The CMU Reconfigurable Modular Manipulator System

Donald Schmitz, Pradeep Khosla, and Takeo Kanade

CMU-RI-TR-88-7

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

May 1988

Table of Contents

List of Figures

The CMU Reconfigurable Modular Manipulator System

Donald Schmitz, Pradeep Khosla[1] and Takeo Kanade

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Modular manipulator designs have long been been considered for use as research tools, and as the basis for easily modified industrial manipulators. In these manipulators the links and joints are discrete and modular components that can be assembled into a desired manipulator configuration. As hardware advances have made actual modular manipulators practical, various capabilities of such manipulators have gained interest. Particularly desirable is the ability to rapidly *reconfigure* such a manipulator, in order to custom tailor it to specific tasks. This reconfiguration greatly enhances the capability of a given amount of manipulator hardware. This paper discusses the development of a prototype modular manipulator and the implementation of a configuration independent manipulator kinematics algorithm used for path planning in the prototype.

1. Introduction

The major advantage of robotic manipulators over task-specific hardware for automation is their flexibility. In theory, a robot's task can be changed simply by loading a new program into its controller. However, in practice this is rarely the case. Each robot has a specific configuration that supports a limited range of capabilities, appropriate only to the applications for which it was designed. The major factors that define the configurations are the link lengths, joint actuators, and geometry of joint-link connections. For example, horizontal SCARA-configuration manipulators, connected with relatively short links, are suitable for delicate table-top assembly operations requiring accuracy and selective stiffness, but they are not usable for tasks that require a vertically large workspace. On the other hand, medium-sized, vertical Puma-configuration manipulators with a relatively long reach in all directions, are suitable for painting, welding and parts handling. Using manipulators with different configurations for each task is possible when the task requirements are known beforehand. However, in less predictable situations, such as an outdoor construction site, inside a nuclear facility or aboard a space station, a manipulator system would need a wide range of capabilities, probably beyond the limitations of a single fixed-configuration manipulator.

We have proposed a manipulator system, *The Reconfigurable Modular Manipulator System* (RMMS), that addresses the above mentioned shortcomings. It provides a viable alternative to using fixed configuration manipulators by extending the existing concept of modular manipulator design. The term *modular manipulator* generally refers to a robotic manipulator assembled from discrete mechanical joints and links into one of many possible manipulator configurations [WURST]. Such a manipulator has several advantages over conventional

---

[1]Assistant Professor, Department of Electrical and Computer Engineering

designs, most notably economy of manufacture, ease of modification and ease of repair. At least one such modular manipulator is now commercially available [ROBRES].

The Reconfigurable Modular Manipulator System extends the concept of modularity throughout the entire manipulator system to include not only the mechanical hardware, but also the electrical hardware, control algorithms, and software as well. The RMMS (Reconfigurable Modular Manipulator System) utilizes a stock of interchangeable link modules of various lengths, and joint modules of various sizes and performance specifications. This modularity allows a wide range of manipulator architectures to be assembled from a small set of general purpose hardware and software components.

The concept of an RMMS poses challenging technological and theoretical research issues that must be addressed before such a system can be used effectively. In this paper we discuss both theoretical and technological issues and describe our progress in this area. In order to demonstrate our ideas we have built a prototype RMMS in our laboratory. We describe the design and operation of this prototype RMMS. The prototype includes 6 joint and 6 link modules, and a controller consisting of a Motorola 68020 based computer with real-time capabilities. We have also implemented an algorithm that automatically generates forward and reverse manipulator kinematics. The RMMS is presently controlled by independent joint control algorithms. We are now addressing issues such as mapping task specifications to manipulator configurations, automated generation of the manipulator dynamics equations, and reconfigurable model-based control algorithms. Interestingly, a recent survey indicates a need for manipulators with both reconfigurability and extensibility for research in all areas of robotics [Walker]. Our RMMS design provides practically all of the features discussed in this survey.

2. Design Philosophy and Implementation

An RMMS consists of similar subsystems as those found in conventional manipulators:

- A physical structure of joints and links.
- Servo systems for each joint, consisting of actuators, transmissions, and sensors.
- A computer controller and programming environment.

The major differences between an RMMS and a conventional manipulator are the standardized component interfaces and configuration independent control algorithms. The interface standardization must include the mechanical mating of manipulator modules, the format of data communication, the communication protocols between hardware and software, and between various levels of software. Although adopting such standards impose some restrictions on the design of the actual components, this disadvantage is offset by the interchangeability of manipulator components and the capability for rapid reconfiguration. In the following subsections, we present the design, and mechanical and electronics interface of each major component in the prototype RMMS system that we have developed in our laboratory.

2.1. Link and Joint Modules

The mechanical modules making up an RMMS are divided into two groups, joints and links. The design of each module is independent of other modules except for the module interfaces which are standardized. One implication of this modular joint design is that the *entire* joint actuator must be packaged *within* the joint module. Each joint module must include a motor (or some type of actuator), a transmission mechanism, a position sensor, and the

necessary power electronics to control the motor. Electrical power is distributed and communication is multiplexed over a small number of conductors permanently installed in each module. This allows for simple assembly without custom cabling. Although these design constraints limit the power which can be generated by the joint due to the limited size of the motor, transmission, and power amplifier, this is not viewed as a major short coming of the design. By properly selecting the transmission reduction ratio, high torques at low speeds can be obtained, appropriate for most tasks as long as speed of operation is not critical.

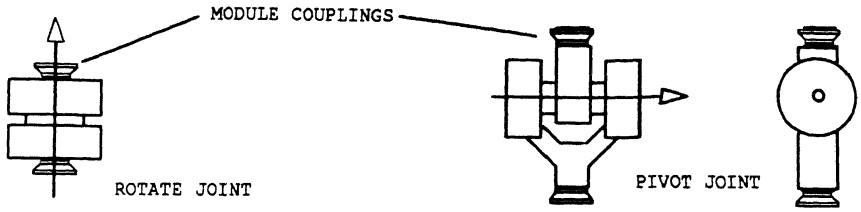

Figure 2-1: Modular Joint Assemblies

For simplicity and convenience, we have considered and built only the two common types of revolute joint in our RMMS. These two types are rotate, and pivot, and are distinguished by the orientation of the joints link axes with the joint axis. Both types of joint are shown schematically in Figure JTS. A rotate type joint has link axes which are co-linear with each other and with the joint axis. A pivot has link axes which are both perpendicular to the joint axis.

