Robot Manipulator Prototyping
(Complete Design Review)

Mohamed Dekhil, Tarek M. Sobh, Thomas C. Henderson, Anil Sabbavarapu, and Robert Mecklenburg[1]

UUSC-94-010

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

March 30, 1994

Abstract

Prototyping is an important activity in engineering. Prototype development is a good test for checking the viability of a proposed system. Prototypes can also help in determining system parameters, ranges, or in designing better systems. The interaction between several modules (e.g., S/W, VLSI, CAD, CAM, Robotics, and Control) illustrates an interdisciplinary prototyping environment that includes radically different types of information, combined in a coordinated way. Developing an environment that enables optimal and flexible design of robot manipulators using reconfigurable links, joints, actuators, and sensors is an essential step for efficient robot design and prototyping. Such an environment should have the right "mix" of software and hardware components for designing the physical parts and the controllers, and for the algorithmic control of the robot modules (kinematics, inverse kinematics, dynamics, trajectory planning, analog control and digital computer control). Specifying object-based communications and catalog mechanisms between the software modules, controllers, physical parts, CAD designs, and actuator and sensor components is a necessary step in the prototyping activities. We propose a flexible prototyping environment for robot manipulators with the required subsystems and interfaces between the different components of this environment.

[1]This work was supported in part by DARPA grant N00014-91-J-4123, NSF grant CDA 9024721, and a University of Utah Research Committee grant. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

Contents

List of Figures

List of Tables

1 Introduction

In designing and building a robot manipulator, many tasks are required, starting with specifying the tasks and performance requirements, determining the robot configuration and parameters that are most suitable for the required tasks, ordering the parts and assembling the robot, developing the necessary software and hardware components (controller, simulator, monitor), and finally, testing the robot and measuring its performance.

Our goal is to build a framework for optimal and flexible design of robot manipulators with software and hardware systems and modules which are independent of the design parameters and which can be used for different configurations and varying parameters. This environment is composed of several subsystems. Some of these subsystems are:

- Design.
- Simulation.
- Control.
- Monitoring.
- Hardware selection.
- CAD/CAM modeling.
- Part Ordering.
- Physical assembly and testing.

Each subsystem has its own structure, data representation, and reasoning strategy. On the other hand, much of the information is shared among these subsystems. To maintain the consistency of the whole system, an interface layer is proposed to facilitate the communication between these subsystems, and set the protocols that enable the interaction between the subsystems to take place.

This project involved the interaction and cooperation of several different research groups. The robotics group (Prof. Thomas Henderson, Prof. Tarek Sobh, Prof. Sam Drake and myself), was involved in the design and analysis of the prototype robot, and also the implementation of the necessary software systems for the prototyping environment and for controlling and simulating the three-link robot. The Alpha_1 group, represented by Mircea Cormos was involved in designing the CAD/CAM model for the robot using the Alpha_1 CAGD system. The VLSI group, represented by Prof. Kent Smith and Anil Sabbavarapu, helped in the analysis stage, particularly, in making the decision of using hardware vs. software solutions. Also this group was involved in the design of the communication circuitry

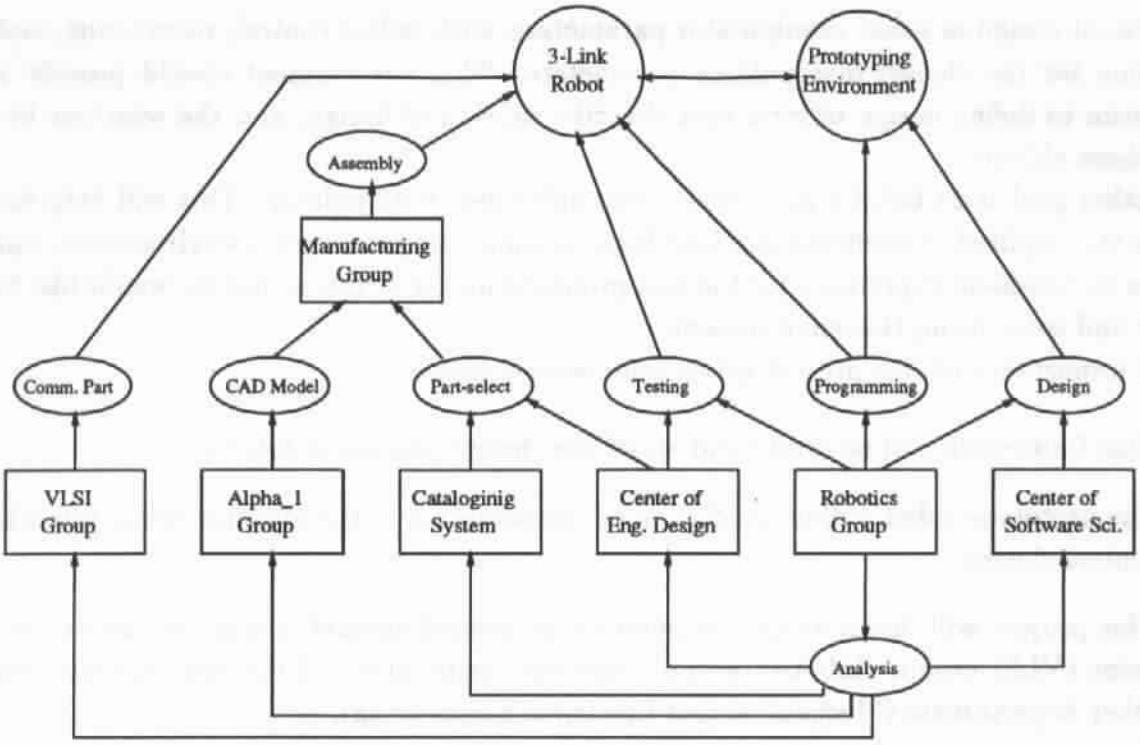


Figure 1: The interaction between the groups involved in the prototyping activity.

between the robot and the workstation. The Center of Software Science (CSS), represented by Prof. Robert Mecklenburg, helped in the design and analysis of the prototyping environment with the required communication protocols and database analysis. The Center of Engineering Design (CED), represented by Prof. Sanford Meek, was involved in selecting the electrical and electronic components and helping out in the overall design and testing procedures for the robot manipulator. Finally, the manufacturing group at the Advanced Manufacturing Lab (AML), represented by Mircea Cormos, the AML manager, Prof. Sam Drake, and Prof. Sanford Meek, was involved in the manufacturing and assembly of the robot. Besides these groups, there was cooperation between the departments of Computer Science and Mechanical Engineering in selecting the required components for the robot. A cataloging system has been recently developed by Prof. Don Brown and Prof. Robert Mecklenburg that automates the selection process for some of the parts, and we would like to incorporate this system with the part-ordering subsystem in the prototyping environment. Figure 1 shows the interaction between these groups during this project.

1.1 Objectives

The objective of this research project is to explore the basis for a consistent software and hardware environment, and a flexible framework that enables easy and fast modifications,

and optimal design of robot manipulator parameters, with online control, monitoring, and simulation for the chosen manipulator parameters. This environment should provide a mechanism to define design objects that describe aspects of design, and the relations between those objects.

Another goal is to build a prototype three-link robot manipulator. This will help determine the required subsystems and interfaces to build the prototyping environment, and will give us hands-on experience for the real problems and difficulties that we would like to address and solve using this environment.

The importance of this project arises from several points:

- This framework will facilitate and speed the design process of robots.
- The prototype robot will be used as an educational tool in the robotics and automatic control classes.
- This project will facilitate the cooperation of several research groups in the department (VLSI group, Robotics group), and the cooperation of the department with other departments (Mechanical and Electrical Engineering).
- This project will establish a basis and framework for design automation of robot manipulators.

A brief background of robot design and modules is presented in Section 2 with the related work in this area. A review about the current research efforts in building special hardware architectures for robotic applications is represented in Section 3. A detailed description of prototyping and simulating a three-link robot manipulator is presented in Section 4. The communication between the robot and the workstation is discussed in detail in Section 5. The optimal design for robot manipulators is discussed, and the proposed optimal design system is described and investigated in Section 6. Section 7 describes the prototyping environment components such as the interface between the systems and the required representations to implement this interface (e.g., knowledge base, object oriented scheme, rule-based reasoning, etc.). Section 8 shows some examples and results of the implemented systems. In Section 9, conclusions from the work are presented along with possible future extensions. The dynamics equations for the three-link robot, before and after simplifications, are described in Appendix a and Appendix B. The assembly program used in the communication between the robot and the workstation is described in Appendix C.

2 Background and Related Work

2.1 Phases of Building a Robot

The process of building a robot can be divided into several phases as follows:

1. **Design Phase:** which includes the following tasks:
   - Specify the required robot tasks.
   - Choose the robot parameters.
   - Set the control equation and the trajectory planning strategy.
   - Study the singular points.
2. **Simulation Phase:** test the behavior and the performance of the chosen manipulator.
3. **Prototyping and Testing Phase:** test the behavior and performance, and compare it with the simulated results.
4. **Manufacturing Phase:** order the required parts and manufacture the actual robot.

2.2 Robot Modules and Parameters

Controlling and simulating a robot is a process that involves a large number of mathematical equations. To be able to deal with the required amount of computation, it is better to divide them into modules, in which each module accomplishes a certain task. The most important modules, as described in [7], are kinematics, inverse kinematics, dynamics, trajectory generation, and linear feedback control. In the following sections, we will briefly describe each of these modules, and the parameters involved in each.

2.2.1 Forward Kinematics

This module is used to describe the static position and orientation of the manipulator linkages. There are two different ways to express the position of any link: using the *Cartesian space*, which consists of position (x, y, z), and orientation, which can be represented by a 3×3 matrix called the rotation matrix; or using the *joint space*, by representing the position by the angles of the manipulator's links. Forward kinematics is the transformation from joint space to Cartesian space.

This transformation depends on the configuration of the robot (i.e., link lengths, joint positions, type of each joint, etc.). In order to describe the location of each link relative to its neighbor, a frame is attached to each link, then we specify a set of parameters that

characterizes this frame. This representation is called *Denavit-Hartenberg notation.* See [7] for more details.

One approach to the problem of kinematics analysis is described in [45], which is suitable for problems where there are one or more points of interest on every link. This method also generates a systematic presentation of all equations required for position, velocity, and acceleration, as well as angular velocity and angular acceleration for each link.

2.2.2 Inverse Kinematics

This module solves for the joint angles given the desired position and orientation in Cartesian space. This is a more complex problem than forward kinematics. The complexity of this problem arises from the nature of the transformation equations, which are nonlinear. There are two issues in solving these equations: *existence of solutions* and *multiple solutions.* A solution can exist only if the given position and orientation lies within the workspace of the manipulator's end-effector. By workspace, we mean all points in space that can be reached by the manipulator's end-effector. On the other hand, the problem of multiple solutions forces the designer to set a criterion for choosing one solution, e.g., a good choice is the solution that minimizes the amount that each joint is required to move.

There are two methods for solving the inverse kinematics problem: *closed form solutions* and *numerical solutions.* Numerical solutions are much slower than closed form solutions, but, for some configurations it is too difficult to find a closed form solution. In our case, we will use closed form solutions, since our models are three link manipulators with easy closed form formulas.

A software package called SRAST (Symbolic Robot Arm Solution Tool) that symbolically solves the forward and inverse kinematics for n-degree of freedom manipulators has been developed by Herrera-Bendezu, Mu, and Cain [18]. The input to this package is the Denavit-Hartenberg parameters, and the output is the direct and inverse kinematics solutions. Another method of finding symbolic solutions for the inverse kinematics problem was proposed in [47]. Kelmar and Khosla proposed a method for automatic generation of forward and inverse kinematics for a reconfigurable manipulator system [23].

2.2.3 Dynamics

Dynamics is the study of the torques required at each joint to cause the manipulator to move in a certain manner. It is also concerned with the way in which a manipulator moves when certain torques are applied to its joints. The serial chain nature of manipulators makes it easy to use simple methods in dynamic analysis.

There are two problems related to the dynamics of a manipulator: *controlling* the manipulator, and *simulating* the motion of the manipulator. In the first problem, we have a set of required positions for each link, and we want to calculate the required torques to

be applied at each joint. This is called *inverse dynamics*. In the second problem, we are given a set of torques applied to each link, and we wish to calculate the new position and the velocities during the motion of each link. The latter is used to simulate a mathematical manipulator model before building the physical model, which makes it possible to update and modify the design without the cost of changing or replacing any physical parts.

The dynamics equations for any manipulator depend on the following parameters:

- The mass of each link.
- The mass distribution for each link, which is called *the inertia tensor*, which can be thought of as a generalization of the scalar moment of inertia of an object.
- Length of each link.
- Joint type (revolute or prismatic).
- Manipulator configuration and joint locations.

The dynamics model we are using to control the manipulator is in the form:

$$\tau = M(\theta)\ddot{\theta} + V(\theta, \dot{\theta}) + G(\theta) + F(\theta, \dot{\theta})$$

To simulate the motion of a manipulator we must use the same model we have used in controlling that manipulator. The model for simulation will be in the form:

$$\ddot{\theta} = M^{-1}(\theta)[\tau - V(\theta, \dot{\theta}) - G(\theta) - F(\theta, \dot{\theta})]$$

The dynamics module is the most time consuming part among the manipulator's modules. That is because of the tremendous amount of calculation involved in the dynamics equations. This fact makes the dynamics module a good candidate for hardware implementation, to enhance the performance of the control and/or the simulation system.

There are some parallel algorithms to calculate the dynamics of a manipulator. One approach described in [37], is to use multiple microprocessor systems, where each one is assigned to a manipulator link. Using a method called *branch-and-bound*, a schedule of the subtasks of calculating the input torque for each link is obtained. The problem with this method is that the scheduling algorithm itself was the bottleneck, thus limiting the total performance. Several other approaches have been suggested [29, 30, 44] based on a multiprocessor controller, and pipelined architectures to speed the calculations. Hashimoto and Kimura [17] proposed a new algorithm called *the resolved Newton-Euler algorithm* based on a new description of the Newton-Euler formulation for manipulator dynamics. Another approach was proposed by Li, Hemami, and Sankar [34] to drive linearized dynamic models

about a nominal trajectory for the manipulator using a straightforward Lagrangian formulation. An efficient structure for real-time computation of the manipulators dynamics was proposed by Izaguirre, Hashimoto, Paul and Hayward [20]. The fundamental characteristic of this structure is the division of the computation into a high-priority synchronous task and low-priority background tasks, possibly sharing the resources of a conventional computing unit based on commercial microprocessors.

2.2.4 Trajectory Generation

This module computes a multidimensional trajectory which describes the manipulator's position, velocity, and acceleration for each link. This module includes the human interface problem of describing the desired behavior of the manipulator. The complexity of this problem arises from the wide meaning of *manipulator's behavior*. In some applications we might need to specify only the goal position, whereas in some other applications, we might need to specify the velocity with which the end effector should move. Since trajectory generation occurs at run time on a digital computer, the trajectory points are calculated at a certain rate, called the *path update rate*. We return to this issue when we consider speed.

There are several strategies to calculate trajectory points which generate a smooth motion for the manipulator. It is important to guarantee this smoothness of the motion due to physical considerations such as the required torque that causes this motion, the friction at the joints, and the frequency of update required to minimize the sampling error.

One of the simplest methods is *cubic polynomials*, which assumes a cubic function for the angle of each link, by differentiating this equation the velocity and acceleration are computed (see [7]).

2.3 Linear Feedback Control

We will use a linear control system in our design, which is an approximation of the nonlinear nature of the dynamics equations of the system, which are more properly represented by nonlinear differential equations. This is a reasonable approximation, and it is used in current industrial practice.

We will assume that there are sensors at each joint to measure the joint angle and velocity, and there is an actuator at each joint to apply a torque on the neighboring link. Our goal is to cause the manipulator joints to follow a desired trajectory. The readings from the sensors will constitute the feedback of the control system. By choosing appropriate gains we can control the behavior of the output function representing the actual trajectory generated. Minimizing the error between the desired and actual trajectories is our main concern. Figure 2 shows a high level block diagram of a robot control system.

When we talk about control systems, we should consider several issues related to that field, such as: *stability*, *controllability*, and *observability*. For any control system to be stable, its poles should be negative, since the output equation contains terms of the form $k_i e^{p_i}$; if p_i is positive, the system is said to be unstable. We can guarantee the stability of the system by choosing certain values for the feedback gains.

We will assume a second order control system of the form:

$$m\ddot{\theta} + b\dot{\theta} + k\theta.$$

Another desired property of the control system is that it be *critically damped*, which means that the output will reach the desired position in minimum time without overshooting. This can be accomplished by making $b^2 = 4mk$. Figure 3 shows the three types of damping: *underdamped*, *critically damped*, and *overdamped*.

Figure 4 shows a block diagram for the controller, and the role of each of the robot modules in the system.

More about robot control can be found in [3, 33, 46].

2.3.1 Local PD Feedback Control

Most of the feedback algorithms used in the current control system are digital implementation of a proportional plus derivative (PD) control. In industrial robots, a local PD feedback control law is applied at each joint independently. The advantages of using a PD controller are the following:

- Very simple to implement.
- Does not require the identification of robot parameters.
- Suitable for real-time control since it has very few computations compared to the complicated nonlinear dynamic equations.
- The behavior of the system can be controlled by changing the feedback gains.

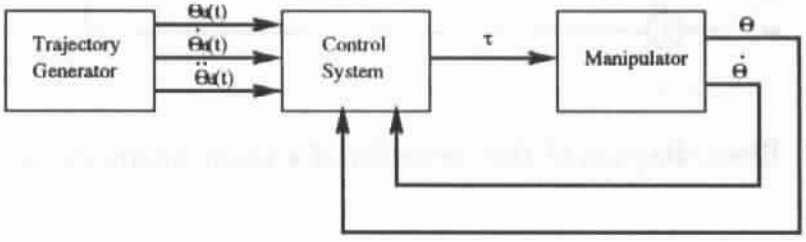


Figure 2: High-level block diagram of a robot control system.

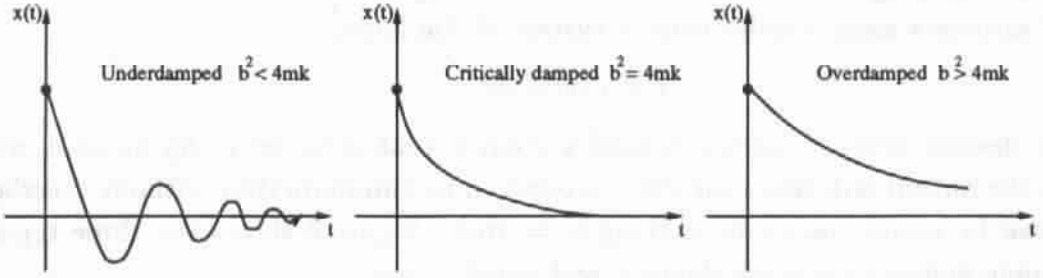


Figure 3: Different types of damping in a second order control system.

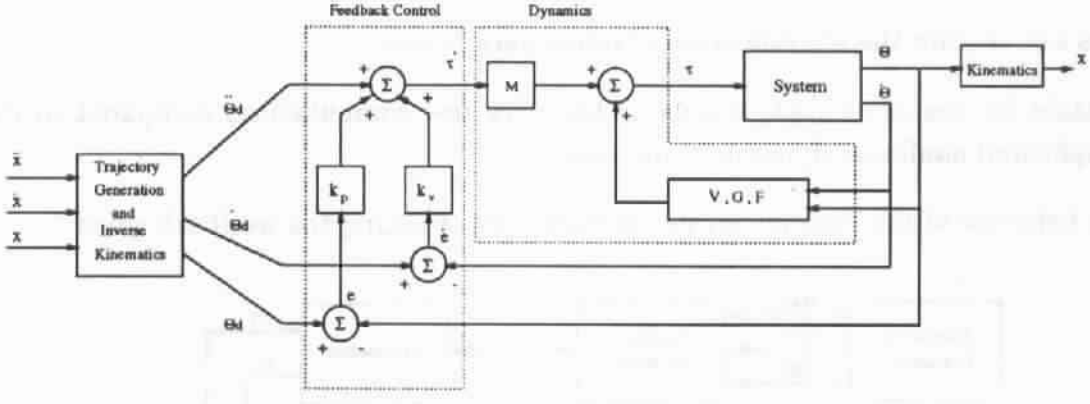


Figure 4: Block diagram of the controller of a robot manipulator.

On the other hand, there are some disadvantages of using a PD controller instead of the dynamic equations such as:

- High update rate is required to achieve reasonable accuracy.
- Dynamic equations should be used to simulate the robot manipulator behavior
- There is always trade-off between static accuracy and the overall system stability.
- Using local PD feedback law at each joint independently does not consider the couplings of dynamics between robot links.

Some ideas have been suggested to enhance the usability of the local PD feedback law for trajectory tracking. One idea is to add a lag-lead compensator using frequency response analysis [5]. Another method is to build an inner loop stabilizing controller using a multivariable PD controller, and an outer loop tracking controller using a multivariable PID (proportional, integral, and derivative) controller [53].

In general, using a local PD feedback controller with high update rates can give an acceptable accuracy for trajectory tracking applications. It was proved that using a linear PD feedback law is useful for positioning and trajectory tracking [21].

2.3.2 Continuous vs. Discrete Time Control

In computer-controlled systems, the calculated actuator forces are not continuous functions in time any more. This is because of the time needed by the computer to perform the required calculations. In this case, we can study the system using *digital control* theory which takes the calculation time into account when analyzing the system. To be able to use the continuous model, we must use high update rates (i.e., reduce the computation time). This can be achieved by using a faster computer, and/or using parallel architectures and using some parallel algorithm to calculate the complicated parts in the computations (usually the dynamics of the system). The effect of choosing the update rate on the system performance and stability is discussed in Section 2.4.

Another method is to use a mixture of continuous and discrete control for the system. This can be done by using the computer to generate the required trajectory and the torques for the actuators in discrete time, and an analog PID controller in the interval between the computer samples. This will enable us to assume a continuous control law and will minimize the error during the computation time.

2.3.3 Disturbance Rejection

In any real-time control system, there is always some amount of external noise $f_{dist}(t)$, and usually this noise is stochastic in nature. The distribution and magnitude of this noise

depends on the working environment, and sometimes it is too difficult to prevent the noise from happening, but we can modify the control model to reduce the effect of such noise to an acceptable degree. This noise can be modeled using statistical measures and some assumptions about its nature. To deal with this noise we must assume that it is *bounded*, that is, there is a constant a such that:

$$\max_i f_{dist}(t) < a$$

This maintains the property of a stable linear system known as *bounded-input bounded-output* (BIBO) stability.

As a simple case, assume that f_{dist} is a constant. In this case, the steady state error can be calculated by analyzing the system at rest (i.e., set all derivatives to zero) as follows:

$$k_p e = f_{dist}$$

or

$$e = f_{dist}/k_p$$

The value of e here represents the steady state error of the system. From the last equation, it is clear the increasing kp will decrease the steady state error. On the other hand, there is a limit on the value of k_p to maintain the stability of the system.

Another way to reduce (and sometimes eliminate) the steady state error, is by adding an integral term to the control low. That is what is known as the (PID) Proportional, Integral, Derivative controller. By adding this term, the steady state error can be calculated as follows:

$$k_p e + k_i \int e dt = f_{dist}$$

or

$$k_p \dot{e} + k_i e = \dot{f}_{dist}$$

We assumed, however, that f_{dist} is a constant, thus, $\dot{f}_{dist} = 0$, which gives:

$$k_i e = 0$$

So, the addition of this integral element can eliminate constant disturbances.

2.4 Speed Considerations

There are several factors that affect the desired speed (frequency of calculations), the maximum speed we can attain using software solutions, and the required hardware we need to build if we are to use a hardware solution. The desired frequency of calculation depends on the type and frequency of input, the noise in the system, and the required output accuracy.

2.4.1 Types of Inputs

The user interface to the system should allow the user to specify the desired motion of the manipulator in different ways depending on the nature of the job the manipulator is designed to do. The following are some of the possible input types the user can use:

- Move from point x_0, y_0, z_0 to point x_d, y_d, z_d in Cartesian space.
- Move in a predefined position trajectory $[x_i, y_i, z_i]$. This is called *position planning.*
- Move in a predefined velocity trajectory $[\dot{x}_i, \dot{y}_i, \dot{z}_i]$. This is called *velocity planning.*
- Move in a predefined acceleration trajectory $[\ddot{x}_i, \ddot{y}_i, \ddot{z}_i]$. This is called *force control.*

The input type will affect the placement of the inverse kinematics module: outside the update loop, as in the first case, or inside the update loop, as in the last three cases. For the last three cases we have two possible solutions; we can include the inverse kinematics module in the main update loop as we mentioned before, or we can plan ahead in the joint space before we start the update loop. We should calculate the time required for each case plus the time required to make a decision.

2.4.2 Desired Frequency of the Control System

We must decide on the required frequency of the system. In this system we have four frequencies to be considered:

- Input frequency, which represents the frequency of changes to the manipulator status (position, velocity, and acceleration).
- Update frequency, representing the speed of calculations involved.
- Sensing frequency, which depends on the A/D converters that feed the control system with the actual positions and velocities of the manipulator links.
- Noise frequency: since we are dealing with a real-time control system, we must consider different types of noise affecting the system such as: input noise, system noise, and output noise (from the sensors).

2.4.3 Error Analysis

The error is the difference between the desired and actual behavior of the manipulator. In any physical real-time control system, there is always a certain amount of error resulting from modeling error or different types of noise. One of the design parameters is the maximum allowable error. This depends on the nature of the tasks the manipulator is designed

to accomplish. For example, in the medical field the amount of error allowed is much less than in a simple laboratory manipulator. The update frequency is the most dominant factor in minimizing the error. It is clear that increasing the update frequency results in decreasing the error. The update frequency, however, is limited by the speed of the machine used to run the system. Khosla performed some experiments to study the effect of changing the control sampling rate on the performance of the manipulator behavior [25] and showed that increasing the update rate decreases the error.

2.5 Optimal Design of Robot Manipulators

It is important to choose the parameters of a robot manipulator (configuration, dimension, motors, etc.) that are most suitable for the required robot tasks. Considerable research has been done in this area. Depkovich and Stoughton [11] proposed a general approach for the specification, design and validation of manipulators. The concept of *Reconfigurable Modular Manipulator System* (RMMS) was proposed by Khosla, Kanade, Hoffman, Schmitz, and Delouis [24] at Carnegie Mellon University. There goal is to create a complete manipulator system, including mechanical and control hardware, and control algorithms that are automatically and easily reconfigured.

Designing an *optimal* manipulator is not yet well defined, and it depends on the definition and criterion of optimality. There are several techniques and methodologies to formalize this optimization problem by creating some objective functions that satisfy certain criteria, and solving these functions with the existence of some constraints.

One criterion that is used is a kinematic criterion for the design evaluation of manipulators by establishing quantitative kinematic distinction among a set of designs [6, 40, 41]. Another criterion is to achieve optimal dynamic performance; that is to select the link lengths and actuator sizes for minimum time motions along specified trajectory [38, 49].

TOCARD (Total Computer-Aided Design System of Robot Manipulators) is a system designed by Takano, Masaki, and Sasaki [52] to design both fundamental structure (degrees of freedom, arm length, etc.), and inner structure (arm size, motor allocation, motor power, etc). They describe the problem as follows: there is a set of design parameters, a set of objective functions, and a set of Gavin data (constraints). The design parameters are:

- Degrees of freedom.
- Joint type and its sequence.
- Arm length and offset.
- Arm cross-sectional dimensions.
- Motor allocations.

- Joint mechanisms and transmission mechanisms.
- Reduction gears.
- Motors.

The objective functions for the design of robot arm are as follows:

- Manipulability.
- Total motor power consumption.
- Arm weight.
- Total weight of robot.
- Cost.
- Workspace.
- Joint displacement limit.
- Maximum joint velocity and acceleration.
- Deflection.
- Natural frequency.
- Position accuracy.

The constraints can be:

- Workpiece and degrees of freedom of orientation.
- Maximum velocity and acceleration of workpiece.
- Position accuracy.
- Weight, gravity center and moment of inertia of workpiece.
- Dimensional data of hand and grasping manner of workpiece.

Hollerbach proposed an optimum kinematic design for a seven-degree of freedom manipulator [19].

2.6 Integration of Heterogeneous Systems

To integrate the work among different teams and sites working in such a large project, there must be some kind of synchronization to facilitate the communication and cooperation between them. A concurrent engineering infrastructure that encompasses multiple sites and subsystems, called Pallo Alto Collaborative Testbed (PACT), was proposed in [8]. The issues discussed in that work were:

- Cooperative development of interfaces, protocols, and architecture.
- Sharing of knowledge among heterogeneous systems.
- Computer-aided support for negotiation and decision-making.

An execution environment for heterogeneous systems called "InterBase" was proposed in [4]. It integrates preexisting systems over a distributed, autonomous, and heterogeneous environment via a tool-based interface. In this environment each system is associated with a *Remote System Interface (RSI)* that enables the transition from the local heterogeneity of each system to a uniform system-level interface.

Object orientation and its applications to integrate heterogeneous, autonomous, and distributed systems are discussed in [43]. The argument in this work is that object-oriented distributed computing is a natural step forward from the client-server systems of today. A least-common-denominator approach to object-orientation as a key strategy for flexibly coordinating and integrating networked information processing resources is also discussed. An automated, flexible and intelligent manufacturing based on object-oriented design and analysis techniques is discussed in [39], and a system for design, process planning and inspection is presented.

Several important themes in concurrent software engineering are examined in [12]. Some of these themes are:

Tools: Specific tools that support concurrent software engineering.

Concepts: Tool-independent concepts are required to support concurrent software engineering.

Life cycle: Increase the concurrency of the various phases in the software life cycle.

Integration: Combining concepts and tools to form an integrated software engineering task.

Sharing: Defining multiple levels of sharing is necessary.

A management system for the generation and control of documentation flow throughout a whole manufacturing process is presented in [13]. The method of quality assurance is used to develop this system that covers cooperative work between different departments for documentation manipulation.

A computer-based architecture program called *the Distributed and Integrated Environment for Computer-Aided Engineering* (Dice), which addresses the coordination and communication problems in engineering, was developed at the MIT Intelligent Engineering Systems Laboratory [51]. The Dice project addresses several research issues such as, frameworks, representation, organization, design methods, visualization techniques, interfaces, and communication protocols.

Some important topics in software engineering, such as the lifetime of a software system, analysis and design, module interfaces and implementation, and system testing and verification, can be found in [28]. Also, a report about integrated tools for product, and process design can be found in [55].

In the environment we are proposing, several subsystems are communicating through a *central interface layer* (CI), and each subsystem has a *subsystem interface* (SSI) responsible for data transformation between the subsystem and the CI. The flexibility of this design arises from the following points:

- Adding new subsystem can be achieved by writing an SSI for this new subsystem, adding it to the list of the subsystems in the CI. There are no changes required to the other SSIs.
- Removing a subsystem only requires removing its name from the subsystems list in the CI.
- Any changes in one of the subsystems require changing the corresponding SSI to maintain correct data transformation to and from this subsystem.

More about this design is discussed in Section 7.

3 Special Computer Architecture for Robotics

When we design real-time systems that involves a huge number of floating point calculations, the speed becomes an important issue. In such situations, a hardware solution might be used to achieve the desired speed. In the following sections we will investigate the different solutions and platforms proposed for robotics.

3.1 Design Issues

VLSI design for robot application is a complex task which requires a conceptual framework that control its complexity. Several decisions should be taken during the design process such as: What is the best architecture for this application, how specific the hardware implementation should be, what kind of tools needed to implement such design, and the cost of the design.

To be able to take such decisions, the computational needs for the applications should be analyzed and the performance requirements of the robot has to be considered.

For a generic robot system there are three major layers of computation proposed in [31]:

- Management layer which includes:
  - user interface
  - operating system support
  - resource allocation
  - coordination
- Reasoning layer which includes:
  - decision making
  - causal reasoning
  - temporal reasoning
  - geometric reasoning
  - planning
  - world modeling
- Device interaction layer includes:
  - adaptive control
  - kinematics and dynamics
  - multi-sensor fusion
  - interpretation
  - feature extraction
  - preprocessing

One of the most important tools the has been developed recently is the *hardware description languages* (DHL) which enables top-down design using high-level descriptions.

3.2 Parallel Architectures and multiprocessors

There are many forms of parallelism and concurrency which can be applied to advanced computational problems in robot control and simulation. This fact leads to the developments of many parallel architectures that utilizes this property.

A real-time robot control based on multi-processor architecture was proposed in [1]. in this design the control tasks are analyzed to obtain a lower bound on the number of mathematical operations required to generate the control signal, then a parallel computation structure is designed according to the maximum sample time based on the stability of the control system. This design can be implemented as a custom VLSI or as a systolic array based system.

An optimal design for multiple-APU (Arithmetic Processing Unit) based robot controllers is discussed in [2]. In this paper it was shown that using eight APUs, it is possible to compute the inverse kinematics, inverse dynamics and the trajectory for the PUMA arm in less than 3ms using 16.7 MHZ 68881.

A dataflow multiprocessor system for robot arm control was proposed in [15]. In this method, the maximum parallelism would require 1834 processing elements. However, a reasonable engineering solution requires 42 processing elements.

SIERA (System for Implementing and Evaluating Robotic Algorithms) is a multiprocessor system that has been developed at the Laboratory for Engineering Man/Machine Systems (LEMS) at Brown University. It incorporates a tightly coupled bus-based system (the Real-time Servo System) and a loosely coupled point-to-point network (the Armstrong Multiprocessor System). Figure 5 shows an overview of the SIERA system and Figure 6 shows the Armstrong processes. More details can be found in [22].

A parallel computer architecture for real-time control application in grasping and manipulation was proposed in [26]. In this paper a new scheduling algorithm for multiprocessor architecture based on either complete or incomplete crossbar interconnection networks is presented. The mean feature of the proposed algorithm is that it takes into account the communication delays between processors and minimizes both the execution time and the communication cost.

Several parallel architectures are proposed in [35, 36, 44, 48, 56]

3.3 Application-Specific Integrated Circuits

The increasing demand for more computation power to meet the current speed requirements of robot controllers made it clear that general purpose processors are no longer satisfactory. The recent ASIC (*Application-Specific Integrated Circuits* technology was the solution that created better opportunities for implementing real-time controller for more sophisticated robot manipulators. An overview of ASIC technology for robotics was pre-

sented in [32]. In this paper, a conceptual framework for ASIC design is presented along with the characteristics of the ASIC design.

The advantages of ASIC for robotic applications include:

- Better performance.
- Smaller size.
- Higher reliability.
- lower non-recurring cost.
- Faster turnaround time.
- Tighter design security.

ASIC includes several custom and simicustom hardware designs including:

- programmable logic devices (PLD).
- gate arrays (GA).
- standard cells (SC).
- full custom design (FC).

The main difference between these styles is the degree of design freedom in layout. The greater the degree of design freedom, the is the design effort and the longer the design turnaround time. Figure 7 summarizes the differences between ASIC and SIC (*standard integrated circuits*).

3.4 Neural Networks and Robotics

Neural networks has a unique feature of robust processing and adaptive capability in changing even in noisy environments. It is estimated that the human brain contains over 100 billion neuron. Several application has been implemented using artificial neural networks (ANN) such as pattern recognition, image processing, and machine learning.

A basic ANN model consists of a large number of neurons linked to each other with connection weights (see Figure 8).

The ANN processing can be divided into two phases:

Attribute \ Approach	SIC Design	ASIC Design
Cost constraint	Component count	Design effort
Performance limitation	Functional unit design	Data communication
Major design alternatives	Major components (e.g., processors)	Design styles (e.g., GA, SC, FC)
Coupling between design steps	Loose	Tight
Testability requirement	Nodes accessible at board level	Must be incorporated early in the design
Verification	Breadboarding	Simulation
Prototyping	Usually in-house	In cooperation with vendor
Last-minute changes	Possible	Costly
Design guidance	Informal	Strong methodology
Tools	Relatively simple	CAE intensive

Figure 7: Comparison of traditional design and ASIC design.

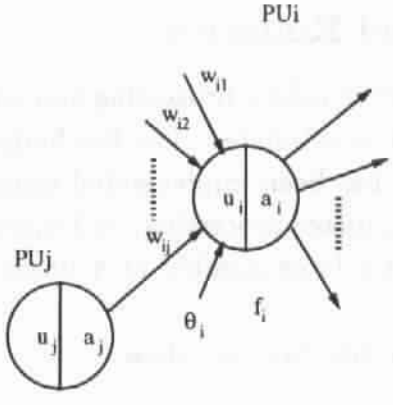


Figure 8: A basic ANN model.

Retrieving Phase: this phase performs the iterative updating of the activation values a_i. A generic iterative formulation for the updating phase is:

$$u_i(l+1) = \sum_{j=1}^{N_l} w_{ij}(l+1)a_j(l),$$
$$a_i(l+1) = f_i(u_i(l+1), \theta_i(l+1))$$

where, u_i is the input to the PU i, a_i is the activation value of PU i, w_{ij} is the effect of PU j on PU i, θ_i an external input to PE i, and f_i is a nonlinear activation function at PE i.

Learning Phase: In this phase, the synaptic weights are updated based on the input and the target training pattern using an adopted learning rule. The following is an example of learning rule:

$$w_{i,j}(l) = w_{ij}(l) + \eta \Delta w_{ij}(l)$$

where η is the update rate parameter, and $\Delta w_{ij}(l)$ is the increment of weight change.

The application of neural networks can be grouped into two classes: *optimization* and *associative retrieval/classification*. Most robot problems can be formulated as one of the two classes. For example, stereo vision for task planning, autonomous robot path planning, and position control can be formulated as optimization problems.

A ring VLSI systolic architecture for implementing ANNs with application to robotic processing was proposed in [27]. It is demonstrated that the ANNs are suitable for several robot processing applications such as: task planning, path planning, and path control. Several models of ANNs are investigated in this paper such as: single-layer feedback neural networks, competitive learning neural networks, and multi-layer feed-forward neural networks.

3.5 2400-MFLOPS Reconfigurable Parallel VLSI Processor

A new concept was introduced in [14] for a *reconfigurable floating-point multiply-adders* to reduce the latency for robot control. This reconfigurations involves direct hardware connections between the multipliers and the adders. A parallel VLSI processor composed of several processor elements (PE) was proposed. In each PE, a switching hardware is used to change the connection between the multipliers and the adders, so that the multiply-adders with a desired numbers of multipliers can be constructed.

Each PE consists of two multipliers, two adders, a local memory (LM) and a switch circuit (SC) as shown in Figure 9. The inner connection of the SC is changed every clock cycle to reconfigure the multiply-adder. Figure 10 shows an example of a reconfigured multiply-adder that contains four multipliers.

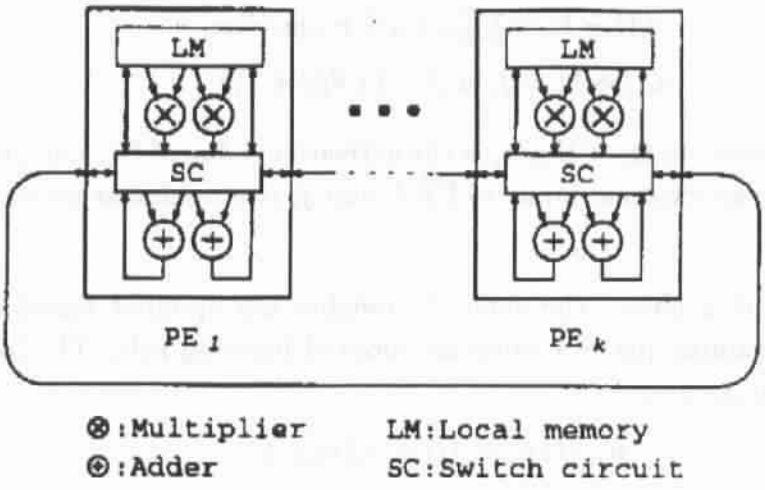


Figure 9: Reconfigurable parallel VLSI processor.

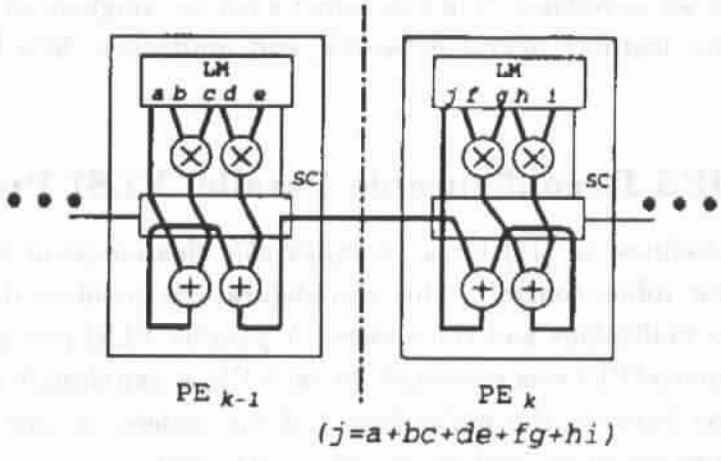


Figure 10: Reconfiguration of a multi-operand multiply-adder.

