Polymer Gel Spinning Machine

by

Christine A. Odero

B.S., Massachusetts Institute of Technology (1992)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author .....
Department of Electrical Engineering and Computer Science
May 12, 1994

Certified by .....
Steven B. Leeb
Carl Richard Soderberg Assistant Professor of Power Engineering
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler
Chairman, Committee on Graduate Students

Polymer Gel Spinning Machine

by

Christine A. Odero

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Science and Engineering

Abstract

A machine was built to produce thin fibers of polymer gel. Gels are crosslinked networks of polymers suspended in a solvent. Under certain conditions, gels have been observed to undergo reversible, discontinuous changes in volume. Gel fibers could be used to create flexible actuators whose mass can be distributed conformally with the underlying mechanical structure of a servomechanism. Potential actuator flexibility, force and speed is being actively researched in the Laboratory for Electromagnetism and Electronic Systems, by investigating gels of different chemical makeup, and fibers of different dimensions. These gels are currently being manually spun into the required fibers and gathered into bundles. The construction of a machine to automate production is desired to make the process more efficient. The machine incorporates components to spin a polymer solution into fibers, cross-link the fibers into gel fibers and gather the fibers to form bundled fibers. The spinning process involves motor speed control to control the dimensions of the fiber. Position control is important in the cross-linking and gathering stages. A microprocessor oversees the fiber spinning process, aiding in motor control and in synchronization of the various stages of production.

Thesis Supervisor: Steven B. Leeb
Title: Carl Richard Soderberg Assistant Professor of Power Engineering

Acknowledgments

I would like to first thank God and my parents for the ever-present support.

Also to Steve Leeb and Ahmed Mitwalii, my eternal thanks for guiding me through this project. To Steve Shaw, forever grateful. Further invaluable help was provided by Felice Swapp, Umair Khan, Justin Foreman and John Ofori.

This research was conducted in the Laboratory for Electromagnetic and Electronic Systems. Invaluable funding, materials, and facilities were made avaiable by Professor Toyoichi Tanaka, Provost Mark Wrighton, the AT&T New Venture Fund, Intel Corporation, and the Center for Materials Science and Engineering.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The thesis will present the research work undertaken in building a polymer gel spinning machine. The machine takes a polymer solution and spins it into fibers which are then cross-linked and bundled together.

The first step in this chapter is to establish the need for this machine. The importance of the gels being researched will be discussed, followed by the project aim. Finally, the main challenges and broad machine specifications will be presented.

1.1 Gels

Gels are a cross-linked network of polymers suspended in a solvent [24], [16]. Different polymer gels have been found to exhibit a sudden, reversible, discontinuous change in volume in response to various stimuli including light ([2]), voltage ([25], [23]) and temperature. Different forces within the gel combine to form what is known as the osmotic pressure of the gel. The stimuli mentioned act to imbalance these forces, causing the gel to expand or shrink towards a new equilibrium point. The expansion or contraction converts some energy to mechanical energy which could be used in robotic or other servo-mechanical applications [19].

As an example, temperature-sensitive N-isopropylacrylamide, or NIPA gel, exhibits this abrupt change in volume at approximately $34°C$. At lower temperatures, the gel swells, and as its temperature is raised past the transition temperature, the gel expels the solvent and shrinks in volume [21].

Current research in the Laboratory for Electromagnetic and Electronic Systems is aimed toward fabricating linear actuators. Polymer gel actuators promise many advantages over currently used actuators. Polymer gels are flexible where other actuators such as the electric motor are heavy and fixed in shape. Polymer gels can be constructed in the shape suited for their application in terms of the range of motion required and the weight distribution or inertia requirements [19]. The production

of a linear actuator calls for fiber shaped gels. The fiber design parameters include the gel's response time, tensile strength, efficiency, behavior under different loading conditions, and dynamic range or linear stroke [19]. Since the speed of response has been found to vary inversely with the square of the fiber diameter [24], thin fibers are preferred to thick cylinders of gel. The fibers currently being fabricated are on the order of tens of micrometers in thickness. However, the thin fibers tend to be weak so many fibers are bundled together to increase strength without compromising speed.

Work is currently being done to characterize the strength and speed of various polymer gels and therefore, their potential for use as novel actuators. One idea is to use the actuators to form "artificial muscles" which will be used in multi-joint manipulators. The polymer gel actuators are the close in flexibility and range of motion to the human muscle [13].

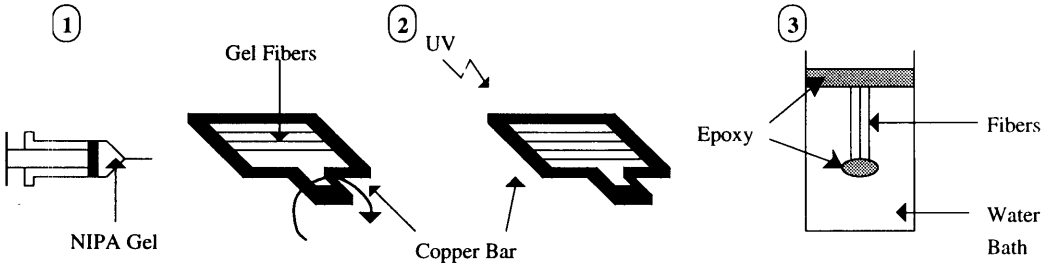

Figure 1-1: Fabricating and Testing NIPA Gel Fibers for Linear Actuators.

The preliminary experiments being carried out are outlined in Figure 1-1 which shows the current fabrication and testing techniques, taken from [21]. In Step 1 of Figure 1-1, a syringe loaded with the NIPA gel is used to spray the viscous stream of polymer solution onto a rotating copper bar. The bar is connected to a motor which is driven by a variac. When spraying is complete, the fibers are cross-linked with ultra-violet light in Step 2. The cross-linking is anisotropic so that when the gel is suspended in a solvent such as water and subjected to heat, it will contract or expand longitudinally. After the cross-linking step, the fibers are bundled and bound with epoxy at each end. The experimental stage, Step 3, consists of running experiments on the bundled fibers with one end fixed and the other suspended in a temperature-controlled water bath. When the water bath is heated, the fibers contract, when the the temperature is decreased, they expand [21]. Tensile strength can be examined by loading the free end of the bundled fibers before running the heat experiments.

There are many parameters involved in the gel production and many different experiments to carry out on the fibers. This involves many iterations of the fiber production cycle. It is desirable, therefore, to build a machine to automate the production of gel fibers for use in the laboratory experiments.

1.2 Project Aim and Challenges

The aim of this project is to design and create a machine that efficiently and quickly produces gel fibers in sufficient quantity for constructing laboratory-scale gel actuators.

The main challenge involves building a machine that automates as much of the fiber bundle production as possible. The design chosen was a machine that would perform spinning and bundling starting from a pre-gel solution.

Since the fiber diameter is an important parameter that governs the speed of the response of the fibers, it is necessary to have an accurate way of producing fibers with precisely controlled diameters. The speed with which the solution is spun, as well as the volume velocity of the solution ejected in the spin stage, can be used to vary the fiber diameter. It is evident that speed control will be an important issue in the spinning stage.

The machine built will have to be able to produce many runs of bundled fibers of different diameters and lengths. Speed control sets the diameter, as discussed previously. The length of the fiber is set by the dimensions of the apparatus on which it is spun. The machine construction would thus need to be flexible to produce the different fibers. It was foreseen that the object on which the solution was spun (spin object) would be removable. The machine performance should be able to adjust to the different spin objects used, that is, the control will need to be robust to these variations, or alternatively, adaptive control could be used.

Another challenge involves calibrating the machine so that fiber dimensions could be related to machine setpoints

1.3 Thesis Outline

An overview of the machine is presented in the Chapter 2. The chapter links the concepts of what the machine is meant to do with the proposed design. The various stages in the production of the bundled fibers and the control of the process is then discussed in detail. Chapters 3 discusses the digital hardware implementation which is responsible for controlling most of the machine tasks. Chapter 4 deals with the control of the machine and includes the mathematical models which were developed to control some machine components. Chapter 5 discusses the testing of the parts and the whole of the machine. The performance of the machine is evaluated listing its strong and weak points. The conclusion, Chapter 6, summarizes the main points of the thesis before making suggestions for completing and improving the machine. Appendices A, B and C include code used in simulation, in the microprocessor and in the programmable memory chips respectively. Appendix D contains the observations used to calculate various parameters used in the mathematical models. Appendix

E contains detailed theory discussion not included in the main thesis. Appendix F contains the circuit schematics.

Chapter 2

Machine Design

This chapter discusses the machine design. The main features are outlined followed by sections detailing the various stages of the fiber production process. Machine control is then briefly discussed.

2.1 Process Overview

A schematic diagram of the mechanical structure of the machine is illustrated in Figure 2-1.

