URK: Utah Robot Kit – A Three-link Robot Prototype

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

UUSC-94-012

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

March 30, 1994

Abstract

In this paper we will present the stages of designing and building a three-link robot manipulator prototype that was built as part of a research project for establishing a prototyping environment for robot manipulators. Building this robot enabled us determine the required subsystems and interfaces to build the prototyping environment, and provided hands-on experience for the real problems and difficulties that we would like to address and solve using this environment. Also, this robot will be used as an educational tool in robotics and control classes.

[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. This report has also been submitted as a paper in the *International Journal on Robotics Research*

URK: Utah Robot Kit – A Three-link Robot Prototype

Mohamed Dekhil, Tarek M. Sobh, Thomas C. Henderson,
and Anil Sabbavarapu

March 27, 1994

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

Abstract

In this paper we will present the stages of designing and building a three-link robot manipulator prototype that was built as part of a research project for establishing a prototyping environment for robot manipulators. Building this robot enabled us determine the required subsystems and interfaces to build the prototyping environment, and provided hands-on experience for the real problems and difficulties that we would like to address and solve using this environment. Also, this robot will be used as an educational tool in robotics and control classes.[1]

1 Introduction

Teaching robotics in most engineering schools lakes the practical side and usually students end up taking lots of theoretical background and mathematical basis, and maybe writing some simulation programs, but they do not get the chance to apply and practice what they have learned on real robots. This is due to the fact that most of the robots available in the market are either too advanced, complicated, and expensive (e.g., specialized industrial robots), or toy-like robots which are too trivial and does not give the required level of depth or functionality needed to demonstrate the main concepts of robot design and control. One of our goals in this project, was to build a robot that is simple, flexible, and easy to use and connect to any workstation or PC, and at the same time, is capable of demonstrating some of the design and control concepts. We also tried to keep the cost as low as possible to make it available to any engineering school or industrial organization.

[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.

Therefore, we built URK (Utah Robot Kit) which is a three-link robot prototype that has a small size and reasonable weight which is convenient for a small lab or a class room. URK can be connected to any workstation or PC through the standard serial port with an RS232 cable, and can be controlled using a software program with a graphical user interface.

This paper starts with a brief background of robot design and modules and the related work in this area. Then, a detailed description of designing and building URK is presented in Section 3. The communication between the robot and the workstation is discussed in detail in Section 4. Section 5 is a quick overview of the prototyping environment which URK is part of. Section 6 shows some results of testing and running URK. Finally, Section 7, is our conclusion from this work.

2 Background and Related Work

2.1 Robot Modules and Parameters

Controlling and simulating a robot is a process that involves a large number of mathematical equations. To be able to deal with the required amount of computation, it is better to divide them into modules, in which each module accomplishes a certain task. The most important modules, as described in [2], are kinematics, inverse kinematics, dynamics, trajectory generation, and linear feedback control.

2.1.1 Forward and Inverse Kinematics

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

One approach to the problem of kinematics analysis is described in [13], which is suitable for problems where there are one or more points of interest on every link.

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 [6]. The input to this package is the Denavit-Hartenberg parameters, and the output is the direct and inverse kinematics solutions. Another method of finding symbolic solutions for the inverse kinematics problem was proposed in [14]. Kelmar and Khosla proposed a method for automatic generation of forward and inverse kinematics for a reconfigurable manipulator system [8].

2.2 Robot Dynamics

Dynamics is the study of the forces required to cause the motion. There are two problems related to the dynamics of a manipulator: *controlling* the manipulator (inverse dynamics), and *simulating* the motion of the manipulator (forward dynamics). The dynamics module is the most time consuming among the manipulator's modules. That is because of the tremendous amount of calculation involved in the dynamics equations. This fact makes the dynamics module a good candidate for hardware implementation, to enhance the performance of the control and/or the simulation system.

There are some parallel algorithms to calculate the dynamics of a manipulator. Several approaches have been suggested in [9, 10, 12] based on a multiprocessor controller, and pipelined architectures to speed up the calculations.

2.3 Trajectory Generation

This module computes a trajectory in multidimensional space which describes the motion of the manipulator. There are several strategies to calculate trajectory points which generate a smooth motion for the manipulator. One of the simplest methods is using *cubic polynomials.*

2.4 Linear Feedback Control

A linear feedback control system is used in most control systems for positioning and trajectory tracking. 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. 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 1 shows a block diagram for the controller, and the role of each of the robot modules in the system.

2.5 Local PD feedback Control vs Robot Dynamic Equations

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

- Very simple to implement.
- No need to identify the robot parameters.
- Suitable for real-time control since it has very few computations compared to the complicated non-linear dynamic equations.

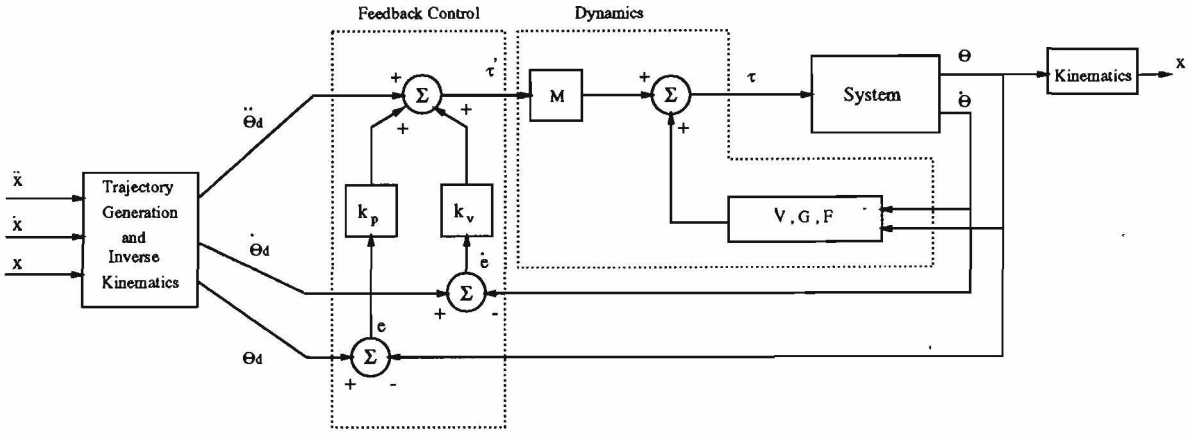

Figure 1: Block diagram of the Controller of a Robot Manipulator.

- 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 have been suggested to enhance the usability of the local PD feedback law for trajectory tracking. One idea is to add a lag-lead compensator using frequency response analysis [1]. Another method is to build an inner loop stabilizing controller using a multi-variable PD controller, and an outer loop tracking controller using a multi-variable PID controller [15]. 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 [7].

3 Prototyping a 3-Link Robot

3.1 Analysis Stage

This project was started with the study of a set of robot configurations and analyzed the type and amount of calculation involved in each of the robot controller modules (kinematics, inverse kinematics, dynamics, trajectory planning, feed-back control, and simulation). This

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

3.2 One Link Manipulator

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

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

One of the problems we faced in this process was to establish the transfer function between the torque and the voltage. The motor parameters were used to form this function by making some simplifications, since some of the motor parameters have nonlinear components that make it too difficult to make an exact model. Figure 2 shows the relation between torque and voltage for a certain input sequence.

