Prototyping Environment for Robot Manipulators

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

UUSC-93-021

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

September 30, 1993

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. We are proposing a prototyping environment for electro-mechanical systems, and we chosen a 3-link robot manipulator as an example. In Designing a robot manipulator, the interaction between several modules (S/W, VLSI, CAD, CAM, Robotics, and Control) illustrates an interdisciplinary prototyping environment that includes different types of information that are radically different but combined in a coordinated way. This environment will enable optimal and flexible design using reconfigurable links, joints, actuators, and sensors. 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 for the robot modules (kinematics, inverse kinematics, dynamics, trajectory planning, analog control and computer (digital) 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. In this report a framework for flexible prototyping environment for robot manipulators is proposed along 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

1 Introduction **4**

2 Objectives **4**

3 Background and Review **6**

3.1 Phases of Building a Robot 6

3.2 Robot Modules and Parameters 6

3.2.1 Forward Kinematics 6

3.2.2 Inverse Kinematics 8

3.2.3 Dynamics 8

3.2.4 Trajectory Generation 10

3.2.5 Linear Feedback Control 11

3.3 Local PD feedback Control vs Robot Dynamic Equations 12

3.4 Speed Considerations 13

3.4.1 Types of Inputs 14

3.4.2 Desired Frequency of the Control System 14

3.4.3 Error Analysis 15

3.5 Special Computer Architecture for Robotics 15

3.6 Optimal Design of Robot Manipulators 17

3.7 Integration of Heterogeneous Systems 19

4 Three link manipulator **20**

4.1 Analysis Stage 20

4.2 One Link Manipulator 20

4.3 Sensors and Actuators Interface 22

4.4 Controller Design 27

4.5 Simulation 28

4.6 Benchmarking 28

4.7 PID controller Simulator 30

4.8 Building the Robot 30

5 The Optimal Design Sub-system **33**

5.1 Performance Specification Interface 33

5.2 Constructing the Optimization Problem 33

5.3 The User Interface 36

5.4 Some Design Examples 36

6 The Interface Layer **40**

6.1 Interaction Between Sub-systems 44

6.2 The Interface Scheme 49

6.3 Interface Design 50

7 Current Development **52**

8 Conclusion **52**

List of Figures

1 Interaction Between Sub-systems in the Prototyping Environment. 5
2 Robot design Phases 7
3 High-level block diagram of a robot control system. 12
4 Different types of damping in a second order control system. 12
5 Block diagram of the Controller of a Robot Manipulator. 13
6 Sampling Theory and the Aliasing Problem. 15
7 The Error When the Update Frequency is 150 Hz. 16
8 The Relation Between the Torque and the Voltage. 21
9 Circuit Diagram of the DC-motor Used in the Experiment 22
10 The Output Window of the Simulation Program for Sequence One. 23
11 The Output Window of the Simulation Program for Sequence Two. 24
12 The Output Window of the Simulation Program for Sequence Three. 25
13 The Difference Between the Actual and the Desired Behavior. 26
14 The three different configurations of the robot manipulator. 27
15 Performance comparison for different platforms: The machines from left to right are: SPARC 2, SPARC 10-30, SPARC 10-41, and HP 700. For each machine, there are three columns; before simplification, after using Mathematica, and after simplifying the trigonometric functions. 30
16 The Interface Window for the PID Controller Simulator. 31
17 The Physical Three-Link Robot manipulator 32
18 Controlling the robot using different schemes 33
19 Schematic View for the Robot Prototyping Environment. 41
20 The Interface Between the Subsystem and the Prototype Robot. 43
21 Examples of Some Interaction Cycles. 50
22 Three Different Ways for sub-system interfaces communication. 51

List of Tables

1 Number of calculation involved in the dynamics module. 28
2 Configuration of the Machines Used in the Benchmark. 29
3 The interaction between the user and the sub-systems. 46
4 The interaction between the optimal design and the other sub-systems. 47
5 The interaction between CAD/CAM and the other sub-systems. 48
6 The interaction between hardware and other sub-systems. 48

1 Introduction

In designing and building an electro-mechanical system, such as robot manipulators, a lot of 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 the required software and hardware systems and modules which are independent of the design parameters, so that it can be used for different configurations and varying parameters. This environment will be 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 methodology. On the other hand, there is a lot of shared information 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 these heterogeneous sub-systems to take place.

Figure 1 shows the interaction between some of those sub-systems. The optimal design system affects the control and the simulation systems. The monitor sub-system takes its data from the simulator and from the robot. There is also feedback information from the monitor to the optimal design system to refine the design according to the performance measurements for each design. The robot is derived by the control system, and feedback information goes from the robot sensors to the control system.

2 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

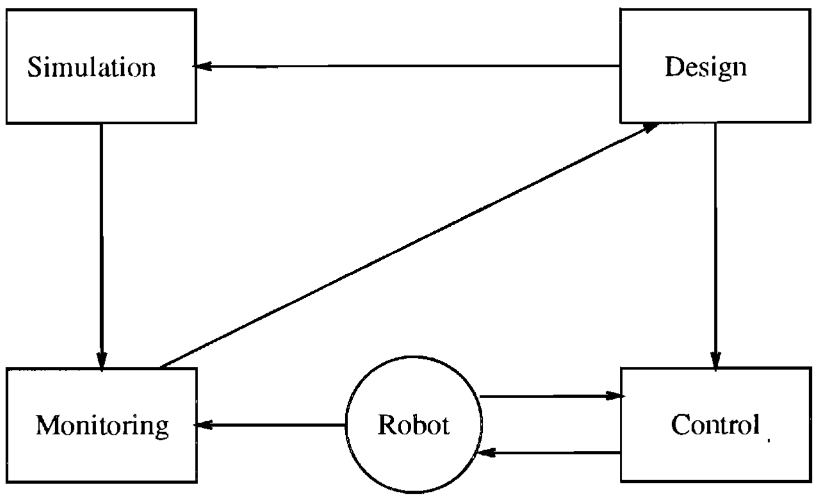


Figure 1: Interaction Between Sub-systems in the Prototyping Environment.

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 we aim for 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 provide 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.
- The optimal design system will speed up the design cycle will supply robot designers with a handy tool that enables them chose the suitable configuration and the appropriate parameters for a specified task. By using the simulation program they will be able to test and analyze the behavior of the robot under each selection of the configuration.
- This project will facilitate the cooperation of several groups in our Computer Science 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.
- The interdisciplinary nature of the proposed research will provide an exceptional educational environment for those involved in the work.

This report is divided into four parts: first, a brief background for robot design and modules is presented with the related work in this area. Second, a description for the prototyping and simulation of a 3-link robot manipulator is presented with some results from the completed phases of that part, then, the optimal design of robot manipulators is discussed and the proposed optimal design system is described and investigated. Finally, The interface between the systems is discussed along with the required representations (knowledge base, object oriented scheme, rule-based reasoning, etc.).

3 Background and Review

3.1 Phases of Building a Robot

We can divide the process of building a robot into several phases as follows:

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.

Simulation Phase: test the behavior and the performance of the chosen manipulator.

Prototyping and Testing Phase: test the behavior and performance, and compare it with the simulated results.

Manufacturing Phase: order the required parts and manufacture the actual robot.

3.2 Robot Modules and Parameters

Controlling and simulating a robot is a process that involves a large amount of mathematical equations. To be able to deal with the required computations, it is better to divide them into modules, where each module accomplishes a certain task. The most important modules, as described in [4], are: kinematics, inverse kinematics, dynamics, trajectory generation, and linear feedback control. In the following sections, we will describe briefly each of these modules, and the parameters involved in each.

3.2.1 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 rotation matrix; or using the *joint space*, 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 link, etc.). In order to describe the location of each link relative to its neighbor, a frame

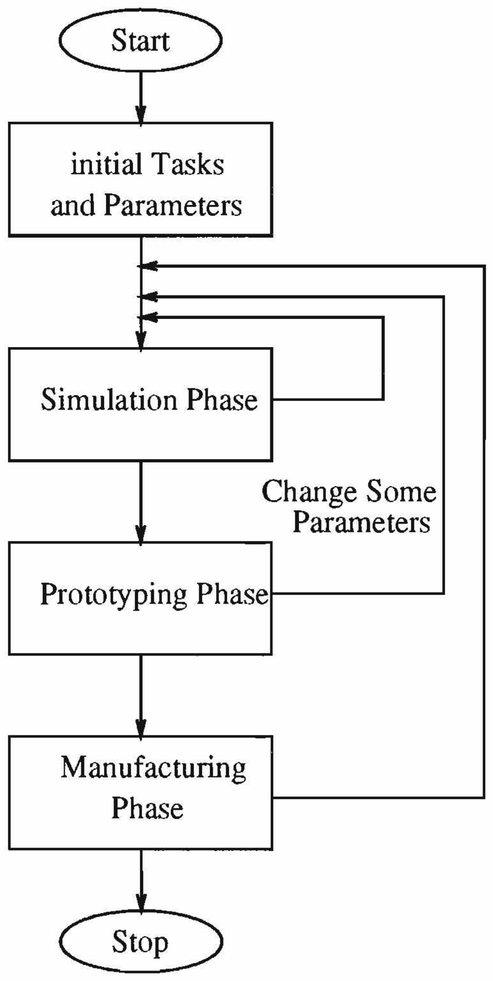


Figure 2: Robot design Phases

is attached to each link, then we specify a set of parameters that characterizes this frame. This representation is called the *Denavit-Hartenberg notation.* See [4] for more details.

One approach to the problem of kinematics analysis is described in [34], 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.

3.2.2 Inverse Kinematics

This module solves for the joint angles given the desired position and orientation in the Cartesian space. This is a more complicated 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 the volume of space which the end-effector of the manipulator can reach. On the other hand, the problem of multiple solutions forces the designer to set a criterion for choosing one solution. A good choice would be 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 solution* and *numerical solutions.* Numerical solutions are much slower than closed form solutions, but, sometimes it is too difficult to find the closed form solution for some configurations. In our case, we will use closed form solutions, since our models are three link manipulators with easy closed form formulae.

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 [13]. The input to this package is the Denavit-Hartenberg parameters, and the output is the direct and inverse kinematics solutions. Another method to find a symbolic solution for the inverse kinematics problem was proposed in [35]. Kelmar and Khosla proposed a method for automatic generation of forward and inverse kinematics for a reconfigurable manipulator system [16].

3.2.3 Dynamics

Dynamics is the study of the forces required to cause the motion. In other words, it is concerned with the way in which motions of the manipulator arise from torques applied by the actuators, or from external forces applied to the manipulator. The serial chain nature of manipulators makes it easy to use simple methods in dynamic analysis that utilizes this nature.

For most manipulators, the parameters that can be directly controlled are the joint forces and/or torques., however, the variables to be controlled are the position and the position of the end-effector. Thus, dynamic analysis is to find the forces and/or torques that should be applied at each joint to produce the desired trajectory of the end-effector.

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 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 later is used to simulate a manipulator mathematical model before building the physical model, which gives the chance to update and modify the design without the cost of changing or replacing any physical part.

