Software architecture for a kinematically dissimilar master-slave telethesis for exploring rehabilitation applications

Joseph Wunderlich, Shoupu Chen, Daniela Pino, Tariq Rahman, William Harwin

Applied Science and Engineering Laboratories
Alfred I. duPont Institute/University of Delaware
Wilmington, Delaware USA

ABSTRACT

A person with limited arm and hand function could benefit from technology based on teleoperation principles, particularly where the mechanism provides proprioceptive-like information to the operator giving an idea of the forces encountered in the environment and the positions of the slave robot. A test-bed system is being prepared to evaluate the potential for adapting telemanipulator technology to the needs of people with high level spinal cord injury. This test-bed uses a kinematically dissimilar master and slave pair and will be adaptable to a variety of disabilities. The master will be head controlled and when combined with auxiliary functions will provide the degrees of freedom necessary to attempt any task. In the simplest case, this mapping could be direct, with the slave amplifying the person's movements and forces. It is unrealistic however to expect that the operator will have the range of head movement required for accurate operation of the slave over the full workspace. Neither is it likely that the person will be able to achieve simultaneous and independent control of the 6 or more degrees of freedom required to move the slave. Thus a set of more general mappings must be available that can be chosen to relate to the intrinsic coordinates of the task. The software structure to implement the control of this master-slave system is based on two IBM PC computers linked via an ethernet. Each computer runs a real-time software kernel that is responsible for; the closed loop control of the master-slave; the messages exchanged via the ethernet; the forward and inverse kinematics; and the set of mappings from master to slave. The software structure simply provides hooks for each of these tasks and the flow of information. This allows a variety of control laws and modes to be implemented and evaluated on the test-bed. Several major problems remain with a system of this nature. Since the problem is essentially that of integrating different systems there have been multiple constraints, the first being the choice of the IBM family of computers as a control platform. The advantages are their low hardware costs and the range and availability of software, the disadvantages are the limitations of the operating system and conflicts in both the memory space and the input/output space of the hardware. A second major limitation is that almost without exception software from different vendors is provided in the C language. Although this has the supposed advantage of speed, it has poor visibility when attempting to debug and test a real-time system and has none of the safety features that Cullyer et al. identified for safety critical systems [Cullyer 91]. These remain as problems and a hardware watchdog system has been implemented that attempts a safe system shutdown in the event of software errors.

2. INTRODUCTION

The research conducted for this project involves the investigation of control theories and real time control hardware and software to lead to the development of a useful rehabilitation robot for individuals with upper spinal chord injuries. The complexity of replacing human movement and manipulation with electromechanical analogues is well known. In 1962, Professor Rajko Tomovic addressed the First International Symposium on the Application of Automatic Control in Prosthetic Design [Reswick90]. He observed:

- Human extremities represent a very complex kinematic system whereas engineering actuators have relatively few degrees of freedom.
- The many degrees of freedom of our extremities require control signal sources of high information content.
- The control signals used are of two distinct origins: one set consists of conscious control signals, whereas the second governs the reflex loops. These signals are of sensory origin.

The control of manipulators by people with severe disabilities has been addressed by a number of research efforts and has been mostly viewed as a problem of limited communication bandwidth. The control signals available to operate the manipulator are reduced due to the limitations imposed by the disabling condition. The aim of this research is to develop interface strat-

0-8194-1322-4/93/$6.00

egies that have not been possible with previous rehabilitation robot approaches. The target is to allow the user to control the robot to:

- Move rapidly through a complex trajectory
- Touch a surface with a desired force
- Increase or decrease the force
- Follow a surface or contour
- Sense where the hand is without using visual feedback.

The function of the proposed system is demonstrated by the example of picking up a glass of water. The user can:

- Direct the robot to the glass located in an unstructured environment
- Instruct the robot to grasp the glass
- Apply an increasing amount of force - the glass, due to gravity, will resist until sufficient force is a applied, thus giving the user a sense of the weight of the glass and liquid
- Move the glass into an appropriate position for drinking.

The control interface will be designed so that the capabilities of the robot cannot be exceeded by the human. This is important if the robot and human are to act in unison and experience the same (or scaled) kinesthetic information.

3. CONTROL THEORY

Early teleoperated devices were mechanically-linked, kinematically similar master and slave manipulators [Goertz63] or man-amplifier systems such as Handyman [Mosher60]. These systems evolved into the more sophisticated teleoperated robot systems which are more dexterous and allow the operator to be physically separated from the slave. Hirtzinger [Hirtzinger89] found that dissimilar master and slave manipulators are possible and the system can be controlled through high-level commands which make it convenient for the user. In a conventional telemanipulation system, each joint of the (slave) robot is controlled by a very stiff position servo. Consequently, the human operator has to control a very stiff system. While a stiff telerobot is well-suited to operation in free-space, when it comes into contact with objects, walls and obstacles, excessive forces result. Kinesthetic feedback dramatically improves the performance of the teleoperated system. This involves feeding back the forces and moments sensed at the end-effector to the human operator [Hannaford91]. This allows the implementation of bilateral control [Inoue71]. The master controls the slave and at the same time disturbances at the slave affect the motion of the slave as well as the master. More generally, the operator should be able to control the position (and orientation) of the slave robot while feeling the forces/moments exerted by the slave. This is referred to as telepresence [Sheridan86]. The dynamics of the environment are transferred to the human operator so that the human operator feels as though he/she is directly interacting with the environment [Hagner89].

