N90-25563

The Application of NASREM to Remote Robot Control

Michael W. Walker, Joe Dionise, Al Dobryden

Artificial Intelligence Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109

Abstract

This paper describes the implementation of a remote robot controller, wherein the distance to the remote robot causes significant communication time delays. The NASREM telrobot control architecture is used as a basis for the implementation of the system. Levels 1 through 4 of the hierarchy were implemented. The solution to the problems encountered during the implementation and those which are unique to remote robot control are described.

1 Introduction

Remote robot control is becoming an increasingly important discipline in robotics. Undersea exploration, mining, and salvaging operations, and the inspection and maintenance of nuclear power plants are examples of terrestrial based applications. Space applications include the construction of large space structures, the in situ maintenance and repair of satellites, and the exploration of the planets and their moons. This relatively new area of robotics is characterized by large distances between the operator and the remote robots.

A testbed has been built for identifying and finding solutions to problems unique to this type of system. The operator station is located at Grumman Aerospace Corporation in Bethpage, New York. The remote robots are located at the University of Michigan, Ann Arbor, Michigan. Thus, the system is truly a remote robot control system.

This paper presents the design of the testbed. Although, the testbed has just been completed, some problems which have already been identified will also be discussed. We begin with a description of the control architecture. This gives a framework from which to describe various components in the system and to specify their modes and methods of interaction. The following sections present some of the details of the construction of the system. The final section concludes the paper.

2 Control Architecture

We have adopted the NASREM architecture for telerobot control [1]. The reference document defines the functional requirements and flight level specifications of the control system for the NASA Space Station IOC Flight Telerobot Servicer. We have used this document as a guideline for the development of the control system architecture. However, the physical remoteness of the robots begin controlled requires some extensions to this architecture. The problem basically stems from the communication delays between the local site where the operator is stationed and the remote site where the robots are located. This section presents a brief description of the NASREM architecture and the required extensions.

2.1 The NASREM Architecture

The NASREM architecture is a six level hierarchical control system as shown in figure 1. The outputs of each level are the inputs to the next lower level in the hierarchy. The inputs at each level of the hierarchy are:

1. Level 6 - Operations Control Level
   Inputs are commands to schedule the servicing of satellites.
2. Level 5 - Service Bay Control Level
   Inputs are commands to a service bay manager to perform operations on specific spacecraft.
3. Level 4 - Object/Task Level
   Inputs are commands to perform a task on an object in order to achieve a desired relationship of that object relative to other objects in the world.
4. Level 3 - E-move Level
   Inputs are symbolic names of "elemental" movements (E-moves), typically expressed as commands to achieve "key-frame" poses in coordinate system of choice.
5. Level 2 - Primitive Level
   Intermediate trajectory poses which define a path which has been checked for obstacles and is guaranteed free of collisions and kinematic singularities.
6. Level 1 - Servo/Coordinate Transfer Level
   Inputs are evenly spaced trajectory points for manipulators, grippers, transporters, and sensor platforms in a con-

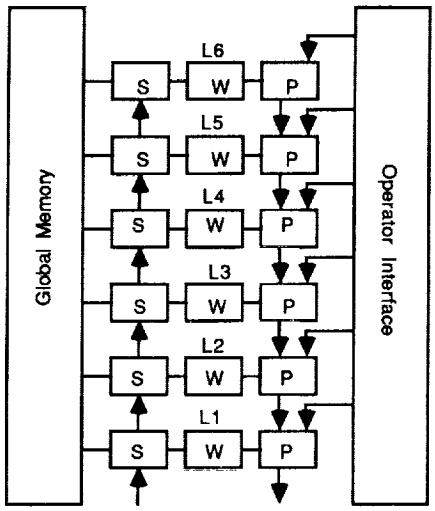

Figure 1: NASREM Telerobot Control Architecture

venient coordinate frame.

The output from Level 1 are commands to physical devices such as D/A converters. As one moves up the hierarchy the commands become less specific and more mission oriented.

