N90-29352

RCTS: A FLEXIBLE ENVIRONMENT FOR SENSOR INTEGRATION AND CONTROL OF ROBOT SYSTEMS - THE DISTRIBUTED PROCESSING APPROACH

R. Allard, B. Mack, M.M. Bayoumi

Department of Electrical Engineering
Queen's University
Kingston, Ontario, Canada K7L 3N6

Abstract

Most robot systems lack a suitable hardware and software environment for the efficient research of new control and sensing schemes. Typically, engineers and researchers need to be experts in control, sensing, programming, communication and robotics in order to implement, integrate and test new ideas in a robot system. In order to reduce this time, the Robot Controller Test Station (RCTS) has been developed. It uses a modular hardware and software architecture allowing easy physical and functional reconfiguration of a robot. This is accomplished by emphasizing four major design goals: flexibility, portability, ease of use and ease of modification. This paper mainly reviews an enhanced distributed processing version of RCTS. It features an expanded and more flexible communication system design. Distributed processing results in the availability of more local computing power and retains the low cost of micro-processors. A large number of possible communication, control and sensing schemes can therefore be easily introduced and tested, using the same basic software structure.

1.0 Introduction

As part of the space station, the Mobile Servicing System (MSS) will be utilized to perform tasks such as carrying out basic repairs, assembling structures, cleaning surfaces and servicing satellites as discussed in [1-3]. Such capabilities in a robot system will require significant advances in the state of the art. A flexible development environment is therefore required. The Robot Controller Test Station (RCTS) has been developed as a tool to facilitate development, integration, testing and verification of robot sensors and controllers. Some other test stations currently exist; however, none of these stations highlight the four basic design goals of RCTS which are: flexibility, portability, ease of use and ease of modification. The user interface consists of a series of menus. The user chooses a number of application routines within a special purpose library and specifies operating parameters. The system features a simulation interface in addition to a robot interface. Graphic facilities are also provided to display the results of the run-time data analysis routines. RCTS is based on a three level hierarchical approach (high, intermediate and low levels). This subdivides the control and sensing problem into different priorities and levels according to their processing times and their functions. The low level control is the direct interface with the robot, and the high level initiates the robot tasks to be carried out (Figure 1). This three level structure permits sensor integration to be carried out on several levels, which allows the robot to respond to sensor input more rapidly. Detailed reviews of the initial version of RCTS are provided in [4-6].

This paper reviews an expanded and more general distributed processing version of RCTS, which is presently being implemented. It features an updated communication interface and a variety of utilities which help the user to easily modify the system and enhance flexibility. In this paper, the basic structure of RCTS is first reviewed, followed by details of the updated distributed processing version and an overview of the hardware implementation.

2.0 Overview of Basic Structure of RCTS

2.1 Arrangement of modules

The RCTS software is organized into six different modules, as outlined in Figure 1. This approach decomposes the robot control and sensing problem into three separate levels (high, intermediate, and low levels) and also separates control from sensor processing. It improves the flexibility and modularity of the system since each part is clearly defined and bounded. A detailed analysis of the module arrangement is provided in [4]. The following is a review of the functions of each basic module:

Robot Task Generator (high level control): This module specifies what task the robot should accomplish. The task may be altered with information received from the high level sensing module.

High Level Sensor Interface: This module processes data from a sensor interfaced to this module or from the intermediate level sensor interface module. It transforms the data into a form which may be used by the robot task generator to alter its operation.

Trajectory Generator (intermediate level control): This module provides the position or torque setpoints used by the low level controller when the information is requested.

Intermediate Level Sensor Interface: This module processes data from a sensor interfaced to this module or from the low level sensor interface module. The data is converted to a form suitable for the trajectory generator. This module may also send data to the high level sensor interface module.

Low Level Controller: This module calculates the drive signals for the robot from the setpoints provided by the trajectory generator and from data provided by the low level sensor interface module.