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.
- Link type (revolute or prismatic).
- Manipulator configuration and joints 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:

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

Once we have the acceleration $\ddot{\theta}$, we can calculate the velocity $\dot{\theta}$ and the position θ using simple numerical integration technique. Given initial conditions for the position and velocity, usually in the form:

$$\theta(0) = \theta_0,$$
$$\dot{\theta}(0) = 0,$$

We can integrate forward in time by steps of size Δt. Using *Euler integration*, starting with $t = 0$, we can compute θ and $\dot{\theta}$ iteratively as follow:

$$\dot{\theta}(t + \Delta t) = \dot{\theta}(t) + \ddot{\theta}(t)\Delta t,$$

$$\theta(t + \Delta t) = \theta(t) + \dot{\theta}(t)\Delta t + \tfrac{1}{2}\ddot{\theta}(t)\Delta t^2$$

Choosing a value for Δt depends on several factors such as:

- The required frequency for simulating the robot.
- The natural frequency of the manipulator. From sampling theory, the sampling frequency should be at least twice as much as the natural frequency of the system.

- the maximum speed available to do the required calculations. This is the most important issue when choosing between software and hardware solutions. We will discuss this point in detail in a later section.

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. This issue will be discussed in more detail in a later section.

There are some parallel algorithms to calculate the dynamics of a manipulator. One approach described in [26], 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 [20, 21, 33] based on a multiprocessor controller, and pipelined architectures to speed up the calculations. Hashimoto and Kimura [12] 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 [25] 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 [14]. 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.

3.2.4 Trajectory Generation

This module computes a trajectory in multidimensional space which describes the motion of the manipulator. *Trajectory* is the time history of position, velocity, and acceleration for each of the manipulator's links. 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 will talk about this point when we talk about the speed consideration.

There are several strategies to calculate trajectory points which generate a smooth motion for the manipulator. It's important to guarantee this smoothness of the motion because of some 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 using *cubic polynomials*, which assumes a cubic function for the angle of each link, then by differentiating this equation, the velocity and acceleration are computed. This will give linear acceleration which provides smooth motion. The problem is to calculate the coefficients of the cubic polynomial equation. A cubic equation has the form:

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3,$$

and so the joint velocity and acceleration along this path are:

$$\dot{\theta}(t) = a_1 + 2a_2 t + 3a_3 t^2,$$
$$\ddot{\theta}(t) = 2a_2 + 6a_3 t.$$

By stating the initial and final positions and velocities, we can calculate the required coefficients as follows:

$$\theta(0) = \theta_0,$$
$$\theta(t_f) = \theta_f,$$
$$\dot{\theta}(t_f) = 0,$$
$$\dot{\theta}(0) = 0.$$

where, t_f is the time to move from θ_0 to θ_f.

Substituting these values and solving for the a_i we obtain:

$$a_0 = \theta_0,$$
$$a_1 = 0,$$
$$a_2 = \frac{3}{t_f^2}(\theta_f - \theta_0),$$
$$a_3 = -\frac{2}{t_f^3}(\theta_f - \theta_0).$$

3.2.5 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 3 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

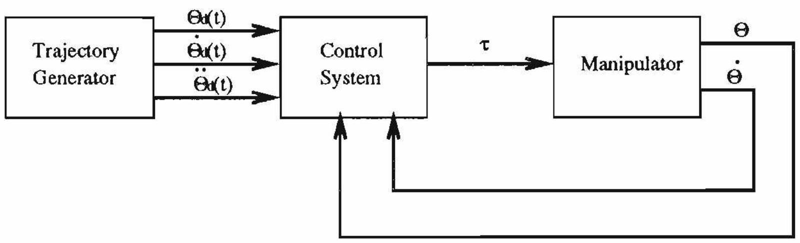


Figure 3: High-level block diagram of a robot control system.

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

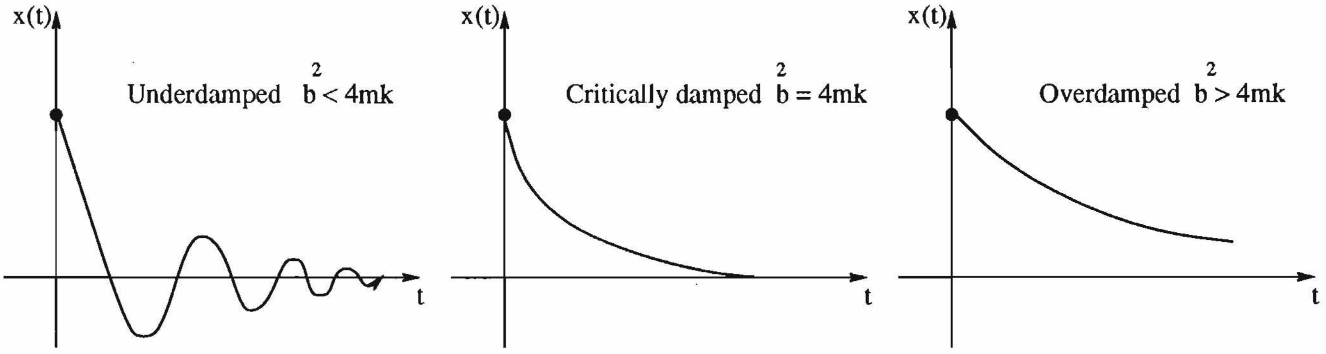


Figure 4: Different types of damping in a second order control system.

Figure 5 shows a block diagram for the controller, and the role of each of the robot modules in the system.

3.3 Local PD feedback Control vs Robot Dynamic Equations

Most of the feedback algorithms used in the current control systems is a 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.

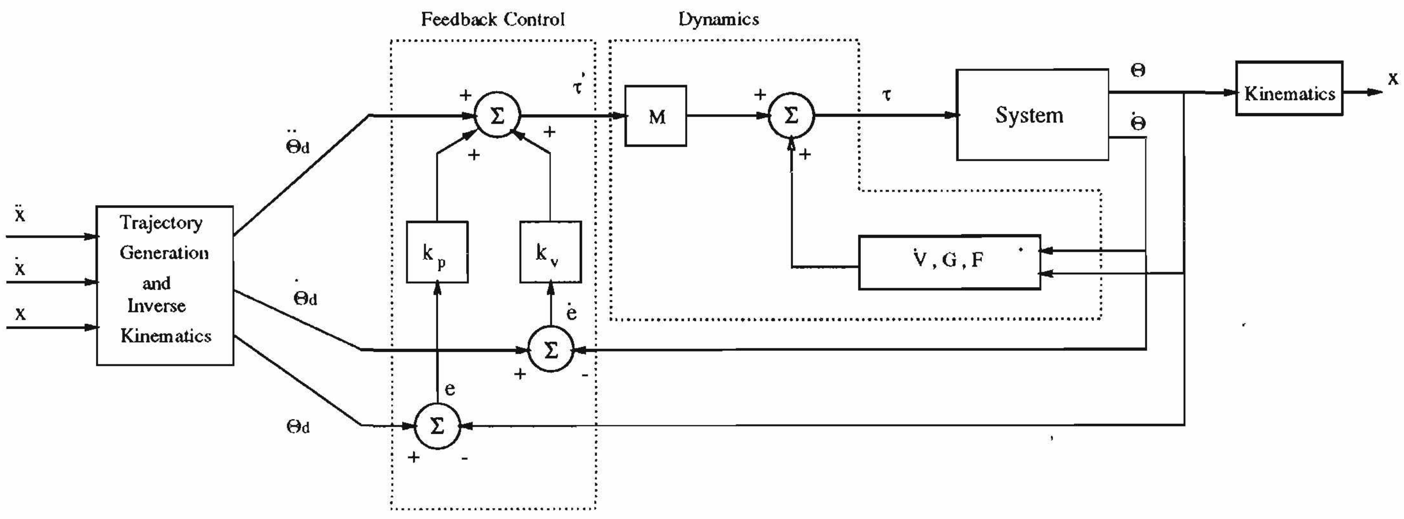


Figure 5: Block diagram of the Controller of a Robot Manipulator.

- 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.

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

- It needs high update rate to achieve reasonable accuracy.
- To simulate the robot manipulator behavior the dynamic equations should be used.
- 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 has 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 [2]. Another method is build an inner loop stabilizing controller using a multi-variable PD controller, and an outer loop tracking controller using a multi-variable PID controller [40].

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 [15].

3.4 Speed Considerations

There are several factors which affect the desired speed (frequency of calculations), the maximum speed we can get 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.

3.4.1 Types of Inputs

The user interface to that system should allow the user to specify the desired motion of the manipulator with different ways depending on the nature of 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.

3.4.2 Desired Frequency of the Control System

We have to decide on the required frequency of the system. In this system there are 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 have to consider different types of noise affecting the system such as: input noise, system noise, and output noise (from the sensors).

From *digital control theory*, to be able to restore the original function after sampling, the sampling frequency should be at least twice as much as the highest frequency in the function. So, if we assume that the system can be approximated by the second order differential equation:

$$s^2 + 2\omega_n + \omega_n^2,$$

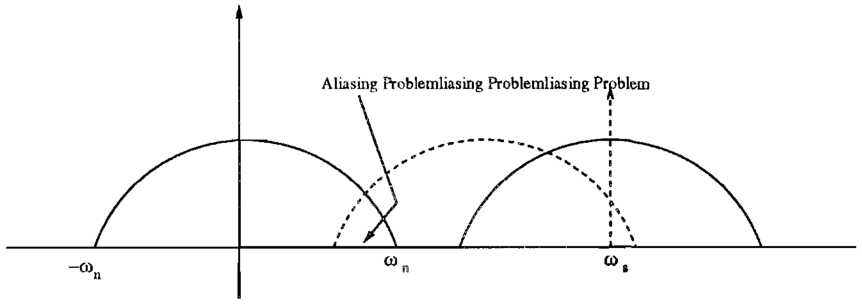


Figure 6: Sampling Theory and the Aliasing Problem.

where ω_n is the natural frequency of the system, then, the sampling frequency ω_s should be:

$$\omega_s \geq \omega_n.$$

Figure 6 shows the function in the frequency domain, and the problem when ω_s is less than $2\omega_n$, *the aliasing problem.*

The problem is to calculate the natural frequency of the control system to be able to decide on the value of the minimum update frequency required for that system. This problem can be solved analytically by the finite element theory for computing the natural frequency of a mechanical system among other methods. But this is beyond the scope of our discussion here. Another approach is by selecting values for the update frequency, and measure the error generated using each value, and choosing the value that causes an acceptable error. This approach is not very accurate, but for large systems, using the analytical method is very complicated and some times is impossible to solve.