The machine consists of a drum spun by a shaft motor whose speed is monitored through the use of a shaft encoder. The shaft motor speed and position is controlled by the electrical circuitry represented by the controller block. The controller block includes an 80C196KC microprocessor. The control and/or signal lines to and from the controller are represented by the lines shown on Figure 2-1. A gel extruder is mounted on a platform which can be moved alongside the rotating drum. A mechanical arm which is used to gather up the fibers into bundles is mounted alongside the extruder. A lead screw and table actuated by a stepper motor moves the extruder and assembly arm along the length of the drum. The extruder content (pregel solution), contained in a syringe, is ejected with the help of a second stepper motor acting on the syringe plunger. The UV light used to cross-link the fibers is mounted on a platform above the rotating drum. The whole setup is encased so that nitrogen gas can be pumped in during the spinning operation. This is to drive out oxygen whose molecules are found to interfere with the cross-linking of the gel.

2.1.1 Spinning Pre-Gel

The spinning of the pre-gel solution into fibers will be done using a rotating drum. The drum consists of two round end plates connected by radially spaced rods. As the

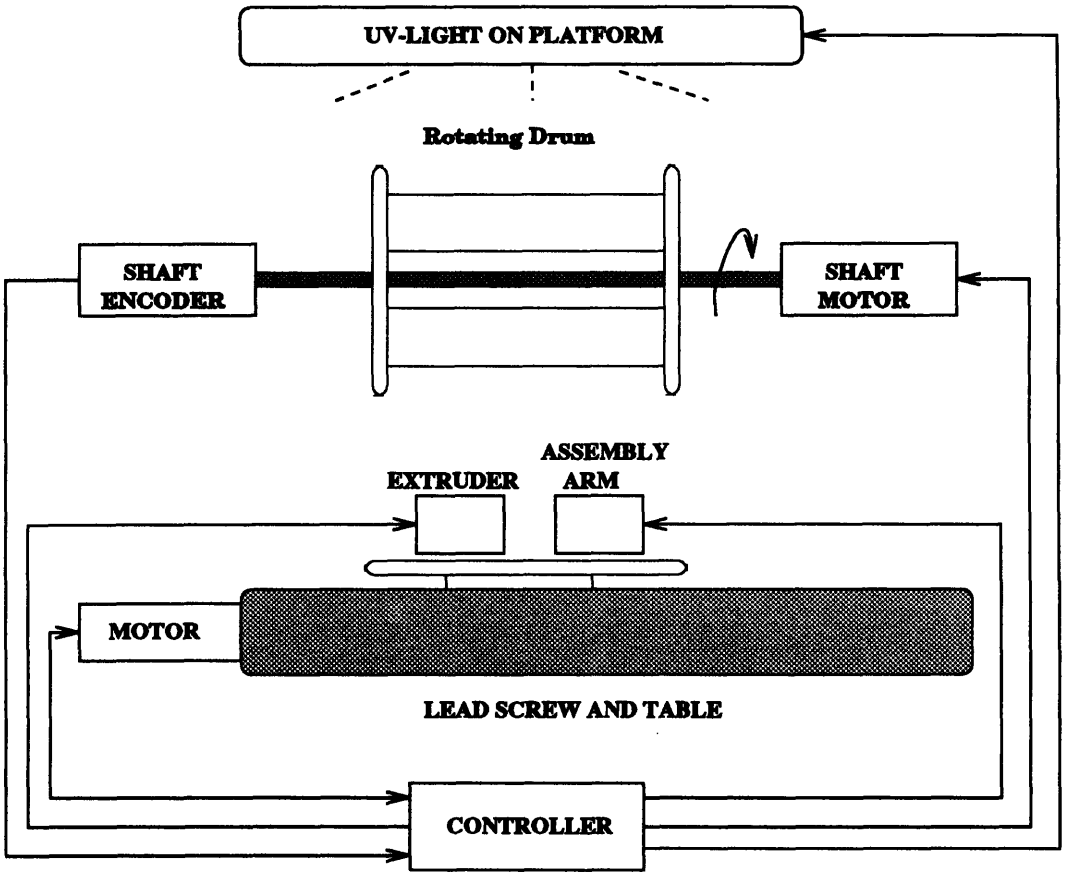

Figure 2-1: Machine Setup.

drum rotates, the thick gel is lightly brought into contact with it at one end. The pre-gel catches onto the rods which make up the drum, and the gel is pulled along as the drum spins. Once the gel catches on, a syringe in the extruder is used to squeeze out the gel. As the extruder moves alongside the drum the pre-gel is pulled out continuously to make fibers. The pre-gel is pulled by the spinning wheel when rotated, so that the fibers produced will hopefully react to stimuli anisotropically, in the axial direction. As mentioned before, it is desirable to make the fibers thin to increase their response rate.

A permanent magnet dc motor (shaft motor) is responsible for the drum rotation. Its speed must be kept constant during the spinning process so that all the fibers in that run have the same diameter. The spin speed is proportional to the fiber diameter, and will be used to predict this vital parameter.

The extruder is mounted on a linear drive table. A syringe in the extruder sprays the pre-gel solution onto the rotating drum. The drive table is mounted on a lead screw which is turned by a stepper motor. The motor speed here is critical in ensuring proper spacing as the pre-gel is sprayed onto the drum. If the drive table speed is too slow, bunching up might take place, where the next ring of fiber sprayed overlaps the previous one. A stepper motor in this case is a good choice because the lateral movement can be controlled precisely at the desired linear velocity without necessarily requiring closed loop control.

A second stepper motor powers the extruder through a lead screw attached to a block of aluminium which in turn, pushes on the end of a syringe loaded with the pre-gel solution. The solution is pushed out as the drive table's lead screw motor moves the syringe laterally across the drum.

Pre-gel fibers produced by this method are, at this point for, ready for cross-linking.

2.1.2 Cross-linking

A platform houses a UV lamp which is turned on when the gel has been spun onto the drum. The UV-light cross-links the polymer solution, turning it into a gel. The shaft motor is rotated every ten minutes to make sure the gel is exposed uniformly to the UV-light during cross-linking.

2.1.3 Collecting Gel

The gathering of the cross-linked fibers is done with the aid of an arm protruding from a solenoid. The solenoid is mounted on the drive table alongside the syringe. The arm consists of two "pickers" which can fit in between the rods on the rotating drum. Epoxy dots are fixed on the arm ends to pick up the fibers.

The drum is first positioned by the controller so that when the arm is pushed out, its sticky fingers will be positioned in between two rods. A solenoid is then activated, which pushes out the arm. The drive table then moves along the drum and the fibers are picked up. When the end of the drum is reached, the solenoid is deactivated, pulling back on the arm. The drive table is then brought back to its initial position. The shaft motor then repositions the drum, allowing the solenoid to pick up the next line of fibers.

2.1.4 Control

An 80C196KC Intel microprocessor will be used to sequence and control the entire process, from the initial spinning of the fiber, to the turning off of the machine once the fibers have been bundled.

The main aspects of microprocessor control will involve speed control of the rotating drum during the spinning stage, drum position control during the gathering stage, setting the stepper motor speeds during the spinning and gathering stages, and synchronizing the entire procedure.

2.2 Spinning

This section details the design and implementation of the components used in spinning the gel. A detailed flowchart is presented in Figure 2-2 showing the actions that take place during a sample spinning run.

The drum starts rotating and when it has reached the desired speed setting, the linear carriage is set in motion and positioned at the point along the drum where pre-gel spraying is to begin. The linear carriage and the extruder stepper rates are set to the desired values and their motors enabled. The pre-gel solution is sprayed onto the rods until the specified end point is reached. The steppers are then disabled and the linear carriage returned to its original position. At this point the drum stops spinning. Precise details of control and synchronization will be explained in Chapters 4 and 5.

2.2.1 Rotating Drum

The drum consists of two thin acrylic plates which are joined by several glass rods. The glass rods are placed radially and spaced equally around the circumference of the plates. (Figure 2-3)

The materials used were chosen for easy cleaning and machining. The glass rods are placed close to the edge of the plates so that the stroke of the collecting arm will not have to be too great to enter and clear the drum. Different lengths of fibers