Proprioception can be broken down into two components, one conveying position of a limb and the other force [Mann74]. A brief description of the two forms of proprioception and how they are inherent characteristics of the control schemes follows.

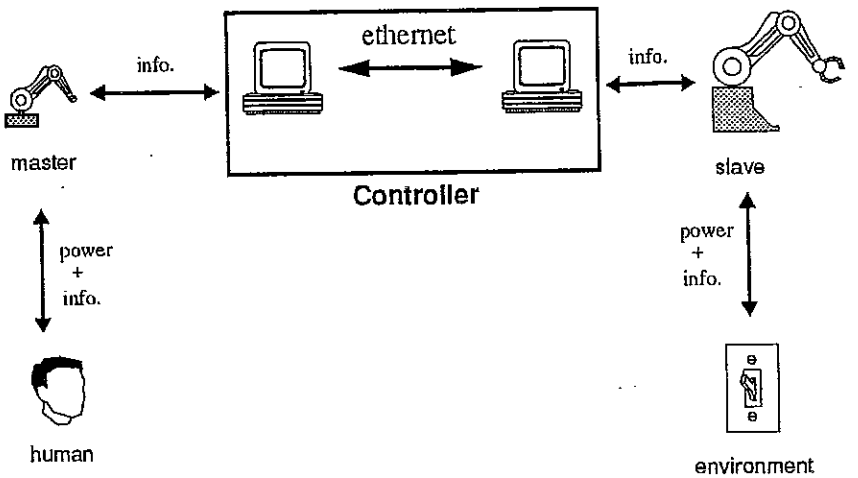
Position Proprioception: Humans are capable of reaching points in space without looking at their hand. This is partly achieved through receptors in the joints and musculature that convey absolute positional information to the central nervous system through a complex mapping. Artificial sensors that are capable of transmitting similar information from robotic links are an impossibility at this point. The alternative is to extend proprioception of residual limbs out to the artificial limb. This control technique is called "extended physiological proprioception" [Simpson77]. The movement of the human's residual limb (head) is tied in a unique relationship to the movement of the artificial limb (robot hand).

Force Proprioception: Just like receptors that provide position information, receptors exist that inform of forces encountered by the extremities [Willis91]. The forces in question could either arise from dynamic interaction between limbs, or through contact with the environment. Hogan [Hogan89] stated that the variable to be controlled in human-machine interaction is not force or position but a relationship between the two, or impedance. Further, he presents a unified theory for unconstrained and constrained motion, which models the robot as an impedance, and which calls for a specification of the force produced in response to a motion imposed by the environment.

3.1 Control schemes

The general control scheme for this project is shown in Figure 1

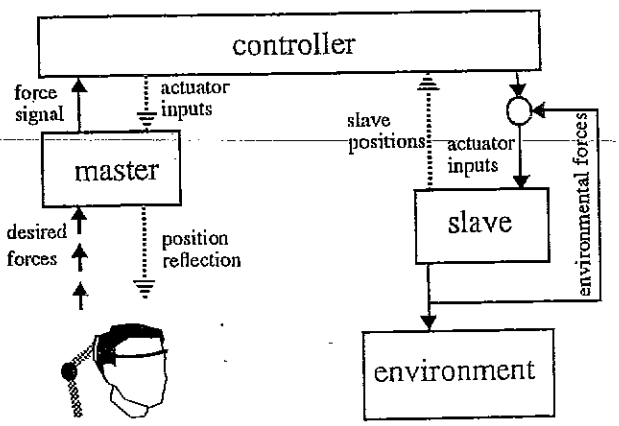
FIGURE 1. Test-Bed Control Scheme.



The human feels a representation of the robot and the environment through the master interface. This exchange of signals in both directions results in a closed loop system. Two specific control schemes that are being investigated are outlined below.

3.1.1 Force control with position reflection

FIGURE 2. Position Reflection Scheme

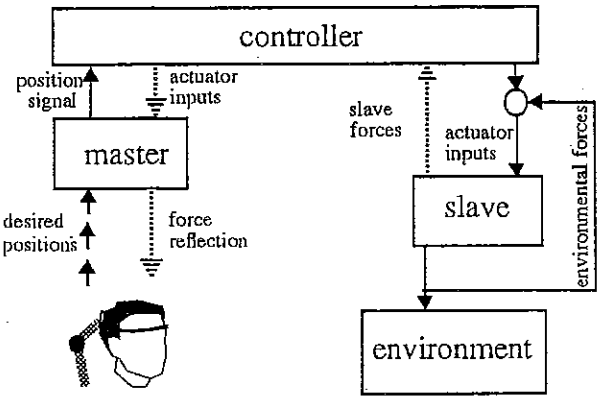


This proposed scheme, Figure 2, allows the user to input forces measured with a 6-axis force/torque sensor at the master which is initially rigid. These forces act as inputs to the slave robot motors. This would be the feed-forward component, whose counterpart in neurophysiology would be efferent, or motor pathways. The sensory feedback, or afference, would be attained through reflection of the slave position on to the master. The master would provide actual slave positional information to the person, while accelerations and forces at the slave robot tip would be felt by the user through increased demands on force.The above scheme can be used for motions in both free space (unconstrained) and contact tasks (constrained). When the user applies a force to the master, the slave moves if there is nothing impeding its path. A similar movement is felt by the user through movement in the master. However, if an obstacle is present in the path of the slave, it will stop moving and cause the master to stop its motion, thereby providing the human with knowledge of the obstacle. The servo-loop for the slave robot is a force control loop. Because of this, the force applied (felt) by the robot is directly related to the force exerted (perceived) by the human force. Further, the force applied by the robot can never reach large values because the operator explicitly specifies this force.