At each level of the control hierarchy there are three different types of modules, the sensory processing module (S), the world modeling module (M), and the task decomposition system (H). The sensory processing system is used to sense physical processes in the FTS system. The world model uses this information to estimate the state of the system and to predict future states. Also, the world model is used to evaluate plans which are produced by the task decomposition system.

The operation of the task decomposition system at each level is characterized by an H-module. These modules convert higher level task descriptions into sequences of lower level subtask. Thus, the operation of the controller is basically defined in terms of the operation of the task decomposition system, or H-modules. Each H-module has three components: the Job Assignment, JA, which partitions the input task command into distinct jobs to be performed by physically distinct mechanisms; Planners, PL, which convert each job created by JA into a sequence of subtasks; and the Executors, EX, which execute the subtasks created by the planners.

2.2 Extensions to NASREM

To handle the problem of remoteness of the robots, we have divided NASREM into two parts which are called: Local NASREM, and Remote NASREM, Figure 2. The remote component is identical to standard NASREM except that the operator interface is replaced by a remote communications link. The local component contains the operator interface as well as an additional world modeling system and a local planning system. A significant difference is the addition of the world modeling system at the local site which is used for planning. This addition is important as the long communication delays imply more recent knowledge about the robot's world at the remote site than is available for planning at the local site.

Communication to the remote site can occur at any level down to level 3 of the hierarchy. Below that level the feedback loops are of such a high bandwidth to preclude closing the loop at the local site.

The final difference is the introduction of the Level 0 controller to indicate hard wired systems below Level 1 of the NASREM hierarchy. For example, a Level 0 controller is used to control the torque produced by the manipulator actuator.

The function of levels 0 through level 3 at the local site is to simulate the future operation of the remote site at levels 0 through 3 for the purpose of operator evaluation of plans produced by level 4 at the local site. In the current system, only level 3 of levels 0 through 3 at the local site are in place. Thus, the operator can view motions planned before the commands are sent to the remote site. However, because of the lack of the remaining levels at the local site, the operator can not preview forces which will be generated at the remote site.

The current configuration of the system is shown in figure 3. Only those components drawn in solid have been implemented. The following sections describe the design of these components of the system.

3 Local - Level 4

The language TROL (TeleRObot Language) was developed for operator input to Level 4. It was implemented using the *yacc* compiler-compiler which is available on all UNIX systems. The language provides a simple way of commanding motions to the robots.

The problems we have encountered are mainly the result of our limited view of the functionality of the robots at the remote site. In developing a robot programming language one is inclined to define a language in which only the motion of a robot can be specified. In retrospect, the language should specify the the way in which changes are to be made in the robot's environment. The language should only implicitly specify the motion of the robot. This would not only make it easier to integrate higher levels in the control hierarchy, but it would also simplify the operation of the servo controller at the remote site. As discussed in section 6 several control laws are available for selection. Examples include adaptive and compliant motion control laws. The appropriate selection should be based upon the particular task to be performed by the manipulator. In other words, the type of changes which are to be made in the robot's environment.

4 Local - Level 3

Level 3 of the hierarchy converts key-frame poses obtained from level 4 into sequences of positions which are free of kinematic singularities and collisions. We have found there are few solutions to this problem, and they are all computationally intensive [2].

It is much easier to determine if a given path is free of collisions than it is to numerically determine a collision free path. Therefore, our approach to this problem is to display the pose of the system at each key-frame pose. After viewing these po-

