Robotic Prototyping Environment
(Progress report)

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

UUSC-94-004

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

February 3, 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 sub-systems 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 sub-systems. Some of these sub-systems are:

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

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

This report starts with a brief background of robot design and modules is presented in Section 2 with the related work in this area. A detailed description of prototyping and simulating a 3-link robot manipulator is presented in Section 3. Section 4 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 5 shows some examples and results of the implemented systems. Finally, conclusions from the work are presented in Section 6 along with possible future extensions.

2 Background and Related Work

2.1 Phases of Building a Robot

We can divide the process of building a robot 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, where each module accomplishes a certain task. The most important modules, as described in [6], 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 [6] for more details.

One approach to the problem of kinematics analysis is described in [41], 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 [16]. 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 [43]. Kelmar and Khosla proposed a method for automatic generation of forward and inverse kinematics for a reconfigurable manipulator system [20].

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 [33], 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 [25, 26, 40] based on a multiprocessor controller, and pipelined architectures to speed the calculations. Hashimoto and Kimura [15] 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 [31] 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 [18]. 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 only need to specify the goal position, while 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 [6]).

2.3 Linear Feedback Control

We will use a linear control system in our design, which is an approximation of the non-linear nature of the dynamics equations of the system, which are more properly represented by non-linear 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 [2, 30, 42].

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.
- No need to identify the robot parameters.
- Suitable for real-time control since it has very few computations compared to the complicated non-linear dynamic equations.
- The behavior of the system can be controlled by changing the feedback gains.

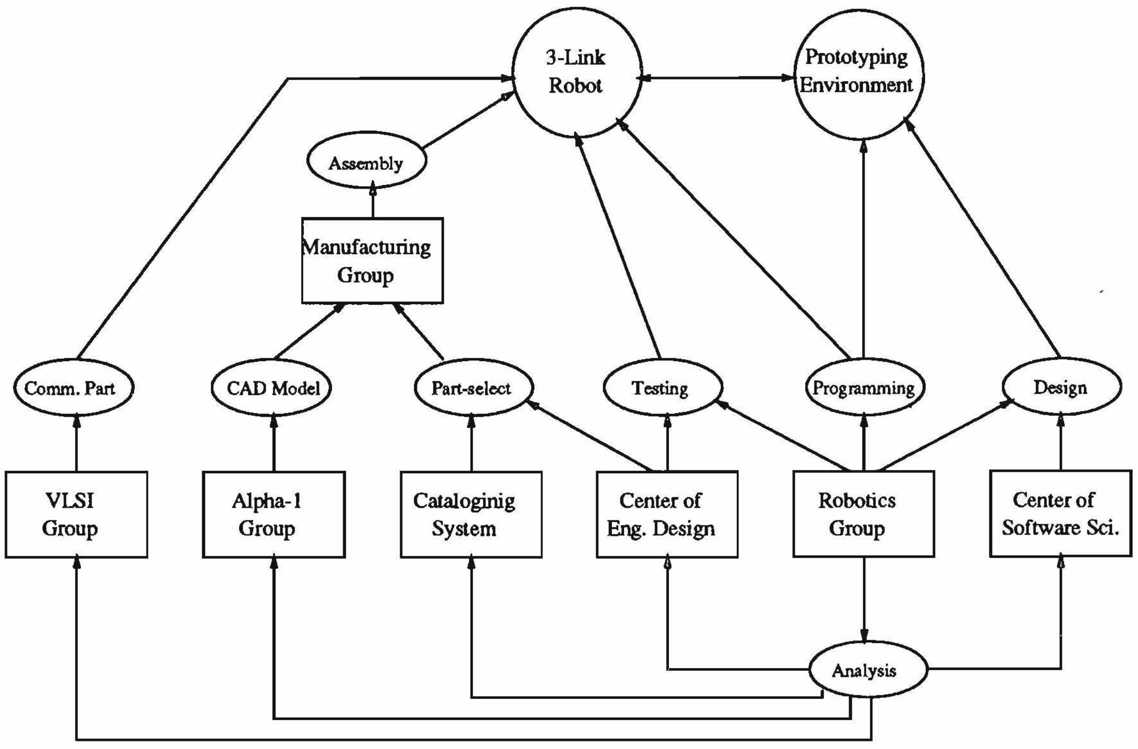


Figure 1: The interaction between the groups involved in the prototyping activity.

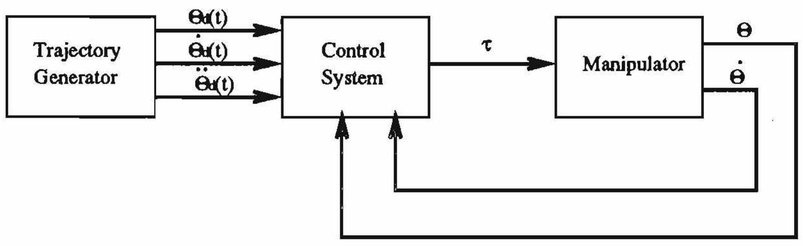


Figure 2: High-level block diagram of a robot control system.

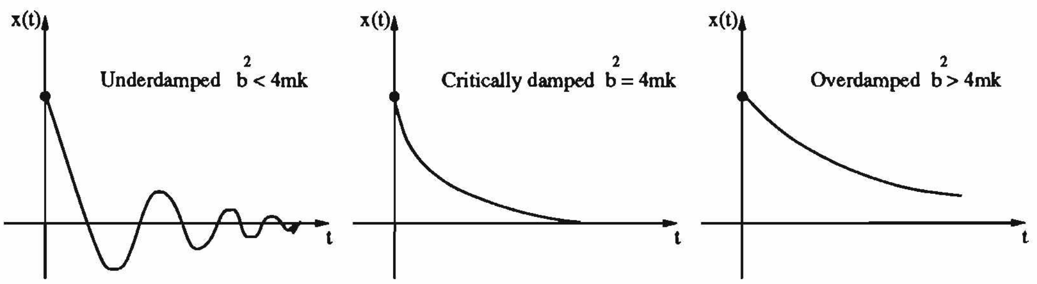


Figure 3: Different types of damping in a second order control system.

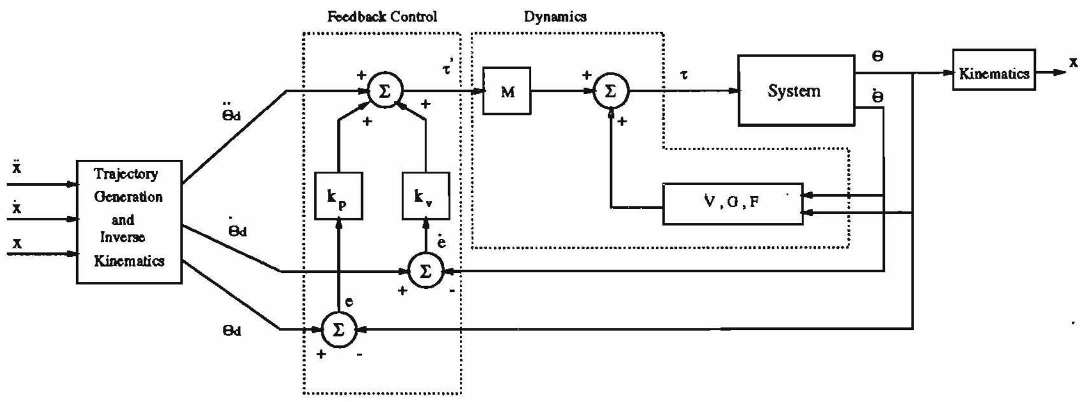


Figure 4: Block diagram of the controller of a robot manipulator.

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

- Requires a high update rate 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 [4]. Another method is to build an inner loop stabilizing controller using a multi-variable PD controller, and an outer loop tracking controller using a multi-variable PID (proportional, integral, and derivative) controller [49].

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 [19].

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, let's 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 controller, which stands for 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

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

But we assumed 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 which 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. In the following sections we will discuss some of these points in more detail.

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 pre-defined position trajectory $[x_i, y_i, z_i]$. This is called *position planning.*
- Move in a pre-defined velocity trajectory $[\dot{x}_i, \dot{y}_i, \dot{z}_i]$. This is called *velocity planning.*
- Move in a pre-defined acceleration trajectory $[\ddot{x}_i, \ddot{y}_i, \ddot{z}_i]$. This is called *force control.*

This 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's clear that increasing the update frequency results in decreasing the error. But the update frequency 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 [22] and showed that increasing the update rate decreases the error.

2.5 Special Computer Architecture for Robotics

When we design real time systems that involve a huge number of floating point calculations, speed becomes an important issue. In such situations, a hardware solution might be used to achieve the desired speed. Graham [14] provides an overview of specially designed computer architectures which enhance the computational capabilities to meet the needs of real-time control and simulation of robotic systems. Leung and Shanblatt [28] have addressed two important issues in this field: the decision on how specific an architecture should be and which architecture styles should be chosen for particular applications. They also devised a hierarchy for the computational needs in robotics applications which is composed of: *management, reasoning*, and *device interaction.*

The concept of the ASIC (Application-Specific Integrated Circuit) has created great opportunities for implementing robotic control algorithms on VLSI chips. In [29] a description is given of a conceptual framework for designing robotic computational hardware using ASIC technology. The advantages of ASIC for robotic applications include:

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

There are other approaches and applications for using the ASIC technology in robotic applications. Some of them can be found in [1, 23, 32, 44].

A VLSI architecture designed to compute the direct kinematic solution (DKS) on a single chip is described in [27]. It uses fixed-point operations and on-chip generation of trigonometric functions.

One of the latest advances in this area is the design of a 2400-MFLOPS reconfigurable parallel VLSI processor for robotic control. The speed of the chip is about 60 times faster than that of a parallel processor approach using conventional DSPs [13]. There are other approaches and applications for using the ASIC technology in robotic applications

2.6 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 [10] 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 [21] 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 [5, 36, 37]. 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 [34, 45].

TOCARD (Total Computer-Aided Design System of Robot Manipulators) is a system designed by Takano, Masaki, and Sasaki [48] 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 [17].

2.7 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 sub-systems, called Pallo Alto Collaborative Testbed (PACT), was proposed in [7]. 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 [3]. 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 is discussed in [39]. 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 [35], and a system for design, process planning and inspection is presented.

Several important themes in concurrent software engineering are examined in [11]. 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 [12]. The method of quality assurance is used to develop this system which 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 address the coordination and communication problems in engineering, was developed at the MIT Intelligent Engineering Systems Laboratory [47]. In their project they address several research issues such as, frameworks, representation, organization, design methods, visualization techniques, interfaces, and communication protocols.

Some important topics in software engineering can be found in [24], such as, the lifetime of a software system, Analysis and design, module interfaces and implementation, and system testing and verification. Also, a report about integrated tools for product, and process design can be found in [50].

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

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

More about this design is discussed in Section 4.

3 Three-link Robot Manipulator

To explore the basis of building a flexible environment for robot manipulators, we started with the design of a 3-link robot. This enabled us to determine the required sub-systems and interfaces for such an environment. This prototype robot will be used as an educational tool in control and robotics classes.

3.1 Analysis Stage

We started this project 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). We accomplished this phase by working through a generic example for a 3-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.

3.2 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.

We used a motor from the Mechanical Engineering lab, that is controlled by a PID controller. We also used an analog I/O card, named PC-30D, connected to a Hewlett Packard 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. We used the motor parameters to form this function by making some simplifications, since some of the motor parameters have non-linear components which makes it too

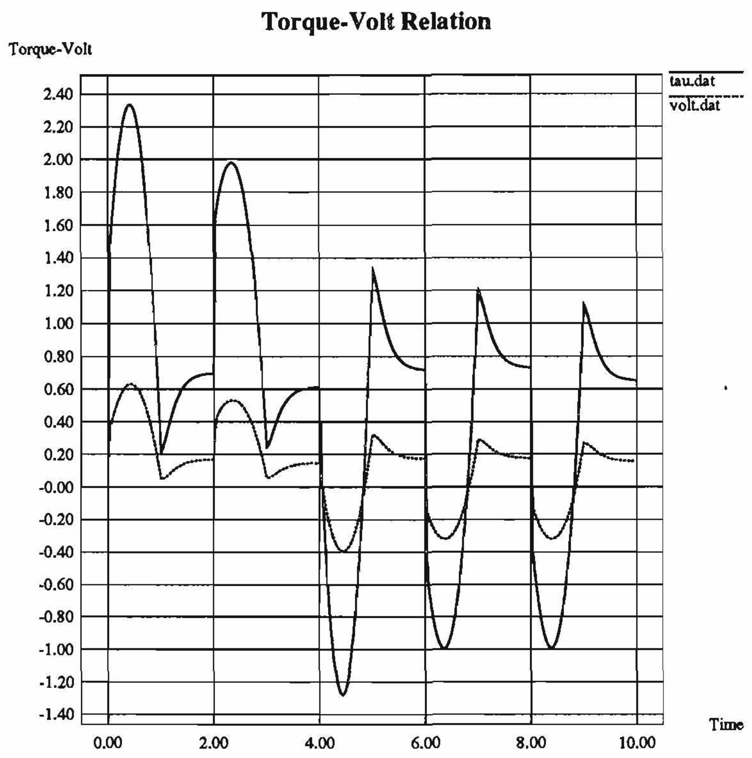


Figure 5: The relation between torque the voltage.

difficult to make an exact model. Figure 5 shows the relation between torque and voltage for a certain input sequence, and Figure 6 shows the circuit diagram of the motor and its parameters.

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 [8, 46].

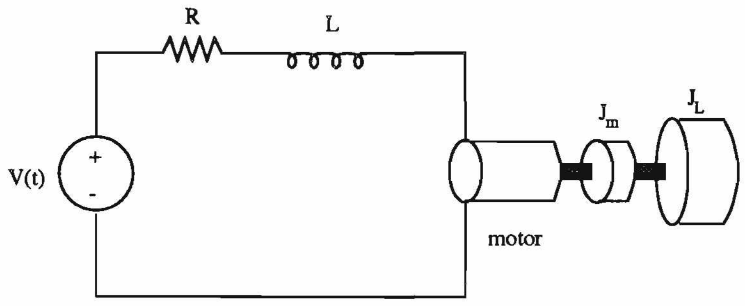


Figure 6: Circuit diagram of the DC-motor used in the experiment

3.3 Sensor and Actuator 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.[1] Basically, we have three digital to analog (D/A) lines that transmit the required torque (or voltage) from the workstation to the manipulator's three actuators, and we have three (analog to digital) A/D lines that transmit the sensors readings at each joint to the workstation. These readings are used in the controller for feedback information.

The problem requires interfacing a workstation with the motors and the sensors, such that a resident program on the workstation can send voltage values that will drive the motor in a certain direction (forward or backward), and read values from sensors placed on the motor that correspond to velocity and position of the motor.

We used a micro-controller system (the MC68HC11EVBU — Universal Evaluation Board) which has a micro-controller, 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. The MC68HC11E9 high-density complementary semiconductor (HCMOS) high-performance micro-controller unit(MCU) includes the following features:

- 12 Kbytes of ROM
- 512 bytes of EEPROM
- 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 very low frequencies. The chip has been programmed to first start a continuous A/D conversion which keeps updating the result registers as the data is ready. It then establishes communication with the workstation. Next the chip reads the voltage to be sent to the motor, then transmits the sensor values from the A/D result register to the workstation, and sends the voltage value to the DAC and goes back to getting the new voltage value from the workstation. The chip does not wait for an A/D conversion but gets the last updated value. The A/D conversion takes place rapidly in the background at a 2 MHz clock rate. For more details about this chip see [38].

3.4 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.

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

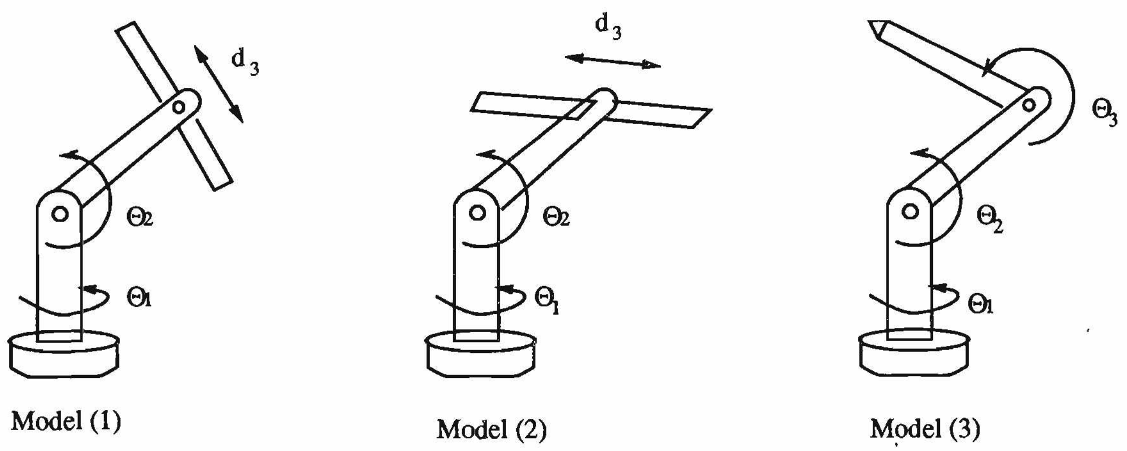


Figure 7: Three different configurations of the robot manipulator.

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 7).