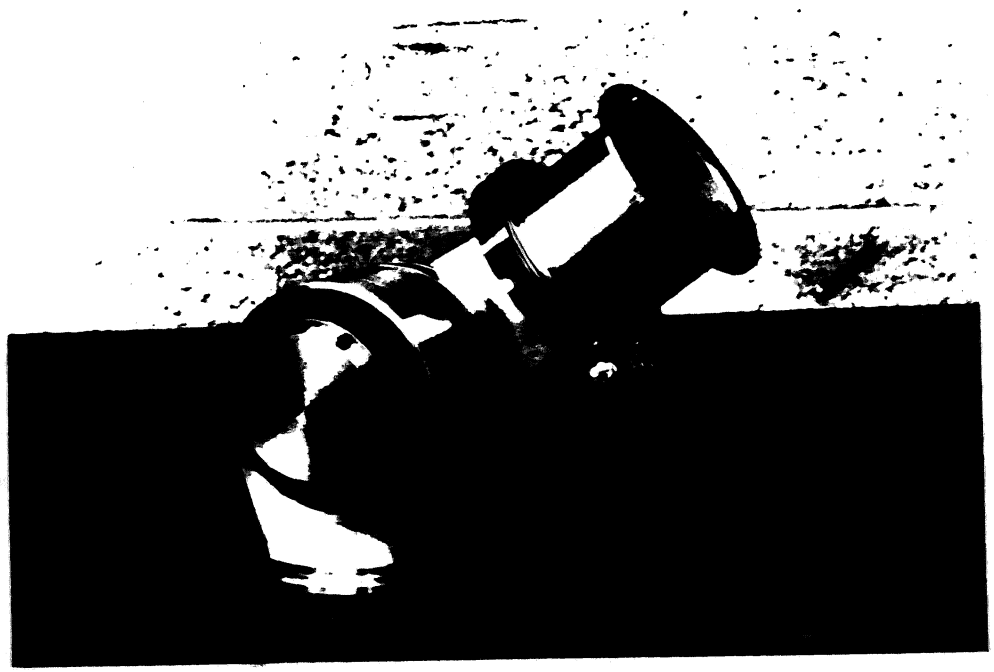

Figure 2-2: Photo of CMU RMMS Prototype Pivot Joint

Our current designs for pivot and rotate joints are shown in the photographs in Figures CMUPIVOT and CMUROTATE. The actuator in each joint consists of a conventional servo motor and linear amplifier driving a harmonic drive with 200:1 reduction ratio. This design yields a maximum output torque of 200 ft-lbf, and maximum

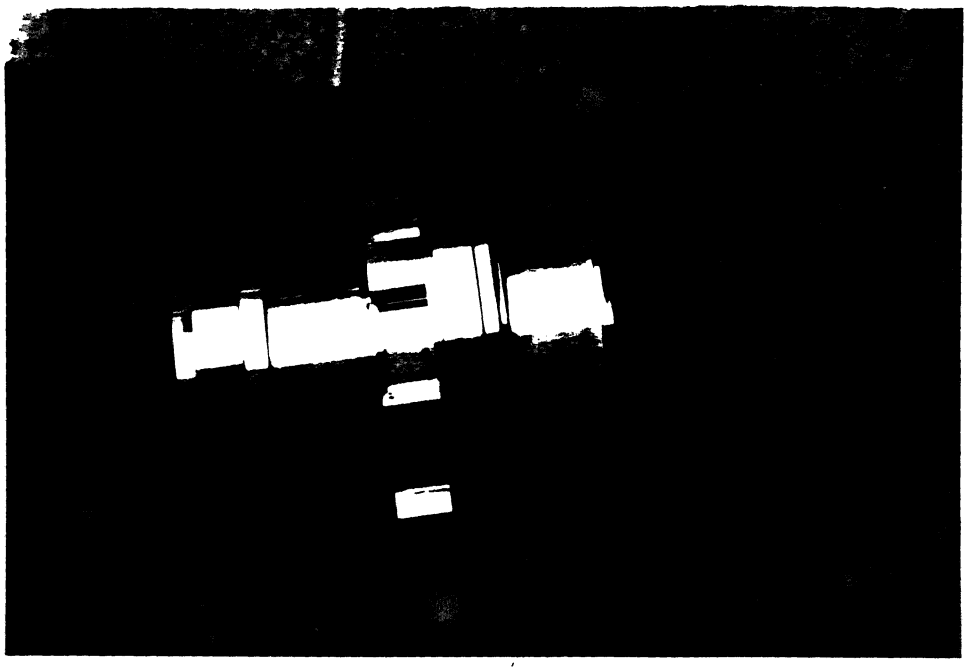

Figure 2-3: Photo of CMU RMMS Prototype Rotate Joint

axis speed of 0.7 radian/second. Also integral with the joint assembly is a brushless resolver mounted coaxially with the output shaft, providing position feedback with a resolution of 0.0001 radians. A wire windup allows the resolver (and output shaft) to turn up to 480° before damaging the resolver electrical connections. In our design we have also allowed for incorporating a tachometer that is directly coupled to the motor shaft. The tachometer will provide output shaft velocity measurements with a resolution of 0.001 radians/second. All of the actuator components are packaged in a sub-assembly of the joint module, allowing a number of kinematically different types of module to be manufactured from this common assembly. The total weight of both types of joint is 17 lbs.

We tested the joint modules using a fixed gain, PSD feedback control algorithm. The control loop gains and sampling rate were determined by an experimental procedure [khosla8]. In our experiments, we obtained static positioning accuracies of ±0.001 radians, and closed-loop stability of the system was demonstrated at sampling rates as low as 100 Hz. We are currently developing techniques for dynamics identification to evaluate the use of model-based reconfigurable controllers for the RMMS.