Low Level Sensor Interface: This module processes data from a sensor interfaced to this module. It sends the information to the low level controller or to the intermediate level sensor interface. This module interfaces sensors which usually have a rapid execution time, since the information is required at a high rate by the low level controller.

2.2 Module structure

Each RCTS module has an identical structure, consisting of a pre-process, process and post-process (Figure 2). The pre-process and post-process are the communication interfaces with other modules. The objective is to separate the communication interface from the actual application processing so that any section can be easily modified. The pre-process section receives commands, data and status signals from the other modules. The process uses a state table approach which consists of a set of conditions testing the internal logic states and inputs to a module. If its associated test result is positive, an application routine will execute. The routines executed in the main process update the logic state and determine the module's output signals. When all the tests have been conducted, control is then transferred to the post-process. The post-process transmits information to other modules or the robot. All the communication signals (commands, module status and internal states) are standardized for all of the modules. This makes the software easier to comprehend and modify.

3.0 Distributed Version

3.1 Principle

A laboratory or advanced robot systems similar to the MSS may require rapid integration of new sensors and reallocation of the modules to different processors. In the context of the MSS, for example, a substantial expansion of capabilities can be expected over the lifespan of the system. This may require significant changes to the processing architecture of the system. The RCTS software design philosophy can accommodate a wide range of communication schemes, both internal and external. Distributed processing is incorporated into RCTS to increase the processing power available to different modules by executing them on different processors. This also promotes parallel execution. It is particularly useful when modules are computationally intensive or need to execute at high rate (e.g., vision processing and low level control). A review of communication issues in robotics can be found in [7]. The initial version of RCTS features some distributed processing capabilities and is able to share information between processors, using locally controlled communication. This could typically be interrupt driven communication or messaging. Interrupt driven communication requires that interrupt lines be wired every time a sensor is added. These interrupt lines customize the system and may complicate the system design for the user; this violates the original design goals of RCTS (flexibility, portability, ease of use and ease of modification). The initial version is also not efficient for centrally controlled communication (e.g., polling), which may use a master--slave hierarchy, for example. In centrally controlled schemes, all information transfers are controlled by a single node. In the case of polling, for example, only the processor with the status of master can send information and interrogate other processors.

The upgraded version of RCTS features a modified communication software structure and a variety of enhanced capabilities. This eases system reconfiguration, the reallocation of processing power and the introduction of new communication interface schemes. The original pre-process and post-process sections are generalized and expanded. A new utility (Communication Relay Utility--CRU) is added and provides some synchronization and intermediate information storage functions. This permits the transfer of information between modules which cannot communicate due to a lack of direct physical link. Other functions are added to allow the user to define the network topology through the menu system (as opposed to a system designer changing the source code). The task of adding a new sensor, relocating a module to another processor or using a different communication scheme can therefore be carried out by the user.

3.2 Initial communication interface design

In the initial implementation, the pre-process and post-process sections of the RCTS modules are divided into three layers: upper, middle and lower as discussed in [5]. The upper communication interface layer determines the information to be transferred and its destination. The middle interface layer formats the information to suit the transfer method and the lower interface layer is responsible for the actual physical transfer. This layered approach emphasizes modularity, which permits each section to be designed and modified separately. It can accommodate both synchronous (e.g., memory sharing) and asynchronous (e.g., message passing) communication. For example, the following functions are carried out in the case of internal messaging: the upper layer determines the data content of the message and the destination information (target module). The middle layer formats this message to suit the mailbox size and affixes a command byte. The lower layer uses system calls or customized drivers to accomplish the physical transfer by loading the destination mailbox. The pre-processor is structured similarly, but it is executed in the reverse order. Since all the operating system calls are located into one section of the software, modifications are easily carried out.