3.1.2 Position control with force reflection

This scheme is used primarily to reflect environmental forces back to the user. The servo-control loop for the slave is a position control loop and the human operator commands the position of the slave through the master input device.

FIGURE 3. Force Reflection Scheme



As the slave arm moves through space, any forces and torques experienced by the slave end effector will arise due to the dynamics of the slave. These forces will depend primarily on slave link masses and inertias. The operator, through the back-drivable master link, will feel scaled and filtered versions of these forces. Thus the operator receives feedback on the energetic interchange that provides a sense of effort required in unconstrained manipulation. If motions involving contact are to be executed then the forces of interaction between the environment and the slave robot will be the primary signals fed back to the master link.

Both control schemes proposed offer force and information exchange between the operator and the slave robot. The user thereby feels forces and positions experienced by the robot through the interface. The advantage of the first control scheme is that it offers a unified scheme that would include constrained, as well as unconstrained movements. Tasks that involve environmental interaction would be more stable because the variable under operator control is force, and not position as in the second scheme.When applied to tasks performed in an unconstrained environment, such as point-to-point motions or reaching tasks, the position control, force reflection scheme (scheme 2) offers a more robust control scheme [Funda90]. The disadvantage of scheme2 becomes apparent in contact tasks when the interaction forces between the robot and the environment may instantaneously become high. If the slave robot is operated through a position loop, contacts with the environment will result in impacts and possibly impulsive forces. Only after sensing the high impulsive forces is the operator aware of the environmental constraint. Although both control schemes are to be investigated, scheme 2 is presently being implemented in the test-bed and is therefore the focus of the software architecture presented in this paper.

4. AUXILARY INPUTS

Since all of the head movements of the user are being used for positioning the end effector of the slave robot, auxiliary inputs are needed for controlling the gripper, emergency stopping of the system(s), and normal system power up/down . The following auxiliary input systems are being investigated:

- Voice recognition system
- Mouth controlled devices (e.g., sip and puff device, tongue touch pad, ect.)
- Shoulder controlled device
- Chin controlled device
- Eyebrow controlled device
- Electromyograph (EMG) device

5. SYSTEMS INTEGRATION

The test-bed is being constructed using a Zebra ZERO slave robot and a PER-Force master. The Zebra ZERO is a six degree of freedom robot equipped with a force/torque sensor at the wrist and a simple parallel jaw gripper. The actuators are D.C. servo-motors controlled by HP HCTL-1100 motor control chips which receive feedback form incremental optical encoders and send out pulse width modulated signals to drive the motors. This is closed loop position control using the position control mode of the HCTL chips (i.e. the HCTL velocity profiling modes are not used). The PER-Force is a small backdrivable six degree of freedom robot which can measure 3 linear positions(x, y, and z) and 3 attitudes (roll, pitch, and yaw). The controller is being built from two Intel 486 based systems connected through a dedicated 8 mbit/sec ethernet link. The programming language for both the master and slave is C.

The advantages of the PER-Force master are:

- Force-feedback
- Device designed for human interaction
- Device can be moved even when control loops are inactive (i.e., for safety in the event of power failure)
- Hardware watch-dog system for safe shutdown
- Low inertia
- Gravity compensation

The advantages of the Zebra slave robot are:

- Force/torque sensing capability integrated into the control loop
- A "force control mode" which allows the robot to push the end-effector with a specified bias force
- Robot-specific library functions (e.g., Inverse kinematics, Jacobian)
- A reach of 0.63 meters, weight of 12 kg, and payload of 1 kg. (Ideal for proposed application)
- End-effector speeds of up to 40 in/s are (Comparable to human speeds)

The disadvantages of the Zebra slave robot are:

- Arm collapses when power is lost
- Lack of software visibility
- Mechanical slippage in the gearing results in backlash and poor repeatability

Both the master and the slave operate in a DOS environment. Although DOS is not intended as a real-time operating system, the DOS interrupt routines and DOS keyboard management can be used as real-time features. Also, DOS provides many debugging tools.

6. SOFTWARE ARCHITECTURE

6.1 Master

As shown in Figure 4, the control software routine running on the master will first check if an exit flag has been set by software errors or by the user (for example an emergency stop). The routine then acquires force data felt by the slave. This data is then mapped from the cartesian coordinates of the slave's end-effector in base-frame coordinates (previously transformed on the slaves computer from the slave's force sensor cartesian frame {wrist frame}) to the baseframe coordinates of the master (homogeneous transform). The compliance of the master is then calculated from this transformed force data and the master's current position data.