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 [3].

3.3 Controller Design

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

Three different configurations have been chosen for development and study. The first configuration is revolute-revolute-prismatic with the prismatic link in the same plane as the

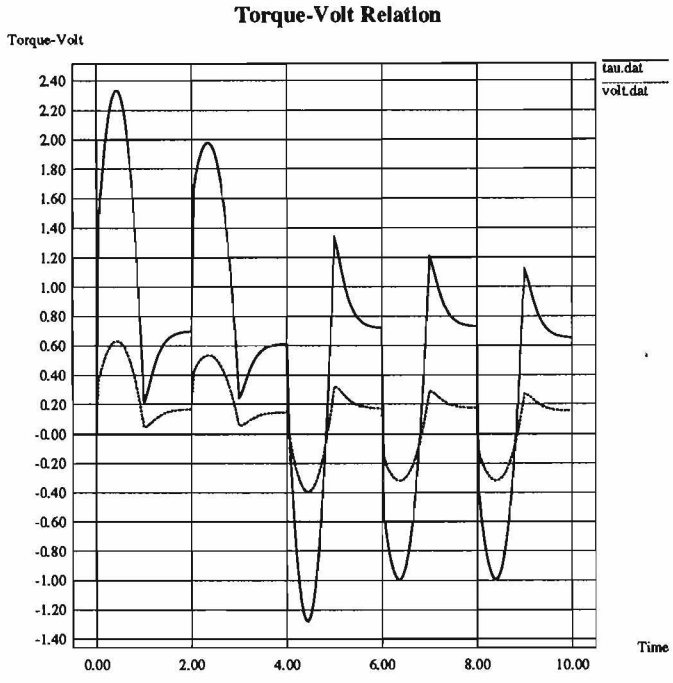

Figure 2: The relation between torque the voltage.

first and second links. The second configuration is also revolute-revolute-prismatic with the prismatic link perpendicular to the plane of the first and second links. The last configuration is three revolute joints (see Figure 3).

The kinematics and the dynamics of the three models have been generated using some tools in the department called *genkin* and *gendyn* that take the configuration of the manipulator in a certain format and generate the corresponding kinematics and dynamics for that manipulator. For the trajectory generation, The cubic polynomials method, described in the trajectory generation section, was used. This method is easy to implement and does not require much computation. It generates a cubic function that describes the motion from a starting point to a goal point in a certain time. Thus, this module will give us the desired trajectory to be followed, and this trajectory will serve as the input to the control module.

The error in position and velocity is calculated using the readings of the actual position and velocity from the sensors at each joint. Our control module simulated a PID controller to minimize that error. The error depends on several factors such as the frequency of update, the frequency of reading from the sensors, and the desired trajectory.

3.4 Simulation

A simulation program has been implemented to study the performance of each manipulator and the effect of varying the update frequency on the system. Also it helps to find approximate

ranges for the required torque and/or voltage, and to determine the maximum velocity to know the necessary type of sensors and A/D. To make the benchmarks, as described in the next section, we did not use a graphical interface to the simulator, since the drawing routines are time consuming, and thus give misleading figures for the speed.

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

3.5 Benchmarking

One important decision that had to be made was: do we need to implement some or all of the controller module in hardware? And if so which modules, or even parts of the modules, should be hardwired? To answer these questions we chose approximate figures for the required speed to achieve a certain performance, the available machines for the controller, the available hardware that can be used to build such modules, and a time chart for each module in the system to determine the bottlenecks. This also involved calculating the number of operations in each module giving a rough estimate of the time taken by each module.

The simulator described in Section 3.4 was used to generate time charts for each module, and to compare the execution time on different machines. The machines used in this benchmarking effort include: SUN SPARCStation-2, Sun SPARCStation-10 model 30, Sun SPARCStation-10 model 41, and HP-700. Table 2 shows the configurations of the machines used in this benchmark, with the type, clock cycle rate, the MIPS and MFLOPS for each.

The simulation program was executed with an update frequency of 1000 Hz for 10 seconds, which means that each routine was called 10,000 times. From this output, it was obvious that the bottleneck was the dynamics routine and usually it took between 25% to 50% of the total execution time on the different machines. From these results we found that the HP-700 was the fastest of all, followed by the SPARC-10 machines. Figure 4 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.

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

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

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

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

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

3.6 PID Controller Simulator

As mentioned in Section 2.5, a simple linear feedback control law can be used to control the robot manipulator for positioning and trajectory tracking. For this purpose, a PID controller simulator was developed to enable testing and analyzing the robot behavior using this control strategy.

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

This simulator has a user friendly interface that enables the user to change any of the feedback coefficients and the forward gains on-line. It can also read a pre-defined position trajectory for the robot to follow. It also serves as a monitoring system that provides several graphs and reports. The system is implemented using a graphical user interface development kit called GDI.[2] Figure 5 shows the interface window of that simulator.

3.7 Building the Robot

The assembly process of the mechanical and electrical parts was done in the Advanced Manufacturing Lab (AML) with the help of Mircea Cormos and Prof. Stanford Meek. In this design the last link is movable, so that different robot configurations can be used (see Figure 6).

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

This robot can be controlled using analog control by interfacing it with an analog PID controller. Digital control can also be used by interfacing the robot with either a workstation

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

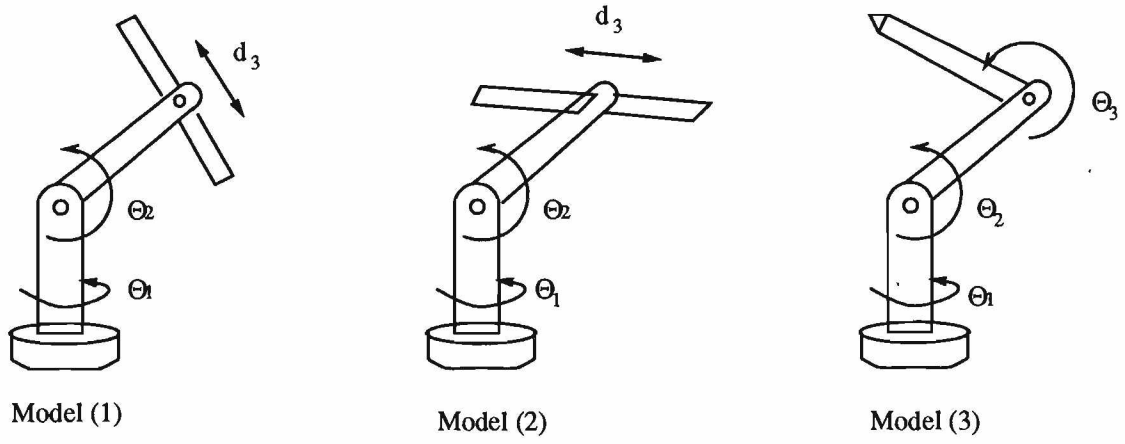

Figure 3: Three different configurations of the robot manipulator.