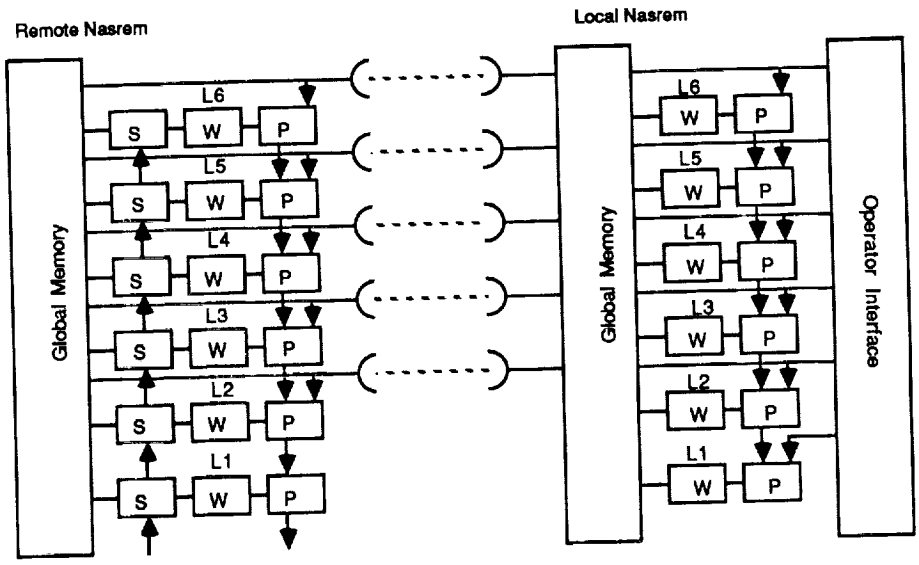

Figure 2: Extended NASREM

sitions, if a collision has occurred, the operator is compelled to create a new plan. He has to reenter a new program at level 4. Thus, the difference between the operation of our system and the NASREM system is the requirement of human interaction to provide collision free paths.

5 Remote - Level 2

Level 2 control is a simple a path interpolator [3]. Trajectories are composed of sequences of straight line segments connected together by smooth arcs. The positions obtained from Level 3 specify the end points of these line segments. For a trajectory having m segments, the $m-1$ intersection points of every two adjacent segments are denoted by the 6×1 vectors $\boldsymbol{p}_i$, $i = 1..m-1$. Accordingly, $\boldsymbol{p}_0$ and $\boldsymbol{p}_m$ denote the initial and final points, respectively.

During the execution of the $i-th$ segment, the manipulator is either in transition from the $i-1st$ segment, or on the $i-th$ segment, or in transition to the $i+1st$ segment.

The acceleration is choosen to be zero when the manipulator is in the middle of a segment and a constant non-zero value during transition to or from an adjacent segment. Since the acceleration is zero along the middle portion of a segment, the velocity, $\boldsymbol{v}_i(t)$, is constant. For example, in the middle of the $i-th$ segment

$$\boldsymbol{v}(t) = \boldsymbol{v}_i$$

where

$$\boldsymbol{v}_i = (\boldsymbol{p}_i - \boldsymbol{p}_{i-1})/T_i$$

T_i is the time duration required to go from points $\boldsymbol{p}_{i-1}$ to $\boldsymbol{p}_i$. During the transition from one segment onto another the acceleration is

$$\dot{\boldsymbol{v}}(t) = \dot{\boldsymbol{v}}_i$$

where

$$\dot{\boldsymbol{v}}_i = (\boldsymbol{v}_{i+1} - \boldsymbol{v}_i)/(2\tau_i)$$

$2\tau_i$, the time required to accelerate between segments i and $i+1$.

The equation describing the position of the manipulator is:

$$\frac{d\boldsymbol{p}}{dt} = \boldsymbol{v} \quad (1)$$

$$\frac{d\boldsymbol{v}}{dt} = \dot{\boldsymbol{v}} \quad (2)$$

where $\dot{\boldsymbol{v}}$ is either the null vector, corresponding to the time interval where the hand is in the middle of a segment, or $\dot{\boldsymbol{v}}_i$ corresponding to the time interval when the hand is in transition from the $i-1st$ segment.

The position and velocity of the hand are computed as a function of time by integrating Equations 1 and 2. Since the acceleration during any one sample period is always constant, this equation can be most easily solved by converting it into a sampled data equation. The resulting equations are:

$$\boldsymbol{p}(k+1) = \boldsymbol{p}(k) + \boldsymbol{v}(k)\Delta t + 1/2\dot{\boldsymbol{v}}(k)(\Delta t)^2 \quad (3)$$

$$\boldsymbol{v}(k+1) = \boldsymbol{v}(k) + \dot{\boldsymbol{v}}(k)\Delta t \quad (4)$$

where k represents the $k-th$ sample time, and Δt is the sample period.