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.[2] One problem with the resultant equations is that they are not simplified at all, therefore, we simplified the results using the mathematical package *Mathematica*, which gives more simplified results, but still, not totally factorized. The comparison between the number of calculations before and after simplification will be discussed in the benchmarking section.

For the trajectory generation, we used the cubic polynomials method that was described above in the trajectory generation section. 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).

3.5 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

[2]These tools were developed by Patrick Dalton.

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. We used the third model 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.

3.6 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, 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.

We used the simulator described in Section 3.5 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, we used a program called *gprof* which 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 we used *xgraph* to draw charts showing these time profiles. We ran the simulation program 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. One thing we noticed was that: after simplification using Mathematica, the execution time

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

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

	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

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

increased, but that was because the results contained many different trigonometric functions, and it seemed that these machines do not use lookup tables for such functions. So, we rewrote all non-basic trigonometric functions, such as $sin2\theta$ in terms of basic trigonometric functions as $2sin\theta cos\theta$. Using this conversion, the performance improved significantly. Figure 8 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.

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.

3.7 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 3-link prototype robot. One of the most complicated parameters is the inertia tensor matrix for each link, especially when the links are non-uniform 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.[3] Figure 9 shows the interface window of that simulator.

3.8 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 we can set the robot in different configurations (see Figure 10).

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

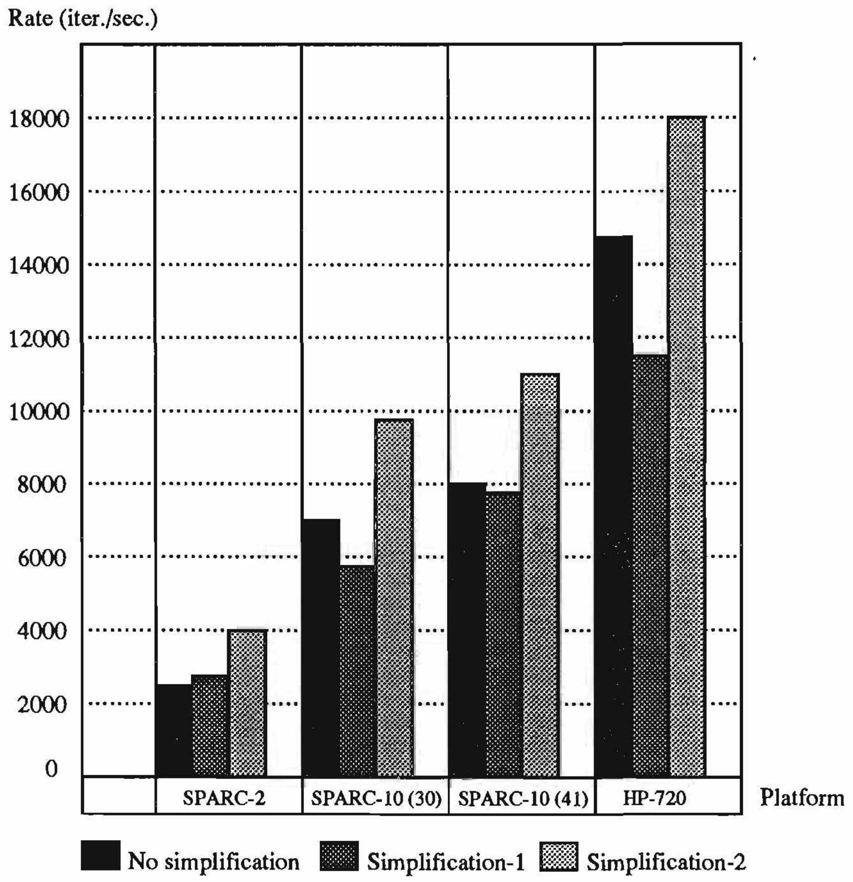


Figure 8: Performance comparison for different platforms.

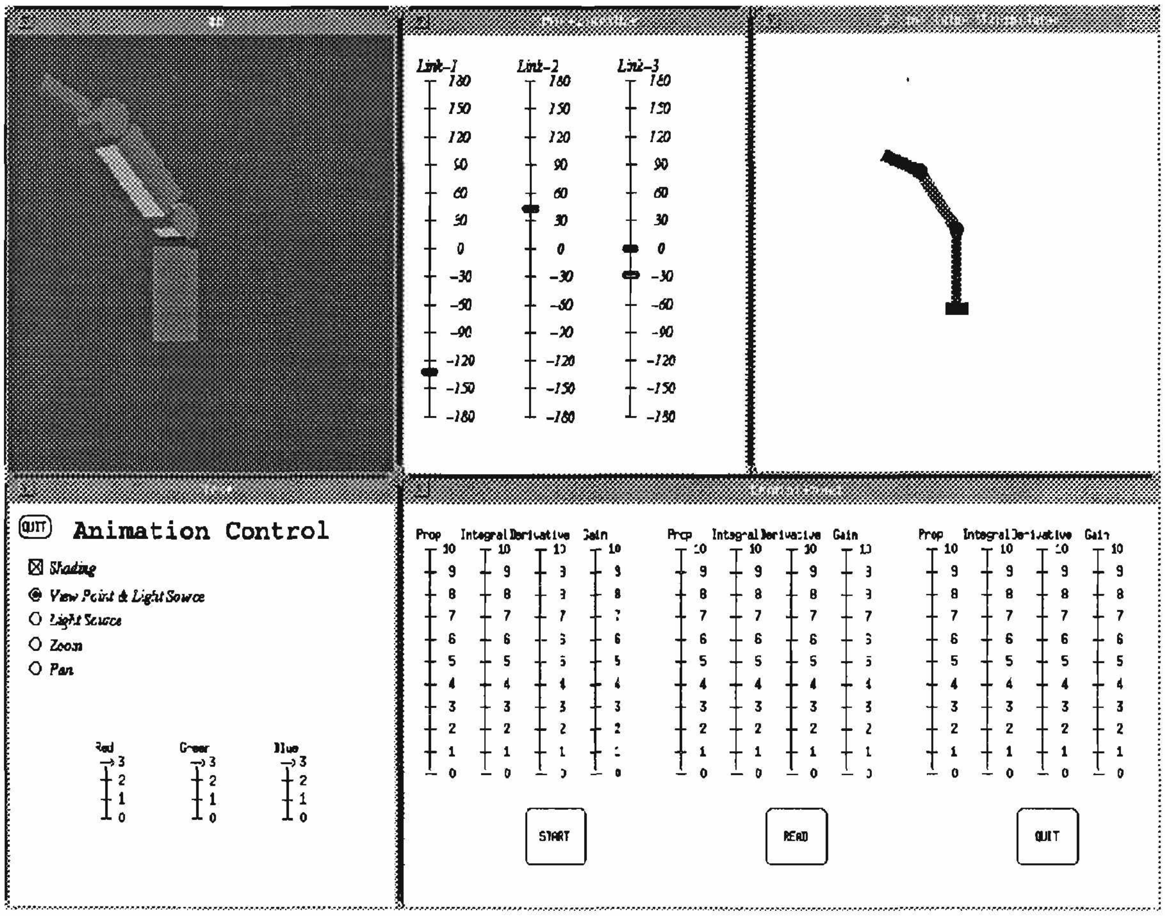


Figure 9: The interface window for the PID controller simulator.

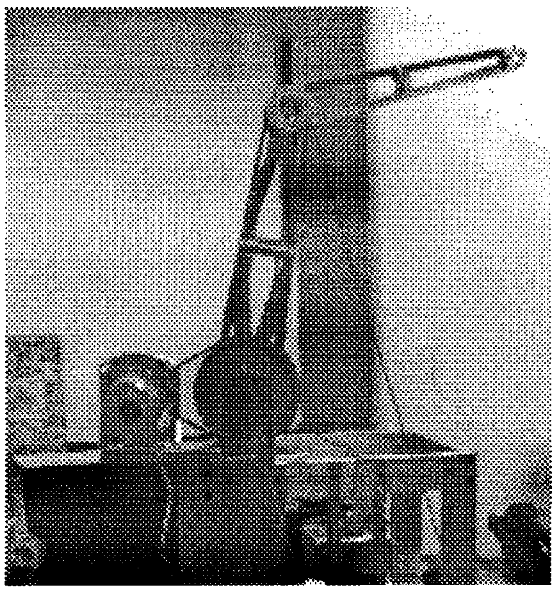


Figure 10: The physical three-link robot manipulator.

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, and monitoring its behavior with an oscilloscope. 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 11 shows an overall view of the different interfaces and platforms that can control the robot. A summary of this design can be found in [9].

4 The Prototyping Environment

The prototyping environment consists of several sub-systems such as:

- Design.
- Simulation.
- Control.
- Monitoring.
- Hardware selection.
- CAD/CAM modeling.
- Part Ordering.

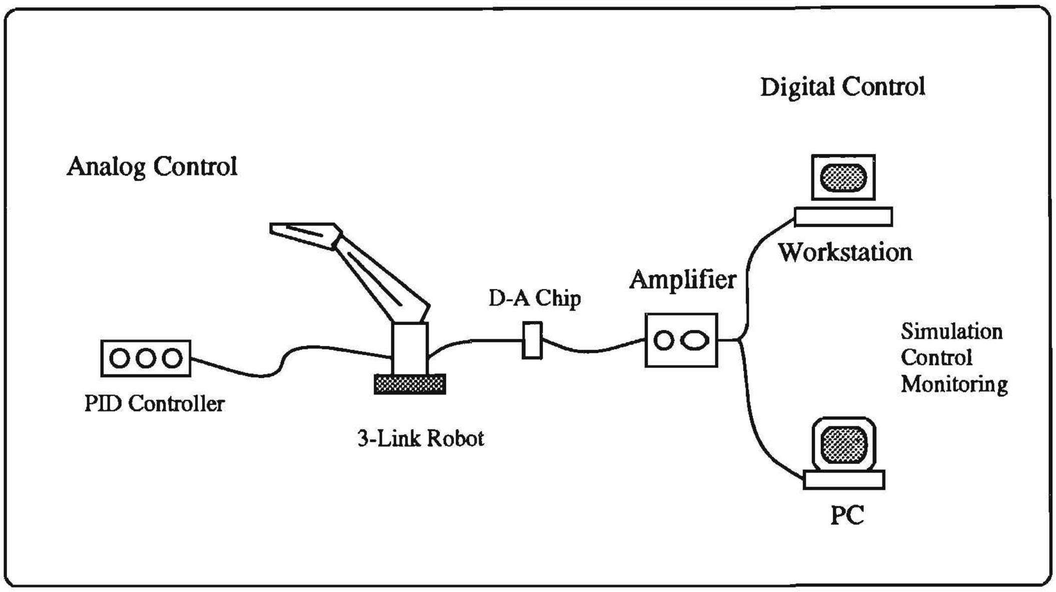


Figure 11: Controlling the robot using different schemes.

- Physical assembly and testing.

Figure 12 shows a schematic view of the prototyping environment with its sub-systems and the interface.

These sub-systems 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 robot prototype and the sub-systems, and it will also serve as a communication channel between the different sub-systems.

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 sub-systems.
- Maintaining a set of rules that governs the design and modeling of the robot.
- Handling the communication between the sub-systems using a specified protocol for each system.
- Identifying the data format needed for each sub-system.
- Maintaining comments fields associated with some of the sub-systems 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

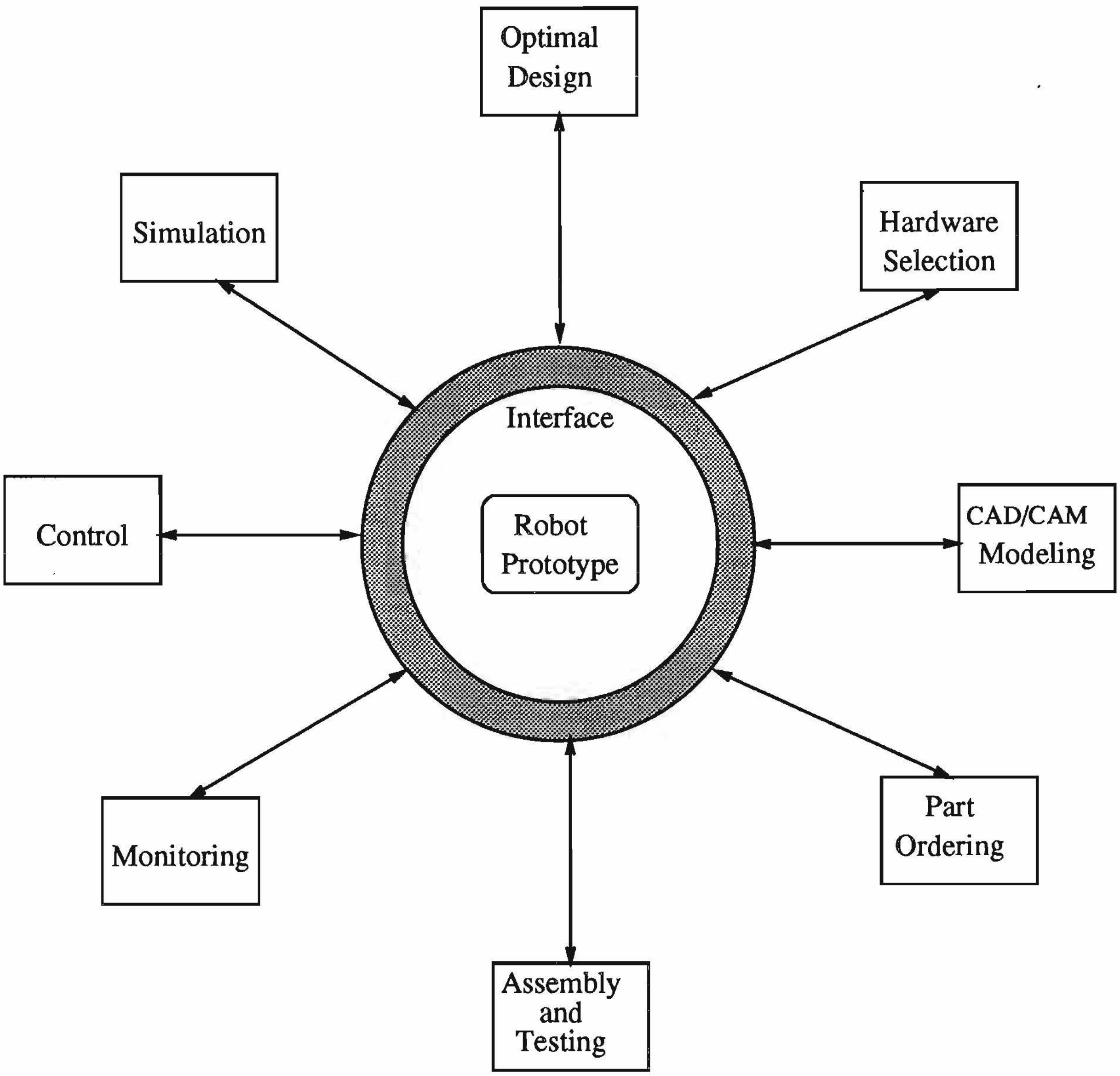


Figure 12: Schematic view for the robot prototyping environment.

of the reasoning strategy for each sub-system, 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 sub-system 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.

4.1 Interaction Between Sub-systems

To be able to specify the protocols and data transformation between the sub-systems in the environment, the types of actions and dependencies among these sub-systems must be identified. Also, the knowledge representation used in each sub-system 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.).

4.2 The Interface Scheme

There are several schemes that can be used for the interface layer. One possible scheme in which each sub-system has a sub-system interface (SSI) which has the following tasks:

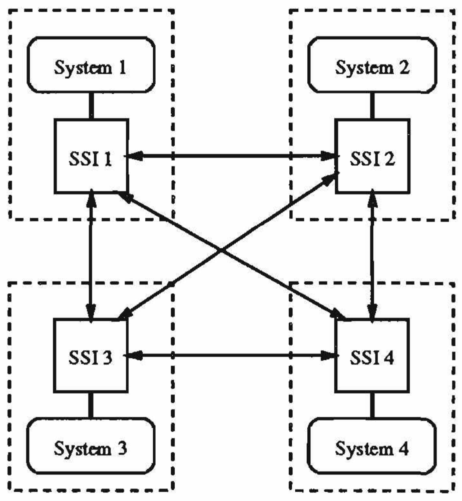
- Transfer data to and from the sub-system.
- Send requests from the sub-system to the other interfaces through the central interface.
- Receive requests from other sub-system interfaces and translate them to the local language.

These sub-system interfaces can communicate in three different ways (see Figure 13):

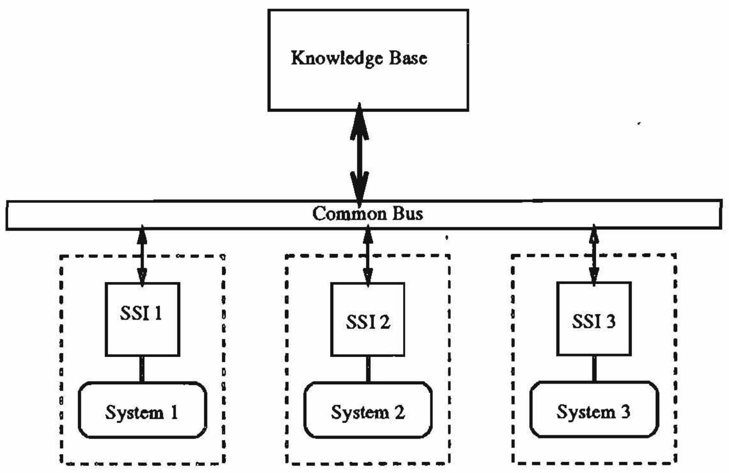
Direct connection: which means that all interfaces can talk to each other. The advantage of this is that it has a high communication speed; however, it makes the design of such interfaces more difficult, and the addition or modification of one of the interfaces requires the modification of all other interfaces.