3.3 Updated communication interface design

In the updated version of RCTS, the three communication layers (upper, middle and lower) are expanded into four layers (upper, middle, linking and lower layers) (Figure 3). The system contains both internal (between the modules residing on the same processor) and external (between modules located on different processors) communication. Examples of internal schemes are memory sharing and internal messaging (using mailboxes). Examples of external schemes are polling, interrupts and multiple access with collision detection (such as ethernet). These are different and pose special requirements. Network topology information is provided by the user from the menu system. The network topology files thus generated define which processors are linked together, the type of link, the processor communication status and the physical location of each RCTS module. In the event that an RCTS module is added or re-allocated to a new processor, the user modifies the files through the menu. The source code does not require any modifications. The system carries out an automatic reconfiguration by downloading all the necessary information to the correct node and readjusting the communication system.

Within the post-process, the upper communication layer is identical to the initial version and determines the data content and destination of the information. In the external case, when using a centrally controlled system, the communication routines used may depend on the network status of the sender and receiver (e.g., master or slave). Before the middle layer is accessed, a test is therefore made to determine this status. This uses the network topology information supplied by the user. It is a useful addition to the initial version, since it permits easier reconfiguration of the network (the source code is not modified), use of centrally controlled communication and information transfer to modules that do not have a direct link with the sender. When the communication status has been determined, the correct communication routines within the middle layer are accessed according to the relationship between the sender and the receiver. This allows the selection of any communication scheme, providing the required communication drivers are included in the system. The network topology files contain information specifying the correct driver. Similarly to the initial version of RCTS, the middle communication interface layer then formats the data to suit the communication method and adds a command byte (containing internal module states, commands and module status). For example, if the information transfer is determined to be internal and the system uses internal messaging, the middle layer will format the data to suit the mailbox size. In the case of an external message, with a system using polling, the middle layer will format the data according to the packet size (multiple packets may be generated). The linking layer is then accessed. This is an addition to the initial version. The function of the linking layer is to establish a communication link with the receiving processor. This may require sending of an interrupt signal, or detecting a possible collision within an ethernet network. The lower communication interface layer performs the actual physical transmission of the message. Usually, only the linking and lower layers will contain operating system calls. This simplifies any modifications required when using a different operating system. In the pre-process section, the linking layer also has the added function of sending any acknowledgement signals required by the sending node (fully synchronous external communication).

Some applications may require that two processors must communicate through a intermediate processor, because they are not directly linked. For example, two sensing modules may be located on different communication buses and need to share information. In this case, the information is received from the sending processor, and is retransmitted to the correct destination. For this purpose, a new utility has been created ("communication relay utility"--CRU). It is created as a separate entity and has an identical structure to the other RCTS modules (pre-process, process, post-process and uses state tables). The process section of the CRU will typically alter the message header bits to reflect the final destination. The CRU is triggered internally by an RCTS module embedded on the same processor.

4.0 Hardware and Software Implementation

The current hardware implementation is based on the iRMX-II real-time operating system (Figure 4). This is an object oriented, multitasking operating system. It permits the transfer of information between tasks using a series of user-defined mailboxes. RCTS is implemented on Intel 310 and 320 microcomputers (based on the 80286 and 80386 microprocessors, respectively) and controls a Puma 550 robot. The system also utilizes Intel single board controllers (8044 based) to interface simple devices such as proximity sensors. The updated version of RCTS is implemented with a Bitbus network linking all the processors. Bitbus is a serial communication bus which uses a polling scheme. With this scheme, a processor is either a master or a slave. Only a processor which has the master status can initiate communication. The ease of use, flexibility and the low cost of Bitbus are its major advantages. This section reviews the details the implementation of a centrally controlled communication scheme in the updated version of RCTS. The low level control and low level sensing modules are located on the same processor. Because of speed requirements, the robot and the low level sensors are interfaced to the processor by a parallel interface.