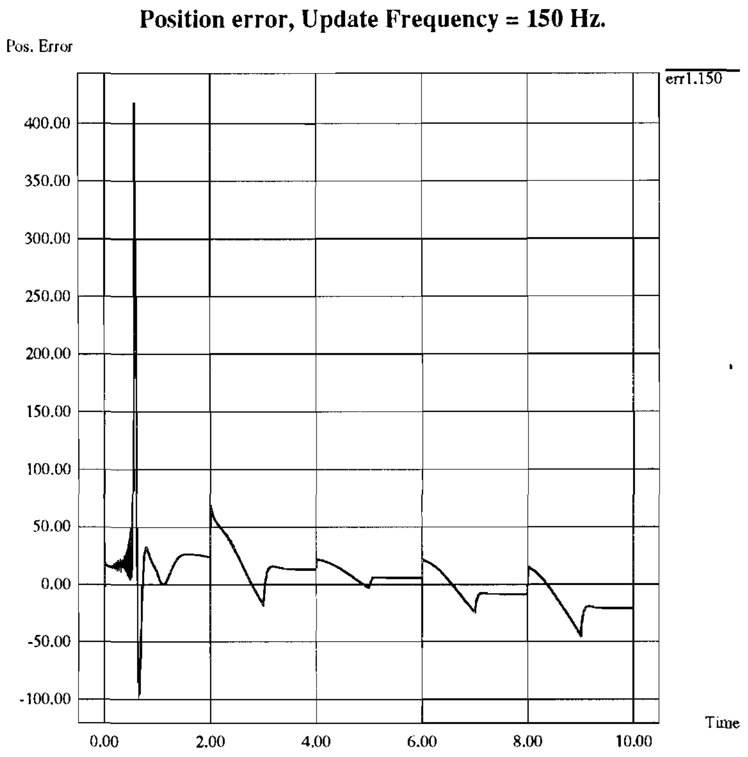
3.4.3 Error Analysis

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. Figure 7 shows the error in position of a three link manipulator using two different update frequencies.

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 [18] and showed that increasing the update rate decreases the error.

3.5 Special Computer Architecture for Robotics

When we design real time systems that involves a huge number of floating point calculations, the speed becomes an important issue. In such situations, a hardware solution might be used to achieve



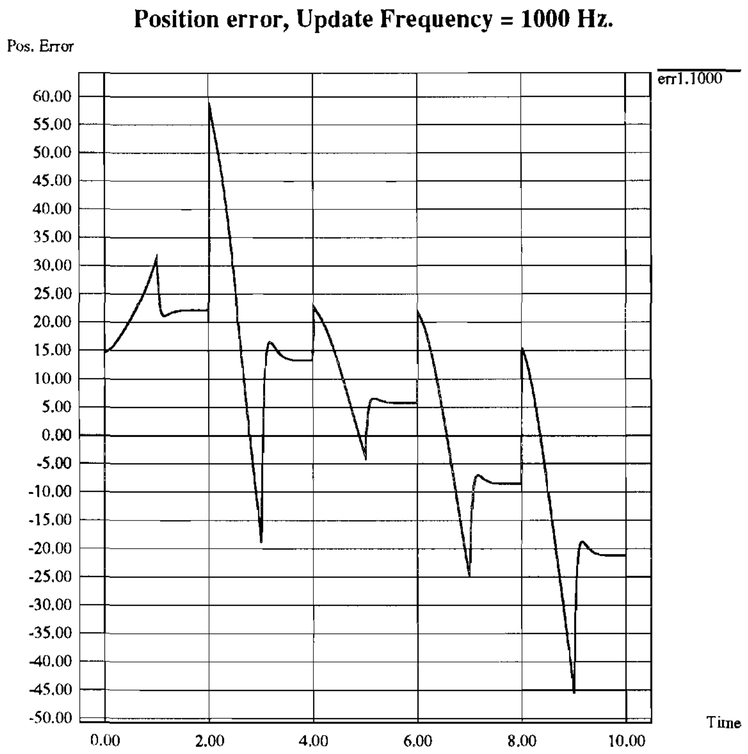


Figure 7: The Error When the Update Frequency is 150 Hz.

the desired speed. Graham [11] 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 [23] 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 put a hierarchy for the computational needs in robotics applications which is composed of: *management, reasoning*, and *device interaction.*

The concept of ASIC (Application-Specific Integrated Circuit) has created great opportunities for implementing robotic control algorithms on VLSI chips. In [24] 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.

A VLSI architecture designed to compute the direct kinematic solution (DKS) on a single chip is described in [22]. 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 [9].

3.6 Optimal Design of Robot Manipulators

It is important to choose the parameters of a robot manipulator (configuration, dimensions, motors, etc.) that are most suitable for the required robot tasks. A lot of research has been done in this area. Depkovich and Stoughton [6] proposed a general approach for the specification, design and validation of manipulators. A reconfigurable modular manipulator system was developed at Carnegie Mellon University which provide a viable alternative configuration manipulator [17].

Designing an *optimal* manipulator is yet not 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 [3, 29, 30]. Another criterion is to achieve optimal dynamic performance; that is to select the link lengths and actuator sizes for minimum time motions along a specified trajectory [27, 36].

TOCARD (Total Computer-Aided Design System of Robot Manipulators) is a system designed by Takano, Masaki, and Sasaki [39] 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 given 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
- Payload
- 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

3.7 Integration of Heterogeneous Systems

To integrate the work among the different teams and sites working in such a big project, there must be some kind of synchronization to facilitate the communication and cooperations between them. A concurrent engineering infrastructure that encompasses multiple sites and subsystems called Pallo Alto Collaborative Testbed (PACT), was proposed in [5]. The issues discussed in this 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 [1]. 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 [32]. 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 [28], and a system for design, process planning and inspection is presented.

Several important themes in concurrent software engineering are examined in [7]. Some of these themes are:

Tools: Specific tool 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 [8]. 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 [38]. 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 [19], such as, the lifetime of a software system, Analysis and design, module interfaces and implementation, and system testing and verification.

4 Three link manipulator

As part of this research project, a three-link robot manipulator was designed along with its controller and simulator. This enabled us determine the required sub-systems and interfaces for such environment. This prototype robot will be used as an educational tool in the control and robotics classes.

4.1 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 specify the specifications of the robot for performing various tasks, it also helped us determine which parts (algorithms) should be hardwired to achieve specific mechanical performances, and also how to supply the control signals efficiently and at what rates.

4.2 One Link Manipulator

Controlling a one-link robot in a real-time manner is not too 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 might arise in a larger environment. The idea here 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

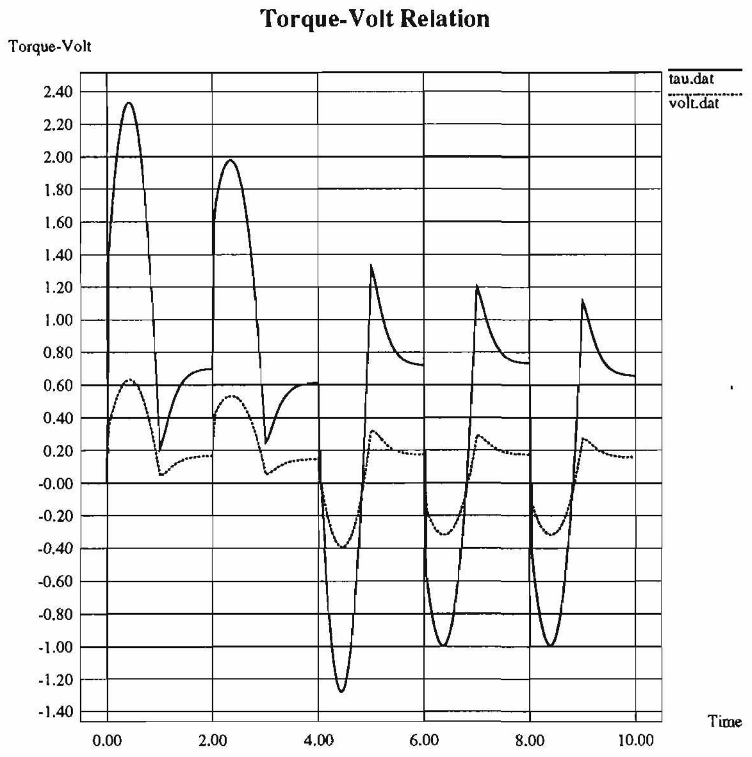


Figure 8: The Relation Between the Torque and the Voltage.

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 is 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 difficult to make an exact model. Figure 9 shows the circuit diagram of the motor and its parameters.The transfer function is in the form:

$$v(t) = \frac{T_m}{K_T}R + \frac{\dot{T_m}}{K_T}L + K_E\dot{\theta},$$

where,

$V(t)$ is the voltage at time t.
K_T is the torque from the motor.
$L = 13.4 \times 10^{-3}H$.
$R = 4.96\Omega$.
$K_T = 20.8 oz.in.sec^2$.
$K_E = 0.147 v/rad/sec$.

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

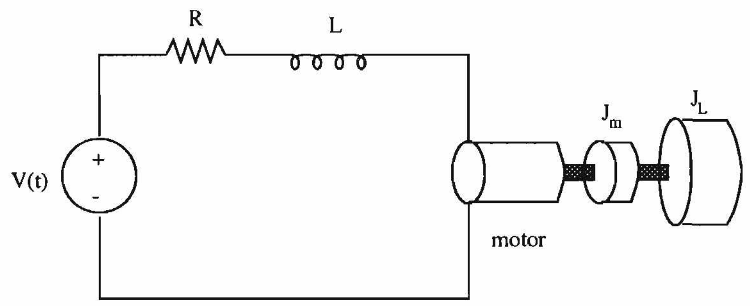


Figure 9: Circuit Diagram of the DC-motor Used in the Experiment

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 10, 11, and 12 show the output window displaying the link and graphs for the position and the velocity.

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 [37].

4.3 Sensors and Actuators Interface

This is an essential part of the project. It is concerned with the communication between the manipulator and the workstation used to control it. This part has been done by Anil Sabbavarapu, a graduate student in the Computer Science Department. Basically, we have three D/A lines that transmit the required torque (or voltage) from the workstation to the three manipulator's actuators, and we have six A/D lines that transmit the sensors readings at each joint to the workstation (three for the position, and three for the velocity). These readings are used in the controller for feedback information.