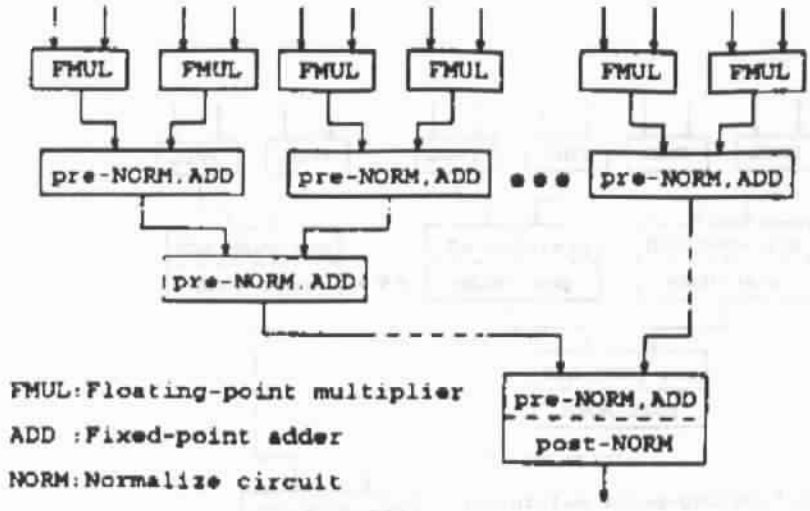


Figure 11: Reconfiguration for the floating-point multi-operand multiply-adder.

The following examples shows the speed improvement of using this processor. The latency for differential inverse kinematics (DIK) computations of twelve-DOF manipulator is about 7μ *sec* which is about 180 times faster than the latency of a parallel processor approach using general-purpose microprocessors. Also, the latency for resolved acceleration control of a twelve-DOF manipulator is 32μ *sec* which is about 60 times faster than the latency of a parallel processor approach using conventional DSPs.

Figure 11 shows the reconfigured floating-point multi-operand multiply-adder in which there is a pre-normalize circuit before each stage of the addition, and only one post-normalize circuit only in the final stage adder, this reduces the time needed for pre- and post-normalization of the operands about one half using this method in comparison with the multi-operand adder shown in Figure 12.

To perform multiplication in one clock cycle, the PE has pipeline registers as shown in Figure 13. For matrix operations, a reconfigurable parallel VLSI processor is shown in Figure 14. In this configuration, each PE has seven sixty-for-bit wide I/O channels to construct a two-dimensional linear array processor. Three I/O channels are provided for common data busses. The other four are to connect the neighboring PEs for the reconfiguration.

Figure 15 shows the chip layout of the PE, and Figure 16 shows the features of this chip.

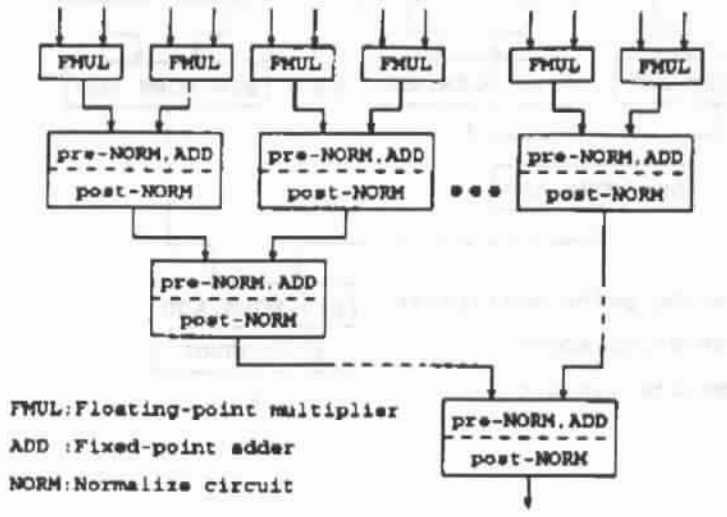


Figure 12: Conventional floating-point multi-operand multiply-adder.

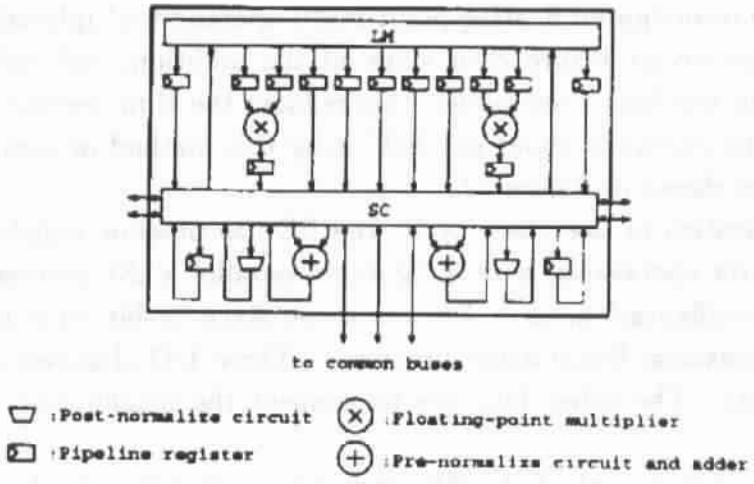


Figure 13: Structure of the PE.

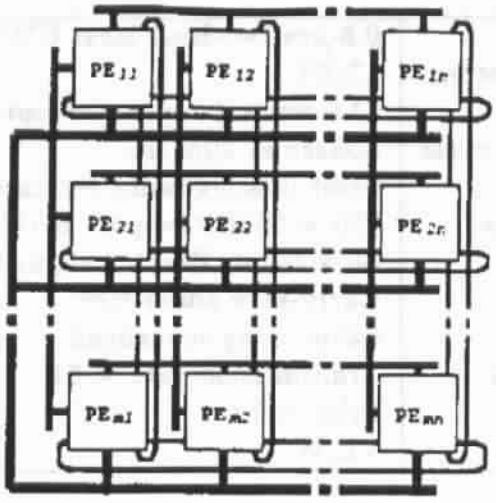


Figure 14: Reconfigurable parallel VLSI for matrix operations.

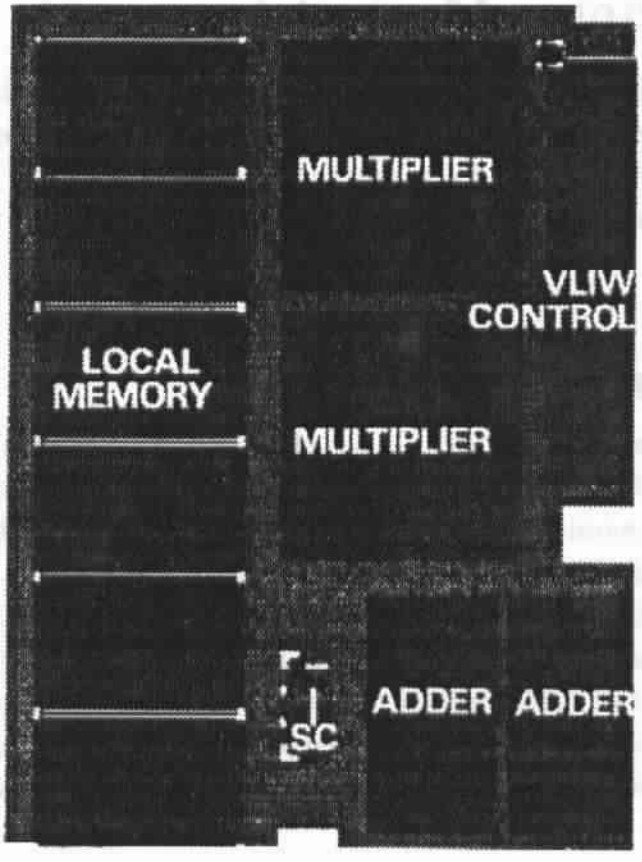


Figure 15: Chip layout of the PE.

Technology	0.8-μm two-level-layer CMOS
No. of transistors	1745K
Chip size	11.0mm × 14.8mm (except I/O)
Int. machine cycle	50$nsec$ at 20MHz
Arithmetic	64bit floating-point representation
Local memory	256 × 64bit two-port RAM × 6
Multiplier	64 × 64bit, Booth's algorithm, Carry-save adder tree × 2
Adder	64bit, carry lookahead × 2
Switch circuit	Transmission gate × 64 × 107
I/O channel	64bit × 7
Control	VLIW

Figure 16: Features of the PE.

4 Three-link Robot Manipulator

To explore the basis of building a flexible environment for robot manipulators, A three-link robot manipulator was designed. This enabled us determine the required subsystems and interfaces for such an environment. This prototype robot will be used as an educational tool in control and robotics classes.

4.1 Analysis Stage

This project was started with the study of a set of robot configurations and analyzed the type and amount of calculation involved in each of the robot controller modules (kinematics, inverse kinematics, dynamics, trajectory planning, feed-back control, and simulation). This phase was accomplished by working through a generic example for a three-link robot to compute symbolically the kinematics, inverse kinematics, dynamics, and trajectory planning; these were linked to a generic motor model and its control algorithm. This study enabled us to determine the specifications of the robot for performing various tasks, it also helped us decide which parts (algorithms) should be hardwired to achieve specific mechanical performances, and also how to supply the control signals efficiently and at what rates.

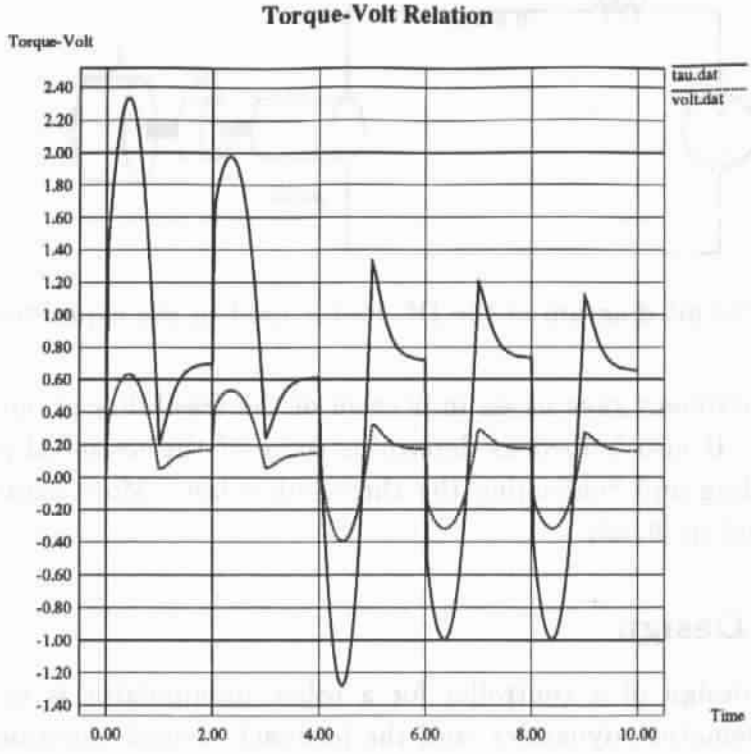


Figure 17: The relation between torque the voltage.

4.2 One Link Manipulator

Controlling a one-link robot in a real-time manner is not difficult, but on the other hand it is not a trivial task. This is the basis of controlling multi-link manipulators, and it gives an indication of the type of problems and difficulties that arise in a larger environment. The idea is to establish a complete model for controlling and simulating a one-link robot, starting from the analysis and design, through the simulation and error analysis.

A motor from the Mechanical Engineering lab was used. This motor is controlled by a PID controller. An analog I/O card, named PC-30D, connected to a Hewlett Packard PC was used to connect the motor with the serial port of the PC. This card has sixteen 12-bit A/D input channels, two 12-bit D/A output channels. There are also the card interface drivers with a Quick BASIC program that uses the card drivers to control the DC motor.

One of the problems we faced in this process was to establish the transfer function between the torque and the voltage. The motor parameters were used to form this function by making some simplifications, since some of the motor parameters have nonlinear components that make it too difficult to make an exact model. Figure 17 shows the relation between torque and voltage for a certain input sequence, and Figure 18 shows the circuit diagram of the motor and its parameters.

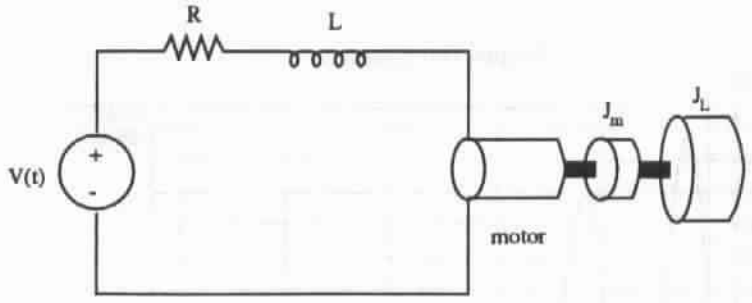


Figure 18: Circuit diagram of the DC-motor used in the experiment

In general, this experiment gave us an indication of the feasibility of our project, and good practical insight. It also helped us determine some of the technical problems that we might face in building and controlling the three-link robot. More details about this experiment can be found in [9, 50].

4.3 Controller Design

The first step in the design of a controller for a robot manipulator is to solve for its kinematics, inverse kinematics, dynamics, and the feedback control equation that will be used. Also the type of input and the user interface should be determined at this stage. We should also know the parameters of the robot, such as: link lengths, masses, inertia tensors, distances between joints, the configuration of the robot, and the type of each link (revolute or prismatic). To make a modular and flexible design, variable parameters are used that can be fed to the system at run-time, so that this controller can be used for different configurations without any changes.

Three different configurations have been chosen for development and study. The first configuration is revolute-revolute-prismatic with the prismatic link in the same plane as the first and second links. The second configuration is also revolute-revolute-prismatic with the prismatic link perpendicular to the plane of the first and second links. The last configuration is three revolute joints (see Figure 19).

The kinematics and the dynamics of the three models have been generated using some tools in the department called *genkin* and *gendyn* that take the configuration of the manipulator in a certain format and generate the corresponding kinematics and dynamics for that manipulator.[1] The kinematics and the dynamics of the three models are shown in Appendix A. One problem with the resultant equations is that they are not simplified at all; therefore, The results were simplified using the mathematical package *Mathematica*, which gives more simplified results, but still, not totally factorized. Appendix B shows the dynamics after simplifying the equations using *Mathematica*. The comparison between the

[1]These tools were developed by Patrick Dalton.

number of calculations before and after simplification will be discussed in the benchmarking section.

For the trajectory generation, The cubic polynomials method, described in the trajectory generation section, was used. This method is easy to implement and does not require much computation. It generates a cubic function that describes the motion from a starting point to a goal point in a certain time. Thus, this module will give us the desired trajectory to be followed, and this trajectory will serve as the input to the control module.

The error in position and velocity is calculated using the readings of the actual position and velocity from the sensors at each joint. Our control module simulated a PID controller to minimize that error. The error depends on several factors such as the frequency of update, the frequency of reading from the sensors, and the desired trajectory (for example, if we want to move through a angle in a very small time interval, the error will be large).

4.4 Simulation

A simulation program has been implemented to study the performance of each manipulator and the effect of varying the update frequency on the system. Also it helps to find approximate ranges for the required torque and/or voltage, and to determine the maximum velocity to know the necessary type of sensors and A/D. To make the benchmarks, as described in the next section, we did not use a graphical interface to the simulator, since the drawing routines are time consuming, and thus give misleading figures for the speed.

In this simulator, some reasonable parameters have been chosen for our manipulator. The user can select the length of the simulation, and the update frequency. The third model was used for testing and benchmarking because its dynamics are the most difficult and time consuming compared to the other two models. Table 1 shows the number of calculations in the dynamics module for each model.

4.5 Benchmarking

One important decision that had to be made was: do we need to implement some or all of the controller module in hardware? And if so which modules, or even parts of the modules,

Table 1: Number of calculations involved in the dynamics module.

	Additions	Multiplications	Divisions
Model 1	89	271	13
Model 2	85	307	0
Model 3	195	576	22

Table 2: Configuration of the machines used in the benchmark.

	SPARC-2	SPARC-10 (30)	SPARC-10 (41)	HP-700
Clock Rate(MHz)	40.0	36.0	40.0	66.0
MIPS	28.5	101.6	109.5	76.0
MFLOPS	4.3	20.5	22.4	23.0

should be hardwired? To answer these questions we chose approximate figures for the required speed to achieve a certain performance, the available machines for the controller, the available hardware that can be used to build such modules, and a time chart for each module in the system to determine the bottlenecks. This also involved calculating the number of operations in each module giving a rough estimate of the time taken by each module.

The simulator described in Section 4.4 was used to generate time charts for each module, and to compare the execution time on different machines. The machines used in this benchmarking effort include: SUN SPARCStation-2, Sun SPARCStation-10 model 30, Sun SPARCStation-10 model 41, and HP-700. Table 2 shows the configurations of the machines used in this benchmark, with the type, clock cycle rate, the MIPS and MFLOPS for each.

To generate time charts for the execution time of each module, a program called *gprof* was used. This program produces an execution profile of C, Pascal, or Fortran77 programs. It gives the execution time for each routine in the program, and the accumulated time for all the routines. Then *xgraph* was used to draw charts showing these time profiles. The simulation program was executed with an update frequency of 1000 Hz for 10 seconds, which means that each routine was called 10,000 times. From this output, it was obvious that the bottleneck was the dynamics routine and usually it took between 25% to 50% of the total execution time on the different machines.

From these results we found that the HP-700 was the fastest of all, followed by the SPARC-10 machines. After the simplification using Mathematica, the execution time increased because the results contained many different trigonometric functions, and it seemed that these machines do not use lookup tables for such functions. So, all nonbasic trigonometric functions were replaced by basic trigonometric functions. For example, $sin\ 2\theta$ was formulated as $2\ sin\ \theta\ cos\ \theta$. Using this conversion, the performance improved significantly. Figure 20 shows a speed comparison between the machines. The graph represents the speed of each machine in terms of iterations per second. The machines are SPARC-2, SPARC-10-30, SPARC-10-41, and HP-730, respectively. For each machine, the first column is the speed before any simplification, the second column is the speed after using Mathematica (notice the performance degradation here), and the third column after simplifying the trigonometric functions.

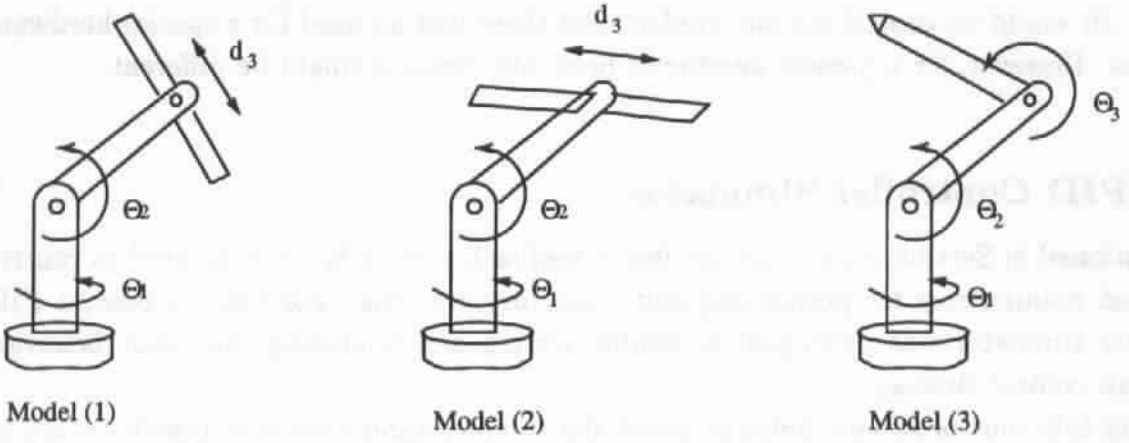


Figure 19: Three different configurations of the robot manipulator.

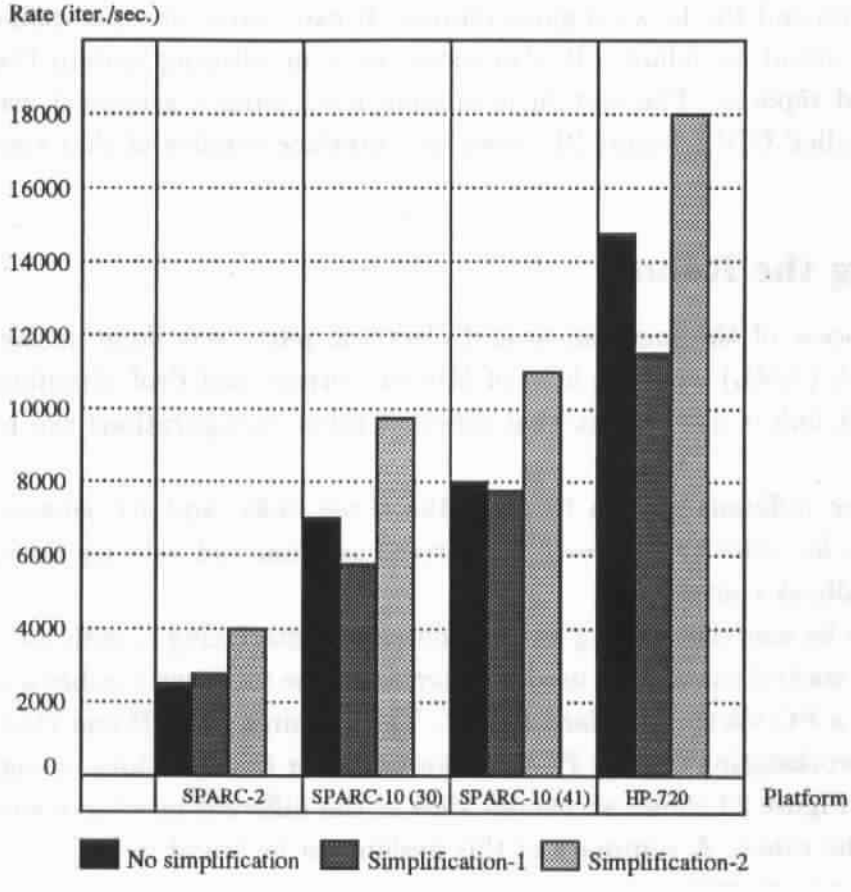


Figure 20: Performance comparison for different platforms.

These benchmarks helped us decide that a software solution on a machine like the Sun SPARC-10 would be enough for our models, and there was no need for a special hardware solutions. However, for a greater number of links, the decision might be different.

4.6 PID Controller Simulator

As mentioned in Section 2.3.1, a simple linear feedback control law can be used to control the robot manipulator for positioning and trajectory tracking. For this purpose, a PID controller simulator was developed to enable testing and analyzing the robot behavior using this control strategy.

Using this control scheme helps us avoid the complex (and almost impossible) task of determining the robot parameters for our three-link prototype robot. One of the most complicated parameters is the inertia tensor matrix for each link, especially when the links are nonuniform and have complicated shapes.

This simulator has a user friendly interface that enables the user to change any of the feedback coefficients and the forward gains on-line. It can also read a pre-defined position trajectory for the robot to follow. It also serves as a monitoring system that provides several graphs and reports. The system is implemented using a graphical user interface development kit called GDI.[2] Figure 21 shows the interface window of that simulator.

4.7 Building the Robot

The assembly process of the mechanical and electrical parts was done in the Advanced Manufacturing Lab (AML) with the help of Mircea Cormos and Prof. Stanford Meek. In this design the last link is movable, so that different robot configurations can be used (see Figure 22).

There are three different motors to drive the three links, and six sensors (three for position and three for velocity), to read the current position and velocity for each link to be used in the feedback control loop.

This robot can be controlled using analog control by interfacing it with an analog PID controller. Digital control can also be used by interfacing the robot with either a workstation (Sun, HP, etc.) or a PC via the standard RS232. This requires an A/D and D/A chip to be connected to the workstation (or the PC) and an amplifier that provides enough power to drive the motors. Figure 23 shows an overall view of the different interfaces and platforms that can control the robot. A summary of this design can be found in [10].

[2]GDI was developed in the department of Computer Science, University of Utah, under supervision of Prof. Beat Brüderlin.

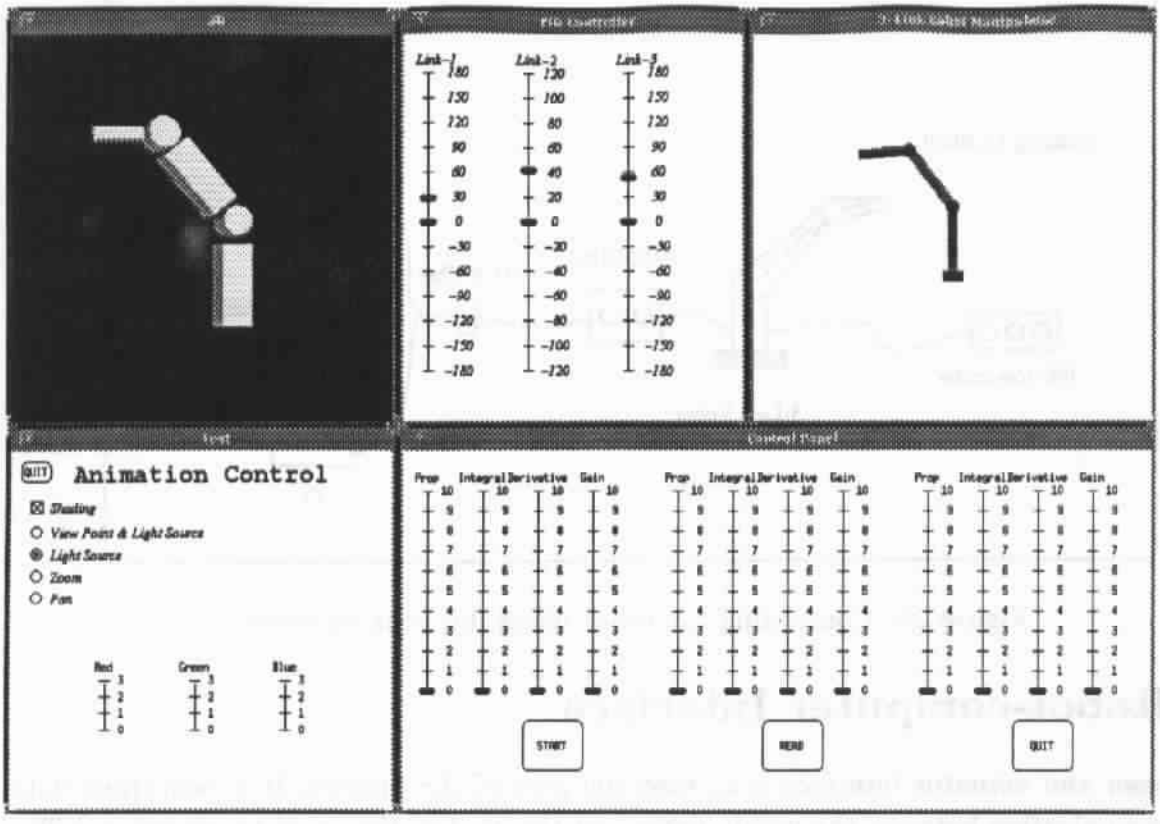


Figure 21: The interface window for the PID controller simulator.

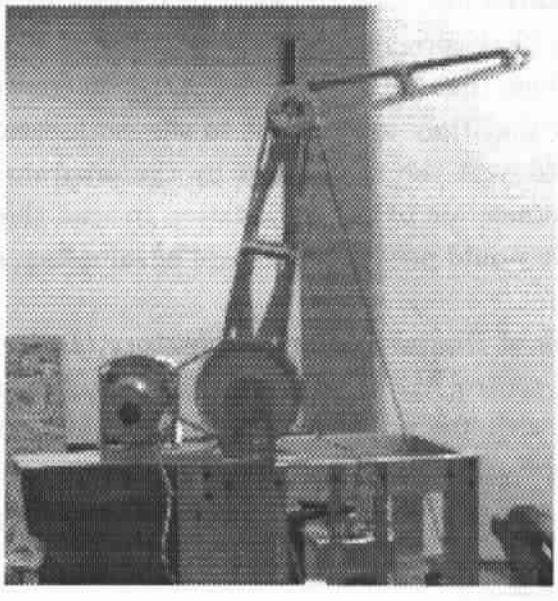


Figure 22: The physical three-link robot manipulator.

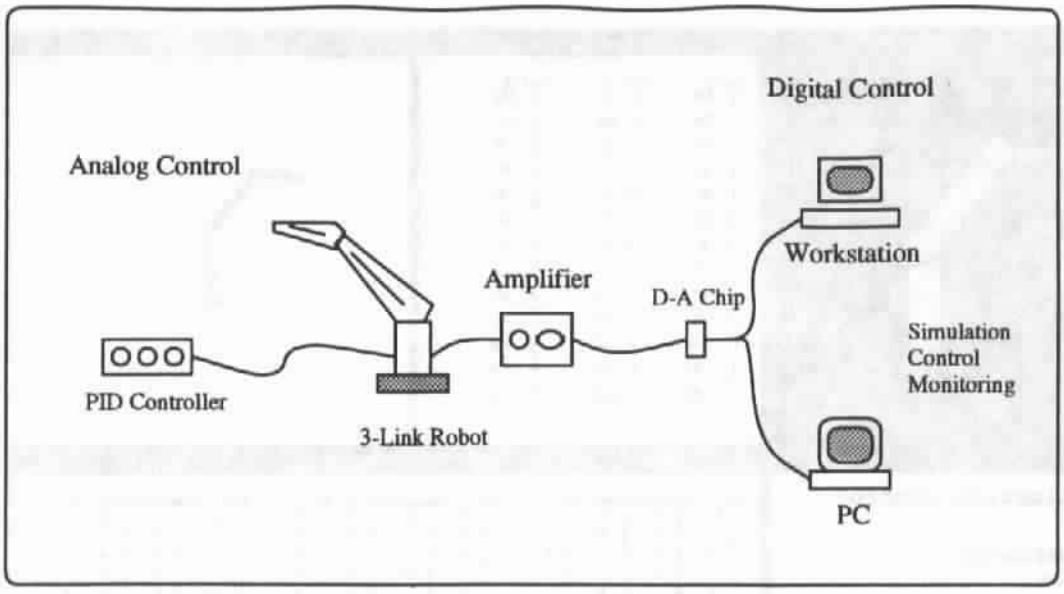


Figure 23: Controlling the robot using different schemes.

5 Robot-computer Interface

The sensor and actuator interface is an essential part of the project. It is concerned with the communication between the manipulator and the workstation used to control it.[3] The problem required interfacing a SUN workstation with 3 Motors, each of which drives a link of the 3 link robot arm. A resident program on the SUN can send out voltage values that will drive the motors in a desired direction (forward or backward), and read values from sensors placed on the motors that correspond to the position of the motors. So thinking at a higher level, it was obvious that we would need A/Ds to convert the values coming from the motors to digital so that they can be sent to the workstation (where the control program resides), D/As to convert the values sent by the program to the actual analog voltage and an RS-232 communication to the workstation to send these digital data to and from the workstation. Also we would need some control of sampling, sending and receiving data outside the workstation.

So based on the suggestion of Digital Systems Laboratory (DSL), we decided to go for a Microcontroller which can control the A/D conversion , D/A conversion and the RS-232 communication protocol with the workstation. The one that came in handy was the MC68HC11EVBU - Universal Evaluation Board, which has a microcontroller, an 8-channel A/D, an RS-232 compatible terminal I/O port, and some wire wrap area for additional circuitry like the D/A unit.

[3]This part has been done by Anil Sabbavarapu, a graduate student in the Computer Science Department.

5.1 The MC68HC11EVBU Chip

The MC68HC11 MCU device is an advanced single-chip MCU with on-chip memory and peripheral functions. The EVBU comes with a monitor/debugging program called BUFFALO (Bit User Fast Friendly Aid to Logical Operations), which is contained in the MCU ROM. User code can be assembled using the line assembler in the BUFFALO monitor program, or else by assembling code on a host computer, and then downloading the code to the EVBU user RAM via Motorola S-records. In the later case the monitor program can be used to debug the assembled user code. There are a lot of utility subroutines in the BUFFALO program that can be used for any program of our own. The MCU that is being used here is MC68HC11E9FN1.

Evaluation and debugging control of the EVBU is provided by the monitor program via terminal interaction. RS-232C terminal I/O port interface circuitry provides communication and data transfer operations between the EVBU and external terminal/host computer devices. A fixed 9600 baud rate is provided for the terminal I/O port.

The EVBU requires a user-supplied +5 Vdc power supply and an RS-232C compatible terminal for operation. A host computer is used with the EVBU to download Motorola S-record information.

The MC68HC11 MCU SCI(serial communication interface) has been set for 9600 baud using a 2MHz E clock. This baud rate can be changed by software by reprogramming the BAUD register in the MCU. The EVBU is wired as data communication equipment(DCE), whereas a dumb terminal (which I used to check the communication protocol before going to the workstation), and most serial modem ports on host computers are wired as data terminal equipment(DTE). This requires a straight-through cable to be used. The monitor program uses the MCU internal RAM located at $0048-$00FF. The control registers are located at $1000-$103F. The EVBU allows the user to use all the features of the BUFFALO software, but then the user can only use locations $0000-$0047 and $0100-$01FF of RAM (325 bytes) and 512 bytes of EEPROM($B600-$B7FF). The total memory map is shown below.

ADDRESS	RESTRICTIONS
$0000-$01FF	512-Bytes of RAM(can be remapped to any 4K page by the IN Register).
$0000-$0047	Available to user.
$0048-$0065	BUFFALO monitor stack area.

```
$0066-$00C3          BUFFALO variables.

$00C4-$00FF          Interrupt pseudo vectors(jumps).

$0100-$01FF          User available.

$1000-$103F          MCU control registers.

$4000                Some versions of EVBs have a D  flip-flop addressed at th
                     location. During initialization, BUFFALO writes $00 to th
                     location to retain compatibility.

$9800-$9801          BUFFALO supports serial I/O to a terminal via a ACIA
     (external IC) located at $9800 in the memory map.

$B600-$B7FF          512-Bytes of EEPROM.

$D000-$FFFF          12K ROM.

$D000-$D00F          BUFFALO supports serial I/O to a terminal and/or host via
                     a DUART(external IC) located at $D0000 in the memory map.

$FFC0-$FFFF          Normal Interrupt Vectors.
```

==========

There are some Jumpers on the evaluation board which decide as to which program to run when reset. In one configuration (monitor mode) a program called PCBUG11 can be run from a PC to control, load and verify programs into various memory locations. Programming and erasing the EEPROM is also done using this software. If we change the configuration of the jumpers, we can run the BUFFALO program. But if the Jumper J2 which connects the pin E0 to logic 0 or logic 1, is changed to logic 1, then the program jumps to location $B600 on reset. This is very useful because this is the starting location of EEPROM. So once you program the EEPROM, then the required program can be run on reset. I have the communication control program start at this location, and each time I reset the board, we enter this program, and it remains in an infinite loop. There are a lot of other jumpers which change the modes of operation but which are beyond our requirements (like expanded-multiplexed, special-bootstrap, or special-test modes) to write about them

in this report.

5.1.1 Operating Instructions

The operating procedures consist of assembly-disassembly and downloading descriptions and examples. The EVBU contains a user reset switch S1. This switch is a momentary action pushbutton switch that resets the EVBU and MCU circuits.

Upon reset, the monitor detects the state of the PE0 line (governed by the position of jumper J2). If a low state is detected, the monitor program is executed and the prompt displayed. If a high state is detected, the monitor will automatically jump directly to EEPROM (address location $B600) and execute user program code without displaying the monitor prompt.

As mentioned earlier, user code can be assembled in two ways, one by using the line assembler in the BUFFALO monitor program, and the other by downloading the assembled code from a host computer in Motorolla S-record format. A download to EEPROM will work if the baud rate is slow enough to allow EEPROM programming. Since erasure and programming both require 10 milliseconds, a slow baud rate (300 baud) will have to be used to ensure enough time between characters. If the EEPROM is bulk erased prior to downloading, 600 baud allows enough time between characters.

The Baud rate can be changed by writing an appropriate number into the Baud register in the MCU, and also by using one of the menu options on the communication program (Kermit or Procomm) on the PC. We will discuss about the Baud rates when we come to the communication part of the project.

A standard input routine controls the EVBU operation while the user types a command line. Command processing begins only after the command line has been terminated by depressing the keyboard carriage return key. The command line format is as follows:

```
> <command> [<parameters>] (RETURN)
where
> EVBU monitor prompt.
        <command> Command mnemonic(single letter for most commands).
<parameters> Expression or address.
(RETURN) RETURN keyboard key - depressed to enter command.

The command line format is defined using special characters which have the
following syntactical meanings:

<> Enclose syntactical variable
[] Enclose optional fields
```

All input numbers are interpreted as hexadecimal. All input commands are converted automatically to upper case lettering except for downloading commands sent to the host computer, or when operating in the transparent mode. A maximum of 35 characters may be entered on a command line. Command line errors can be corrected by backspacing or by aborting the command(CTRL-X or DELETE). Pressing (RETURN) will repeat the most recent command. The LOAD command is an exception. The list of commands with their exact syntax can be looked up from the EVBU User's Manual.

5.2 The MC68HC11E9 Chip

The MC68HC11E9 high-density complementary semiconductor(HCMOS) high- performance microcontroller unit(MCU) includes the following features: 12 Kbytes of ROM, 512 bytes of EEPROM, and 512 bytes of RAM. The MC68HC11E9 is a high-speed, low-power chip with a multiplexed bus capable of running at up to 3 MHz. Its fully static design allows it to operate at frequencies down to dc.

Some of the Features are:

- M68HC11 CPU.
- Power Saving STOP and WAIT Modes.
- 12 Kbytes of On-Chip ROM.
- 512 Bytes of On-Chip EEPROM with Block Protect for Extra Security.
- 512 Bytes of On-Chip RAM.
- 8-Bit Pulse Accumulator.
- Real-Time Interrupt Circuit.
- Synchronous Serial Peripheral Interface(SPI).
- Asynchronous Nonreturn to Zero (NRZ) Serial Communications Interface(SCI).
- 8-Channel 8-Bit Analog-to-Digital (A/D) Converter.
- 38 General-Purpose Input/Output (I/O) Pins.
- 16 Bidirectional I/O Pins.
- 11 Input-Only Pins, 11 Output-Only Pins.

The structure of the MC68HC11E9 MCU is shown in Figure 1.

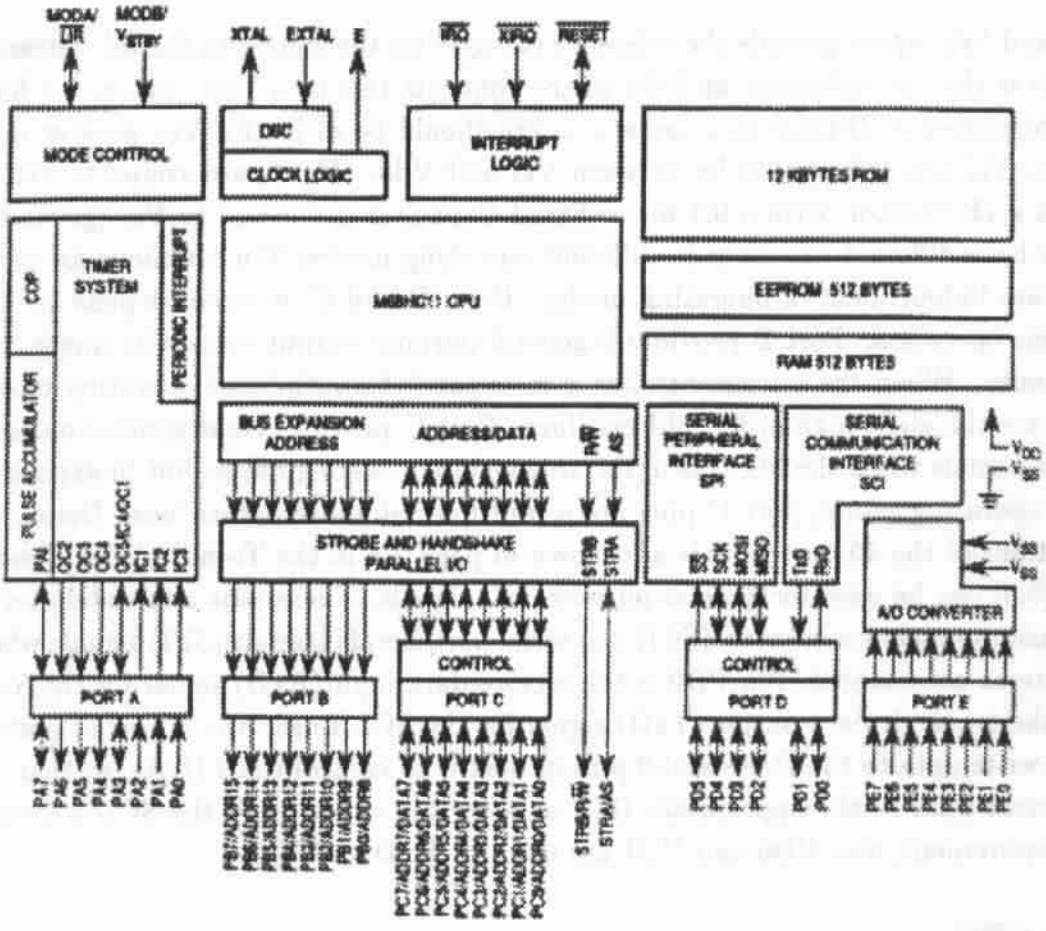


Figure 24: The MC68HC11E9 block diagram.