2.2. Joint - Link Interface

In order to assemble the joint and link modules into a manipulator, a method of mechanically coupling the modules is required. This coupling must both align the modules, and lock them together with sufficient strength to transmit the internal forces generated by the movement of the manipulator. In addition to structurally coupling the modules together, this interface must also electrically couple the modules, and be able to sense the coupling orientation of successive modules.

The current interface design is shown in the photograph in Figure VBAND. The mechanical coupling is

accomplished using commercial V-band clamps. V-band flanges are an integral part of the link and joint modules, as shown in Figure 2-2 and VBAND. An arrangement of pins and holes in each flange limits the coupling orientation to four, equally spaced positions that are 90 degrees apart. An LED in one flange and four phototransistors in the other allow the controller to sense which of the four possible orientations is in use. Although rudimentary, this design provides the necessary functionality for the module interface. We are currently investigating the use of quick release V-band clamps and more sophisticated designs with locking mechanisms that allow automatic "peg-in-hole" type coupling.

Figure 2-4: Photo of Prototype Module Interface

2.3. Communication Interface

Each joint houses the power and sensor electronics for the actuator. To control the joint actuators and obtain sensor feedback, a communication link between the joint modules and a computer controller is required. To allow standard connections between joint modules, this communication link must be implemented using a fixed number of conductors while being capable of supporting an arbitrary number of modules. This implies a multiplexed communication link, similar to a computer bus or Local Area Network (LAN).

Due to the data transmission overhead associated with existing LANs, our prototype utilizes a bus type implementation, referred to as the *armbus*. The armbus design is shown schematically in Figure ARMBUS. This design is based on a conventional 8-bit bi-directional data/address bus, an additional 5 control lines, and a rather unconventional 4 bit daisy chained node address bus. The daisy chained address bus provides automatic node address configuration; the first module in the manipulator is node address 1, the second module is node address 2, and so on. This is accomplished by including a "subtract one" circuit in each module which is in the path of the node

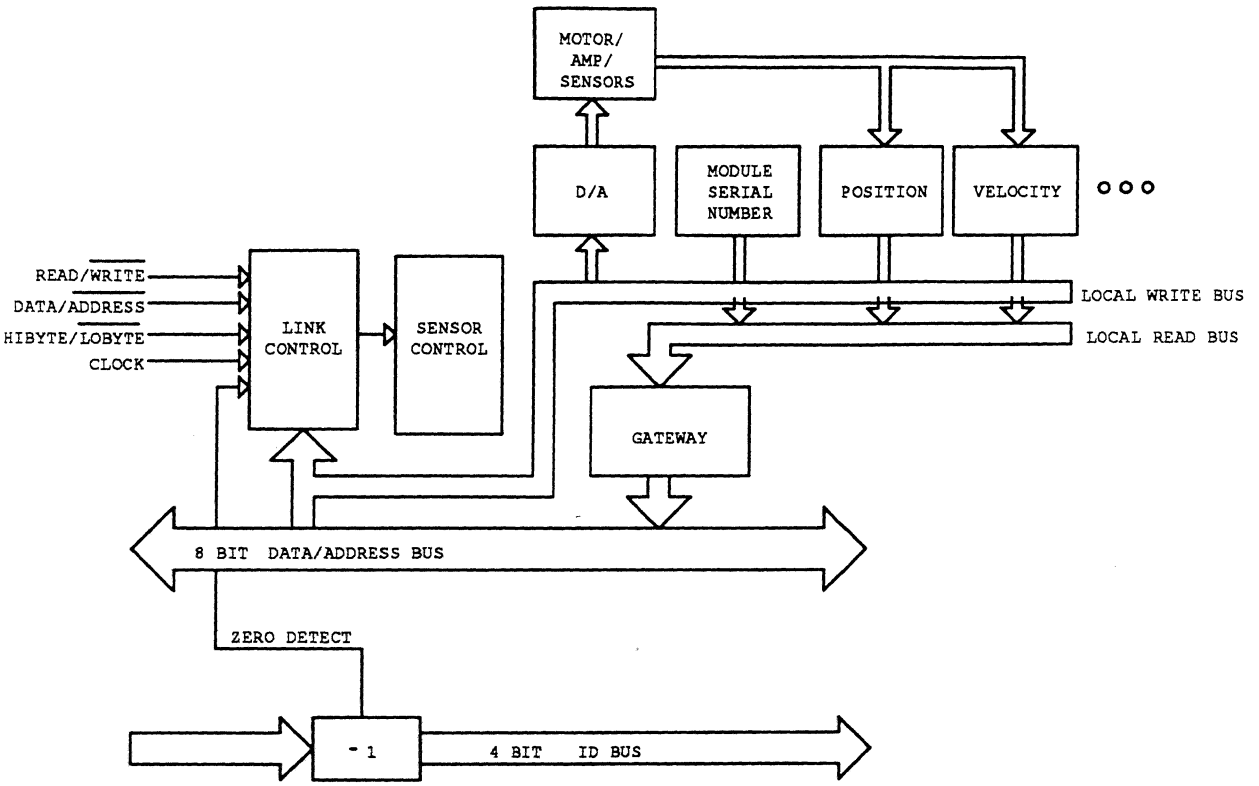

Figure 2-5: Manipulator Communication Bus Logic

address lines. Each joint can thus detect "address equals zero" as the node address. Due to the low data rate of the bus (current bus clock is 500 KHz), the propagation delay added by the subtract circuit is negligible.

2.4. RMMS Computing Environment

RMMS software is easily divided into two functional classes: real-time critical control programs and event-driven application programs. Real-time programs are those which must be executed at a predetermined sampling rate, such as control law calculation. In contrast, event-driven programs rely on detecting conditions, such as the manipulator reaching a certain position, to schedule future manipulator actions. In our implementation, we have chosen this distinction (between real-time and event driven programs) as a natural module boundary for organizing the manipulator control software.