The problem requires interfacing a workstation with the motors and the sensors, so that a resident program on the workstation can send out 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 microcontroller system (the MC68HC11EVBU - Universal Evaluation Board) which has a microcontroller, an 8-channel A/D, an RS-232 compatible terminal I/O port, and some wire wrap area for additional circuitry like the D/A unit. The MC68HC11E9 high-density complementary semiconductor (HCMOS) high-performance microcontroller unit (MCU) includes the following features:

- 12 Kbytes of ROM
- 512 bytes of EEPROM

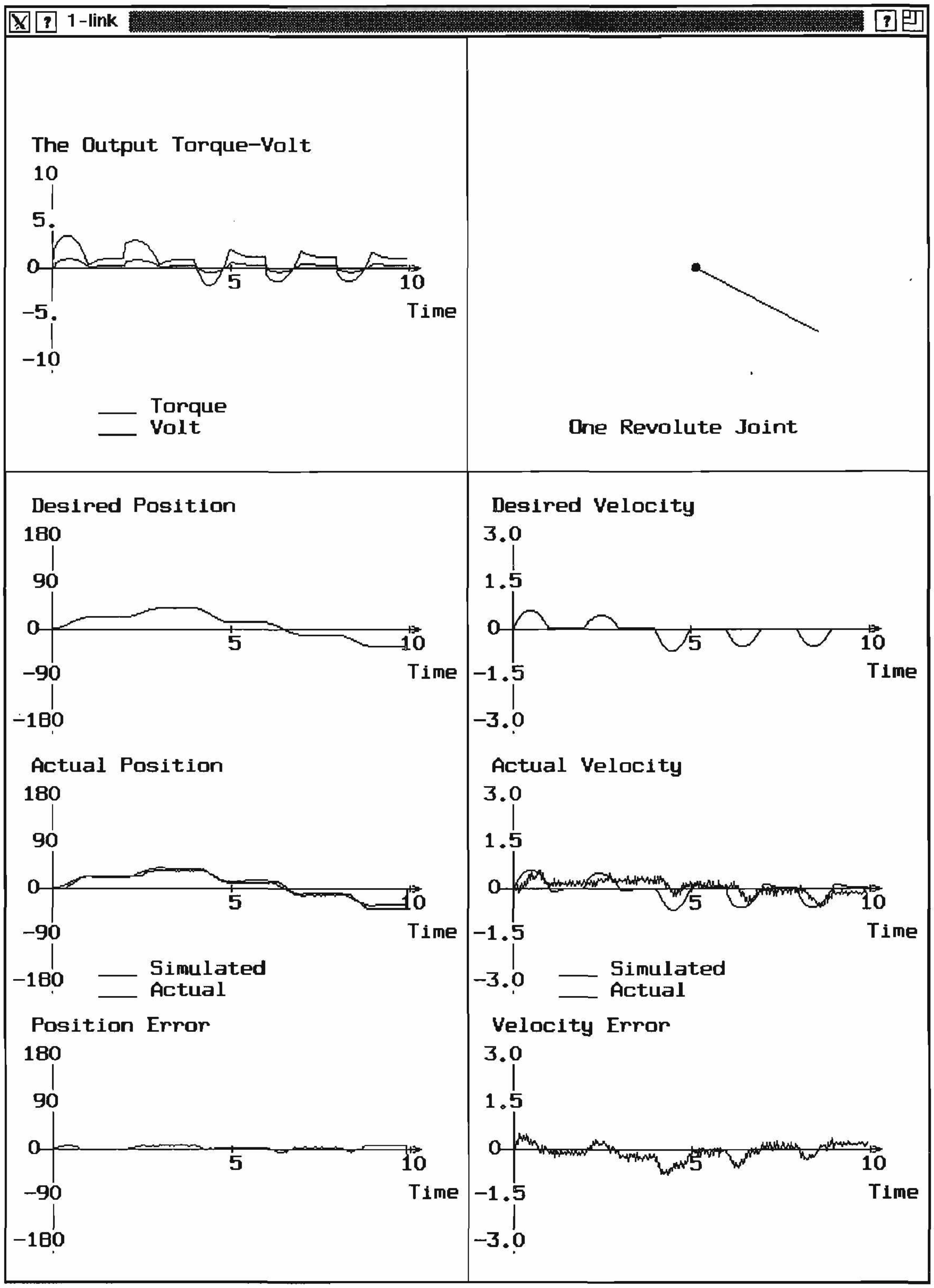


Figure 10: The Output Window of the Simulation Program for Sequence One.

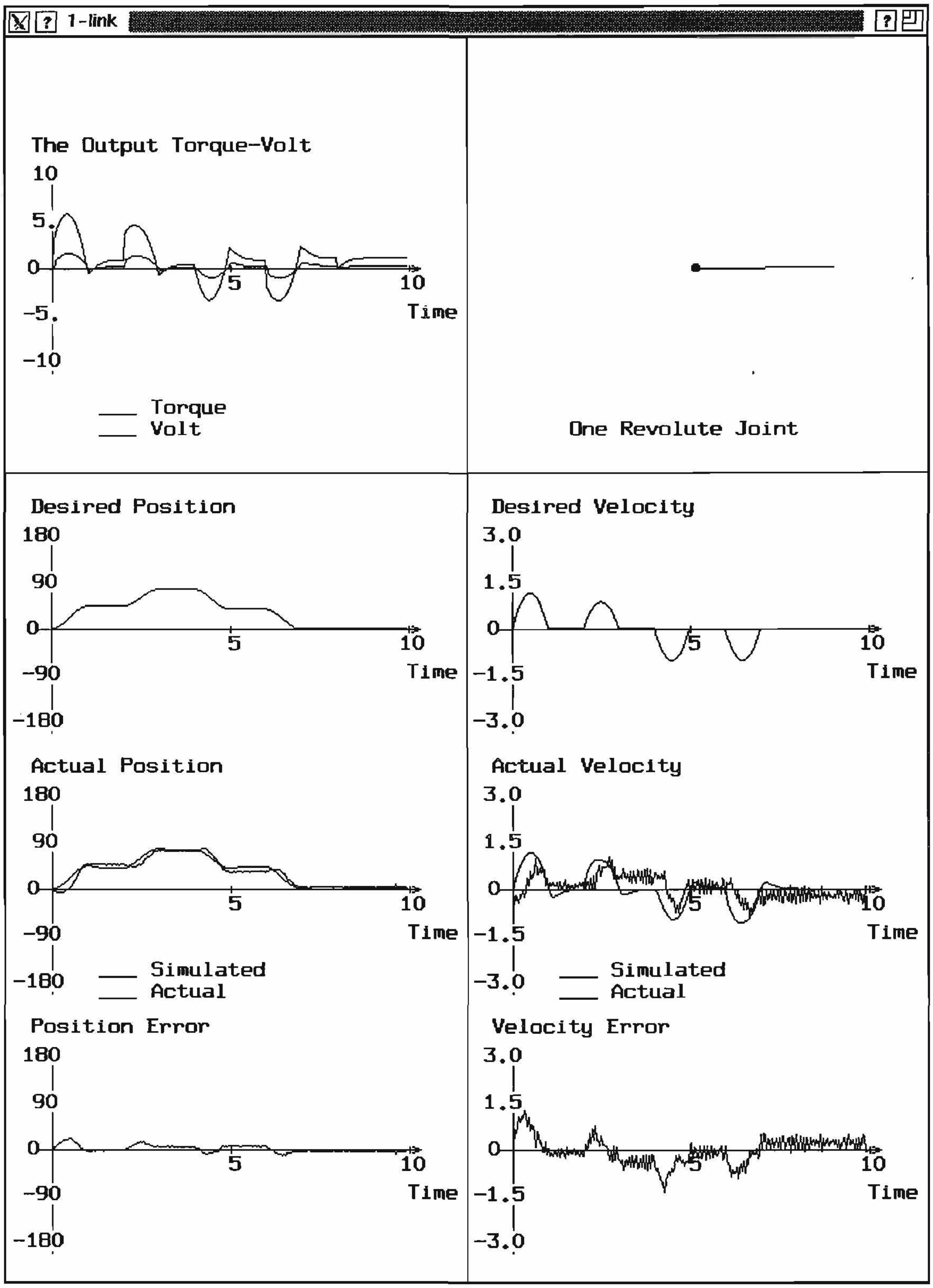


Figure 11: The Output Window of the Simulation Program for Sequence Two.

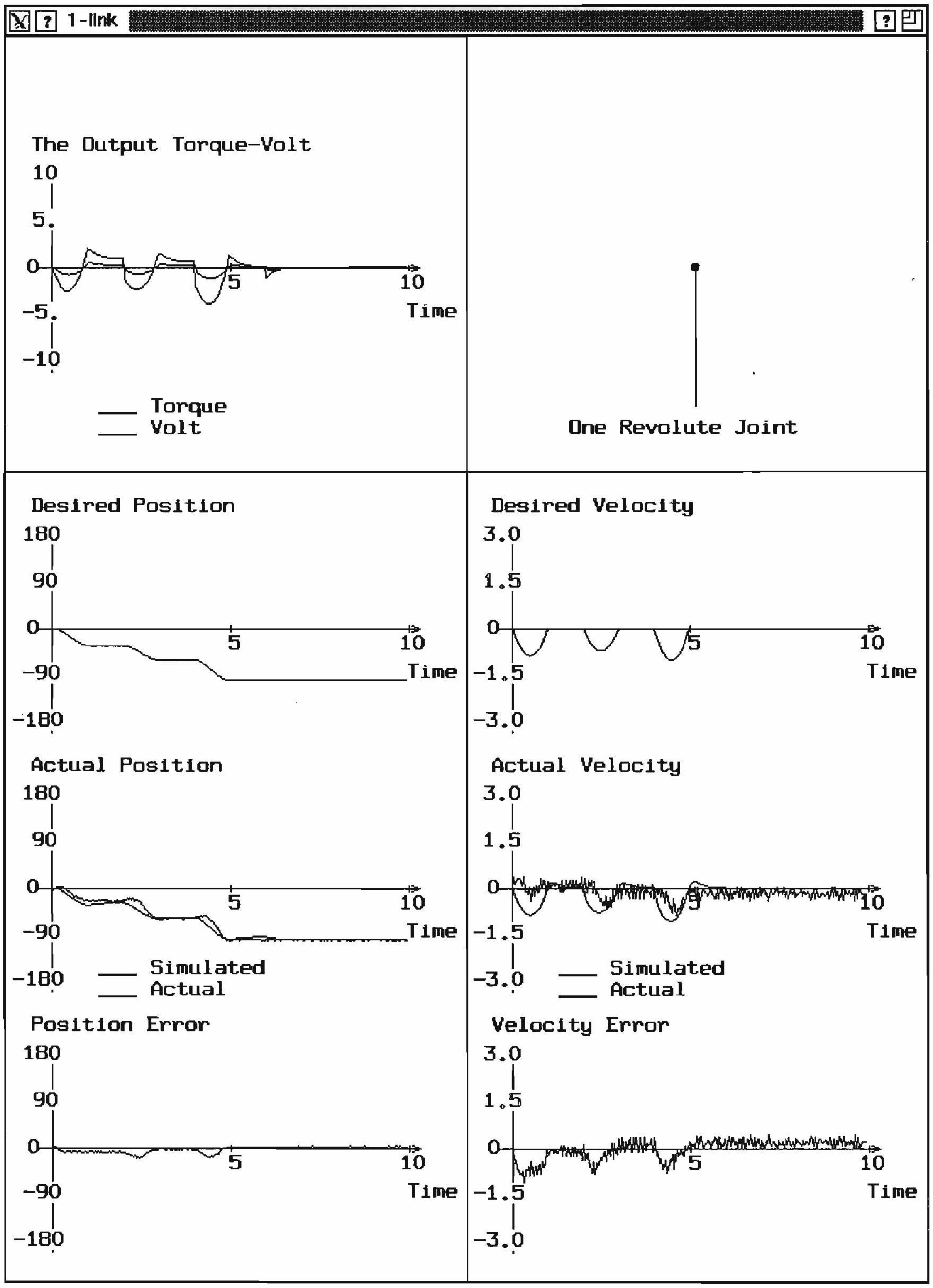


Figure 12: The Output Window of the Simulation Program for Sequence Three.