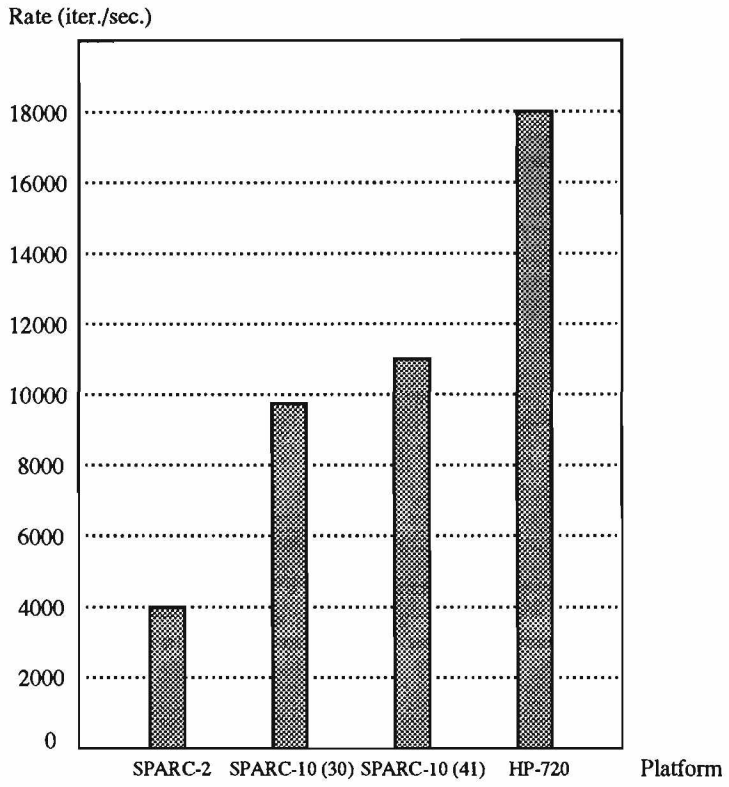

Figure 4: Performance comparison for different platforms.

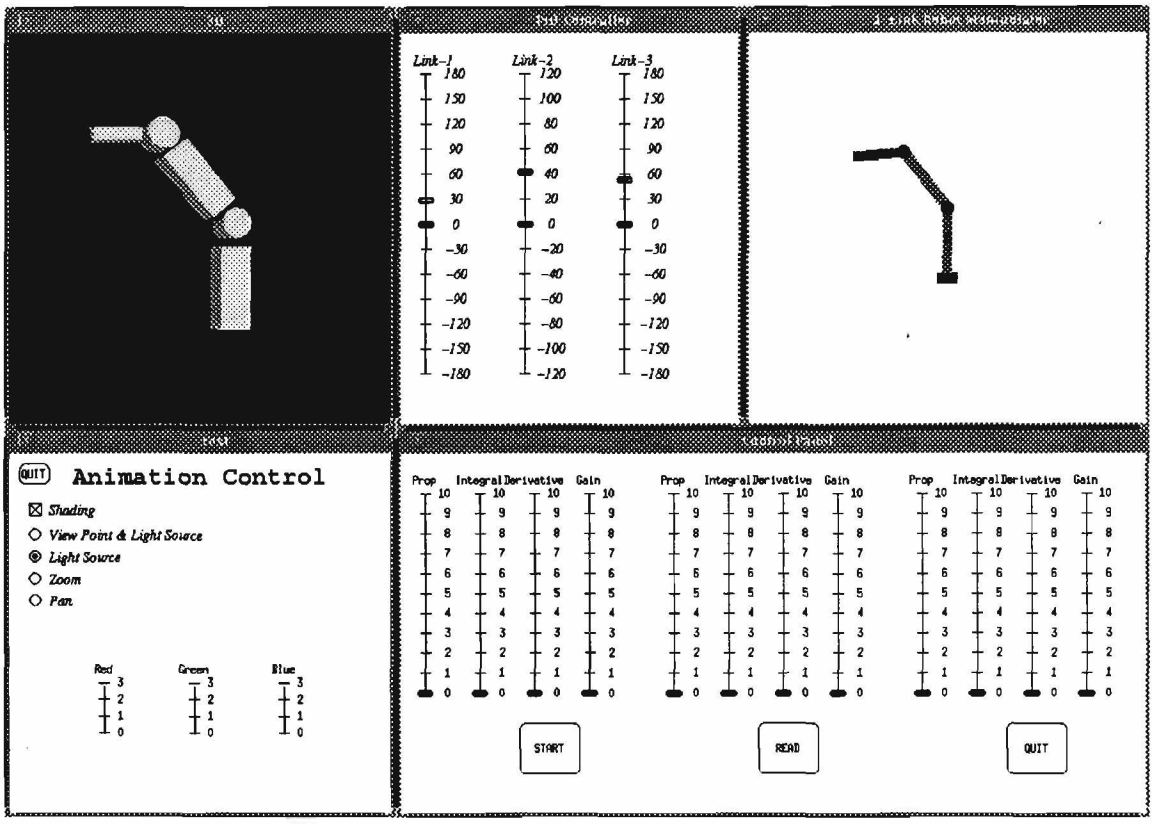

Figure 5: The interface window for the PID controller simulator.

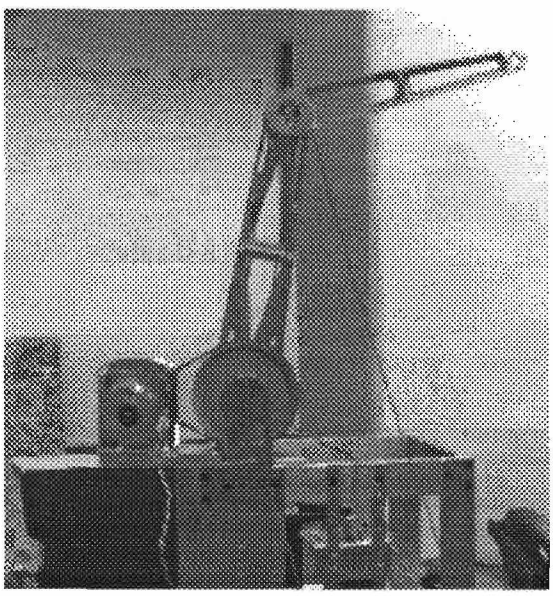

Figure 6: The physical three-link robot manipulator.