A safety log file is shown in Figure 4 to be generated after the compliance is calculated. However, since this file is generated mainly for debugging purposes, it can be specified to be generated at any appropriate point in the routine. The data specified to be captured into this file also varies depending on the program run.

Lastly, before the routine loops, the master's position data is mapped to desired slave end-effector positions and sent to the slave over the ethernet.

5. SYSTEMS INTEGRATION

The test-bed is being constructed using a Zebra ZERO slave robot and a PER-Force master. The Zebra ZERO is a six degree of freedom robot equipped with a force/torque sensor at the wrist and a simple parallel jaw gripper. The actuators are D.C. servo-motors controlled by HP HCTL-1100 motor control chips which receive feedback form incremental optical encoders and send out pulse width modulated signals to drive the motors. This is closed loop position control using the position control mode of the HCTL chips (i.e. the HCTL velocity profiling modes are not used). The PER-Force is a small backdrivable six-degree of freedom robot which can measure 3 linear positions(x, y, and z) and 3 attitudes (roll, pitch, and yaw). The controller is being built from two Intel 486 based systems connected through a dedicated 8 mbit/sec ethernet link. The programming language for both the master and slave is C.

The advantages of the PER-Force master are:

- Force-feedback
- Device designed for human interaction
- Device can be moved even when control loops are inactive (i.e., for safety in the event of power failure)
- Hardware watch-dog system for safe shutdown
- Low inertia
- Gravity compensation

The advantages of the Zebra slave robot are:

- Force/torque sensing capability integrated into the control loop
- A "force control mode" which allows the robot to push the end-effector with a specified bias force
- Robot-specific library functions (e.g., Inverse kinematics, Jacobian)
- A reach of 0.63 meters, weight of 12 kg, and payload of 1 kg. (Ideal for proposed application)
- End-effector speeds of up to 40 in/s are (Comparable to human speeds)

The disadvantages of the Zebra slave robot are:

- Arm collapses when power is lost
- Lack of software visibility
- Mechanical slippage in the gearing results in backlash and poor repeatability

Both the master and the slave operate in a DOS environment. Although DOS is not intended as a real-time operating system, the DOS interrupt routines and DOS keyboard management can be used as real-time features. Also, DOS provides many debugging tools.