In a polling based environment, a processor is either a master or a slave. Only the processors having the status of master can initiate communications. A slave cannot initiate any communications on its own, but must wait for a handshaking signal (called a poll) from the master. When a slave wants to transmit a message, it must first store it in a buffer which will later be accessed by the master. If a master wants to transmit to a slave under its jurisdiction, it must first send a poll message containing the data and wait for a message reception acknowledgement signal. The presence of acknowledgement signals complicates the communication system. It can cause large delays to a processor wanting to send messages to different processors. Bitbus, however, has the advantage of permitting the formation of a multi-layer network. Each level is able to communicate with its neighbour. The current system uses two buses (Figure 4) communicating through the processor hosting the intermediate level control module. This structure is more suited to the hierarchy of RCTS modules (high, intermediate and low level control and sensing). In a centrally controlled bus, the demands on a single master to poll every other processor and then transfer the message to the correct destination would introduce excessive communication delay. Since control information flows from the high to low level (sensing information flows from low to high level), a master node containing a control module is therefore able to request information to the control module situated on a higher level, or to sensors on the same level. This, therefore, prioritizes the control modules over the sensing modules, and permits them to meet their real-time requirements. For the sensing modules, it increases the execution time and results in a slower response. If no new sensing data is generated between two polls sent by the control modules, old data is transmitted. This ensures that data is always available and eliminates the need to send multiple poll signals to capture new data.

(1) Master to slave communication on the same bus level

Message transmission by the master: In common with all the situations explained in this section, a verification of the functional status of the sender and receiver processors is first carried out. For example, in the present case, the sender is a master and the receiver is a slave under its jurisdiction. The correct communication routines are then chosen using this information. The message is formatted by the middle communication interface layer, according to the format used by Bitbus (13 bytes of data and 7 bytes of header). The linking layer receives a request to send a message from the middle communication interface layer and interrupts the application routine at the slave by sending a poll signal. The master then waits for an acknowledgement signal (coming from the slave pre-processor's linking layer). The message then goes to the pre-processor of one of the slave modules. Upon completion of the message transmission, control is passed back to the linking layer which waits for a reply message.

Message reception by the slave: The slave's pre-process receives a polling signal and interrupts its processing to access the linking layer of the pre-processor. The message is first transferred to the pre-process lower layer. The upper interface communication layer then decodes the message in a form usable by the process (the header bytes are removed and the data is assigned to the correct variables).

(2) Slave to master communication on the same bus level

Message transmission by the slave: The slave cannot transmit a message to the master without having first been interrogated by the latter. It first waits to receive a poll signal and then returns a reply message containing the data. Normally, the master will only issue a poll when it wants to receive the message. This results in some synchronizing problems and causes the slave task to lock up until the message transmission can be completed. To circumvent this problem, a temporary memory storage area is used to unload the module's post-process. When a poll is received, the buffer (located in the lower communication interface layer) is accessed and the data is then transferred externally to the receiving module.

Message reception by the master: When a request for data or command is encountered in the pre-process of the receiving module (at the master), a poll signal is issued to the slave module. If no new data or command is available (the sending module has not finished executing once), the old message is retransmitted. This ensures there is no synchronization problem and that timing requirements are met. If the old message was not retransmitted (or a code indicating the data has not changed), the receiving module (master) would have to reissue another poll signal later. This could cause processing delays or lock-up. This approach allows minimal delays of the control modules by providing them with data (new or old) at set interval times.

(3) Slave to slave or master to slave communication on different levels

An example of this case occurs when the intermediate level sensor interface module requests communication with the high level sensor interface (both are on different bus levels). The two modules do not have any direct links, so the messages have to be temporarily stored in a node that has communication access to both. Moreover, both are slaves, so they cannot initiate the communication. This results in a sizeable transmission delay, because of the time expended in waiting for poll signals generated externally. For this purpose, the communication relay utility (CRU) is used in each master node for message relaying. The absence of the CRU would not prevent the current implementation from being used. It would only limit sensor integration opportunities, since the different levels of sensing would not be able to communicate. The function of the CRU is to poll all the slave processors located on the level under its jurisdiction, for messages that would have to be re-transmitted to another node. It is triggered by one of the RCTS module embedded on the same processor. It has the same structure as the RCTS modules (pre-process, process and post-process, with all the sub-layers), and uses the same standard command and status input signals. The sending slave module must modify the header bytes (done in the middle interface layer), to reflect the temporary destination of the message. Some timing functions are also added to make sure the real-time constraints are respected.