5.2.1 PIN CONFIGURATION

Some important pins are Vdd and Vss, Reset, XTAL and EXTAL, E-clock Output,Vrl and Vrh, and Port Signals. The XTAL and EXTAL provide the interface for either a crystal or a CMOS compatible clock to control the internal clock generator circuitry. The frequency applied to these pins is four times higher than the desired E-clock rate. The board uses a 8-MHz crystal and the E (system clock) is 2-MHz. The XTAL pin is normally left unterminated when an external CMOS compatible clock input is connected to the EXTAL pin.

The Vrl and Vrh inputs provide the reference voltages for the analog-to digital converter circuitry. Vrl is the low reference, and the board connects this to 0 Vdc. Vrh is the high reference. For proper A/D converter operation, Vrh should be at least 3 Vdc greater than Vrl, and both Vrl and Vrh should be between Vss and Vdd. The board connects Vrh to Vdd through a 1K resistor, with a 0.1 micro Farad Capacitor connected to Vss (ground).

Port pins have different functions in different operating modes. Pin functions for ports A,D, and E are independent of operating modes. Ports B and C, however depend on the mode for their operation. Port B provides 8 general-purpose output signals in single-chip operating modes. When the microcontroller is in expanded multiplexed operating mode, port B pins are the eight high-order address lines. Port C provides eight general-purpose input/output signals when the MCU is in the single-chip operating mode, but in expanded multiplexed operating mode, port C pins are a multiplexed address/data bus. Details of various functions of the 40 port signals are shown in page 2-8 in the Technical Data book.

Pins PD[5:0] can be used for general-purpose I/O signals. These pins alternately serve as the serial communication interface(SCI) and serial peripheral interface(SPI) signals when those subsystems are enabled. Pin PD0 is the receive data input(RxD) signal for the SCI. Pin PD1 is the transmit data output(TxD) signal for the SCI. I used the 8 pins of port B as output to write data to the D/As and 3 pins of port C to control the 3 D/As, so that we write the correct data to the appropriate D/A at a certain time. Since the SCI is always on for our requirement, pins PD0 and PD1 are used for RxD and TxD.

5.2.2 The CPU

The CPU is designed to treat all peripheral, I/O, and memory locations identically as addresses in the 64 Kbyte memory map. This is referred to as memory- mapped I/O. There are no special instructions for I/O that are separate from those used for memory. This architecture also allows accessing an operand from an external memory location with no execution-time penalty.

The CPU registers are an integral part of the CPU and are not addressed as memory locations. The seven registers are shown in Figure 2.

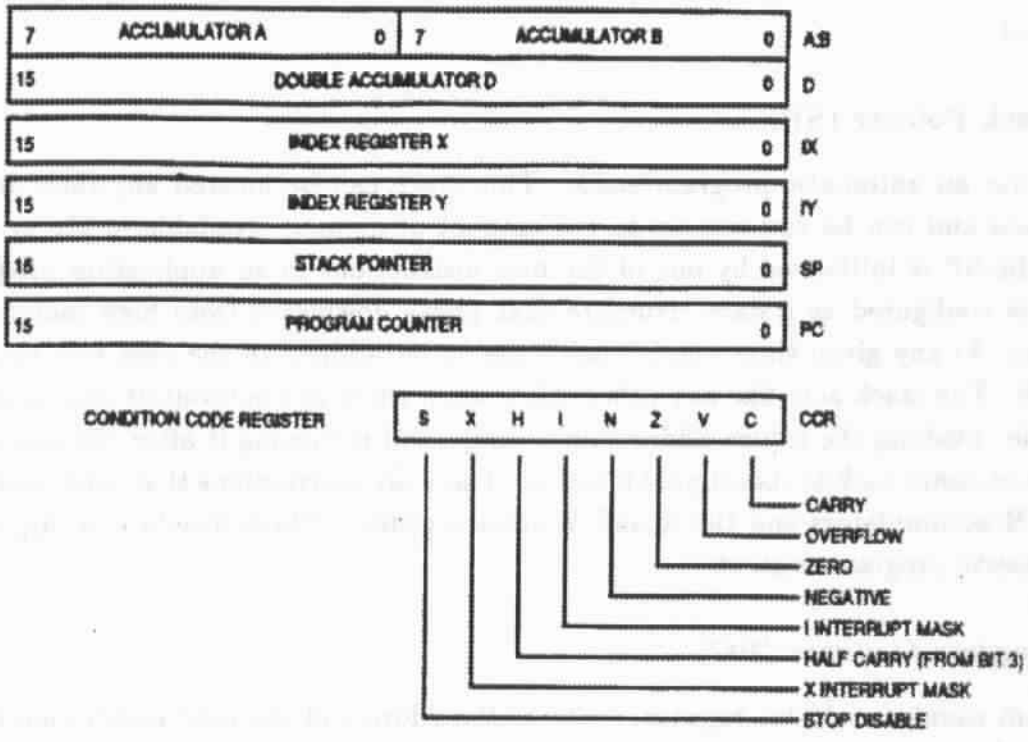


Figure 25: Programming model.

5.2.3 Accumulators A, B and D

These are general purpose 8-bit registers that hold operands and results of arithmetic calculations or data manipulations. For some instructions, these two accumulators are treated as a single double-byte(16-bit) accumulator called D.

5.2.4 Index Registers X and Y

These registers provide a 16-Bit indexing value that can be added to the 8-bit offset provided in an instruction to create an effective address. The IY register needs an extra byte of machine code and an extra cycle of execution time because of the way the opcode map is implemented.

5.2.5 Stack Pointer (SP)

The CPU has an automatic program stack. This stack can be located anywhere in the address space and can be any size up to the amount of memory available in the system. Normally the SP is initialized by one of the first instructions in an application program. The stack is configured as a data structure that grows downward from high memory to low memory. At any given time, the SP holds the 16-bit address of the next free location in the stack. The stack acts like any other stack when there is a subroutine call or on an interrupt. ie. pushing the return address on a jump, and retrieving it after the operation is complete to come back to its original location. There are instructions that push and pull the A and B accumulators and the X and Y index registers. These instructions are often used to preserve program context.

5.2.6 Program Counter (PC)

The program counter, a 16-bit register, contains the address of the next instruction to be executed. After reset, the program counter is initialized from one of six possible vectors, depending on operating mode and the cause of reset.

5.2.7 Condition Code Register (CCR)

This 8-bit register contains five condition code indicators (C,V,Z,N and H), two interrupt masking bits, (IRQ and XIRQ) and a stop disable bit (S). These condition codes are automatically updated by most instructions.

5.2.8 Opcodes and Addressing Modes

This MCU uses 8-bit opcodes. A complete instruction consists of a prebyte, if any, an opcode, and zero, one, two, or three operands. Complete instructions can therefore be

from one to five bytes long.

Six addressing modes can be used to access memory. They are : immediate, direct, extended, indexed, inherent, and relative. All modes except inherent mode use an effective address. The effective address is the memory address from which the argument is fetched or stored, or the address from which execution is to proceed. The effective address can be specified within an instruction, or it can be calculated. The detailed instruction set is available in the Technical Data Book for MC68HC11E9.

5.2.9 Memory Map

The operating mode determines memory mapping and whether memory is addressed on- or off-chip. Memory locations for on-chip resources are the same for both expanded multiplexed and single-chip modes. Control bits in the CONFIG register allow EPROM and EEPROM to be disabled from the memory map. The 512-byte RAM is mapped to $0000 after reset. It can be placed at any other 4K boundary ($x000) by writing an appropriate value to the INIT register. The 64-byte register block is mapped to $1000 after reset and can also be placed at any 4K boundary($x000) by writing an appropriate value to the INIT register. The Table for MCU register and control bit assignments can be looked up from the Technical Data Manual.

5.3 Serial Communications Interface (SCI)

The SCI is a universal asynchronous receiver transmitter (UART), one of two independent serial I/O subsystems in the MC68HC11E9. It has a standard non return to zero(NRZ) format (one start, eight or nine data, and one stop bit). Several baud rates are available. The SCI transmitter and receiver are independent, but use the same data format and bit rate.

The serial data format requires the following conditions:

- An idle-line in high state before transmission or reception.
- A start bit, logic zero, transmitted or received, that indicates the start of each character.
- Data that is transmitted and received least significant bit(LSB) first.
- A stop bit, logic one, used to indicate the end of a frame. (A frame consists of a start bit, a character of eight or nine data bits, and a stop bit.)
- A break (defined as the transmission or reception of a logic zero for some multiple number of frames).

Selection of the word length is controlled by the M bit of SCI control register SCCR1.

5.3.1 Transmit Operation

The SCI transmitter includes a parallel data register(SCDR) and a serial shift register, which can only be written through the SCDR. This double buffered operation allows a character to be shifted out serially while another character is waiting in the SCDR to be transferred into the serial shift register. The output of the serial shift register is applied to TxD as long as transmission is enabled.

5.3.2 Receive Operation

During receive operations, the transmit sequence is reversed. The serial shift register receives data and transfers it to a parallel receive data register (SCDR) as a complete word. An advanced data recovery scheme distinguishes valid data from noise in the serial data stream. The data input is selectively sampled to detect receive data, and a majority voting circuit determines the value and integrity of each bit.

Two methods of wakeup are available: idle-line wakeup and address-mark wakeup. During idle-line wakeup, a sleeping receiver awakens as soon as the RxD line becomes idle. In the address-mark wakeup, logic one in the most significant bit (MSB) of a character wakes up all sleeping receivers. Since we are using only one chip, we use the idle-line wakeup.

Three error conditions occur during generation of SCI system interrupts. They are the SCDR overrun, received bit noise, and framing errors. Three bits (OR, NF, and FE) in the SCSR register are set to indicate that the respective error has occured. A read of the SCSR (with the respective bit set) followed by a read of the SCDR clears the bit, and ensures normal operation.

5.3.3 SCI Registers

There are five addressable registers in the SCI:

1. Serial Communications Data Register: SCDR is a parallel register that receives data when it is read, and transmits data when it is written. Reads access the receive data buffer and writes access the transmit data buffer. Receive and transmit are double buffered.

2. Serial Communications Control Register 1: SCSR1 provides the control bits that determine word length and select the method used for the wakeup feature.

3. Serial Communications Control Register 2: SCSR2 provides the control bits that enable or disable individual SCI functions like transmit interrupt enable, receiver interrupt enable, idle-line interrupt enable, transmitter enable and receiver enable among others.

4. Serial Communication Status Register: SCSR provides inputs to the interrupt logic circuits for generation of the SCI system interrupt. It contains all the error detection flags, besides the transmit complete and receive data register full flags.

5. Baud Rate Register: This register is used to select different baud rates for the SCI system. The SCP[1:0] bits function as a prescaler for the SCR[2:0] bits. Together these five bits provide multiple baud rate combinations for a given crystal frequency. Right now the EVBU has an 8MHz crystal, and we are using a 9600 Baud rate.

For our communication, I used a standard available subroutine called INIT to initialize the SCI, and then wrote two subroutines to read and write data, which were very similar to the standard ones, except that they do not modify the data, read carriage return or line feed as they are and also use all 8 bits instead of masking off the 8th bit for parity.

5.4 Analog to Digital Converter

The S/D system is an 8-channel. 8-bit, multiplexed-input converter. It does not require external sample and hold circuits because of the type of charge redistribution technique used. A/D converter timing can be synchronized to the system E clock, or to an internal RC oscillator. The A/D converter system consists of four functional blocks: multiplexer, analog converter, digital control, and result storage.

5.4.1 Multiplexer:

The multiplexer selects one of 16 inputs for conversion. Input selection is controlled by the value of bits CD-CA in the ADCTL register. The eight port E pins are fixed direction analog inputs to the multiplexer, and additional internal analog signal lines are routed to it.

5.4.2 Analog Converter:

Conversion of an analog input selected by the multiplexer occurs in this block. It contains a digital-to-analog capacitor (DAC) array, a comparator and a successive approximation register. Each conversion is a sequence of eight comparisons operations, beginning with the MSB. Each comparison determines the value of a bit in the successive approximation register(SAR). The DAC array performs two functions. It acts as a sample and hold circuit during the entire conversion sequence, and provides comparison voltage to the comparator during each successive comparison.

The result of each successive comparison is stored in the SAR, and when the conversion sequence is complete, the contents of the SAR are transferred to the appropriate result register.

5.4.3 Digital Control:

All A/D converter operations are controlled by bits in register ADCTL. In addition to selecting the analog input to be converted, ADCTL bits indicate conversion status, and control whether single or continuous conversions are performed. Also the ADCTL bits determine whether conversions are performed on single or multiple channels.

5.4.4 Result Registers:

Four 8-bit registers (ADR[4:1]) store conversion results. The conversion complete flag (CCF) indicates when valid data is present in the result registers. The result registers are written during a portion of the system clock cycle when reads do not occur, so there is no conflict.

A/D converter operations are performed in sequences of four conversions each. A conversion sequence can repeat continuously or stop after one iteration. The conversion complete flag (CCF) is set after the fourth conversion in a sequence to show the availability of data in the result registers.

We used multiple channel conversion, continuous scan and the lower four channels since we need 3 sensor values converted in one shot. Since the Vrl was 0V and Vrh was 5V, a 0V input got converted to 0x00 and a 5V input got converted to 0xFF. To change the range of the analog input, we have to scale the input and consequently interpret the digital data accordingly. But since the sensors that we have on the robot arm have a range from 0V to 5V, we can directly connect their output to the A/D input.

5.5 Digital to Analog Converter

For the D/A conversion, we used an 8-Bit microprocessor compatible, double buffered DAC0830. The DAC0830 is an advanced CMOS 8-bit multiplying DAC designed to interface directly with most of the popular microprocessors. The circuit uses CMOS current switches and control logic to achieve low power consumption and low output leakage current errors. Double buffering allows these DACs to output a voltage corresponding to one digital word while holding the next digital word. The DAC can be used in different modes of operation. We used it in a Voltage Switching Configuration with bipolar output with increased output voltage swing which ranges from -10V to +10V (ie. 0x00 would correspond to -10V and 0xFF to +10V)

There are two important things to keep in mind when using this DAC in the voltage switching mode. The applied reference voltage must be positive since there are internal parasitic diodes from ground to the Iout1 and Iout2 terminals which would turn on if the applied reference went negative. There is also a dependence of conversion linearity and gain error on the voltage difference between Vcc and the voltage applied to the normal

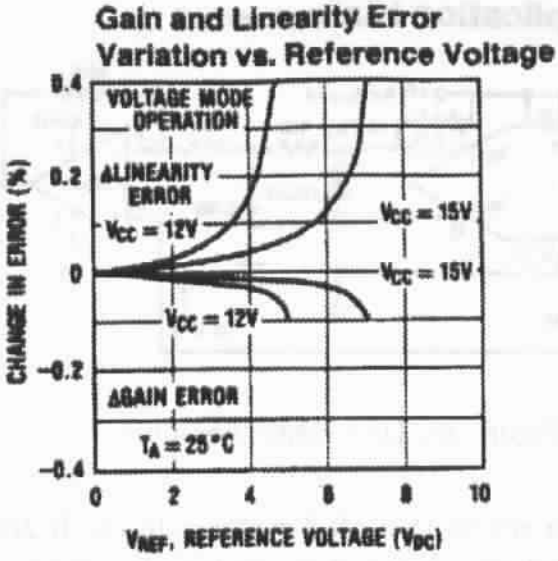


Figure 27: Error vs reference voltage.

5.7 Workstation's Role

The workstation has the control program running on it. For the communication protocol, the program opens the port /dev/ttya using the system call "open" and then controls the configuration of the port by using the call "ioctl" which can set the baud rate, type of transmission and reception, length of each character etc. The port was configured to operate at 9600 baud rate, non-echoing, raw mode where all the 8bits are received and transmitted as they are without checking or masking the 8th bit for parity. Also there is absolutely no processing done on the characters, so that the carriage return and line feed are read and sent as different characters, and the lower and upper case letters are mapped differently. Also the port was configured to be non-blocking using another system call "fcntl", which makes the read operations easier. So once the port has been opened in the correct format, all the program has to do is read or write to this port, using commands "read" and "write".

So the program gets the sensor values from the chip by reading from the port, and then scales them appropriately from Hex to Integers, uses them in the program to calculate the voltage to be sent out, and then again scales them to be between 0x00 and 0xFF (so that 0x00 corresponds to -10V and 0xFF to +10V), before writing to the port so that the chip can read it. Once the whole program is completed , the port is closed to prevent further communication, and allow access to other programs.

5.8 Problems

There were a series of problems all along the project. First the port /dev/ttya was not configured, and later it was configured to be a terminal. A terminal is configured by default to be echoing, carriage return line feed mapped together and a cooked mode ie. lines of input are collected and input editing is done, and the edited line is available when it is completed by a newline. Without knowing this default mode, I had the protocol set up such that the chip sends some data and waits for the workstation to send back some data. But instead the chip would read the data it had just sent which got echoed back from the port. Also the workstation wouldn't read any data since it did not receive a carriage return. Later with a carriage return- line feed inserted with each data, the workstation was reading the data OK, but the input and output data was being processed for parity and the 8th bit was masked, so the range of data in the communication was now reduced to 0x00 to 0x7F. The in step protocol had no meaning at all and the communication would just lock up. After I had the configuration set up properly, the communication worked correctly.

Also a different cable had to be used for communication with the workstation. I had the communication tested with a PC, but then it wouldn't work on the workstation, because it needed some extra jumpers on the connector. DSL helped me out with the right cables. Testing with the dumb terminal t10 caused some problems because it also did parity checking and masking and I was checking to see if output processing was being done by the workstation or not.

Though the communication on the sun works perfect, the read operations from the sensors through the chip take up a lot of time. After running a lot of tests on the communication, like reading and writing different number of bytes from and to the chip, we figured out that most of the time was being used up in the OS during the read operations. Though we couldn't find out what exactly was going on in the kernel that was causing the delay (we couldn't find any documentation on it), we guessed that there was a read buffer that was being filled up each time a byte is sent by the chip, but the buffer is not available in user space for the program to read it until the buffer contains a certain number of bytes, or a certain time-out value has expired. But we had no way of either proving it or changing it.

So we moved from hayduke (Sparc 10) to kahlua (HP - 730), because we were assured of some low level hacking by Mike Hibler of CSS, to see what the problem was, and correct it, if it persists. On running the program a few times on the HP, Mike figured out that it was a buffer timeout value problem on an HP, and found that the default value was set to 14 characters. So he changed the timeout to one character, and rebooted the machine. He also mentioned that usually this timeout value cannot be changed by a normal user. So once this change was done, the 10,000 loop program with 2 reads and 1 write per loop ran in 30 seconds, which is around the optimum speed that can be attained at 9600 baud rate.

To get a higher update rate , we have to use 19200 or 38400 baud rate.

But the EVBU uses a crystal of 8-MHz frequency. With this frequency, the maximum attainable useful baud rate is 9600. To get 19.2K or 38.4K, we will have to replace the crystal by a different crystal. But with a 10-MHz crystal, the system E-clock rate increases to 2.5-MHz which might cause problems since the part has been tested only for 2.0MHz. Another option is to go to a lower frequency crystal (4.9152 MHz) which also gives the standard baud rate options, but this slows down the internal clock and hence execution of the instructions, and the A/D conversions. Also the PCBUG11 and the BUFFALO monitor programs have been written for 8MHz crystal frequency, and they have to be modified for any other crystal. But after Mohammed simplified the control program to normal linear control, the communication speed on the SUN was sufficient for control of the 3 links.

5.9 Motors and Sensors

Mircea Cormos took care of the motors and sensors that would be used on the robot arm. There are 3 different motors, one for the base, one for the shoulder, and one for the elbow. The Base motor is a Clifton Precision Products Servo Motor with Double Shaft. It is rated at 24VDC, 2200 RPM at no load and a peak torque of 350 oz-in. The current at peak torque is rated at 24 Amps. The Shoulder motor is a PM motor with optical incremental encoder. It is rated at 18 VDC, 3000 RPM at no load and a peak current of 51 Amps. The Elbow motor is a Canon geared DC motor with ratings of 24 VDC and 86 RPM at no load.

One of the sensors is a tachometer. It has an output of 6.5 Volts/1000 RPM. The A/D on the chip has a limit of 5 VDC. But the motor is never expected to reach a very high RPM and hence the input to the A/D would be well within the limit. The other sensors are optical encoders, which have an input rating of 5VDC and 60 mA, and the output varies within this range, which is also perfect for our purpose.

The D/A's give out an output swing of +/-10V, and the current rating is only 2mA. So the motors need power amplifiers that will transfer this voltage to the motors and withstand the current requirements of the motors. But right now, the setup has 3 PIDs which get the output voltage from the D/As and amplifies them before feeding them to the motors.

6 The Optimal Design Subsystem

The role of this subsystem is to assist robot designers in determining the optimal configuration and parameters given some task specifications and some of the parameters. The following sections describe the required tasks to be done to accomplish this part along with some design examples.

6.1 Constructing the Optimization Problem

Any optimization problem has three main components:

- Objective function to be minimized or maximized.
- Optimization variables.
- Set of constraints.

A set of objective functions that can be used in the optimization problem are specified. This set will form the database for the formation of the final objective functions for some of the parameters using the task specification and the performance requirements.

Some of the criteria that can be used to form objective functions are:

- Work space.
- Manipulability.
- Speed.
- Accuracy.
- Power consumption of motors.

To form the objective functions, we need to find quantitative measures for the manipulator specification and the performance requirements. In some cases, a closed form expression is not available. In such cases, the simulation programs can be used to determine the required quantitative measure. For example, the maximum velocity is a function of most of the parameters (link lengths, masses, friction, motor parameters), but it is not easy to get a closed form expression for the velocity as a function of all of these parameters; therefore, the simulation program can be used to measure the maximum velocity for different values of these parameters.

In addition to these quantitative measures, there are some rules and assumptions that will be used to solve for some of the parameters, and to give guidance during the design cycle. Some of the assumptions we made to simplify the problem are:

- The robot type and the degrees of freedom are given.
- Only revolute and prismatic joints are considered.
- The links are uniform with rectangular or cylindrical cross section.
- There is a finite set of materials used to build the robot with known densities.

- There is a finite number of actuators and sensors with known specifications that can be used in the design.

Some of the rules that can serve as additional constraints are:

- Select the solution with equal link lengths or masses because this will simplify the manufacturing process (i.e., minimize the cost).
- Choose the feedback controls k_p, k_v that give critically damped behavior ($k_v = 2\sqrt{k_p}$).
- Consider a minimum length for each link to satisfy some assembly and manufacturing constraints, such as actuator and sensors sizes.

Our strategy for solving this optimization problem will be to divide it into stages, at each stage solve for some of the parameters, then use the values obtained for these parameters in the following stage. The reason for choosing this strategy is that, some of the robot parameters must be determined before we can start solving for other parameters. For example, the robot type must be determined first. The other parameters are largely affected by the choice of the robot type. The selection of the robot type depends on the tasks and performance requirements. For the time being, we assume that the robot type is given, and later the selection of the robot type can be added to the system.

There are many algorithms for solving the optimization problem. In our case most of the objective functions will have more than one variable. In this case *multidimensional optimization techniques* are recommended. One of the simplest methods is *pattern search* which alternates sequences of local exploratory moves with extrapolations (or pattern moves). Another method is *simple random search* which selects random search points and evaluates the function at each of those points. More details about these methods and other optimization techniques can be found in [16, 54].

The following are some quantitative measures that can be used as objective functions for some of the design parameters with some examples of forming the optimization problem from the robot specification.

6.1.1 Structural Length Index

This measures the efficiency of the design in terms of generated workspace. It is defined as the ratio of the manipulator length sum to the cube root of the workspace volume. The objective is to minimize this value as a function of link lengths.

$$Q_L = \frac{L}{\sqrt[3]{W}}$$

where L is the length sum and is defined as:

$$L = \sum_{i=1}^{n}(a_{i-1} + d_i)$$

where a_i is link length, and d_i is the maximum offset for prismatic joints.

As an example of using this measure, suppose that the given specification for the manipulator is three degrees of freedom with a certain workspace shape (sphere, cylinder, etc.), the maximum total length for its links is L, and the first two links are equal. The optimization problem will be:

$$\min f(l_i)$$

where:

$$\sum l_i = L,$$
$$l_1 = l_2$$

The last two equations constitute the constraints in the optimization problem. This problem is very easy to solve for the lengths. The problem here is to calculate the volume of the workspace, because sometimes it is too difficult to calculate when the workspace is irregular and if it has some gaps (nonreachable areas).

6.1.2 Manipulability

Another measure is the *manipulability*. This measures the ability of the manipulator to move uniformly in all directions. At the singular points, the manipulator loses one or more degrees of freedom. In other words, some tasks cannot be performed at or near singular points. A quantitative measure for the manipulability of a robot is defined as:

$$w = \mid \det(J(\Theta)) \mid$$

where $J(\Theta)$ is the Jacobian matrix which is the first derivative of the position vector of the end-effector. By maximizing this measure for the length of each link, the manipulator will have a maximally large well-conditioned workspace.

6.1.3 Force Transmissibility

Another measure for robot capability is $u = 1/w$, which called *force transmissibility*. If motion capability is the desired behavior then we maximize for w (the manipulability), but if a powerful work capability is the desired characteristic, then we maximize for u. On the other hand, if we want a flexible robot that can handle both situations reasonably efficiently, then we find the average value of u and w.

6.1.4 Accuracy

Frequency of update and sampling rates are the main parameters affecting the accuracy of the manipulator motion. In general, increasing the frequency of update and the sensor readings results in smaller error patterns. There is no formal or closed form solution to determine the optimal value for the frequency of update that gives a specified maximum

allowable error. The only practical way is to use simulation programs and change the value for both frequencies until the desired behavior is obtained. The constraint in this case is the maximum speed of the machine used and the maximum speed for the interface between the robot and the actuators and sensors.

6.2 The User Interface

The user interface is interactive and enables the user to select the performance criteria and specify some of the parameter values (which will form the system constraints); it also displays the results of the recommended values for the other parameters. After selecting values for the required parameters, the new configuration can be tested using the simulation and monitoring subsystems, and each performance criterion can be measured and compared with the prespecified performance requirements, then, changes can be made to the tasks or the constraints if necessary, and the design subsystem run again to get new values. This design cycle is shown in Figure 28.

6.3 Some Design Examples

In this section we will demonstrate the strategy used in the optimal design module by showing some design examples. In each example the performance requirements are stated which form one or more objective functions, a set of constraints are formed from the given specifications, the parameters to be determined for optimal performance are specified, and finally the strategy for solving the problem is explained.

6.3.1 Example (1)

- Performance Criteria:
  - Efficient link lengths.
  - Maximum manipulability.
- Optimization Parameters:
  - Link lengths.
- Optimization Functions:

$$Q_L = \frac{L}{\sqrt[3]{W}}$$

$$w = \mid \det(J(\Theta)) \mid$$

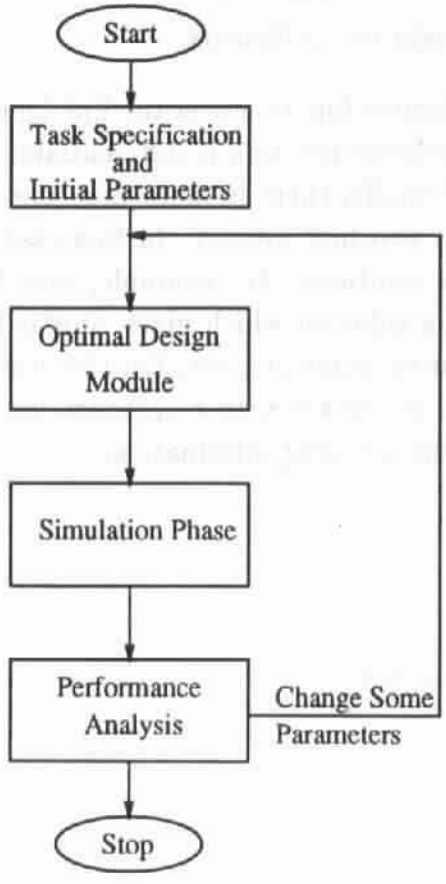


Figure 28: The optimal design cycle.

- Constraints:
  - Total link lengths (L).
- Strategy:

  Combining these two objective functions using weighting coefficients according to the importance of of each one, we get one objective function:

$$c_1 Q_L + c_2 w$$

  where c_1 and c_2 are the weighting coefficients.

  By minimizing the new objective function over the link lengths, we can get values for the link lengths. If the generated function is differentiable, then we can get a direct solution to the problem. Usually, there is more than one solution for selecting the link lengths (for more than two link robots). In this case, some other rules can be used to select among these solutions. For example, each link may have bounds on its length. Also, selecting a solution which gives similar length for several links is better from the manufacturing point of view. On the other hand, if the function is not differentiable, then we can use a heuristic optimization technique such as *pattern search* which does not require gradient information.

6.3.2 Example (2)

- Performance Criteria:
  - Minimum position error (e).
  - Maximum speed ($\dot{x}$).
- Optimization Parameters:
  - The feedback gains k_p, k_v.
  - Joint friction f_r.
- Optimization Functions:

$$e = f(k_p, k_v, f_r)$$
$$\dot{x} = g(k_p, k_v, f_r)$$

  where k_p, k_v, and f_r are vectors of length N, where N is the number of links.

- Constraints:

- Link lengths and masses.
- Maximum torque available.
- Motor parameters.
- Update frequency.
- Feedback frequency.

- Strategy:

  In this case, the functions f and g are not in a closed form since the error and the velocity are calculated iteratively using the dynamic and the feedback control modules. Therefore, the simulation program will be used to determine the optimal values for the required parameters. Also here, we can form one objective function as in the first example. Notice here that we can consider that a critically damped behavior is preferred, so we can use the relation between k_p and k_v that produces this behavior, which is:

$$k_v = 2\sqrt{k_p}$$

  This will reduce the number of optimization variables from three to two.

6.3.3 Example (3)

- Performance Criteria:
  - Maximum acceleration ($\ddot{x}$).
  - Minimum position error (e).
  - Minimize power consumption for the motors (P).

- Optimization Parameters:
  - Link lengths.
  - Link masses.

- Optimization Functions:

$$\ddot{x} = f(l_i, m_i)$$
$$e = g(l_i, m_i)$$
$$P = h(l_i, m_i)$$

  The overall objective function is:

$$a_1 f + a_2 g + a_3 h$$

- Constraints:
  - Feedback gains.
  - Update frequency.
  - Feedback frequency.
  - Friction.
  - Set of available densities for the link material.
  - Catalog of available motors.
- Strategy:

  This problem is solved in two stages: first the manipulability and the structured length index can be used to determine the optimum link lengths (as in the first example); then, we use these lengths to get the optimum masses. From the assumptions stated before, there is a finite set of densities, and the links are uniform, which means we need to select the density that gives optimum performance, since we already have the lengths. This problem can be solved using pattern search on the densities, or using some other integer optimization techniques.

  The power consumption of the motor is related to the torque, which means we need to minimize the maximum torque. Also, here we use the simulation program to get a quantitative measure for the overall objective function.

6.3.4 Example (4)

- Performance Criteria:
  - Maximum speed ($\dot{x}$).
  - Minimum position error (e).
- Optimization Parameters:
  - Update frequency (u).
  - Feedback frequency (sensor reading rate, r).
- Optimization Functions:

$$\dot{x} = f(u, r)$$
$$e = g(u, r)$$

The overall objective function is:

$$a_1 f + a_2 g$$

- Constraints:
  - Link lengths and masses.
  - Feedback gains.
  - Motor parameters.
  - Friction.
  - Sensor ranges.
  - Maximum computer speed.
  - Maximum speed for the communication part (A/D and D/A).
- Strategy:

  We again use simulation to get a quantitative measure for the overall objective function, with the pattern search on the two frequencies given the maximum speed for the computer and the sensor reading rate.

7 The Prototyping Environment

The prototyping environment consists of several subsystems such as:

- Design.
- Simulation.
- Control.
- Monitoring.
- Hardware selection.
- CAD/CAM modeling.
- Part ordering.
- Physical assembly and testing.

Figure 29 shows a schematic view of the prototyping environment with its subsystems and the interface.

These subsystems share many parameters and information. To maintain the integrity and consistency of the whole system, a central interface (CI) is proposed with the required rules and protocols for passing information. This interface will be the layer between the

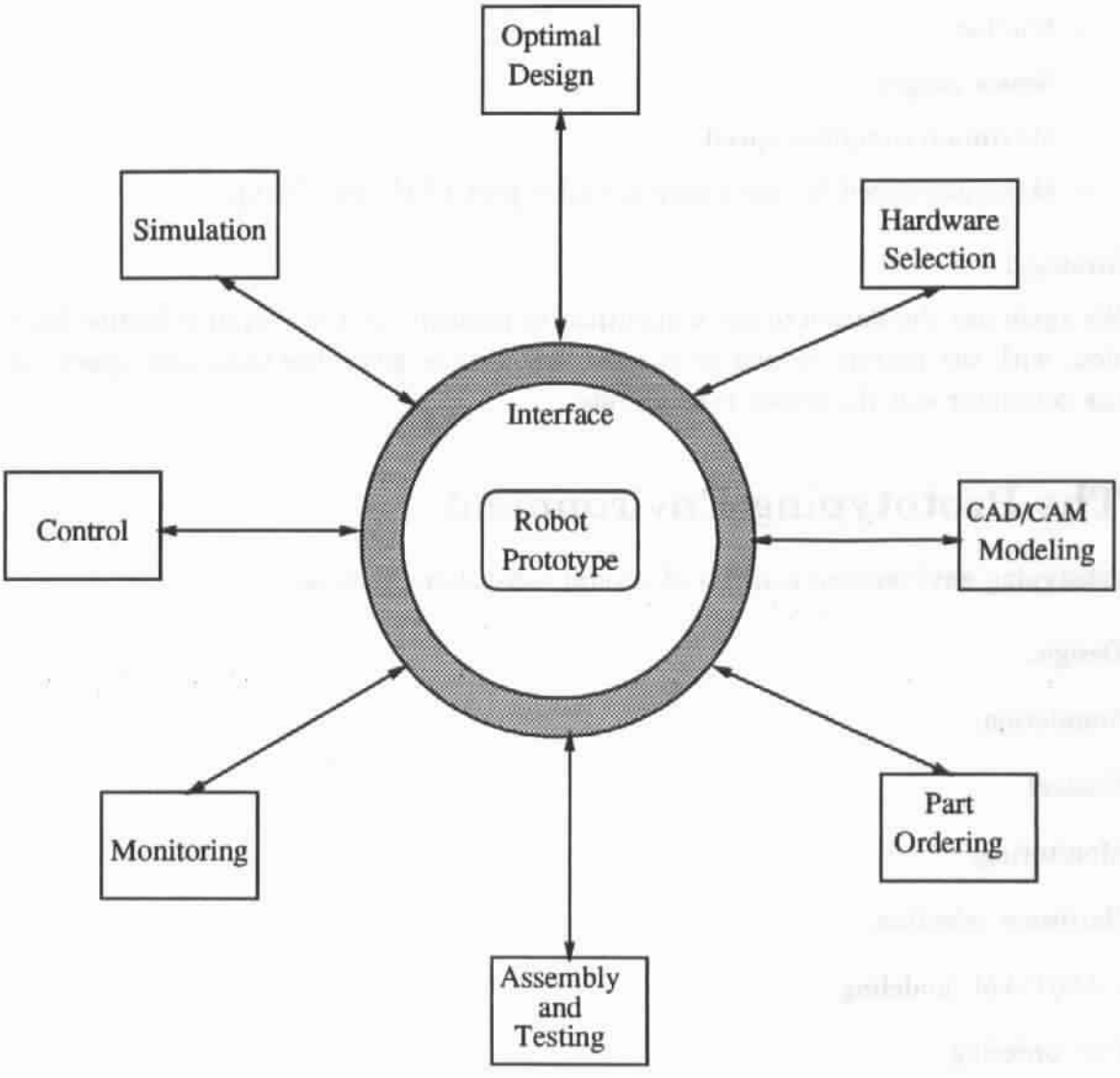


Figure 29: Schematic view for the robot prototyping environment.

robot prototype and the subsystems, and it will also serve as a communication channel between the different subsystems.

The tasks of this interface include:

- Building relations between the parameters of the system, so that changes in any of the parameters will automatically perform a set of modifications to the related parameters on the same system, and to the corresponding parameters in the other subsystems.
- Maintaining a set of rules that governs the design and modeling of the robot.
- Handling the communication between the subsystems using a specified protocol for each system.
- Identifying the data format needed for each subsystem.
- Maintaining comments fields associated with some of the subsystems to keep track of the design reasoning and decisions.

The difficulty of building such an interface arises from the fact that it deals with different systems, each with its own architecture, knowledge base, and reasoning mechanisms. In order to make these systems cooperate to maintain the consistency of the whole system, we have to understand the nature of the reasoning strategy for each subsystem, and the best way of transforming the information to and from each of them.

In this environment the human role should be specified and a decision should be taken about which systems can be fully automated and which should be interactive with the user. The following example illustrates the mechanism of this interface and the way these systems can communicate to maintain system consistency.

Assume that the designer wants to change the length of one of the links and wants to see what the motor parameters should be that give the same performance requirements. The optimal design subsystem is used to determine the new values for the motor parameters given the new length, then it sends a request to the CI to look for the motor with the required specifications in the part-ordering system. Here we have two cases: a motor with the required specifications is found in the catalogs, or no motor is available with this specification. In the second case, this will be reported and another motor with the closest specifications will be selected. Next, the motor specifications will be updated in the database; then the CAD/CAM system is used to generate the new model and to check the feasibility of the new design. For example, the new motor might have a very high rpm, which requires gears with high reduction ratio. This might not be possible in some cases when the link length is relatively small. In this case, this will be reported and the user will be notified of this problem and will be asked to either change some of the parameters

or the performance requirements and the loop will start again. Once the parameters are determined, the monitoring program is used to give a performance analysis and compare the results with the required performance. Finally, a report with the results is produced.

7.1 Interaction Between Subsystems

To be able to specify the protocols and data transformation between the subsystems in the environment, the types of actions and dependencies among these subsystems must be identified. Also, the knowledge representation used in each subsystem should be determined.

The following are the different types of actions that can occur in the environment:

- Apply relations between parameters.
- Check constraints.
- Make decisions. (Usually, the user makes the decisions.)
- Search in tables or catalogs.
- Update data files.
- Deliver reports (text, graphs, tables, etc.).

There are several data representations and sources such as:

- Input from the user.
- Data files.
- Text files (documentation, reports, messages).
- Geometric representations (Alpha_1).
- Mathematical Formulae.
- Graphs.
- Catalogs and tables.
- Rules and constraints.
- Programs written in different languages (C, C++, Lisp, Prolog, etc.).

7.2 The Interface Scheme

There are several schemes that can be used for the interface layer. We will consider a scheme in which each subsystem has a subsystem interface (SSI). The SSI has the following tasks:

- Transfer data to and from the subsystem.
- Send requests from the subsystem to the other interfaces through the central interface.
- Receive requests from other subsystem interfaces and translate them to the local language.

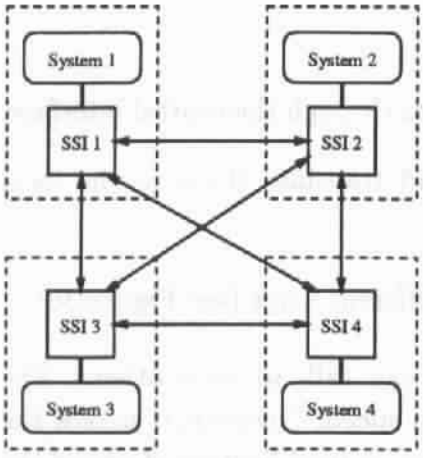
These subsystem interfaces can communicate in three different ways (see Figure 30):

1. Direct connection: which means that all interfaces can talk to each other. The advantage of this is that it has a high communication speed; however, it makes the design of such interfaces more difficult, and the addition or modification of one of the interfaces requires the modification of all other interfaces.
2. Message routing: in which any request or change in the data will generate a message on a common bus, and each SSI is responsible for taking the relevant messages and translating then to its subsystem. The problem with this scheme is that it makes the synchronization of the subsystems very difficult, and the design of the interface is more complicated.
3. Centralized control: in which all interfaces talk with one centralized interface that controls the data and controls flow in the environment. The advantage of this scheme is that it makes it much easier to synchronize between the subsystems, and the addition or modification of any of the SSIs will not affect the other SSIs.