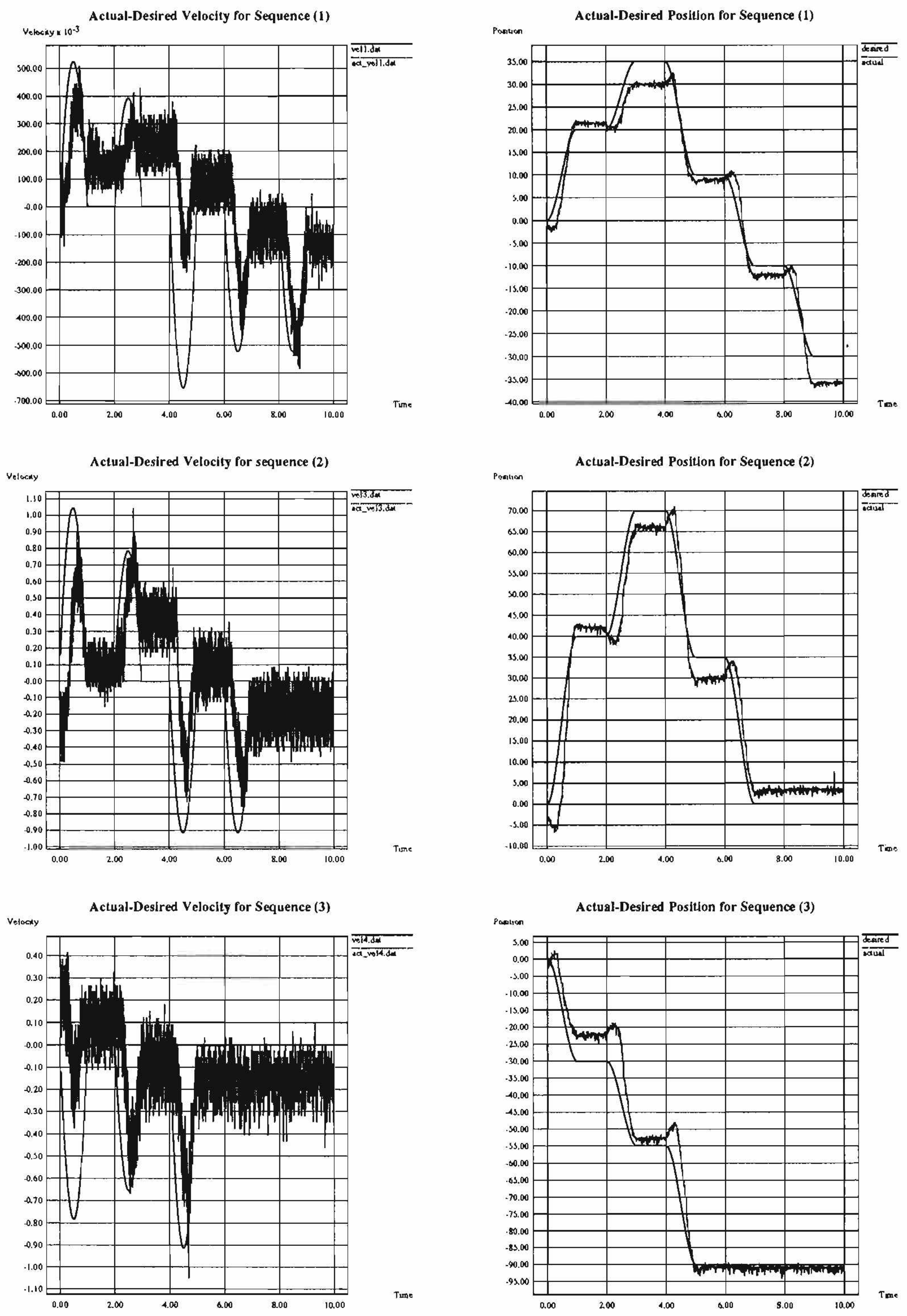


Figure 13: The Difference Between the Actual and the Desired Behavior.

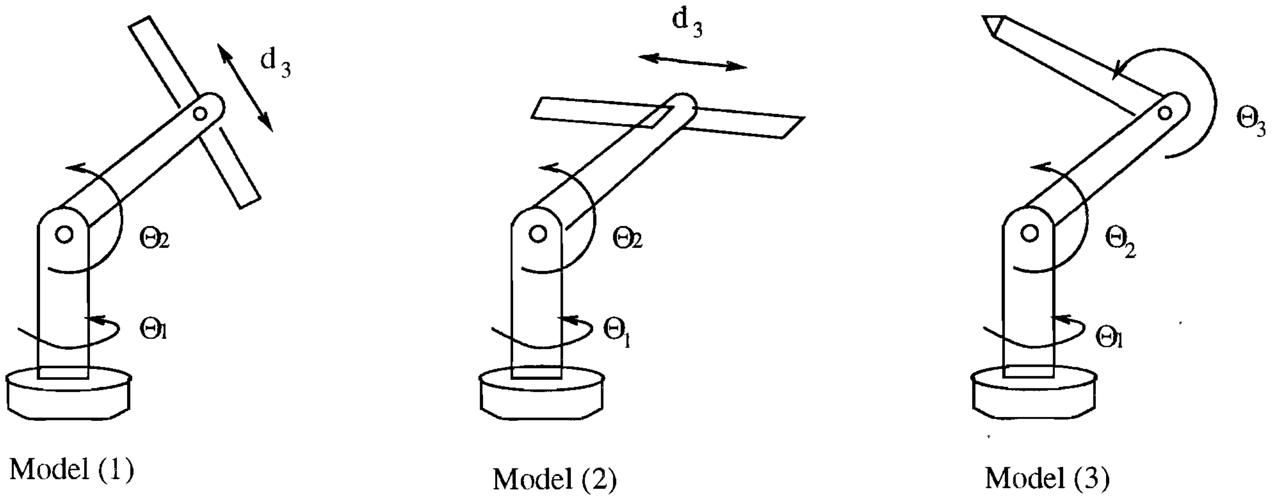


Figure 14: The three different configurations of the robot manipulator.

- 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 detailed about this chip see [31].

4.4 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 will be used that can be fed to the system at run-time, so that this controller can be used for different configurations without any changes.

Three different configurations have been chosen for development and study. The first configuration is revolute-revolute-prismatic with the prismatic link in the same plane of second link. The second configuration is also revolute-revolute-prismatic with the prismatic link perpendicular to the plane of the second link. The last configuration is three revolute joints (see Figure 14).

The kinematics and the dynamics of the three models have been generated using some tools in our 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. 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 a lot of calculations. 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 will simulate a PID controller to minimize that error. The error will depend 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 with a big angle in a very small time interval, the error will be large).

4.5 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 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.

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 is 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.

	Addition	Multiplication	Division
Model 1	89	271	13
Model 2	85	307	0
Model 3	195	576	22

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

4.6 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 approximate figures for the required speed to achieve 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 involves calculating the number of operations involved in each module to have a rough estimate of the time each module takes.

We used the simulator described in the previous section 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, the clock cycle rate, the MIPS and MFLOPS for each of them.

	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.

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 give 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 run the simulation program with update frequency of 1000 Hz for 10 seconds, which means that each routine will be called 10,000 times. From this output, it is obvious that the bottleneck is the dynamics routine and usually it takes 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 is that: after simplification using Mathematica, the execution time increases, but that is because the results contains a lot of different trig. functions, and it seems that these machines do not use lookup tables for such functions. So, we rewrite all non basic trig. functions, such as $\sin 2\theta$ in terms of basic trig. functions as $2\sin\theta\cos\theta$. Using this conversion, the performance was much better. Figure 15 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 trig. functions.

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

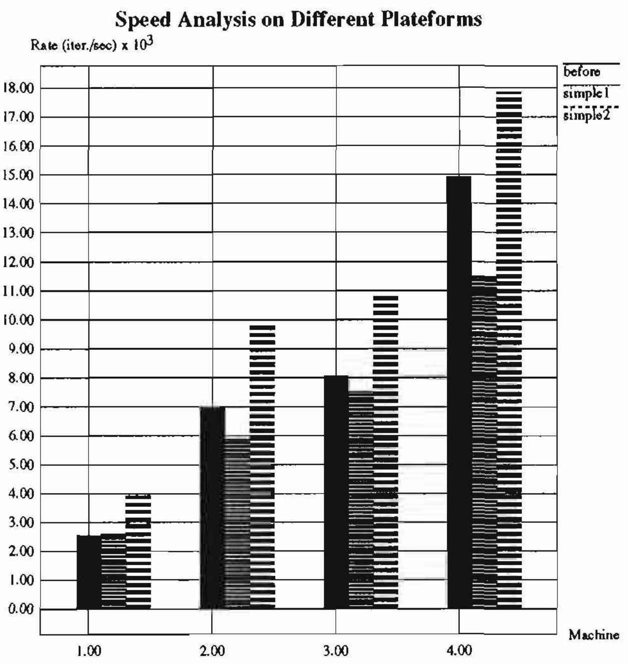


Figure 15: Performance comparison for different platforms: The machines from left to right are: SPARC 2, SPARC 10-30, SPARC 10-41, and HP 700. For each machine, there are three columns; before simplification, after using Mathematica, and after simplifying the trigonometric functions.

4.7 PID controller Simulator

As mentioned earlier, 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 has complicated shape.

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 will also serve as a monitoring system that provides several graphs and reports. This system is implemented using a graphical user interface development kit called GDI that was developed in our department under supervision of Prof. Beat Bruderlin. simulation program is still under development and testing. Figure 16 shows the interface window of that simulator.

4.8 Building the Robot

The design of the robot is completed and the required parts such as motors, actuators, and sensors, are now available. The assembly process of the mechanical and electrical parts was done at the Advanced Manufacturing Lab (AML) with the help of Prof. Stanford Meek and Mircea Cormos. In