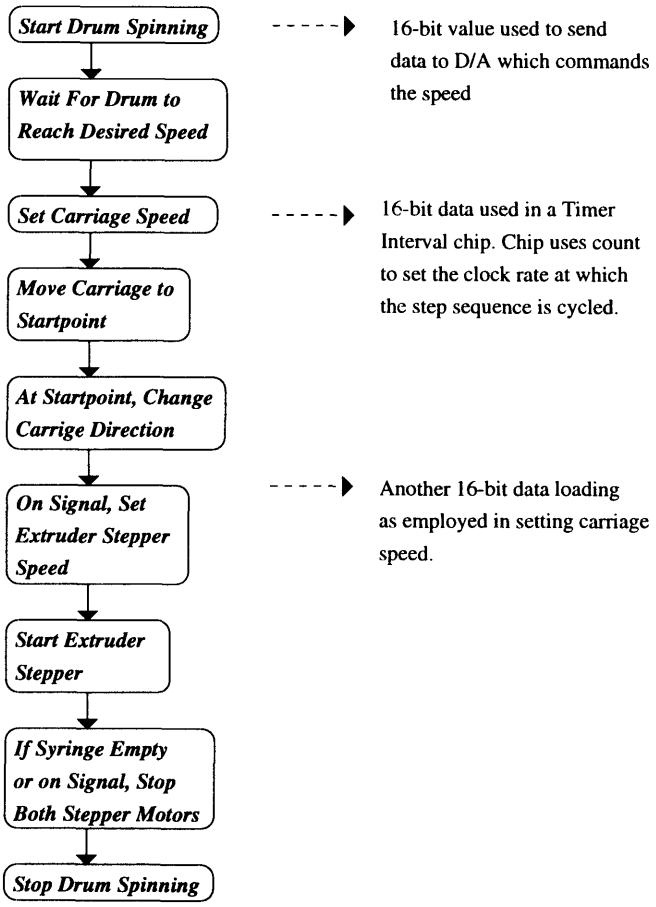

Figure 2-2: Gel Spin Flowchart

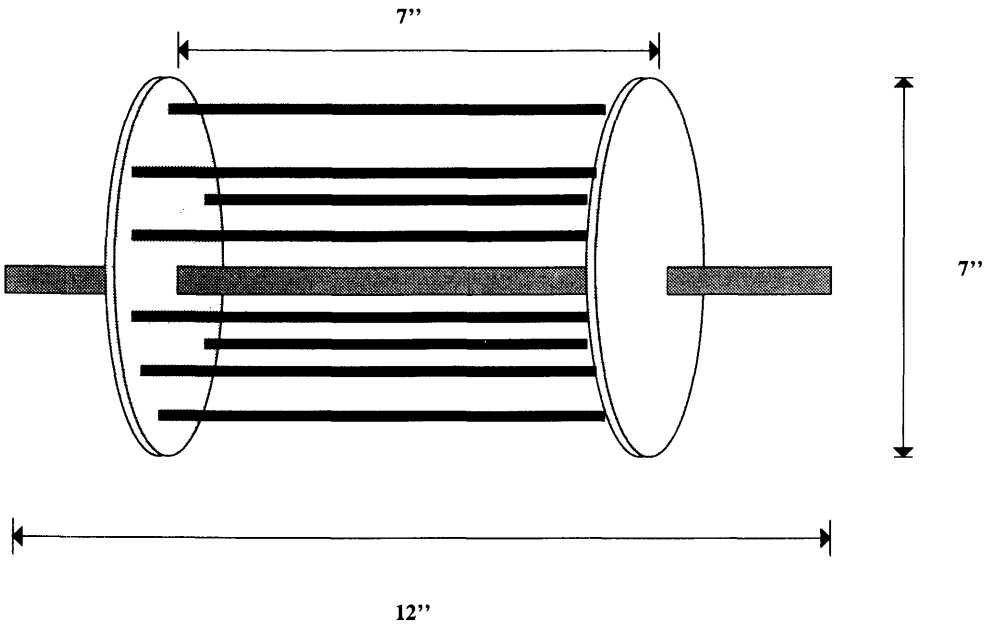

Figure 2-3: Rotating Drum

will be needed, and thus different drums were machined. The number of rods used determines the length of the fibers, i.e., the shorter the fibers, the more the rods. The fiber lengths, or rod spacing cannot be too long or the fibers which have just been spun, and are still weak before cross-linking, are likely to break. The maximum allowable spacing is about 5cm.

There is a center rod (shaft), through the drum which connects the drum, on one end to the shaft motor and on the other, to the shaft encoder. The coupling of the shaft to the encoder and motor was designed to enable easy changing of drums. Flexible couplings are thus used on either end.

2.2.2 Linear Carriage

There is a need for an apparatus which would move the extruder along the drum during the spraying of the pre-gel solution. The required linear motion suggested the use of a lead-screw on a motor. If the operational speeds are low, then a stepper motor is a good choice because the number of discrete angular steps the stepper motor makes is equal to the number of input pulses, allowing the stepper motor to be operated open loop [22]. The linear table setup thus consists of a stepper motor driving a lead-screw on which a carriage is mounted. The extruder is mounted on the carriage.

The length of the lead-screw needed is set by the length of the drum, which in turn

is set by the length of the glass rods. This was about 7 inches. The lead-screw length needs to be longer to utilize the whole length of the drum when spinning the solution. The other issue in choosing the linear actuator involved picking one which would be able to support the weight of both the extruder and the assembly arm, at the speeds of operation. A Techno Motors [1] lead-screw drive table with an attached stepper motor met the anticipated requirements. Its stepper motor can be configured for both bipolar and unipolar operations. In bipolar mode, the windings are switched between positive and negative voltages whereas in unipolar, the winding voltages are either positive or zero. We chose unipolar operation to simplify the power electronic drive requirements. The main disadvantage of unipolar operation over bipolar operation is that the torque available is lower [3]. However, the drive circuit is simpler because full bridge inverters are not required. A full "step" or step angle of the motor corresponds to 1.8°, and a linear motion of 5 mm, which was considered adequate for this work. The linear actuator chosen also had a limit switch which was used in the reference positioning of the carriage before spin and gathering operations.

The inputs to the stepper motor are a set of signals which activate the stepper windings in the proper sequence to achieve rotation in a given direction. (Figure 2-4) In our case, we need four drive signals since the unipolar configuration is used.

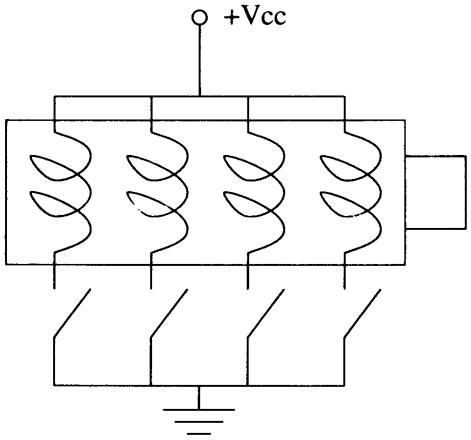

Figure 2-4: Stepper Unipolar Configuration.

The frequency with which the sequence is stepped through directly controls the rotational speed. Feedback control is unnecessary for the low operating speeds anticipated. The clock rate which sets the sequence cycle rate, is a measure of the linear speed of the carriage. The carriage moves 5mm per revolution with a maximum mass of 2.5 kg according to the lead-screw specifications. The current loading on the carriage is less than 2.5 kg. The linear speed of the carriage is then calculated at $5 * F/200$ $[mm/second]$, where F is the clock input to the stepper board i.e., the step rate, 5 mm is the linear motion per step, and 200 is the steps per revolution for a

1.8° step angle.

Controlling the speed of the carriage involves generating the switch signals, see Figure 2-4, at the desired rate. Figure 2-5 shows the control block diagram. The rate is generated and fed into the block which generates the right logic sequence given the rate information, the rotation direction, and the enable signals. The logic outputs are not powerful enough to activate the stepper windings. Amplification is achieved with a 4 channel push-pull driver (LM18293) followed by MOSFETs capable of absorbing the high currents needed to drive the linear actuator. The details are shown in the schematics in Appendix F.

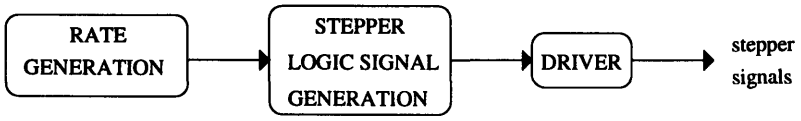

Figure 2-5: Stepper Drive

2.2.3 Extruder

The extruder shown in Figure 2-6 pushes the pre-gel solution onto the glass rods of the rotating drum. It consists of a syringe whose contents are pushed out with the help of a second stepper motor. The stepper shaft is coupled to a lead-screw which is screwed into a rectangular block of aluminium positioned against the syringe end. As the lead screw turns, the block moves forward, pushing the syringe plunger and forcing the solution out. The stepper-syringe assembly is adjustable for height and for angle of delivery. The lead-screw has 20 threads per inch, corresponding to 1.27 mm per revolution and $1.27 * F/48$ $[mm/second]$ where F is the step rate and 7.5° the step angle corresponding to 48 steps per revolution.

The pre-gel delivery rate is important in determining the fiber diameter. A stepper motor is again chosen here for its accuracy at low speeds, eliminating the need for closed loop control. The flow rate is related to the step rate and the syringe dimensions.

The stepper rate control circuitry is the similar to the linear carriage's circuitry. The circuits implemented are the same except that the extruder stepper motor has lower current requirements and MOSFETs are not required to follow the push-pull drivers (details in Appendix F).

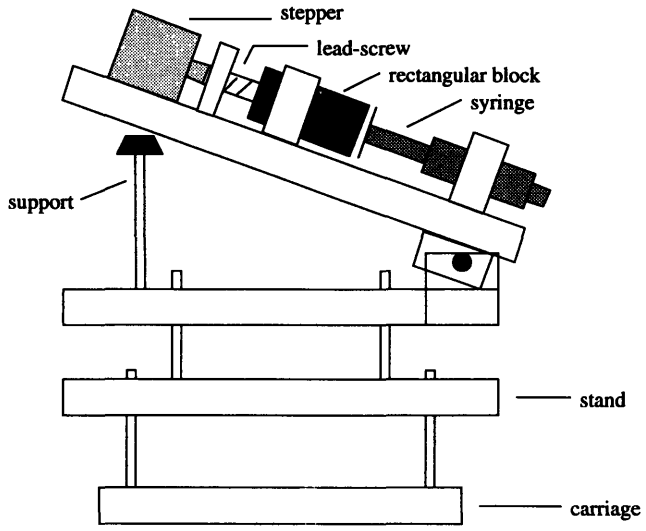

Figure 2-6: Extruder Setup.