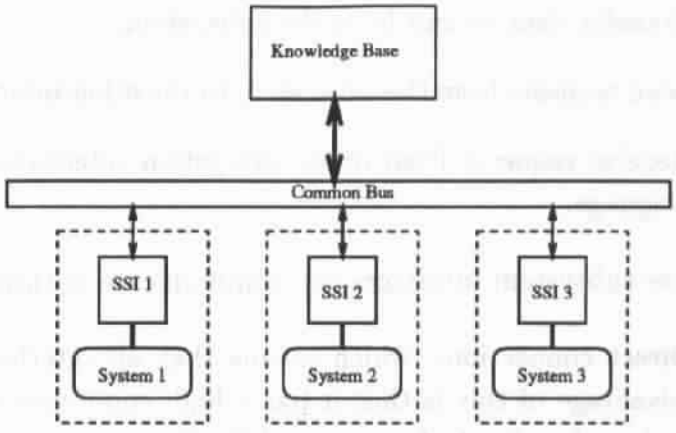
7.3 Overall Design

The Prototyping Environment (PE) consists of a *central interface* (CI) and *subsystem interfaces* (SSI). The tasks of the central interface are to:

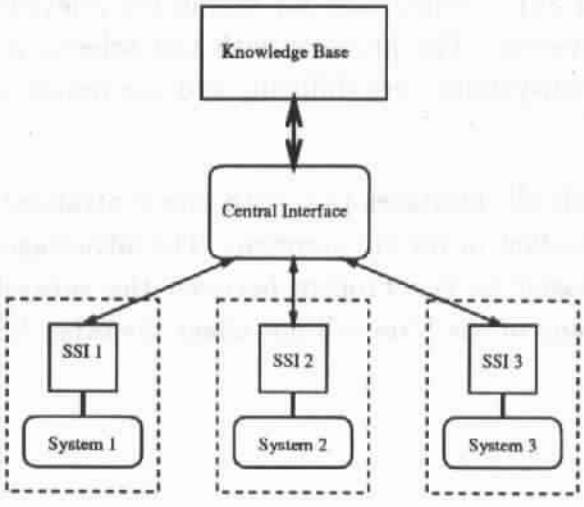
- Maintain a global database of all the information needed for the design process.
- Communicate with the subsystems to update any changes in the system. This requires the central interface to know which subsystems need to know these changes and send messages to these subsystems informing them of the required changes.



(1) Direct Connection



(2) Message Routing



(3) Centralized Control

Figure 30: Three different methods for subsystem interface communication.

- Receive messages and reports from the subsystems when any changes are required, or when any action has been taken (e.g., update complete).
- Transfer data between the subsystems upon request.
- Check constraints and apply some of the update rules.
- Maintain a design history containing the changes and actions that have been taken during each design process with date and time stamps.
- Deliver reports to the user with the current status and any changes in the system.

The subsystem interfaces are the interface layers between the CI and the subsystems. This makes the design more flexible and enables us to change any of the subsystems without much change in the CI — only the corresponding SSI need to be changed. The role of the SSIs are:

- Report any changes to the CI.
- Receive messages from the CI with required updates.
- Perform the necessary updates in the actual files of the subsystem.
- Send acknowledgments or error messages to the CI.

The assumption is that there is a user at each subsystem (by a user here we mean one or more skilled persons who understand this subsystem), and there is a user operating the central interface as a general director and coordinator for the design process. In other words, the CI is to assist in the coordination of the job and to help communicate with all subsystems. Figure 31 shows an overall view of the suggested design.

In the first phase of implementing the PE, the users have more work to do. The CI and SSIs maintain the information routing between the subsystems by sending messages to the corresponding user at each subsystem, then the action itself (e.g., update a file) is accomplished by the user. Later on, the system will be automated to perform most of these actions itself and the user will simply be informed of the actions taken.

7.3.1 Communication Protocols

The main purpose of this environment is keep all the subsystems informed of any changes in the design parameters. Therefore, passing information between the subsystems is the most important part of this environment. To be able to control the information flow, some protocols were developed to enable the communication between these subsystems in an

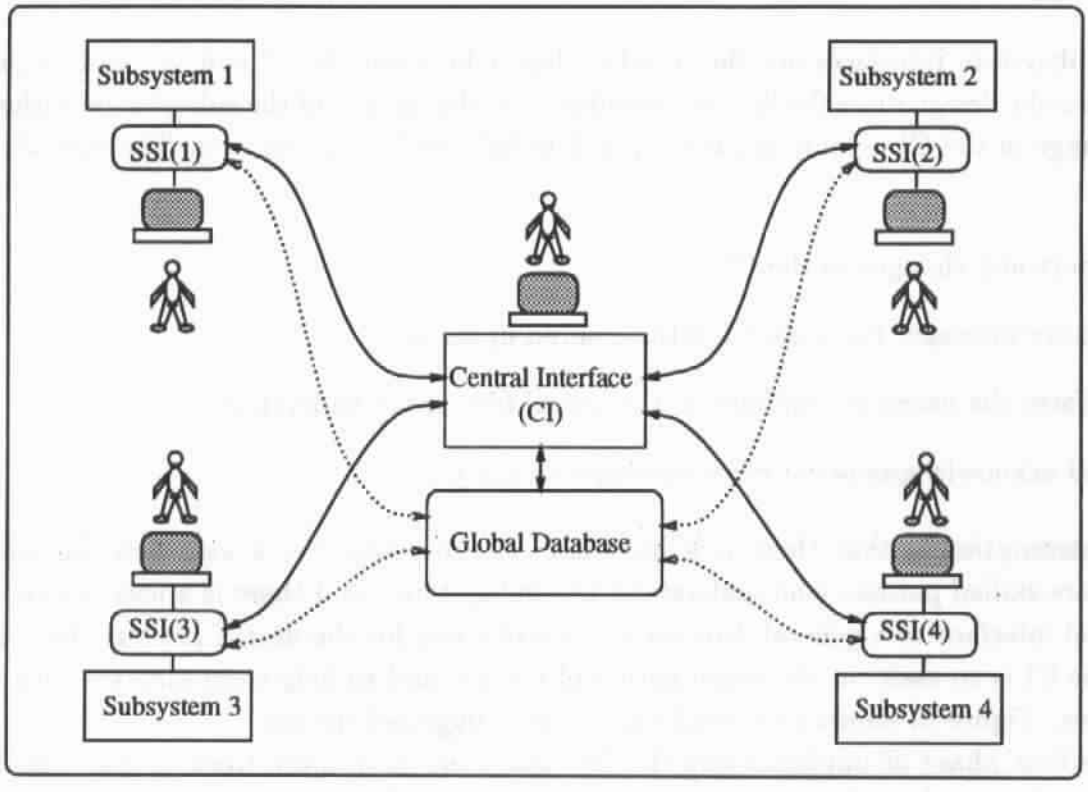


Figure 31: Overall design of the prototyping environment.

organized manner. In our design, all subsystems communicate through the CI which is responsible for passing the information to the subsystems that need to know.

There are two types of events that can occur in this system:

1. Change reported from one of the subsystems.
2. Request for data from one subsystem to another.

Figure 32 shows the protocol used for the first event represented by a finite state machine (FSM). The states of this FSM are:

1. Steady state: Do nothing.
2. Change has been reported: send lock message to all subsystems. Apply relations and check constraints. If constraints are satisfied, go to state 3. If constraints are not satisfied, report these to sender and go to steady state.
3. Constraints are satisfied: Notify the subsystems with the changes and wait for acknowledgments.
4. Acknowledgments received from all subsystems: Send the final acknowledgment to the subsystems and go to steady state.
5. Acknowledgments not Ok: Send a "change-back" command to the subsystems and go to steady state.

Figure 33 shows the protocol for the second event. The states in this FSM are:

1. Steady state: Do nothing.
2. Request for S2 received from S1. Send the request to S2.
3. Required data found at S2. Send data to S1 and go to steady state.
4. Required data not found at S2. Send report to S1 and go to steady state.

The suggested protocol can be described in algorithmic notation as follows:

```
do while true
   if change reported then
       lock messages
       apply relations
       check constraints
```

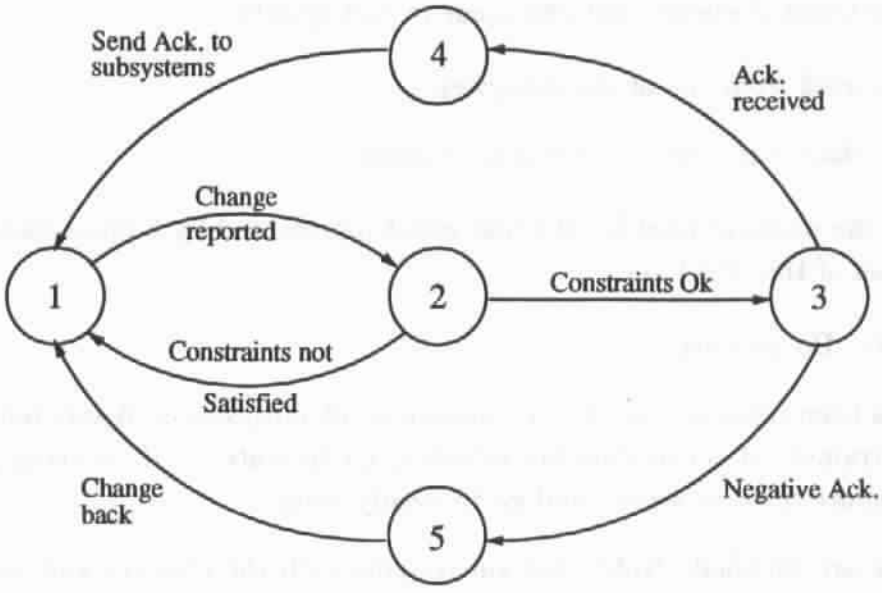


Figure 32: Finite state machine representation for the change protocol.

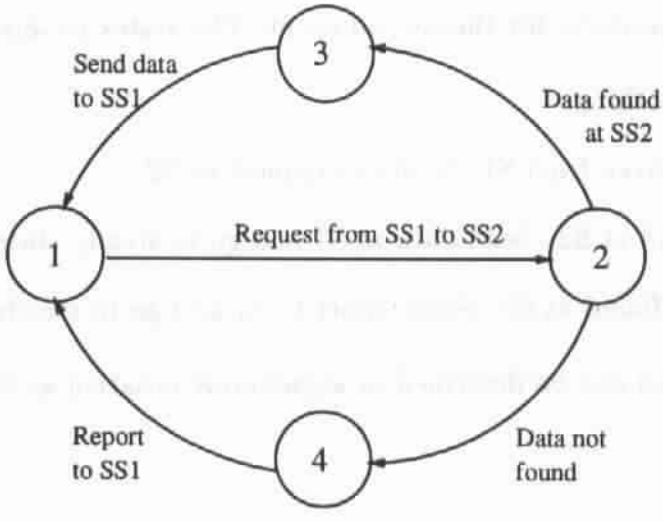


Figure 33: Finite state machine representation for the data request protocol.

```
    if constraint satisfied then
        report changes to subsystems
        wait for subsystems acknowledgment
        if all acknowledgments ok
            update database
            report the new status
        else
            send a change-back message to subsystems
            report failure to sender
    else
        report nonsatisfied constraints to sender
    send final acknowledgment to subsystems
else if data-request reported then
    send request to the appropriate subsystem
    if data received then
        send data to sender
    else
        send negative acknowledgment to sender.
```

Figure 34 shows a possible scenario when applying this protocol. In this algorithm we assume that all system constraints are located in the CI; however, any subsystem may reject the proposed values by other subsystems due to some unmodeled constraints. This can happen either because there are some "new" constraints that are not reported to the CI, or because some constraints are too hard to be easily represented in the constraint format in the CI.

7.3.2 Design Cycles and Infinite Loops

One problem that arises in our PE is that in some cases infinite design loops might occur due to some conflict between the constraints in different subsystems. For example, assume that the design system changed the link length to some value, say from 3.0 to 2.0 inches, to satisfy some performance requirements. This change would change the link mass as well, say from 1.5 to 1.0 lbs. According to the mass change the gear ratio has to change or the motor should be replaced, but if there is a constraint on the sprocket radius such that it can be increased, and there is no other motor with lower rpm, then the mass should be changed again to be 1.5 lbs, which requires the length to be 3.0 inches again. If we let the system continue, the design system will change the link length again and the loop will continue.

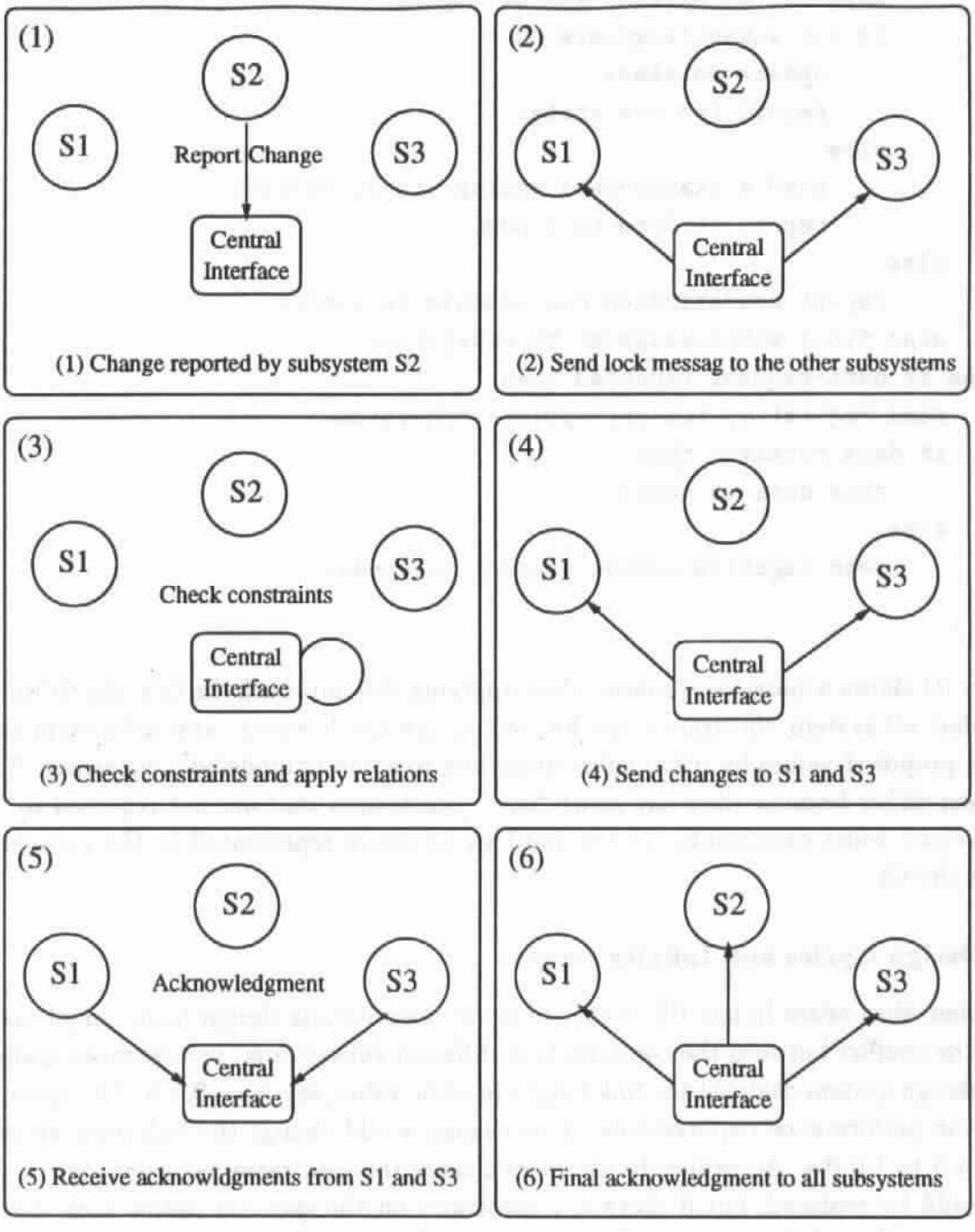


Figure 34: Possible scenario for the communication between the subsystems.

There are several solutions to this problem. One way is to make the user part of this loop so that some of the performance requirements can be changed, or a solution can be selected even if it does not meet some required criteria. This requires the user to be a skilled person who has the knowledge and experience in the design process, and also to have the authority to change and select solutions irrespective of the original requirements. Another solution is to put some limitations on the subsystem regarding its ability to change some of the design parameters. These limitations should guarantee infinite loop prevention in the system. A third solution is to put all the constraints in the CI. This allows the CI to check the solution and detect any violation to any of the constraints; then it may ask the user to decide on another solution or to change some of the performance requirements and run the design subsystem again. The last solution has the user in the loop as well, but incorporating all the constraints in the CI reduces the interprocess communication and speeds up the checking process. This last solution was chosen in our design.

7.3.3 Central Interface Design Options

There are several design choices for the CI. The following is a description of these options along with the advantages and disadvantages of each one.

- The CI is responsible for any changes in the system and no other subsystem can perform any changes, but they can make suggestions to the CI. This means that the design subsystem is part of the CI. The advantages of this are:
  - More control on the design process.
  - No infinite cycles can happen.

  The disadvantages of this option are:
  - The user interface for the CI is more complicated.
  - More data should be kept in the global database, such as performance requirements, objective functions, etc.
  - A highly skilled user is required. This user should be able to perform both design and coordination at the same time.
  - An optimal design subsystem cannot be used in the system.

- The design subsystem gives initial values for some of the parameters and the CI supplies the rest of the parameters and also can override some of the parameters supplied by the optimal design subsystem. The advantages are:
  - Any optimization packedge can be used to obtain the initial values.

- Some quick changes can be done from the CI directly.

The disadvantages are:

- The user interface for the CI is complicated.
- Skilled users are required at both the design subsystem and the CI.

- No changes can be done by the CI, and the CI is only informed of any changes and reports them to the other subsystems. Also, the CI is responsible for checking the constraints and applying the update rules. The advantages are:

  - The user of the CI does not need to know much about the design details and technicalities.
  - Any design subsystem can be used by writing the required SSI for it and including it in the system.
  - The infinite design cycles are eliminated since the design constraints will be checked in the CI.

  The disadvantages are:

  - The CI has no control on the design parameters and any required changes should be done through one of the subsystems.
  - This scheme requires heavy use of *interprocess communication* which needs more sophisticated protocols to maintain reliable data transmission.

The last option has been chosen for our design. Thus, the CI will not change any of the parameters directly; however some of the parameters will be changed by the CI only when applying the update rules. For example, the link mass is calculated as the link length times the link cross-sectional area times the material density. So if any of the three parameters (length, area, density) is changed (usually by the design subsystem), then the mass will change by the corresponding update rule. The update rules should be cycle-free, i.e., any derived parameter must not change — either directly or indirectly — any of the parameters that are used in its calculations.

7.4 Object-Oriented Analysis

Object analysis approach is used to determine the system components and functions, and the relations between them. The top-down approach is used starting from the main objects in the PE, then analyze each of these objects in more detail until the primitive data items are reached. Second, the functionality of the system has been analyzed and described using high level algorithms. Finally the corresponding member functions of the suggested classes has been implemented. Figure 35 shows the top view of the main components in the system, and Figure 36 shows one of these components in detail.

7.5 Prototyping Environment Database

A database for the system components and the design parameters is necessary to enable the CI to check the constraints, to apply the update rules, to identify the subsystems that should be informed when any change happens in the system, and to maintain a design history and supply the required reports.

This database contains the following:

- Robot configuration.
- Design parameters.
- Available platforms.
- Design constraints.
- Subsystems information.
- Update rules.
- General information about the system.

Now the problem is to maintain this database. One solution is to use a database management system (DBMS) and integrate it in the prototyping environment. This requires writing an interface to transform the data from and to this DBMS, and this interface might be quite complicated. The other solution is to write our own DBMS. This sounds difficult, but we can make it very simple since the amount of data we have is limited and does not need sophisticated mechanisms to handle it. A relational database model is used in our design, and a user interface has been implemented to maintain this database. For the current design, by making a one-to-one correspondence between the classes and the files, reading and writing a file can be accomplished by adding member functions to each class. In this case no need for a special DBMS and all operations can be performed by simple functions.

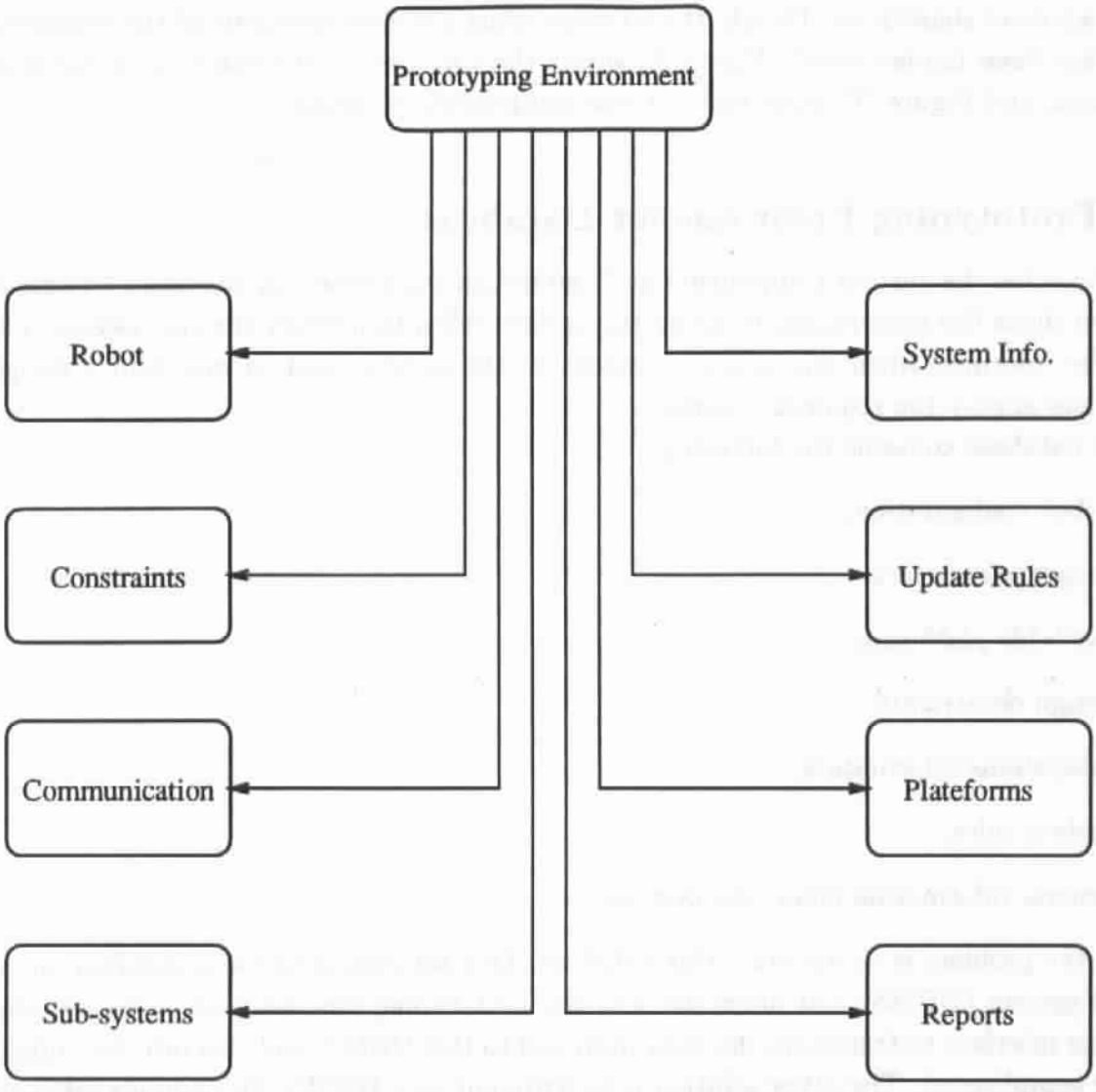


Figure 35: The main components of the robot prototyping environment.

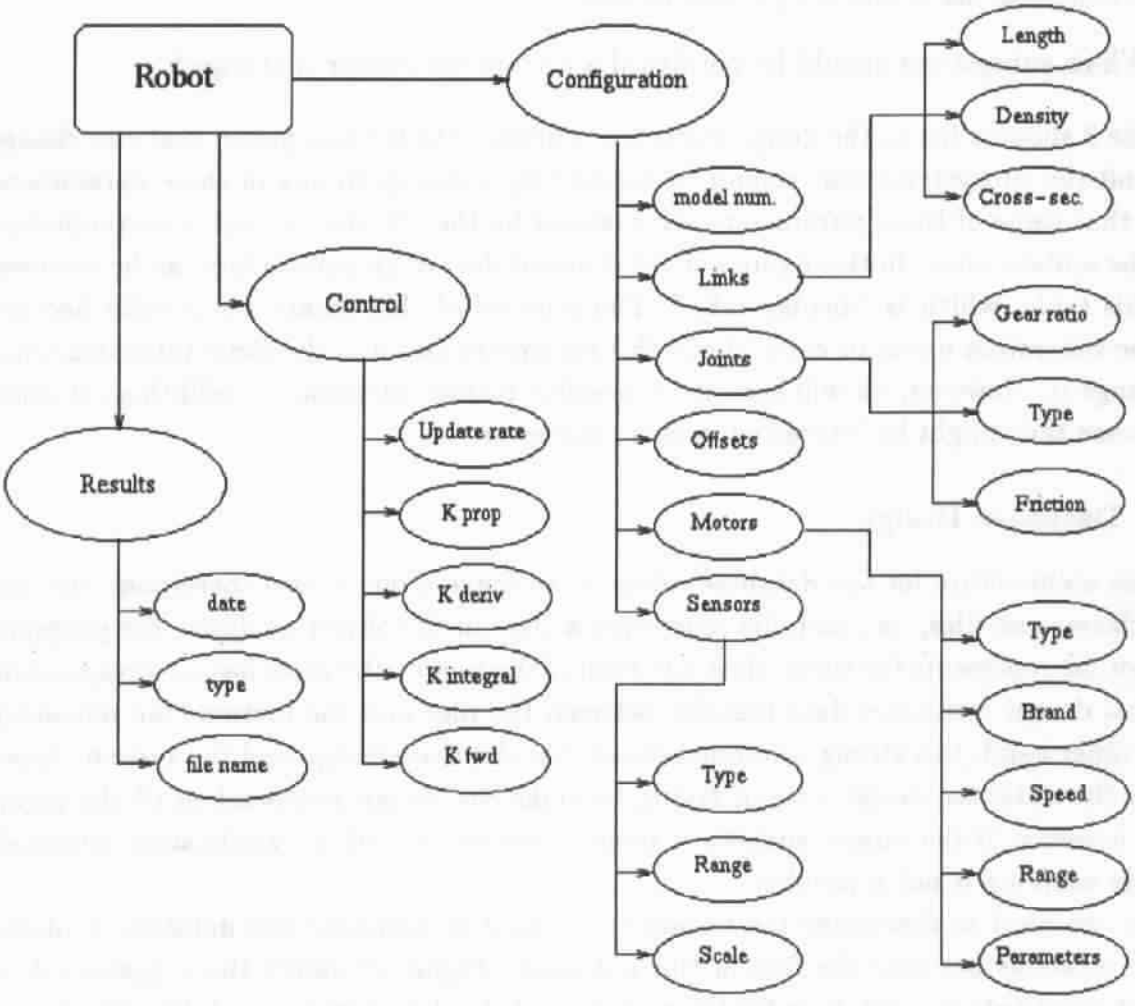


Figure 36: Detailed analysis for the robot classes.

7.5.1 Design Parameters

The design parameters are the most important data items in this environment. The main purpose of this system is to keep track of these parameters and notify the subsystems of any changes that occur to any of these parameters. For the system to perform this task, it needs to know the following data:

- A complete list of the design parameters.
- Which subsystems should be notified if a certain parameter is changed.

Table 3 shows a list of the design parameters along with the subsystem that can change them and the subsystems that should be notified by a change in any of these parameters. Notice that some of these parameters are changed by the CI, this change is accomplished using the update rules. In this figure note that one of the design parameters can be removed from this table, which is "display rate." The removal of this parameter is valid because only one subsystem needs to know about this parameter and it is the same subsystem that can change it. However, we will keep it for possible future extensions or additions of other subsystems that might be interested in this parameter.

7.5.2 Database Design

A simple architecture for the database design is to make a one to one correspondence between classes and files, i.e., each file represents a class in the object analysis. For example, the robot file represents the robot class and each of the robot subclasses has a corresponding file. This design facilitates data transfer between the files and the system (the memory). On the other hand, this strong coupling between the database design and the system classes violates the database design rule of trying to make the design independent of the application; however, if the object analysis is done independently of the application intended, then this coupling is not a problem.

Now, we need to determine the format to be used to represent the database contents and the relations between the files in this database. Figure 37 shows the suggested data files that constitute the database for the system, and the data items in each file. The figure also shows the relations between the files. The single arrow arcs represent a one-to-one relation, and the double arrow arcs represent a one-to-many relation.

7.5.3 The Design History

In this database design, a history of the design changes can be maintained to assist the designers while developing the prototype robot. This history includes the following:

- Date of the design.

Table 3: Subsystem notification table according to parameter changes.

Design Parameter	CI	Design	Control	Simulation	Monitor	HW-Select	CAD/CAM	Ordering	Assembly
robot model	○	●	○	○	○		○		○
link length	○	●	○	○	○		○		○
link mass	●		○	○			○		○
link density	○	●					○		○
link cross area	○	●					○		○
joint friction	○	●	○	○			○		○
joint gear-ratio	●						○		○
update rate	○	●	○	○	○	○			
comm. rate	○	○	○	○		●			
motor rpm	○							●	○
motor range	○	●	○	○	○			○	○
sensor range	○	●	○	○	○	○		○	○
PID parameters	○	●	○	○					
display rate	○				●				
plateform	○				○	●			○

○ To be notified

● Make change

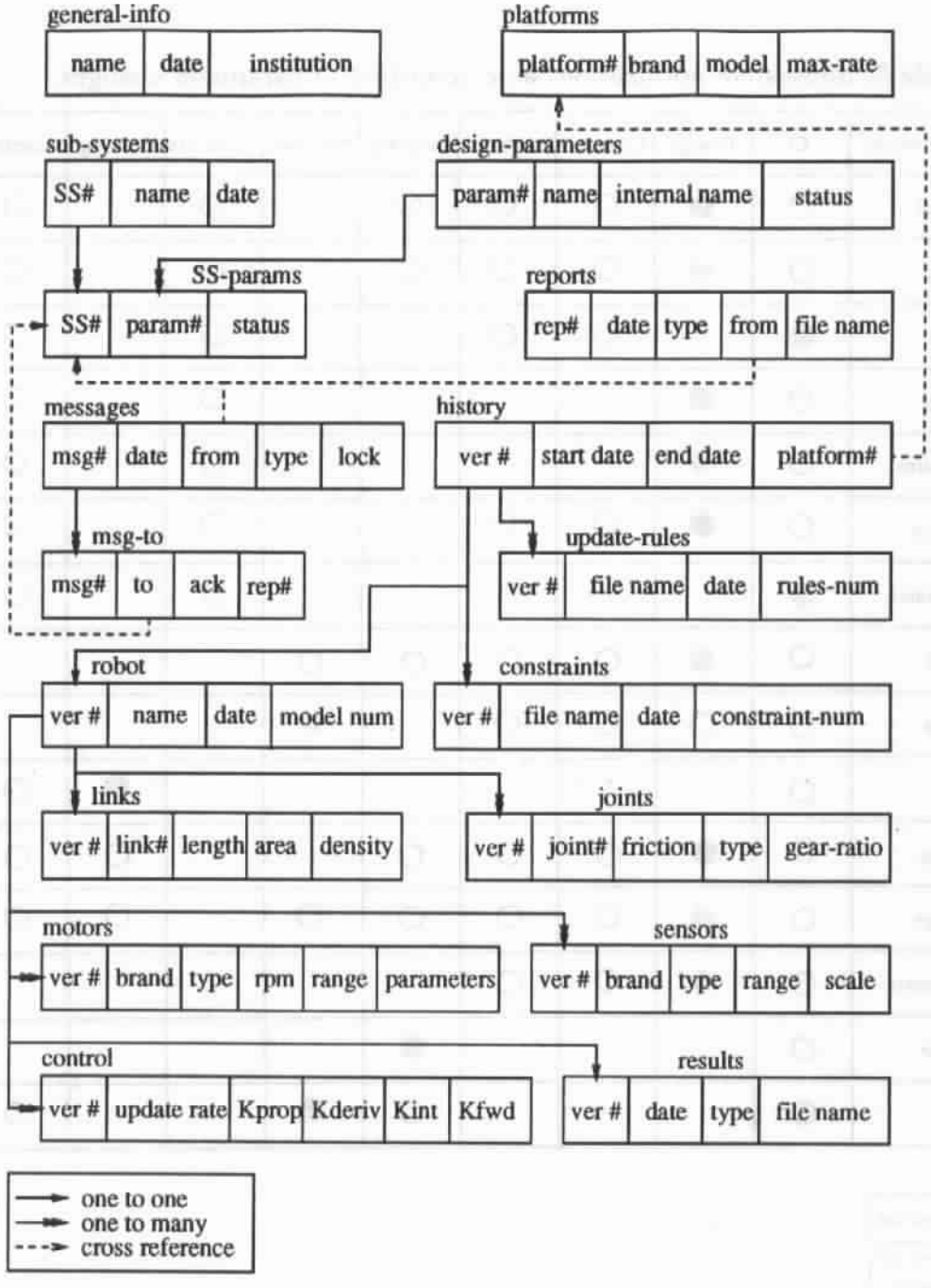


Figure 37: Database design for the system.

- Values of the design parameters.
- Constraints and update rules at that time.
- Robot configuration (links, motors, sensors, etc.)
- Platform used for this design.
- All messages between the systems during this design.

The design can be added to the design history upon the user request. This is accomplished by adding new records to the database files with a version number specified by the user. For example, if the user wants to add the current design status to the design history, this is accomplished by clicking on the "history" button, and typing the version number (e.g., "design-dec-9-93"). A copy of the necessary records from the current design then will be added to the files.

The retrieval of any design from the design history requires the user to input the version number, and then the information about this design will be displayed. The file "history" shown in Figure 37 contains some information about the design such as the design number, the starting date, the finishing date, and the platform used for that design.

7.6 Constraints and Update Rules Compiler

A compiler is provided to generate C++ code for the constraints and the update rules. First, the syntax of the language that is used to describe the constraints and the update rules is described. Second, the generated code is determined.

Using a compiler instead of generic on-line evaluator for the constraints and the update rules has the following advantages:

- All constraints are saved in one text file (likewise the update rules). This makes the data entry very easy. We can add, update, and delete any constraint or update rule using any text editor.
- Complicated data structures are not required for evaluation.
- The database is very simple, which facilitates maintaining the design history.
- Format changes, or changes in the generated code require only changes to the compiler, and no changes in the system are required.

On the other hand, it has the following disadvantages:

- The generated code has to be included in the system and the whole system must be recompiled.
- A compiler needs to be implemented.

Notice that the changes in the constraints or the update rules are not frequent, so recompiling the system is not a big problem. Also, the syntax used is very simple; therefore the compiler for such language is not difficult to implement.

7.6.1 Language Syntax

By analyzing the design constraints and the update rules, we constructed a simple description of the language to be input to the compiler. There are two options in this design, either to have one compiler for both the constraints and the rules, or to build two compilers, one for each. From the analysis of the constraints and the rules we found that there are many similarities between them; thus building one compiler for both is the logical option in this case.

The following is the language definition in Backus Naur Form (BNF):

```
          <program> ::  <constraint-prog> | <rule-prog>
  <constraint-prog> ::  begin-constraints
                          <constraint-sequence>
                        end-constraints
       <rule-prog> ::  begin-rules
                          <rule-sequence>
                        end-rules
<constraint-sequence> ::  <constraint> ; <constraint-sequence> |
                          <constraint> ;
    <rule-sequence> ::  <rule> ; <rule-sequence> | <rule> ;
       <constraint> ::  <exp> <comparison-op> <exp>
             <rule> ::  <variable> = <exp>
              <exp> ::  <exp> * <term> | <exp> / <term> | <term>
             <term> ::  <term> + <factor> | <term> - <factor> |
                        <factor>
           <factor> ::  <variable> | <constant> | (<exp>)
         <variable> ::  <alphabet> <alphanum> | <alphabet>
         <constant> ::  <int>.<int> | - <int>.<int> |
                        <int> | - <int>
              <int> ::  <digit> <int> | <digit>
```

7.6.2 The Generated Code

As mentioned before, this compiler generates C++ code which is integrated with the CI system to check the constraint or apply the update rule. Each variable in the input to the compiler corresponds to one design parameter. For example, "link1_length" corresponds to the variable in the CI system that represents the length of link number one in the robot configuration. The code generator uses a lookup table to find the corresponding variable name, and this table is part of the CI database. A simple flat file is used to store this table since the number of the design parameters is small.

The generated code for the constraints is the function "pe.check_constraints" that returns true if all constraints are satisfied, else it returns false, and reports which constraints are not satisfied. For the rules, the code generated is the function "pe.apply_rules" which calculates all corresponding design variables according to the given rules. The following examples are the code generated for the two examples shown in the previous section.

```
bool
ci::check_constraints()
{
    bool status[no_of_constraints] ;
    int i = 0 ;

    status[i++] = robot.configuration.link[0].length > 1.2 ;
    status[i++] = robot.configuration.link[1].length > 1.5 ;
    status[i++] = robot.configuration.link[2].length > 0.8 ;
    status[i++] = robot.configuration.link[1].length +
                  robot.configuration.link[2].length < 3.0 ;
    status[i++] = robot.configuration.link[0].mass < 1.4 ;
    status[i++] = robot.configuration.link[1].mass +
                  robot.configuration.link[2].mass  < 4.0 ;
    status[i]   = robot.configuration.joint[1].gear_ratio < 5.0 ;

    constraints.generate_report(status) ;   // report the result

    return (and_all(status)) ;
}


void
ci::apply_rules()
```

```
<alphanum> ::  <alphabet> <alphanum> |
               <digit> <alphanum> |
               <alphabet> | <digit>
<alphabet> ::  a..z | A..Z | _
   <digit> ::  0..9
<comparison-op> ::  = | < | > | <= | >= | <>
```

The following is an example of some constraints described using this syntax:

```
begin-constraints
  link1_length > 1.2 ;
  link2_length > 1.5 ;
  link3_length > 0.8 ;
  link2_length + link3_length < MAX_TOT_LEN ;
  link1_mass < 1.4 ;
  link2_mass + link3_mass < 4.0 ;
  joint1_gear_ratio < 5.0 ;
end-constraints
```

Another example showing some update rules using the same syntax:

```
begin-rules
  link1_mass = link1_length * link1_density * link1_cross_area ;
  link2_mass = link2_length * link2_density * link2_cross_area ;
  link3_mass = link3_length * link3_density * link3_cross_area ;
  joint1_gear_ratio = motor1_speed / link1_max_speed ;
end-rules
```

From these examples it is clear that adding arrays to this language can reduce the length of the programs, but given the fact that these constraints and rules will be entered once at installation time, then adding or changing these rules and constraints will not be so frequent, thus, we will not complicate the compiler, at least in the first design phase. Some error detection and recovery modules for syntax error handling can be added to this compiler later.

```
{
    robot.configuration.link[0].mass =
        robot.configuration.link[0].length *
        robot.configuration.link[0].cross_area *
        robot.configuration.link[0].density ;
    robot.configuration.link[1].mass =
        robot.configuration.link[1].length *
        robot.configuration.link[1].cross_area *
        robot.configuration.link[1].density ;
    robot.configuration.link[2].mass =
        robot.configuration.link[2].length *
        robot.configuration.link[2].cross_area *
        robot.configuration.link[2].density ;
    robot.configuration.joint[0].gear_ratio =
        robot.motor[0].speed /
        robot.configuration.joint[0].max_speed ;
}
```

In the first example, the function *generate_report* reports the results of checking the constraints; if all constraints are satisfied it reports that, otherwise, it will generate a list of the unsatisfied constraints. The function *and_all* is obvious. It returns the result of ANDing the elements in the array *status*.

In the second example, some of the design parameters are calculated given the values of some other parameters. The compiler should not allow the change of any parameter that should not be changed by the CI system. This can be detected using the *alter_flag* in the design parameters table.