In the RMMS environment, a CPU is dedicated to each class of software. Real-time control programs execute on a dedicated *controller CPU*, with a hardware interface to the inter-module communication network. This controller CPU performs the necessary realtime control of the manipulator, and receive commands from a second, *master CPU*. This master CPU executes the event-driven application program. In this architecture the manipulator controller appears as a peripheral device. An interrupt driven communication channel between the two processors provides a well defined interface between the two software/computing modules.

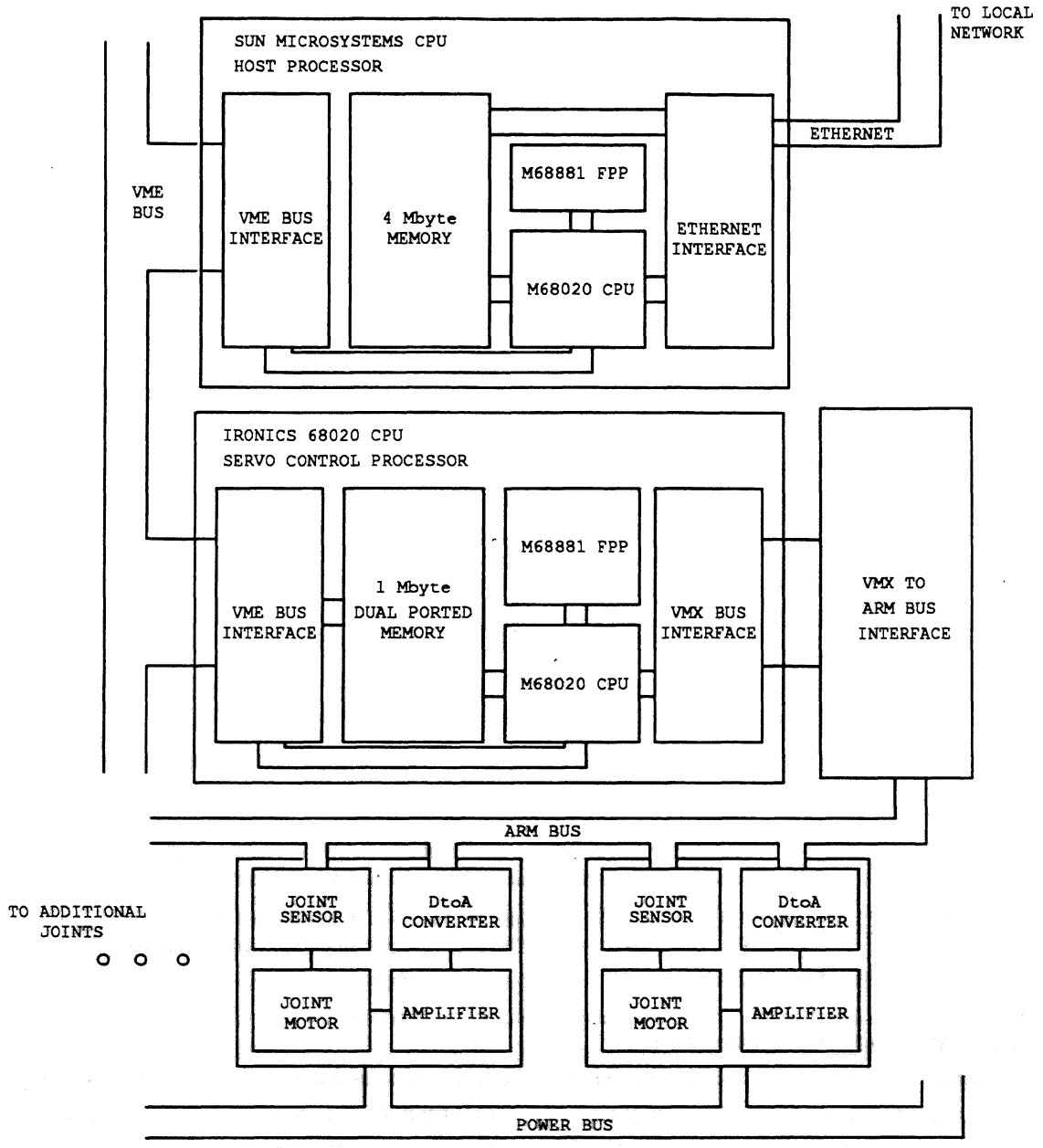

Figure 2-6: Schematic of RMMS Computing Architecture

We have implemented this architecture, depicted in Figure IRON, for controlling the RMMS. The controller CPU is an Ironics single-board computer, based on a Motorola 68020 processor and VME bus, with 1 MByte of dual ported RAM. The master CPU is a SUN-3 workstation, also based on the Motorola 68020 and VME bus. This basic architecture (and the support software) can be expanded to include additional Ironics CPUs for greater computational power. The similarity between the Ironics and SUN's CPU allows us to use the same editor and compiler for both processors thus simplifying software development and inter-processor communication. Real-time control programs, at all levels, are written entirely in C programming language. The interface to the manipulator communication network is via the VMX bus interface included on the Ironics. The VMX bus is a recognized extension to the VME

bus and is intended to be a local IO bus in multiprocessor systems such as the one we have built for controlling the RMMS.

2.4.1. Real-Time Operating System

Manipulator control programs executing on the Ironics real-time CPU are linked with a locally developed real-time operating system or kernel. This kernel provides a number of concurrency and scheduling primitives, allowing users to write control programs as a series of concurrent processes. It also supports many Unix-like utilities, particularly memory allocation and access to the SUN file system. These features have two important implications to the development of manipulator control code:

- Control algorithms are written without regard to the specific hardware and low level software implementation of the system. At the same time, the programmer is forced to more fully understand the data flow and timing relationships of the algorithm being coded, to specify those relationships via the concurrency primitives.
- By providing real-time programming utilities that mimic their Unix counterparts, a large base of existing Unix/C code is easily ported to real-time applications. Similarly, a large base of existing Unix/C programming expertise is also readily available.