Message routing: in this scheme, 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 sub-system. The problem with this scheme is that it makes the synchronization of the sub-systems very difficult, and the design of the interface is more complicated.

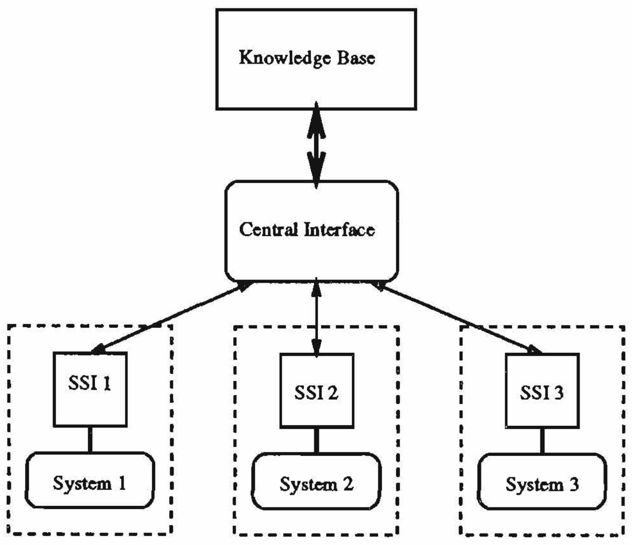
Centralized control: in which all interfaces talk with one centralized interface that controls the data and controls flow in the environment. The advantage of this scheme is that it makes it much easier to synchronize between the sub-systems, and the addition or modification of any of the SSIs will not affect the other SSIs.



(1) Direct Connection



(2) Message Routing



(3) Centralized Control

Figure 13: Three different methods for sub-system interface communication.

4.3 Overall Design

The Prototyping Environment (PE) consists of a *central interface* (CI) and *sub-system 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 sub-systems to update any changes in the system. This requires the central interface to know which sub-systems need to know these changes and send messages to these sub-systems informing them of the required changes.
- Receive messages and reports from the sub-systems when any changes are required, or when any action has been taken (e.g., update complete).
- Transfer data between the sub-systems 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 sub-system interfaces are the interface layers between the CI and the sub-systems. This makes the design more flexible and enables us to change any of the sub-systems 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 sub-system.
- Send acknowledgments or error messages to the CI.

The assumption is that there is a user at each sub-system (by a user here we mean one or more skilled persons who understand this sub-system), 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 sub-systems. Figure 14 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 sub-systems by sending messages to the corresponding user at each sub-system, 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.

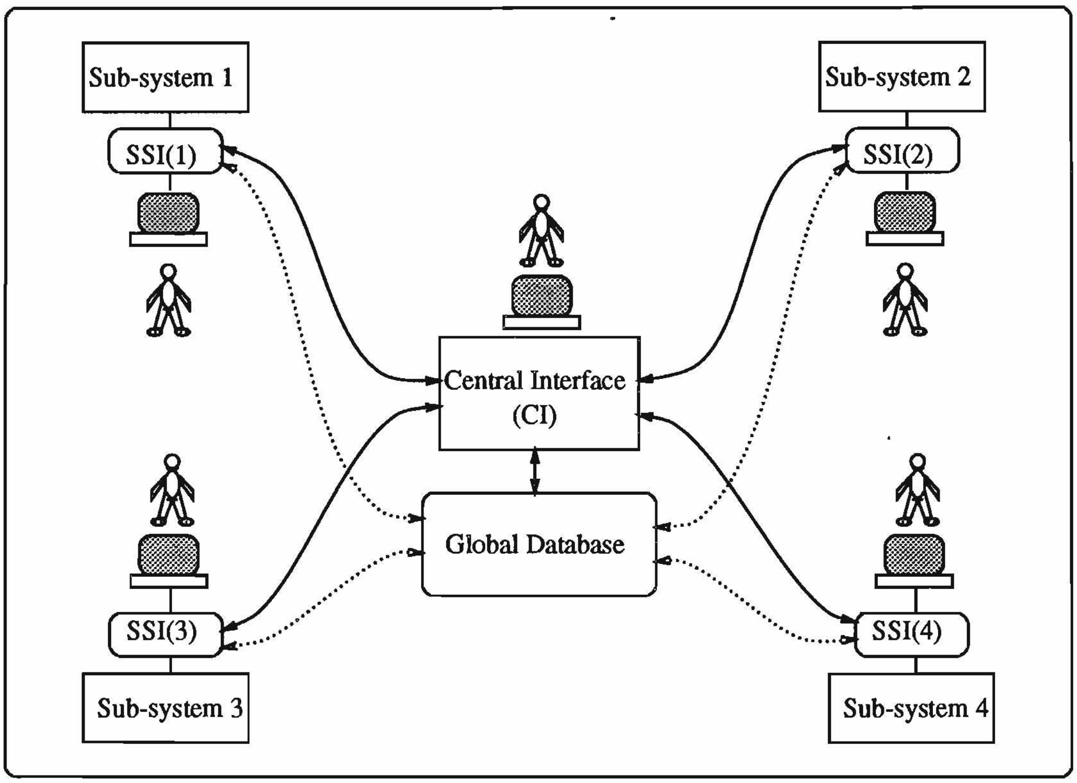


Figure 14: Overall design of the prototyping environment.

4.3.1 Communication Protocols

The main purpose of this environment is keep all the sub-systems informed of any changes in the design parameters. Therefore, passing information between the sub-systems is the most important part of this environment. To be able to control the information flow, we have developed some protocols that enable the communication between these sub-systems in an organized manner. In our design, all sub-systems communicate through the CI which is responsible for passing the information to the sub-systems that need to know.

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

1. Change reported from one of the sub-systems.

2. Request for data from one sub-system to another.

Figure 15 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 sub-systems, Apply relations and check constraints. If constraints are satisfied, go to state 3, else, report non-satisfied constraints to sender and go to steady state.

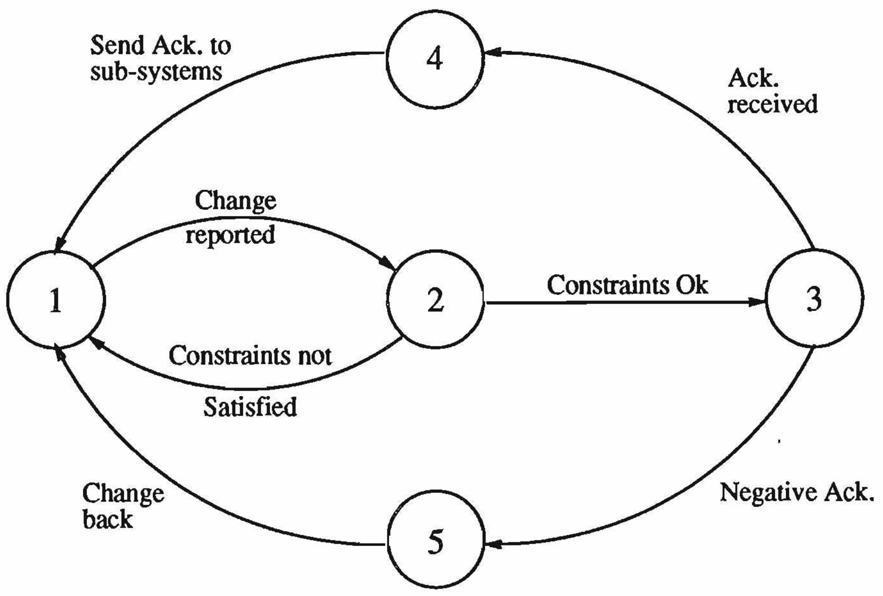


Figure 15: Finite state machine representation for the change protocol.

3. Constraints are satisfied: Notify the sub-systems with the changes and wait for acknowledgments.
4. Acknowledgments received from all sub-systems: Send final acknowledgment to the sub-systems and go to steady state.
5. Acknowledgments not Ok: Send a "change-back" command to the sub-systems and go to steady state.