2.2.4 Spin Sequence Signals

Infra-red (IR) Emitters are lined up on a strip in front of the linear drive table. They are lined up opposite the drum at the positions where different operations might be performed. An IR detector is attached on the underside of the carriage to signal the microprocessor to undertake the various functions. For example, currently there are four emitters. The first can signal the start of spraying, the second the change of flow rate speed to produce fibers of a different dimension, the third a similar function to the second, and the fourth will be used to signal that the end point has been reached.

2.3 Cross-linking

An ultra-violet (UV) lamp mounted on a platform cross-links the pre-gel to form gel fibers. For safety reasons, the machine will need to be covered with an opaque box during this stage. The UV light is not activated until the casing is closed. The UV lamp and safety interlock are controlled by the microprocessor.

Figure 2-7 shows the circuit used to turn on the UV lamp. The reactance ballast is used to induce a momentary high voltage to start the lamp. There are two switches used, a power switch and a starter switch which is turned on momentarily. During the interval in which the starter switch is on, a current path is formed through the two filaments and the ballast. When the starter switch is disconnected, there is a high voltage buildup in the ballast. This high voltage ionizes the gas in between

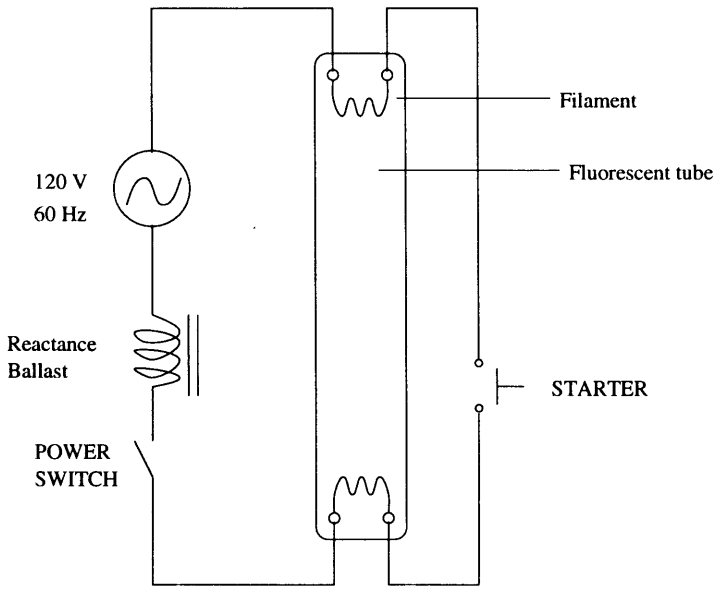

Figure 2-7: UV Lamp Circuit

the filaments. Current is subsequently conducted through the fluorescent tube from filament to filament, through the ballast and the power supply. As electrons move through the tube, they collide with the ionized gas knocking its electrons into higher energy levels. When these electrons fall back into the stabler energy levels, they give out the excess energy in the form of ultraviolet light [15].

Relays are used to turn on the starter and power switches. The starter switch relay is momentarily activated through a 555 timer which provides a one second, one shot pulse. This pulse is an input to the base of a transistor which drives the relay's inductor. A second transistor drives the power switch relay which is activated at the same time as the starter switch relay. This relay however, is on for as long as the lamp is on. The microprocessor or a switch can be used to provide the signal to start the process. The start signal passes through circuitry which generates the correct relay signals. The detailed circuit is included in Appendix F. A safety feature is a mechanism by which the signals will not be generated unless the entire apparatus is covered. A simple switch on the cover lock informs the circuitry that the cover is in place. This switch then enables the relay switching.

To make sure cross-linking is uniform, after every ten minutes the drum is rotated a certain amount. The cross-linking step takes approximately one hour.

2.4 Constructing Fiber Bundles

This section discusses the design and implementation of the apparatus used to collect the fibers and bundle them to form the final product.

An arm with two sticky fingers is attached to a solenoid which, when activated, pushes forward to begin collecting the fibers. The solenoid and arm are mounted on the linear carriage, next to the extruder, so that they can move alongside the drum.

2.4.1 Solenoid

A solenoid was chosen to insert and remove the arm from in between the drum rods. An estimate of about an inch of travel was required for the arm to enter and clear the drum. A rod, acting as an arm is readily attached to the solenoid, and can be activated mechanically by means of a relay, or electrically by employing MOSFETs as switches.

The solenoid consists of a rod or plunger, that is partly surrounded by two windings. The first wound coil operates at a high current level to provide the maximum push. The second wound coil simply holds the plunger in place after it has completed its stroke. The rod is attached to a return spring. In the de-energized state, the return spring is unstretched and the rod is retracted. When the solenoid is activated, the high current push coil forces the solenoid rod and the gathering arm forward until the magnetic circuit permeability rises when the rod completely enters the windings. At this point, the lower current hold coil takes over and holds the rod in position. The return spring is stretched at this point, but when the solenoid is de-energized, it is responsible for returning the solenoid rod and arm to the original position.

The push solenoid chosen was a 1757ES Synchro Start solenoid [17], rated at a maximum travel distance of 1 inches. The push winding is activated with 18 amps of current, and the hold winding with 0.4 amps of current.

Activation of the solenoid was done using an electrical circuit. The detailed schematic is presented in Appendix F. The circuit was designed so that activation could be started manually with a switch, for test purposes, or by the microprocessor during the automated fiber production. The push and hold windings are simultaneously activated except that the push winding activation is a pulsed signal whereas the hold winding signal is on until the switch is turned off, see Figure 2-8.

Figure 2-8 shows the activation signal which is generated by either a switch or a microprocessor signal. This signal in the input to a transistor which drives the gate of a MOSFET. The MOSFET turns the hold winding on. The activation signal is also an input into a 555 Timer which is configured as a one shot pulse, providing a 250ms pulsed signal to activate the high current push winding. The output of the 555, the push signal shown in Figure 2-8, is connected to the gate input of a

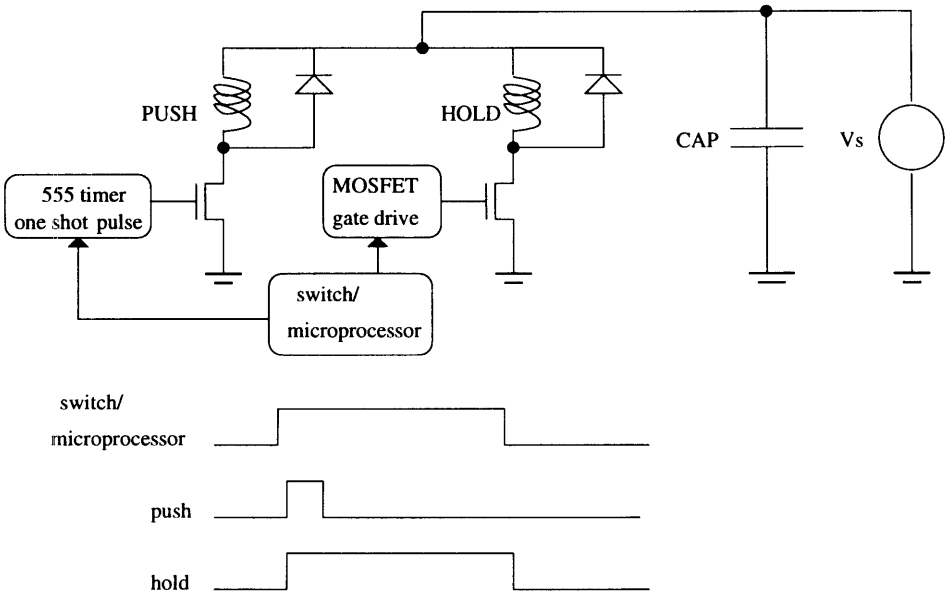

Figure 2-8: Solenoid Workings

MOSFET whose drain input is connected to the push winding. During the pulse, the winding has current flowing through it. The large amount of current needed by the push coil is provided by first allowing a 0.1F capacitor time charge up to Vs which is 20 volts. When the MOSFET turns on, the push coil, which looks like an inductor, is supplied with the necessary current through the capacitor discharge ($I_c = Cd/dt(V_c)$), providing the solenoid with sufficient push force to push the arm forward. The capacitor was chosen by trial and error until the solenoid had the required push force to completely push the arm forward. During the hold interval, the desired current is easily supplied by most power supplies. Protective diodes are placed across the windings to avoid destroying the drive MOSFETs, due to high voltage spikes that could occur in the windings when the current is suddenly switched off.

Even though the solenoid data promised a 1 inch travel, this was experimentally determined to be about 1/4 of an inch less. The constraint this imposes on the glass rods is that they not be more than 3/4 inch into the plate edge.

2.4.2 Arm

Two rods with hooks at the end comprise the arm. Epoxy is placed on the tip of each finger to pick up the fibers. The finger spacings can be adjusted to fit in between two

glass rods.