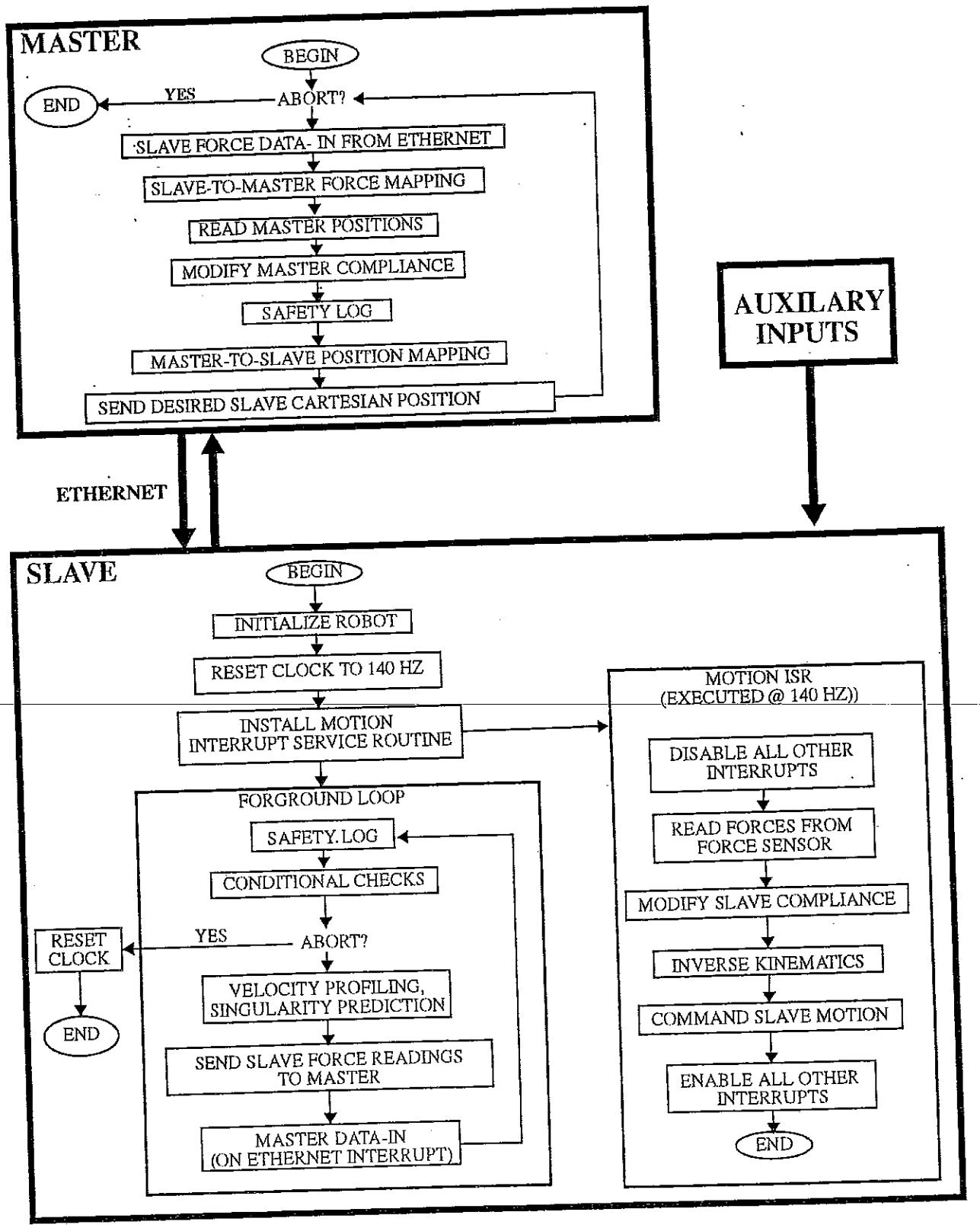
6. SOFTWARE ARCHITECTURE

6.1 Master

As shown in Figure 4, the control software routine running on the master will first check if an exit flag has been set by software errors or by the user (for example an emergency stop). The routine then acquires force data felt by the slave. This data is then mapped from the cartesian coordinates of the slave's end-effector in base-frame coordinates (previously transformed on the slaves computer from the slave's force sensor cartesian frame {wrist frame}) to the baseframe coordinates of the master (homogeneous transform). The compliance of the master is then calculated from this transformed force data and the master's current position data.

A safety log file is shown in Figure 4 to be generated after the compliance is calculated. However, since this file is generated mainly for debugging purposes, it can be specified to be generated at any appropriate point in the routine. The data specified to be captured into this file also varies depending on the program run.

Lastly, before the routine loops, the master's position data is mapped to desired slave end-effector positions and sent to the slave over the ethernet.


MASTER
BEGIN
ABORT?
YES
END
SLAVE FORCE DATA- IN FROM ETHERNET
SLAVE-TO-MASTER FORCE MAPPING
READ MASTER POSITIONS
MODIFY MASTER COMPLIANCE
SAFETY LOG
MASTER-TO-SLAVE POSITION MAPPING
SEND DESIRED SLAVE CARTESIAN POSITION
AUXILARY INPUTS
ETHERNET
SLAVE
BEGIN
INITIALIZE ROBOT
RESET CLOCK TO 140 HZ
INSTALL MOTION INTERRUPT SERVICE ROUTINE
FORGROUND LOOP
SAFETY.LOG
CONDITIONAL CHECKS
ABORT?
YES
RESET CLOCK
END
VELOCITY PROFILING, SINGULARITY PREDICTION
SEND SLAVE FORCE READINGS TO MASTER
MASTER DATA-IN (ON ETHERNET INTERRUPT)
MOTION ISR (EXECUTED @ 140 HZ))
DISABLE ALL OTHER INTERRUPTS
READ FORCES FROM FORCE SENSOR
MODIFY SLAVE COMPLIANCE
INVERSE KINEMATICS
COMMAND SLAVE MOTION
ENABLE ALL OTHER INTERRUPTS
END

6.2 Slave

As shown in Figure 4, the slave's software routine begins by initializing the robot arm's end-effector to a ready position, the force sensor's readings are offset to compensate for gravity and all of the robot's parameters (e.g., damping, force thresholds, ect.) are set to default settings. The system clock is then set to 140Hz for an interrupt service routine (ISR) that is then installed and executes every 7.14 milliseconds. This ISR commands all of the slave's motion (i.e. joint angle displacements). Since 7.14 msec is a relatively short period of time for a high level language program including library routines (kinematics, ect.) to execute, only the most necessary code is contained within the ISR. Therefore, the modifying of slave compliance and the inverse kinematics of desired cartesian slave displacements (previously mapped in master slave routine and received over the ethernet) are the only tasks which have been specified to be executed within the ISR. These two tasks may be combined if necessary to save execution time. (i.e., calculate compliance with respect to joint angles). These tasks have been specified by redefining a library function which is called by the ISR. The other tasks of the ISR shown in Figure 4 (e.g., read force sensor, command slave joint angles, ect.) are not transparent. However, the force readings are put into global variables and therefore can be used in compliance calculations. Although the slave robot comes with a "stiffness control mode" it does not perform conventional compliance calculations. Therefore, code has been written to implement conventional compliance calculations.

The ISR is not interruptible; therefore, the reading in and sending out of ethernet data is performed outside of the ISR (i.e., in the foreground loop). The result of this is that the sampling rate of the master and slave can be adversely affected by the limited time remaining in the foreground loop between the completion of one ISR call and the start of the next. This is because the foreground loop is spawned and continues running and looping while the ISR's interrupt and execute every 7.14 milliseconds.

The foreground loop, other than being a message passing center for ethernet data, is also charged with performing the following tasks:

- Predict and avoid singularity points along a trajectory
- Profile the velocity to allow smooth start-up and slow-down of a slave robot motion
- Manipulate auxiliary input data for control of the gripper and the power supplies

7. MASTER-SLAVE COMMUNICATION

Both the master and the slave robot need to communicate with each other sending either position or force information. Since the paths for both robots have to be created in real time, one of the main goals of this project has been to reduce delays in the communication process. With this in mind, the decision was made to connect the PCs using an ethernet mainly because of its portability.