Figure 16 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
       if constraint satisfied then
```

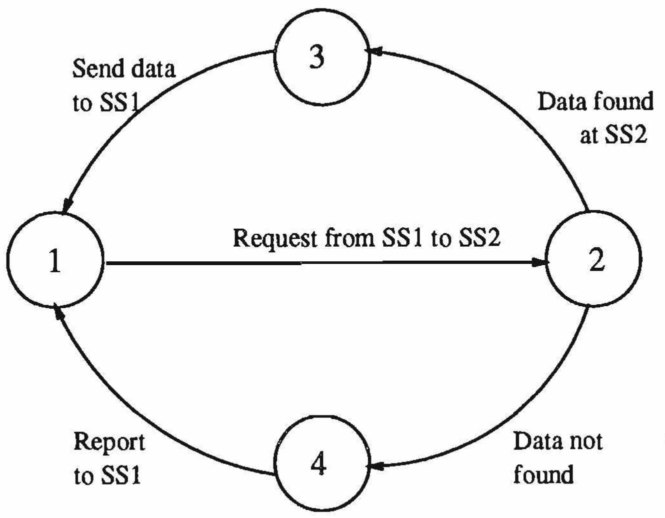


Figure 16: Finite state machine representation for the data request protocol.

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

Figure 17 shows a possible scenario when applying this protocol. In this algorithm we assume that all system constraints are located in the CI; however, any sub-system may reject the proposed values by other sub-systems 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.

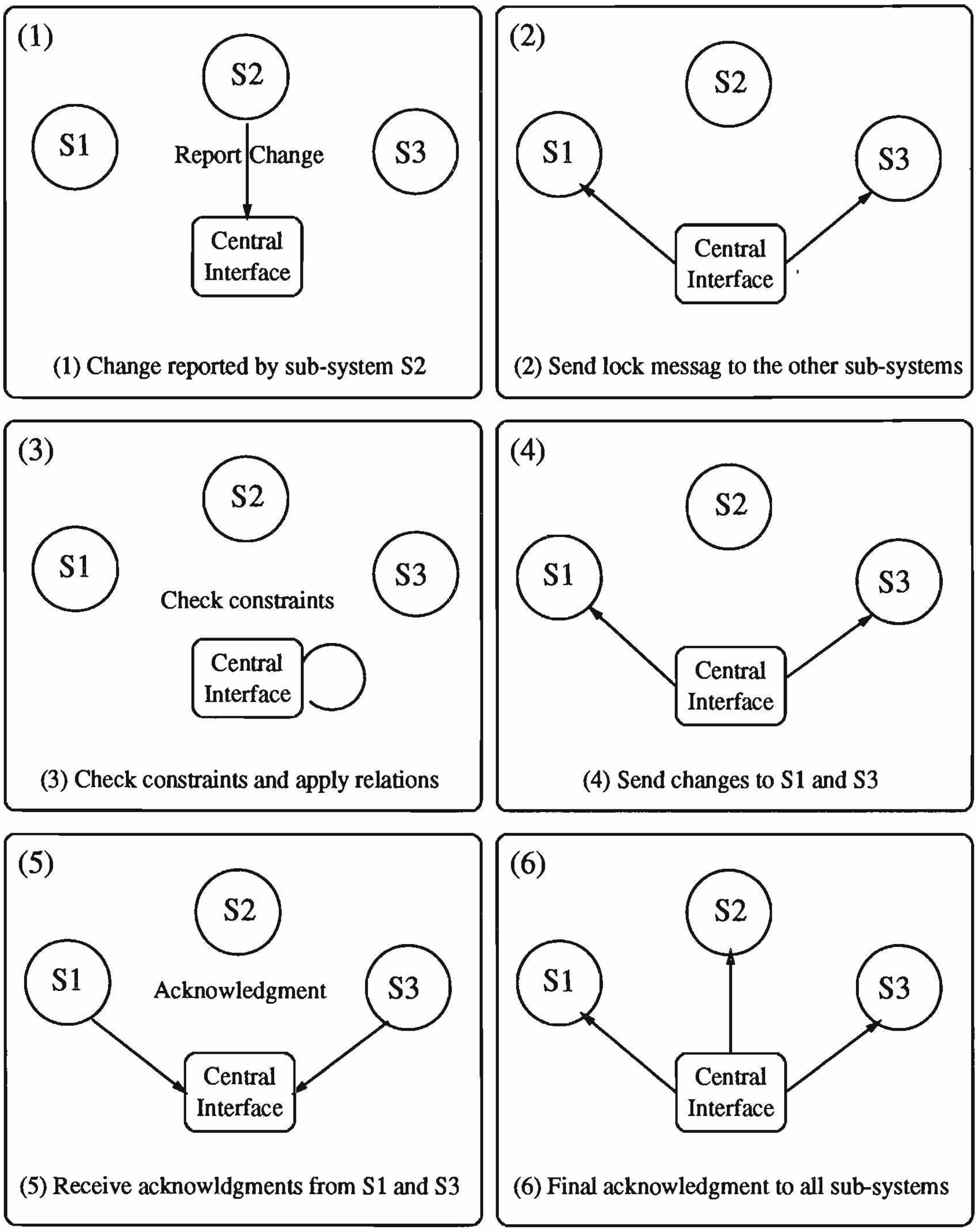

(1)
S2
S1
Report Change
S3
Central Interface
(1) Change reported by sub-system S2
(2)
S2
S1
S3
Central Interface
(2) Send lock messag to the other sub-systems
(3)
S2
S1
S3
Check constraints
Central Interface
(3) Check constraints and apply relations
(4)
S2
S1
S3
Central Interface
(4) Send changes to S1 and S3
(5)
S2
S1
Acknowledgment
S3
Central Interface
(5) Receive acknowldgments from S1 and S3
(6)
S2
S1
S3
Central Interface
(6) Final acknowledgment to all sub-systems

Figure 17: Possible scenario for the communication between the sub-systems.

4.3.2 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 sub-systems. 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 we can increase it, 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.

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 sub-system 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 sub-system again. The last solution has the user in the loop as well, but incorporating all the constraints in the CI reduces the inter-process communication and speeds up the checking process. This last solution was chosen in our design.

4.3.3 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 sub-system can perform any changes, but they can make suggestions to the CI. This means that, the design sub-system 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:

  - More complicated user interface for the CI.
  - More data should be kept in the global database, such as, performance requirements, objective functions, etc.
  - It requires a highly skilled user who is able to perform both design and coordination at the same time.

  - An optimal design sub-system cannot be used in the system.

- The design sub-system 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 sub-system. 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 sub-system 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 sub-systems. Also, the CI is responsible for checking the constraints and applying the update rules. The advantages are:

  - The user of the CI doesn't need to know much about the design details and technicalities.
  - Any design sub-system 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 sub-systems.
  - This scheme requires heavy use of *inter-process communication* which needs more sophisticated protocols to maintain reliable data transmission.

We have chosen the last option 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 sub-system), 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.

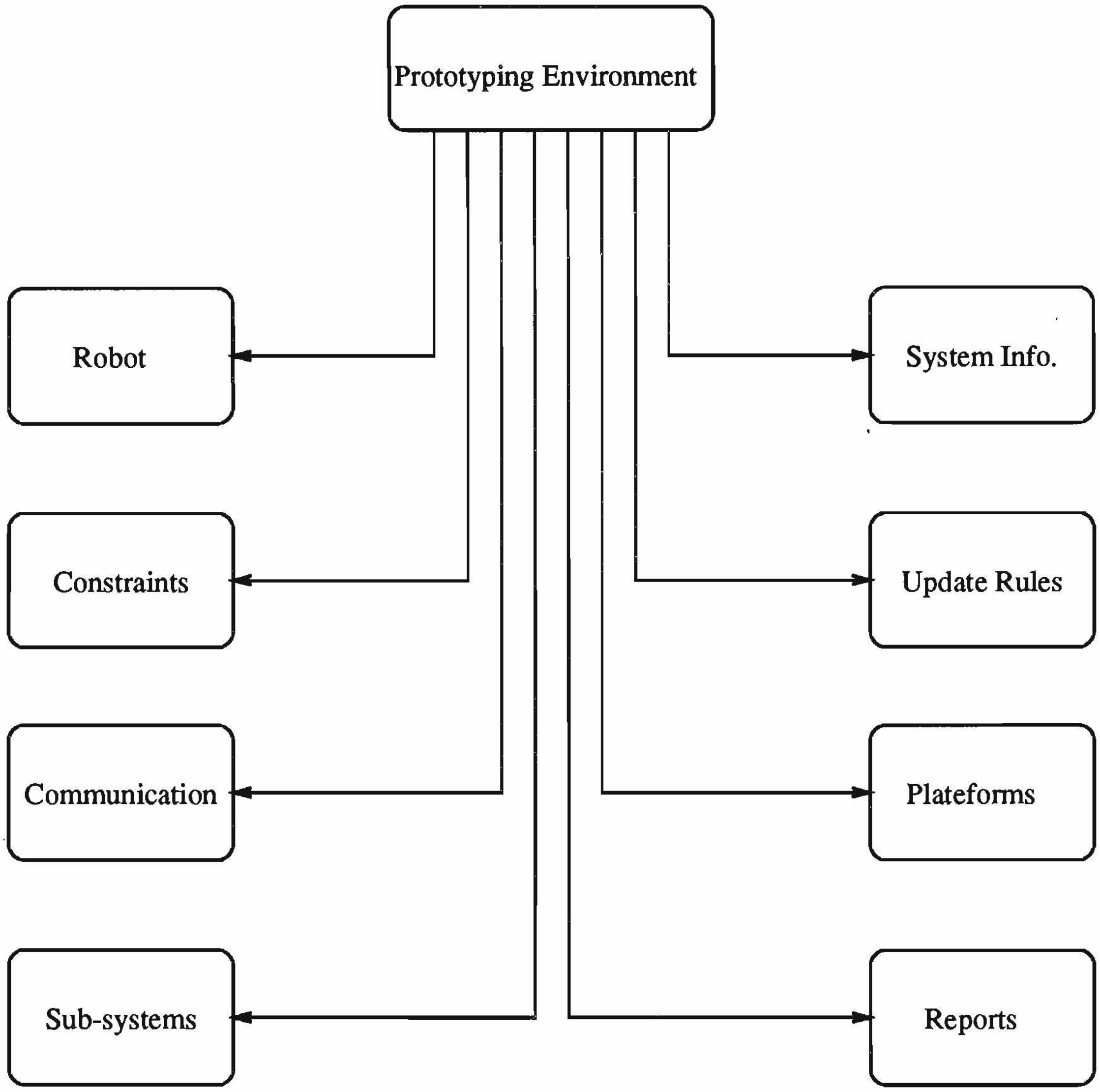


Figure 18: The main components of the robot prototyping environment.

4.4 Object-Oriented Analysis

Object analysis approach is used to determine the system components and functions, and the relations between them. Top-down approach is used starting from the main objects in the PE, then analyze each of these objects in more detail until we reach the primitive data items. 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 18 shows the top view of the main components in the system, and Figure 19 shows one of these components in detail.

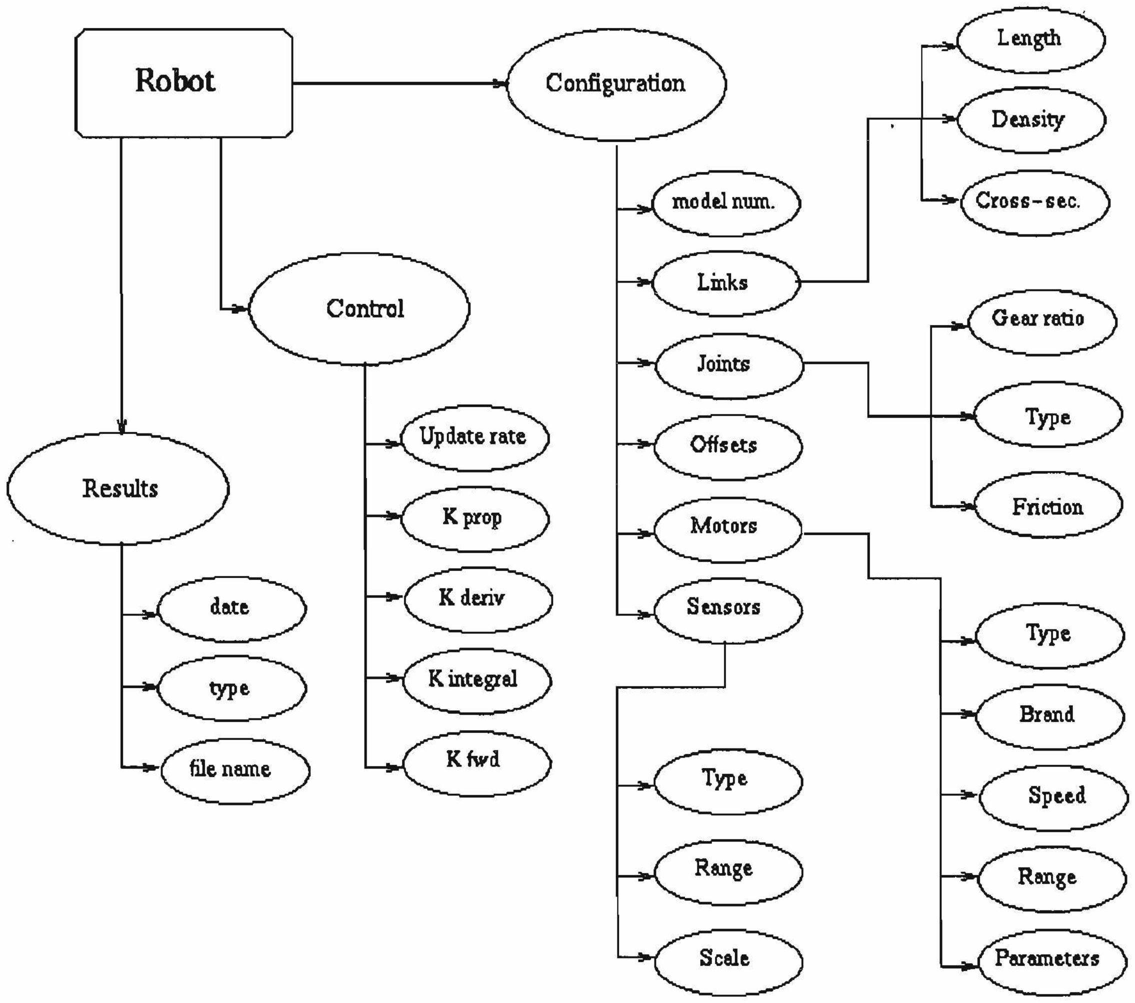


Figure 19: Detailed analysis for the robot classes.

4.5 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 sub-systems 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.
- Sub-systems 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 we do not even need a special DBMS and all operations can be performed by simple functions.

4.5.1 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 sub-systems of any changes that occur to any of these parameters. For the system to perform this task, it needs to know two things:

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

Figure 20 shows a list of the design parameters along with the sub-system that can change them and the sub-systems 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," this is because only one sub-system needs to know about this parameter and it is the same sub-system that can change it. But we will keep it for possible future extensions or additions of other sub-systems that might be interested in this parameter.

4.5.2 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 same for the robot sub-classes, each of them having 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 21 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, while the double arrow arcs represent a one-to-many relation.