To update the constraints or the update rules the file containing the old definition will be displayed and the user can add, delete, or update any of the old definitions. Then the new file will be compiled and integrated with the system.

7.7 Implementation

In the following subsections some implementation issues are investigated, and the different components in our design and how we implemented each of them are described.

7.7.1 The Central Interface

The central-interface (CI) is the core program that handles the communication between the subsystems, and maintains a global database for the current design and a history of

previous designs. There are several types of messages used in the communication. Table 4 shows the different types of messages with a brief description and the direction of each.

The CI is the implementation of the communication protocols described in Section 7.3.1. There are some features and enhancement to the protocols has been added to the CI. For example, When the CI receives a *change* message from an SSI, it directly sends lock messages to the other subsystems so that no more changes can be sent from any SSI until they receive a *steady* message. This solves the concurrency problem of more than one system send changes to the CI at the same time. The first message received by the CI will be handled and the others will be ignored. If an SSI receives a *lock* message after it sent a *change* message, that means its message was ignored. Another feature added to the CI is the ability to detected if an SSI is working or not by tracing the *SSI_Start* and *SSI_Stop* messages.

The CI is managing a number of data files that contains information about the robot configuration, platforms, reports, design history, subsystems, and some general information about the project. The basic file operation was implemented by defining a file class, and by adding some member functions to each class in the system that performs the required file management operations. The file operations that are implemented in the system are:

open: open a file in one of three modes: input, output, or input-output mode.

close: close an open file.

top: go to the first record in the file.

end: go after the last record in the file.

next: go to next record.

prev: go to previous record.

read: read the current record.

write: write a record to the end of the file.

find: find a record that contains a certain key.

file_size: returns the number of records in the file.

Some of these operations are class-specific functions such as, *read*, *write*, and *find*, while the rest are general operations that are implemented as member functions in the basic file class.

7.7.2 The PE Control System

The CI as described above has no user interface. To be able to control and manage the coordination between the subsystems, the PE control system (PECS) was implemented with some functionalities that enable the user to have some control over the CI.

The PECS is on top of the simple DBMS and a simple compiler for the update rules and the constraints. The user specifies the constraints and/or the update rules using a certain format (a language), then the compiler transforms this to C code that will be integrated with the system for constraint checking, and for applying the update rules. The compiler consists of two parts, a parser and a code generator. In the first phase the complexity of the compiler was reduced by making the user language less sophisticated. Later on this can be easily replaced by a more complicated compiler with an easier interface and more sophisticated error checking and optimization capabilities. A schematic view of the PECS is shown in Figure 38. Figure 39 shows the user interface for the PECS.

The PECS functions include:

Queries: which are some simple reports about the current robot configuration, previous configuration, general information about the system, the platforms, and the subsystems of the prototyping environment. Figure 40 shows a query for the current robot configuration.

Actions: which are the actual operations that control the CI. these actions include updating the constraints and the update rules, compiling the CI after including the new constraints and update rules, activating, and terminating the CI. Figure 41 shows one of these operations which is updating the constraints.

Reports: which are operations to manage the reports in the system, and to send and receive reports to and from the subsystems. The report can be text, graph, figure, postscript, or data file. Each report is saved with its type, date, sender, and the file that contains the report contents.

7.7.3 Initial Implementation of the SSIs

In the first phase of implementation, the SSIs serve as a simple interface layer between the CI and the user at each subsystem. They receive messages from the CI and display them to the user who takes any necessary actions. They also report any changes to the CI, and this is done by sending a message to the CI with the changes. Figure 42 shows that user interface for one of the SSIs.

In the next implementation phase, some of the actions will be automated and the user at each subsystem will be notified with any action taken. For example, updating a data

Table 4: Message types used in the communication protocols.

Type	Description	Direction
Change	Data change reported	SSI $\longrightarrow$ CI
Const_Not_Ok	Constraints not satisfied	CI $\longrightarrow$ SSI
Notify	Send changes to subsystems	CI $\longrightarrow$ SSI
Ack.	Positive acknowledgment	SSI $\longrightarrow$ CI
Neg_Ack.	Negative acknowledgment	SSI $\longrightarrow$ CI
Back	Change back	CI $\longrightarrow$ SSI
Steady	Final acknowledgment	CI $\longrightarrow$ SSI
Request	Request for data	CI $\longleftrightarrow$ SSI
Found	Data found	CI $\longleftrightarrow$ SSI
Not_Found	Data not found	CI $\longleftrightarrow$ SSI
Lock	lock messages	CI $\longrightarrow$ SSI
SSI_Start	SSI is activated	SSI $\longrightarrow$ CI
SSI_Stop	SSI is terminated	SSI $\longrightarrow$ CI
Terminate	Terminate the CI.	PE control $\longrightarrow$ CI

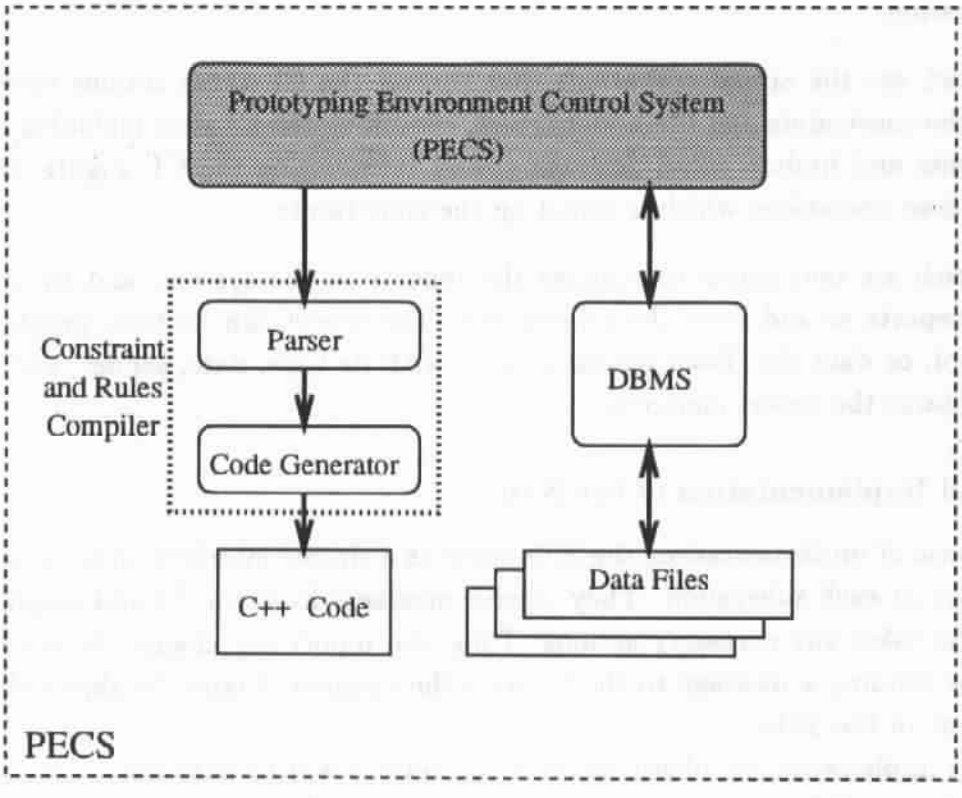


Figure 38: Schematic overview of the PECS.

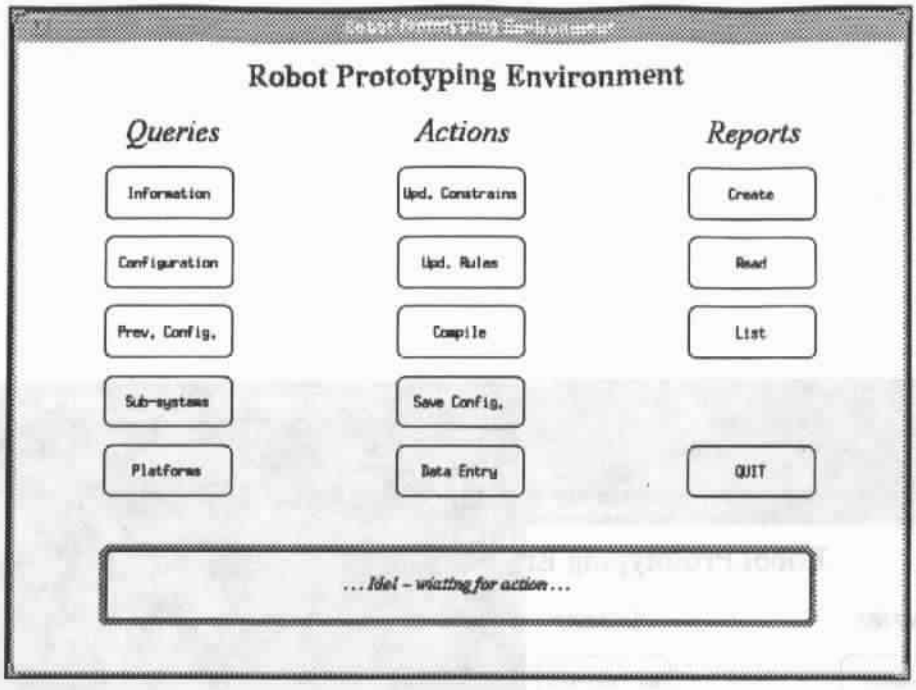


Figure 39: The main window for the PE control system.

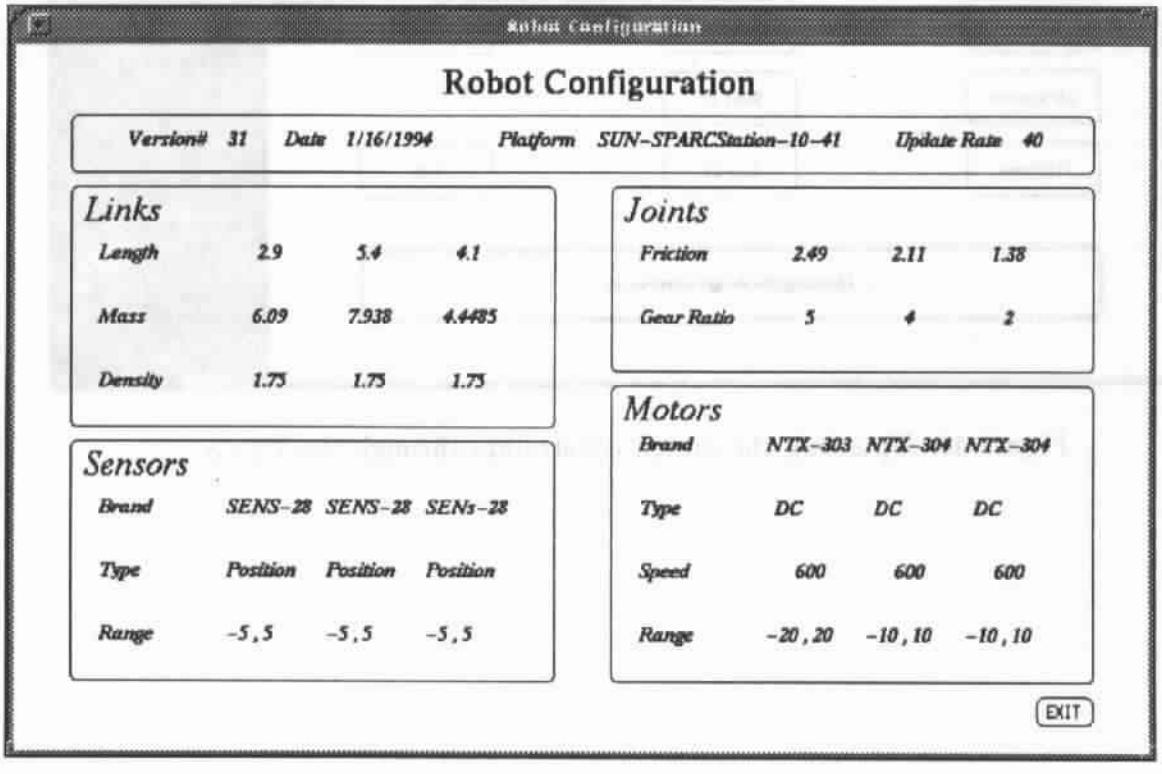


Figure 40: The current robot configuration window.

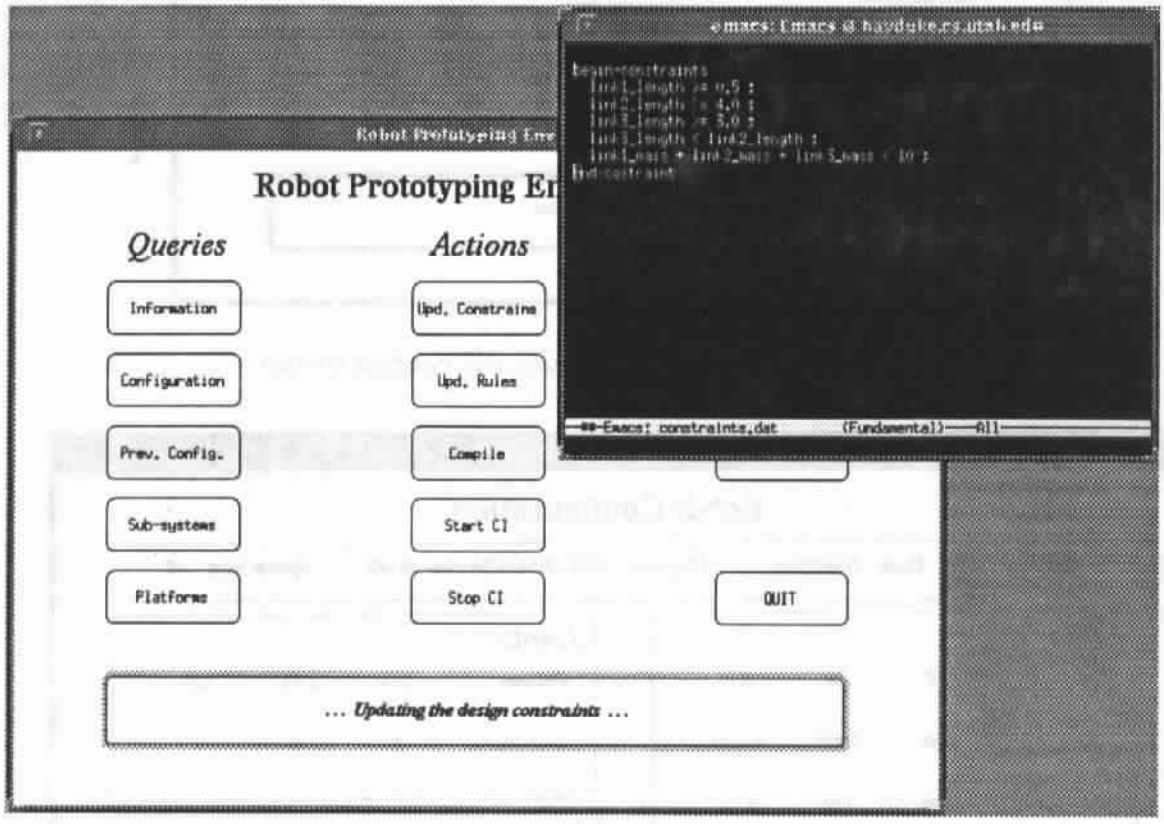


Figure 41: Updating the design constraints through the PECS.

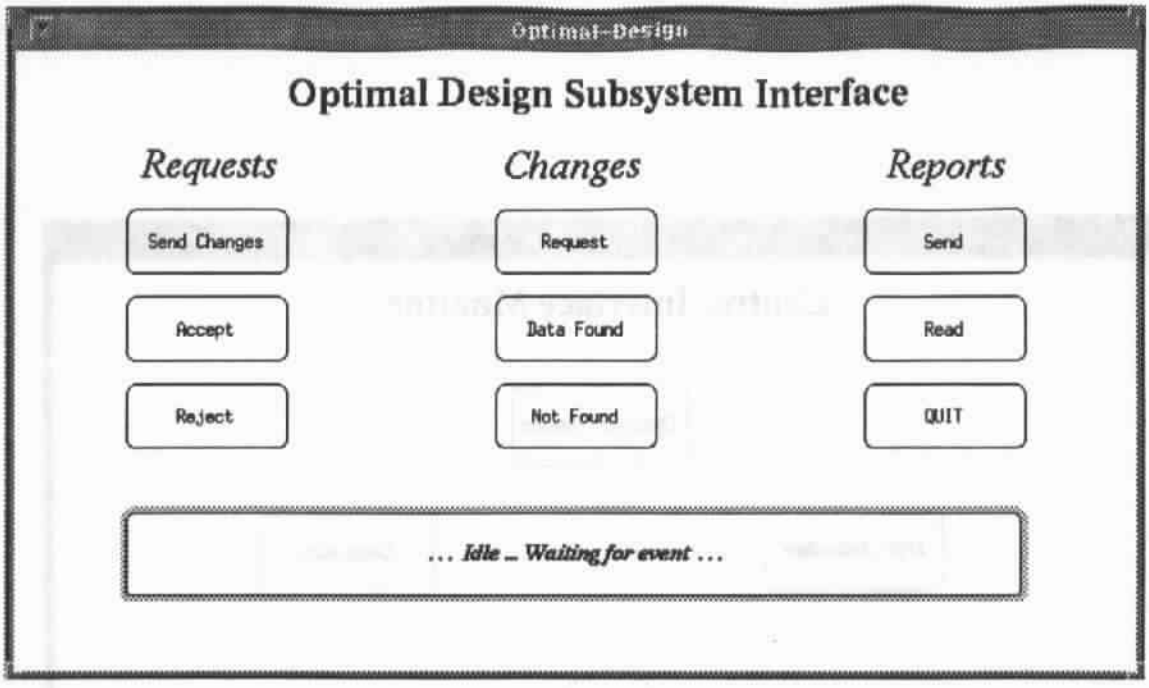


Figure 42: The user interface for the SSI.

file that is used by the subsystem can be automatically done by the SSI, given that it has the necessary information about the file format and the location of the changed data.

7.7.4 The Central Interface Monitor

The central interface monitor (CIM) enables the user to monitor the actions and the messages passing between the CI and the SSIs with a graphical interface. This interface shows the CI in the middle and the SSIs as small boxes surrounding the CI. The CIM also has a small text window at near the bottom. This text window displays a text describing the current action (See Figure 43). The messages are represented by an arrow from the sender to the receiver. Some results of testing the CI and the SSIs are represented in Section 8.4 with sequences of the CIM window showing the activities that took place in each experiment.

8 Testing and Results

In this section, several test cases are described along with the results obtained for the different components of the system that has been implemented. Some experiments that were performed for the one-link and the three-link robot are described, with the results shown graphically.

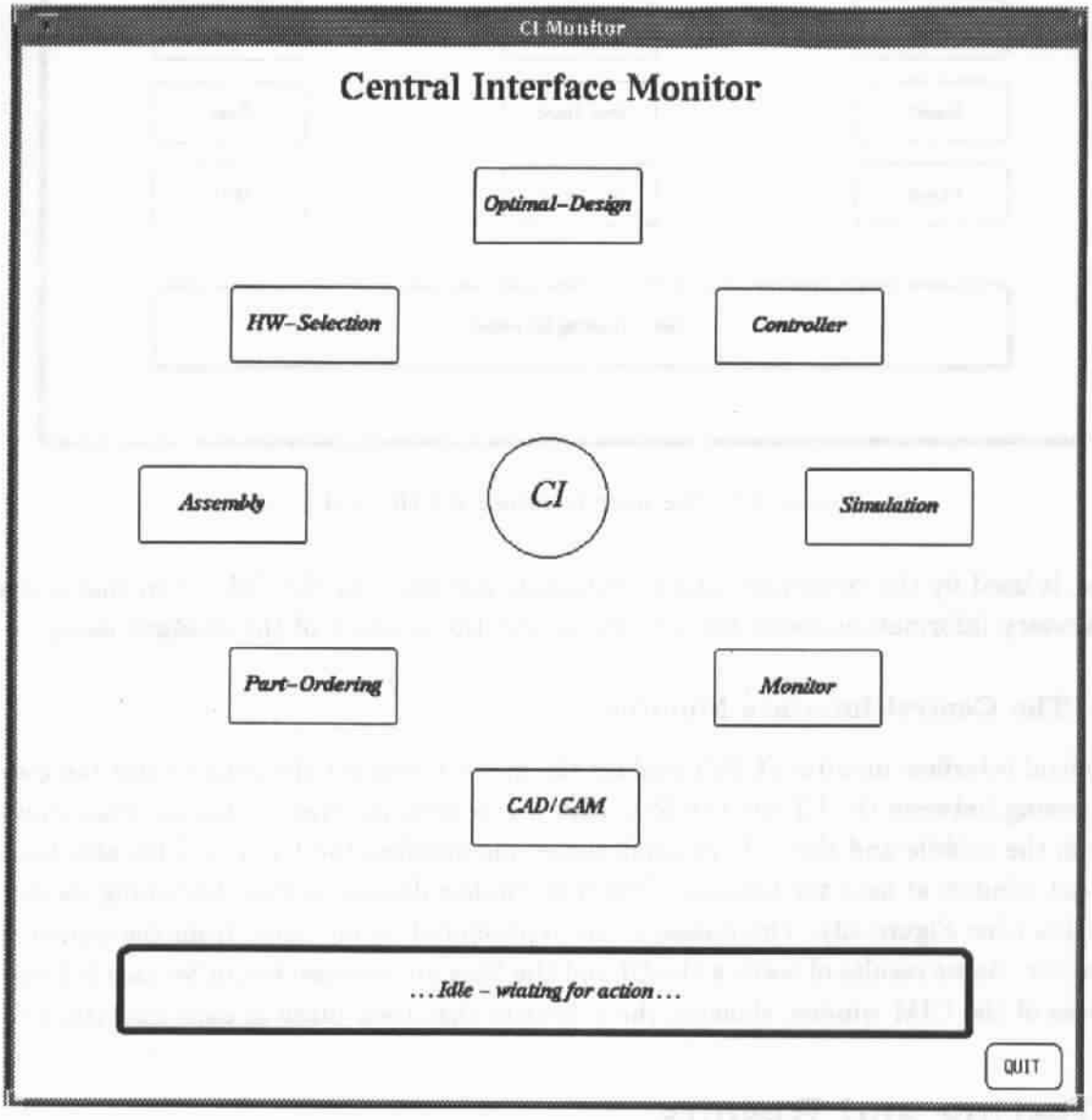


Figure 43: The user interface for the SSI.

8.1 One-link Robot

Building the three-link robot has passed through several stages until the final version was reached. As mentioned before, The first phase was controlling a one-link robot.

Three input sequences have been used for the desired positions, and after applying the voltage files to the motor using the I/O card, the actual positions and velocities are measured using a potentiometer for the position, and a tachometer for the angular velocity. These measured values are saved in other files, then a graphical simulation program was used to display the movement of the link, the desired and actual positions, the desired and actual velocity, and the error in position and velocity. Figures 44, 45, and 46 show the output windows displaying the link and graphs for the position and the velocity.

8.2 Simulator for three-link Robot

This simulator was used to give some rough estimates about the required design parameters such as link lengths, link masses, update rate, feedback gains, etc. It is also used in the benchmarking described earlier. Figure 47 shows the simulated behavior of a three-link robot. It shows the desired and actual position and velocity for each link and the error for each of them. It also shows a line drawing for the robot from two different view points.

This simulator uses an approximate dynamic model for the robot, and it allows any of the design parameters to be changed. For example, the effect of changing the update rate on the position error is shown in Figure 48. From this figure, it is clear that increasing the update rate decreases the position error.

8.3 Software PID Controller

A software controller was implemented for the three-link robot. This controller uses a simple local PID control algorithm, and simulates three PID controllers; one for each link. Several experiments and tests have been conducted using this software to examine the effects of changing some of the control parameters on the performance of the robot.

The control parameters that can be changed in this program are:

- forward gain (k_g)
- proportional gain (k_p)
- differential gain (k_v)
- integral gain (k_i)

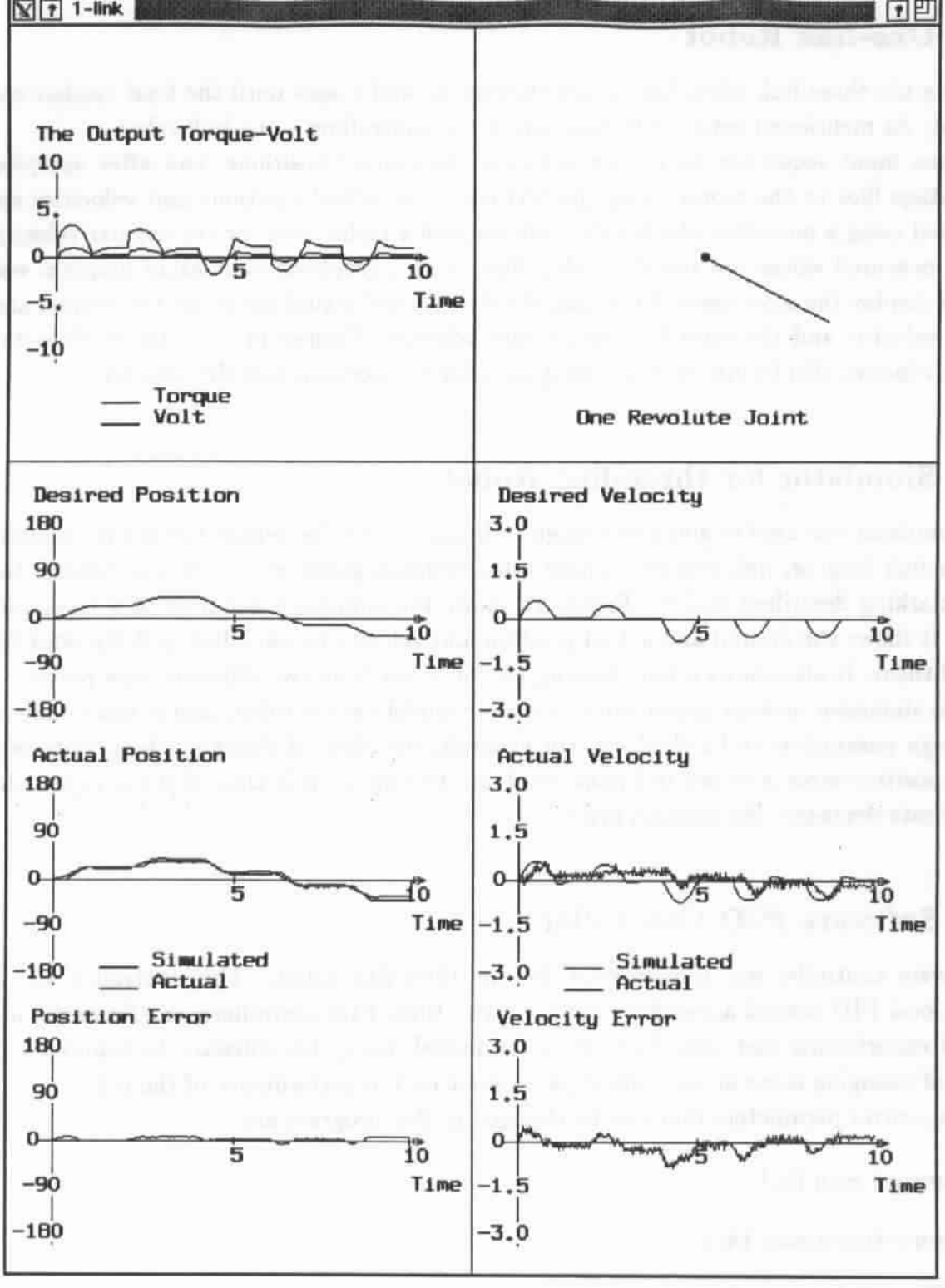


Figure 44: The behavior of the one-link robot for the first input sequence.

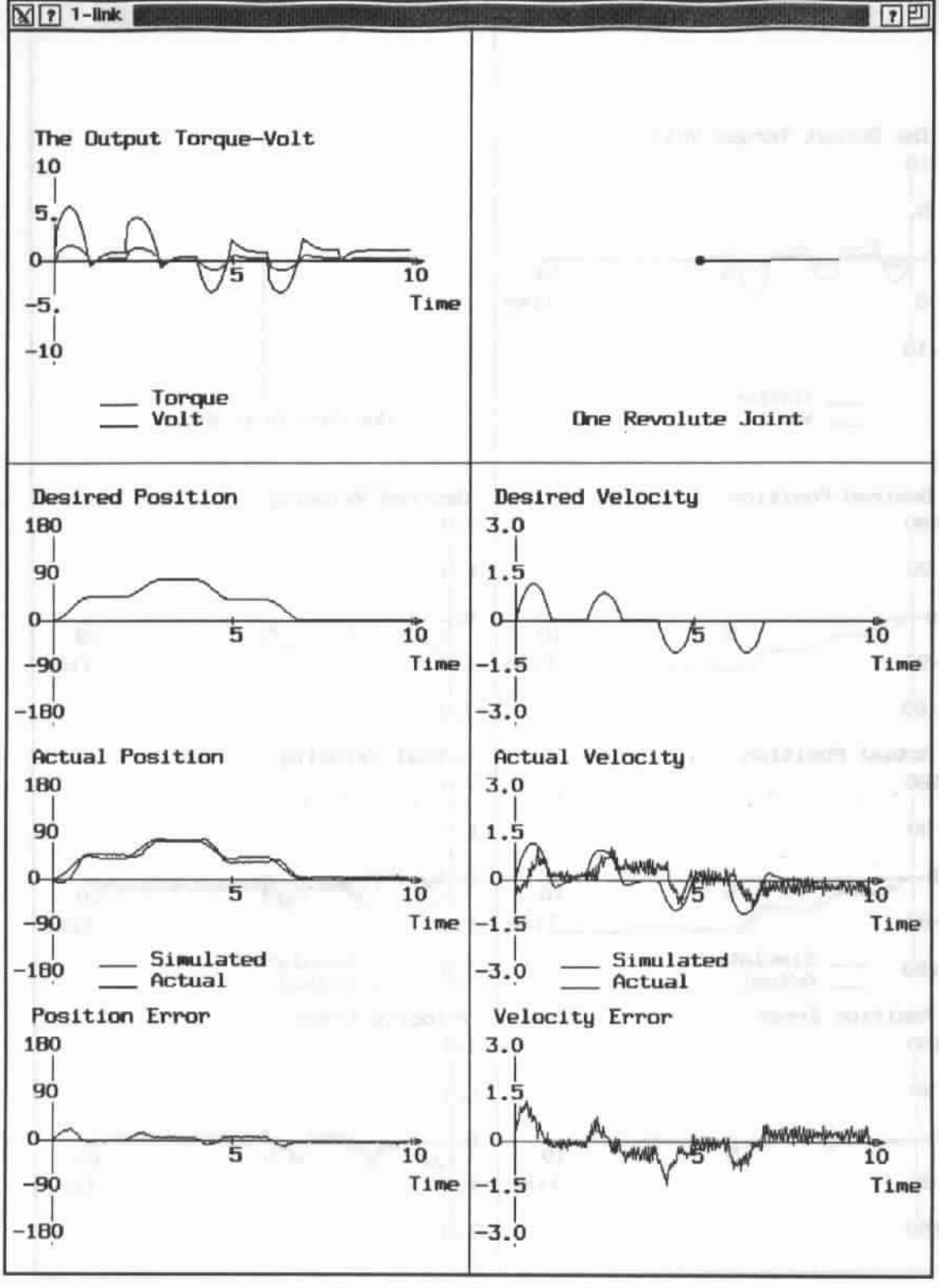


Figure 45: The behavior of the one-link robot for the second input sequence.

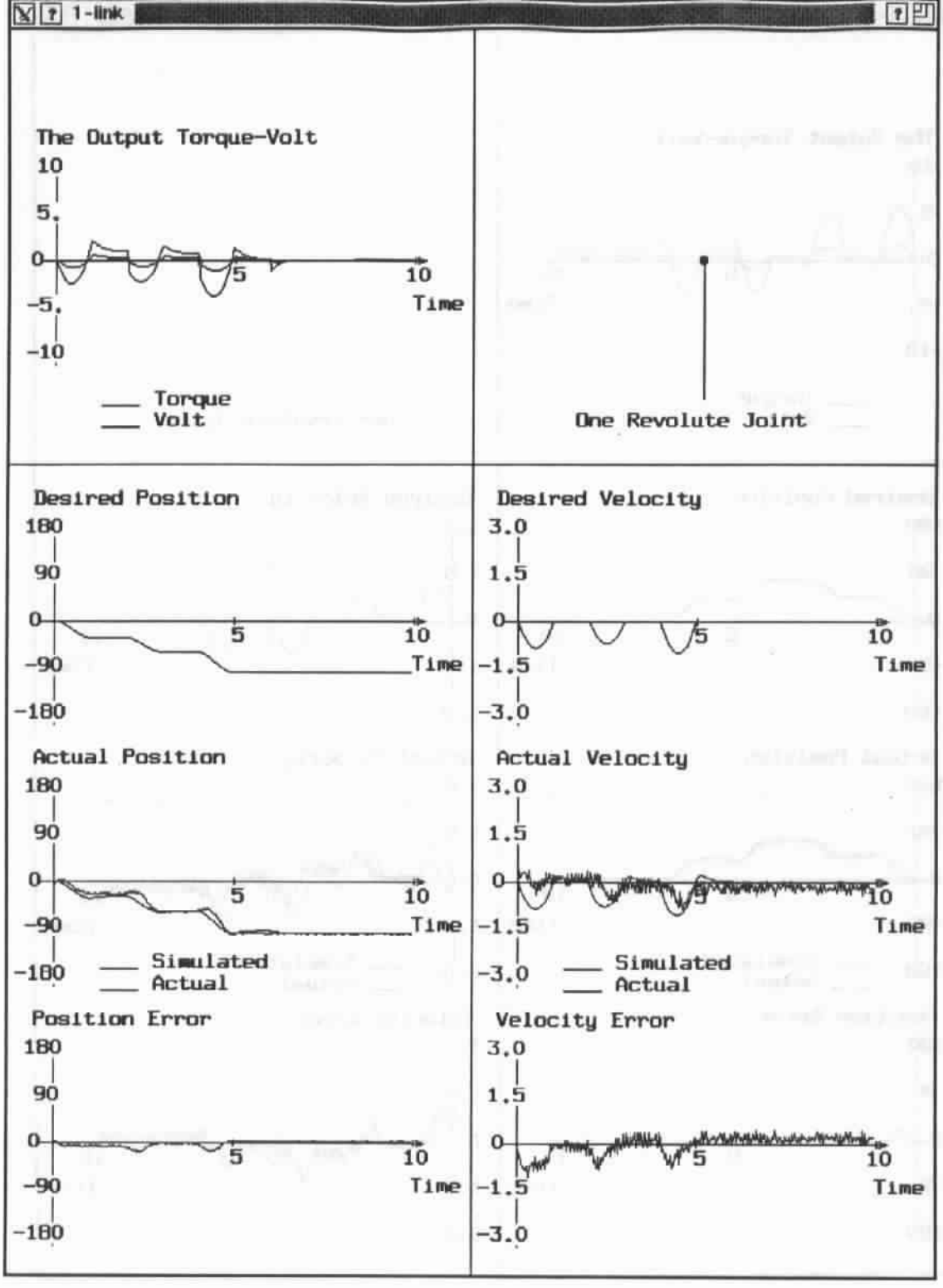


Figure 46: The behavior of the one-link robot for the third input sequence.

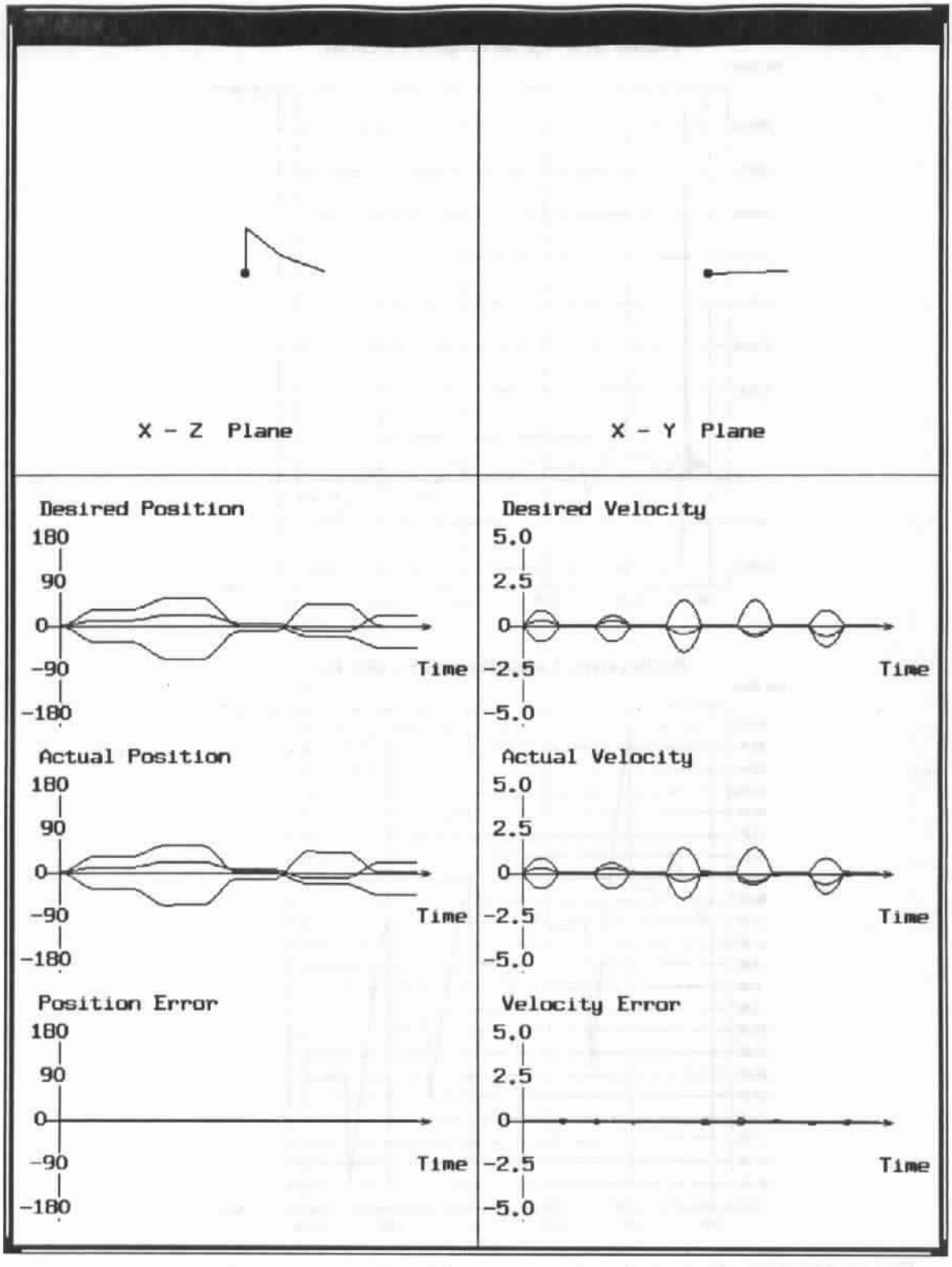
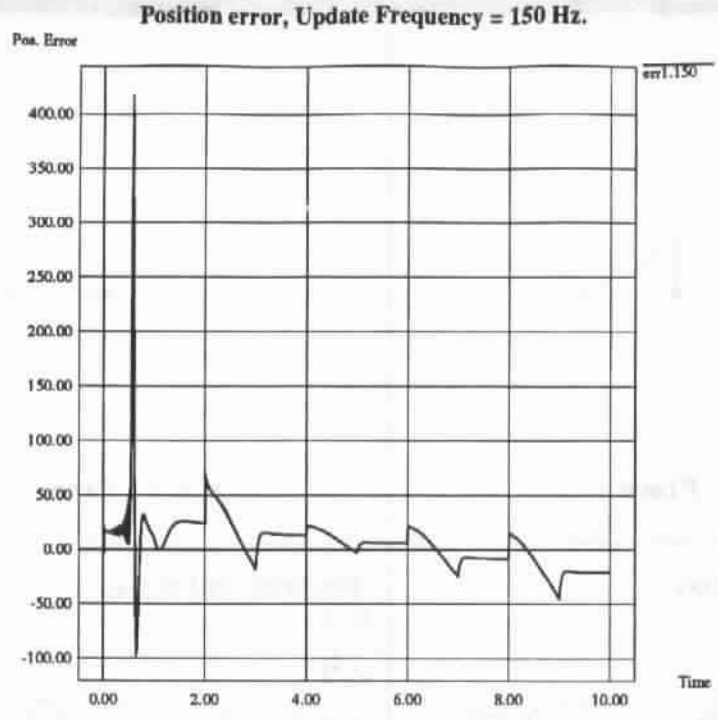


Figure 47: The output window of the simulator for the three-link robot.