The ethernet hardware consists of western digital network cards and a thin net cable dedicated link. Thin net has the advantage that the information packets to be sent between the robots are relatively small, and most importantly, transceivers which might introduce delays in the communication process are not needed.

The ethernet software initially choosen was a standard PC-TCP software package and its software development kit. In order to initiate the communication process between both computers, Client/Server Model was choosen. The ethernet software has been written to use the connectionless Universal Datagram Protocol (UDP). Despite the reliability offered by the connection oriented Transport Control Protocol (TCP), UPD was selected because of its speed and because a very small percentage of lost packages can be expected in a dedicated link. The next issues to be addressed were the issues of using non-blocking v.s. blocking primitives, and buffering v.s. non-buffering primitives. Since both robots must be running asynchronously, and the robot's paths are being created using interrupt service routines, blocking primitives would be unacceptable. A send request that blocks until the package has been transmitted, or a request to read that waits until a package arrives, would cause a Interrupt Service Overrun. To avoid this, non-blocking primitives where used. To avoid losing packages, local buffering was used for both the transmit and the receive requests. When trying to integrate the robot and ethernet software, several problems where encountered; The motion interrupt service routine for the Zebra Zero robot was written with the assumption that the only other interrupt that was going to be generated after the ISR had been installed was a keyboard interrupt to terminate and remove the motion ISR. Because of these conflicts with the interrupts, another network driver (8003pkdr) was used which services the network interrupts faster than the wd8003 driver. The 8003pkdr comes with four C interface functions which allow the user to write his/her own network software to send IP packages to any machine. These four C interface functions are represented below in Z notation [Spivey89]. This notation is used as the specification of all of the project's software. Z notation allows a

compact representation that can be readily adapted to design changes. The notation for the ethernet link seperates the master, where only the position information is available from the slave. Thus, we have:

$$FORCE \subset R^6$$
$$POSITION \subset R^6$$
$$AllowableForce \subset FORCE$$
$$AllowablePosition \subset POSITION$$

The State Space for the Master is then defined as

LocalMasterData

$$\Delta MasterPositionHistory : \mathbb{P}\ POSITION$$
$$NewPosition? : POSITION$$

$$NewPosition \in AllowablePosition$$
$$MasterPositionHistory' = MasterPositionHistory \cup NewPosition$$

and

RemoteSlaveData

$$\Delta SlavePositionHistory : \mathbb{P}\ POSITION$$
$$\Delta SlaveForceHistory : \mathbb{P}\ FORCE$$
$$NewPosition? : POSITION$$
$$NewForce?' : FORCE$$

$$NewPosition \in AllowablePosition$$
$$NewForce \in AllowableForce$$
$$SlavePositionHistory' = SlavePositionHistory \cup NewPosition$$
$$SlaveForceHistory' = SlaveForceHistory \cup NewForce$$

and for the Slave as

LocalSlaveData

$$\Delta SlavePositionHistory : \mathbb{P}\ POSITION$$
$$\Delta SlaveForceHistory : \mathbb{P}\ FORCE$$
$$NewPosition? : POSITION$$
$$NewForce? : FORCE$$

$$NewPosition \in AllowablePosition$$
$$NewForce \in AllowableForce$$
$$SlavePositionHistory' = SlavePositionHistory \cup NewPosition$$
$$SlaveForceHistory' = SlaveForceHistory \cup NewForce$$

and

RemoteMasterData

$$\Delta MasterPositionHistory : \mathbb{P}\ POSITION$$
$$NewPosition? : POSITION$$

$$NewPosition \in AllowablePosition$$
$$MasterPositionHistory' = MasterPositionHistory \cup NewPositon$$

Force and position are encoded and decoded into ethernet packets, and each packet is treated as a single natural number sent in as a single transaction between the master and the slave. The following is the State Space for encoding and decoding the messages on the ethernet:

$$
\begin{array}{|l}
\hline
\textit{EthernetPackets} \\
\textit{Message} : \mathbb{P}\ N \\
\textit{DecodedMessage} : \textit{Message} \nrightarrow (\textit{NewForce} \bigvee \textit{NewPosition}) \\
\textit{EncodedMessage} : (\textit{NewForce} \bigvee \textit{NewPosition}) \nrightarrow \textit{Message} \\
\hline
\textit{Message} = (\textit{dom}(\textit{DecodedMessage}) = \textit{range}(\textit{EncodedMessage})) \\
\hline
\end{array}
$$