4.5.3 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.
- 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"), then a copy of the necessary records from the current design 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 21 contains some information about the design such as the design number, the starting date, the finishing date, and the platform used for that design.

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 20: Sub-system notification table according to parameter changes.

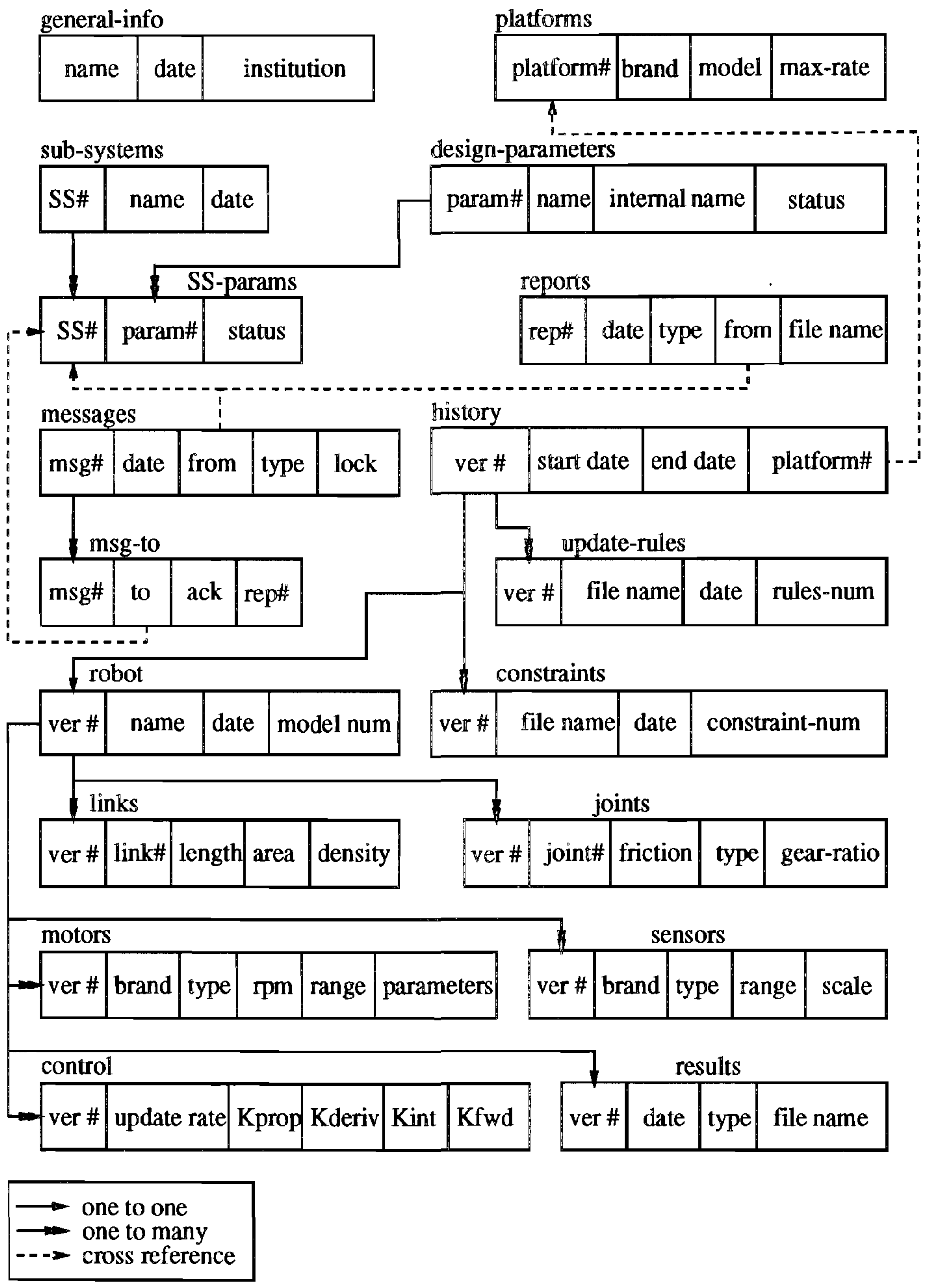


Figure 21: Database design for the system.

4.6 Constraints and Update Rules Compiler

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

4.6.1 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>
           <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. One more thing that can be added to this compiler later is some error detection and recovery modules for syntax error handling.

4.6.2 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()
{
    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.

4.7 Implementation

In the following sub-sections we investigate some implementation issues, and describe the different components in our design and how we implemented each of them.

4.7.1 The Central Interface

The central-interface (CI) is the core program that handles the communication between the sub-systems, 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 3 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 4.3.1. 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 sub-systems 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 which contains information about the robot configuration, platforms, reports, design history, sub-systems, 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 which 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 such as, *read*, *write*, and *find*, while the rest are general operations that are implemented as member functions in the basic file class.

4.7.2 The PE Control System

The CI as described above has no user interface. To be able to control and manage the coordination between the sub-systems, we implemented the PE control system (PECS) with some functionalities that enables 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

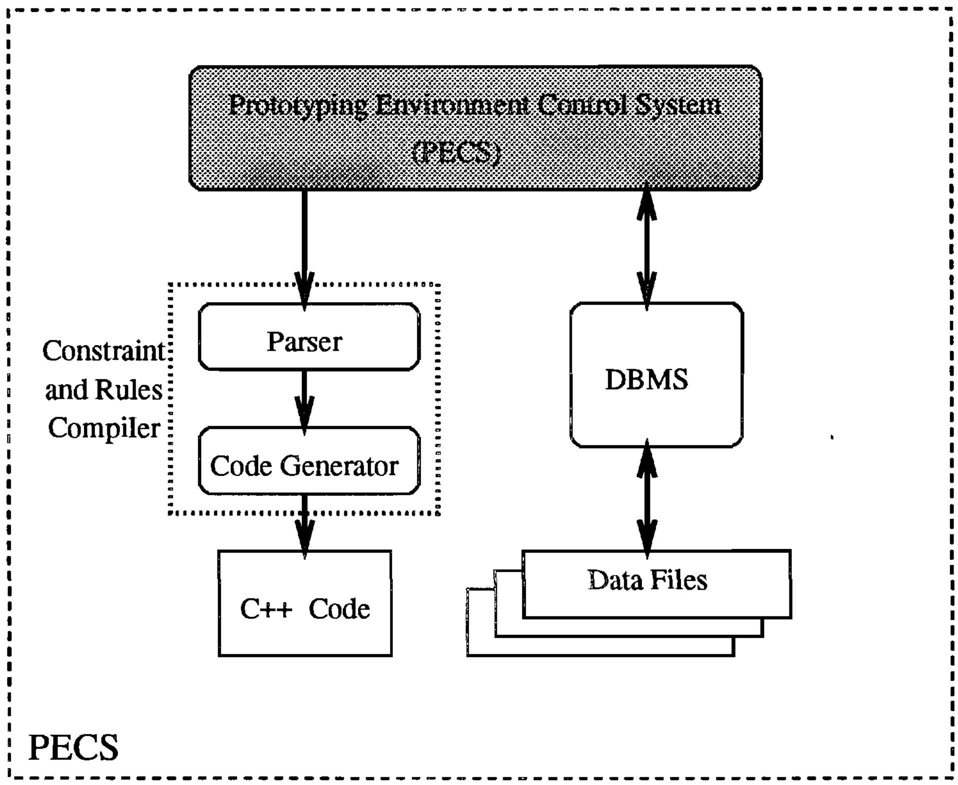

Prototyping Environment Control System (PECS)
Constraint and Rules Compiler
Parser
Code Generator
C++ Code
DBMS
Data Files
PECS

Figure 22: Schematic overview of the PECS.

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 we reduce the complexity of the compiler 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 22. Figure 23 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 sub-systems of the prototyping environment. Figure 24 shows a query for the current robot configuration.

Actions: these are the actual operations that control the CI. these actions includes updating the constraints and the update rules, compiling the CI after including the new constraints and update rules, activate, and terminate the CI. Figure 25 shows one of these operations which is updating the constraints.

Reports: these operations for managing the reports in the system, and sending and receiving reports to and from the sub-systems. 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.

Type	Description	Direction
Change	Data change reported	SSI $\longrightarrow$ CI
Const_Not_Ok	Constraints not satisfied	CI $\longrightarrow$ SSI
Notify	Send changes to sub-systems	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

Table 3: Message types used in the communication protocols.

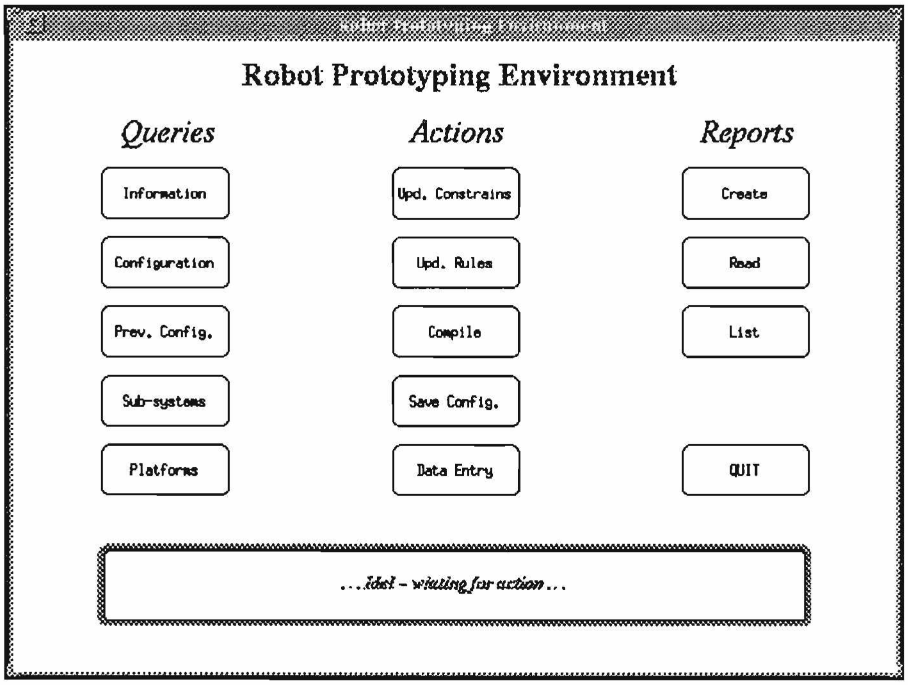
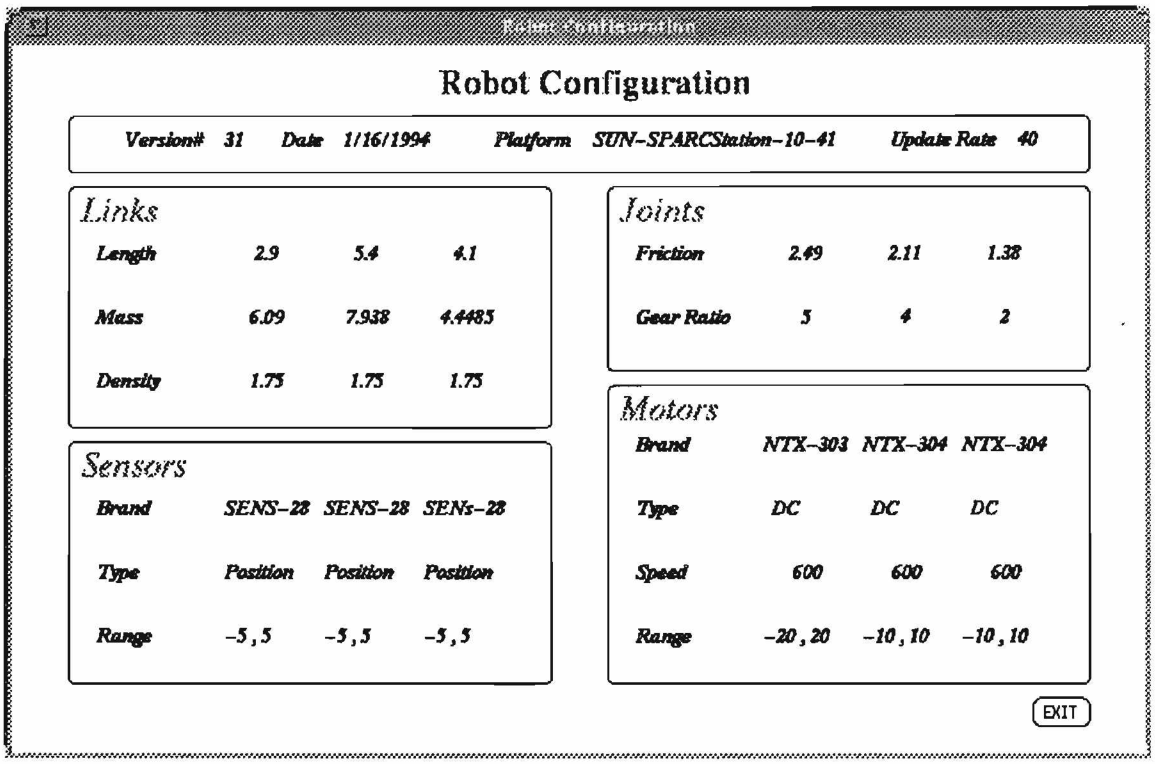


Figure 23: The main window for the PE control system.



Robot Configuration

Version# 31 Date 1/16/1994 Platform SUN-SPARCStation-10-41 Update Rate 40

Links

Length	2.9	5.4	4.1
Mass	6.09	7.938	4.4485
Density	1.75	1.75	1.75

Joints

Friction	2.49	2.11	1.38
Gear Ratio	5	4	2

Sensors

Brand	SENS-28	SENS-28	SENs-28
Type	Position	Position	Position
Range	-5 , 5	-5 , 5	-5 , 5

Motors

Brand	NTX-303	NTX-304	NTX-304
Type	DC	DC	DC
Speed	600	600	600
Range	-20 , 20	-10 , 10	-10 , 10

Figure 24: The current robot configuration window.

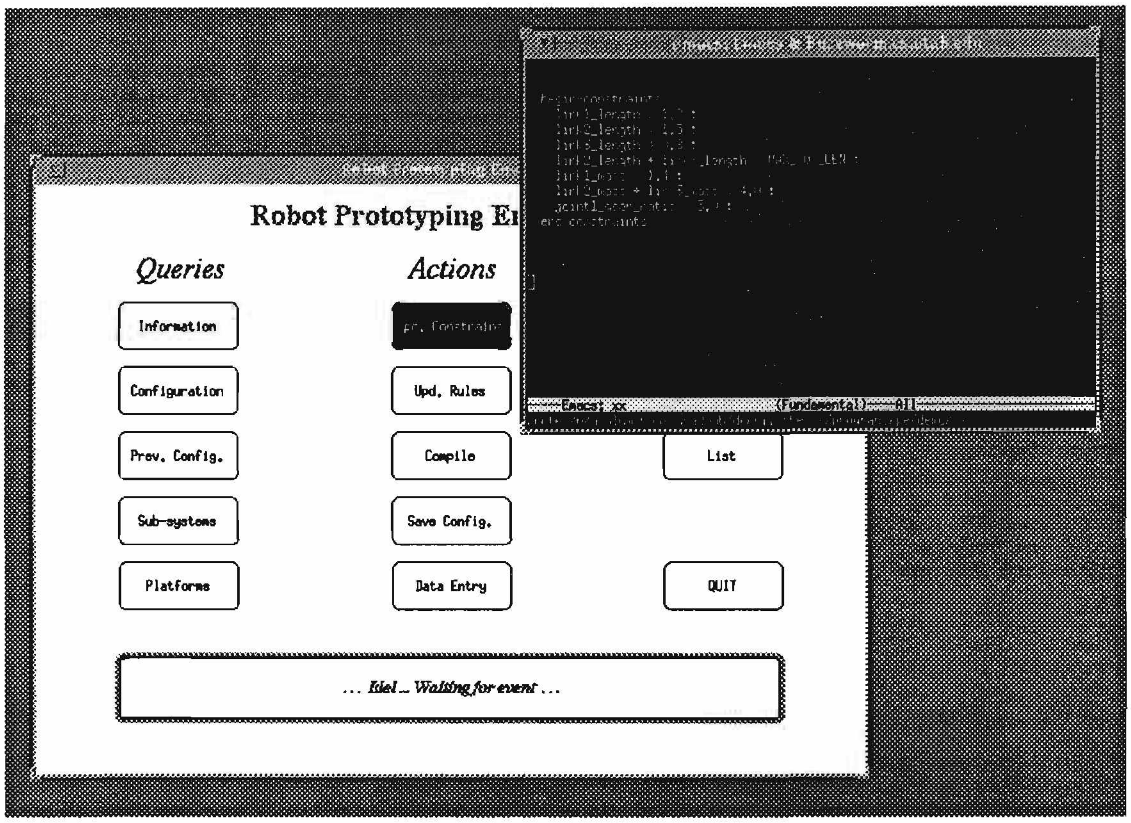
4.7.3 Initial Implementation of the SSIs

In the first phase of implementation, the SSIs serve as a simple interface layer between the CI and the user at each sub-system. 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 26 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 sub-system will be notified with any action taken. For example, updating a data file that is used by the sub-system 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.

4.7.4 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 27). 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 5.4 with sequences of the CIM window showing the activities that took place in each experiment.