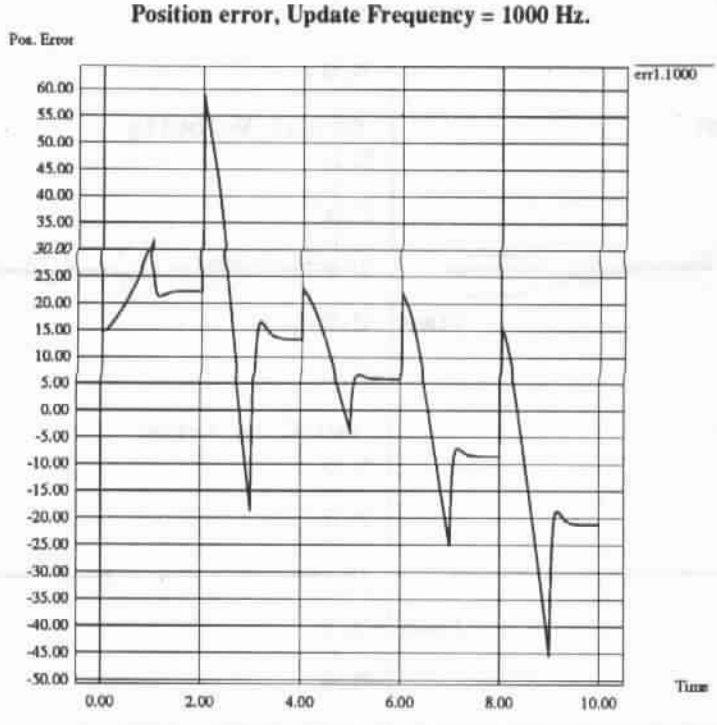


Figure 48: The effect of changing the update rate on the position error.

- input trajectory
- update rate

In these experiments, the program was executed on a Sun SPARCStation-10, and the A/D chip was connected to the serial port of the workstation. One problem we encountered with this workstation is the slow protocol for reading the sensor data, since it waits for an I/O buffer to be filled before it returns control to the program. We tried to change the buffer size or the time-out value that is used, but we had no success in that. This problem causes the update rate to be very low (about 30 times per second), and this affects the positional accuracy of the robot. We were able to solve this problem on an HP-700 machine, and we reached an update rate of 120 times per second which was good enough for our robot.

Figures 49, 50, 51, and 52 show the desired and actual position for different test cases using different feedback gains.

8.4 The Prototyping Environment

In this section, we will show several test cases for the prototyping environment. In the first test (Figure 53), the optimal design subsystem sent a data-change message to the CI. The CI in turn sent lock messages to all other subsystems notifying them that no changes will be accepted until they receive a final acknowledgment message. Then, the CI applied the relations and checked the design constraints. In this test case the constraints were satisfied, so the CI sent these changes to the subsystems that needed to be notified. After that, the CI waited for acknowledgments from the subsystems. In this case it received positive acknowledgments from the specified subsystems. Finally, the CI updated the database and sent final acknowledgment messages to all subsystems.

The second test case (Figure 54), was the same as the first case except that one of the subsystems (the CAD/CAM subsystem) has rejected the changes by sending negative acknowledgment message to the CI. Thus, the CI sent a change-back message to the specified subsystems and then sent a final acknowledgment messages to all subsystems. No changes in the database took place in this case.

In the last test case (Figure 55), the design constraints were not satisfied. Therefore, the CI sent a report about the nonsatisfied constraints to the sender (the optimal design subsystem). Then it sent final acknowledgment messages to all subsystems. Again, in this case, no changes in the database took place.

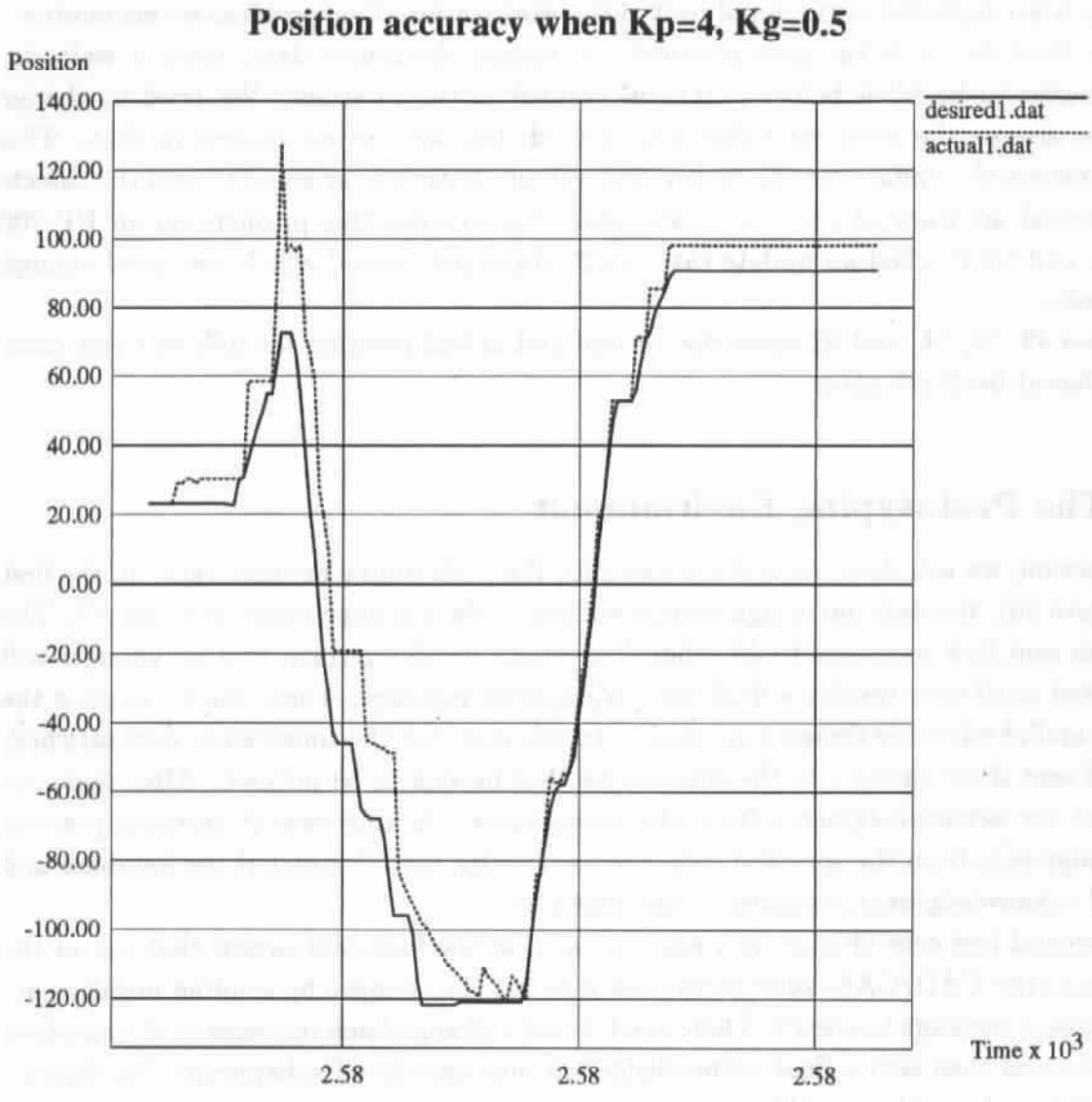


Figure 49: Desired and actual position for test case (1).

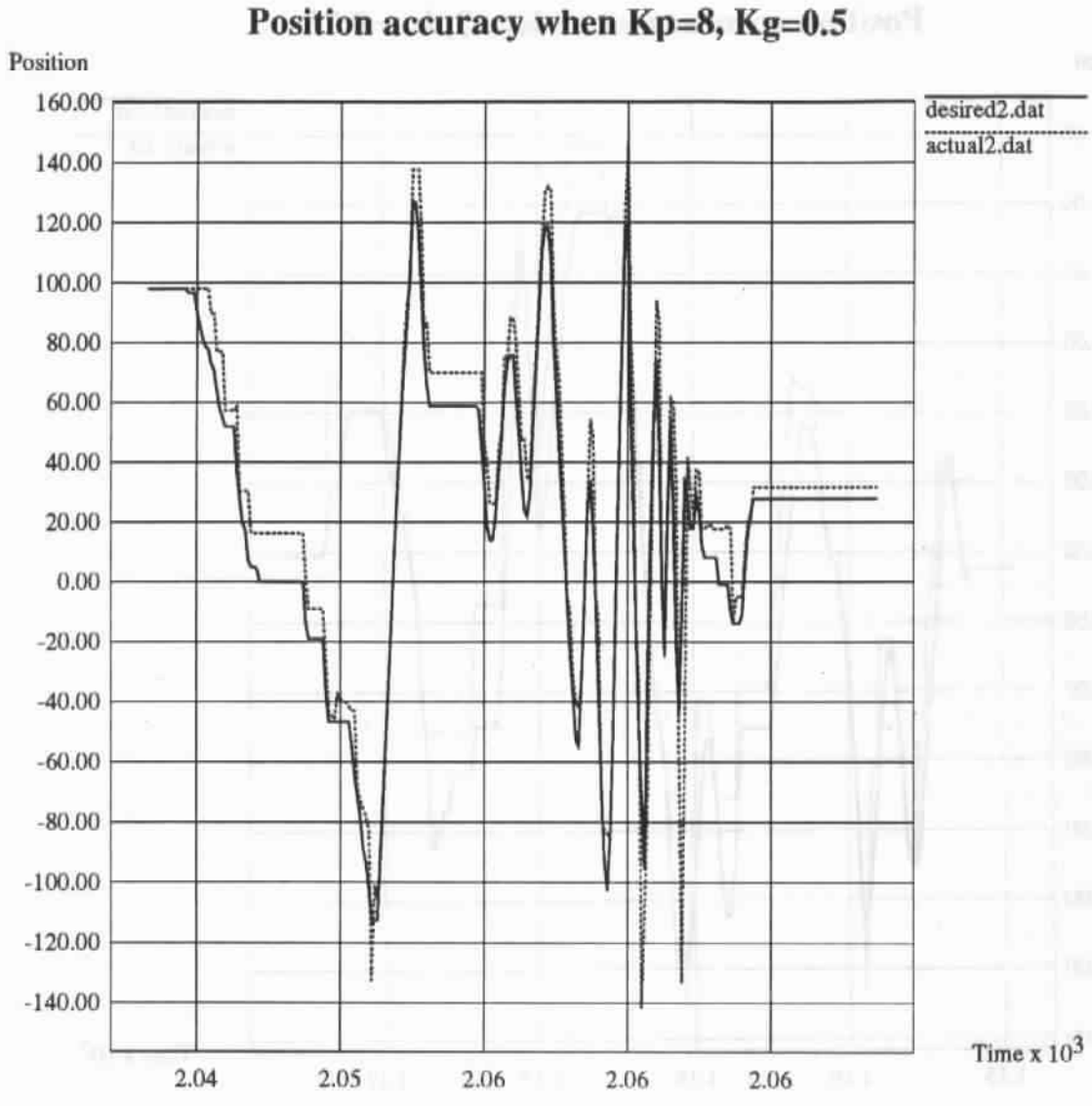


Figure 50: Desired and actual position for test case (2).

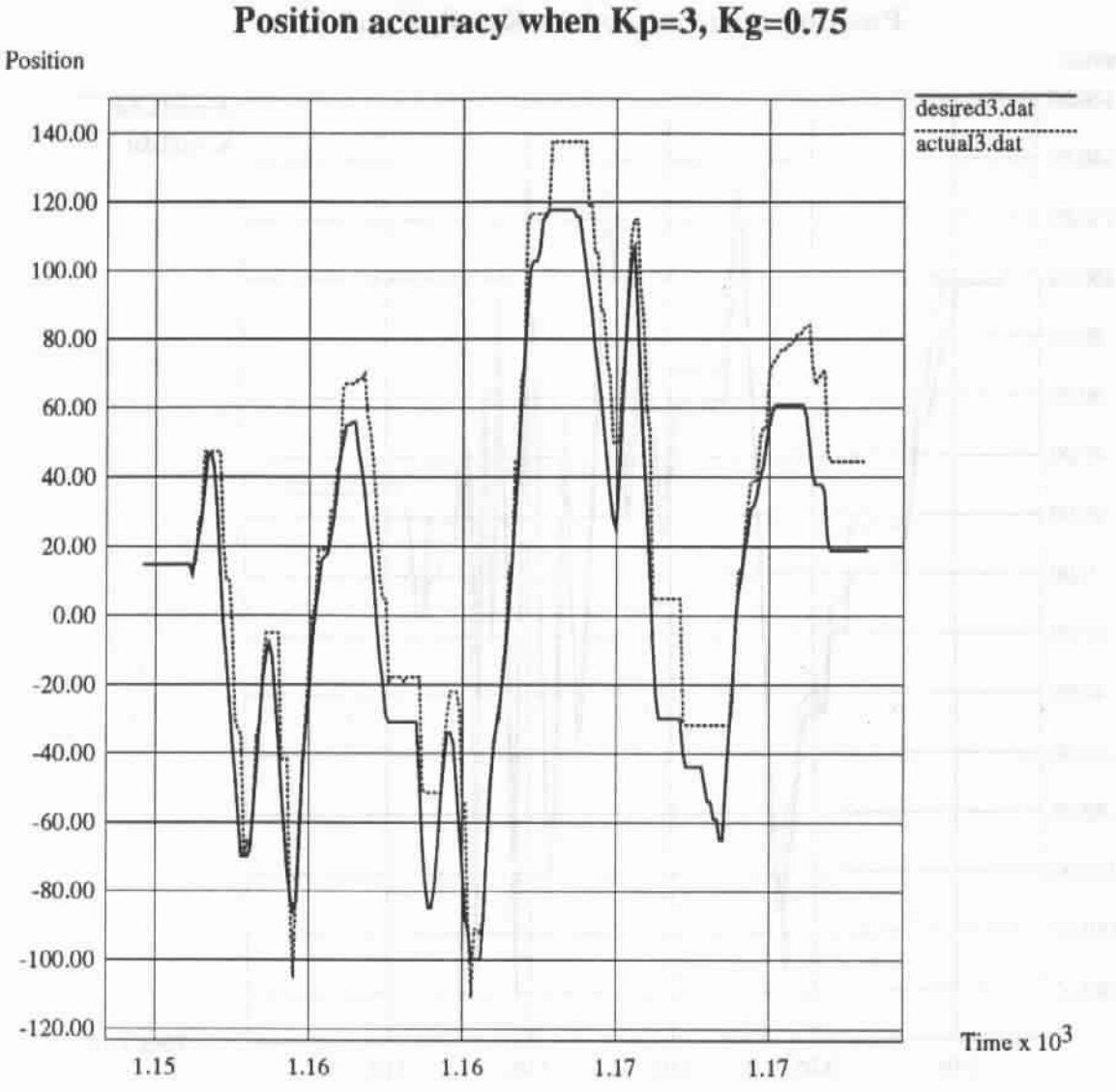


Figure 51: Desired and actual position for test case (3).

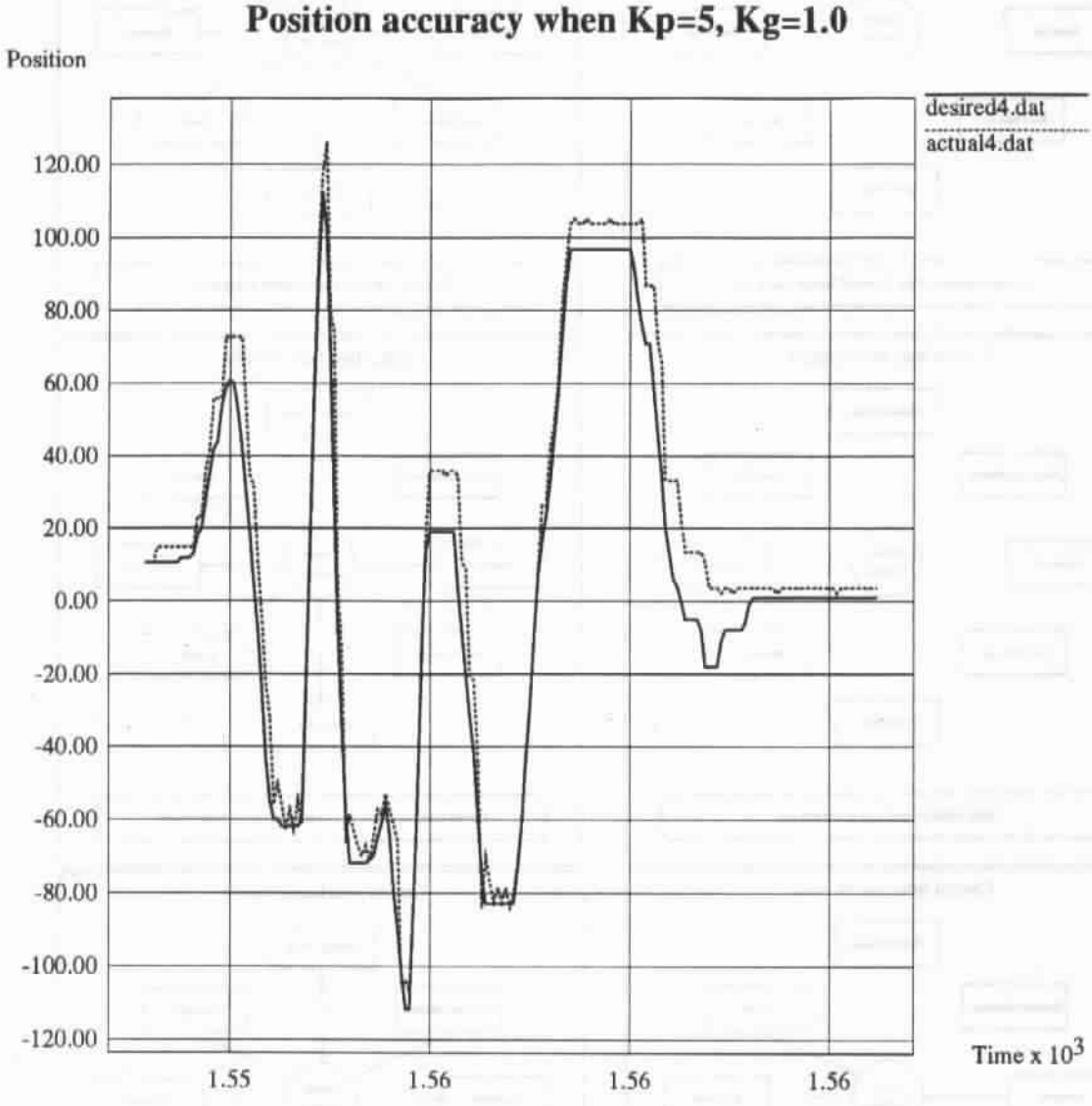


Figure 52: Desired and actual position for test case (4).

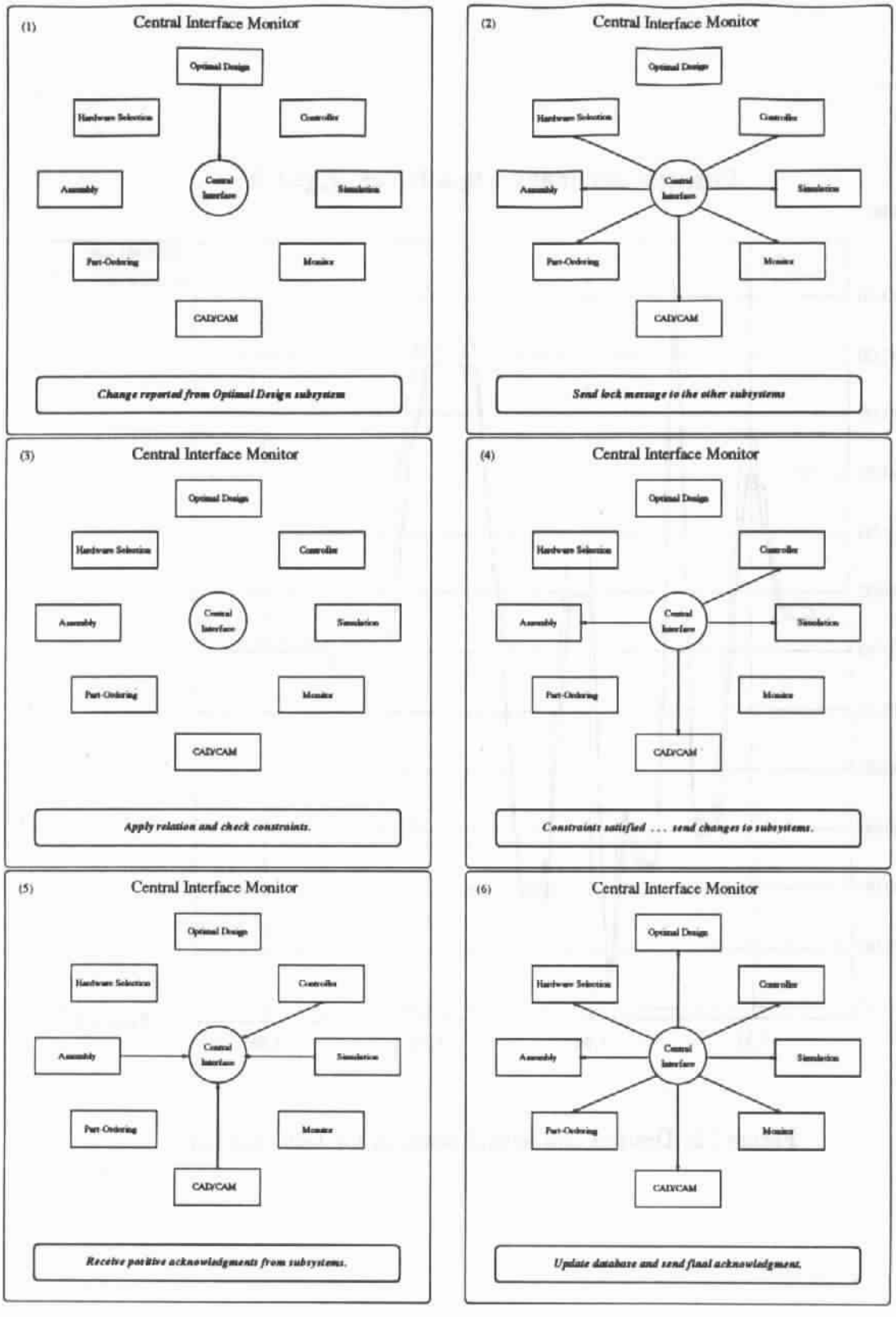


Figure 53: CI test case one, success case for data change.

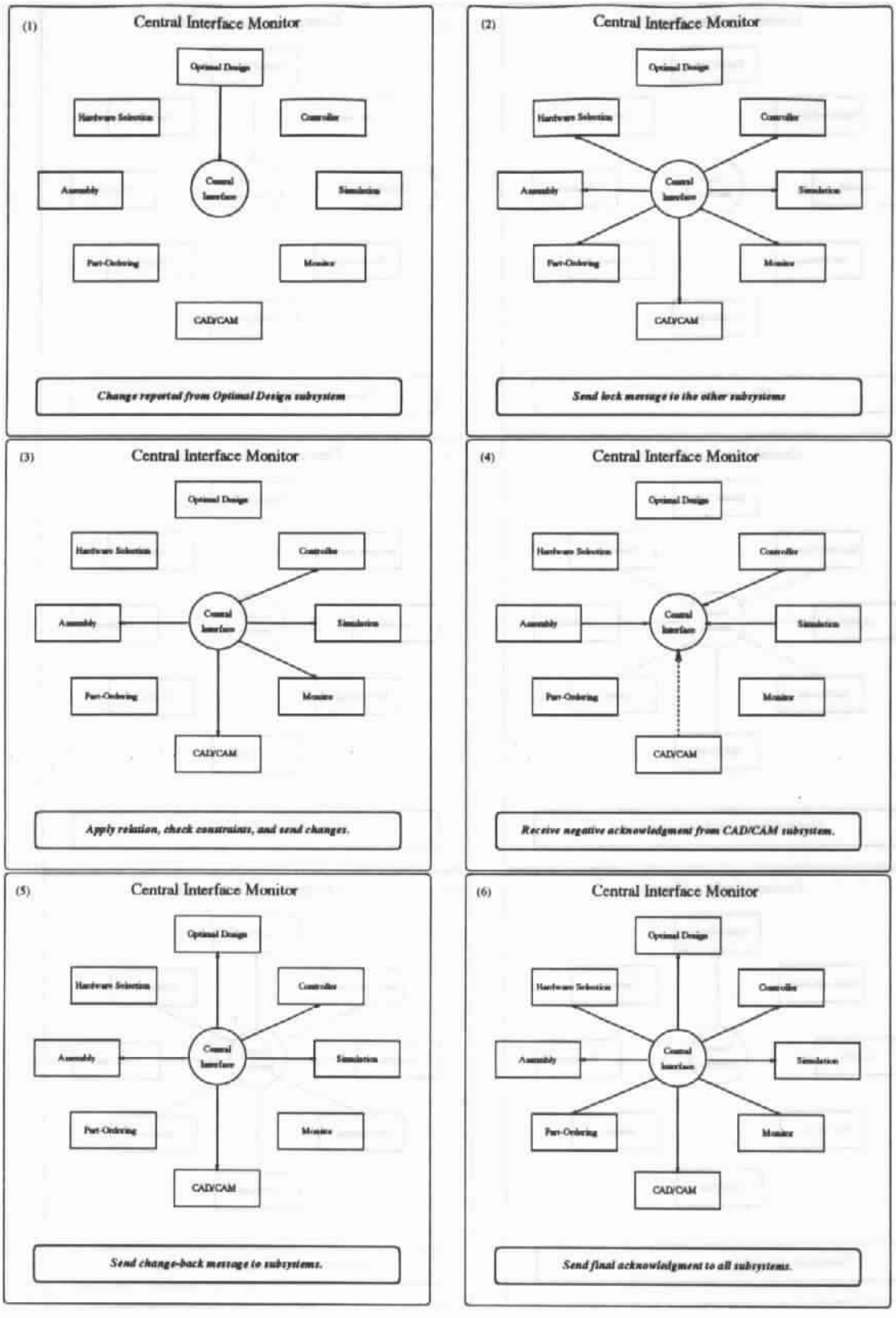


Figure 54: CI test case two, negative acknowledgment case.

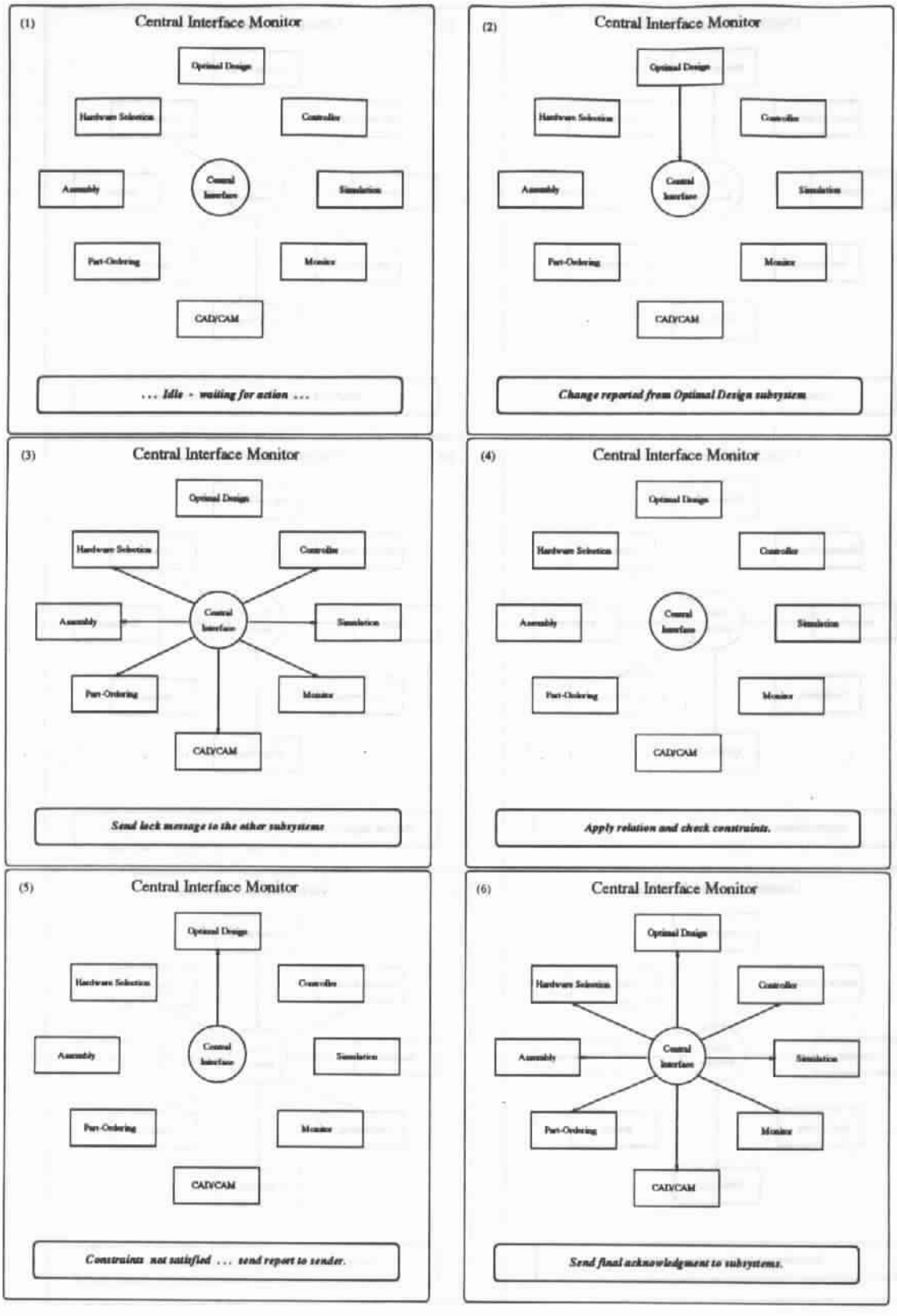


Figure 55: CI test case three, nonsatisfied constraints case.

8.5 Case Study

So far we have been talking about the three-link robot and the prototyping environment (PE) as separate subjects. In this section we will relate them by analyzing the problems that we have faced while designing and building the three-link robot, and how some of these problems could have been avoided if the prototyping environment was used in the design process. We will do that by addressing some of the design problems and what facilities the PE offers to solve some of these problems. Also we will discuss some of the problems that the PE — with its current design — will not be able to solve.

Most of the problems we had were due to the lack of communication between the different groups involved in the design. This lack of communication resulted in data inconsistency among the different groups. One of the problems was changing the mass of the links by the CAD/CAM group without notifying the robotics group. The reason for this change was that the links were too heavy to be driven by small motors. All simulations and benchmarking that were done by the robotics group were based on the original design parameters, and they had to repeat all these tests and simulations after they knew about these changes. The PE can solve this problem since there is an SSI at each subsystem. This SSI will report any changes in the design parameters to the CI, which in turn will report these changes to all subsystems that need to know.

Another problem was selecting the necessary motors to drive the robot links, and satisfy the speed requirements specified by the robotics group. All motors available in the market that can drive the robot links have high rpm. To reduce the speed, gears needed to be used at each joint. Adding these gears caused increases in the weight of each link, and again, the other groups did not know about this change until the assembly process was started. The part-ordering subsystem, suggested in the PE design, can solve this problem by sending a request from the robotics or the CAD/CAM groups to the part-ordering system asking for information about the available motors that satisfy the design requirements. This information would have informed the robotics and CAD/CAM groups about the necessity of adding gears at each joint earlier in the design phase.

The major problem we have faced in this project was the communication between the robot and the workstation. The problem was that the communication rate was too low due to the protocol in the operating system of the Sun Station which waits until a buffer is filled or timeout occurs before it accepts any readings through the serial port. We were able to solve this problem for the HP machine by changing the buffer size to be one byte, but we were not able to do that for the Sun machine. This problem caused the update rate to be as low as 30 Hz. Using The HP-720 we were able to reach an update rate of 120 Hz. However, we used the Sun machine even with its low update rate and we were able to control the three-link robot with an acceptable performance. The results shown in Section 6.3 were generated using a Sun Station-10 model 41, with update rate of 33 Hz.

This problem would not have been avoided even using the PE with its current design, since the PE database does not include detailed information about the platforms. This can be solved by adding more information about the platforms, or by calculating the actual update rate using each platform and put this value as a field in the platforms data file.

Another problem was to select a power amplifier to amplify the signals from the D/A chip to the motors. The power amplifier that we bought was not compatible with the motors we had, and we ended up using some power amplifiers from the ME lab to run our tests. This problem can also be solved using the part-ordering subsystem to select a suitable power amplifier given the motor parameters that we had.

The PE has some limitations with its current implementation. For example, there might be some data inconsistency due to the nonautomated SSIs. Currently, the SSI just informs the user of any change in the design parameters and the user makes the changes in the local files at each subsystem. This process is subject to human error and might yield an inconsistent situation. To solve that, all SSIs need to be automated so that the changes in the local data files of each subsystem are done automatically, and the user at each subsystem is notified of this change.

The automation of the SSIs requires that the subsystems used in the PE should be flexible enough to enable the SSI to make the necessary changes. In other words, it is not possible to make automatic changes if some of the design parameters are hard-wired in the code of the subsystem, because this will require changing the source code (which might not be available), and recompiling the program each time we need to change any of the "hard-wired" parameters. For example, we can not use a simulation subsystem that has a fixed update rate, since we will not be able to study the behavior of the robot under different values for the update rate.

This puts limitations on the subsystems that can be used in the PE. However, most of the "general-purpose" software robotic systems provide an easy way to alter any of the design parameters.

9 Conclusions

A prototype three-link robot manipulator was built to determine the required components for a flexible prototyping environment for electro-mechanical systems in general, and for robot manipulators in particular. A local linear PD feedback law was used for controlling the robot for positioning and trajectory tracking. A graphical user interface was implemented for controlling and simulating the robot. This robot is intended to be an educational tool; therefore it was designed to be easy to install and manipulate. The design process of this robot helped us determine the necessary components for building a prototyping environment for electro-mechanical systems.

- This project establishes the basis and the framework for design automation of robot manipulators in the department.

9.3 Possible Future Extensions

The following are some possible extensions and enhancements to the current design.

- Complete implementation for the central interface with more functionality and a user friendly interface.
- Use a database query language to enable generating more sophisticated queries and to enhance the report generating capabilities.
- Implement some of the subsystems with their SSIs and increase the automation in these interfaces.
- Extend this environment to deal with generic n-link robots by using automatic generation of the kinematics and dynamics equations. Also this will require a robot description language to specify the robot configuration and parameters.
- Implement the PC version of the controller to enable using any PC to control the robot.
- Use special hardware solution to implement some parts of the communication and the control.

References

[1] AHMAD, S. Real-time multi-processor based robot control. In *IEEE Int. Conf. Robotics and Automation* (1986), pp. 858–863.

[2] AHMAD, S., AND LI, B. Optimal design of multiple arithmetic processor-based robot controllers. In *IEEE Int. Conf. Robotics and Automation* (1987), pp. 660–663.

[3] ASADA, H., AND SLOTINE, J. J. E. *Robot Analysis and Control.* J. Wiley and Sons, 1986.

[4] BUKHRES, O. A., CHEN, J., DU, W., AND ELMAGARMID, A. K. Interbase: An execution environment for heterogeneous software systems. *IEEE Computer Magazine* (Aug. 1993), 57–69.

[5] CHEN, Y. Frequency response of discrete-time robot systems - limitations of pd controllers and improvements by lag-lead compensation. In *IEEE Int. Conf. Robotics and Automation* (1987), pp. 464–472.

[6] CHIU, S. L. Kinematic characterization of manipulators: An approach to defining optimality. In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 828–833.

[7] CRAIG, J. *Introduction To Robotics.* Addison-Wesley, 1989.

[8] CUTKOSKY, M. R., ENGELMORE, R. S., FIKES, R. E., GENESERETH, M. R., GRUBER, T. R., MARK, W. S., TENENBAUM, J. M., AND WEBER, J. C. PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer Magazine* (Jan. 1993), 28–37.

[9] DEKHIL, M., SOBH, T. M., AND HENDERSON, T. C. Prototyping environment for robot manipulators. Tech. Rep. UUCS-93-021, University of Utah, Sept. 1993.

[10] DEKHIL, M., SOBH, T. M., AND HENDERSON, T. C. URK: Utah Robot Kit - a 3-link robot manipulator prototype. In *IEEE Int. Conf. Robotics and Automation* (May 1994).

[11] DEPKOVICH, T. M., AND STOUGHTON, R. M. A general approach for manipulator system specification, design, and validation. In *IEEE Int. Conf. Robotics and Automation* (1989), pp. 1402–1407.

[12] DEWAN, P., AND RIEDL, J. Toward computer-supported concurrent software engineering. *IEEE Computer Magazine* (Jan. 1993), 17–27.

[13] DUHOVNIK, J., TAVCAR, J., AND KOPOREC, J. Project manager with quality assurance. *Computer-Aided Design 25*, 5 (May 1993), 311–319.

[14] FUJIOKA, Y., AND KAMEYAMA, M. 2400-mflops reconfigurable parallel VLSI processor for robot control. In *IEEE Int. Conf. Robotics and Automation* (1993), pp. 149–154.

[15] GEFFIN, S., AND FURHT, B. A dataflow multiprocessor system for robot arm control. *Int. J. Robotics Research 9*, 3 (June 1990), 93–103.

[16] GOTTFRIED, B. S., AND WEISMAN, J. *Introduction To Optimization Theory.* Printice-Hall, 1973.

[17] HASHIMOTO, K., AND KIMURA, H. A new parallel algorithm for inverse dynamics. *Int. J. Robotics Research 8*, 1 (Feb. 1989), 63–76.

[18] HERRERA-BENDEZU, L. G., MU, E., AND CAIN, J. T. Symbolic computation of robot manipulator kinematics. In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 993–998.

[19] HOLLERBACH, J. Optimum kinematic design for a seven degree of freedom manipulator. In *Robotics Research: 2nd Int. Symp.* (1985), H. Hanafusa and H. Inous, Eds., MIT Press, pp. 215–222.

[20] IZAGUIRRE, A., HASHIMOTO, M., PAUL, R. P., AND HAYWARD, V. A new computational structure for real-time dynamics. *Int. J. Robotics Research 8*, 1 (Feb. 1989), 346–361.

[21] KAWAMURA, S., MIYAZAKI, F., AND ARIMOTO, S. Is a local linear pd feedback control law effictive for trajectory tracking of robot motion? In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 1335–1340.

[22] KAZANZIDES, P., WASTI, H., AND WOLOVICH, W. A. A multiprocessor system for real-time robotic control: Design and applications. In *IEEE Int. Conf. Robotics and Automation* (1987), pp. 1903–1908.

[23] KELMAR, L., AND KHOSLA, P. K. Automatic generation of forward and inverse kinematics for a reconfigurable manipulator system. *Journal of Robotic Systems 7*, 4 (1990), 599–619.

[24] KHOSLA, P., KANADE, T., HOFFMAN, R., SCHMITZ, D., AND DELOUIS, M. The Carnegie Mellon reconfigurable modular manipulator system project. Tech. rep., Carnegie Mellon University, 1992.

[25] KHOSLA, P. K. Choosing sampling rates for robot control. In *IEEE Int. Conf. Robotics and Automation* (1987), pp. 169–174.

[26] KIRCANSKI, N., PETROVIC, T., AND VUKOBRATOVIC, M. A parallel computer architecture for real-time control applications in grasping and manipulation. In *IEEE Int. Conf. Robotics and Automation* (1993), pp. 410–415.

[27] KUNG, S., AND HWANG, J. Neural network architectures for robot applications. *IEEE Trans. Robotics and Automation 5*, 5 (Oct. 1989), 641–657.

[28] LAMB, D. A. *Software Engineering; Planning for Change.* Prentice Hall, 1988.

[29] LATHROP, R. H. Parallelism in manipulator dynamics. *Int. J. Robotics Research 4*, 2 (1985), 80–102.

[30] LEE, C. S. G., AND CHANG, P. R. Efficient parallel algorithms for robot forward dynamics computation. In *IEEE Int. Conf. Robotics and Automation* (1987), pp. 654–659.

[31] LEUNG, S. S., AND SHANBLATT, M. A. Computer architecture design for robotics. In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 453–456.

[32] LEUNG, S. S., AND SHANBLATT, M. A. A conceptual framework for designing robotic computational hardware with asic technology. In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 461–464.

[33] LEWIS, F. L., ABDALLAH, C. T., AND DAWSON, D. *Control of Robot Manipulator.* Macmillan, 1993.

[34] LI, C., HEMAMI, A., AND SANKAR, T. S. A new computational method for linearized dynamics models for robot manipulators. *Int. J. Robotics Research 9*, 1 (Feb. 1990), 134–146.