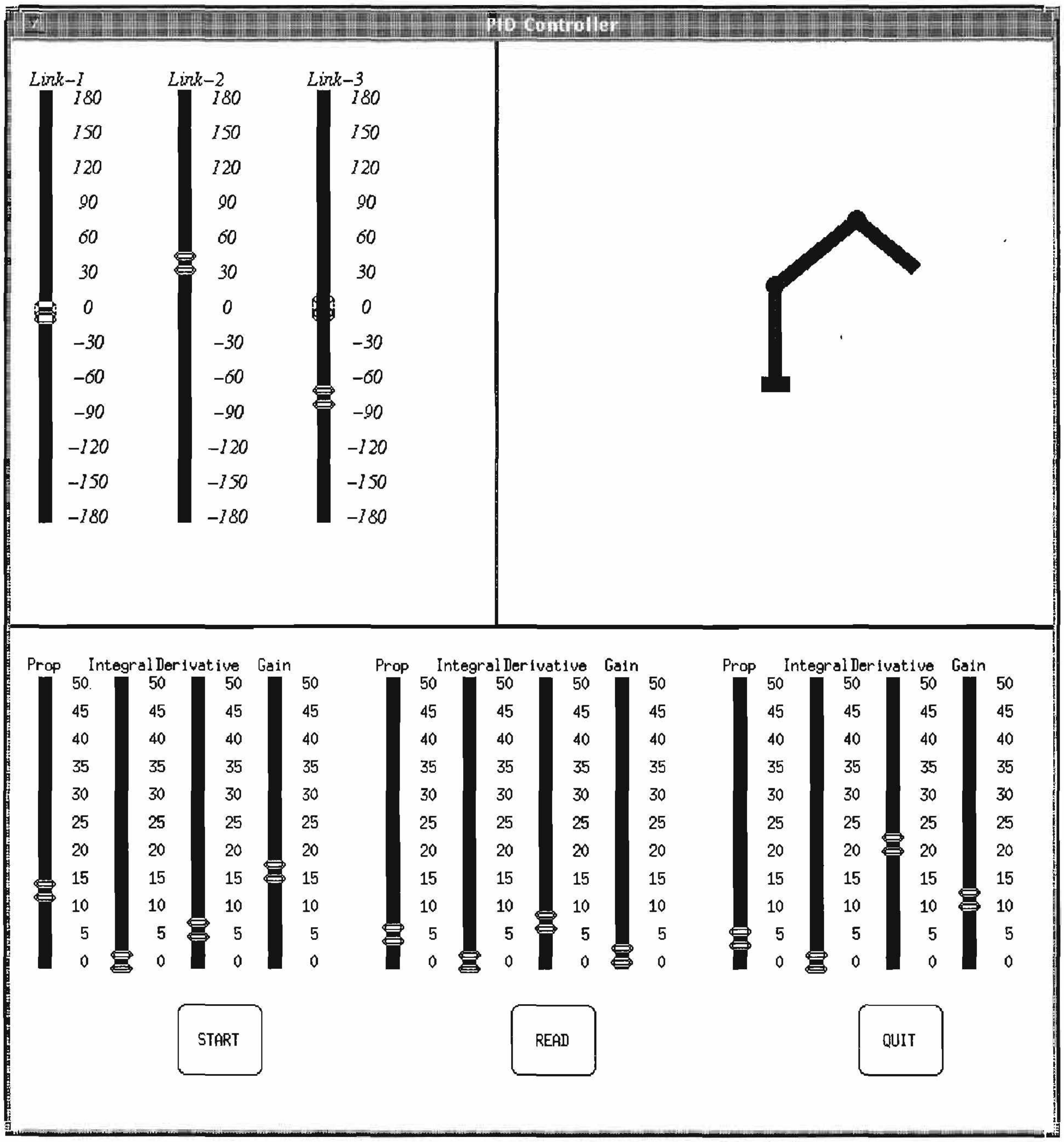


Figure 16: The Interface Window for the PID Controller Simulator.

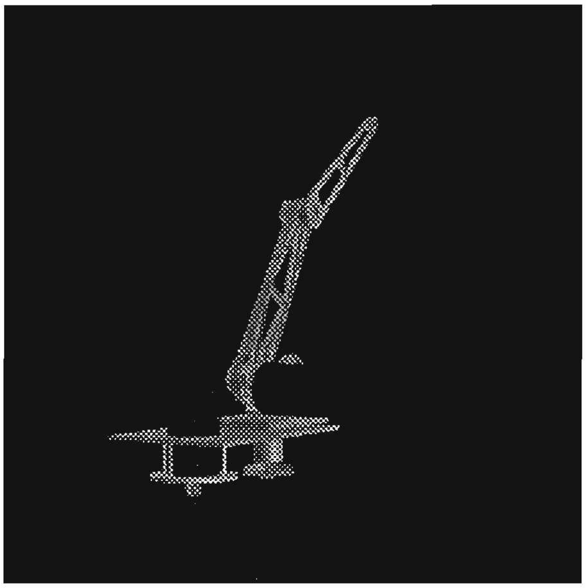


Figure 17: The Physical Three-Link Robot manipulator

this design the last link will be movable, so that we can set the robot in different configurations (the three models mentioned earlier).

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 an oscilloscope. Digital control can be also 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 18 shows an overall view of the different interfaces and platforms that can control that robot.

To complete this part, the following tasks have to be accomplished:

- Complete the sensors and actuators interface
- Test the robot and do some performance measurements
- Build a user interface for the system
- Document and package the robot and its software

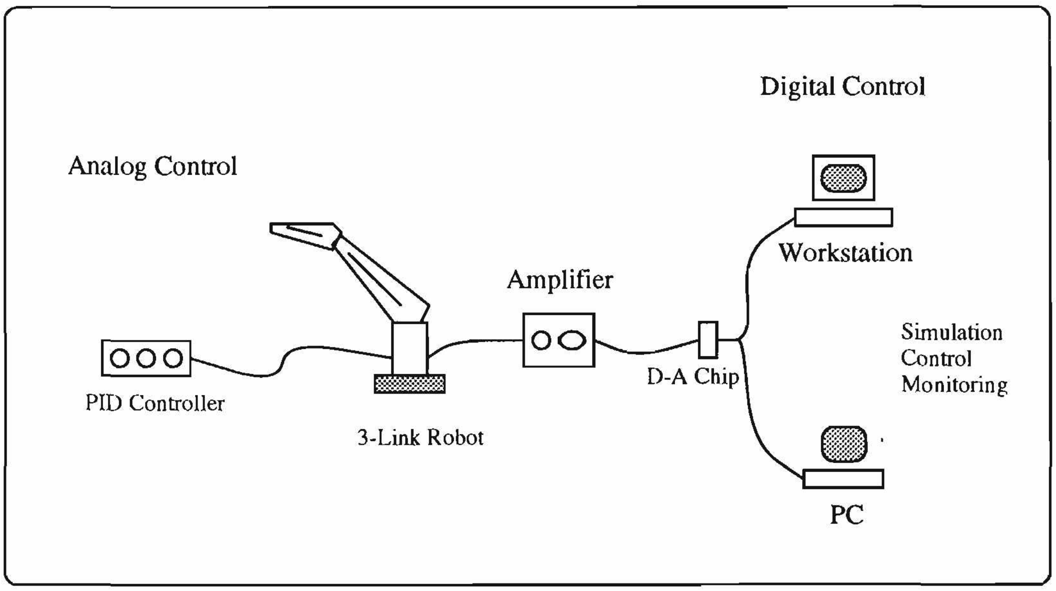


Figure 18: Controlling the robot using different schemes

5 The Optimal Design Sub-system

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

5.1 Performance Specification Interface

To be able to specify the required tasks of the robot and some performance requirements, an interface should be implemented to enable an easy and precise interface. Some of the tasks and performance requirements are:

- The required workspace.
- The control strategy (position control, force control, etc.)
- Number of degrees of freedom.
- The maximum speed required for the robot links.
- The maximum allowable error for each link.
- The maximum total weight and length of the robot.

5.2 Constructing the Optimization Problem

Any optimization problem has three main components:

- Objective function to be minimized or maximized.

- Optimization variables.
- Set of constraints.

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

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

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

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

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

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

Some of the rules that can serve as additional constraints:

- Select the solution with equal link lengths or masses because this will simplify the manufacturing process (minimize the cost).
- Choose the feedback controls k_p, k_v that give critically damped behavior ($k_v = 2\sqrt{k_p}$).

- There is a minimum length for l_{min} for each link to satisfy some assembly and manufacturing constrains, such as actuator and sensors sizes.

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

There are many algorithms for solving the optimization problem. In our case most of the objective functions will have more than one variable. In this case *multi-dimensional optimization techniques* are recommended. One of the simplest method is *pattern search* which alternates sequences of local exploratory moves with extrapolations (or pattern moves). One other method is *simple random search* which selects random search points and evaluate the function at each of those points. More details about these methods and other optimization techniques can be found in [10, 41]

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

Structural Length Index

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

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

where L is the length sum and is defined as:

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

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

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

$$\min f(l_i)$$

where:

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

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

Manipulability

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

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

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

Force Transmissibility

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

Accuracy

Frequency of update and sampling rates are the main parameters affecting the accuracy of the manipulator motion. In general, increasing the frequency of update and the sensor readings, results in smaller error patterns. There is no formal or closed form solution to determine the optimal value for the frequency of update that gives a specified maximum allowable error. The only practical way is by using simulation programs and changing the value for both frequencies until the desired behavior is obtained. The constraint in this case is the maximum speed of the machine used and the maximum speed for the interface between the robot and the actuators and sensors.

5.3 The User Interface

It is an interactive interface that enables the user to select the robot tasks (using the task description interface) and specify some of the parameter values (which will form the system constraints), then it displays the results of the recommended values for the other parameters.

After selecting values for the required parameters, the new configuration can be tested using the simulation and monitoring sub-systems, and each performance criteria can be measured and compared with the pre-specified performance requirements, then, changes can be made to the tasks or the constraints if necessary, and the design sub-system run again to get new values.

5.4 Some Design Examples

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

Example (1)

Performance Criteria:

- Efficient link lengths.
- Maximum manipulability.

Optimization Parameters:

- Link lengths.

Optimization Functions:

$$Q_L = \frac{L}{\sqrt[3]{W}}$$
$$w = \mid \det(J(\Theta)) \mid$$

Constraints:

- Total link lengths (L).

Strategy:

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

$$\min c_1 Q_L + c_2 w$$

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

Example (2)

Performance Criteria:

- Minimum position error (e).
- Maximum speed ($\dot{x}$).

Optimization Parameters:

- The feedback gains k_p, k_v.
- Joint friction fr.

Optimization Functions:

$$e = f(k_p, k_v, fr)$$

$$\dot{x} = g(k_p, k_v, fr)$$

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

Constraints:

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

Strategy:

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

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

This will reduce the optimization variables to two instead of three.

Example (3)

Performance Criteria:

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

Optimization Parameters:

- Link lengths.
- Link masses.

Optimization Functions:

$$\ddot{x} = f(l_i, m_i)$$

$$e = g(l_i, m_i)$$

$$P = h(l_i, m_i)$$

The overall objective function will be:

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

Constraints:

- Feedback gains.
- Update frequency.
- Feedback frequency.
- friction.
- Set of available densities for the link material.
- Catalog of available motors.

Strategy:

This problem will be solved in two stages: first the manipulability and the structured length index can be used to determine the optimum link lengths (as in the first example), Then, we use these lengths to get the optimum masses.

From the assumptions stated before, there is a finite set of densities, and the links are uniform, that means we need to select the density that gives optimum performance, since we already have the lengths. This problem can be solved using pattern search on the densities, or using some other integer optimization techniques.

The Power consumption of the motor is related to the torque, that mean we need to minimize the maximum torque.

Also here we will be using the simulation program to get a quantitative measure for the overall objective function.

Example (4)

Performance Criteria:

- Maximum speed ($\dot{x}$).
- Minimum position error (e).

Optimization Parameters:

- Update frequency (u).
- Feedback frequency (sensor reading rate, r).

Optimization Functions:

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

$$e = g(u, r)$$

The overall objective function will be:

$$a_1 f + a_2 g$$

Constraints:

- Link lengths and masses.
- Feedback gains.
- motor parameters.
- friction.
- Sensor ranges.
- Maximum computer speed.
- Maximum speed for the communication part (A/D and D/A).

Strategy:
Also here we will be using the simulation program to get a quantitative measure for the overall objective function, with the pattern search on the two frequencies given the maximum speed for the computer and the sensor reading rate.