Although Equations 3 and 4 are exact equations for computing $\boldsymbol{p}(k)$ and $\boldsymbol{v}(k)$, the procedure does suffer from round off errors and will tend to drift from the correct position and velocity. Thus, the numerical values of $\boldsymbol{p}(k)$ and $\boldsymbol{v}(k)$ are corrected for errors at the beginning of each straight line segment.

Note that the acceleration, velocity and position along the initial starting and the final stopping segment can also be specified by using this method. To do this, an additional segment of zero length is appended to the beginning and another to the end of the trajectory. This results in an initial and final velocity of zero and a start up time of $2\tau_1$ and a stopping time of $2\tau_{m-1}$.

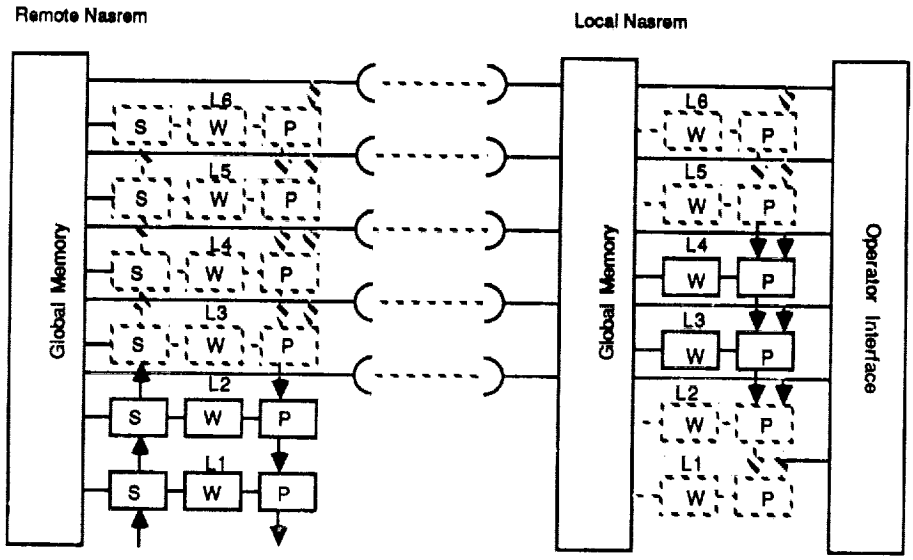

Figure 3: Current System Configuration

6 Remote - Level 1

The manipulator is modeled as a graph structured system consisting of $m+1$ links, $r \geq 0$ closed kinematic loops, and n primary joints whose motions are independent of the remaining joints. Note that if $r = 0$ then $n = m$, and the manipulator is a serial link manipulator. However, we have found that true serial link manipulators are nearly nonexistent. Even the PUMA 560 manipulators we use, which are usually considered serial link manipulators, are really graph structured systems, due to the internal gearing.

We use the spatial notation popularized by Featherstone, [4, 5, 6] for the modeling and control law formulation. This notation is convenient not only for analysis purposes, but also in simplifying the implementation of the controller. Since Ada was used in writing the software of the controller, we can overload the operators such as multiplication and addition to apply to the spatial quantities defined by Featherstone. Thus, the syntax of the software developed for the controller is very similar to the notation used in the analysis of the robot dynamics and control.

6.1 Kinematic Constraint Equations

Since, in general, the manipulator is graph structured, there exist constraints in the relative position of each of the joints of the manipulator. These constraints can be represented by a set of equations of the following form:

$$q = U(Q) \tag{5}$$

where Q is an $n \times 1$ vector containing the positions of the primary joints. The primary joints are a set of joints such that their position uniquely determines the position of all other joints in the manipulator. Note that, for each Q_i there is a q_j such that, $Q_i \equiv q_j$.