5.0 Conclusion

The RCTS system is proving valuable to test and compare new control and sensing schemes. A researcher with limited knowledge of robot hardware and software can easily perform system integration duties that now require hours, instead of days or weeks. The distributed version allows greater computing power to be allocated towards RCTS, therefore permitting the development of more complex routines. This updated version integrates a larger variety of communication schemes (e.g., centrally controlled with master--slave hierarchy) with added flexibility and ease of use. The reconfiguration is also more easy and does not require a system designer. Future work regarding RCTS will concentrate on increasing the computing capacity and adding several utilities to the system. A new communication bus providing a larger bandwidth will likely be implemented in the near future (such as multibus II or S/NET). Parallel processing will also be investigated, using a new operating system (e.g., Harmony). The portability of RCTS permits such major changes to be carried out easily.

6.0 Acknowledgements

The authors would like to thank: John Griffioen and Peter Luscher for their contributions in creating software; Intel Corp. for some equipment donations; and funding from NSERC and the Manufacturing Research Corporation of Ontario.

References

[1] Werstiuk, H. and Gossain, D., "The Role of the Mobile Servicing System on Space Station," Proc. of the 1987 International Conference on Robotics and Automation, Raleigh, NC, USA, March 30 - April 3, 1987.

[2] Hunter, D., "An Overview of the Space Station Special Purpose Dexterous Manipulator (SPDM)," National Research Council of Canada, Technical Report No. 28817 (issue A), Ottawa, Canada, April 7, 1988.

[3] Vigneron, F., Caswell, R., Sachdev, S., and Ravindran, R., "Technology Research and Development Associated with the Mobile Servicing System," Presented at the Fourth CASI Conference on Astronautics, Ottawa, Canada, November 3 - 4, 1987.

[4] Mack, B. and Bayoumi M.M., "A Robot Controller Test Station (RCTS)," Proc. of the 19th International Symposium on Industrial Robots, Sydney, Australia, November 6 - 10, 1988.

[5] Mack, B., Allard R., and Bayoumi, M.M., "An Environment for the Development of Sensor-based Robot Software," Proc. of SPIE Conference on Intelligent Robots and Computer Vision, Cambridge, MA, USA, Vol. 1002, November 6 - 11, 1988.

[6] Mack, B. and Bayoumi, M.M., "Analysis of Sensing and Control Algorithms Using the Robot Controller Test Station," Presented at the Fifth CASI Conference on Astronautics, Ottawa, Canada, November 15 - 16, 1988.

[7] Gauthier, D., Freedman, P., Carayannis, G., and Malowany, A., "Interprocess Communication for Distributed Robotics," IEEE Journal of Robotics and Automation, Vol. RA-3, No. 6, December 1987, pp. 493 - 504.