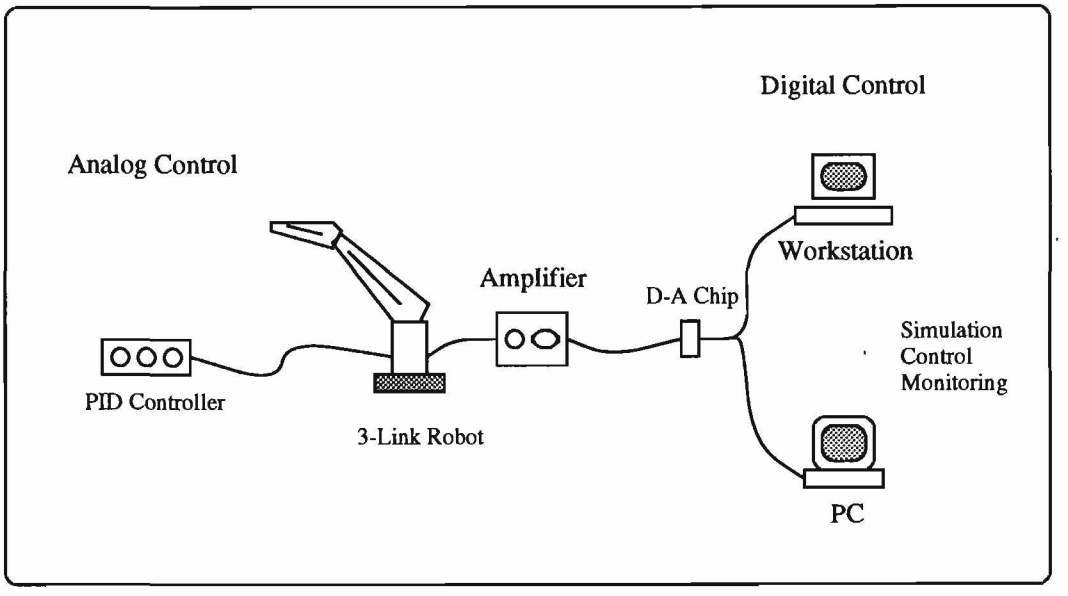

Figure 7: Controlling the robot using different schemes.

(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 7 shows an overall view of the different interfaces and platforms that can control the robot. A summary of this design can be found in [4].

4 Robot-computer Interface

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

4.1 The MC68HC11EVBU Chip

The MC68HC11 MCU device is an advanced single-chip MCU (Micro Control Unit) with on-chip memory and peripheral functions. The EVBU comes with a monitor/debugging program called BUFFALO (Bit User Fast Friendly Aid to Logical Operations), which is contained in the

MCU ROM. User code can be assembled using the line assembler in the BUFFALO monitor program, or else by assembling code on a host computer, and then downloading the code to the EVBU user RAM via Motorola S-records. In the later case the monitor program can be used to debug the assembled user code. There are a lot of utility subroutines in the BUFFALO program that can be used for any program of our own. The MCU that is being used here is MC68HC11E9FN1.

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

The chip that was used a a MCU was the MC68HC11E9, a high-density complementary semiconductor(HCMOS) high- performance microcontroller unit(MCU) includes the following features: 12 Kbytes of ROM, 512 bytes of EEPROM, and 512 bytes of RAM. The MC68HC11E9 is a high-speed, low-power chip with a multiplexed bus capable of running at up to 3 MHz. Its fully static design allows it to operate at frequencies down to dc. For more details about this chip can be found in [11].

4.2 Serial Communications Interface (SCI)

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

4.3 Analog to Digital Converter

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

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

4.4 Digital to Analog Converter

For the D/A conversion, we used an 8-Bit microprocessor compatible, double buffered DAC0830. The DAC0830 is an advanced CMOS 8-bit multiplying DAC designed to interface directly with most of the popular microprocessors. The circuit uses CMOS current switches and control logic to achieve low power consumption and low output leakage current errors. Double buffering

allows these DACs to output a voltage corresponding to one digital word while holding the next digital word. The DAC can be used in different modes of operation.

5 The Prototyping Environment

As mentioned earlier, URK was built at part of a research project for building a prototyping environment for robot manipulators. This prototyping environment (PE) consists of several subsystems such as:

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

Figure 8 shows a schematic view of the prototyping environment with its subsystems and the interface.

The overall design of the PE consists of a *central interface* (CI) and *subsystem interfaces* (SSI). The main tasks of the CI are: maintaining a global database of all the information needed for the design, communicating with the SSIs to update any changes in the system, and checking some design constraints. It also maintains a design history for future reference.

The SSIs serves as interface layers between the CI and the subsystems. This scheme gives more flexibility in the design and enables us to change any subsystem without much change in the rest of the system. The main tasks of the SSI are: reporting any changes in the design to the CI, receiving messages from the CI with the required changes, and update the local database of the subsystem according to the changes received from the CI. Figure 9 shows an overall view of the suggested design.

Some protocols have been implemented to coordinate the communication between the CI and the SSIs, and to control the information flow throughout the system. In this design, all subsystems communicate through the CI which is responsible for passing the information to other subsystems.

Object analysis approach is used to determine the system components and functions, and the relations between them. The top-down approach is used starting from the main objects in the PE, then analyze each of these objects in more detail until the primitive data items

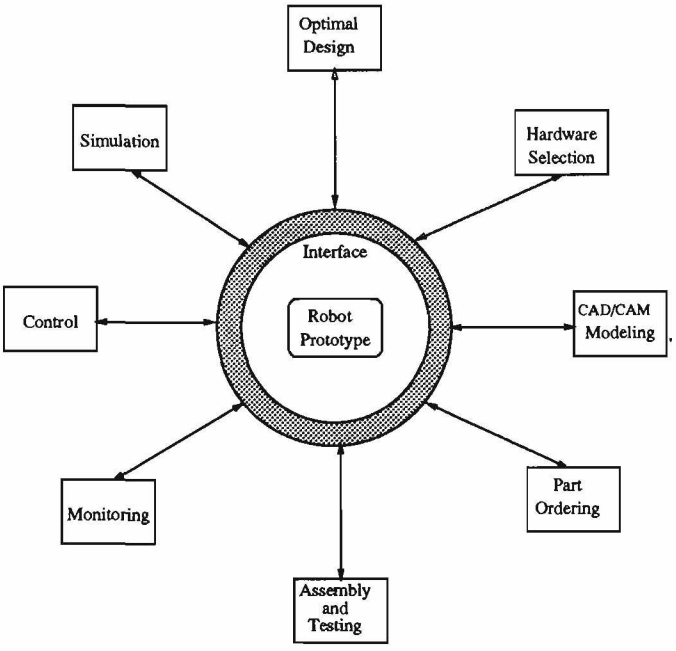

Figure 8: Schematic view for the robot prototyping environment.