Since there are n primary joints, there must be at least n actuators in the manipulator. If the manipulator has more than n actuators it will be redundantly actuated [7]. That is, there would be more actuators than are required for motion. Although it is acceptable to choose the primary joints to be those where the motor armatures are located, it is not a requirement. Other joints are usually better. For example, the six primary joints of the PUMA 560 manipulator are those located between the end-effector and the base of the manipulator. Thus, all solutions of the inverse kinematics problem for the PUMA 560 have been presented for these joints of the manipulator. However, motor number four is kinematically coupled to joints four, five and six. Thus, changing any of these joint positions causes motion of the fourth motor.

Given $U(Q)$, $\dot{Q}$, and $\ddot{Q}$, the velocity of all the joints can be determined.

$$\dot{q} = E(Q)\dot{Q} \tag{6}$$

where

$$E(Q) = \frac{\partial U(Q)}{\partial Q} \tag{7}$$

The acceleration is given by:

$$\ddot{q} = E(Q)\ddot{Q} + \dot{E}(Q)\dot{Q} \tag{8}$$

For a given manipulator these equations are usually fairly simple. Often the matrix $E(Q)$ is constant. For example, it is a constant 6×6 matrix for the PUMA 560 manipulator. However, for some manipulators these equations can become complex. For these manipulators we have found that the ϵ-algebra is very convenient for evaluating the time derivatives of the joint positions [8]. Using this algebra one only has to program the solution to Equation 5, change the order of the algebra and automatically obtain the solution to equations 6 and 8.

6.2 Feedback

Feedback signals used by the control come in two forms: spatial feedback vectors, $\hat{F}_j$, which are associated with the links of the manipulator and scalar feedback functions, $\hat{\eta}_j$, which are associated with the joints in the manipulator. These can be any functions of the desired state and the actual state of the manip-

ulator. The method used to calculate the required primary joint forces is as follows.

1. The values of the spatial feedback vectors, $\hat{F}_j$, are computed for each link, and feedback functions, $\hat{\eta}_j$, are computed for each joint of the manipulator.

2. The primary joint forces, τ_i, are computed using the $\hat{F}_j$ and $\hat{\eta}_j$ as in the equation:

$$\tau_i = \sum_{j=1}^{n+r} (e_{ji}\hat{\eta}_j + s'_{ji}\hat{F}_j)$$

where e_{ji} is the $ji-th$ column of the matrix $E(Q)$, and s_{ji} is the spatial velocity of link j due to a unit velocity of the $i-th$ primary joint. An algorithm for the efficient evaluation of the τ_i has been presented, [6].

Thus, the controller simultaneously encorporates feedback at the joint level and at the Cartesian level. The choice of the $\hat{F}_j$ and $\hat{\eta}_j$ dictate the character of the controller being implemented. For example, adaptive controllers for a single manipulator [6] and for dual arms [9] have been presented.

An important property of this controller is that regardless of the choice of these feedback functions, they all satisfy the following property.

$$\sum_{j=1}^{m+r} (\tilde{v}'_j(F_j + I_j g) + \dot{\tilde{q}}_j \eta_j) = \sum_{j=1}^{m+r} (\tilde{v}'_j \hat{F}_j + \dot{\tilde{q}}_j \hat{\eta}_j) \qquad (9)$$

where F_j is the actual inertial force on link j, I_j is the spatial moment of inertia matrix, g is the spatial acceleration due to gravity, η_j is the joint j friction force, the $\dot{\tilde{q}}_j$ are any joint velocities which are consistent with the constraint equations and $\tilde{v}_j$ is the resulting link spatial velocity. This property is very useful for showing stability of the some feedback functions. Basically, it is simply a restatement of the principle of virtual work. If the $\dot{\tilde{q}}_j$ are the same as the actual joint velocities, $\dot{q}_j$, then the term on the left is the rate at which energy is absorbed by the manipulator and the term on the right is the rate at which energy is delivered to the arm by the actuators. In fact, by thinking of the $\hat{F}_j$ and $\hat{\eta}_j$ as inputs to the system, equation 9 is simply another form of the equations of motion. That is, the equations of motion are:

$$\sum_{j=1}^{m+r} (s'_{ji}(F_j + I_j g) + e_{ji}\eta_j) = \sum_{j=1}^{m+r} (s'_{ij}\hat{F}_j + e_{ij}\hat{\eta}_j) \quad i = 1 \ldots n$$

Thus, Level 1 of the controller calculates the spatial feedback functions $\hat{F}_j$ and the scalar feedback functions $\hat{\eta}_j$, converts them to equivalent primary joint torques, τ, and outputs τ to the Level 0 controller. As described in the next section, the Level 0 controller converts these to the equivalent motor actuator torque, and then controls the current in the motor to attain that desired torque.