2.4.3 Fiber Collection

The flow chart shown in Figure 2-9 shows the process steps for fiber collection. The linear carriage is first positioned at the starting point. The drum is positioned so that when the solenoid is activated, the fingers lie in between the rods. The carriage then starts moving and the gel is collected on the fingers. When the arm reaches the end of the drum or the carriage has reached some preset endpoint, the solenoid is deactivated and the arm pulled back. The drum is rotated so that the next line of fibers can be collected. The carriage travels back to its original position. The process above is repeated until the drum is empty.

The accuracy of the position controller is important because the arm might destroy the bars or itself, if it rams into the rods because of poor positioning.

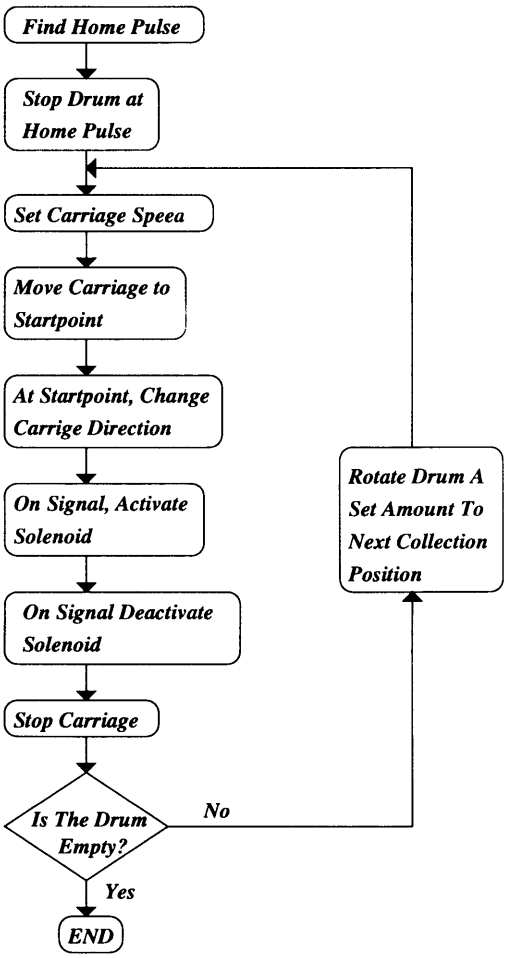

Figure 2-9: Fiber Collection Flowchart

Chapter 3

Digital Hardware Implementation

The sequencing of the various tasks is most easily performed with a programmable controller, such as a microcontroller. The following sections concentrate on describing the chosen microcontroller and its relevant features, along with a general description of the peripheral devices used to implement control of the machine. The peripheral circuits are described in more detail in the thesis sections which discusses machine component under their control. The sections will be indicated accordingly for further referral. The peripheral circuit schematics are included in the appendices.

3.1 Intel 80C196KC Microcontroller

An Intel 80C196KC microcontroller was chosen to supervise the various tasks as well as to implement the digital control loop. It is well suited for the high-speed calculations needed in the digital control implementation, and for the fast input/output (I/O) operations needed to control various parts of the machine.

The 80C196KC microcontroller has a 16-bit Central Processing Unit (CPU) that connects to both an interrupt controller and a memory controller via a 16-bit CPU bus[10]. Figure 3-1 shows the relevant features of the microcontroller. The 16-bit bus also connects the CPU to internal peripheral modules, including the timer, A/D converter, High Speed I/O and parallel Port 1 (IOPORT1) modules. The 80C196KC contains 512 bytes of internal RAM, 64 Kbytes of addressable memory space and operates at 16 MHz. It uses 256 bytes of general purpose register RAM to execute code. For example, the Register Arithmetic-Logic Unit (RALU) uses these registers to perform calculations. The registers replace an accumulator which could be used with the RALU. The code execution is much faster because the data does not flow through a single accumulator, but uses 256 accumulators instead. Register operations additionally control many other peripheral devices.

Figure 3-1 also shows components available on an evaluation board (EV80C196KC)

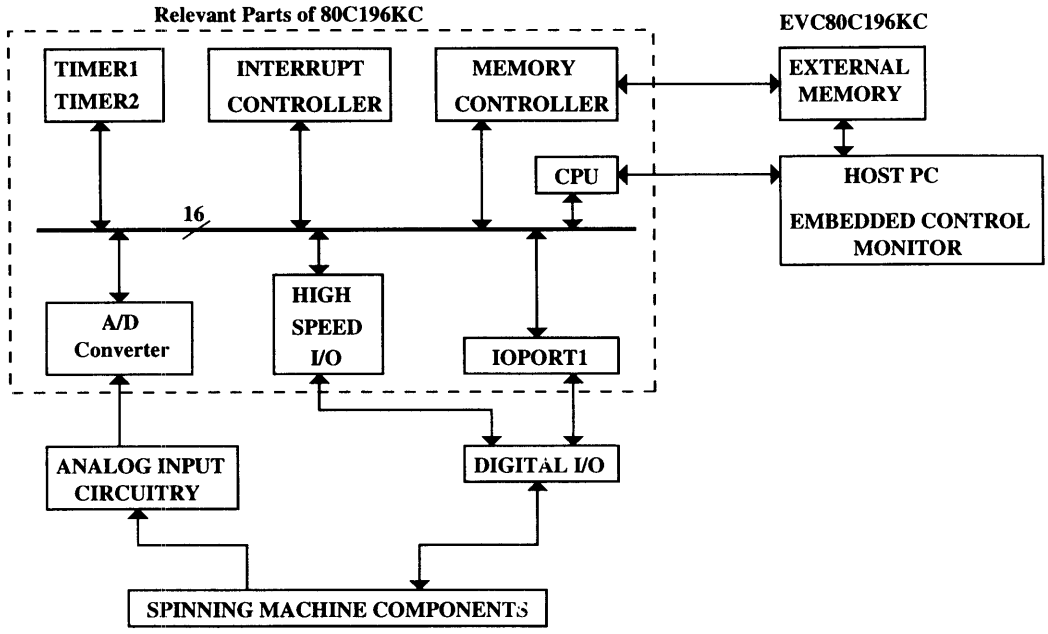

Figure 3-1: 80C196KC Important Features

[11] made available by Intel. It contains a software evaluation monitor for the 80C196KC, enabling control of the board through an Embedded Control Monitor (ECM). The ECM is divided into two independent programs, one of these executes in the EV80C196KC (iRISM96KB software), and the other executes in an IBM PC compatible clone (iECM-96) [11]. The Reduced Instruction Set Monitor (RISM) or the embedded monitor, runs on the external memory or the 80C196KC while the ECM runs on the PC. The PC and evaluation board systems are connected via an RS232 cable which provides an asynchronous interface. The downloading of program code into the 80C196KC's external memory, as well as the monitoring and control of the processor state during code execution, can be controlled from the PC. For example, during the running of the code, the ECM can be used to check and modify code variables, a powerful aid in the debugging process. A detailed discussion on the 80C196KC and EV80C196KC can be found in [10] and [11].

3.2 Digital Hardware Tasks

The various tasks that the microprocessor and peripheral circuitry have to perform are shown in Figure 3-2. The rounded blocks represent circuits or physical parts and the rectangular blocks functions performed. Inputs preceded by circles represent latch

enable inputs. This section gives a broad overview of the associated circuitry used to control and operate the spinning machine.

3.2.1 Spinning

Figure 2-2 illustrated the flowchart of a typical spin process steps. The digital hardware used for each block is discussed in this section. The first step is to start the drum spinning, followed by a pause while it reaches the desired speed setting. Digital PI or analog PI control could be used in the drum speed control as both can be initiated by the 80C196KC. If analog PI control is used, then the only function the microprocessor has is starting and stopping the analog PI control block. The microprocessor is more involved with speed control when digital control is chosen. This section focuses on the digital control implementation.

Referring to Figure 3-2, the rotating drum is labeled with the number 1. It is preceded by a DAC which outputs the control output used to command the drum speed. The control issues are discussed in Chapter 3. The DAC takes a 16-bit value, which is loaded from IOPORT1 in two successive bytes. The 8 lines of Port 1 act like the system bus. The 8 lines are shared with five latches including the two used for the speed command. Each destination of Port 1 is preceded by an 8-bit tri-state latch, which when enabled, loaded in data on the positive transition of the clock input. A PAL uses three High Speed Output (HSO) lines to determine which latch to enable at any given time. The speed command's 16-bit data loading is accomplished as follows:

- Port 1 (IOPORT1) is loaded with the low byte data.
- The HSO line 0 (HSO.0) is set.
- IOPORT1 is loaded with the high byte data after a slight delay.
- The HSO.0 line is cleared.

During this time, HSO.1 is kept low so that the PAL loads the DAC and not the stepper board's low and high byte data (discussed shortly). The speed command's 16-bit data has to be presented to the D/A at the same time. The low byte latch is followed by a second latch. This second latch is clocked with the same signal as the high byte data, so that the low byte from the second latch and the high byte, are presented to the D/A at the same time.