```
begin constraints
  link1_length [illegible] ;
  link2_length [illegible] 1.5 ;
  link3_length [illegible] ;
  link2_length + link3_length [illegible] ;
  link1_mass [illegible] ;
  link2_mass + link3_mass [illegible] ;
  joint1_acc_ratio [illegible] ;
end constraints

----Emacs: xx                    (Fundamental)----All------------
```

Figure 25: Updating the design constraints through the PECS.

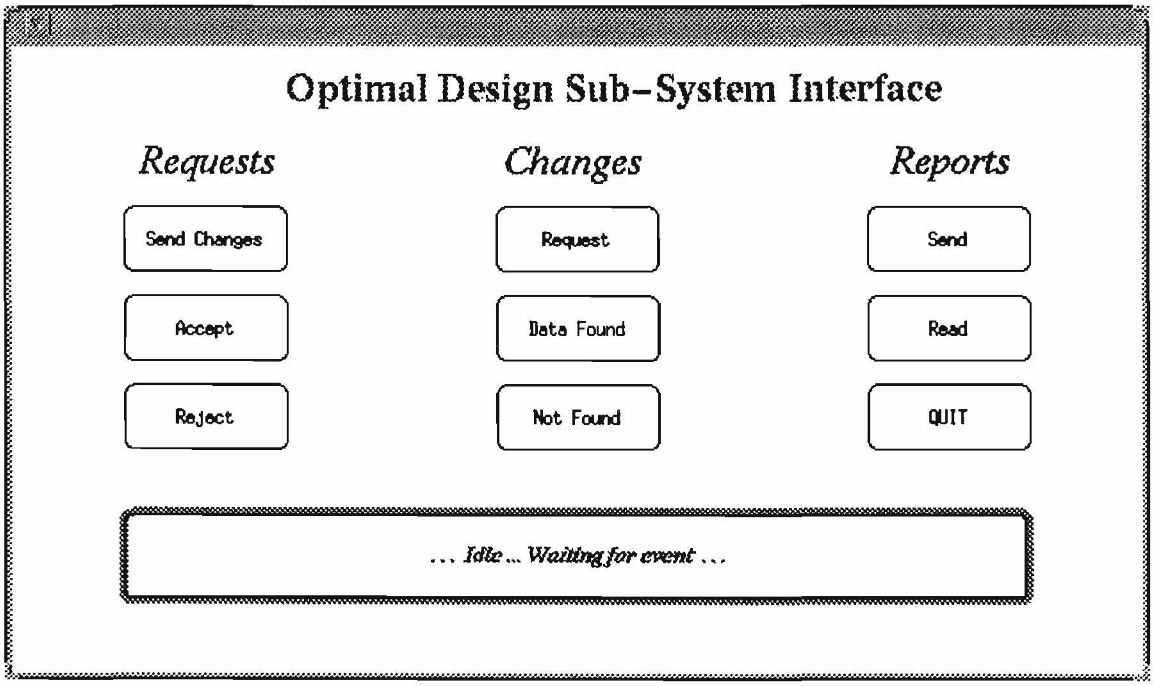


Figure 26: The user interface for the SSI.

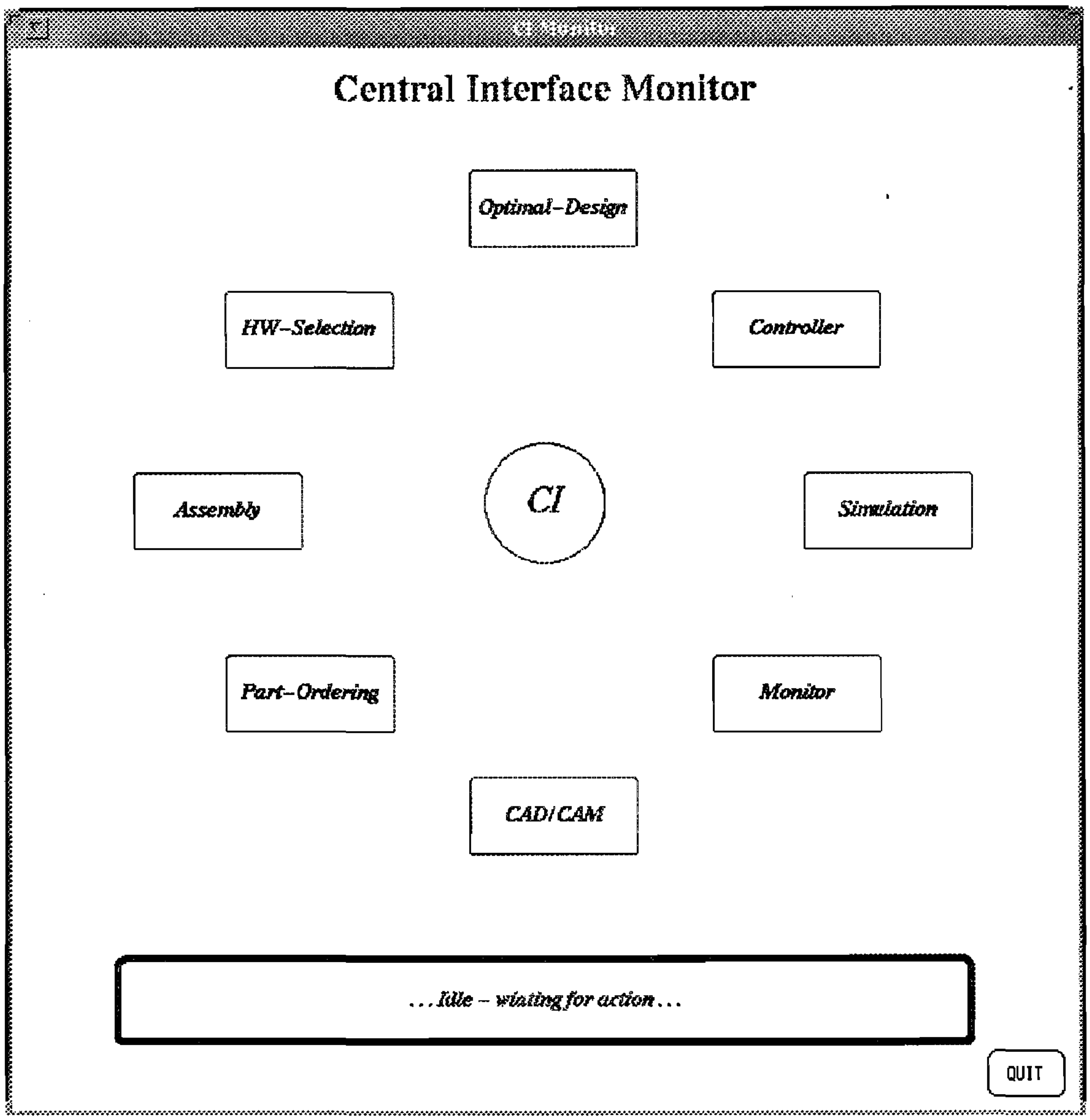


Figure 27: The user interface for the SSI.

5 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 1-link and the 3-link robot are described, with the results shown graphically.

5.1 One-link Robot

Building the 3-link robot has passed through several stages until we reached the final version. As mentioned before, we started by 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 we run a graphical simulation program 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 28, 29, and 30 show the output windows displaying the link and graphs for the position and the velocity. Figure 31 shows the graphs for the actual and desired position and velocity for the three sequences.

5.2 Simulator for 3-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 32 shows the simulated behavior of a 3-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 33. From this figure, it is clear that increasing the update rate decreases the position error.

5.3 Software PID Controller

A software controller was implemented for the 3-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)

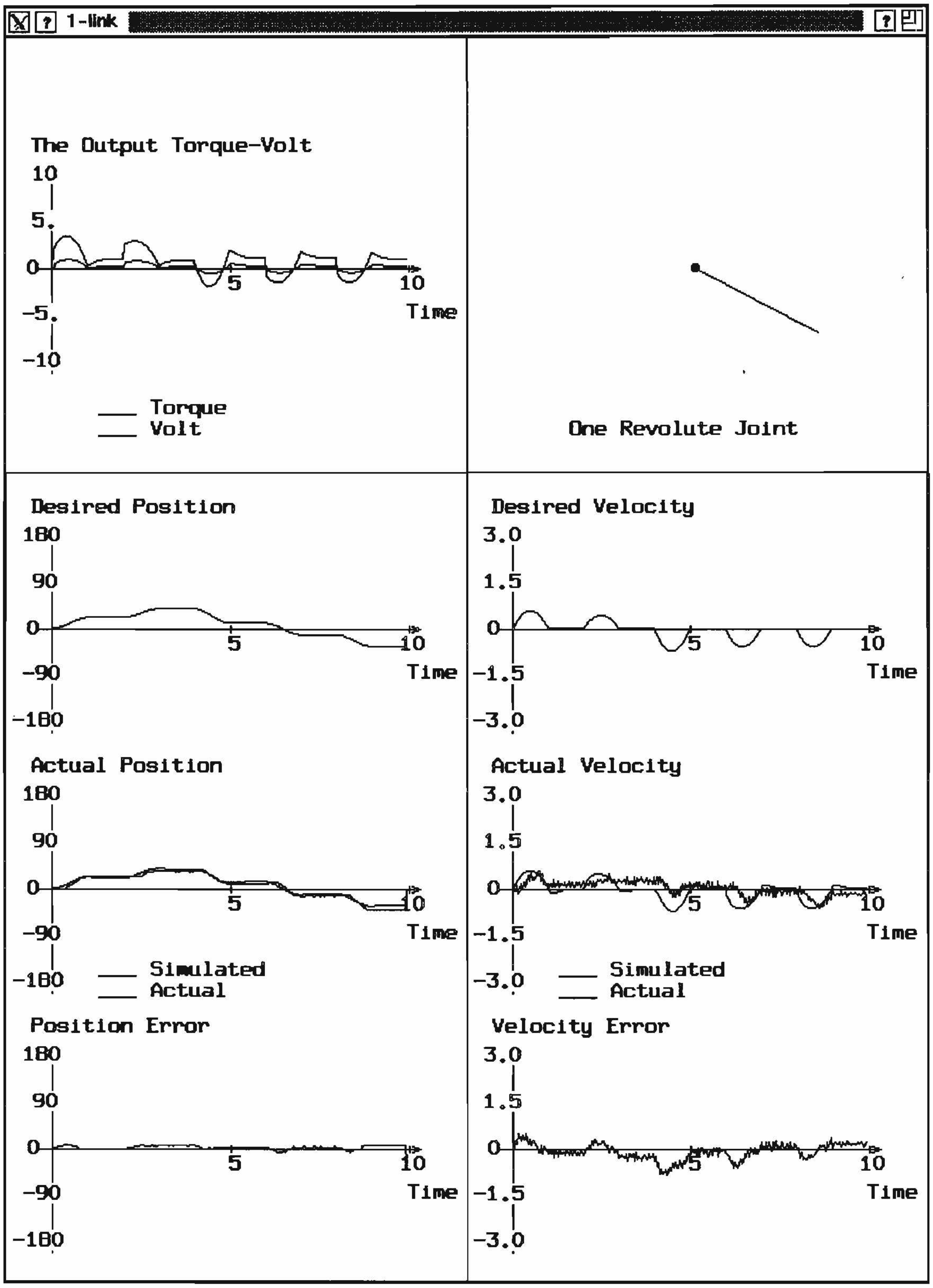


Figure 28: The behavior of the one-link robot for the first input sequence.

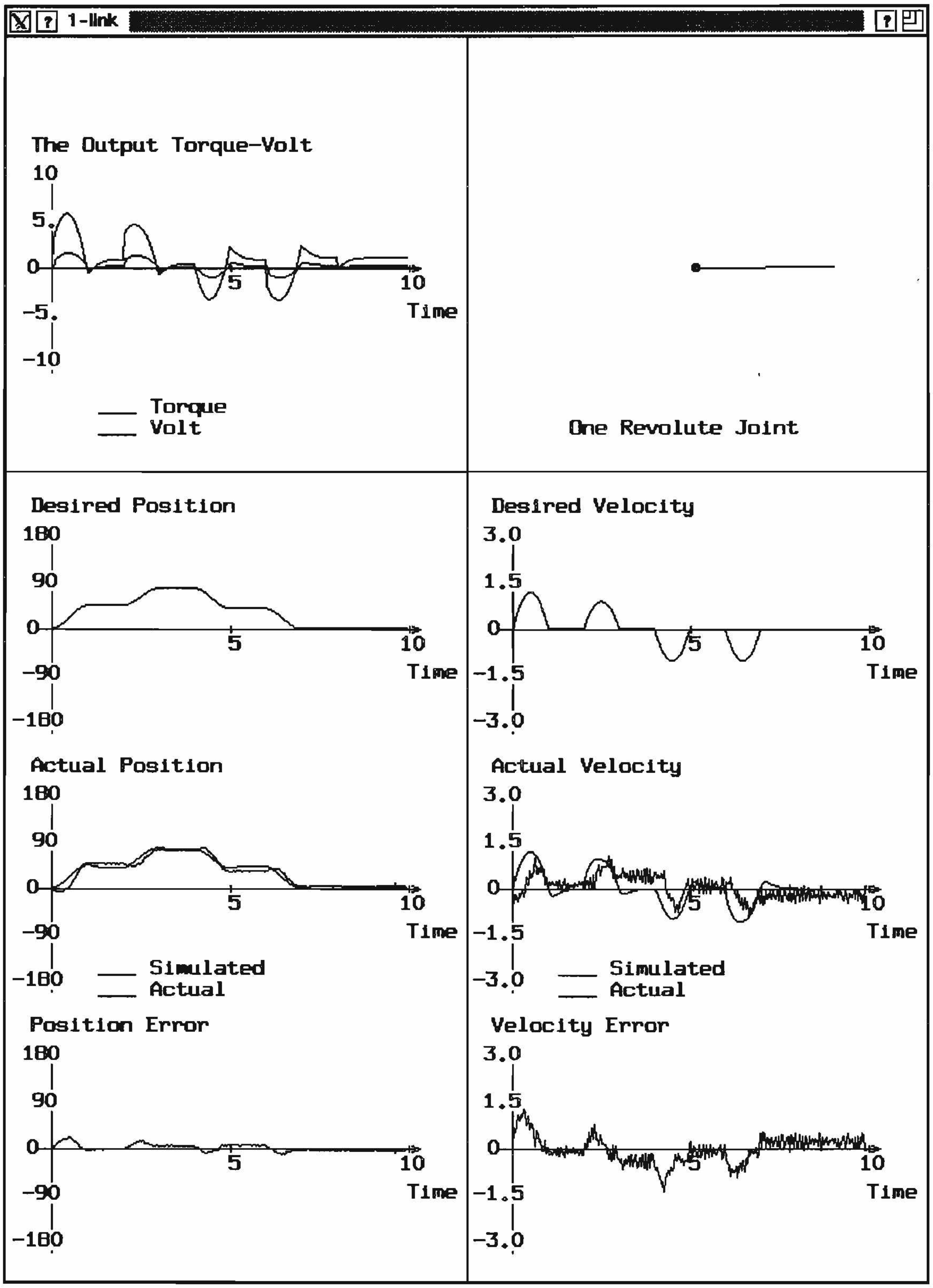


Figure 29: The behavior of the one-link robot for the second input sequence.

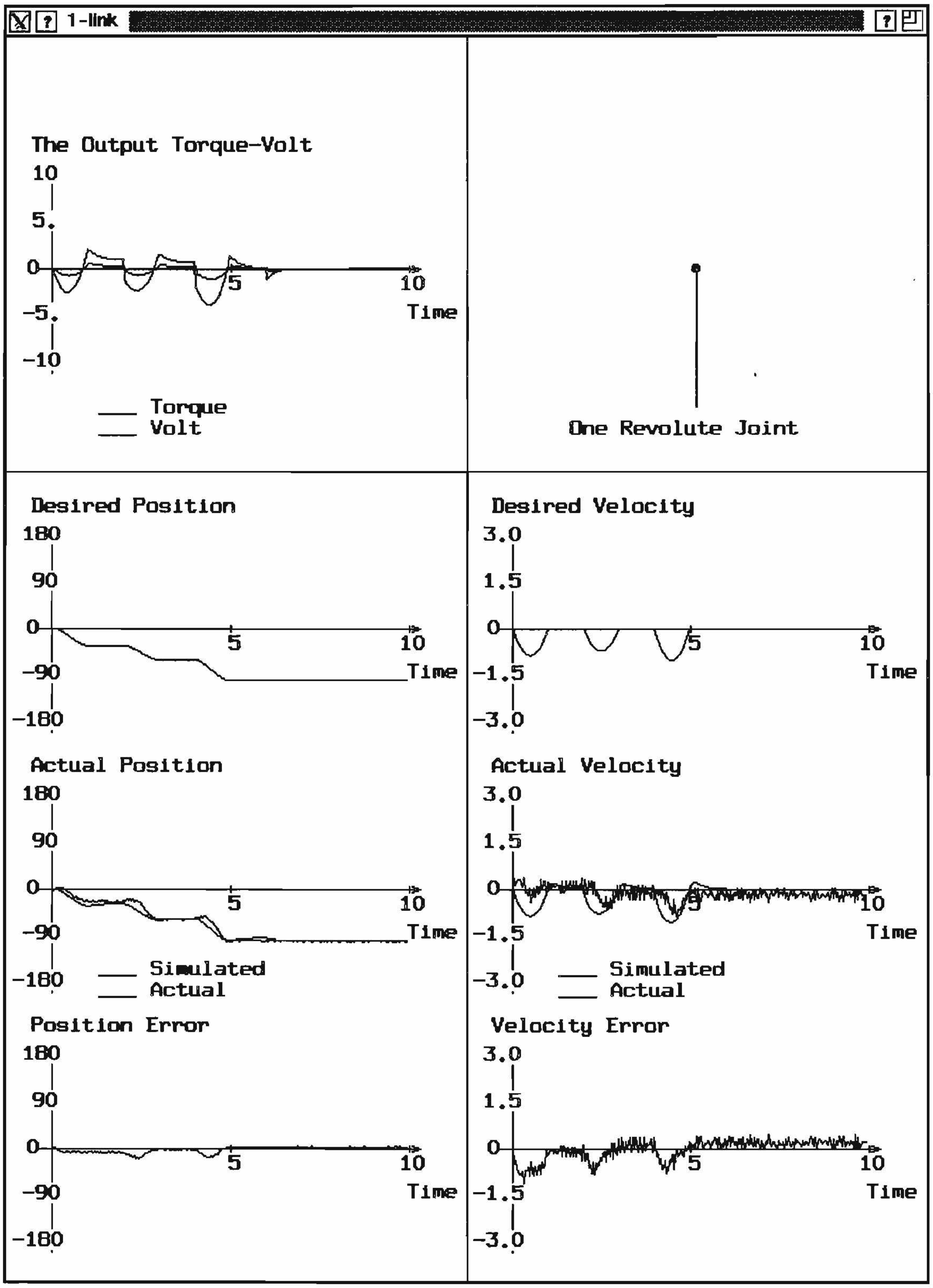


Figure 30: The behavior of the one-link robot for the third input sequence.

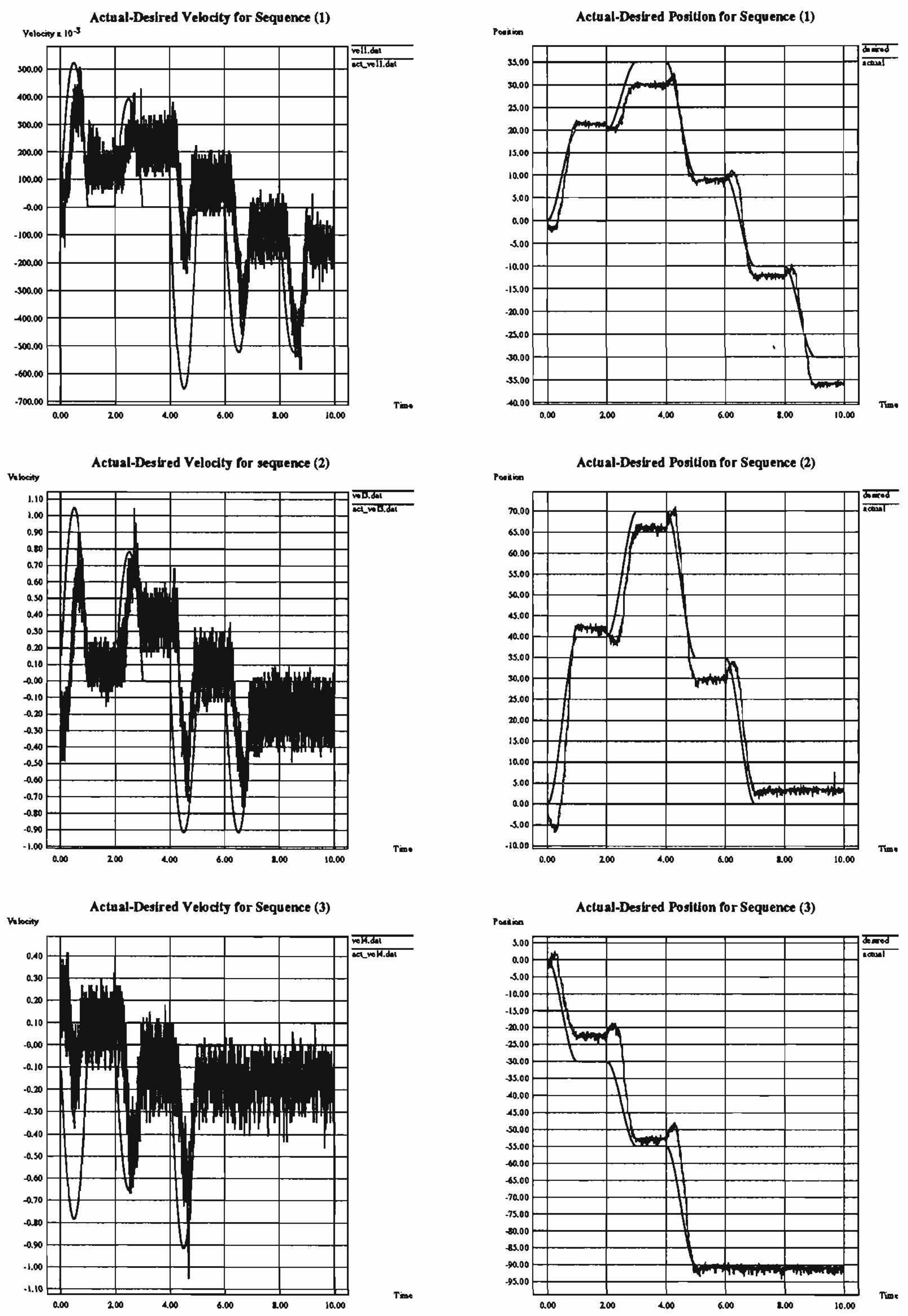


Figure 31: The difference between the actual and the desired behavior.

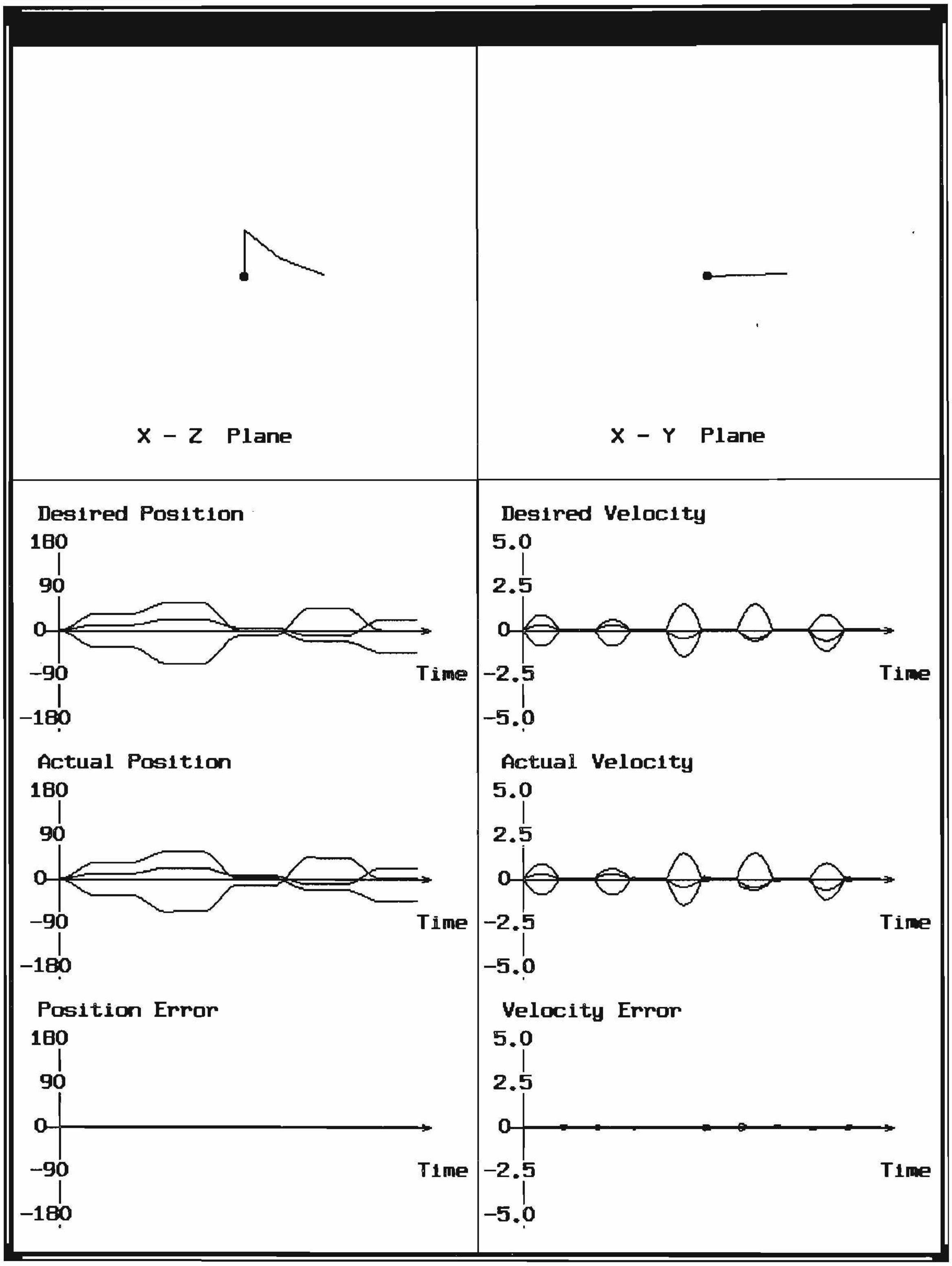
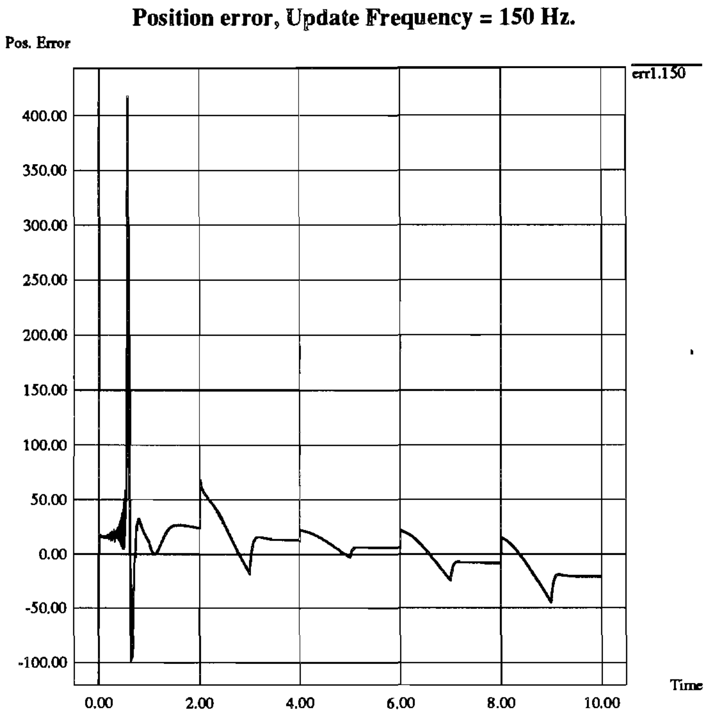


Figure 32: The output window of the simulation program for 3-link robot.



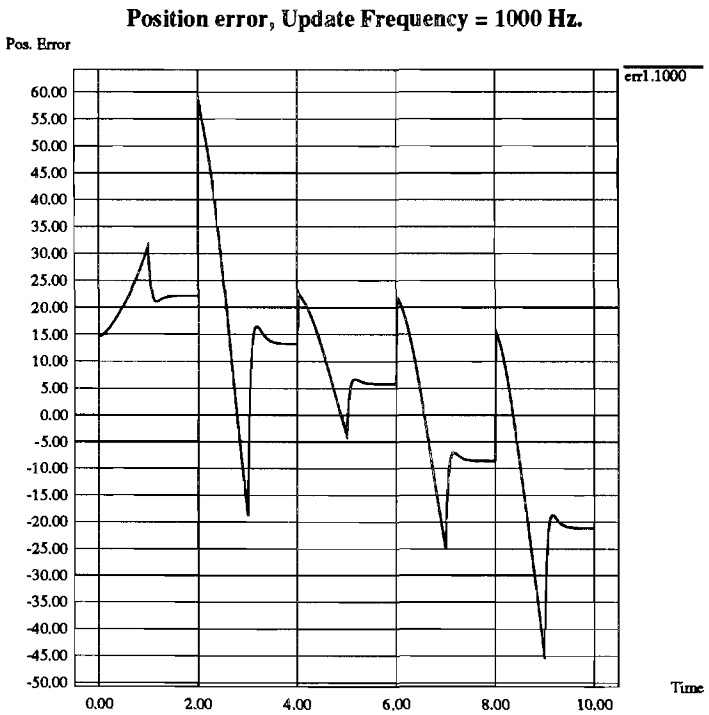


Figure 33: The effect of changing the update rate on the position error.

- integral gain (k_i)
- input trajectory
- update rate

In these experiments, we run the program 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 200 times per second which was good enough for our robot.

Figures 34, 35, 36, and 37 show the desired and actual position for different test cases using different feedback gains.

5.4 The Prototyping Environment

In this section, we will show several test cases for the prototyping environment. In the first test (Figure"38), the optimal design sub-system sent a data-change message to the CI. The CI in turn sent lock messages to all other sub-systems 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 constrains. In this test case the constraints were satisfied, so it the CI sent these changes to the sub-systems that needed to be notified. After that, the CI waited for acknowledgments from the sub-systems. In this case it received positive acknowledgments from the specified sub-systems. Finally, the CI updated the database and sent final acknowledgment messages to all sub-systems.

The second test case (Figure 39), was the same as the first case except that one of the sub-systems (the CAD/CAM sub-system) has rejected the changes by sending negative acknowledgment message to the CI. Thus, the CI sent a change-back message to the specified sub-systems, then it sent a final acknowledgment messages to all sub-systems. No changes in the database took place in this case.

In the last test case (Figure 40), the design constraints were not satisfied. Therefore, the CI sent a report about the non-satisfied constraints to the sender (the optimal design sub-system). Then it sent final acknowledgment messages to all sub-systems. Again, in this case, no changes in the database took place.

5.5 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

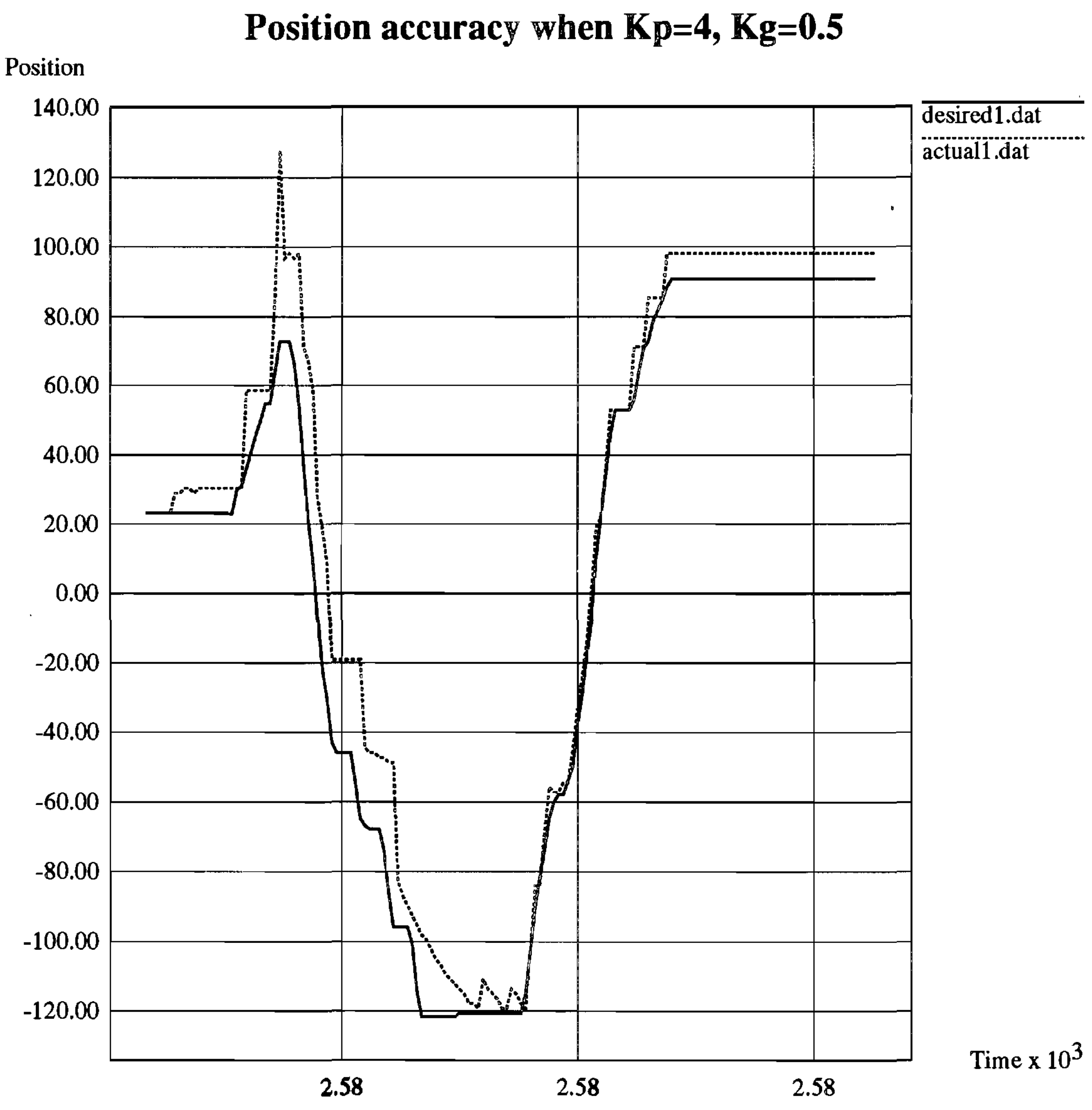


Figure 34: Desired and actual position for test case (1).