The software of the controller is organized so that adding new feedback functions only requires the user to add two new procedures called, **forward** and **backward**. At each sample period, these procedures are called once for each link of the manipulator. The procedure **forward** is used to update any local state information such as local digital filters that are used by the feedback function. The procedure **backward** evaluates and returns the values of the spatial feedback functions $\hat{F}_i$ and the scalar feedback functions $\hat{\eta}_i$. Again, the particular feedback function used is dependant upon the task being performed by the manipulator. For example, adaptive control requires different feedback than compliant motion control.

7 Remote - Level 0

Level 0 is the hard wired control of physical devices to make them look like the model of the devices used in the formulation of the controller at level 1. For the case of the manipulator controller, all level 1 controllers assume they are controlling a manipulator with motors at the primary joints. However, because of the use of gearing in most manipulators, The motor armatures are not located at the joints chosen as primary joints. The primary joint positions and the motor armature positions are related by the following equation.

$$Q = R(q_m)$$

where q_m are the motor armature positions. Given τ from the level 1 controller, the level 0 controller converts this to an equivalent motor torque by the equation:

$$\tau_m = B(q_m)^T \tau$$

where $()^T$ indicates the transpose of the matrix and

$$B(q_m) = \frac{\partial R(q_m)}{\partial q_m}$$

The level 0 controller then drives the current through the DC motor used for actuation to produce the desired motor torque. There are two types of motor actuation systems that could be used. The PUMA 560 manipulator is an example of the first type. In this system the only sensory signals are the current flowing through the armature and the position of the motor. It is an indirect control of the torque produced by the motor. The torque is controlled by directly controlling the currents through the motor armatures. The torque is the assumed proportional to this current. The Robotics Research Arm [10] is an example of the second type. This arm has a sensors to measure the output torque produced motor in addition to the motor position. The control of torque in the Robotics Research Arm is a direct control since the output torque is directly measured and fed back to the input of the motor.

8 Hardware

The connections to the Level 0 control are through the same umbilical utilized by the standard PUMA 560 to connect to the VAL II system controller. Thus, the Level 0 hardware, in effect, replaces the controller hardware which comes standard with the PUMA 560. Between the PUMA 560 and the Level 0 controller hardware is the robot interface box. This box contains all the electronics specific to the manipulator being controlled. It contains the arm power supply, the joint amplifiers, the encoder signal conditioning circuitry, and the brake release circuitry. Digital signals between a VME system and the robot interface box use differential transmitters and receivers for noise immunity, and to allow a large distance between the two systems. The joint amplifiers are PWM-type power amplifiers. The signals from this box are then directly connected to the digital I/O, and D/A and

A/D converts resident in the VME chasis. A Motorola 68020 based single board computer is used to implement the Level 0 controller algorithms. All software written for this computer is in the *C* programming language. *C* was chosen because of the low cost of the compiler (Alycon Corp.), the speed of the *C* language and the simplicity and relatively small size of the software developed at this level.

Level 1 and 2 of the controller were implemented on a Compaq 386/20 computer using Ada as the programming language. The interface between Level 1 and Level 0 controls is through a 64k shared memory block. Thus, the Compaq simply writes out torque commands by writing to a specific location of the shared memory block and the Motorola 68020 presents the sensory information such as primary joint positions and velocities by writing to the same shared memory block. Thus, a very fast communication mechanism is used between these two levels.