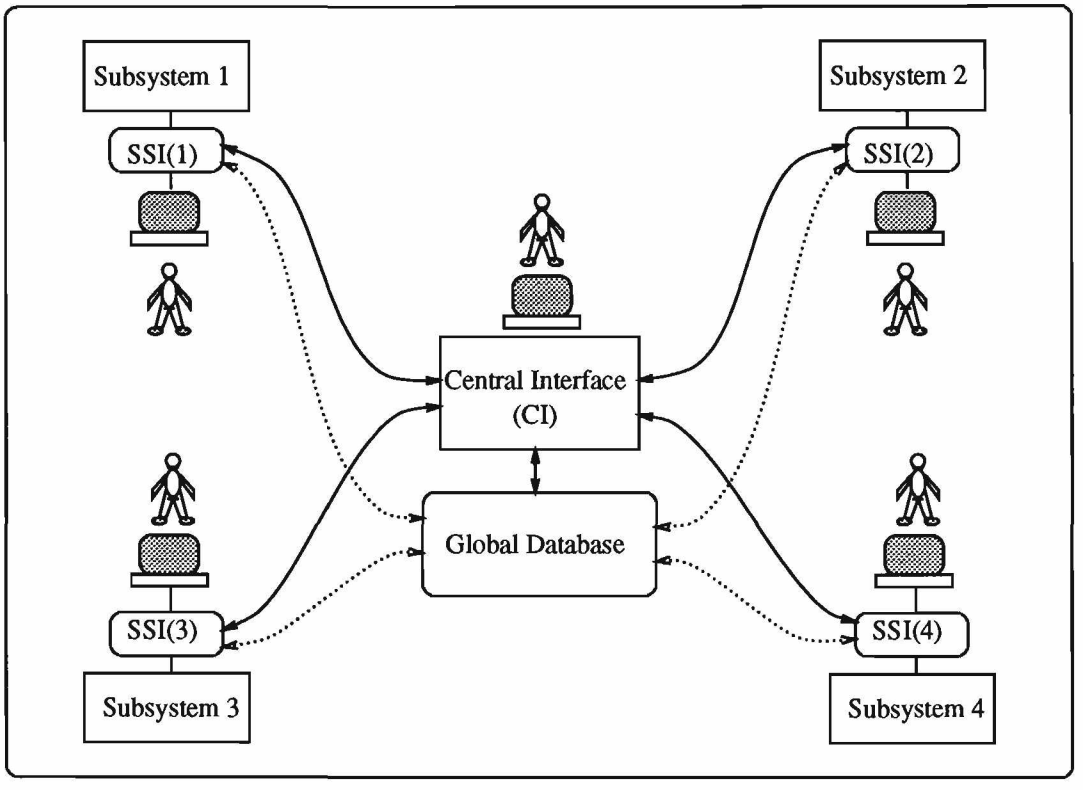

Figure 9: Overall design of the prototyping environment.

are reached. Second, the functionality of the system has been analyzed and described using high level algorithms. Finally the corresponding member functions of the suggested classes has been implemented. Figure 10 shows the top view of the main components in the system, and Figure 11 shows one of these components in detail.

A database for the system components and the design parameters was built. This data base contains information about the robot configuration, the design constraints, subsystem information, design parameters, and some general information about the system.

The first implementation phase of the prototyping environment has been done to demonstrate the functionality of the system. The following is a quick description of the implemented components of the system:

The central interface (CI): which handles the communication between the subsystems, and maintains a global database for the current design and a history of previous designs.

The PE control system (PECS): which provides a graphical user interface to control and manage the coordination between the subsystems. and provides some reports and queries about the system and the design process.

The subsystem interfaces (SSI): which serves as the interface layer between the CI and the user at each subsystem.

A complete description and detailed design of these systems can be found in [5].

6 Testing and Results

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

6.1 One-link Robot

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

Several input sequences have been used for the desired positions, and after applying the voltage to the motor using the I/O card, the actual positions and velocities are measured using a potentiometer for the position, and a tachometer for the angular velocity. These measured values were shown graphically using a program that displays the movement of the link, the desired and actual positions, the desired and actual velocity, and the error in position and velocity. Figure 12, shows the output window displaying the link and graphs for the position and the velocity for one of the test cases.

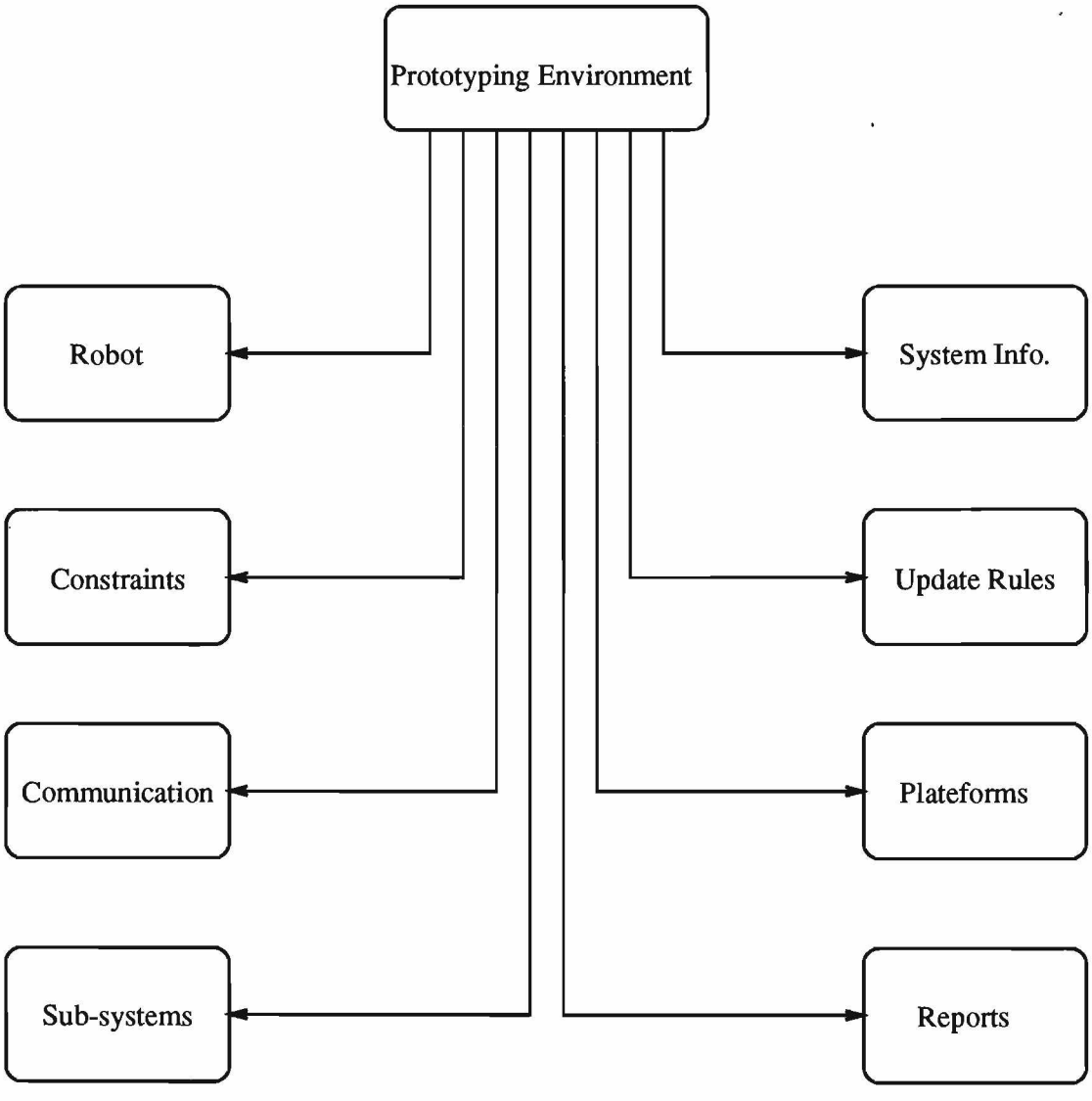

Figure 10: The main components of the robot prototyping environment.