2.4.2. Real-Time Software Architecture

The current software control architecture is shown in Figure SOFTWARE. In the current design there are four principal processes executing concurrently:

- The feedback control law which is implemented for each manipulator axis can be executed at sampling rates of 50-500 Hz. Our current implementation employs a sampling rate of 200 Hz.
- The path planning algorithm updates the control loop inputs to drive the manipulator to a desired position in a specified manner (eg. straight line, minimum time, etc). This can operate at sampling rates of 5-30 Hz. We are presently using a sampling rate of 20 Hz.
- A data logging process that records specified values of the manipulator state. This information is required for off-line analysis and for monitoring manipulator control experiments.
- An interactive command interpreter that implements a low level manipulator control language. This allows a user or an application program on the SUN-3 to issue commands, to the control package, for displaying data about the manipulator state.

2.4.3. Real-Time Computing Performance

The Motorola 68020/68881 CPU has been extensively benchmarked for many applications, with typically reported performances of 2 MIPS and 0.25 MFLOPS [SUNLIT1, MOTOROLA1]. In order to determine the performance of the actual system executing a typical manipulator control program, the RMMS realtime CPU was benchmarked performing a single iteration of a PSD position control loop. The control law calculation is given by the following pseudo-C code. All variables are double precision floating point variables, referenced indirectly by an offset from an address register (the benchmark thus includes a typical level of addressing overhead). The actual code was written with no attempt at optimization other than that performed by the compiler.

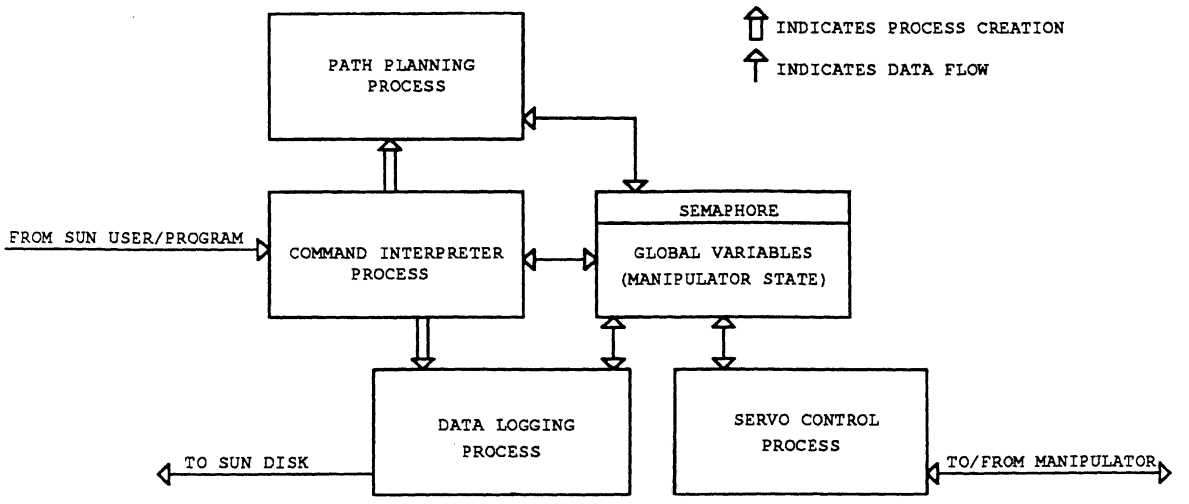

Figure 2-7: Control Software Organization

```
pos_error = reference_position - position;
vel_error = reference_velocity - velocity;
integral = (integral * alpha) + pos_error;
torque_command = (pos_error * Kp) + (vel_error * Kv)
                                          + (integral * Ki);
if(torque_command > Tlim) torque_command = Tlim;
else if (torque_command < -Tlim) torque_command = -Tlim;
```

This computation requires 11 floating point operations (4 multiplies, 5 additions/subtractions, and 2 comparisons). The actual code is fairly typical of first pass code written by an average C programmer. This segment executes in 0.12 milliseconds, indicating floating point performance of approximately 0.1 MFLOPS. Obviously this is a rough measurement of system performance, however this is quite good considering the unoptimized nature of the code. With simple code optimization, it is quite possible that compiled C code could approach the 0.25 MFLOP performance claim.

2.4.4. Application Control Software

Within the RMMS computing environment, application programs are SUN-3 programs, written in a SUN supported language. Currently, we are using the C programming language for developing application programs. Access to the manipulator controller is via special *Unix devices* which implement *pipe* like communication channels to the real-time program. This mechanism has been used to build a message passing protocol between the two processors. This has been done for the existing manipulator control package, allowing a SUN program to call an appropriate library routine which signals the manipulator control program to execute the desired command.

3. Automatic Kinematics Generation

Specifying a manipulator task typically requires specifying the end effector position (with reference to the manipulator base) as a function of time and system conditions. This method of task specification is well suited to an RMMS, as it is completely independent of the manipulator configuration; the manipulator is simply considered a motion transducer. Since the end effector position is controlled indirectly by controlling each joint's axis position, the relationship between these two quantities, known as the manipulator forward and reverse kinematics, is required. Deriving a set of Denavit-Hartenberg parameters (for the forward kinematics) and a closed-form reverse kinematics solution requires both mathematical manipulation and geometric intuition [PAUL1]. Further, since an arbitrary manipulator may be created from the RMMS, the forward and reverse kinematics solutions have to be derived for each configuration of the manipulator.

To alleviate the above difficulty we have proposed algorithms that create the forward and reverse kinematics solutions automatically from a description of the joint and link modules and the sequence in which they have been connected. For the reverse kinematics we have adopted a numerical approach that allows for complete generality and can also accommodate redundant manipulators. A general numerical solution to the reverse kinematics is often computationally inefficient and mathematically poorly behaved especially close to singularities. To address this issue, we have developed a robust reverse kinematics solution that is well behaved close to a singularity and can be computed at real-time rates. In the ensuing paragraphs we present our approach to generating the kinematics of a RMMS automatically.

3.1. Generating the Forward Kinematics

The forward kinematic equations of a manipulator describe the position and orientation of the end-effector as a function of the joint variables. The forward kinematic transformation is typically obtained from a set of parameters known as the Denavit-Hartenberg (D-H) parameters of the manipulator. These parameters are obtained through a predefined sequence of transformations and are a function of the geometry of the manipulator. The input to our forward kinematics algorithm is the geometry of each module, the type of each module, and the sequence of connection of the modules that comprise the manipulator. The output of our forward kinematics algorithm is the set of D-H parameters of the manipulator.