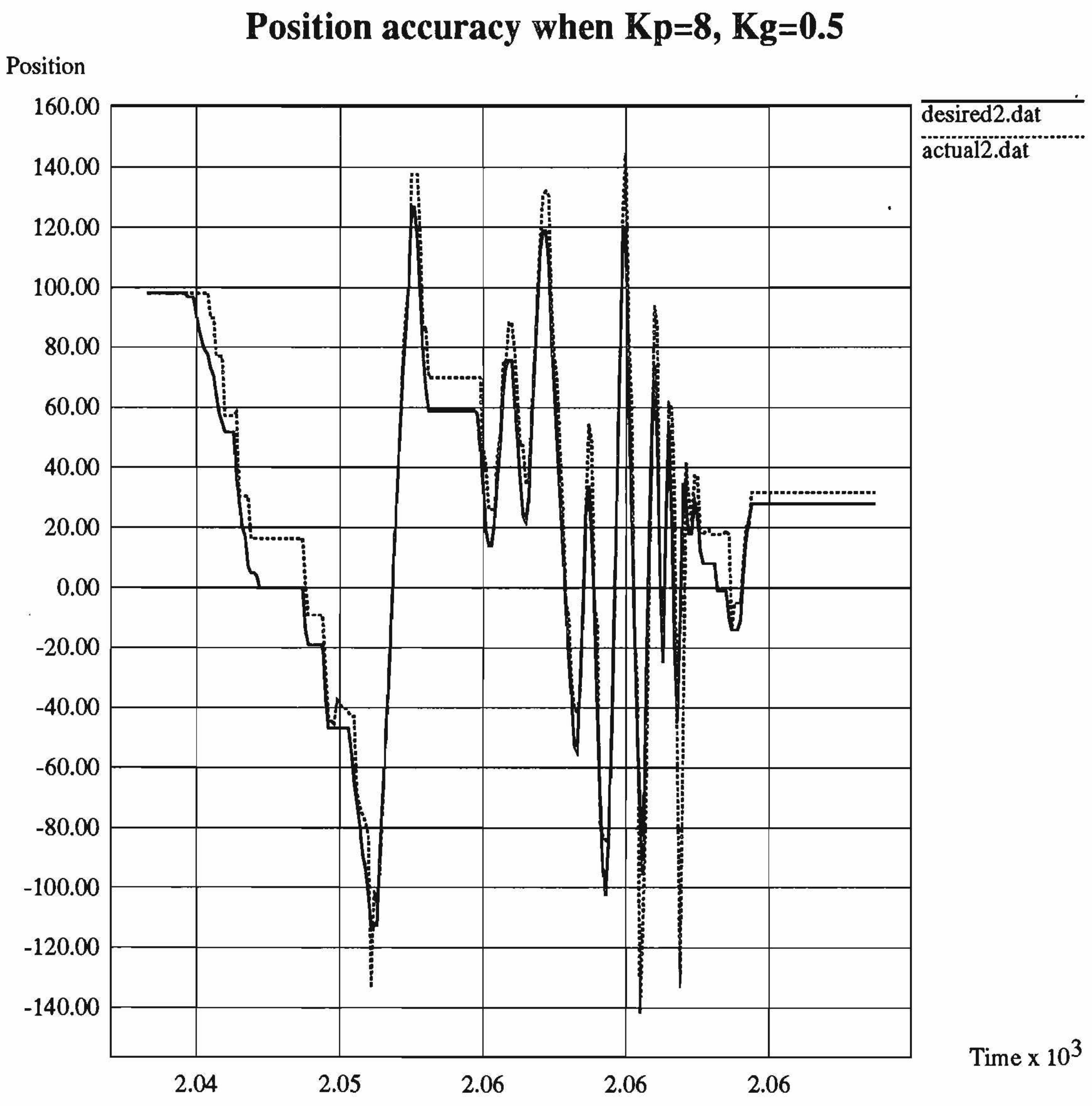


Figure 35: Desired and actual position for test case (2).

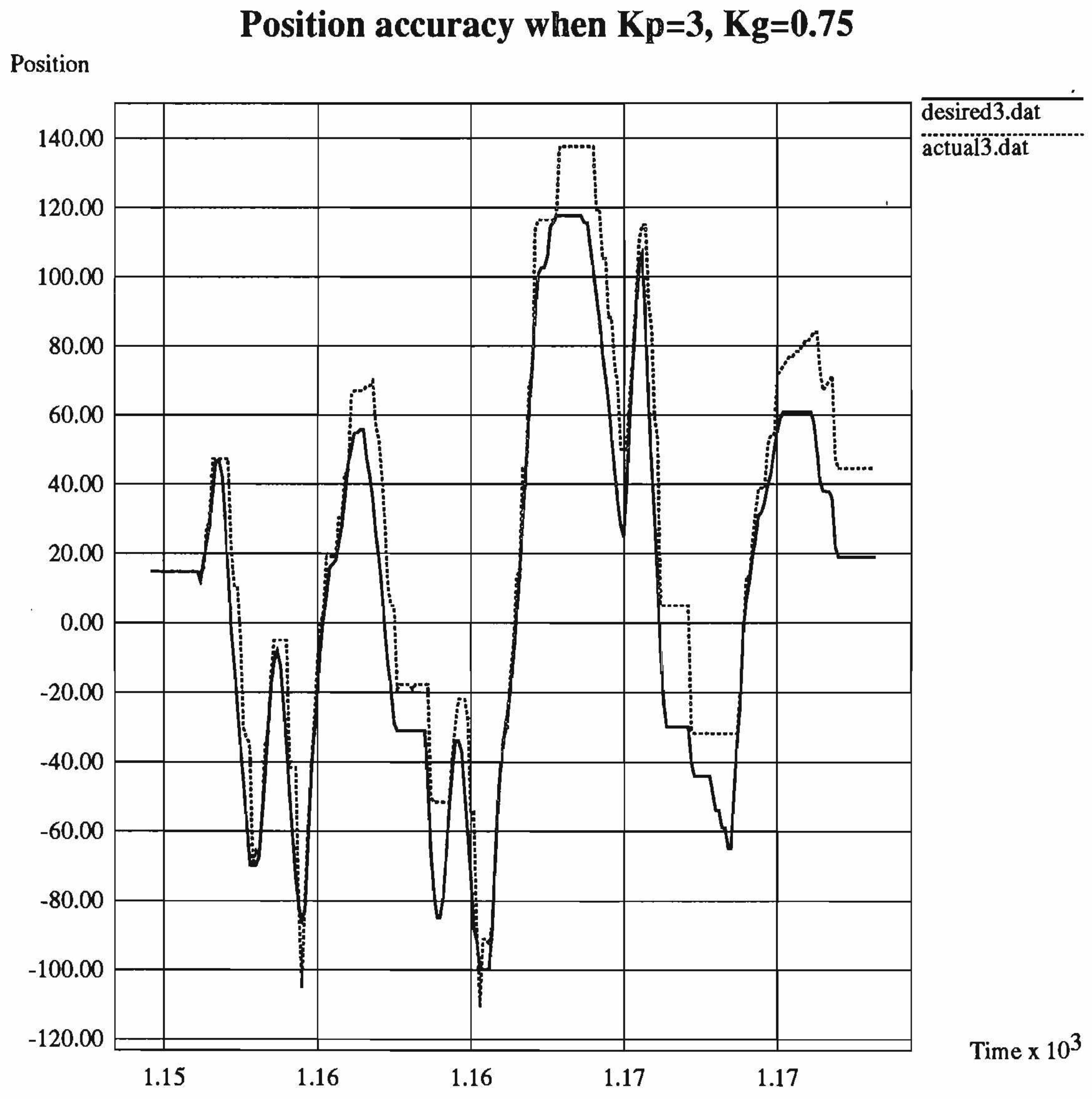


Figure 36: Desired and actual position for test case (3).

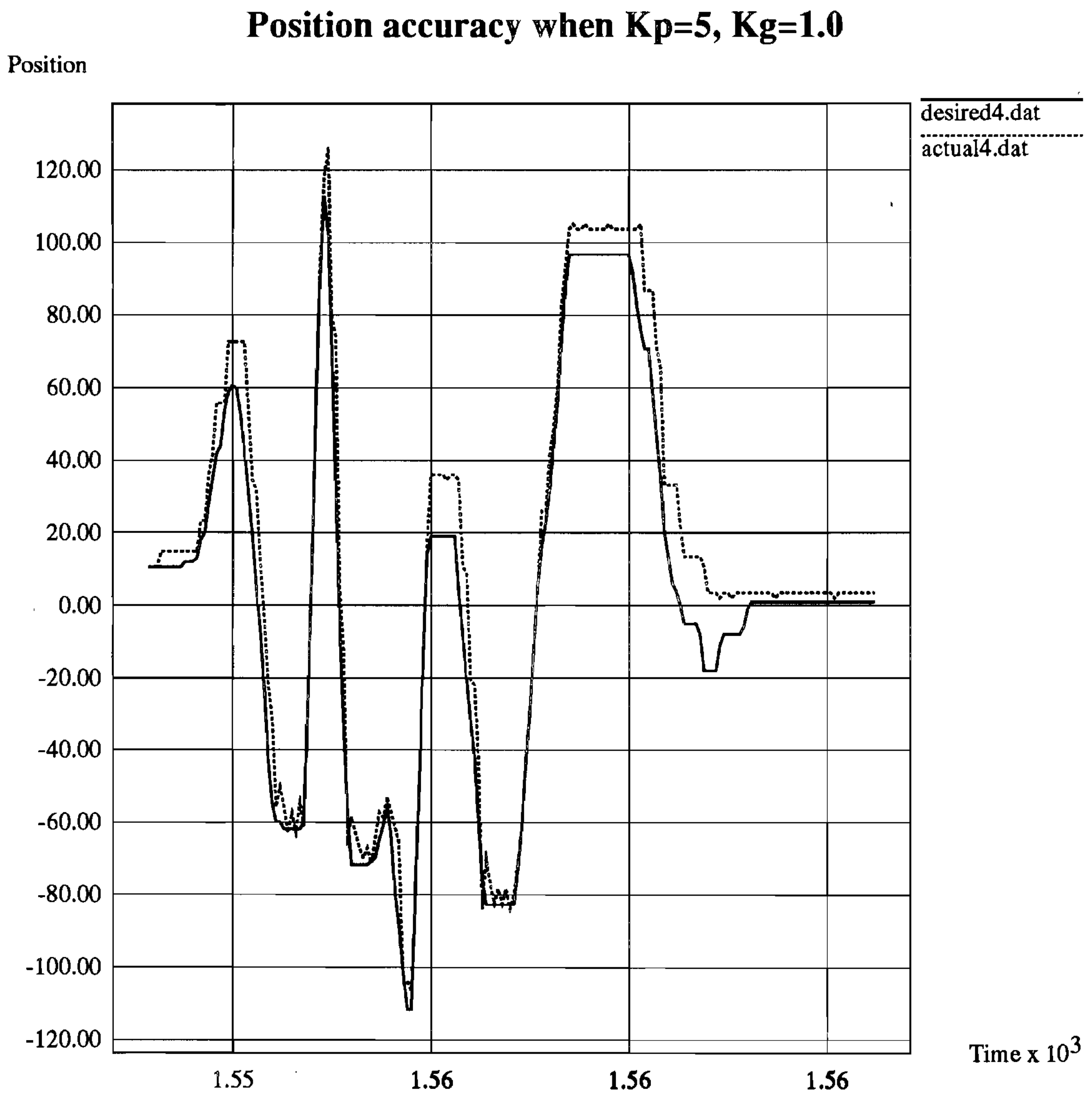


Figure 37: Desired and actual position for test case (4).

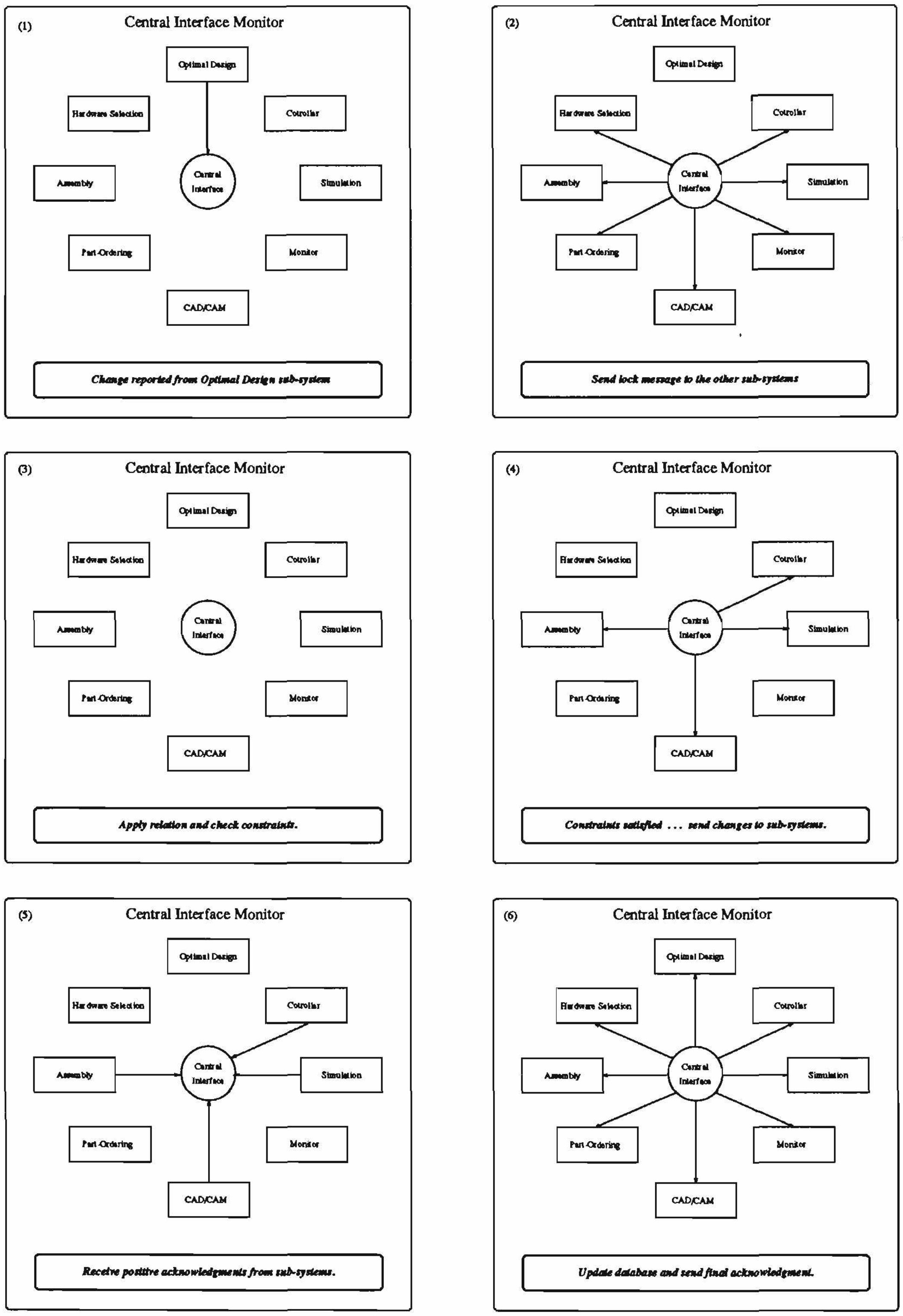

(1)
Central Interface Monitor
Optimal Design
Hardware Selection
Cotroller
Assembly
Central Interface
Simulation
Part-Ordering
Monitor
CAD/CAM
Change reported from Optimal Design sub-system
(2)
Central Interface Monitor
Send lock message to the other sub-systems
(3)
Central Interface Monitor
Apply relation and check constraints.
(4)
Central Interface Monitor
Constraints satisfied ... send changes to sub-systems.
(5)
Central Interface Monitor
Receive positive acknowledgments from sub-systems.
(6)
Central Interface Monitor
Update database and send final acknowledgment.

Figure 38: CI test case one, success case for data change.

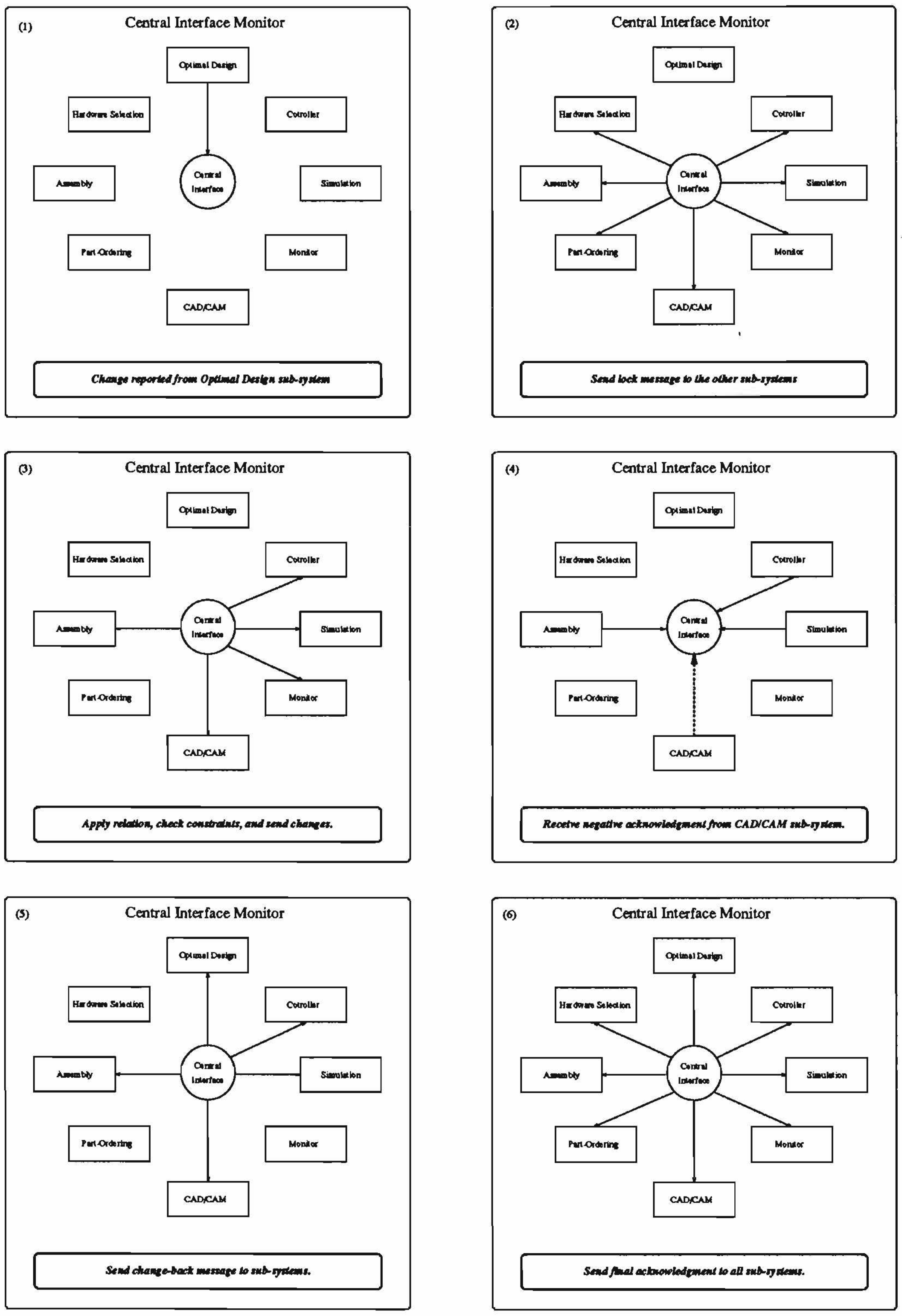

(1)
Central Interface Monitor
Optimal Design
Hardware Selection
Cotroller
Assembly
Central Interface
Simulation
Part-Ordering
Monitor
CAD/CAM
Change reported from Optimal Design sub-system
(2)
Send lock message to the other sub-systems
(3)
Apply relation, check constraints, and send changes.
(4)
Receive negative acknowledgment from CAD/CAM sub-system.
(5)
Send change-back message to sub-systems.
(6)
Send final acknowledgment to all sub-systems.

Figure 39: CI test case two, negative acknowledgment case.

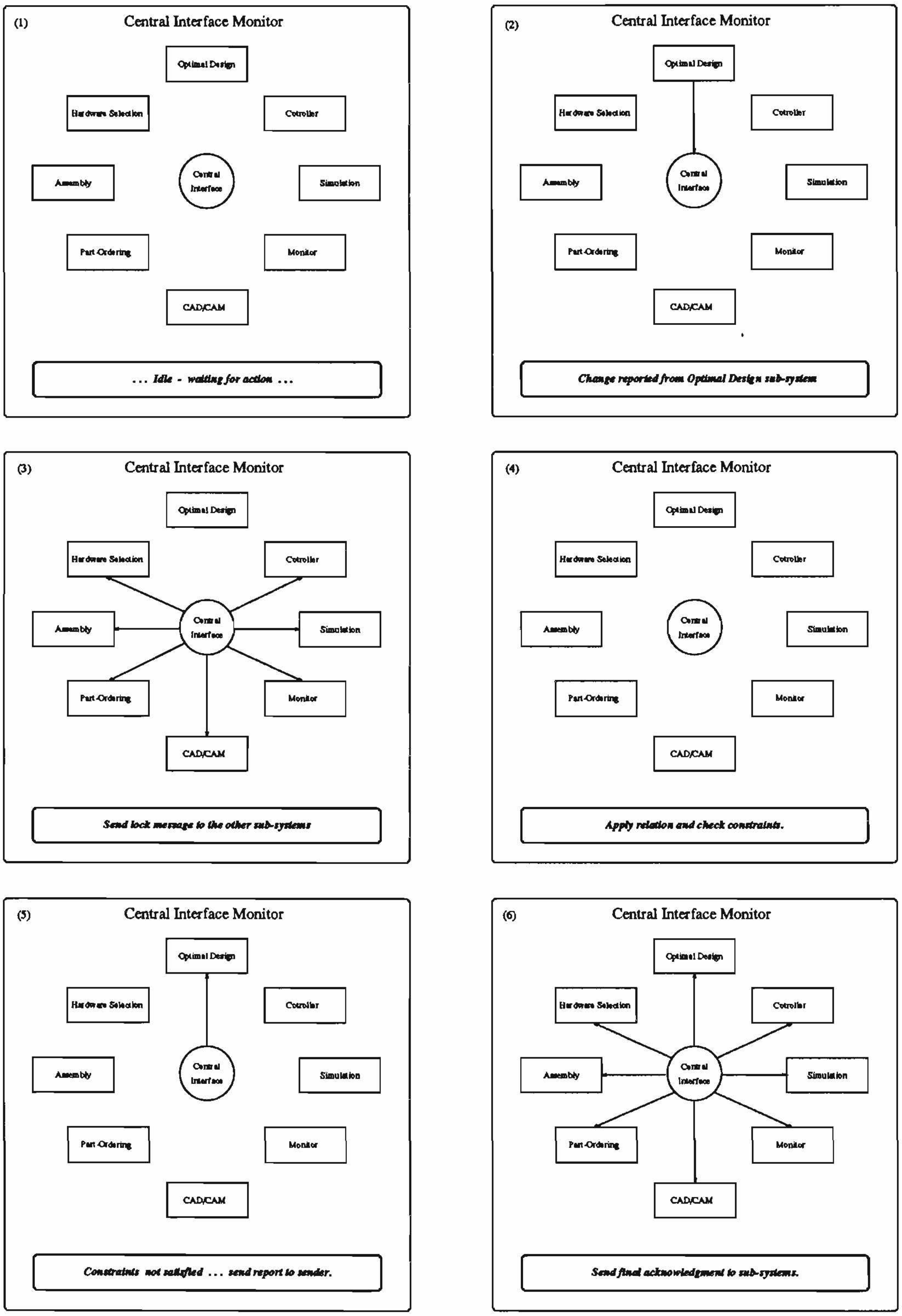

(1)
Central Interface Monitor
Optimal Design
Hardware Selection
Cotroller
Assembly
Central Interface
Simulation
Part-Ordering
Monitor
CAD/CAM
... Idle - waiting for action ...
(2)
Central Interface Monitor
Change reported from Optimal Design sub-system
(3)
Central Interface Monitor
Send lock message to the other sub-systems
(4)
Central Interface Monitor
Apply relation and check constraints.
(5)
Central Interface Monitor
Constraints not satisfied ... send report to sender.
(6)
Central Interface Monitor
Send final acknowledgment to sub-systems.

Figure 40: CI test case three, non-satisfied constraints case.

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 lake of communication between the different groups involved in the design. This lake 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 test and simulations after they knew about these changes. The PE can solve this problem since there is an SSI at each sub-system. This SSI will report any changes in the design parameters to the CI, which in turn will report these changes to all sub-systems that needs to know.

Another problem was selecting the necessary motors to drive the robot links, and satisfies 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 didn't know about this change until the assembly process was started. The part-ordering sub-system, 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 satisfies the design requirements. This information would have acknowledged the robotics and CAD/CAM groups with the necessity of adding gears at each joint earlier in the design phase.

The major problem we have faced in this project was communicating the robot with the work-station. The problem was that the communication rate was too low due to some protocol in the operating system of the Sun Station which waits until some buffer is filled or timeout occurred 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 make our testing. Also this problem can be solved using the part-ordering sub-system 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 non-automated SSIs. Currently, the SSI is just informing the user of any change in the design parameters and the user makes the changes in the local files at each sub-system. This process is subject to human errors and might yield a non-consistent situation. To solve that, all SSIs need to be automated so that the changes in the local data files of each sub-system is done automatically, and the user at each sub-system will be notified with this change.