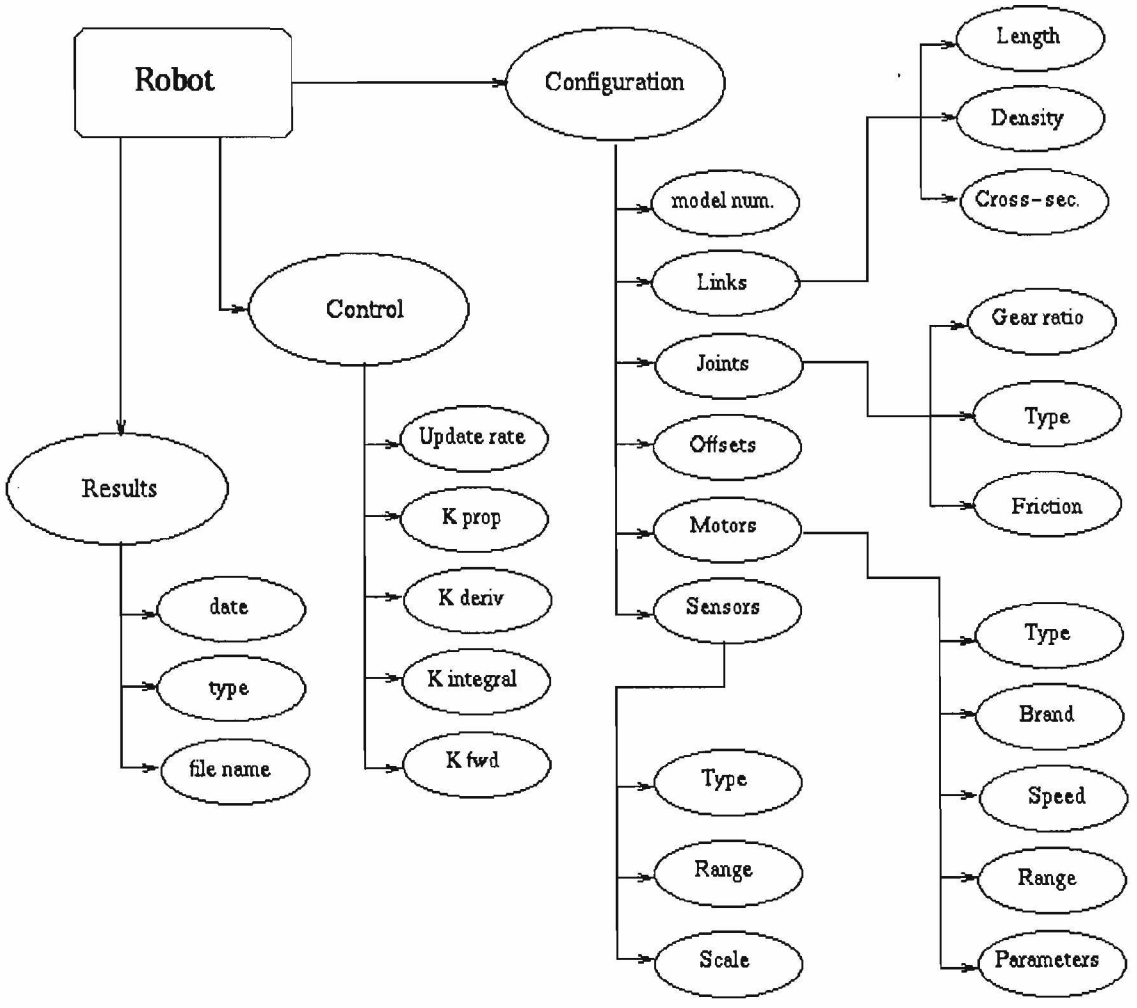

Figure 11: Detailed analysis for the robot classes.

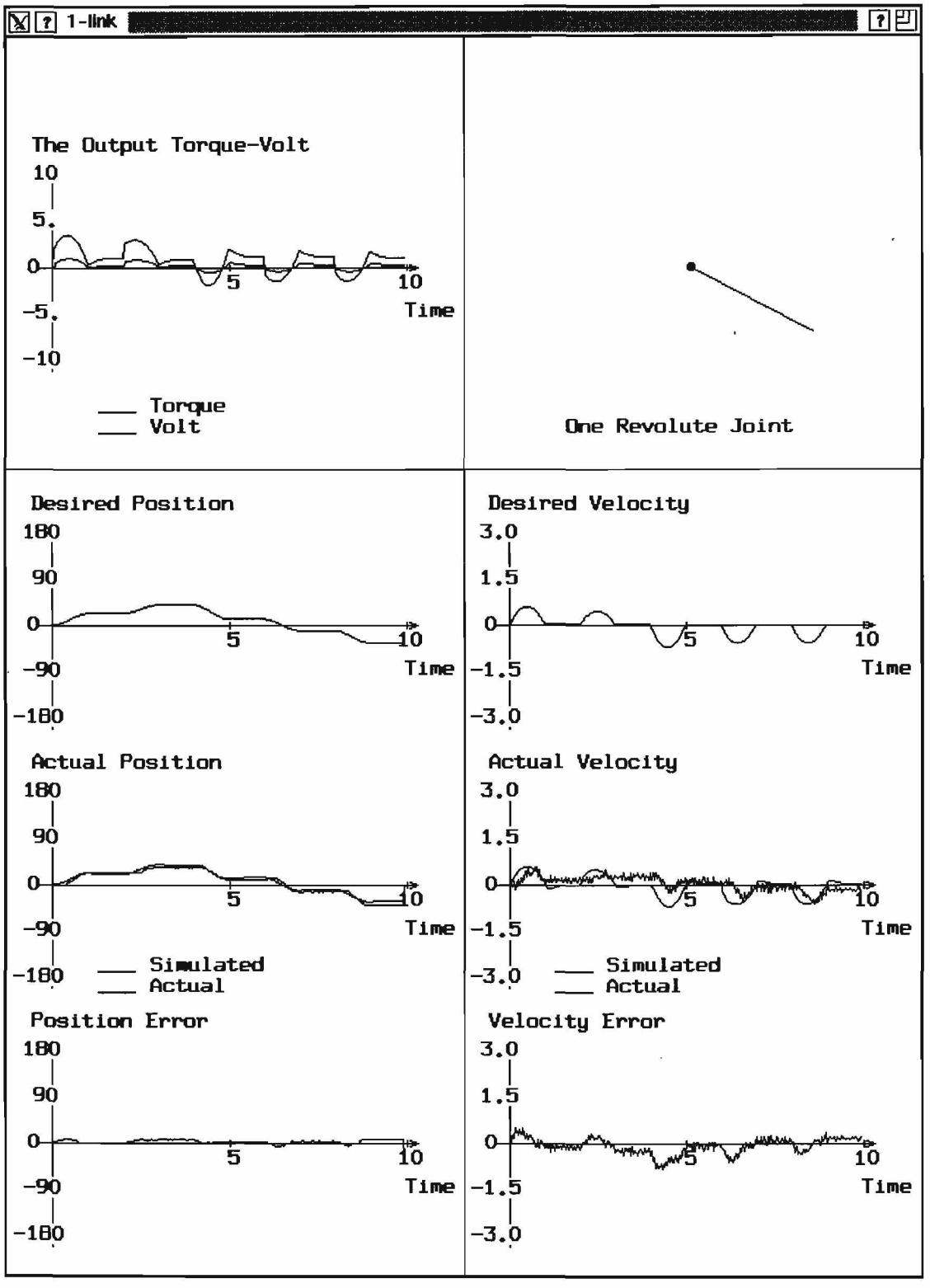

Figure 12: The behavior of the one-link robot.