We use homogeneous transformation matrices to specify the geometry of modules. For a link module we use one homogeneous transformation that relates one end of the link to the other as depicted in Figure LINKMOD. In order to incorporate both the degree-of-freedom of a joint and its shape we use two homogeneous transformations: one from the lower left connector to the origin of the joint (${}^{l}J_{o}$) and another from the origin to the upper right connector (${}^{o}J_{u}$). A typical joint module and its database description is shown in Figure JTMOD. The definition of the origin of the joint module is arbitrary as long as it is chosen to be a point lying along the axis of rotation. Based on the above systematic description, we have implemented an algorithm that automatically creates the forward kinematics of an RMMS. For the sake of brevity we have excluded the details of the algorithm in this paper, they are presented in [KELMAR1].

A simpler method of generating the forward kinematics of an RMMS would be to sequentially multiply all the module transformations. However, it is desirable (particularly when the manipulator Jacobian is also required) to represent the forward kinematics in terms of the Denavit-Hartenberg parameters. In the present implementation, the

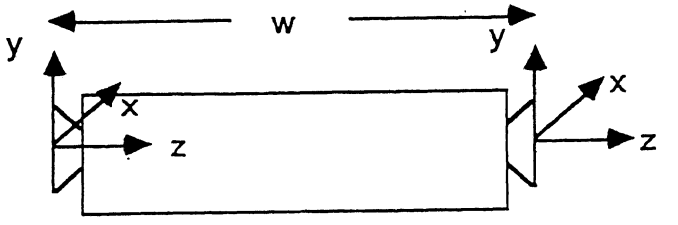

Figure 3-1: Link Module Coordinate Assignment

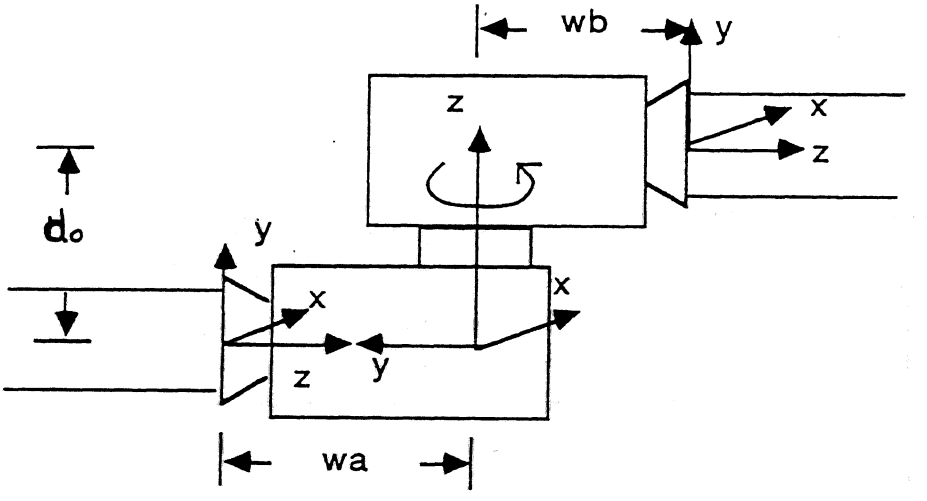

Figure 3-2: Joint Module Coordinate Assignments

control computer reads the description of the joint and link module descriptions through a database file. However, in the future each joint and link module will have a ROM which will include the kinematic information pertaining to that module.

3.2. Reverse Kinematics of RMMS

In order to do any controlled movement it is necessary to have an inverse kinematic model to determine the joint angles required to achieve a desired position and orientation of the end-effector. Ideally, one derives closed form equations for the inverse kinematics where each joint variable is expressed in terms of other known quantities. However, existence of a closed form inverse kinematics solution depends on the kinematic structure of the manipulator [Pieper, Wolovich]. For example, it is known that a closed form solution exists for a manipulator which

has three consecutive axes that intersect, such as in a spherical wrist [Pieper]. This solvability condition is not necessary, but only sufficient. Because an RMMS manipulator can assume any configuration, including one that is redundant, it may not be possible to find a closed form solution. In order to provide for generality we have adopted a numerical approach for solving the inverse kinematics of an RMMS. In the ensuing paragraphs we describe a numerical method to compute the inverse kinematics of non-redundant manipulators [Khosla85]. We also describe an extension of this method that is applicable for redundant manipulators.

3.2.1. Inverse Kinematics of Non-Redundant Manipulators

A closed-loop method for solving the inverse kinematics equations using the Newton Raphson method is proposed in [Khosla85] and is depicted in Block diagram form in Figure BLOCK. The iterative method determines the necessary changes in the joint angles to achieve a differential change in the position and orientation of the end-effector. The forward kinematics are described in functional form as:

$$\mathbf{x} = \mathbf{f}(\mathbf{q}). \tag{1}$$

where $\mathbf{x}$ is the vector of Cartesian position and orientation and $\mathbf{q}$ is a vector of joint displacements. The corresponding differential changes $\mathbf{dx}$ and $\mathbf{dq}$, in the Cartesian and joint space, respectively, are related through the manipulator Jacobian as:

$$\mathbf{dx} = \mathbf{J}(\mathbf{q})\mathbf{dq}. \tag{2}$$

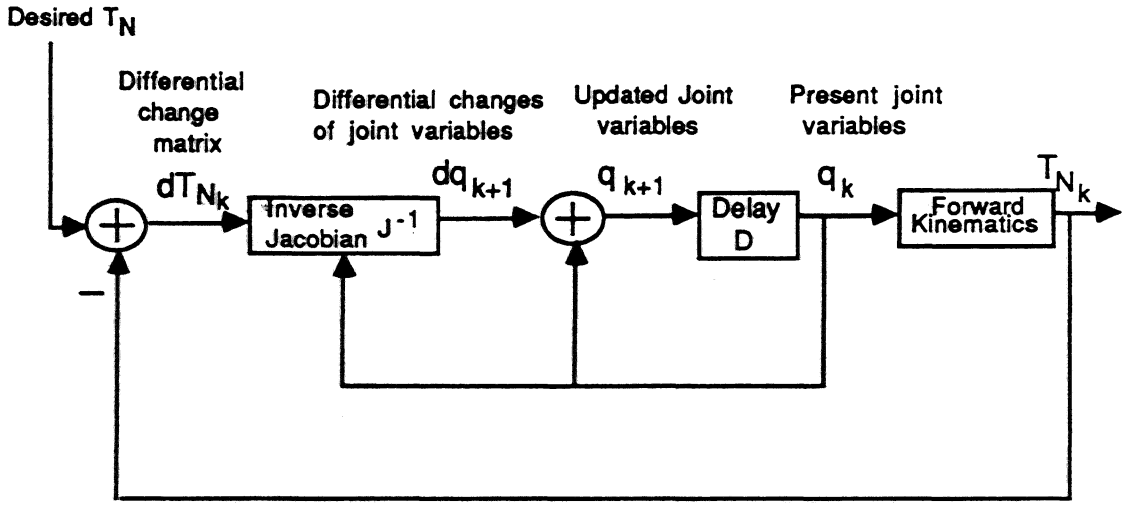

Figure 3-3: Block diagram of iterative inverse kinematics procedure

Inverting Equation (TR2) to obtain an expression for the differential inverse kinematics we obtain:

$$\mathbf{dq} = \mathbf{J}^{-1}(\mathbf{q})\mathbf{dx} \tag{3}$$

where $\mathbf{J}^{-1}$ is the inverse Jacobian. The above equation may be written, in an iterative form, as:

$$\mathbf{dq}_{k+1} = \mathbf{J}^{-1}(\mathbf{q}_k)\mathbf{dx}_k \tag{4}$$

where the differential change in position and orientation at the k-th iteration is computed from the differential homogeneous transformation matrix $\mathbf{dT}_{N_k}$ [paul1]. The joint displacements are computed as:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathbf{dq}_{k+1}$$

Equation (TR4) is solved iteratively, until each term in $\mathbf{T}_{N_k}$ (or correspondingly in $\mathbf{dx}_k$) is within a prespecified error

tolerance, ε.

We have performed experiments using the above algorithm and have shown it to work well for non-redundant systems. Including redundancy introduces complications in the computation of the inverse kinematics solution. The Jacobian, which relates differential changes in the joint variables to differential changes in the Cartesian variables is of dimension M x N, where M is the number of degrees of freedom of the workspace and N is the number of degrees of freedom in the manipulator. When M and N are not equal (which is the case for redundant manipulators), the Jacobian is no longer invertible and we must substitute a generalized inverse to provide an inverse equivalent.

Much of the previous research on inverse kinematics for redundant manipulators has focused on the pseudoinverse [Baillieul, Chang, Klein]. The pseudoinverse is a generalized inverse which provides the minimum norm solution [Noble]. Because standard pseudoinverse control has proved to be inadequate in the neighborhood of singularities, many methods have been developed which augment the pseudoinverse so as to use the kinematic redundancy to optimize an objective function [Baillieul, Baill2, Klein].

While methods cited above are configuration dependent, computationally intensive, or both, the method we propose for RMMS achieves singularity avoidance while requiring negligibly more computations than the standard pseudoinverse. It is called the singularity robust inverse [Nakamura]. The pseudoinverse solution is problematic in the neighborhood of a singularity. In an effort to converge to an exact solution, the pseudoinverse may generate an infeasible solution. That is, it may generate a solution for which one, or more, of the **dq** values is so large that it cannot be physically realized. The singularity robust inverse method circumvents this problem by providing continuous and feasible solutions even at, or in the neighborhood of, singular points.

The singularity robust inverse is based upon an evaluation index,

$$\mathbf{d\phi} = \begin{pmatrix} \mathbf{dx} - \mathbf{J \cdot dq} \\ \mathbf{dq} \end{pmatrix}, \tag{5}$$

which simultaneously considers the exactness of the solution, as measured by the top term, and the feasibility of the solution, as measured by the bottom term. When solving the inverse kinematics problem one must find the minimum weighted Euclidean norm of the evaluation index. The weighting of the terms in the evaluation index manifests itself with the scale factor λ. The singularity robust inverse, $\mathbf{J}^*$ becomes:

$$\mathbf{J}^* = \mathbf{J}^T(\mathbf{J}^T\mathbf{J} + \lambda \mathbf{I})^{-1}. \tag{6}$$

In the next section we discuss a technique for choosing the the parameter λ.

3.3. A Method for Choosing the Scale Factor

In order to employ the singularity robust inverse for RMMS, we must develop a method to automatically generate an appropriate scale factor for any manipulator. The scale factor, λ, must have a large value in the neighborhood of singular points and must be small value, or zero, far from singular points. This is achieved by computing λ as [Nakamura]:

$$\lambda = \begin{cases} \lambda_0\left(1 - \frac{\omega}{\omega_0}\right) & \text{if } \omega < \omega_0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where $\omega = \sqrt{determinant(\mathbf{J}\cdot\mathbf{J}^{\mathbf{T}})}$ is a manipulatability measure for the manipulator [Yoshikawa], λ_0 is the magnitude of the scale factor at singular points, and ω_0 is a threshold which represents the neighborhood of singular points. Equation (7) automatically adjusts λ according to the manipulator's distance from a singular point.

To experimentally implement the above method it is necessary to choose values for the parameters λ_0 and ω_0. Further, the choice of these parameters must be configuration independent and work without a priori knowledge of the location of manipulator's singularities or kinematic parameters. While the value of ω approaches zero as the manipulator approaches a singular point, it's absolute magnitude is dependent on the the dimensions and the units of measure of the links and joints of the manipulator. For example, an ω of 10^2 may imply that one manipulator is near a singular point, but another manipulator, which has much smaller dimensions, may be far from one. In order to remove the dependency of ω on the units of measure and the absolute values of the kinematic lengths, we have introduced the idea of a scaling a manipulator. Scaling is accomplished by dividing all the kinematic lengths by the largest length of a manipulator. This forces all the kinematic lengths to lie between zero and one thus diminishing the disparity in the magnitudes of ω between different manipulators. However, different scaled manipulators may still generate vastly different ω values.