The digital command is output to the DAC at regular sample intervals. For reasons which will be explained in Chapter 4 the sample interval was set at $500\mu s$. At the sample instant, the control algorithm uses sampled inputs to compute the command output. This process involves the 80C196KC's A/D converter, which samples

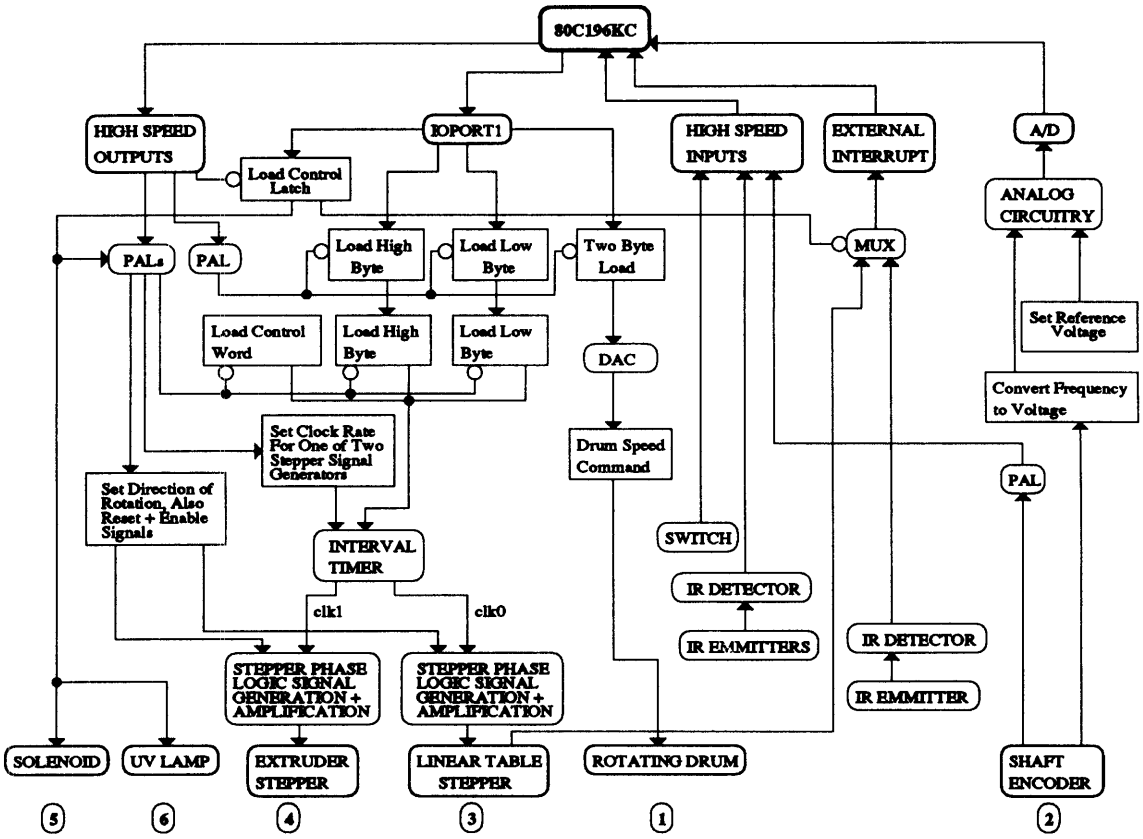

Figure 3-2: Microprocessor and Interface Tasks

the inputs, and the interrupt controller which is responsible for the execution of the control algorithm and the outputting of the command at the set times.

The microcontroller's interrupt handling scheme synchronizes the digital control loop. A software timer interrupt can be set up to interrupt the microprocessor at the sampling instants. It uses the HSO module and the timer module to accomplish this. The 80C196KC has two 16-bit timers, Timer1 and Timer2. Timer1 is a free-running timer which is incremented once every eight state times, that is, every microsecond. Timer1 was selected to be the time base for the HSO module. In setting up the software interrupt, the 80C196KC needs to know what to do and when to do it. An HSO_COMMAND register is first loaded with the action to be taken. In speed control, the action is interrupting the microprocessor to start the control algorithm. The HSO_TIME register is then loaded with the time to take this action. This is simply the current Timer1 value plus the sample interval, $500\mu s$. The HSO module then continually compares the HSO_TIME value with the current Timer1 value, generating the software timer interrupt request when they are equal. When the microprocessor can be interrupted, control is transferred to the software timer interrupt service routine. In the interrupt handling routine, the current sampled inputs are used to compute the control output which is then output in two bytes. Control can then be transferred back to the program code that was interrupted.

The sampling of the inputs, the reference voltage and the measured speed, is initiated in the software timer interrupt handler. When A/D conversion is complete, an interrupt is requested. When control is transferred to the A/D conversion interrupt routine, the channel which completed the conversion is determined and either the digitized values of the analog reference voltage value, or the measured voltage is updated. The A/D converter can be used as an 8-bit or 10-bit converter. To increase resolution and decrease quantization, 10-bit A/D conversion was chosen.

The shaft encoder has outputs which are used in drum speed control. The implementation is discussed further in Section 7.1.5. In summary, use is made of a 300 pulse per revolution shaft output to convert the frequency of this signal to a proportional voltage reading. This voltage value, ω_m, is used in both analog and digital PI control. In digital PI control, ω_m passes through an analog circuit which performs appropriate scaling before A/D conversion. A reference command voltage also passes through the analog circuitry before A/D conversion. The converted digital values are used in the control algorithm implemented in the 80C196KC.

Once the drum has reached its steady state speed, the linear table stepper, labeled 3 in Figure 3-2, moves the carriage is moved toward the start or reference position. The starting position is located at one end of the lead-screw. The setting of the stepper speed, which will be discussed for the carriage stepper, generally applies to the setting of the extruder stepper speed.

The extruder and stepper motors need a number of control signals to activate them

properly. In unipolar stepper operation, discussed in Section 2.2.2, essentially four bits are written in a specific, repetitive sequence to cause stepper shaft rotation in a given direction. The Stepper Phase Logic Signal Generation and Amplification block, shown in Figure 3-2, is responsible for generating the four signals. The block takes four inputs, a clock signal which sets the frequency at which the four bit sequence is repeated, a direction signal used to set the direction of stepper shaft rotation, an enable signal allowing the block to output the four bits, and a reset signal initially pulsed to start proper sequencing. The direction, reset and enable signals are generated via Programmable Array Logic (PAL) chips which are in turn controlled by certain HSO pins as well as bits 0-2 of the CONTROL LATCH.

The CONTROL LATCH is an 8-bit latch, external to the 80C196KC, whose individual bits perform a specific control functions. It is referred to as LCONTROL in the microprocessor code. The implementation was motivated by the need to provide more output pins for machine control. The functions are summarized in Table 3.1.

Table 3.1: CONTROL LATCH

7	6	5	4	3	2	1	0
X	Value Disable	Analog Speed Start	UV Start	Solenoid Start	Stepper1 Direction	Stepper0 Direction	Stepper Enable

LCONTROL.0 enables the stepper rate generation, LCONTROL.1, sets the direction of the linear carriage (Stepper0), LCONTROL.2 does the same for the extruder (Stepper1), LCONTROL.3 activates the solenoid, LCONTROL.4 starts the UV lamp, LCONTROL.5 starts the analog PI speed control loop, LCONTROL.6 controls an input to a PAL controlling the stepper board's clock generation, and LCONTROL.7 is currently unused, but can be used if a further output pin is needed. The CONTROL LATCH is cleared on powerup, powerdown and can also be cleared with a reset pushbutton.

The CONTROL LATCH, loaded from IOPORT1, is latched by toggling the HSO.2 line. Since it is necessary to be able to clear the individual bits to prevent components such as the solenoid from turning on especially after powerup, the latch used was different from the others used in the circuits which could not be cleared. Two 4-bit counters (LS163's) were used as the latches. The count function was disabled, the load input performed the input latching, and the clear pin was used to clear the output values.

Bits 0-2 of the CONTROL LATCH control the direction and enabling of the stepper motors. The remaining input to the Stepper Phase Logic Signal Generation and Amplification block is a clock signal that sets the sequencing rate. The clock

signal is generated by an Intel 8254 Interval Timer Chip. This chip can set up to three independent output signals. The outputs can be configured to provide the required clock signals. The 8254 has control inputs which are provided by PALs controlled by the HSO module. The 8254 uses three internal 8-bit registers, connected by an 8-bit bus, to set up the clock output and its frequency. The details of how this was implemented is given in Section 4.3. One of the registers, called the Control Word, sets up the mode in which the 8254 is operating. Since this is always fixed in our implementation, the external latch providing the Control Word input is hard wired. The two other registers, High and Low byte registers are loaded from IOPORT1 through intermediate 8-bit latches. To load these intermediate latches, the HSO.1 line is kept high when HSO.0 line is toggled. This is different for the speed command DAC loading where the HSO.1 line is kept low. A PAL uses the HSO.0 and HSO.1 lines to determine which latches to send the 16-bit data to. To avoid bus contention in the 8254's 8-bit bus, either the Control Word, High or Low byte drive the bus at any given time. A PAL determines which of the three output latch is enabled to drive the bus.

Once the carriage stepper speed is set and it is enabled in the right direction, it moves towards the reference position, or startpoint, which is on the end of the rail closest to the stepper motor. A switch on that end is opened when the carriage makes contact with it. This limit switch is used to signal that the starting position has been reached. The signal is sent to an external interrupt pin in Port2 (P2.2). The external interrupt pin input is multiplexed because there are more inputs than the 80C196KC can handle. The other multiplexed input is the home pulse from the IR detector used in position control. The latter will discussed in Section 3.2.3. When the interrupt occurs the Port 2 value, IOPORT2, is read to check the value of the limit switch. If the switch is off, then the stepper direction is reversed and the signal to start pre-gel solution spraying is awaited.