6 The Interface Layer

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

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

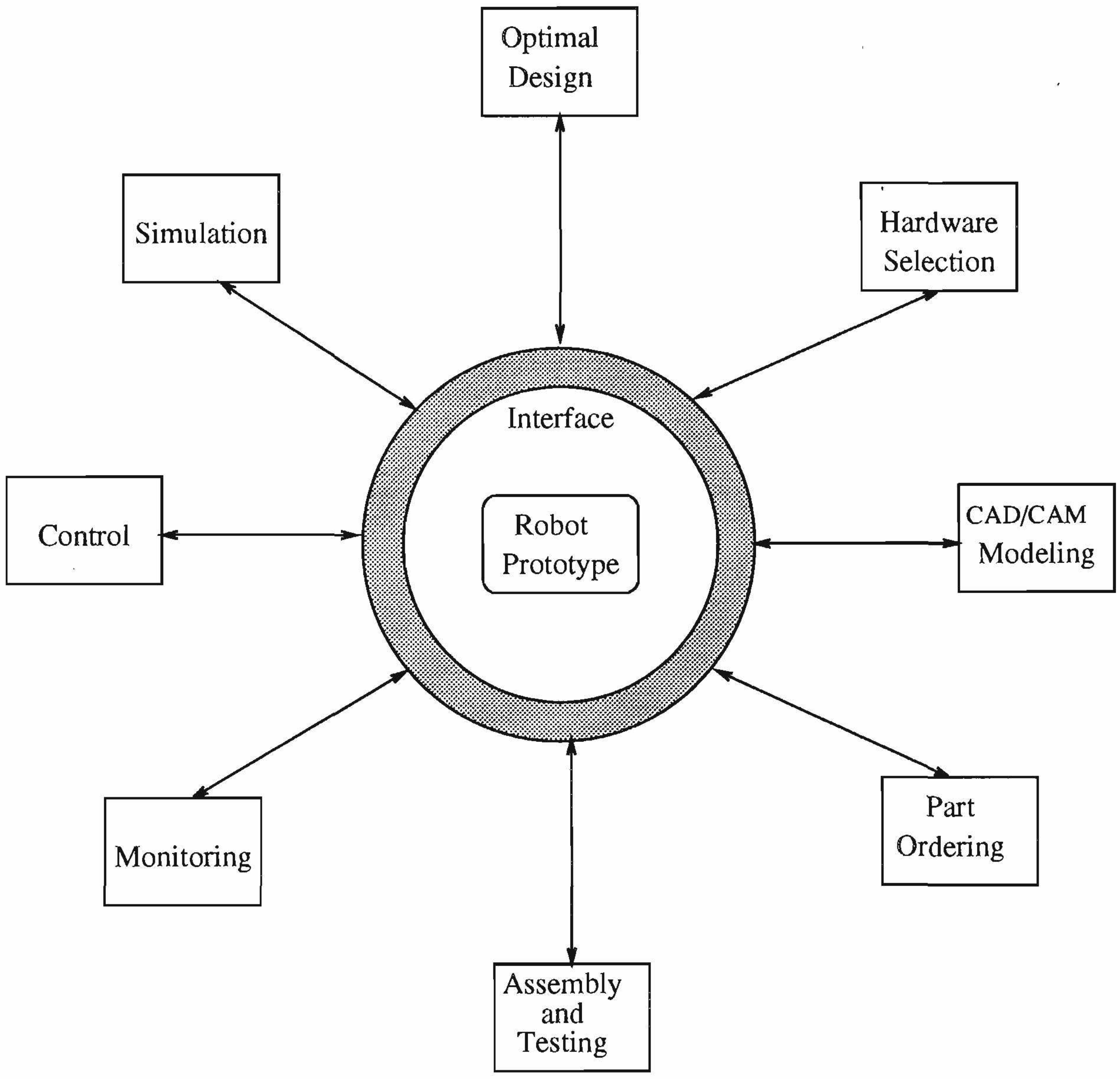


Figure 19: Schematic View for the Robot Prototyping Environment.

Figure 19 Shows a schematic view of the prototyping environment with its sub-systems and the interface.

There is a lot of shared parameters and information among these sub-systems. To maintain the integrity and consistency of the whole system, a multi-site interface is proposed with the required rules and protocols for passing information through the whole system. This interface will be the layer between the robot prototype and the sub-systems, and also it will serve as a communication channel between the different sub-systems.

The tasks of this interface will include:

- Building relations between the parameters of the system, so that any change 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-system to keep track of the design reasoning and decisions.

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

There are several mechanisms used in these sub-systems which can be classified as follows:

- **Constrained-based approach:** this approach is used in the optimal design sub-system.
- **Ruled-based approach:** used in the the CAD/CAM and the hardware selection sub-systems. These rules are used to assist decision making during the design process.
- **Search-based approach:** used in the part-ordering sub-system, which is basically, catalog search for the required parts (motors, sensors, amplifiers, link-materials, etc). This system will be the front-end of an internet-based cataloging system developed at the Mechanical Engineering Department.
- **Functional relations:** used for building the relations between some of the design parameters. For example, link lengths is one of the parameters that has relations with other parameters such as masses and inertia tensors, and also it takes place in the design, control, and simulation systems. If we change the length of one of the links, we want the corresponding mass and inertia tensor to change with a pre-specified functions that relates the length to each of them. We also want the length in the other sub-systems to change as well according to pre-specified mathematical relations.

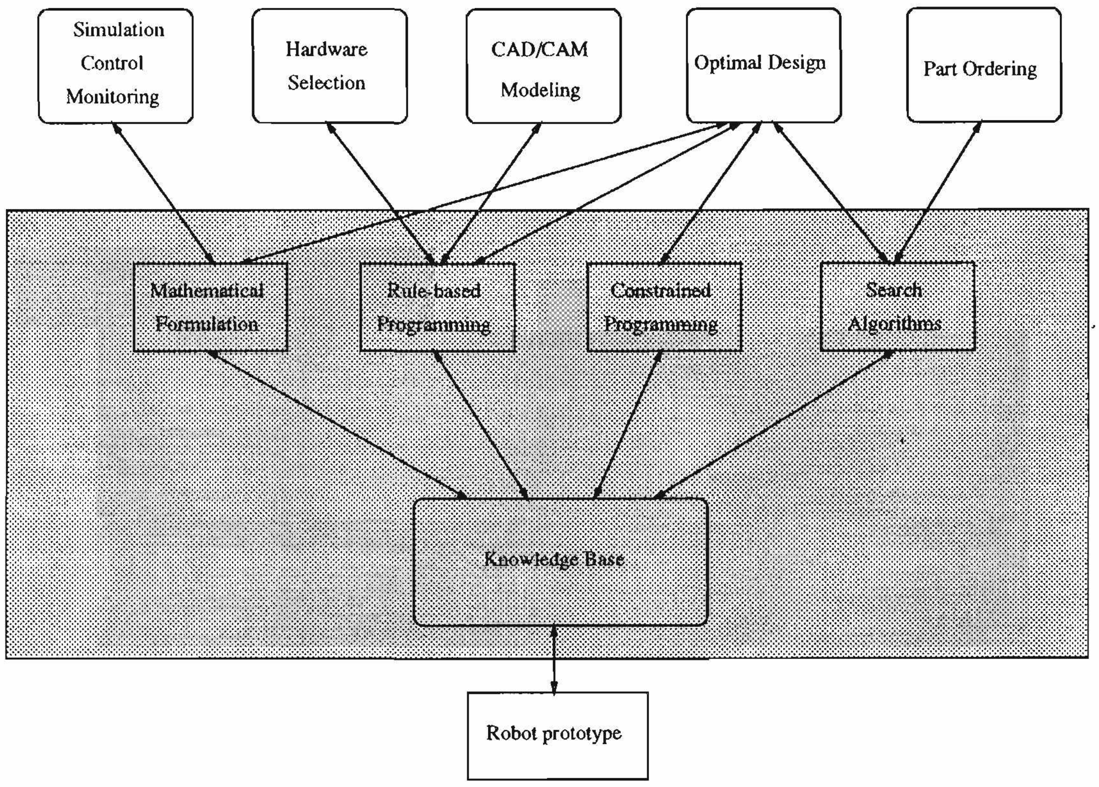


Figure 20: The Interface Between the Subsystem and the Prototype Robot.

- **Mathematical Formulation:** used in the simulation, and control sub-systems to define the robot modules (kinematics, inverse kinematics, dynamics, etc).
- **Shared Database Manipulation:** used in most of the sub-systems. For example, the simulation and control are just retrieving data from the shared database, while the monitor subsystem adds analysis information to the database that will be used as a feedback to the design sub-system. The design sub-system updates the parameters of the system. The CAD/CAM system uses this database to check the validity of the chosen parameters and adds to the database some comments about the design and manufacturing problems that might exist.

Since we are dealing with different architectures and approaches, we will use an *object-oriented* scheme to design this interface. Each object deals with one of the sub-systems in its own language. This will make it easier to change the approach or the structure of any of the sub-system without affecting the other sub-systems, by only changing the corresponding object in the interface. Figure 20 shows the proposed interface layer.

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 will illustrate the mechanism of this interface and the way these systems can communicate to maintain the system consistency.

Suppose that the designer wants to change the length of one of the links and he wants to see what should be the motor parameters that give the same performance requirements. First, this change will be recorded and the length field will be updated in the shared database for each sub-system. Then the optimal design will be used to determine the new values for the motor parameters using the simulation program. Then search techniques will be used to look up 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 these specification, in this case, this will be recorded in the comments field and another motor with closest specifications will be selected. Next, The motor specifications will be updated in the database, then the CAD/CAM system will be 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 width is relatively small and a sprocket is hard to install. In this case, this will be recorded in the comments filed and the user will be notified with this problem and will be asked to either change the some of the parameters or the performance requirements and the loop starts again. Once the parameters is determined, the monitoring program will be used to give some performance analysis and compare the results with the required performance, and produces a report with the results.

As another example, suppose that we need to select link masses and motor parameters that give maximum speed and minimum position error. The design sub-system will select density of the links material from the finite density set, and will use the part-ordering system to select the motor and try to get the best combination of motor parameters and link masses that give best value for the combined objective function (speed and position error). The optimization problem here will be solved using the simulation programs. After selecting the required parameters, the CAD/CAM system will be used to generate the model and again to see if this is a valid model to be manufactured. In some cases the motor might be too heavy relative to the link weights (usually when we have small links). In this case the simulation results will show that and another motor should be selected, or another density should be chosen. Finally the parts can be ordered and the assembly can take place.

6.1 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 should be identified, also the knowledge representation used in each sub-system should be determined.

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

- Apply relations between parameters.
- Satisfy rules.
- Satisfy 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 Formula.
- Graphs.
- Catalogs and tables.
- Rules and constraints.
- Programs written in different languages (C, C++, Lisp, Prolog, etc.).

Some of the sub-systems can change some of the parameters and the configuration of the prototype system. The Optimal design sub-system is the one that make almost all the changes in the design parameters. The CAD/CAM sub-system can also make some design changes according to some geometric and manufacturing rules and constraints. The user can change any of the design parameters, make decisions, and run any of the sub-systems.

Tables 3, 4, 5, and 6 describe the interaction between the sub-systems; that is, what each sub-system needs to know when if some of the design parameters are changed by one of the sub-systems or by the user, and what actions it might take as a consequence of this change.

The following is a description for the actions that may take place in the environment as a result of changing some of the design parameters.

Change constraints and optimize: When any change occurs to one of the design parameters, that means changing in one of the constraints for the optimization problem, then the user can decide whither to rerun the optimal design system or not.