The singularity robust inverse chooses an absolute threshold value to specify ω_0. As mentioned before this choice is manipulator dependent. In order to alleviate this difficulty, we propose checking for a sudden drop in the value of ω between iterations. This is motivated by the observation that as a manipulator approaches singular configuration the value of ω decreases dramatically. We detect the neighborhood of a singularity when the ratio $\frac{\omega_{k+1}}{\omega_k}$ falls below a threshold μ. That is, we examine the ratio of ω between the k^{th} and the $k+1^{th}$ iterations of the Newton-Raphson algorithm.

Based upon the above discussion, the equation for computing the scale factor λ (for a scaled manipulator) is:

$$\lambda = \begin{cases} \lambda_0\left(1 - \frac{\omega_{k+1}}{\omega_k}\right) & \text{if } \frac{\omega_{k+1}}{\omega_k} < \mu \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

Our experiments with the above technique suggest $\mu \approx 0.1$ to be reasonable value.

We choose λ_0 based on the tradeoff that is the premise for the singularity robust inverse method. Namely, by adding a larger scale factor we make the solution less exact, but more feasible or robust. In order to generate a less exact solution we must increase ε. (Recall ε is the convergence error tolerance for the Newton-Raphson algorithm.) While increased error tolerance is acceptable for many applications, we cannot assume so for the general case. Alternatively, we maintain the error tolerance and increase the number of iterations of the Newton-Raphson algorithm until the error is less than ε.

Before choosing a value for λ_0 we must determine how large λ can be before the system fails to converge. In order for the Newton-Raphson iteration to converge, the residual error must be less than the error tolerance ε. Therefore, λ must be also be less than ε. Rather than defining an absolute value for λ_0, we propose setting λ_0 equal to one order of magnitude smaller than ε ($\lambda_0 = 0.1\varepsilon$). This choice is based upon our experimental results with $\mu = 0.1$.

4. Summary

In this paper we have describe the design of an RMMS. The feasibility of such a system has been demonstrated through the construction of a prototype RMMS built using readily available commercial components. A powerful computer control system with both real-time scheduling and Unix compatibility has also been built, and used to control the current RMMS manipulator.

As part of the effort to develop reconfigurable control programs, an algorithm for automatic forward and reverse kinematics generation has been implemented and tested. The algorithm is implemented as a computer program, which can find the Denavit-Hartenberg parameters for an arbitrary configuration manipulator, and then perform an iterative inverse kinematics solution. The inverse kinematics algorithm has been extended to work for redundant manipulators. The extended algorithm generates manipulator solutions which avoid singular positions. Both algorithms have been optimized for computational efficiency and robustness, and have been implemented on an Motorla 68020/68881 based single board computer, at rates on the order of 20 Hz.

References

[1] J. Baillieul, J. Hollerbach, R. Brockett.
Programming and Control of Kinematically Redundant Manipulators.
Proc. 23rd Conference on Decision and Control :768-774, December, 1984.

[2] J. Baillieul.
Kinematic Programming Alternatives for Redundant Manipulators.
IEEE International Conference on Robotics and Automation 1:722-728, March, 1985.

[3] P.H. Chang.
A Closed-form Solution for the Control of Manipulators with Kinematic Redundancy.
IEEE Internaltional Conference on Robotics and Automation 1:9-14, April, 1986.

[4] L. Kelmar and P. Khosla.
Automatic Generation of Kinematics for a Reconfigurable Modular Manipulator System.
In *IEEE Conference on Robotics and Automation*. IEEE, April, 1988.

[5] P.K. Khosla, C.P. Neuman, and F.B. Prinz.
An Algorithm for Seam Tracking Applications.
The International Journal of Robotics Research 4(1):27-41, Spring, 1985.

[6] Khosla, P. K.
Real-Time Control and Identification of Direct-Drive Manipulators.
PhD thesis, Department of Electrical and Computer Engineering, Carnegie-Mellon University, August , 1986.

[7] C. A. Klein and C-H Huang.
Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators.
IEEE Trans. on Systems, Man, and Cybernetics SMC-13(3):245-250, March/April, 1983.

[8] Beims, Bob.
The Floating-Point Performance Standard Gets Even Faster!
In (editor), *WESCON 1986 Professional Program Papers, Session 35/1*. Electronic Conventions, 1986.

[9] Y. Nakamura and H. Hanafusa.
Inverse Kinematic Solutions with Singularity Robustness for Robot Manipulator Control.
Journal of Dynamic Systems, Measurement, and Control 108:163-171, September, 1986.

[10] B. Noble and J.W. Daniel.
Applied Linear Algebra.
Prentice Hall, NJ., 1977.
Second Edition.

[11] Paul, R. P.
Robot Manipulators : Mathematics, Programming and Control.
MIT Press, Cambridge, MA, 1981.

[12] Pieper, D. L.
The Kinematics of Manipulators under Computer Control.
PhD thesis, Department of Computer Science, Stanford University, 1968.

[13] Anon.
Technical Brochure on Modular Arms
Robotics Research Inc., Ohio, 1987.

[14] Anon.
The SUN-3 Family: An Overview
SUN Mircosystems Inc., California, 1986.

[15] Walker, M.
A Survey of Research Robots.
Technical Report, University of Michigan, Ann Arbor, 1987.

[16] W.A. Wolovich.
Robotics: Basic Analysis and Design.
Holt, Rinehart and Winston, New York, 1987.

[17] Wurst, K. H.
The Conception and Construction of a Modular Robot System.
In *Proceedings of the 16-th International Symposium on Industrial Robotics*, pages 37-44. ISIR, 1986.

[18] Yoshikawa, T.
Manipulability of Robotic Mechanisms.
In *Proceedings of the Second International Symposium on Robotics Research.* MIT, Kyoto, Japan, August 20-23, 1985.