Section 2.2.4 discusses the IR emitter-detectors used to signal the start of spraying. This IR detector, different from that used in home pulse generation, is an input to a High Speed Input (HSI). The 80C196KC's high speed inputs can be used to monitor various external events and can generate an interrupt when these events occur. The interrupts can be generated on the positive edge, the negative edge, both edges or on every eighth positive edge transition of the HSI input. One of the four HSI pins, pin HSI.3 signals that the IR detector on the carriage has crossed the path of an emitter. When an event is detected, the HSI status register holds information about the pin that generated the interrupt, and the Timer1 value when that interrupt occurred. These are loaded into an 8x16 holding register. The interrupt can then be generated every time the register is loaded, or whenever the register overflows. The register loading interrupt option was chosen so that the interrupt would occur as soon as an event took place. Once it has been determined that the IR detector generated the

HSI interrupt by crossing the first emitter, the extruder is activated so that pre-gel solution spraying can begin.

Next in the spin process, is the setting of the extruder stepper signals including the clock signal which sets the stepper speed. The procedure is similar to that used to set up the carriage stepper motor. The PAL which generates the stepper control signals, uses HSO.3 to determine which stepper motor's clock rate is being set.

Once the stepper speed has been set, its value can be changed. Whenever the stepper low and high byte latch signals are toggled, a signal labeled value, is pulsed by the PAL which routes the 16-bit data from IOPORT1. This value signal is automatically pulsed after the high byte stepper data latch signal is pulsed. Changing a stepper speed then simply involves loading in a new 16-bit value into the stepper latches. Another PAL responds to the value signal and signals the 8254 accordingly. The value signal must be suppressed when the count is for a stepper different from the one previously set. Otherwise, due to the way the PALs have been programmed, the previous stepper will take on the value being set for the new stepper. Setting bit 6 of the CONTROL LATCH suppresses the value signal so that it is not toggled whenever a new 16-bit count is loaded into the stepper clock generation circuitry.

The IR detector provides subsequent signals to either stop the extruder and carriage stepper, or change the extruder and/or carriage speeds. The last of the four emitters can signal the end of the spraying step. The extruder and carriage stepper board clock generation is disabled effectively stopping the motors. The drum is then stopped by the microprocessor. The drum can also be manually stopped at any time during the run by turning on a switch which is connected to the HSI.2 input. The steppers can also be manually stopped by a switch input which disables the stepper signal generation block. These switches are debugging and safety features.

Timer2 could be used to keep track of the solution in the syringe so that it does not get finished during a run. However, when the syringe currently being used is fully loaded, it can produce many drum loads of gel fibers. Hence, as long as the syringe is kept relatively full, then the use of Timer2 will be unnecessary. Also, due the problem of fixing a reference point, using Timer2 to monitor whether the solution is finished does not make sense unless there is the possibility of always finishing one syringe load in a run. If it was likely to used up the solution in every run, then the syringe plunger could begin at the same point every time and software would precompute the time to finish the solution given the stepper frequency and the plunger stroke distance, i.e, the distance the plunger travels to empty the syringe. A better solution might be to place a mechanical switch next to the block holding the syringe. The switch can be activated whenever the plunger completed its stroke. As has been stated though, so long as the syringe can be kept relatively filled, the solution should not be used up during one run and these precautionary measures are unnecessary.

3.2.2 Cross-linking

The cross-linking step uses the 5th bit of the CONTROL LATCH to turn on the UV lamp. The UV lamp circuit was discussed in Section 2.3.

Position control is used to turn the drum every 10 or so minutes ensuring uniform cross-linking. Since position control is central to the fiber bundling step, its implementation is discussed in Section 3.2.3.

Timer2 can be used to set up the 10 minute intervals and the overall cross-linking time interval. Timer2 is a 16-bit timer. It can be an up/down counter and can use an internal or external source to set its period. The other features of Timer2 are, it can be cleared either externally or in software, it can cause interrupts when it overflows on a either the 7FFFH/8000H boundary or the FFFFH/0000H boundary, it can count in fast or slow modes. In setting up Timer2 for the long time intervals needed, the best mode of operation is using it in slow mode, FFFFH/0000H overflow interrupt, up count and internal clocking. A count variable in software counts overflows until approximately the right amount of time has expired. Then either the drum is rotated or the UV lamp is turned off.

3.2.3 Bundling

A sample flowchart for the bundling step was presented in Figure 2-9 and is used here to explain the essential features of the digital hardware used in the fiber bundling step. Position control is discussed in greater detail in Section 4.2.

The first step is to position the drum at the home pulse. The shaft encoder is the transducer which measures the drum's angular position from some reference which we call the home pulse. An IR emitter-detector pair provide the home pulse. The detector output is multiplexed with the stepper limit switch because home pulse positioning and stepper movements do not occur simultaneously. The shaft encoder can also provide a reference pulse which is not used for reasons explained in Section 4.2. A 600 pulse/revolution signal, which we call multiple, is used to provide the angular position measurement. The 300 pulse/revolution encoder output is an input to a PAL which produces a signal at doubles the frequency thus doubling the angular resolution from 300 to 600 pulses per revolution. The PAL output is the multiple signal. The multiple signal is an input to HSI.0 and during position control, an interrupt due to HSI.0 either increments or decrements a software count variable used in position control. The direction the drum is spinning determines whether the count variable is incremented or decremented. The PAL and other circuitry uses a second 300 pulse/revolution signal from the encoder along with the first one to indicate a change in drum direction and to determine the direction of rotation (Section 4.2.1). The direction value (0 clockwise, 1 anticlockwise, when looking into drum from motor end) is an input to HSI.1. Every edge change of the HSI.1 input generates an interrupt

request, which when serviced, samples the input pin value and updates the direction value.

The home pulse search involves some open loop and closed loop proportional control (Section 4.2.2). As in speed control, the digital command to spin the drum is output every $500\mu s$. A software timer interrupt implements this control feature, using the HSO module in much the same way as described for digital PI control.

Once the home pulse has been found, the carriage is moved to the same startpoint defined in the spinning step. The setting of the speed and the changing of direction when the limit switch is hit, is the same as described in polymer-gel solution spinning step.

When the signal to activate the solenoid is detected (the IR detector), the CONTROL LATCH's 4th pin is set. This is the input to the solenoid circuit described in Section 2.4.1. On maybe the 4th IR detection, the solenoid is deactivated. The CONTROL LATCH is also used to disable the stepper motors at this time. If the drum is not empty, the drum is then rotated a set amount using open and closed loop control similar to that used in finding the home pulse. However, if the drum is empty, the bundling process is complete.

3.2.4 Interrupt Priorities

All pending interrupt requests are prioritized and the interrupt routine which is first serviced is that with the highest priority. Of the interrupt routines used, A/D conversion had the lowest priority, followed by HSI Data Available, HSI.0 pin, Software Timer, and the highest, the External Interrupt. Care must be taken that the interrupts with the highest priorities do not interrupt often, as they will always be serviced at the expense of lower priority interrupts. If this cannot be avoided then high priority interrupt routines should be made as short as possible.

3.3 Limitations

If there were more ports to work with, then the extra circuitry that is necessary for two 8-bit loads could be avoided. Unfortunately, ports other than Port 1 have alternate features that are useful and this eliminates their use as outputs. For example, P2.2 is used as an external interrupt, and Port 0 as the A/D channel. Port 3 and 4 could be used as output ports, but unfortunately, on the EV80C196KC, they are used for memory access, and cannot be used for output without off-board latches and decoding. The extra decoding circuits would be more complicated than the latching circuitry being used currently.

Since there are many components to control, many individual control pins are necessary. That was the motivation behind setting up the LCONTROL register. A

microcontroller with more I/O pins would reduce the amount of external circuitry required to effectively provide more I/O pins, for example, by getting rid of the multiplexer and the CONTROL LATCH.

Lastly, the microcontroller cannot handle floating point operations, which limits the accuracy of the digital PI control loop.

Chapter 4

Modeling and Control

A significant effort in this project involved the control and sequencing of various machine components and operating parameters, including drum speed, drum position, stepper speeds, and solenoid and UV lamp activation. In controlling some components such as the drum, a good physical model of the relevant parts was required. This chapter covers the hardware details and develops appropriate mathematical models where necessary. The control design process for each part is then discussed.

4.1 Drum Speed Control

To spin gel fibers of constant diameter, it is important to keep the drum rotating at a constant speed. A plant model was developed and simulations run to confirm empirical observations and to help explain any problems which arose in assembly and testing. A model of the plant to be controlled is first developed, followed by the estimation of its relevant parameters and finally, the controller employed is discussed.

4.1.1 Plant Model

The model in Figure 4-1 is used to describe the model of the current amplifier motor and load used in the servo-mechanism [5]. Included in the figure is the open loop block diagram.