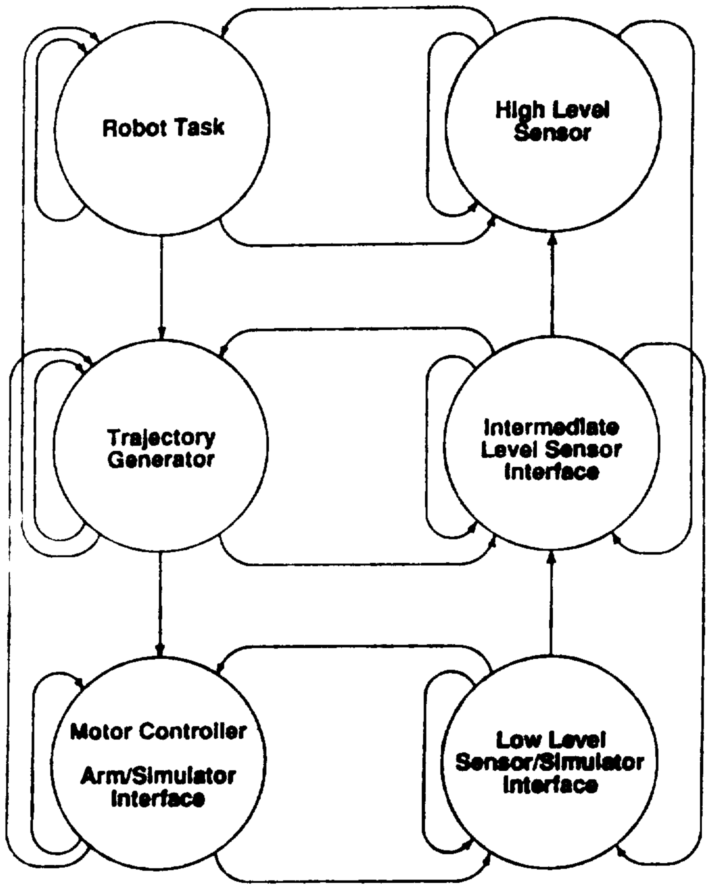

FIGURE 1: MODULE LAYOUT FOR RCTS

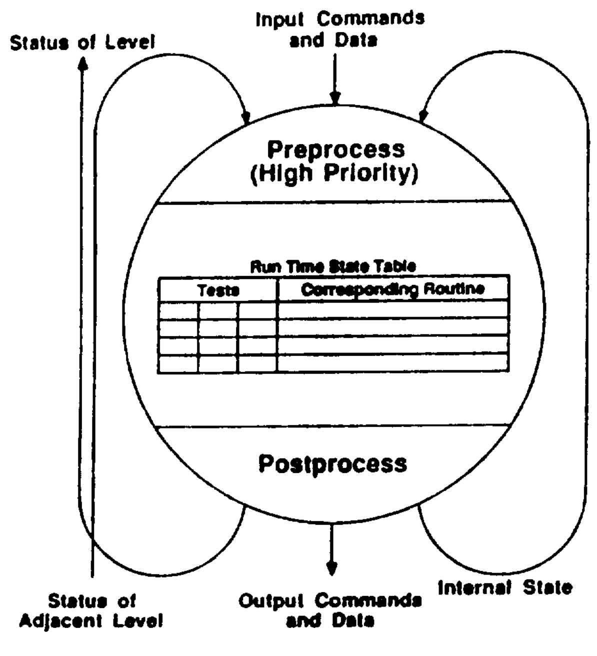

FIGURE 2: RCTS MODULE STRUCTURE