The world modeling system and levels 3 and 4 of the local system reside on a Silicon Graphics 4D-GT graphics work station. This graphics work station is fast enough to simultaneously present a graphic display of the current operation of the remote robots and also a second display of the plans that are being developed for future operation of the robots. The only real video that is used is though a video telephone, which transmits pictures at about a 5 second frame rate. The real video is only used by the operator for verification of the operation of the remote robots.

9 Remote Communication

The current system allows commands to be sent to the remote robots via two transmission mediums : the Internet Network and the common telephone line. This section describes this implementation.

9.1 The Internet Transmission Control Protocol

For hosts on the Internet Network, the Internet Transmission Control Protocol (TCP) provides a convenient medium of passing commands between the manipulator and the controlling agent. At its highest level of abstraction, the TCP protocol provides a potentially reliable, sequenced, full-duplex connection-based byte-stream. This communication protocol is referred to as the socket stream. A socket stream must be connected before any data can be sent or received on it. Hence, the controlling agent who wishes to control the manipulator, must first make a request for connection. If the manipulator grants the request, then a full-duplex stream of commands may be passed between the two agents. These commands are encoded into fixed sized command packets and then sent over the socket stream. The agent on the opposite end of the socket stream, will then decode the command packet and take the appropriate action. When the session is over, both agents will close their end of the socket stream, and the manipulator will reset and listen for further connection requests.

9.2 Serial Socket

For those controlling agents not on the Internet Network, a method of communication over the standard phone lines was developed. At the heart of this serial communication is a protocol dubbed the *serial socket*. The serial socket is a protocol which implements socket-like attributes over a standard RS-232 serial line. Specifically, the serial socket provides a potentially reliable, sequenced, full-duplex byte stream over a serial line, or with the aid of a pair of modems, over a standard telephone line. Similar to the socket stream, the serial socket must be connected before any data can be sent or received on it.

The controlling agent requires minimal hardware and software :

- A computer running Unix BSD 4.2 or Unix System V.
- A 2400-baud Hayes-compatible modem supporting the full AT-command set.
- The serial socket and associated software.
- A reliable phone line.

Connection with the manipulator is achieved in a similar manner as the socket stream, except that the controlling agent must first utilize the serial socket software to dial the phone number of the telephone at the remote site. If the connection request was granted, then a full-duplex stream of commands may be passed between the two agents over the serial socket.

As mentioned above, the serial socket shares the high level attributes of the socket stream. To achieve this level of abstraction, a three layer communication protocol was developed. The lowest layer of the protocol directly interacts with the serial port. Routines in this layer set the baud rate of the port, read bytes from the port, write bytes to the port, etc. Similar to the Kermit protocol, the serial socket library makes only minimal assumptions about the serial port over which the transfer occurs; namely that the port is capable of sending and receiving all printable ASCII characters. It also requires that the system be able to send and receive a SOH control character. Most Unix systems provide this facility by incorporating the serial port as a special file.

The middle layer in the serial socket communication protocol involves the transmission and reception of fixed size packets over the serial port. By default, the size of these packets is 128 bytes. Included in the packet are fields for the packet type, sequence number, encoded data, and check value. To meet the requirement mentioned above, only printable ASCII characters are allowed to reside in the packets. To this end, the binary data is encoded to a purely printable form when a packet is written to the serial port, and decoded back to binary when it is read. This mapping is the same as that used in the Kermit file transfer program. To detect errors during the transmission of a packet, a check value field is included in each packet. By default, a 16-bit Cyclic Redundancy Check (CRC-16) is used. The CRC is good at detecting all kinds of errors (single-bit, double-bit, odd-numbered, etc), but especially those that occur in bursts over a relatively long time.

The topmost layer in the serial socket communication protocol implements the automatic repeat request (ARQ) packet protocol. An error detected in a received packet or an unacknowledged packet automatically results in the retransmission of that packet. A high level description of this protocol follows. During transmission, the binary data is packetized by surrounding it with service fields. The entire packet is then transmitted with no flow control, after which the sender waits for the receiver to acknowledge its receipt. The receiver inputs the packet and, after

verifying that the packet is in the correct sequence, computes a local checkvalue on the data portion of the packet. If this checkvalue matches the one in the packet, the receiver acknowledges by sending an acknowledge (ACK) packet. If the packet was in error, then a negative acknowledge (NAK) packet is transmitted. Upon receipt of an ACK, the sender transmits the next packet; if a NAK was received, then the same packet is transmitted again. Transmission proceeds in this manner until the serial socket is closed.