6.2 Simulator for three-link Robot

This simulator was used to give some rough estimates about the required design parameters such as link lengths, link masses, update rate, feedback gains, etc. It is also used in the benchmarking described earlier. Figure 13 shows the simulated behavior of a three-link robot. It shows the desired and actual position and velocity for each link and the error for each of them. It also shows a line drawing for the robot from two different view points.

This simulator uses an approximate dynamic model for the robot, and it allows any of the design parameters to be changed. For example, the effect of changing the update rate on the position error is shown in Figure 14. From this figure, it is clear that increasing the update rate decreases the position error.

6.3 Software PID Controller

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

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

- forward gain (k_g)
- proportional gain (k_p)
- differential gain (k_v)
- integral gain (k_i)
- input trajectory
- update rate

In these experiments, the program was executed on a Sun SPARCStation-10, and the A/D chip was connected to the serial port of the workstation. One problem we encountered with this workstation is the slow protocol for reading the sensor data, since it waits for an I/O buffer to be filled before it returns control to the program. We tried to change the buffer size or the time-out value that is used, but we had no success in that. This problem causes the update rate to be very low (about 30 times per second), and this affects the positional accuracy of the robot. We were able to solve this problem on an HP-700 machine, and we reached an update rate of 120 times per second which was good enough for our robot. Figure 15 shows the desired and actual position for different test cases using different feedback gains.

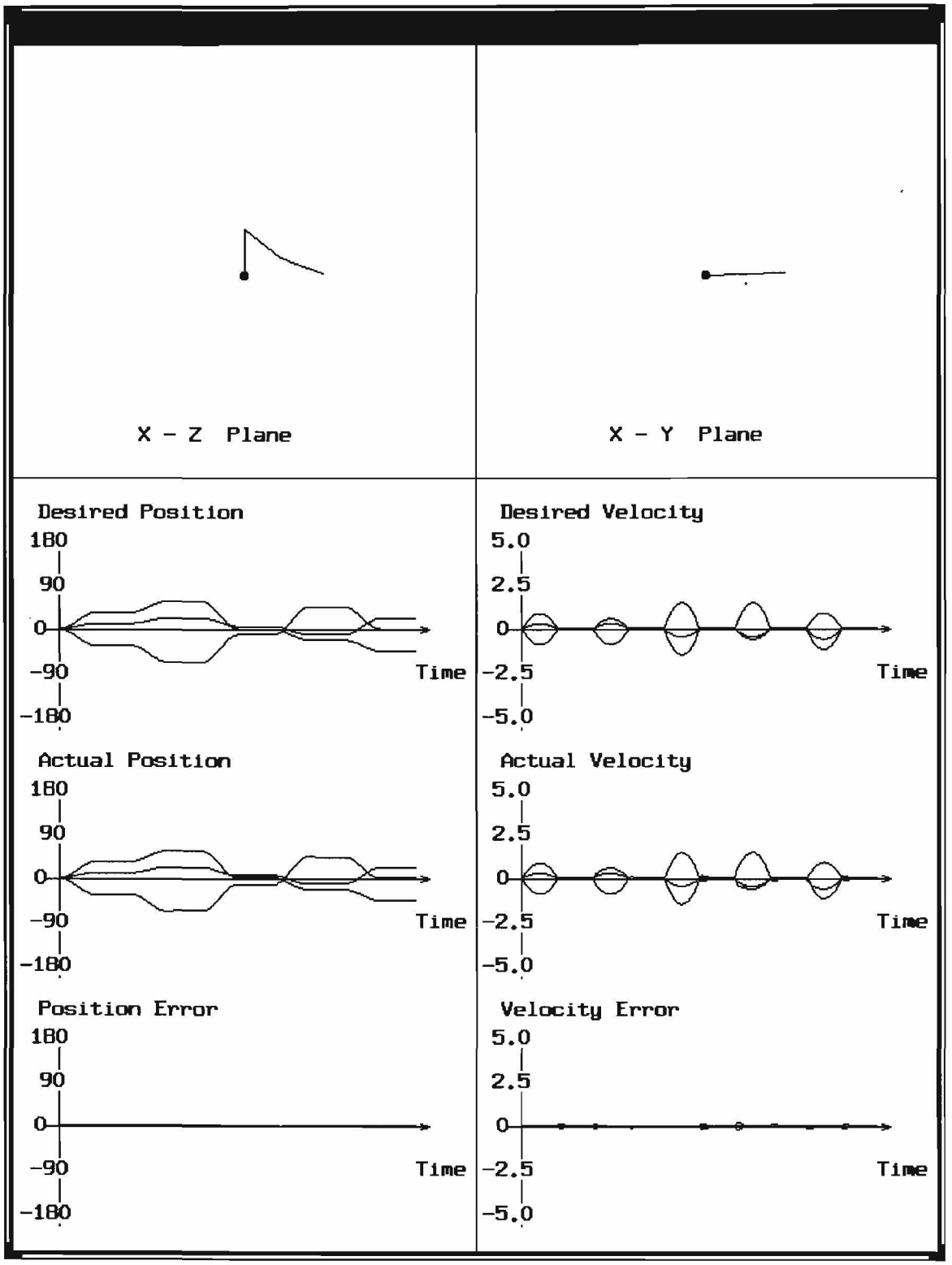

Figure 13: The output window of the simulator for the three-link robot.