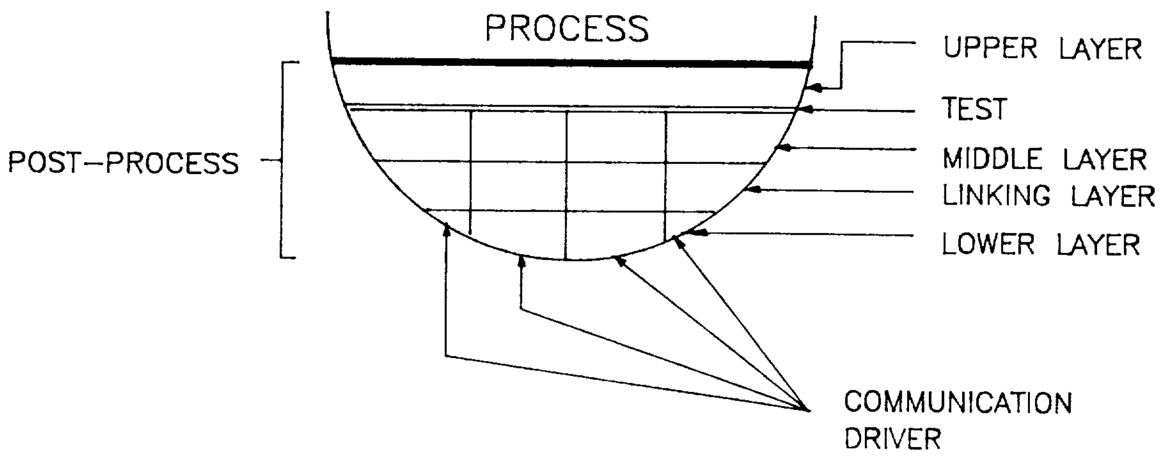

FIGURE 3: POST-PROCESS STRUCTURE

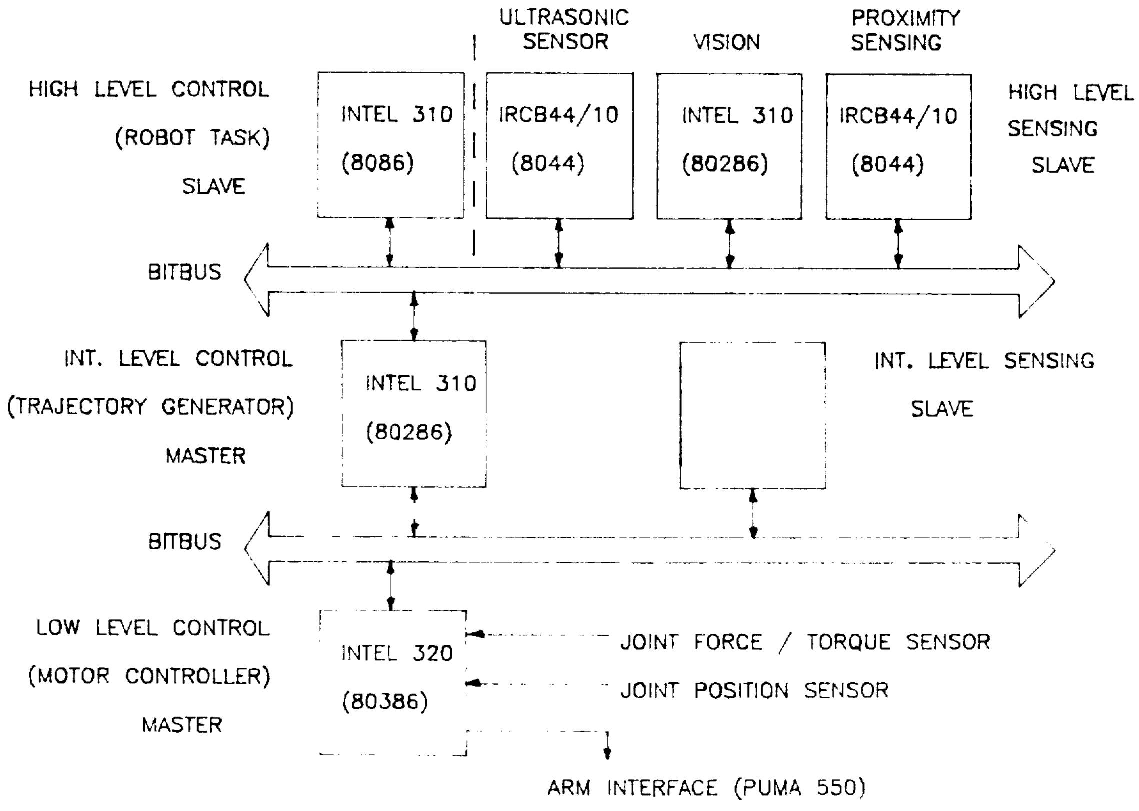

FIGURE 4: HARDWARE IMPLEMENTATION