[35] LING, Y. L. C., SADAYAPPAN, P., OLSON, K. W., AND ORIN, D. E. A VLSI robotics vector processor for real-time control. In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 303–308.

[36] LIU, C., AND CHEN, Y. Multi-processor-based cartesian-space control techniques for a mechanical manipulator. *IEEE Trans. Robotics and Automation 2*, 2 (June 1986), 110–115.

[37] LUH, J. Y. S., AND LIN, C. S. Scheduling of parallel computation for a computer-controlled mechanical manipulator. *IEEE Trans. Systems Man and Cybernetics 12*, 2 (1984), 214–234.

[38] MA, O., AND ANGELES, J. Optimum design of manipulators under dynamic isotropy conditions. In *IEEE Int. Conf. Robotics and Automation* (1993), pp. 470–475.

[39] MAREFAT, M., MALHORTA, S., AND KASHYAP, R. L. Object-oriented intelligent computer-integrated design, process planning, and inspection. *IEEE Computer Magazine* (Mar. 1993), 54–65.

[40] MAYORGA, R. V., RESSA, B., AND WONG, A. K. C. A kinematic criterion for the design optimization of robot manipulators. In *IEEE Int. Conf. Robotics and Automation* (1991), pp. 578–583.

[41] MAYORGA, R. V., RESSA, B., AND WONG, A. K. C. A kinematic design optimization of robot manipulators. In *IEEE Int. Conf. Robotics and Automation* (1992), pp. 396–401.

[42] MOTOROLA INC. *MC68HC11E9 HCMOS Microcontroller Unit*, 1991.

[43] NICOL, J. R., WILKES, C. T., AND MANOLA, F. A. Object orientation in heterogeneous distributed computing systems. *IEEE Computer Magazine* (June 1993), 57–67.

[44] NIGAM, R., AND LEE, C. S. G. A multiprocessor-based controller for mechanical manipulators. *IEEE Journal of Robotics and Automation 1*, 4 (1985), 173–182.

[45] PAUL, B., AND ROSA, J. Kinematics simulation of serial manipulators. *Int. J. Robotics Research 5*, 2 (Summer 1986), 14–31.

[46] PAUL, R. P. *Robot Manipulators: Mathematics, Programming, and Control.* The MIT Press, 1981.

[47] RIESELER, H., AND WAHL, F. M. Fast symbolic computation of the inverse kinematics of robots. In *IEEE Int. Conf. Robotics and Automation* (1990), pp. 462–467.

[48] SADAYAPPAN, P., LING, Y. C., AND OLSON, K. W. A restructable VLSI robotics vector processor architecture for real-time control. *IEEE Trans. Robotics and Automation 5*, 5 (Oct. 1989), 583–599.

[49] SHILLER, Z., AND SUNDAR, S. Design of robot manipulators for optimal dynamic performance. In *IEEE Int. Conf. Robotics and Automation* (1991), pp. 344–349.

[50] SOBH, T. M., DEKHIL, M., AND HENDERSON, T. C. Prototyping a robot manipulator and controller. Tech. Rep. UUCS-93-013, Univ. of Utah, June 1993.

[51] SRIRAM, D., AND LOGCHER, R. The MIT dice project. *IEEE Computer Magazine* (Jan. 1993), 64–71.

[52] TAKANO, M., MASAKI, H., AND SASAKI, K. Concept of total computer-aided design system of robot manipulators. In *Robotics Research: 3rd Int. Symp.* (1986), pp. 289–296.

[53] TAROKH, M., AND SERAJI, H. A control scheme for trajectory tracking of robot manipulators. In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 1192–1197.

[54] TOOLE, H. *Optimization Methods.* Springer-Verlag, 1975.

[55] WILL, P. Information technology and manufacturing. CSTB/NRC Preliminary Report 1, National Academy Press, Nov. 1993.

[56] ZHANG, H., AND PAUL, R. P. A parallel inverse kinematics solution for robot manipulators based on multiprocessing and linear extrapolation. In *IEEE Int. Conf. Robotics and Automation* (1990), pp. 468–474.

10 Appendix A

The following is the dynamics and kinematics of the three robot models which was generated form the *gendyn* program. These dynamics equations are not simplified.

```
/* Dynamics equations for the first model */

#include <math.h>
#include "dyn1.h"

void dyn1Dyn (M, V, G, F, J_pos, J_vel, B_acc, External_F, External_M)
double **M, *V, *G, *F, *J_pos, *J_vel, *B_acc, *External_F, *External_M;

{
  double external_force_x = External_F[ 0];
  double external_force_y = External_F[ 1];
  double external_force_z = External_F[ 2];
  double external_moment_x = External_M[ 0];
  double external_moment_y = External_M[ 1];
  double external_moment_z = External_M[ 2];
  double base_x = B_acc[ 0];
  double base_y = B_acc[ 1];
  double base_z = B_acc[ 2];
  double T1 = J_pos[ 0];
  double T2 = J_pos[ 1];
  double D3 = J_pos[ 2];
  double vel_T1 = J_vel[ 0];
  double vel_T2 = J_vel[ 1];
  double vel_D3 = J_vel[ 2];
  double sinT1 = sin(T1);
  double cosT1 = cos(T1);
  double sinT2 = sin(T2);
  double cosT2 = cos(T2);

  M[ 0][ 0] = 2 * -JXY * cosT2+90 * sinT2+90
              - 2.0 * -KXZ * cosT2+90 * sinT2+90
              + 0.5*L2 * 0.5*L2 * M2 * cosT2+90 * cosT2+90
              + A2 * A2 * M3 * cosT2+90 * cosT2+90
              + A2 * C3+D3 * M3 * cosT2+90 * sinT2+90
```

```
            + 2 * A2 * C4-0.5*L3 * M3 * cosT2+90 * sinT2+90
            + 2.0 * C3+D3 * C4-0.5*L3 * M3 * sinT2+90 * sinT2+90
            + C4-0.5*L3 * C4-0.5*L3 * M3 * sinT2+90 * sinT2+90 + IZZ
            + JXX * sinT2+90 * sinT2+90 + JYY * cosT2+90 * cosT2+90
            + KXX * sinT2+90 * sinT2+90 + KZZ * cosT2+90 * cosT2+90  ;

M[ 0][ 1] = -JXZ * sinT2+90 + -JYZ * cosT2+90 + -KXY * sinT2+90
            - 1.0 * -KYZ * cosT2+90  ;

M[ 0][ 2] = 0  ;

M[ 1][ 0] = -JXZ * sinT2+90 + -JYZ * cosT2+90 + -KXY * sinT2+90
            - 1.0 * -KYZ * cosT2+90  ;

M[ 1][ 1] = 0.5*L2 * 0.5*L2 * M2 + A2 * A2 * M3
            + 2.0 * C3+D3 * C4-0.5*L3 * M3 + C4-0.5*L3 * C4-0.5*L3 * M3
            + JZZ + KYY  ;

M[ 1][ 2] = -1.0 * A2 * M3  ;

M[ 2][ 0] = 0  ;

M[ 2][ 1] = -1.0 * A2 * M3  ;

M[ 2][ 2] = M3  ;

V[ 0] = 2 * -JXY * cosT2+90 * cosT2+90 * vel_T1 * vel_T2
          - 2 * -JXY * sinT2+90 * sinT2+90 * vel_T1 * vel_T2
          + -JXZ * cosT2+90 * vel_T2 * vel_T2
          - 1 * -JYZ * sinT2+90 * vel_T2 * vel_T2
          + -KXY * cosT2+90 * vel_T2 * vel_T2
          - 2.0 * -KXZ * cosT2+90 * cosT2+90 * vel_T1 * vel_T2
          + 2.0 * -KXZ * sinT2+90 * sinT2+90 * vel_T1 * vel_T2
          + -KYZ * sinT2+90 * vel_T2 * vel_T2
          - 2 * 0.5*L2 * 0.5*L2 * M2 * cosT2+90 * sinT2+90 * vel_T1 * vel_T2
          - 2 * A2 * A2 * M3 * cosT2+90 * sinT2+90 * vel_T1 * vel_T2
          + 2.0 * A2 * C3+D3 * M3 * cosT2+90 * cosT2+90 * vel_T1 * vel_T2
          + 2.0 * A2 * C4-0.5*L3 * M3 * cosT2+90 * cosT2+90 * vel_T1 * vel_T
          - 2 * A2 * C4-0.5*L3 * M3 * sinT2+90 * sinT2+90 * vel_T1 * vel_T2
```

```
          + 2 * A2 * M3 * cosT2+90 * sinT2+90 * vel_D3 * vel_T1
          + 3.0 * C3+D3 * C4-0.5*L3 * M3 * cosT2+90 * sinT2+90 * vel_T1 * ve
          + 2.0 * C4-0.5*L3 * C4-0.5*L3 * M3 * cosT2+90 * sinT2+90 * vel_T1
          + 2 * C4-0.5*L3 * M3 * sinT2+90 * sinT2+90 * vel_D3 * vel_T1
          + 2 * JXX * cosT2+90 * sinT2+90 * vel_T1 * vel_T2
          - 2 * JYY * cosT2+90 * sinT2+90 * vel_T1 * vel_T2
          + 2.0 * KXX * cosT2+90 * sinT2+90 * vel_T1 * vel_T2
          - 2.0 * KZZ * cosT2+90 * sinT2+90 * vel_T1 * vel_T2  ;

V[ 1] = -1 * -JXY * cosT2+90 * cosT2+90 * vel_T1 * vel_T1
          + -JXY * sinT2+90 * sinT2+90 * vel_T1 * vel_T1
          + -KXZ * cosT2+90 * cosT2+90 * vel_T1 * vel_T1
          - 1 * -KXZ * sinT2+90 * sinT2+90 * vel_T1 * vel_T1
          + 0.5*L2 * 0.5*L2 * M2 * cosT2+90 * sinT2+90 * vel_T1 * vel_T1
          + A2 * A2 * M3 * cosT2+90 * sinT2+90 * vel_T1 * vel_T1
          + A2 * C3+D3 * M3 * sinT2+90 * sinT2+90 * vel_T1 * vel_T1
          + A2 * C3+D3 * M3 * vel_T2 * vel_T2
          - 1 * A2 * C4-0.5*L3 * M3 * cosT2+90 * cosT2+90 * vel_T1 * vel_T1
          + A2 * C4-0.5*L3 * M3 * sinT2+90 * sinT2+90 * vel_T1 * vel_T1
          - 1.0 * C3+D3 * C4-0.5*L3 * M3 * cosT2+90 * sinT2+90 * vel_T1 * ve
          - 1.0 * C4-0.5*L3 * C4-0.5*L3 * M3 * cosT2+90 * sinT2+90 * vel_T1
          + 2 * C4-0.5*L3 * M3 * cosT2+90 * vel_D3 * vel_T1
          - 1 * JXX * cosT2+90 * sinT2+90 * vel_T1 * vel_T1
          + JYY * cosT2+90 * sinT2+90 * vel_T1 * vel_T1
          - 1.0 * KXX * cosT2+90 * sinT2+90 * vel_T1 * vel_T1
          + KZZ * cosT2+90 * sinT2+90 * vel_T1 * vel_T1  ;

V[ 2] = -1.0 * A2 * M3 * cosT2+90 * sinT2+90 * vel_T1 * vel_T1
          - 1.0 * C3+D3 * M3 * sinT2+90 * sinT2+90 * vel_T1 * vel_T1
          - 1.0 * C3+D3 * M3 * vel_T2 * vel_T2
          - 1 * C4-0.5*L3 * M3 * sinT2+90 * sinT2+90 * vel_T1 * vel_T1
          - 1 * C4-0.5*L3 * M3 * vel_T2 * vel_T2  ;

G[ 0] = 0.5*L2 * GRAVITY * M2 * cosT1 * cosT2+90
          + A2 * GRAVITY * M3 * cosT1 * cosT2+90
          + C4-0.5*L3 * GRAVITY * M3 * cosT1 * sinT2+90  ;

G[ 1] = -1 * 0.5*L2 * GRAVITY * M2 * sinT1 * sinT2+90
          - 1.0 * A2 * GRAVITY * M3 * sinT1 * sinT2+90
```

```
          + C4-0.5*L3 * GRAVITY * M3 * cosT2+90 * sinT1  ;

  G[ 2] = GRAVITY * M3 * sinT1 * sinT2+90  ;

/* Torque due to external moments */

  F[ 0] = -1.0 * cosT2+90 * external_moment_z
          + external_moment_x * sinT2+90  ;

  F[ 1] = external_moment_y  ;

  F[ 2] = 0  ;

/* Torque due to external forces */

  F[ 0] += -1 * A2 * cosT2+90 * external_force_y
           - 1.0 * C3+D3 * external_force_y * sinT2+90
           - 1 * C4 * external_force_y * sinT2+90  ;

  F[ 1] += -1.0 * A2 * external_force_z + C3+D3 * external_force_x
           + C4 * external_force_x  ;

  F[ 2] += external_force_z  ;

/* Torque due to base movement */

  F[ 0] += 0.5*L2 * M2 * base_y * cosT1 * cosT2+90
           + A2 * M3 * base_y * cosT1 * cosT2+90
           + C4-0.5*L3 * M3 * base_y * cosT1 * sinT2+90  ;

  F[ 1] += -1 * 0.5*L2 * M2 * base_x * cosT1 * sinT2+90
           - 1 * 0.5*L2 * M2 * base_y * sinT1 * sinT2+90
           + 0.5*L2 * M2 * base_z * cosT2+90
           - 1.0 * A2 * M3 * base_x * cosT1 * sinT2+90
           - 1.0 * A2 * M3 * base_y * sinT1 * sinT2+90
           + A2 * M3 * base_z * cosT2+90
           + C4-0.5*L3 * M3 * base_x * cosT1 * cosT2+90
           + C4-0.5*L3 * M3 * base_y * cosT2+90 * sinT1
           + C4-0.5*L3 * M3 * base_z * sinT2+90  ;
```

```
  F[ 2] += M3 * base_x * cosT1 * sinT2+90 + M3 * base_y * sinT1 * sinT2+90

           - 1.0 * M3 * base_z * cosT2+90  ;

/* Force due to friction forces */

  F[ 0] += 0.0  ;
  F[ 1] += 0.0  ;
  F[ 2] += 0.0  ;
}


void dyn1Frm (TransformList, J_pos)
double ***TransformList, *J_pos;

{
  double T1 = J_pos[ 0];
  double T2 = J_pos[ 1];
  double D3 = J_pos[ 2];
  double sinT1 = sin(T1);
  double cosT1 = cos(T1);
  double sinT2 = sin(T2);
  double cosT2 = cos(T2);
  double **T;

  T = TransformList[ 0];
  T[ 0][ 0] = cosT1  ;
  T[ 0][ 1] = -1 * sinT1  ;
  T[ 0][ 2] = 0  ;
  T[ 0][ 3] = 0  ;
  T[ 1][ 0] = sinT1  ;
  T[ 1][ 1] = cosT1  ;
  T[ 1][ 2] = 0  ;
  T[ 1][ 3] = 0  ;
  T[ 2][ 0] = 0  ;
  T[ 2][ 1] = 0  ;
  T[ 2][ 2] = 1  ;
  T[ 2][ 3] = 0  ;
```

```
T[ 3][ 0] = 0  ;
T[ 3][ 1] = 0  ;
T[ 3][ 2] = 0  ;
T[ 3][ 3] = 1  ;

T = TransformList[ 1];
T[ 0][ 0] = cosT1 * cosT2+90  ;
T[ 0][ 1] = -1 * cosT1 * sinT2+90  ;
T[ 0][ 2] = sinT1  ;
T[ 0][ 3] = 0  ;
T[ 1][ 0] = cosT2+90 * sinT1  ;
T[ 1][ 1] = -1 * sinT1 * sinT2+90  ;
T[ 1][ 2] = -1.0 * cosT1  ;
T[ 1][ 3] = 0  ;
T[ 2][ 0] = sinT2+90  ;
T[ 2][ 1] = cosT2+90  ;
T[ 2][ 2] = 0  ;
T[ 2][ 3] = 0  ;
T[ 3][ 0] = 0  ;
T[ 3][ 1] = 0  ;
T[ 3][ 2] = 0  ;
T[ 3][ 3] = 1  ;

T = TransformList[ 2];
T[ 0][ 0] = cosT1 * cosT2+90  ;
T[ 0][ 1] = sinT1  ;
T[ 0][ 2] = cosT1 * sinT2+90  ;
T[ 0][ 3] = A2 * cosT1 * cosT2+90 + C3+D3 * cosT1 * sinT2+90  ;
T[ 1][ 0] = cosT2+90 * sinT1  ;
T[ 1][ 1] = -1.0 * cosT1  ;
T[ 1][ 2] = sinT1 * sinT2+90  ;
T[ 1][ 3] = A2 * cosT2+90 * sinT1 + C3+D3 * sinT1 * sinT2+90  ;
T[ 2][ 0] = sinT2+90  ;
T[ 2][ 1] = 0  ;
T[ 2][ 2] = -1.0 * cosT2+90  ;
T[ 2][ 3] = A2 * sinT2+90 - 1.0 * C3+D3 * cosT2+90  ;
T[ 3][ 0] = 0  ;
T[ 3][ 1] = 0  ;
T[ 3][ 2] = 0  ;
```

```
  T[ 3][ 3] = 1  ;

  T = TransformList[ 3];
  T[ 0][ 0] = cosT1 * cosT2+90  ;
  T[ 0][ 1] = sinT1  ;
  T[ 0][ 2] = cosT1 * sinT2+90  ;
  T[ 0][ 3] = C4 * cosT1 * sinT2+90 + A2 * cosT1 * cosT2+90
              + C3+D3 * cosT1 * sinT2+90  ;

  T[ 1][ 0] = cosT2+90 * sinT1  ;
  T[ 1][ 1] = -1.0 * cosT1  ;
  T[ 1][ 2] = sinT1 * sinT2+90  ;
  T[ 1][ 3] = C4 * sinT1 * sinT2+90 + A2 * cosT2+90 * sinT1
              + C3+D3 * sinT1 * sinT2+90  ;

  T[ 2][ 0] = sinT2+90  ;
  T[ 2][ 1] = 0  ;
  T[ 2][ 2] = -1.0 * cosT2+90  ;
  T[ 2][ 3] = -1.0 * C4 * cosT2+90 + A2 * sinT2+90 - 1.0 * C3+D3 * cosT2+90  ;

  T[ 3][ 0] = 0  ;
  T[ 3][ 1] = 0  ;
  T[ 3][ 2] = 0  ;
  T[ 3][ 3] = 1  ;

}
/*****************************************************************************/

/* Dynamics equations for the second model */

#include <math.h>
#include "dyn2.h"

void dyn2Dyn (M, V, G, F, J_pos, J_vel, B_acc, External_F, External_M)
double **M, *V, *G, *F, *J_pos, *J_vel, *B_acc, *External_F, *External_M;

{
  double external_force_x = External_F[ 0];
  double external_force_y = External_F[ 1];
```

```
double external_force_z = External_F[ 2];
double external_moment_x = External_M[ 0];
double external_moment_y = External_M[ 1];
double external_moment_z = External_M[ 2];
double base_x = B_acc[ 0];
double base_y = B_acc[ 1];
double base_z = B_acc[ 2];
double T1 = J_pos[ 0];
double T2 = J_pos[ 1];
double D3 = J_pos[ 2];
double vel_T1 = J_vel[ 0];
double vel_T2 = J_vel[ 1];
double vel_D3 = J_vel[ 2];
double sinT1 = sin(T1);
double cosT1 = cos(T1);
double sinT2 = sin(T2);
double cosT2 = cos(T2);

M[ 0][ 0] = 2 * -JXY * cosT2 * sinT2 + 2 * -KXZ * cosT2 * sinT2
            + 0.5*L2 * 0.5*L2 * M2 * cosT2 * cosT2
            + A2 * A2 * M3 * cosT2 * cosT2
            - 2.0 * A2 * C4-0.5*L3 * M3 * cosT2 * sinT2
            - 2.0 * A2 * D3 * M3 * cosT2 * sinT2
            + C4-0.5*L3 * C4-0.5*L3 * M3 * sinT2 * sinT2
            + 2.0 * C4-0.5*L3 * D3 * M3 * sinT2 * sinT2
            + D3 * D3 * M3 * sinT2 * sinT2 + IZZ + JXX * sinT2 * sinT2
            + JYY * cosT2 * cosT2 + KXX * sinT2 * sinT2
            + KZZ * cosT2 * cosT2  ;

M[ 0][ 1] = -JXZ * sinT2 + -JYZ * cosT2 - 1.0 * -KXY * sinT2
            - 1.0 * -KYZ * cosT2  ;

M[ 0][ 2] = 0  ;

M[ 1][ 0] = -JXZ * sinT2 + -JYZ * cosT2 - 1.0 * -KXY * sinT2
            - 1.0 * -KYZ * cosT2  ;

M[ 1][ 1] = 0.5*L2 * 0.5*L2 * M2 + A2 * A2 * M3
            + C4-0.5*L3 * C4-0.5*L3 * M3 + 2.0 * C4-0.5*L3 * D3 * M3
```

```
            + D3 * D3 * M3 + JZZ + KYY  ;

M[ 1][ 2] = A2 * M3  ;

M[ 2][ 0] = 0  ;

M[ 2][ 1] = A2 * M3  ;

M[ 2][ 2] = M3  ;

V[ 0] = 2 * -JXY * cosT2 * cosT2 * vel_T1 * vel_T2
          - 2 * -JXY * sinT2 * sinT2 * vel_T1 * vel_T2
          + -JXZ * cosT2 * vel_T2 * vel_T2
          - 1 * -JYZ * sinT2 * vel_T2 * vel_T2
          - 1.0 * -KXY * cosT2 * vel_T2 * vel_T2
          + 2.0 * -KXZ * cosT2 * cosT2 * vel_T1 * vel_T2
          - 2.0 * -KXZ * sinT2 * sinT2 * vel_T1 * vel_T2
          + -KYZ * sinT2 * vel_T2 * vel_T2
          - 2 * 0.5*L2 * 0.5*L2 * M2 * cosT2 * sinT2 * vel_T1 * vel_T2
          - 2.0 * A2 * A2 * M3 * cosT2 * sinT2 * vel_T1 * vel_T2
          - 2.0 * A2 * C4-0.5*L3 * M3 * cosT2 * cosT2 * vel_T1 * vel_T2
          + 2.0 * A2 * C4-0.5*L3 * M3 * sinT2 * sinT2 * vel_T1 * vel_T2
          - 2.0 * A2 * D3 * M3 * cosT2 * cosT2 * vel_T1 * vel_T2
          + 2.0 * A2 * D3 * M3 * sinT2 * sinT2 * vel_T1 * vel_T2
          - 2.0 * A2 * M3 * cosT2 * sinT2 * vel_D3 * vel_T1
          + 2.0 * C4-0.5*L3 * C4-0.5*L3 * M3 * cosT2 * sinT2 * vel_T1 * vel
          + 4.0 * C4-0.5*L3 * D3 * M3 * cosT2 * sinT2 * vel_T1 * vel_T2
          + 2 * C4-0.5*L3 * M3 * sinT2 * sinT2 * vel_D3 * vel_T1
          + 2.0 * D3 * D3 * M3 * cosT2 * sinT2 * vel_T1 * vel_T2
          + 2.0 * D3 * M3 * sinT2 * sinT2 * vel_D3 * vel_T1
          + 2 * JXX * cosT2 * sinT2 * vel_T1 * vel_T2
          - 2 * JYY * cosT2 * sinT2 * vel_T1 * vel_T2
          + 2.0 * KXX * cosT2 * sinT2 * vel_T1 * vel_T2
          - 2.0 * KZZ * cosT2 * sinT2 * vel_T1 * vel_T2  ;

V[ 1] = -1 * -JXY * cosT2 * cosT2 * vel_T1 * vel_T1
          + -JXY * sinT2 * sinT2 * vel_T1 * vel_T1
          - 1.0 * -KXZ * cosT2 * cosT2 * vel_T1 * vel_T1
          + -KXZ * sinT2 * sinT2 * vel_T1 * vel_T1
```

```
            + 0.5*L2 * 0.5*L2 * M2 * cosT2 * sinT2 * vel_T1 * vel_T1
            + A2 * A2 * M3 * cosT2 * sinT2 * vel_T1 * vel_T1
            + A2 * C4-0.5*L3 * M3 * cosT2 * cosT2 * vel_T1 * vel_T1
            - 1 * A2 * C4-0.5*L3 * M3 * sinT2 * sinT2 * vel_T1 * vel_T1
            + A2 * D3 * M3 * cosT2 * cosT2 * vel_T1 * vel_T1
            - 1 * A2 * D3 * M3 * sinT2 * sinT2 * vel_T1 * vel_T1
            - 1.0 * C4-0.5*L3 * C4-0.5*L3 * M3 * cosT2 * sinT2 * vel_T1 * vel
            - 2.0 * C4-0.5*L3 * D3 * M3 * cosT2 * sinT2 * vel_T1 * vel_T1
            - 2.0 * C4-0.5*L3 * M3 * cosT2 * vel_D3 * vel_T1
            - 1 * D3 * D3 * M3 * cosT2 * sinT2 * vel_T1 * vel_T1
            - 2 * D3 * M3 * cosT2 * vel_D3 * vel_T1
            - 1 * JXX * cosT2 * sinT2 * vel_T1 * vel_T1
            + JYY * cosT2 * sinT2 * vel_T1 * vel_T1
            - 1.0 * KXX * cosT2 * sinT2 * vel_T1 * vel_T1
            + KZZ * cosT2 * sinT2 * vel_T1 * vel_T1  ;

  V[ 2] = A2 * M3 * cosT2 * sinT2 * vel_T1 * vel_T1
            - 1 * C4-0.5*L3 * M3 * sinT2 * sinT2 * vel_T1 * vel_T1
            - 1.0 * C4-0.5*L3 * M3 * vel_T2 * vel_T2
            - 1 * D3 * M3 * sinT2 * sinT2 * vel_T1 * vel_T1
            - 1 * D3 * M3 * vel_T2 * vel_T2  ;

  G[ 0] = 0.5*L2 * GRAVITY * M2 * cosT1 * cosT2
            + A2 * GRAVITY * M3 * cosT1 * cosT2
            - 1.0 * C4-0.5*L3 * GRAVITY * M3 * cosT1 * sinT2
            - 1.0 * D3 * GRAVITY * M3 * cosT1 * sinT2  ;

  G[ 1] = -1 * 0.5*L2 * GRAVITY * M2 * sinT1 * sinT2
            - 1 * A2 * GRAVITY * M3 * sinT1 * sinT2
            - 1.0 * C4-0.5*L3 * GRAVITY * M3 * cosT2 * sinT1
            - 1 * D3 * GRAVITY * M3 * cosT2 * sinT1  ;

  G[ 2] = -1 * GRAVITY * M3 * sinT1 * sinT2  ;

/* Torque due to external moments */

  F[ 0] = cosT2 * external_moment_z + external_moment_x * sinT2  ;
  F[ 1] = -1.0 * external_moment_y  ;
  F[ 2] = 0  ;
```

```
/* Torque due to external forces */

  F[ 0] += A2 * cosT2 * external_force_y - 1 * C4 * external_force_y * sinT2
           - 1.0 * D3 * external_force_y * sinT2  ;

  F[ 1] += A2 * external_force_z - 1.0 * C4 * external_force_x
           - 1 * D3 * external_force_x  ;

  F[ 2] += external_force_z  ;

/* Torque due to base movement */

  F[ 0] += 0.5*L2 * M2 * base_y * cosT1 * cosT2
           + A2 * M3 * base_y * cosT1 * cosT2
           - 1.0 * C4-0.5*L3 * M3 * base_y * cosT1 * sinT2
           - 1.0 * D3 * M3 * base_y * cosT1 * sinT2  ;

  F[ 1] += -1 * 0.5*L2 * M2 * base_x * cosT1 * sinT2
           - 1 * 0.5*L2 * M2 * base_y * sinT1 * sinT2
           + 0.5*L2 * M2 * base_z * cosT2
           - 1 * A2 * M3 * base_x * cosT1 * sinT2
           - 1 * A2 * M3 * base_y * sinT1 * sinT2
           + A2 * M3 * base_z * cosT2
           - 1.0 * C4-0.5*L3 * M3 * base_x * cosT1 * cosT2
           - 1.0 * C4-0.5*L3 * M3 * base_y * cosT2 * sinT1
           - 1.0 * C4-0.5*L3 * M3 * base_z * sinT2
           - 1 * D3 * M3 * base_x * cosT1 * cosT2
           - 1 * D3 * M3 * base_y * cosT2 * sinT1
           - 1 * D3 * M3 * base_z * sinT2  ;

  F[ 2] += -1 * M3 * base_x * cosT1 * sinT2
           - 1 * M3 * base_y * sinT1 * sinT2 + M3 * base_z * cosT2  ;

/* Force due to friction forces */

  F[ 0] += 0.0  ;
  F[ 1] += 0.0  ;
  F[ 2] += 0.0  ;
```

```
}


void dyn2Frm (TransformList, J_pos)
double ***TransformList, *J_pos;

{
  double T1 = J_pos[ 0];
  double T2 = J_pos[ 1];
  double D3 = J_pos[ 2];
  double sinT1 = sin(T1);
  double cosT1 = cos(T1);
  double sinT2 = sin(T2);
  double cosT2 = cos(T2);
  double **T;

  T = TransformList[ 0];
  T[ 0][ 0] = cosT1  ;
  T[ 0][ 1] = -1 * sinT1  ;
  T[ 0][ 2] = 0  ;
  T[ 0][ 3] = 0  ;
  T[ 1][ 0] = sinT1  ;
  T[ 1][ 1] = cosT1  ;
  T[ 1][ 2] = 0  ;
  T[ 1][ 3] = 0  ;
  T[ 2][ 0] = 0  ;
  T[ 2][ 1] = 0  ;
  T[ 2][ 2] = 1  ;
  T[ 2][ 3] = 0  ;
  T[ 3][ 0] = 0  ;
  T[ 3][ 1] = 0  ;
  T[ 3][ 2] = 0  ;
  T[ 3][ 3] = 1  ;

  T = TransformList[ 1];
  T[ 0][ 0] = cosT1 * cosT2  ;
  T[ 0][ 1] = -1 * cosT1 * sinT2  ;
  T[ 0][ 2] = sinT1  ;
  T[ 0][ 3] = 0  ;
```

```
T[ 1][ 0] = cosT2 * sinT1  ;
T[ 1][ 1] = -1 * sinT1 * sinT2  ;
T[ 1][ 2] = -1.0 * cosT1  ;
T[ 1][ 3] = 0  ;
T[ 2][ 0] = sinT2  ;
T[ 2][ 1] = cosT2  ;
T[ 2][ 2] = 0  ;
T[ 2][ 3] = 0  ;
T[ 3][ 0] = 0  ;
T[ 3][ 1] = 0  ;
T[ 3][ 2] = 0  ;
T[ 3][ 3] = 1  ;

T = TransformList[ 2];
T[ 0][ 0] = cosT1 * cosT2  ;
T[ 0][ 1] = -1.0 * sinT1  ;
T[ 0][ 2] = -1 * cosT1 * sinT2  ;
T[ 0][ 3] = A2 * cosT1 * cosT2 - 1 * D3 * cosT1 * sinT2  ;
T[ 1][ 0] = cosT2 * sinT1  ;
T[ 1][ 1] = cosT1  ;
T[ 1][ 2] = -1 * sinT1 * sinT2  ;
T[ 1][ 3] = A2 * cosT2 * sinT1 - 1 * D3 * sinT1 * sinT2  ;
T[ 2][ 0] = sinT2  ;
T[ 2][ 1] = 0  ;
T[ 2][ 2] = cosT2  ;
T[ 2][ 3] = A2 * sinT2 + D3 * cosT2  ;
T[ 3][ 0] = 0  ;
T[ 3][ 1] = 0  ;
T[ 3][ 2] = 0  ;
T[ 3][ 3] = 1  ;

T = TransformList[ 3];
T[ 0][ 0] = cosT1 * cosT2  ;
T[ 0][ 1] = -1.0 * sinT1  ;
T[ 0][ 2] = -1 * cosT1 * sinT2  ;
T[ 0][ 3] = -1 * C4 * cosT1 * sinT2 + A2 * cosT1 * cosT2
            - 1 * D3 * cosT1 * sinT2  ;

T[ 1][ 0] = cosT2 * sinT1  ;
```

```
  T[ 1][ 1] = cosT1  ;
  T[ 1][ 2] = -1 * sinT1 * sinT2  ;
  T[ 1][ 3] = -1 * C4 * sinT1 * sinT2 + A2 * cosT2 * sinT1
              - 1 * D3 * sinT1 * sinT2  ;

  T[ 2][ 0] = sinT2  ;
  T[ 2][ 1] = 0  ;
  T[ 2][ 2] = cosT2  ;
  T[ 2][ 3] = C4 * cosT2 + A2 * sinT2 + D3 * cosT2  ;

  T[ 3][ 0] = 0  ;
  T[ 3][ 1] = 0  ;
  T[ 3][ 2] = 0  ;
  T[ 3][ 3] = 1  ;

}
/*****************************************************************************/

/* Dynamics equations for the third model */

#include <math.h>
#include "dyn3.h"

void dyn3Dyn (M, V, G, F, J_pos, J_vel, B_acc, External_F, External_M)
double **M, *V, *G, *F, *J_pos, *J_vel, *B_acc, *External_F, *External_M;

{
  double external_force_x = External_F[ 0];
  double external_force_y = External_F[ 1];
  double external_force_z = External_F[ 2];
  double external_moment_x = External_M[ 0];
  double external_moment_y = External_M[ 1];
  double external_moment_z = External_M[ 2];
  double base_x = B_acc[ 0];
  double base_y = B_acc[ 1];
  double base_z = B_acc[ 2];
  double T1 = J_pos[ 0];
  double T2 = J_pos[ 1];
  double T3 = J_pos[ 2];
```

```
double vel_T1 = J_vel[ 0];
double vel_T2 = J_vel[ 1];
double vel_T3 = J_vel[ 2];
double sinT1 = sin(T1);
double cosT1 = cos(T1);
double sinT2 = sin(T2);
double cosT2 = cos(T2);
double sinT3 = sin(T3);
double cosT3 = cos(T3);

M[ 0][ 0] = 2 * -JXY * cosT2 * sinT2
            + 2 * -KXY * cosT2 * cosT2 * cosT3 * sinT3
            + 2 * -KXY * cosT2 * cosT3 * cosT3 * sinT2
            - 2 * -KXY * cosT2 * sinT2 * sinT3 * sinT3
            - 2 * -KXY * cosT3 * sinT2 * sinT2 * sinT3
            + 0.5*L2 * 0.5*L2 * M2 * cosT2 * cosT2
            + 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT2 * cosT3 * cosT3
            - 1 * 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT3 * sinT2 * sinT3
            + 2 * 0.5*L3 * A2 * M3 * cosT2 * cosT2 * cosT3
            - 1 * 0.5*L3 * A2 * M3 * cosT2 * sinT2 * sinT3
            + A2 * A2 * M3 * cosT2 * cosT2 + IZZ + JXX * sinT2 * sinT2
            + JYY * cosT2 * cosT2 + KXX * cosT2 * cosT2 * sinT3 * sinT3
            + 2 * KXX * cosT2 * cosT3 * sinT2 * sinT3
            + KXX * cosT3 * cosT3 * sinT2 * sinT2
            + KYY * cosT2 * cosT2 * cosT3 * cosT3
            - 2 * KYY * cosT2 * cosT3 * sinT2 * sinT3
            + KYY * sinT2 * sinT2 * sinT3 * sinT3  ;

M[ 0][ 1] = -JXZ * sinT2 + -JYZ * cosT2 + -KXZ * cosT2 * sinT3
            + -KXZ * cosT3 * sinT2 + -KYZ * cosT2 * cosT3
            - 1 * -KYZ * sinT2 * sinT3  ;

M[ 0][ 2] = -KXZ * cosT2 * sinT3 + -KXZ * cosT3 * sinT2
            + -KYZ * cosT2 * cosT3  ;

M[ 1][ 0] = -JXZ * sinT2 + -JYZ * cosT2 + -KXZ * cosT2 * sinT3
            + -KXZ * cosT3 * sinT2 + -KYZ * cosT2 * cosT3
            - 1 * -KYZ * sinT2 * sinT3  ;
```

```
M[ 1][ 1] = 0.5*L2 * 0.5*L2 * M2 + 0.5*L3 * 0.5*L3 * M3
          + 2 * 0.5*L3 * A2 * M3 * cosT3
          + A2 * A2 * M3 * cosT3 * cosT3
          + A2 * A2 * M3 * sinT3 * sinT3 + JZZ + KZZ  ;

M[ 1][ 2] = 0.5*L3 * 0.5*L3 * M3 + 0.5*L3 * A2 * M3 * cosT3 + KZZ  ;

M[ 2][ 0] = -KXZ * cosT2 * sinT3 + -KXZ * cosT3 * sinT2
          + -KYZ * cosT2 * cosT3 - 1 * -KYZ * sinT2 * sinT3  ;

M[ 2][ 1] = 0.5*L3 * 0.5*L3 * M3 + 0.5*L3 * A2 * M3 * cosT3 + KZZ  ;

M[ 2][ 2] = 0.5*L3 * 0.5*L3 * M3 + KZZ  ;

V[ 0] = 2 * -JXY * cosT2 * cosT2 * vel_T1 * vel_T2
        - 2 * -JXY * sinT2 * sinT2 * vel_T1 * vel_T2
        + -JXZ * cosT2 * vel_T2 * vel_T2
        - 1 * -JYZ * sinT2 * vel_T2 * vel_T2
        + 2 * -KXY * cosT2 * cosT2 * cosT3 * cosT3 * vel_T1 * vel_T2
        + 2 * -KXY * cosT2 * cosT2 * cosT3 * cosT3 * vel_T1 * vel_T3
        - 2 * -KXY * cosT2 * cosT2 * sinT3 * sinT3 * vel_T1 * vel_T2
        - 2 * -KXY * cosT2 * cosT2 * sinT3 * sinT3 * vel_T1 * vel_T3
        - 7 * -KXY * cosT2 * cosT3 * sinT2 * sinT3 * vel_T1 * vel_T2
        - 7 * -KXY * cosT2 * cosT3 * sinT2 * sinT3 * vel_T1 * vel_T3
        - 2 * -KXY * cosT3 * cosT3 * sinT2 * sinT2 * vel_T1 * vel_T2
        - 2 * -KXY * cosT3 * cosT3 * sinT2 * sinT2 * vel_T1 * vel_T3
        + -KXY * sinT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T2
        + -KXY * sinT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T3
        - 1 * -KXZ * cosT2 * cosT2 * sinT2 * sinT3 * sinT3 * sinT3 * vel_T
        - 2 * -KXZ * cosT2 * cosT3 * sinT2 * sinT2 * sinT3 * sinT3 * vel_T
        + -KXZ * cosT2 * cosT3 * vel_T2 * vel_T2
        + 2 * -KXZ * cosT2 * cosT3 * vel_T2 * vel_T3
        + -KXZ * cosT2 * cosT3 * vel_T3 * vel_T3
        - 1 * -KXZ * cosT3 * cosT3 * sinT2 * sinT2 * sinT2 * sinT3 * vel_T
        - 1 * -KYZ * cosT2 * cosT2 * cosT3 * sinT2 * sinT3 * sinT3 * vel_T
        - 1 * -KYZ * cosT2 * cosT3 * cosT3 * sinT2 * sinT2 * sinT3 * vel_T
        + -KYZ * cosT2 * sinT2 * sinT2 * sinT3 * sinT3 * sinT3 * vel_T1 *
        - 1 * -KYZ * cosT2 * sinT3 * vel_T2 * vel_T2
        - 2 * -KYZ * cosT2 * sinT3 * vel_T2 * vel_T3
```

```
        - 1 * -KYZ * cosT2 * sinT3 * vel_T3 * vel_T3
        + -KYZ * cosT3 * sinT2 * sinT2 * sinT2 * sinT3 * sinT3 * vel_T1 *
        - 1 * -KYZ * cosT3 * sinT2 * vel_T2 * vel_T2
        - 2 * -KYZ * cosT3 * sinT2 * vel_T2 * vel_T3
        - 1 * -KYZ * cosT3 * sinT2 * vel_T3 * vel_T3
        - 2 * 0.5*L2 * 0.5*L2 * M2 * cosT2 * sinT2 * vel_T1 * vel_T2
        - 2 * 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT2 * cosT3 * sinT3 * vel_T
        - 2 * 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT2 * cosT3 * sinT3 * vel_T
        - 2 * 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT3 * cosT3 * sinT2 * vel_T
        - 2 * 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT3 * cosT3 * sinT2 * vel_T
        - 2 * 0.5*L3 * A2 * M3 * cosT2 * cosT2 * sinT3 * vel_T1 * vel_T2
        - 2 * 0.5*L3 * A2 * M3 * cosT2 * cosT2 * sinT3 * vel_T1 * vel_T3
        - 4 * 0.5*L3 * A2 * M3 * cosT2 * cosT3 * sinT2 * vel_T1 * vel_T2
        - 2 * 0.5*L3 * A2 * M3 * cosT2 * cosT3 * sinT2 * vel_T1 * vel_T3
        - 2 * A2 * A2 * M3 * cosT2 * sinT2 * vel_T1 * vel_T2
        + 2 * JXX * cosT2 * sinT2 * vel_T1 * vel_T2
        - 2 * JYY * cosT2 * sinT2 * vel_T1 * vel_T2
        + 2 * KXX * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 * vel_T2
        + 2 * KXX * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 * vel_T3
        + 2 * KXX * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T2
        + 2 * KXX * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T3
        - 1 * KXX * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T2
        - 1 * KXX * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T3
        - 1 * KXX * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T2
        - 1 * KXX * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T3
        - 2 * KYY * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 * vel_T2
        - 2 * KYY * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 * vel_T3
        - 2 * KYY * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T2
        - 2 * KYY * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T3
        + 2 * KYY * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T2
        + 2 * KYY * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T3
        + 2 * KYY * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T2
        + 2 * KYY * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T3
        - 1 * KZZ * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T2
        - 1 * KZZ * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T3
        - 1 * KZZ * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T2
        - 1 * KZZ * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T3  ;

V[ 1] = -1 * -JXY * cosT2 * cosT2 * vel_T1 * vel_T1
```

```
        + -JXY * sinT2 * sinT2 * vel_T1 * vel_T1
        - 1 * -KXY * cosT2 * cosT2 * cosT3 * cosT3 * vel_T1 * vel_T1
        + -KXY * cosT2 * cosT2 * sinT3 * sinT3 * vel_T1 * vel_T1
        + 4 * -KXY * cosT2 * cosT3 * sinT2 * sinT3 * vel_T1 * vel_T1
        + -KXY * cosT3 * cosT3 * sinT2 * sinT2 * vel_T1 * vel_T1
        - 1 * -KXY * sinT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T1
        + 0.5*L2 * 0.5*L2 * M2 * cosT2 * sinT2 * vel_T1 * vel_T1
        + 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 *
        + 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 *
        - 1 * 0.5*L3 * 0.5*L3 * M3 * cosT2 * sinT2 * sinT3 * sinT3 * vel_T
        - 1 * 0.5*L3 * 0.5*L3 * M3 * cosT3 * sinT2 * sinT2 * sinT3 * vel_T
        + 0.5*L3 * A2 * M3 * cosT2 * cosT2 * sinT3 * vel_T1 * vel_T1
        + 0.5*L3 * A2 * M3 * cosT2 * cosT3 * cosT3 * cosT3 * sinT2 * vel_T
        + 0.5*L3 * A2 * M3 * cosT2 * cosT3 * sinT2 * sinT3 * sinT3 * vel_T
        + 0.5*L3 * A2 * M3 * cosT2 * cosT3 * sinT2 * vel_T1 * vel_T1
        - 1 * 0.5*L3 * A2 * M3 * cosT3 * cosT3 * sinT2 * sinT2 * sinT3 * v
        - 1 * 0.5*L3 * A2 * M3 * sinT2 * sinT2 * sinT3 * sinT3 * sinT3 * v
        - 2 * 0.5*L3 * A2 * M3 * sinT3 * vel_T2 * vel_T3
        - 1 * 0.5*L3 * A2 * M3 * sinT3 * vel_T3 * vel_T3
        + A2 * A2 * M3 * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T1
        + A2 * A2 * M3 * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T1
        - 1 * JXX * cosT2 * sinT2 * vel_T1 * vel_T1
        + JYY * cosT2 * sinT2 * vel_T1 * vel_T1
        - 1 * KXX * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 * vel_T1
        - 1 * KXX * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T1
        + KXX * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T1
        + KXX * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T1
        + KYY * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 * vel_T1
        + KYY * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T1
        - 1 * KYY * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T1
        - 1 * KYY * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T1  ;

V[ 2] = -1 * -KXY * cosT2 * cosT2 * cosT3 * cosT3 * vel_T1 * vel_T1
        + -KXY * cosT2 * cosT2 * sinT3 * sinT3 * vel_T1 * vel_T1
        + 4 * -KXY * cosT2 * cosT3 * sinT2 * sinT3 * vel_T1 * vel_T1
        + -KXY * cosT3 * cosT3 * sinT2 * sinT2 * vel_T1 * vel_T1
        - 1 * -KXY * sinT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T1
        + 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 *
        + 0.5*L3 * 0.5*L3 * M3 * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 *
```

```
          - 1 * 0.5*L3 * 0.5*L3 * M3 * cosT2 * sinT2 * sinT3 * sinT3 * vel_
          - 1 * 0.5*L3 * 0.5*L3 * M3 * cosT3 * sinT2 * sinT2 * sinT3 * vel_
          + 0.5*L3 * A2 * M3 * cosT2 * cosT2 * sinT3 * vel_T1 * vel_T1
          + 0.5*L3 * A2 * M3 * cosT2 * cosT3 * sinT2 * vel_T1 * vel_T1
          + 0.5*L3 * A2 * M3 * sinT3 * vel_T2 * vel_T2
          - 1 * KXX * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 * vel_T1
          - 1 * KXX * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T1
          + KXX * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T1
          + KXX * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T1
          + KYY * cosT2 * cosT2 * cosT3 * sinT3 * vel_T1 * vel_T1
          + KYY * cosT2 * cosT3 * cosT3 * sinT2 * vel_T1 * vel_T1
          - 1 * KYY * cosT2 * sinT2 * sinT3 * sinT3 * vel_T1 * vel_T1
          - 1 * KYY * cosT3 * sinT2 * sinT2 * sinT3 * vel_T1 * vel_T1  ;

  G[ 0] = 0.5*L2 * GRAVITY * M2 * cosT1 * cosT2
          + 0.5*L3 * GRAVITY * M3 * cosT1 * cosT2 * cosT3
          + A2 * GRAVITY * M3 * cosT1 * cosT2  ;

  G[ 1] = -1 * 0.5*L2 * GRAVITY * M2 * sinT1 * sinT2
          - 1 * 0.5*L3 * GRAVITY * M3 * cosT3 * sinT1 * sinT2
          + A2 * GRAVITY * M3 * cosT2 * cosT3 * sinT1 * sinT3
          - 1 * A2 * GRAVITY * M3 * cosT3 * cosT3 * sinT1 * sinT2
          - 1 * A2 * GRAVITY * M3 * sinT1 * sinT2 * sinT3 * sinT3  ;

  G[ 2] = -1 * 0.5*L3 * GRAVITY * M3 * cosT3 * sinT1 * sinT2  ;

/* Torque due to external moments */

  F[ 0] = cosT2 * cosT3 * external_moment_y
          + cosT2 * external_moment_x * sinT3
          + cosT3 * external_moment_x * sinT2  ;

  F[ 1] = external_moment_z  ;

  F[ 2] = external_moment_z  ;

/* Torque due to external forces */

  F[ 0] += -1 * A2 * cosT2 * external_force_z
```

```
          - 1 * A3 * cosT2 * cosT3 * external_force_z  ;

  F[ 1] += A2 * cosT3 * external_force_y + A2 * external_force_x * sinT3
          + A3 * external_force_y  ;

  F[ 2] += A3 * external_force_y  ;

/* Torque due to base movement */

  F[ 0] += 0.5*L2 * M2 * base_y * cosT1 * cosT2
          + 0.5*L3 * M3 * base_y * cosT1 * cosT2 * cosT3
          + A2 * M3 * base_y * cosT1 * cosT2  ;

  F[ 1] += -1 * 0.5*L2 * M2 * base_x * cosT1 * sinT2
          - 1 * 0.5*L2 * M2 * base_y * sinT1 * sinT2
          + 0.5*L2 * M2 * base_z * cosT2
          - 1 * 0.5*L3 * M3 * base_x * cosT1 * cosT3 * sinT2
          - 1 * 0.5*L3 * M3 * base_y * cosT3 * sinT1 * sinT2
          + 0.5*L3 * M3 * base_z * cosT2 * cosT3
          + A2 * M3 * base_x * cosT1 * cosT2 * cosT3 * sinT3
          - 1 * A2 * M3 * base_x * cosT1 * cosT3 * cosT3 * sinT2
          - 1 * A2 * M3 * base_x * cosT1 * sinT2 * sinT3 * sinT3
          + A2 * M3 * base_y * cosT2 * cosT3 * sinT1 * sinT3
          - 1 * A2 * M3 * base_y * cosT3 * cosT3 * sinT1 * sinT2
          - 1 * A2 * M3 * base_y * sinT1 * sinT2 * sinT3 * sinT3
          + A2 * M3 * base_z * cosT2 * cosT3 * cosT3
          + A2 * M3 * base_z * cosT2 * sinT3 * sinT3
          + A2 * M3 * base_z * cosT3 * sinT2 * sinT3  ;

  F[ 2] += -1 * 0.5*L3 * M3 * base_x * cosT1 * cosT3 * sinT2
          - 1 * 0.5*L3 * M3 * base_y * cosT3 * sinT1 * sinT2
          + 0.5*L3 * M3 * base_z * cosT2 * cosT3  ;

/* Force due to friction forces */

  F[ 0] += 0.0  ;
  F[ 1] += 0.0  ;
  F[ 2] += 0.0  ;
}
```

```
void dyn3Frm (TransformList, J_pos)
double ***TransformList, *J_pos;

{
  double T1 = J_pos[ 0];
  double T2 = J_pos[ 1];
  double T3 = J_pos[ 2];
  double sinT1 = sin(T1);
  double cosT1 = cos(T1);
  double sinT2 = sin(T2);
  double cosT2 = cos(T2);
  double sinT3 = sin(T3);
  double cosT3 = cos(T3);
  double **T;

  T = TransformList[ 0];
  T[ 0][ 0] = cosT1  ;
  T[ 0][ 1] = -1 * sinT1  ;
  T[ 0][ 2] = 0  ;
  T[ 0][ 3] = 0  ;
  T[ 1][ 0] = sinT1  ;
  T[ 1][ 1] = cosT1  ;
  T[ 1][ 2] = 0  ;
  T[ 1][ 3] = 0  ;
  T[ 2][ 0] = 0  ;
  T[ 2][ 1] = 0  ;
  T[ 2][ 2] = 1  ;
  T[ 2][ 3] = 0  ;
  T[ 3][ 0] = 0  ;
  T[ 3][ 1] = 0  ;
  T[ 3][ 2] = 0  ;
  T[ 3][ 3] = 1  ;

  T = TransformList[ 1];
  T[ 0][ 0] = cosT1 * cosT2  ;
  T[ 0][ 1] = -1 * cosT1 * sinT2  ;
  T[ 0][ 2] = sinT1  ;
```

```
T[ 0][ 3] = 0  ;
T[ 1][ 0] = cosT2 * sinT1  ;
T[ 1][ 1] = -1 * sinT1 * sinT2  ;
T[ 1][ 2] = -1.0 * cosT1  ;
T[ 1][ 3] = 0  ;
T[ 2][ 0] = sinT2  ;
T[ 2][ 1] = cosT2  ;
T[ 2][ 2] = 0  ;
T[ 2][ 3] = 0  ;
T[ 3][ 0] = 0  ;
T[ 3][ 1] = 0  ;
T[ 3][ 2] = 0  ;
T[ 3][ 3] = 1  ;

T = TransformList[ 2];
T[ 0][ 0] = cosT1 * cosT2 * cosT3 - 1 * cosT1 * sinT2 * sinT3  ;
T[ 0][ 1] = -1 * cosT1 * cosT2 * sinT3 - 1 * cosT1 * cosT3 * sinT2  ;
T[ 0][ 2] = sinT1  ;
T[ 0][ 3] = A2 * cosT1 * cosT2  ;
T[ 1][ 0] = cosT2 * cosT3 * sinT1 - 1 * sinT1 * sinT2 * sinT3  ;
T[ 1][ 1] = -1 * cosT2 * sinT1 * sinT3 - 1 * cosT3 * sinT1 * sinT2  ;
T[ 1][ 2] = -1.0 * cosT1  ;
T[ 1][ 3] = A2 * cosT2 * sinT1  ;
T[ 2][ 0] = cosT3 * sinT2 + cosT2 * sinT3  ;
T[ 2][ 1] = -1 * sinT2 * sinT3 + cosT2 * cosT3  ;
T[ 2][ 2] = 0  ;
T[ 2][ 3] = A2 * sinT2  ;
T[ 3][ 0] = 0  ;
T[ 3][ 1] = 0  ;
T[ 3][ 2] = 0  ;
T[ 3][ 3] = 1  ;

T = TransformList[ 3];
T[ 0][ 0] = cosT1 * cosT2 * cosT3 - 1 * cosT1 * sinT2 * sinT3  ;
T[ 0][ 1] = -1 * cosT1 * cosT2 * sinT3 - 1 * cosT1 * cosT3 * sinT2  ;
T[ 0][ 2] = sinT1  ;
T[ 0][ 3] = A3 * cosT1 * cosT2 * cosT3 - 1 * A3 * cosT1 * sinT2 * sinT3
          + A2 * cosT1 * cosT2  ;
```

```
  T[ 1][ 0] = cosT2 * cosT3 * sinT1 - 1 * sinT1 * sinT2 * sinT3  ;
  T[ 1][ 1] = -1 * cosT2 * sinT1 * sinT3 - 1 * cosT3 * sinT1 * sinT2  ;
  T[ 1][ 2] = -1.0 * cosT1  ;
  T[ 1][ 3] = A3 * cosT2 * cosT3 * sinT1 - 1 * A3 * sinT1 * sinT2 * sinT3
              + A2 * cosT2 * sinT1  ;

  T[ 2][ 0] = cosT3 * sinT2 + cosT2 * sinT3  ;
  T[ 2][ 1] = -1 * sinT2 * sinT3 + cosT2 * cosT3  ;
  T[ 2][ 2] = 0  ;
  T[ 2][ 3] = A3 * cosT3 * sinT2 + A3 * cosT2 * sinT3 + A2 * sinT2  ;

  T[ 3][ 0] = 0  ;
  T[ 3][ 1] = 0  ;
  T[ 3][ 2] = 0  ;
  T[ 3][ 3] = 1  ;

}
```