10 Conclusion

The design of a testbed for the control of a remote robotic system has been presented. The NASREM control architecture was used as a basis for the system development. Because of the remoteness of the robots some extensions to the NASREM architecture were proposed.

In the current operation of the system we have observed that the limitations of our system basically stem from the deficiencies of our communications language and the *side effects* of commands.

The communications problem stems from our basing the communications scheme on an enumerate of all possible commands to the remote system. We are now in the process of designing a system which uses the Ada programming language as the communications language. Thus, we are, in effect, using an Ada interpreter for the implementation of level 4 at the remote site. This will give us the capability of doing such things as defining a variable for the location of an object and then referring to the location of the object through the name of that variable. Thus, the local site need not know the exact location of the object. It can refer to this exact location which is stored at the remote site through the variable it defined for that purpose. Also, by using Ada as a basis for communications we will derive all the benefits of a well defined language. Any person who knows Ada could program the robot.

The other main problem with our control is the problem with side effects. That is , many commands which are issued to the remote site may or may not have side effects which are different that the primary effect intended for that command. An example is the command *Close Gripper*. The primary effect of this command is to actuate the fingers which may or may not cause something to be grasped. If something is grasped, then there will be a side effect in future motions of the end-effector. Specifically, the position of the grasped object will change with a change in position of the end-effector. In addition, if the object is attached to another object, as for example, a door handle connected to a door, then as the manipulator end-effector is moved, the location of the door is moved. Whats more, this side effect places constraints on the motion of the end-effector. The controller designer must include these constraint effects in his controller design.

Our current effort is directed towards providing a communications language which better describes the desired changes in the robots environment and to modify the robot controller to carry out those desired changes.

Acknowledgement

This work was supported by a grant from the NASA sponsored Center for Autonomous and Man-Controlled Robotics and Sensing Systems at ERIM, Ann Arbor, MI.

References

[1] J. S. Albus, H. G. McCain, and R. Lumia, "Nasa/nbs standard reference model for telerobot control system architecture (nasrem)," Technical Report NASA:SS-GSFC-0027, National Bureau of Standards, March 13 1987.

[2] T. Lozano-Perez, "A simple motion-planning algirithm for general robot manipulators," *IEEE J. Robotics and Automation*, vol. RA-3, pp. 224–238.

[3] R. P. Paul, *Robot Manipulators: Mathematics, Programming and Control*, MIT Press, Cambridge, Mass, 1981.

[4] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *The Int. J. of Robotics Res.*, vol. 2, no. 1, pp. 13–30, Spring 1983.

[5] R. Lathrop, "Constrained (closed-loop) robot simulation by local constraint propagation," In *I.E.E.E. Int'l Conference on Robotics and Automation*, San Francisco, CA, 1986.

[6] M. W. Walker, "An efficient algorithm for the adaptive control of a manipulator," In *I.E.E.E. Int'l Conf. on Robotics and Automation*, Philadelphia, Pennsylvania, April 1988.

[7] J. Gardner, V. Kumar, and J. Ho, "Kinematics and control of redundantly actuated closed chains," In *I.E.E.E. Int'l Conf. on Robotics and Automation*, Scottsdale, Arizona, May 1989.

[8] M. W. Walker, "Manipulator kinematics and the epsilon algebra," In *I.E.E.E. Int'l Conf. on Robotics and Automation*, Raleigh, North Carolina, 1987.

[9] M. W. Walker, D. Kim, and J. Dionise, "Adaptive coordinated motion control of two manipulator arms," In *I.E.E.E. Int'l Conf. on Robotics and Automation*, Scottsdale, Arizona, May 1989.

[10] "Specification for type 1 & type 2 motion controller servo-level interface," Technical Report OPS SLI-02888-1A, Robotics Research Corporation.