NIP itself have to be changed.

The NIP is specifically designed to facilitate robotic communications, which possess slightly different characteristics from file transfer or mailbox applications. Thus, the NIP must support these characteristics. One difference is timing. In robotics, transactions typically consist of a command issued from a "master" machine to a "slave" machine. The transaction remains open until the slave machine replies to the command. However, instead of replying directly, or in a few tenths of seconds, in typical robotic applications the slave (for instance) might execute a motion with a physical arm and not reply back until the motion has finished, which may require seconds or even tens of seconds—and the delay time may be unknown ahead of time.

Another difference is completion. A robotic motion may be prevented from finishing (especially in the case of force servoing) but still be active, so that it is problematical to state even objectively whether the motion has failed or not. Similar real-world effects exist in vision applications, where (for instance, if presented with a textured pattern, perhaps caused by a bad reflection) an unknown, significantly large number of "blobs" or "edges" can effectively cause the vision system to stop returning answers, while the vision system itself believes that it is properly performing its functions. In addition, robotic applications are notorious for finding unexpected ways to crash the computer they are running on. Problems such as trapping on division by zero or stack overflow, or handling a dropped synchronous communications line to a hardware device, must be detectable and recoverable. Thus, robotic communications must be flexible: they must support transactions that remain open over long periods of time, where it is problematical whether the slave will actually return with an answer or not.

However, other characteristics must be supported. The master might not be able to afford waiting for the return, so the communications must have the option to be pollable: the master can continue processing, and periodically check back to see whether a message has arrived yet or not. Some communications are urgent and should not remain in an input queue, so they should be able to trigger interrupt servicing routines in the application program. With some communications, normal processing cannot proceed until an answer is returned, so the NIP must also support waiting for a response, with an optional time-out clock. Other requirements, such as supporting simultaneous conversations, are too numerous to mention here. The NIP provides such a communication package that is tailored for robotic applications.

4.2 The Robotic Simulator with Collision Detection

The Robotic Simulator with Collision Detection (RCODE) presents a spatial occupancy model of the robots and parts in the scene, that is used to determine whether a collision would occur with a certain movement or not [3]. Objects are modeled using a CSG (constructive solid geometry) system, employing the volume primitives sphere, cylinder, box, and half-space, and the construction operator union (intersection and subtraction are not supported, due to the nature of the algorithms). Device-independent wire-frame and Z-buffer shaded-surface graphics are provided by the system; an example of a model of the TDS is shown in Figure 3. Joint-interpolated, straight-line, and user-specified trajectories are supported. Movement simulation is performed using the stepped-move approach; that is, a single, stationary scene is tested, the moving robots' positions are incremented slightly, and the next scene is tested. The system can test an average scene in about 0.2 seconds on a VAX 750, through use of a hierarchy of enclosing volumes.

The RCODE package has been installed at JPL. It is used by the RTC subsystem to verify arm motions to be sent to the MCM. Other collision detection algorithms, such as that proposed by Canny [4], are also being investigated. Currently, the arm can only move directly to a specified goal location, and the collision detection package verifies the proposed path. In the future, a spatial occupancy simulator may be used by a routine to generate original collision-free paths, such as the one reported in [5]. Collision-free path planning is important because it is the link between artificial intelligence and robotics that allows the system to start to become truly autonomous.

4.3 The System Configuration Package

At startup time, the System Configuration package downloads executable and data files to all of the other computers, and establishes computer processes and communication links in the system, in order to bring the system up. A schematic diagram of the configuration package is shown in Figure 7. The configuration package is currently in the design stage, and will be delivered by the end of 1987.

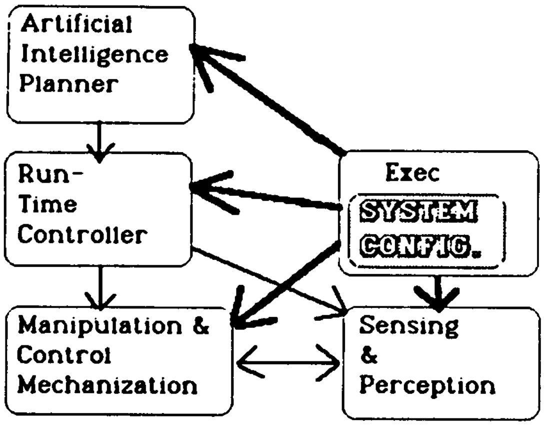
Artificial Intelligence Planner
Run-Time Controller
Exec
SYSTEM CONFIG.
Manipulation & Control Mechanization
Sensing & Perception

Figure 7: System Configuration Package

Configuration actions that the System Configuration package must perform include: downloading executable files, downloading data files, killing all unauthorized processes, starting a process by executing a file, initializing (starting up) a running process by downloading initialization parameters, downloading an initialization run-time data-base to a running process, establishing communication links between running processes, prompting the user and waiting for verification (for such things as turning on a hardware device), testing the status of a process, killing a specified running process, testing a hardware device controlled by a process, testing a communications link between two processes, and allowing an initialized, running process to "take off" and actually perform in the system.