Data is exchanged over an ethernet link which must be initialized in the first instance, and closed once the master and slave are shut down. The initialization sequence that must happen and be confirmed before the control loops can be instantiated is as follows:

$$
\begin{array}{l}
\textit{Handle} \in N \\
\textit{BufferSize} \in N \\
\textit{BufferLimit} \in N \\
\textit{EtherInitREPORT} ::= \textit{InitializationSuccess} \mid \textit{InitializationFailure}
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{EtherInit} \\
\textit{MaxBufferSize?} : \textit{BufferSize} \\
\textit{EthernetHandle!} : \textit{HANDLE} \\
\textit{Result!} : \textit{EtherInitREPORT} \\
\hline
((0 < \textit{MaxBufferSize} \leq \textit{BufferLimit}) \bigwedge (\textit{EthernetHandle} > 0) \bigwedge \\
\qquad (\textit{Message} = \emptyset) \bigwedge (\textit{Result!} = \textit{InitializationSuccess})) \bigvee \\
((\textit{EthernetHandle} \leq 0) \bigwedge (\textit{Result!} = \textit{InitializationFailure})) \\
\hline
\end{array}
$$

For the 8003pkdr network driver, BufferLimit = 1526 bytes.

$$
\begin{array}{|l}
\hline
\textit{EtherCleanup} \\
\textit{EthernetHandle?} : \textit{Handle} \\
\hline
\end{array}
$$

The schemas for exchanging data are defined as:

$$
\begin{array}{l}
\textit{Channel} \in N \\
\textit{RemoteData} ::= \textit{MasterRemoteData} \mid \textit{SlaveRemoteData} \\
\textit{EtherSendREPORT} ::= \textit{SuccessfulSend} \mid \textit{UnsuccessfulSend} \\
\textit{EtherReceiveREPORT} ::= \textit{SuccessfulReceive} \mid \textit{UnsuccessfulReceive}
\end{array}
$$

EtherSendPkt

$\Delta EthernetPackets$
$Buffer? : Message$
$EtherChannel! : Channel$
$Result! : EtherSendREPORT$

$((EtherChannel = Buffer) \wedge (EthernetPackets' = EthernetPackets \cup Buffer)$
$\quad \wedge (Result! = SuccessfulSend)) \vee$
$(Result! = UnsuccessfulSend)$

EtherReceivePkt

$LastMessage? : Message$
$LastMessageStatus? : MessageStatus$
$EtherChannel! : Channel$
$Result! : EtherReceiveREPORT$

$((LastMessage' = EtherChannel) \wedge (RemoteData) \wedge$
$\quad (Result! = SuccessfulReceive)) \vee$
$(Result! = UnsuccessfulReceive)$

These schemas do not specify the action to be taken on an unsuccessful send or receive, only that it occurs! Finally it can be noted that the control loops at this point can be simply specified as a set of functions where the current control law maps the input space to the output motor torques, $\tau \subset R^6$
where $\{MasterPositionHistory, SlavePositionHistory, SlaveForceHistory\} \rightarrow \tau$.

8. SAFETY

The dual robot system design will have three levels of safety in addition to the robot panic buttons. The first will be to use software verification methods as typified by Z notation and implemented by structured programming methods. A "watch dog" circuit will be included to ensure that run time errors in the software initiate an emergency stop procedure. Thirdly, the mechanical link between the master and the person will have a quick release facility and endstops to ensure that the manipulandum always remains within the person's range of movement. Issues related to software are discussed below.

8.1 Software in a safety-critical system

A rehabilitation robotics system can be classified as a safety-critical system as defined in McDermod's paper [McDermid91]. In such a system, software cannot directly cause loss of life or injury but it may control the robot. Thus, software can contribute to the safety of a system. Unlike hardware, software can fail frequently but still not lead to unsafe behavior, if the failures do not cause hazardous consequences. On the other hand, reliable software can be unsafe, if in the rare event of failure there are catastrophic consequences. Further, it is not easy to predict future performance of the system from the past performance since software behaves discontinuously, i.e. a small change in inputs can cause a very large change in outputs since the control flow in the program depends on the input values. Therefore there is considerable risk associated with using software in safety-critical system, and it is common practice to avoid the use of software in such applications as rehabilitation robot systems. However, it is impractical to achieve the requisite functionality without making software a critical component of the system.

8.2 The application of the formal methods to the development of software

There are five typical stages in the development cycle of software [McDermid91]:

1) Requirement analysis; description of the system and its operational environment
2) System specification; description of the system input, the system outputs and their relationships without describing internal system structure
3) Architectural design; a high level internal view of the structure of the system
4) Detailed design; details of algorithms and data structures needed to implement the system
5) Implementation; the program source code

The formal methods of the software design are classified as below. [McDermid91]

- Model-based methods; providing an explicit definition of system state and operations that transform the sate (no explicit representation of concurrency)
- Algebraic methods; providing an implicit definition of operations by relating the behavior of different operations without defining state (no explicit representation of concurrency)
- Process algebras; providing an explicit model of concurrent processes and representing behavior by means of constraints on allowable observable communication between the processes
- Logic-based methods; using logic to describe properties of systems including low-level specification of program behavior and specification of system timing behavior
- Net-based methods; providing an implicit concurrent model of the system in terms of data flow through a network including representing conditions under which data can flow from one node in the net to another

In this project the Model-based method is considered; in particular, the Z notation is to be used in the development cycle of software. The Z language is based on the set theory and first-order predicate calculus. A unique feature of Z is the use of schemas and the schema calculus. Schemas are 'modules' of specifications, and the schema calculus gives a way of linking the modules to build up complex specifications from simple parts in a clear manner.