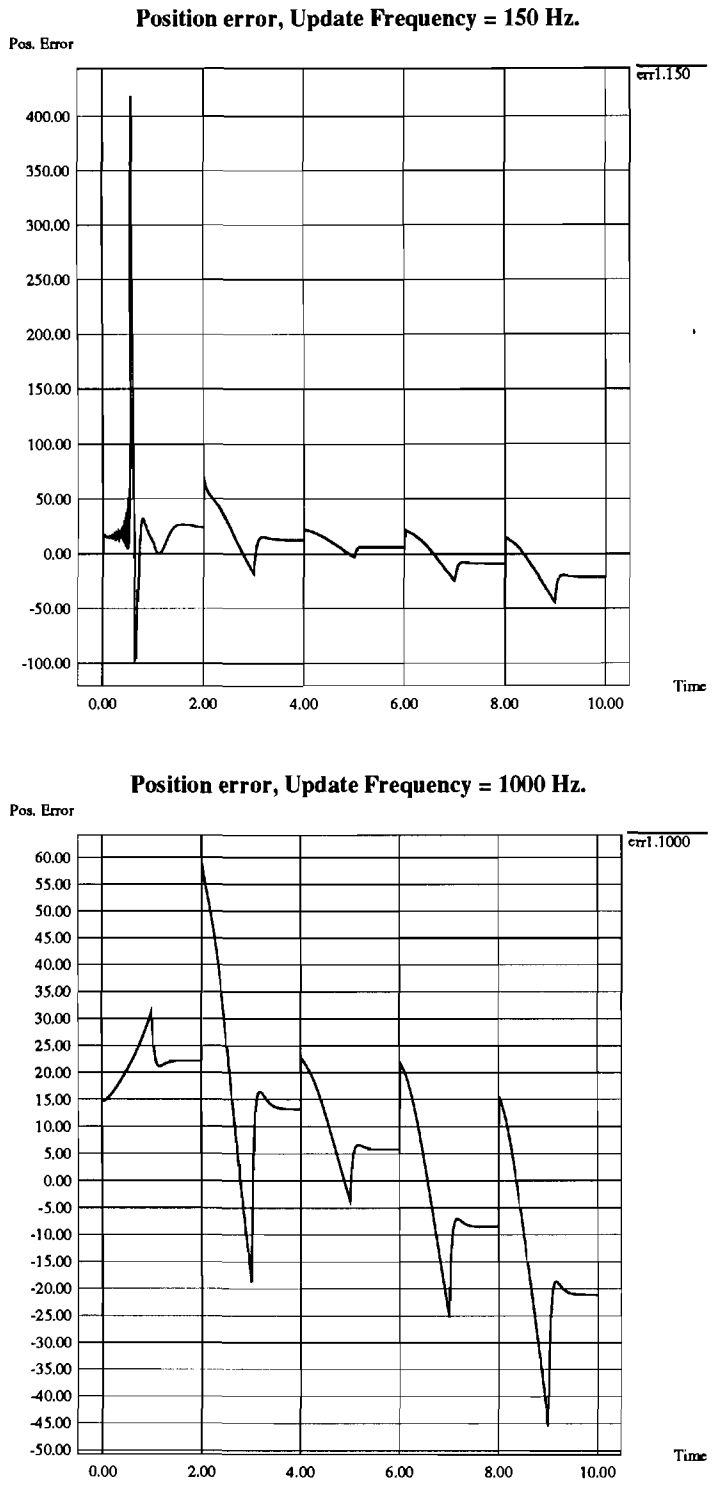

Figure 14: The effect of changing the update rate on the position error.

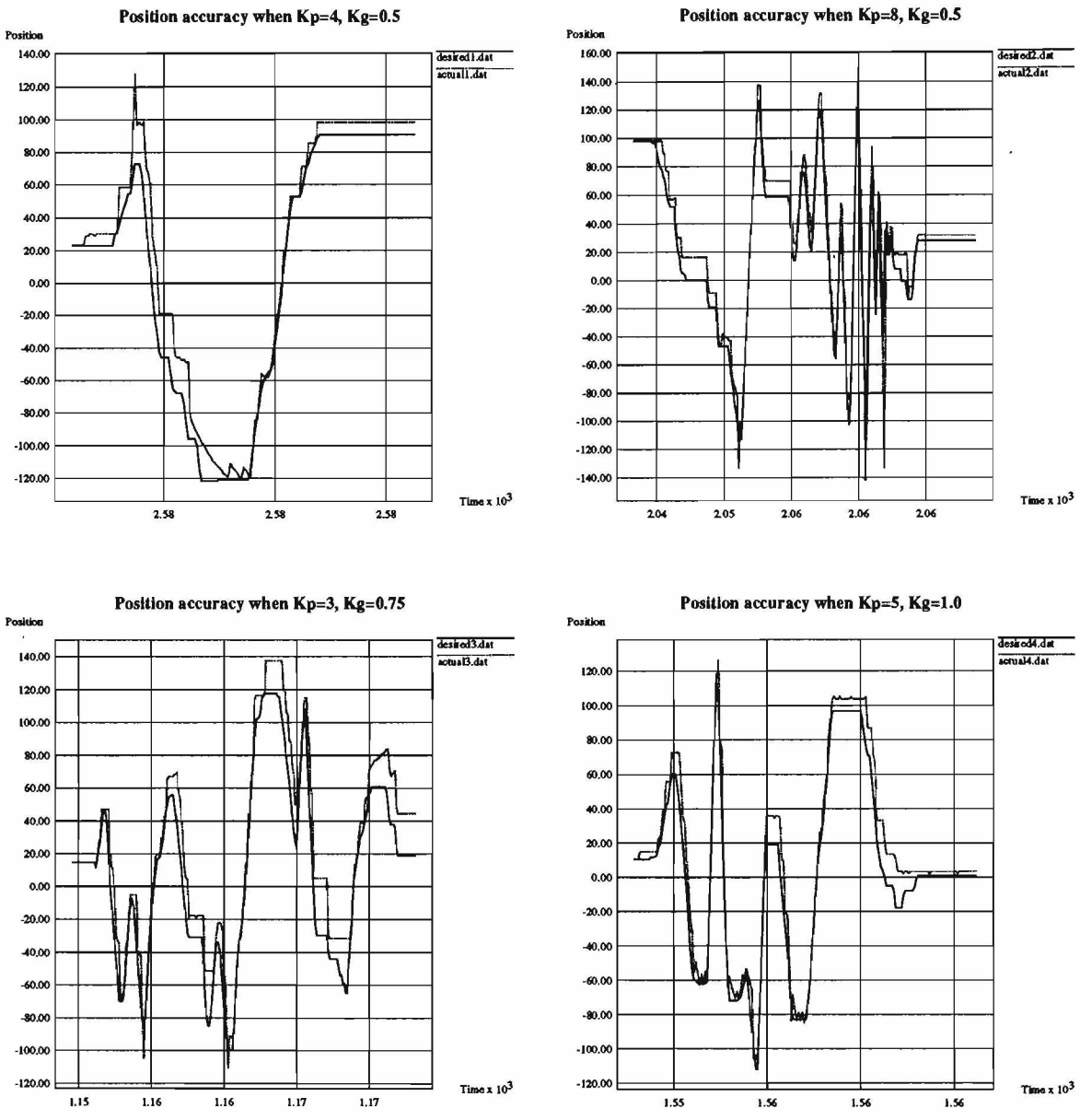

Figure 15: Desired and actual position for several test cases.

7 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.

8 Acknowledgments

We would like to express our thanks to Mircea Cormos, Prof. Sanford Meek, and Prof. Beat Brüderlin for helping make this robot come to life.

References

[1] 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.

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

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

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

[5] DEKHIL, M., SOBH, T. M., HENDERSON, T. C., AND MECKLENBURG, R. Robotic prototyping environment (progress report). Tech. Rep. UUCS-94-004, University of Utah, Feb. 1994.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.