Even given the actions needed to set the system up, configuring a robot system is not straightforward because of dependencies that exist in the order in which the configuration actions must be performed. For instance, in the TDS, the subsystems are arranged in a control hierarchy so that some computers send command messages to other computers. This requires those computers to be running, initialized, and ready to receive those commands, before those commands are sent. Before any system messages can be sent at all, the different communication links must be established between processes. Many of the processes require initialization, or special data-bases to be downloaded to them, after they are running but before they are ready to become part of the system.

In the initial version, the order dependencies will be handled by a programmer creating a command file of configuration actions that is sent to a command interpreter driving the configuration package. Subsequently, a simple backwards-chaining rule-based system will be created, to take a list of dependencies, automatically generate the proper order, and drive the configuration package.

4.4 The System-Wide Initialization Database and Editor

The Initialization Database will be used as part of the system configuration sequence to initialize the system with a given status, i.e. appropriate parts of the database are downloaded to running processes. The Initialization Databas is responsible for the entire system; the database must store all the data used by all of the different subsystems in the TDS. Therefore, the design of the database must be general enough to store all types of data currently used by the system, and to allow for expansion for future types of data that may be introduced. For this reason, SRI is designing a "flavor" based modeling scheme [6] that is able to represent both objects and network relationships between objects.

Each object in the database has a number of slots, each having a name and an indication of the type of information that may be stored there. The permissible types of information are not limited; in particular, the specified type can be a scalar, vector, character, or string, an array, a member of a user-defined set, a pointer to a link, or even a list of items. Objects are connected into networks with directed links; the link is also allowed to have information slots. Slot values for objects and links may be

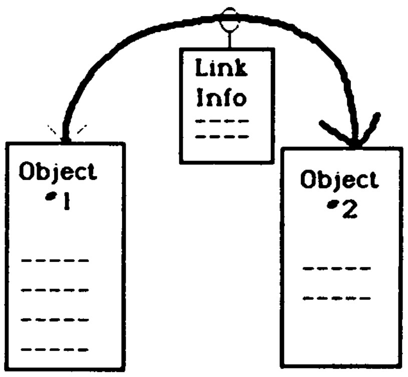

Figure 8: Objects and Network Connections

specified, or they can be defaulted to the normal value for that type. Since objects can be composed of slots of any type, and since arbitrary network structures can be built out of directed links, we anticipate that the design should be general enough for future expansion.

An example object might be the data-base entry for storing the information about a bolt. The bolt has such information slots as name, object type, absolute location, weight, visual model, graphics display model, and spatial occupancy model. It also has links to various networks such as those called relative location, obstructs, virtual enclosing object, articulation (attachment type), and is-an-assembly-of. A relative location link between the bolt and the door for instance might have the slots link-type, forward [4×4] transformation, relative-from-object, backward transformation, and relative-to-object.

In addition to the database, SRI is developing an editor to be used in entering information into the database. Besides the customary adding or deleting an object or modifying an object's slots, users will be able to define and modify object types. Both the editor and the database are expected to be delivered by the end of 1987.

5 Conclusion

This paper has presented a discussion of the current state of JPL's Telerobotic Demonstration System testbed, and the system integration work that SRI International is performing to help realize this testbed. The system was divided into autonomous mode subsystems, and additional subsystems (including Teleoperations); the workings of each subsystem by itself and how the subsystems integrate into a complete system were discussed. Finally, specific deliverables being contributed by SRI were explained. The goal of the TDS is to pull together the current state-of-the-art in teleoperations and different robotics areas. The different autonomous mode subsystems are expected to be demonstrated separately at the end of 1987; the entire system is expected to be demonstrated at the end of 1988.

References

[1] D.B. Gennery. Tracking known three-dimensional objects. In *Proc. AAAI-82*, pages 13–17, 18-20 August 1982.

[2] C.P. Fong, R.S. Dotson, and A.K. Bejczy. Distributed microcomputer control system for advanced teleoperation. In *Proc. 1986 Int. Conf. on Robotics and Automation*, IEEE, April 1986.

[3] J.K. Myers. A robotic simulator with collision detection: rcode. In *Proc. of First Annual Workshop on Robotics and Expert Systems-1985*, pages 205–213, Houston, Texas, June 1985.

[4] J. Canny. Collision detection for moving polyhedra. *IEEE Trans. on PAMI*, PAMI-8(2):200–209, March 1986.

[5] J.K. Myers and G.J. Agin. A supervisory collision-avoidance system for robot controllers. In *Robotics Research and Advanced Applications*, pages 225–232, American Society of Mechanical Engineers, Phoenix, Arizona, 1982. Republished in Robotics World, Vol. 1, No. 1, January 1983.

[6] *Symbolics Common Lisp: Language Concepts.* Symbolics, August 1986.