11 Appendix B

The following are the simplified dynamics using *Mathematica* and also using some manual simplifications for the trig functions.

```
/* Simplified dynamics equations for the first model */


M[1][1] =  2.*K3 + A2*K3 + K4 + 2*A2*K4 + K4*D3 + IZZ - 0.5*K4*L3 +
           JYY*s2*s2 + KZZ*s2*s2 + 0.25*L2*L2*m2*s2*s2 -
           A2*A2*m3*s2*s2 + JXX*c2*c2 + KXX c2*c2  -
           L3*m3*c2 + JXY*sin(2*T2) + 0.5*KXZ*sin(2*T2) -
           0.5*D3*m3*sin(2*T2) + 0.25*L3*m3*sin(2*T2) ;

M[1][2] = -1. + JYZ*s2 + KYZ*s2 - JXZ*c2 - KXY*c2 ;

M[1][3] = 0 ;

M[2][1] =  -1. + JYZ*s2 + KYZ*s2 - JXZ*c2 - KXY*c2 ;

M[2][2] = 2.*K3 + K4 + K4*D3 + JZZ + KYY - 0.5*K4*L3 + 0.25*L2*m2 + A2*m3 - L3

M[2][3] = - A2*m3 ;

M[3][1] = 0 ;

M[3][2] = -A2*m3 ;

M[3][3] = m3 ;

================================================================================

V[1] = -1. + 3.*K3 + 2.A2*K3 + 4.*K4 + K4*D3 - 0.5*K4*L3 -
       0.25*L3*m3*vel_D3*vel_T1 +
       0.25*L3*m3*cos(180+2*T2)*vel_D3*vel_T1 -
       JXY*vel_T1*vel_T2 - KXZ*vel_T1*vel_T2 +
       0.5*D3*m3*vel_T1*vel_T2 - 0.5*L3*m3*vel_T1*vel_T2 +
       0.5*D3*m3*cos(180+2*T2)*vel_T1*vel_T2 +
       JXZ*s2*vel_T2*vel_T2 + KXY*s2*vel_T2*vel_T2 -
```

```
        JYZ*vel_T2*vel_T2*c2 - KYZ*vel_T2*vel_T2*c2 +
        A2*m3*vel_D3*vel_T1*sin(180+2*T2) +
        JXX*vel_T1*vel_T2*sin(180+2*T2) -
        JYY*vel_T1*vel_T2*sin(180+2*T2) +
        KXX*vel_T1*vel_T2*sin(180+2*T2) -
        KZZ*vel_T1*vel_T2*sin(180+2*T2) -
        0.25*L2*L2*m2*vel_T1*vel_T2*sin(180+2*T2) -
        A2*A2*m3*vel_T1*vel_T2*sin(180+2*T2) -
        0.5*L3*m3*vel_T1*vel_T2*sin(180+2*T2) ;

V[2] = -2. - K3 + 2.*A2*K3 + K4 + K4*D3 - 0.5 K4*L3 +
        0.5*L3*m3*s2*vel_D3*vel_T1 - JXY*vel_T1*vel_T1
        KXZ*vel_T1*vel_T1 + 0.5*D3*m3*vel_T1*vel_T1 -
        0.5*L3*m3*vel_T1*vel_T1  -
        0.5*D3*m3*cos(180+2*T2)*vel_T1*vel_T1 + D3*m3*vel_T2*vel_T2 -
        0.5*JXX*vel_T1*vel_T1*sin(180+2*T2) + 0.5*JYY*vel_T1*vel_T1*sin(180+2 T
        0.5*KXX*vel_T1*vel_T1*sin(180+2*T2) + 0.5*KZZ*vel_T1*vel_T1*sin(180+2*T
        0.125*L2*L2*m2*vel_T1*vel_T1*sin(180+2*T2) +
        0.5*A2*A2*m3*vel_T1*vel_T1*sin(180+2*T2) -
        0.5*L3*m3*vel_T1*vel_T1*sin(180+2*T2) ;

V[3] = -2.*K3 - 2.*K4 + D3*m3*vel_T2*vel_T2 - 0.5*L3*m3*vel_T2*vel_T2 +
        D3*m3*vel_T1*vel_T1*c2*c2  -
        0.5*L3*m3*vel_T1*vel_T1*c2*c2 -
        0.5*A2*m3*vel_T1*vel_T1*sin(180+2*T2) ;

================================================================================

G[1] = -2.*K3 - 2.*K4 + D3*m3*vel_T2*vel_T2 - 0.5*L3*m3*vel_T2*vel_T2 +
        D3*m3*vel_T1*vel_T1*c2*c2 -
        0.5*L3*m3*vel_T1*vel_T1*c2*c2 -
        0.5*A2*m3*vel_T1*vel_T1*sin(180+2*T2) ;

G[2] = K4 + 0.5*g*L3*m3*s2*s1 -
        0.5*g*L2*m2*s1*c2 - A2*g*m3*s1*c2 ;

G[3] =  g*m3*s1*c2 ;

================================================================================
```

```
        2.*KZZ*(vel_T1)*(vel_T2)*sin2T2 -
        0.5*L2*L2*m2*(vel_T1)*(vel_T2)*sin2T2 -
        2.*A2*A2*m3*(vel_T1)*(vel_T2)*sin2T2 +
        2.*D3*D3*m3*(vel_T1)*(vel_T2)*sin2T2 -
        0.5*L3*m3*(vel_T1)*(vel_T2)*sin2T2 -
        0.5*D3*L3*m3*(vel_T1)*(vel_T2)*sin2T2) ;

V[2] = 0.5*(-4. - 10.*C4 - C4*L3 - 4.*D3*m3*cosT2*(vel_D3)*(vel_T1) -
        L3*m3*cosT2*(vel_D3)*(vel_T1) - 2.*JXY*(vel_T1)*(vel_T1) -
        2.*KXZ*(vel_T1)*(vel_T1) - L3*m3*(vel_T1)*(vel_T1) +
        2.*A2*D3*m3*cos2T2*(vel_T1)*(vel_T1) - JXX*(vel_T1)*(vel_T1)*sin2T2 +
        JYY*(vel_T1)*(vel_T1)*sin2T2 - KXX*(vel_T1)*(vel_T1)*sin2T2 +
        KZZ*(vel_T1)*(vel_T1)*sin2T2 + 0.25*L2*L2*m2*(vel_T1)*(vel_T1)*sin2T2 +
        A2*A2*m3*(vel_T1)*sin2T2 - D3*D3*m3*(vel_T1)*(vel_T1)*sin2T2 -
        0.5*L3*m3*(vel_T1)*(vel_T1)*sin2T2 -
        0.5*D3*L3*m3*(vel_T1)*(vel_T1)*sin2T2) ;

V[3] = 2.*(-C4 - 0.5*D3*m3*(vel_T2)*(vel_T2) -
        0.25*L3*m3*(vel_T2)*(vel_T2) -
        0.5*D3*m3*(vel_T1)*(vel_T1)*sinT2*sinT2  -
        0.25*L3*m3*(vel_T1)*(vel_T1)*sinT2*sinT2 +
        0.25*A2*m3*(vel_T1)*(vel_T1)*sin2T2) ;

================================================================================

G[1] = -C4 + 0.5*g*L2*m2*cosT1*cosT2 +
        A2*g*m3*cosT1*cosT2 - D3*g*m3*cosT1*sinT2 -
        0.5*g*L3*m3*cosT1*sinT2 ;

G[2] =  -C4 - D3*g*m3*cosT2*sinT1 -
        0.5*g*L3*m3*cosT2*sinT1 -
        0.5*g*L2*m2*sinT1*sinT2 - A2*g*m3*sinT1*sinT2 ;

G[3] =  -g*m3*sinT1*sinT2 ;

================================================================================

/* Simplified dynamics equations for the third model */
```

```
/* Simplified dynamics equations for the second model */

M[1][1] =  4. + 3.*C4 - 2.*A2*C4 + IZZ - 0.5*C4*L3 +
           JYY*cosT2*cosT2  + KZZ*cosT2*cosT2  + 0.25*L2*L2*m2*cosT2*cosT2  +
           A2*A2*m3*cosT2*cosT2 + JXX*sinT2*sinT2 + KXX*sinT2*sinT2  +
           D3*D3*m3*sinT2*sinT2 - 0.5*L3*m3*sinT2*sinT2 - 0.5*D3*L3*m3*sinT2*
           0.5*JXY*sin2T2 - 0.5*KXZ*sin2T2 - A2*D3*m3*sin2T2 -
           0.25*L3*m3*sin2T2) ;

M[1][2] =  -2. - JYZ*cosT2 - KYZ*cosT2 - JXZ*sinT2 - KXY*sinT2 ;

M[1][3] = 0 ;

M[2][1] = -2. - JYZ*cosT2 - KYZ*cosT2 - JXZ*sinT2 - KXY*sinT2 ;

M[2][2] =  0.5*(6.*C4 + 2.*JZZ + 2.*KYY - C4*L3 + 0.5*L2*m2 + 2.*A2*m3 +
           2.*D3*D3*m3 - L3*m3 - D3*L3*m3) ;

M[2][3] = A2*m3 ;

M[3][1] = 0 ;

M[3][2] = A2*m3 ;

M[3][3] = m3 ;

=============================================================================

V[1] = 0.5*(-4. + 16.*C4 - C4*L3 + 2.*D3*m3*(vel_D3)*(vel_T1) -
       0.5*L3*m3*(vel_D3)*(vel_T1) - 2.*D3*m3*cos2T2*(vel_D3)*(vel_T1) +
       0.5*L3*m3*cos2T2*(vel_D3)*(vel_T1) - 2.*JXY*(vel_T1)*(vel_T2) -
       2.*KXZ*(vel_T1)*(vel_T2) - L3*m3*(vel_T1)*(vel_T2) -
       4.*A2*D3*m3*cos2T2*(vel_T1)*(vel_T2) - 2.*JXZ*cosT2*(vel_T2)*(vel_T2)
       2.*KXY*cosT2*(vel_T2)*(vel_T2) - 2.*JYZ*(vel_T2)*(vel_T2)*sinT2 -
       2.*KYZ*(vel_T2)*(vel_T2)*sinT2 - 2.*A2*m3*(vel_D3)*(vel_T1)*sin2T2 +
       2.*JXX*(vel_T1)*(vel_T2)*sin2T2 -
       2.*JYY*(vel_T1)*(vel_T2)*sin2T2 +
       2.*KXX*(vel_T1)*(vel_T2)*sin2T2 -
```

```
M[1][1] =  2 + IZZ + (JXX+JYY+KXX+KYY)/2.0 + 0.125*L2*L2*m2 +
           A2*A2*m3/2.0 + 0.0625*L3*L3*m3 -
           (JXX*cos2T2+JYY*cos2T2)/2.0 + 0.125*L2*L2*m2*cos2T2 +
           A2*A2*m3*cos2T2/2.0 + 0.0625*L3*L3*m3*cos2T2 +
           0.125*A2*L3*m3*cos2T2mT3 + 0.5*A2*L3*m3*cosT3 +
           0.0625*L3*L3*m3*cos2T3 - (KXX cos2_T2pT3 +
           KYY cos2_T2pT3)/2.0 + 0.0625*L3*L3*m3*cos2_T2pT3 +
           0.375*A2*L3*m3*cos2T2pT3 -
           (JXY*sin2T2 - KXY*sin2T2 - KXY*sin2T3)/2.0 ;

M[1][2] =  -1 - JYZ*cosT2 - KYZ*cosT2mT3 - JXZ*sinT2 - KXZ*sinT2pT3 ;

M[1][3] = -(KYZ*cosT2*cosT3) - KXZ*cosT3*sinT2 - KXZ*cosT2*sinT3 ;

M[2][1] = -1 - JYZ*cosT2 - KYZ*cosT2mT3 - JXZ*sinT2 - KXZ*sinT2pT3

M[2][2] = JZZ + KZZ + 0.25*L2*L2*m2 + A2*A2*m3 + 0.25*L3*L3*m3 + *A2*L3*m3*co

M[2][3] = KZZ + 0.25*L3*L3*m3 + 0.5*A2*L3*m3*cosT3 ;

M[3][1] = -1 - KYZ*cosT2mT3 - KXZ*sinT2pT3 ;

M[3][2] = JZZ + KZZ + 0.25*L2*L2*m2 + A2*A2*m3 + 0.25*L3*L3*m3 + A2*L3*m3*cos

M[3][3] = KZZ + 0.25*L3*L3*m3 ;

/* ============================================================================


V[1] = (-992 - KXZ*cosT2m3T3*theta_dot[1]*theta_dot[1] -
       7*KXZ*cosT2mT3*theta_dot[1]*theta_dot[1] +
       KXZ*cos3_T2mT3*theta_dot[1]*theta_dot[1] -
       KXZ*cos3T2mT3*theta_dot[1]*theta_dot[1] +
       5*KXZ*cosT2pT3*theta_dot[1]*theta_dot[1] -
       3*KXZ*cos3_T2pT3*theta_dot[1]*theta_dot[1] +
       3*KXZ*cos3T3pT3*theta_dot[1]*theta_dot[1]  +
       3*KXZ*cosT2p3T3*theta_dot[1]*theta_dot[1] -
       32*JXY*theta_dot[1]*theta_dot[2] -
       32*KXY*theta_dot[1]*theta_dot[2] -
```

```
4*KXY*cos2_T2mT3*theta_dot[1]*theta_dot[2] +
4*KXY*cos2_T2pT3*theta_dot[1]*theta_dot[2] -
32*JXZ*cosT2*theta_dot[2]*theta_dot[2] -
16*KXZ*cosT2mT3*theta_dot[2]*theta_dot[2] -
16*KXZ*cosT2pT3*theta_dot[2]*theta_dot[2] -
32*KXY*theta_dot[1]*theta_dot[3] -
4*KXY*cos2_T2mT3*theta_dot[1]*theta_dot[3] +
4*KXY*cos2_T2pT3*theta_dot[1]*theta_dot[3] -
16*KXZ*cosT2mT3*theta_dot[2]*theta_dot[3] -
16*KXZ*cosT2pT3*theta_dot[2]*theta_dot[3] -
16*KXZ*cosT2mT3*theta_dot[3]*theta_dot[3] -
16*KXZ*cosT2pT3*theta_dot[3]*theta_dot[3] -
32*JYZ*theta_dot[2]*theta_dot[2]*sinT2 +
32*JXX*theta_dot[1]*theta_dot[2]*sin2T2 -
32*JYY*theta_dot[1]*theta_dot[2]*sin2T2 +
8*KXX*theta_dot[1]*theta_dot[2]*sin2T2 -
8*KZZ*theta_dot[1]*theta_dot[2]*sin2T2 -
8.*L2*L2*m2*theta_dot[1]*theta_dot[2]*sin2T2 -
32*A2*A2*m3*theta_dot[1]*theta_dot[2]*sin2T2 -
4.*L3*L3*m3*theta_dot[1]*theta_dot[2]*sin2T2 +
8*KXX*theta_dot[1]*theta_dot[3]*sin2T2 -
8*KZZ*theta_dot[1]*theta_dot[3]*sin2T2 -
4.*L3*L3*m3*theta_dot[1]*theta_dot[3]*sin2T2 +
4*KYZ*theta_dot[1]*theta_dot[1]*sinT2m3T3 -
8.*A2*L3*m3*theta_dot[1]*theta_dot[2]*sin2T2mT3 -
4*KYZ*theta_dot[1]*theta_dot[1]*sin3T2mT3 -
16.*A2*L3*m3*theta_dot[1]*theta_dot[2]*sinT3 -
16.*A2*L3*m3*theta_dot[1]*theta_dot[3]*sinT3 +
8*KXX*theta_dot[1]*theta_dot[2]*sin2T3 -
8*KZZ*theta_dot[1]*theta_dot[2]*sin2T3 -
4.*L3*L3*m3*theta_dot[1]*theta_dot[2]*sin2T3 +
8*KXX*theta_dot[1]*theta_dot[3]*sin2T3 -
8*KZZ*theta_dot[1]*theta_dot[3]*sin2T3 -
4.*L3*L3*m3*theta_dot[1]*theta_dot[3]*sin2T3 -
8*KYZ*theta_dot[1]*theta_dot[1]*sinT2pT3 -
32*KYZ*theta_dot[2]*theta_dot[2]*sinT2pT3 -
32*KYZ*theta_dot[2]*theta_dot[3]*sinT2pT3 -
32*KYZ*theta_dot[3]*theta_dot[3]*sinT2pT3 +
24*KXX*theta_dot[1]*theta_dot[2]*sin2_T2pT3 -
```

```
        32*KYY*theta_dot[1]*theta_dot[2]*sin2_T2pT3 +
        8*KZZ*theta_dot[1]*theta_dot[2]*sin2_T2pT3 -
        4.*L3*L3*m3*theta_dot[1]*theta_dot[2]*sin2_T2pT3 +
        24*KXX*theta_dot[1]*theta_dot[3]*sin2_T2pT3 -
        32*KYY*theta_dot[1]*theta_dot[3]*sin2_T2pT3 +
        8*KZZ*theta_dot[1]*theta_dot[3]*sin2_T2pT3 -
        4.*L3*L3*m3*theta_dot[1]*theta_dot[3]*sin2_T2pT3 -
        24.*A2*L3*m3*theta_dot[1]*theta_dot[2]*sin2T2pT3 -
        16.*A2*L3*m3*theta_dot[1]*theta_dot[3]*sin2T2pT3 +
        4*KYZ*theta_dot[1]*theta_dot[1]*sin3T3pT3 +
        4*KYZ*theta_dot[1]*theta_dot[1]*sinT2p3T3) / 32 ;

V[2] = 1 - JXY*theta_dot[1]*theta_dot[1] -
        KXY*theta_dot[1]*theta_dot[1] -
        (KXY*cos2_T2mT3*theta_dot[1]*theta_dot[1])/8 +
        KXY*cos2_T2pT3*theta_dot[1]*theta_dot[1]/8 -
        JXX*theta_dot[1]*theta_dot[1]*sin2T2/2
        JYY*theta_dot[1]*theta_dot[1]*sin2T2/2 +
        0.125 L2*L2  m2 (theta_dot[1]*theta_dot[1]*sin2T2 +
        A2*A2*m3*theta_dot[1]*theta_dot[1]*sin2T2/2 -
        A2*L3*m3*theta_dot[2]*theta_dot[3]*sinT3 -
        0.5*A2*L3*m3*theta_dot[3]*theta_dot[3]*sinT3 -
        KXX*theta_dot[1]*theta_dot[1]*sin2_T2pT3/2 +
        KYY*theta_dot[1]*theta_dot[1]*sin2_T2pT3/2 +
        0.125*L3*L3*m3*theta_dot[1]*theta_dot[1]*sin2_T2pT3 +
        0.5*A2*L3*m3*theta_dot[1]*theta_dot[1]*sin2T2pT3 ;

V[3] = (16 - 8*KXY*theta_dot[1]*theta_dot[1] -
         KXY*cos2_T2mT3*theta_dot[1]*theta_dot[1]  +
         KXY*cos2_T2pT3*theta_dot[1]*theta_dot[1] +
         2.*A2*L3*m3*theta_dot[1]*theta_dot[1]*sinT3 +
         4.*A2*L3*m3*theta_dot[2]*theta_dot[2]*sinT3 -
         4*KXX*theta_dot[1]*theta_dot[1]*sin2_T2pT3 +
         4*KYY*theta_dot[1]*theta_dot[1]*sin2_T2pT3 +
         L3*L3*m3*theta_dot[1]*theta_dot[1]*sin2_T2pT3 +
         2.*A2*L3*m3*theta_dot[1]*theta_dot[1]*sin2T2pT3) / 8 ;

================================================================================
```

```
G[1] = 0.5*(g*L2*m2*cosT1*cosT2 +
       2.*A2*g*m3*cosT1*cosT2 +
       g*L3*m3*cosT1*cosT2*cosT3) ;

G[2] = 0.5*(-1. g*L2*m2*sinT1*sinT2 -
       g*L3*m3*cosT3*sinT1*sinT2 -
       2.*A2*g*m3*cosT3*cosT3*sinT1*sinT2 -
       2.*A2*g*m3*sinT1*sinT2*sinT3*sinT3 +
       A2*g*m3*cosT2*sinT1*sin2T3) ;

G[3] = -0.5*g*L3*m3*cosT3*sinT1*sinT2 ;

================================================================================
```

12 Appendix C

The assembly program used for the chip is described bellow. The comments explain what the program does at each step.

```
* USING OWN COMMUNICATION ROUTINES *
* RECEIVES 3 BYTES FROM COMPUTER AND SENDS OUT TO PORT B AND CONVERTS
* 3 BYTES TO DIGITAL AND SENDS TO COMPUTER *
* PORT B IS USED TO OUTPUT THE 8 BIT DATA TO BE CONVERTED
* TO ANALOG ALONG WITH 3 CONTROL BITS OF PORTC-0,1,2.
ORG     $B600
PORTA   EQU     $1000
PORTB   EQU     $1004
PORTC   EQU     $1003
DDRC    EQU     $1007
IODEV   EQU     $00A7     VALUE TO SPECIFY PORT TO BE USED
AUTOLF  EQU     $00A6     AUTO LF/CR CONTROL
INIT    EQU     $FFA9     USEFUL SUBROUTINES
ADCTL   EQU     $1030     CONTROL FOR A/D CONVERSION
ADR1    EQU     $1031     A/D RESULT REGISTERS
ADR2    EQU     $1032
ADR3    EQU     $1033
ADR4    EQU     $1034
OPTION  EQU     $1039     TO POWER UP THE A/D SYSTEM
INTMP1  EQU     $0101
INTMP2  EQU     $0102
INTMP3  EQU     $0103
INTMP4  EQU     $0104
OUTMP1  EQU     $0111
OUTMP2  EQU     $0112
OUTMP3  EQU     $0113
OUTMP4  EQU     $0114
BAUD    EQU     $102B
SCCR1   EQU     $102C
SCCR2   EQU     $102D
SCSR    EQU     $102E
SCDAT   EQU     $102F
COPRST  EQU     $103A
* MAIN PROGRAM THAT INITIALIZES AND ROUTES CONTROL *
```

```
MAIN    CLR     AUTOLF
INC     AUTOLF
LDAA    #$00
STAA    IODEV
LDAA    #$FF
STAA    DDRC            SET PORTC AS OUTPUT PORT
LDAA    #$00
STAA    PORTC           SELECTING ALL 3 d/a's
LDAA    #$80
STAA    PORTB           INITIALIZE d/a's TO 0V
JSR     INIT
LDAA    #$FF
STAA    PORTC           AFTER A DELAY, DESELECTING ALL d/a's
LDAA    #$41
STAA    INTMP2
LDAA    #$42
STAA    INTMP3
LDAA    #$43
STAA    INTMP4
JSR     INITAD
JSR     CHKST
JSR     TXDAT           first 3 bytes test characters
MLOOP   JSR     RXDAT
JSR     GETANL
JSR     OUTDAT
JMP     MLOOP
* INPUT SUBROUTINE  *
INPUT   PSHX
LDAA    #$55
STAA    COPRST
LDAA    #$AA
STAA    COPRST
LDAA    IODEV
BNE     INPUT1
LDAA    SCDAT
INSCI   LDAA    SCSR
ANDA    #$20
BEQ     INPUT1
LDAA    SCDAT
```

```
INPUT1  PULX
RTS
* OUTPUT SUBROUTINE *
OUTPUT  PSHA
PSHB
PSHX
LDAB    IODEV
BNE     OUTPUT1
OUTSCI  LDAB    SCSR
BITB    #$80
BEQ     OUTSCI          LOOP TILL TRDRE=1
STAA    SCDAT
OUTPUT1 PULX
PULB
PULA
RTS
* SUBROUTINE THAT WAITS FOR 3 CHARS FROM WORKSTATION *
CHKST   JSR     INPUT
TSTA
BEQ     CHKST           CHECK IF ANY CHARACTER RECEIVED
CLOOP1  JSR     INPUT
TSTA
BEQ     CLOOP1          CHECK FOR SECOND BYTE
CLOOP2  JSR     INPUT
TSTA
BEQ     CLOOP2          CHECK FOR THIRD BYTE
RTS
* SUBROUTINE TO INITIALIZE A/D CONVERION UNIT *
INITAD  LDAA    #$80
STAA    OPTION          SET ENABLE BIT FOR THE A/D SYS.
LDAA    #$30
STAA    ADCTL           START CONTINIOUS CONVERSION OF 4 INPUTS
ALOOP   LDAA    ADCTL
ANDA    #$80
BEQ     ALOOP           CHK IF ONE CONVERSION COMPLETE
RTS
* GET ANALOG INPUTS AND OUTPUT THEM TO COMPUTER *
GETANL  LDAA    ADR1            FIRST BYTE STRAY INPUT
LDAA    ADR2
```

```
JSR     OUTPUT
LDAA    ADR3
JSR     OUTPUT
LDAA    ADR4
JSR     OUTPUT
RTS
* SUBROUTINE TO TRANSMIT 3 BYTES OF DATA FROM INTMP REGISTERS *
TXDAT   LDAA    INTMP2
JSR     OUTPUT
LDAA    INTMP3
JSR     OUTPUT
LDAA    INTMP4
JSR     OUTPUT
RTS
* SUBROUTINE TO RECEIVE 3 BYTES OF DATA FROM WORKSTATION *
RXDAT   JSR     INPUT
TSTA
BEQ     RXDAT     IF NO CHR RECEIVED GO BACK
STAA    OUTMP1    IF YES, STORE
RLOOP1  JSR     INPUT
TSTA
BEQ     RLOOP1
STAA    OUTMP2
RLOOP2  JSR     INPUT
TSTA
BEQ     RLOOP2
STAA    OUTMP3
RTS
* SUBROUTINE TO SEND OUT THE DATA TO THE 3 D/A'S *
* ALSO HAVE TO OUTPUT CONTROL ON PORT A TO CHOOSE 1 D/A AT EACH TIME *
OUTDAT  LDAA    OUTMP1
STAA    PORTB
LDAA    #$FE
STAA    PORTC           SELECT FIRST d/a.
NOP
NOP
NOP
LDAA    #$FF
STAA    PORTC           AFTER A DELAY DISABLE
```

```
LDAA    OUTMP2
STAA    PORTB
LDAA    #$FD
STAA    PORTC           SELECT SECOND d/a.
NOP
NOP
NOP
LDAA    #$FF
STAA    PORTC           AFTER A DELAY DISABLE
LDAA    OUTMP3
STAA    PORTB
LDAA    #$FB
STAA    PORTC           SELECT THIRD d/a.
NOP
NOP
NOP
LDAA    #$FF
STAA    PORTC           AFTER A DELAY DISABLE
RTS
```