8.3 Computer languages considerations

In an application such as rehabilitation robotics system, the role of the computer is to accept inputs from various sensor systems, perform some calculations or logic, and then provide output to actuators. It is well understood that the designer of critical systems should know the consequences of a component's failure in service. Given the requirements for safety and the required coupling to formal methods of specifications and verification, the designer is able to choose a language which meets the required safety standards and is also compatible with the existing system componants. In [Cox88], it is suggested that the C++ language has several advantages for real time robot programming. In particular, the advantages of using facilities within C++ to guarantee initialization and proper termination of hardware subsystems. However, in this project, the general purpose C language has been chosen since it is the programming language for the sub-systems. The risk of using C language in a safety-critical system is given in the language assessments table shown below which tabulates the desirable safety features of several common languages. Thus, in the future, we will attempt to reimplement our code in C++.

TABLE 1. Assessment of computer languages after[Cullyer90]

	SPADE Pscal	ISO Pascal	ADA	Modula 2	C	Structured assembler
Prevents memory overwrite	<	r	r (<)	r (<)	x	r
Has a consistent maths with the consequence of overflow, underflow etc. well defined	<	r	r (<)	r(r)	x	r
Has strong data typing	<	r	<	r (<)	x	r
Exception handling	x	x	<	r (r)	r	x
Will detect stack or heap overflow and similar	<	r	x (r)	r (r)	r	<
Separate compilation and type checking between modules	r	r	<	< (<)	x	x
Has language constructs that are well understood	<	<	< (r)	< (<)	r	x

<: indicates that the feature is acceptable

r: indicates that the feature is available but there is still a risk

x: indicates that the feature is absent

9. CONCLUSIONS

The research conducted for this project involves the investigation of control theories and real time control hardware and software to lead to the development of a useful rehabilitation robot for individuals with upper spinal chord injuries. The uniqueness of the master-to-slave mapping for each individual adds a degree of complexity to the project which will necessitate a certain degree of generalizability to the control algorithms. Also, due to the nature of the application, a very high degree of safety is required throughout the test-bed research and extended into the eventual prototype design and development. This will be realized through the use of safety logs, watch-dog hardware and software, structured coding, and a thorough documentation of every step of the research. This includes the specification of all software in Z notation.

REFERENCES

[Cox88] I.J. Cox, D.A. Kapilow, W.J. Kropfl, and J.E. Shopiro. Real-Time Software for Robotics. *A&T Technical Journal*, 67(2), March/April, 1988.

[Cullyer91] W.J. Cullyer, S.J. Goodenough and B.A. Wichman. The choice of computer languages for use in safety-critical systems. *Software Engineering Journal*, 6(2), March, 1991.

[Funda90] J. Funda. *Teleprogramming: An approach to overcoming communication delays in remote manipulation.* PhD thesis, University of Pennsylvania, 1990.

[Goertz63] R.C. Goertz. Manipulators used for handling radioactive materials. In E.M. Bennett, editor, *Human Factors in Technology*. McGraw Hill, 1963.

[Hagner89] David G. Hagner and John G. Webster. Telepresence for touch and proprioception in teleoperator systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(6):1020–1023, November/December 1988.

[Hannaford91] Blake Hannaford, Laurie Wood, Douglas A. McAffee, and Haya Zak. Performance evaluation of a six-axis generalized force-reflecting teleoperator. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(21):620–633, May/June 1991.

[Hirtzinger89] G. Hirtzinger, J. Heindl, and K. Landzettel. Predictive and knoowledge-based telerobotic control concepts. In *IEEE International Conference on Robotics and Automation*, pages 1768–1777, 1989.

[Hogan89] Neville Hogan. Controlling impedance at the man/machine interface. *Proceedings of the I.E.E. E. Conference on Robotics and Automation*, pages 1626–1631, 1989.

[Inoue71] H. Inoue. Computer-controlled bilateral manipulator. *Bulletin of the Japanese Society of Mechanical Engineers*, 14(69):199–207, 1971.

[Mann70] Robert W. Mann and Stephen D. Reimers. Kinesthetic sensing for the emg controlled "boston arm". *IEEE Transactions on Man-machine Systems*, March 1970.

[McDermod91] John.A. McDermid and David J. Thewlis. Safety-critical systems. *Software Engineering Journal*, 6(2), March, 1991.

[Mosher60] R.S. Mosher. Force reflecting electrohydraulic servo-manipulator. *Electro-Tech*, page 138, December 1960.

[Reswick75] James B. Reswick. Rehabilitation engineering center, rancho los amigos hospital, california, annual report. Technical report, Rancho Los Amigos Hospital, 1975.

[Sheridan86] Thomas B. Sheridan. Human supervisory control of robot systems. *Proceedings of the I.E.E. E. Conference on Robotics and Automation*, pages 808–812, 1986.

[Simpson77] D. C. Simpson and J. G. Smith. An externally powered controlled complete arm prosthesis. *Journal of Medical Engineering and Technology*, pages 275–277, September 1977.

[Spivey89] J.M. Spivey. *The Z notation - A reference manual*, Prentice Hall, 1989.

[Willis91] W. D. Willis and E. Coggeshall R. *Sensory Mechanisms of the Spianl Cord*, chapter 10, pages 428–436. Plenum Press, 1991.