Update file: Updating the data files used by the simulation, control, and monitoring sub-systems.

Apply relation: Some of the parameters are related to other parameters in the same sub-system, and to corresponding parameters in other sub-systems. For example, the relation between the link length in the design sub-system and the corresponding drawing length in the monitoring sub-system can be something like:

$$L_{Monitor} = LinkScale * L_{Design},$$

where, *linkScale* is the scaling factor to draw the link on the computer's screen.

System	*Lengths*	*Masses*	*Motors*	*Frequency*	*Sensors*	*Feedback*	*Friction*

Table 3: The interaction between the user and the sub-systems.

System	*Lengths*	*Masses*	*Motors*	*Frequency*	*Sensors*
Simulation	Update file	Update file	Update file	Update file	Update file
Control	Update file	Update file	Update file	Update file	Update file
Monitoring	Apply relation Drawing length		Apply relation Max torque	Apply relation Display rate	Apply relation Max pos,vel
HW Selection			D/A chip	Select platform	A/D chip
CAD/CAM	Update model Check for length	Update model Check for weight	Update model Check gear ratio		
Part-ordering		Search and give report			Search and give report
Assembly	Change link Test, report	Change link Test, report	Change motor Test, report		Change sensors Test, report

Table 4: The interaction between the optimal design and the other sub-systems.

System	*Lengths*	*Masses*	*Gear Ratio (Motor)*
Simulation	Update file	Update file	Update file
Control	Update file	Update file	Update file
Monitoring	Apply relation Drawing length	Update file	Apply relation Max torque
HW Selection			Select D/A chip
Optimal Design	Optimize for other parameters	Optimize for other parameters	Optimize for other parameters
Part-ordering			Search and give report
Assembly	Change link	Change link	Change motor and gears

Table 5: The interaction between CAD/CAM and the other sub-systems.

System	*Platform (Update rate)*	*Communication (Feedback rate)*
Simulation	Update file	Update file
Control	Update file	Update file
Monitoring	Apply relation Display rate	Apply relation Display rate
Optimal Design	Optimize for other parameters	Optimize for other parameters

Table 6: The interaction between hardware and other sub-systems.

Select D/A, D/A chips: When the motors and the sensors are selected, a chip that contains the D/A and A/D converters and the micro-programs that control the conversion should be selected by the hardware selection sub-system.

Select platform: According to the selected update frequency and the number of computation in each sub-system, the hardware selection sub-system will select the machines that can accommodate that frequency.

Update model: The CAD/CAM sub-system will create a new model for the prototype robot according to the changes in the design parameters.

Check for length, mass, gear ratio and friction: Apply the rules and constraints for each of these parameters that are imposed by geometric and manufacturing limitations.

Search and give report: After the motor specification and the sensor ranges are selected, the part-ordering sub-system will search in the parts catalogs to find the required motors and sensors. If no motors or sensors are found with the required specifications, this will be reported to the user, and the some other motors or sensors with close specifications will be recommended.

Change parts, test, and report: This is the final step in the design. After all parameters are selected and all parts are available, the assembly process can take place, then the design can be physically tested, and the results are reported to the user.

In some cases there might be interaction cycles. in such cases, the user has to take decisions that resolve these cycles. For example, suppose that the link length was determine by the design sub-system, but the CAD/CAM system has some rules that requires the length to be changed. In this case the design sub-system needs to be run again to accommodate this change. this might change the motor parameters or sensor ranges, and again, this change may violate another rule in the CAD/CAM sub-system which requires another change, and so on. To resolve this cycle the user can take some design decisions that will satisfy the rules and constrains in the sub-systems.

The part-ordering sub-system can cause some indirect changes to the design parameters. For example if a motor with certain specifications is not found, then this sub-system will report that to the user and may recommend some other motors that have close specifications to the required. The user then can either choose one of the recommended motors, or make some design changes and run the optimal design sub-system to get new motor specifications. Figure 21 shows some interaction cycles.

6.2 The Interface Scheme

There are several schemes that can be used for the interface layer. One possible scheme is that: each sub-system will have 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.

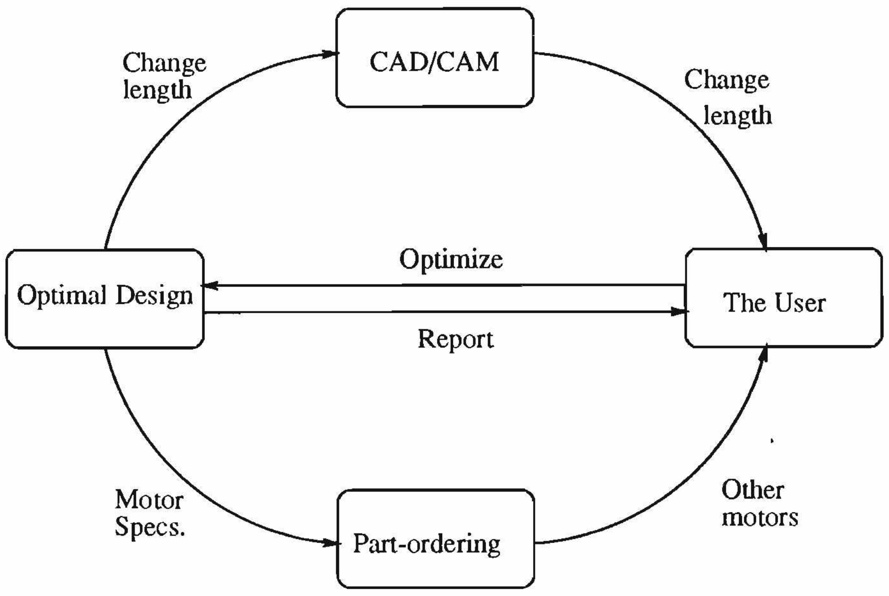


Figure 21: Examples of Some Interaction Cycles.

- Receive requests from other sub-system interfaces and translate it to the local language.

These sub-system interfaces can communicate in three different ways, (see Figure 22):

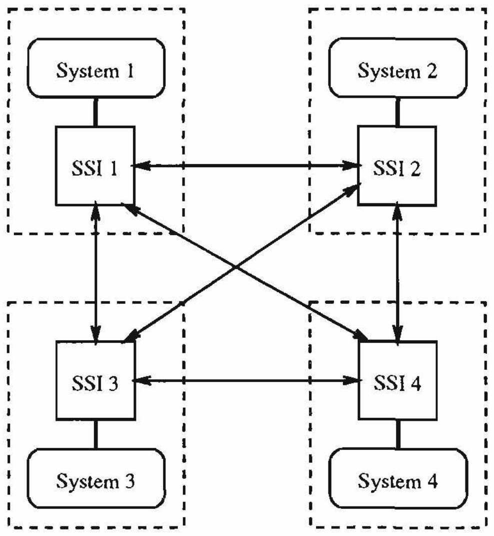
Direct connection: which means that all interfaces can talk to each other. the advantage of this method is that it has a high communication speed, but it makes the design of such interfaces more difficult, and the addition of new interface or changing one of the interfaces requires the modification of 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 to pick the relevant messages and translate it to its sub-system. The problem with this scheme that it makes the synchronization between the sub-systems very difficult, and the design of the interface will be more complicated.

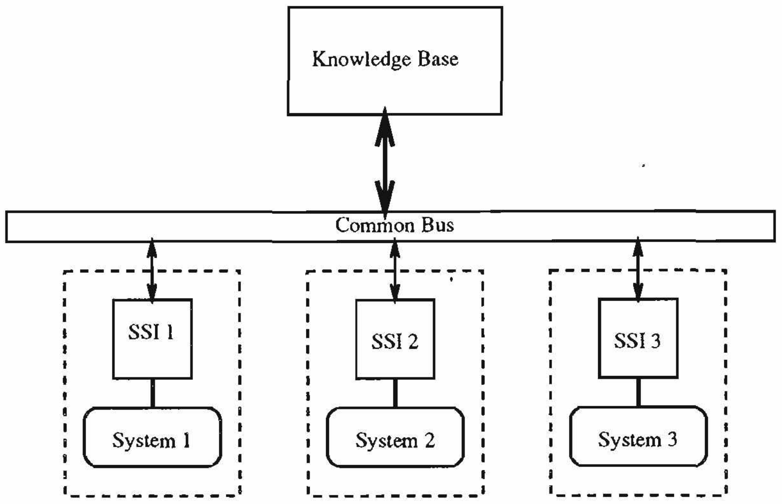
Centralized control: in which all interfaces will talk with one centralized interface that controls the data and control flow in the environment. The advantages of this scheme is that it makes it much easier to synchronize between the sub-systems, and the addition or modification of any of the SSIs will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

6.3 Interface Design

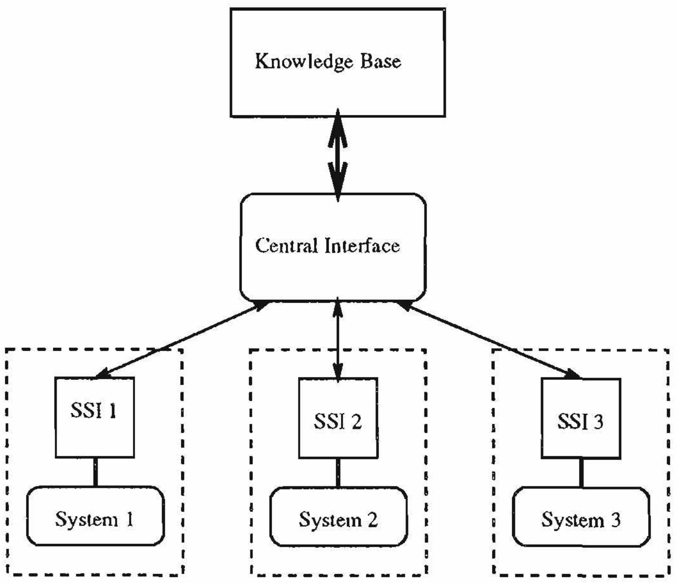
The interface layer contains several components that define the objects in the environment, the relation between these objects, the rules and constraints in the system, the representation of these objects in each sub-system, and the communication protocols between the sub-systems. More details about the design of these components will be discussed in another report.



(1) Direct Connection



(2) Message Routing



(3) Centralized Control

Figure 22: Three Different Ways for sub-system interfaces communication.

7 Current Development

The following is a list of the current activities and developments:

- Complete the object analysis phase for the prototyping environment by defining the objects and their relations.
- Determine the rules, constraints and actions in the environment.
- Design and implement the Optimal design sub-system.
- Complete the communication interface for the 3-link robot and identify the motor parameters.
- Test the 3-link robot prototype and make some performance analysis.

8 Conclusion

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 in such a way that makes it very 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.

So far we have implemented some of the sub-systems such as: controller, simulator, and monitor, We are now in the stage of testing the three-link robot, implementing the optimal design sub-system, and putting the basis for the shared knowledge base and the interface layer.

References

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

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

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

[9] 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.

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

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

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

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

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

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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

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

[33] 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.

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

[35] 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.

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

[37] 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.

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

[39] 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.

[40] 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.

[41] Toole, H. *Optimization Methods.* Springer-Verlag, 1975.