The automation of the SSIs requires that the sub-systems 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 sub-system, because this will require changing the source code (which might not be available), and re-compiling the program each time we need to change any of the "hard-wired" parameters. For example, we can not use a simulation sub-system which 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 sub-systems 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.

6 Conclusions

A prototype 3-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.

The design bases for building a prototyping environment for robot manipulators was investigated and the design options were explained. A simple implementation of a central interface was done to demonstrate the functionality of the proposed environment. Also an initial model of an optimal design sub-system was started and is still under development.

6.1 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 sub-systems 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., AND LI, B. Optimal design of multiple arithmetic processor-based robot controllers. In *IEEE Int. Conf. Robotics and Automation* (1987), pp. 660–663.

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

[3] 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.

[4] 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.

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

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

[7] 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.

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

[9] 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).

[10] 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.

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

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

[13] 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.

[14] GRAHAM, J. H. Special computer architectures for robotics: Tutorial and survey. *IEEE Trans. Robotics and Automation 5*, 5 (Oct. 1989), 543–554.

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

[16] 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.

[17] 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.

[18] 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.

[19] Kawamura, S., Miyazaki, F., and Arimoto, S. Is a local linear pd feedback control law efficitive for trajectory tracking of robot motion? In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 1335–1340.

[20] 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.

[21] 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.

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

[23] Kung, S., and Hwang, J. Neural network architectures for robot applications. *IEEE Trans. Robotics and Automation 5*, 5 (Oct. 1989), 641–657.

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

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

[26] 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.

[27] Leung, S. S., and Shanblatt, M. Real-time dks on a single chip. *IEEE Journal of Robotics and Automation 3*, 4 (Aug. 1987), 281–290.

[28] Leung, S. S., and Shanblatt, M. A. Computer architecture design for robotics. In *IEEE Int. Conf. Robotics and Automation* (1988), pp. 453–456.

[29] 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.

[30] Lewis, F. L., Abdallah, C. T., and Dawson, D. *Control of Robot Manipulator.* Macmillan, 1993.

[31] 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.

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

[38] Motorola Inc. *MC68HC11E9 HCMOS Microcontroller Unit*, 1991.

[39] Nicol, J. R., Wilkes, C. T., and Manola, F. A. Object orientation in heterogeneous distributed computing systems. *IEEE Computer Magazine* (June 1993), 57–67.

[40] 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.

[41] Paul, B., and Rosa, J. Kinematics simulation of serial manipulators. *Int. J. Robotics Research 5*, 2 (Summer 1986), 14–31.

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

[43] 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.

[44] 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.

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

[46] 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.

[47] Sriram, D., and Logcher, R. The MIT dice project. *IEEE Computer Magazine* (Jan. 1993), 64–71.

[48] 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.

[49] 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.

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