The current amplifier or power amplifier, outputs a current proportional to its input voltage,

$$Im = K_a Vb \tag{4.1}$$

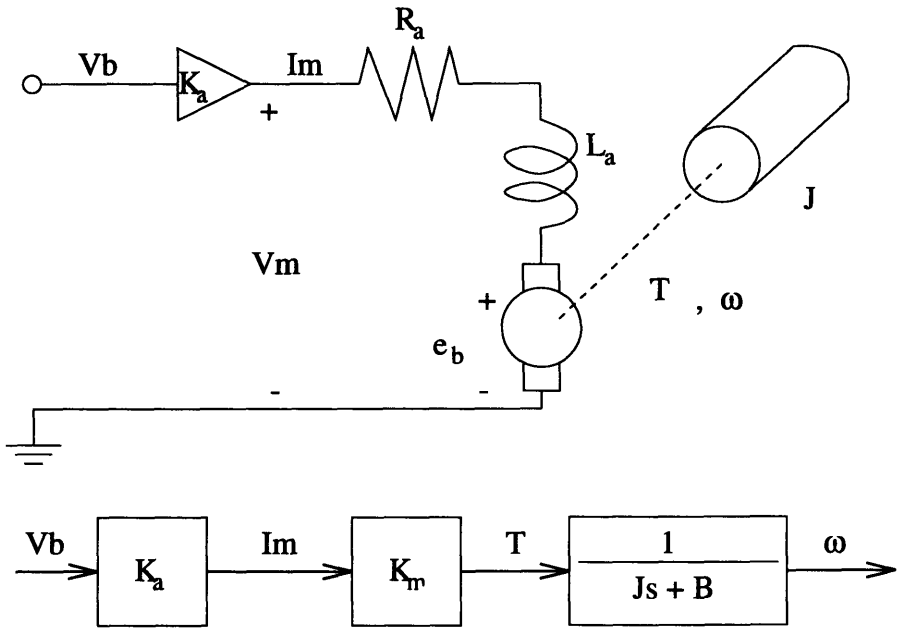

Figure 4-1: Machine Setup.

The constant K_a is the amplifier gain in units of $[amp/volt]$. An important assumption in the development of the plant model is that the voltage to current transfer function has no dynamics, so that the current into the motor is assumed to be instantly set by the drive voltage, Vb. The actual circuit used is shown in Figure 4-7 and explained in Section 4.1.6.

The motor torque is proportional to the current through it. The proportionality constant, K_m, is in units of $[Nm/amp]$.

$$T_m = K_m Im \tag{4.2}$$

The motor torque turns the load which includes its shaft, the drum, shaft, bearings, couplings and shaft encoder. The total load inertia is shown as J. The applied torque has to overcome an opposing torque due to friction. Friction is present in the bearings and couplings for example. The opposing torque, is proportional to the angular speed, with the proportionality constant B. The torque applied to the load, say T_a is given by,

$$T_a = J\dot{\omega} \tag{4.3}$$

that is, the applied torque acts on the load inertia J $[kg - m^2]$ to produce the angular acceleration given by $\dot{\omega}$. The torque applied to the load is thus the motor torque reduced by the frictional torque.

$$J\dot{\omega} + B\omega = T_m \tag{4.4}$$

where $B\omega$ $[Nm - sec/rad]$ is the frictional torque opposing the motor torque.

The motor model shows the term e_b which is the back emf of the motor. This is the voltage which opposes the motor motion and is proportional to it through the back emf constant, $K_e[volt - sec/rad]$. The back emf is given by,

$$e_b = K_e\omega \tag{4.5}$$

When a voltage drive is employed, that is, without the current amplifier, the command voltage, Vb and e_b set I_m. There are dynamics involved due to the presence of the armature inductor, L_a and the armature resistor, R_a. Thus the motor current is builds up with an L_a/R_a time constant. Depending on how fast the mechanical load responds, i.e., the mechanical time constant, the L_a/R_a electrical time constant can be ignored and the motor current assumed to equal to $(V_m - e_b)/R_a$ instantaneously.

In the current drive employed, we could ignore L_a, $R_a and$ e_b in the open loop block picture.

4.1.2 Parameter Estimation

In designing a controller for the plant, it will be necessary to determine the parameter which appear in the open loop block diagram shown in Figure 4-1. These are K_a, K_m, J, B and $\tau_f = J/B$ the open loop mechanical constant. In closed loop control, some measure of the speed of the drum, will be compared with a reference command. The difference between command and measured speed, the error signal, will be used to set the control command to reduce the error signal. The voltage proportional to speed, ω_m, is related to the actual speed through K_T $[volts - sec/rad]$,

$$\omega_m = K_T\omega \tag{4.6}$$

The speed constant also needs to be determined before designing the controller.

Some of the motor parameters can be roughly estimated from the manufacturer specifications [14]. Some manufacturer specifications are given in Table 4.1, for the motor without an attached gearhead where the no load current and friction values are at the maximum un-geared speed. The maximum speed, maximum current and the friction torque are also specified at the nominal voltage of 24 volts. The permanent dc motor used was geared down by a 14:1 ratio gearhead, satisfying the low speed-high torque requirements of the drum [14].

Table 4.1: Motor Specifications

Armature resistance +/- 12%	21.0 (ohms)
Armature inductance	2.5 (milliHenries)
No Load speed +/- 12%	5000 (RPM)
No load current +/- 50%	25 (mA)
Friction torque at no load speed	0.17 (oz-in)
Back emf constant	4.7 (mV/RPM)
Torque constant	6.35 (oz-in/amp)
Armature inertia	$2X10^{-4}(oz-in-sec^2)$

The gearing down changes K_m and K_e increasing them by a factor of 14, the gear ratio. Thus the expected K_m is 0.628 $[Nm/amp]$ or $[volt-sec/rad]$. The expected K_T is 0.477 $[volt-sec/rad]$, (explained in Section 4.1.5), and the expected K_a is 0.0128 $[amps/volt]$, (explained in Section 4.1.6).

The detailed results of the following experiments are tabulated in Appendix D.

Measuring voltage to current gain: The motor drive current is set by the voltage Vb. In measuring K_a, a potentiometer was used to set the reference Vb and the current through the motor was measured by inserting an ammeter in the path of this current. The ratio of current to voltage varied probably due to measurement errors. The average K_a result was 0.01172 [amps/volt], which was close to the expected value obtained from the measured circuit components (Section 4.1.6).

Measuring motor torque constant: K_m was measured by running the motor open loop using a voltage drive. A voltage was applied directly across the motor electrical leads and the corresponding motor current and the drum rotational speed measured. The equation used to determine K_m is,

$$K_m = \frac{V_m - I_m R_a}{\omega} \tag{4.7}$$

Equation 4.7 measures the back emf constant, K_e. For the purposes of analysis, the permanent dc motor can be compared to the separately excited motor [20], [18]. In a separately excited dc motor K_e equals K_m as [18]

$$T_m\omega = e_b I_m \tag{4.8}$$

Equation 4.8 states that the mechanical power equals the electrical power in the motor. This leads to the fact that $K_e = K_m$.

Another methods of estimating K_m would involve using the relation $I_m(K_m/J) = d/dt(\omega)$ where there are two uknowns, I_m, J) and only one equation. If either the load inertia, or the motor-gearbox inertia were known then we could take two sets of measurements, I_m and ω with and without the load and solve simultaneously. However, the load inertias are unknown and the best estimate of K_m will be the result of using Equation 4.7. The value of K_m using linear least square error (LLSE) estimation is 0.61 [volt-sec/rad], where the nominal value for Ra (21 ohms) was used. The value obtained is close to the expected value of 0.628.

Measuring speed constant: This involves measuring the voltage produced by the tachometer circuit as a function of the drum speed. The latter is measured by relating the frequency reading out of the shaft encoder to the shaft rotation speed (see Section 4.1.5). LLSE techniques are used resulting in the tachometer constant estimate of 0.479 [volt-sec/rad], which is close to the expected value of 0.477 (Section 4.1.5).

Measuring mechanical time constant: The motor was run in an open loop, voltage drive mode, and the 10-90% rise time (t_r) of the measured voltage (ω_m) recorded. The mechanical time constant is approximately equal to $t_r/2.2$ for a first order system [5]. The averaged value for τ_f was 0.175s. Attempts to measure J and B by various methods yielded differing results, though J was always of the order 10^{-3}. On the other hand, the value for τ_f was always around 0.2s. J was estimated by measuring the 10-90% rise time of ω_m with a known gain in proportional feedback control. The closed loop mechanical time constant is given by,

$$\frac{1}{\tau'} = \frac{B + GK_aK_mK_T}{J} = \frac{1}{\tau_f} + \frac{GK_aK_mK_T}{J} \tag{4.9}$$

See Appendix E.1 for detailed derivations. For a gain G = 16.1, $\tau_f = 0.175$, and $\tau' = 127.5\text{ms}/2.2$, the value of J was calculated to be $8.072X10^{-3}$ [kg-m^2], leading to B = $4.61X10^{-2}$ [Nm-sec/rad].

4.1.3 PI Control

Two types of control were implemented. An analog controller was designed as an initial test of the machine concept. The final machine design employed digital control for flexibility versatility, as well as ease of sequencing. The two implementations are discussed next.

Analog Control

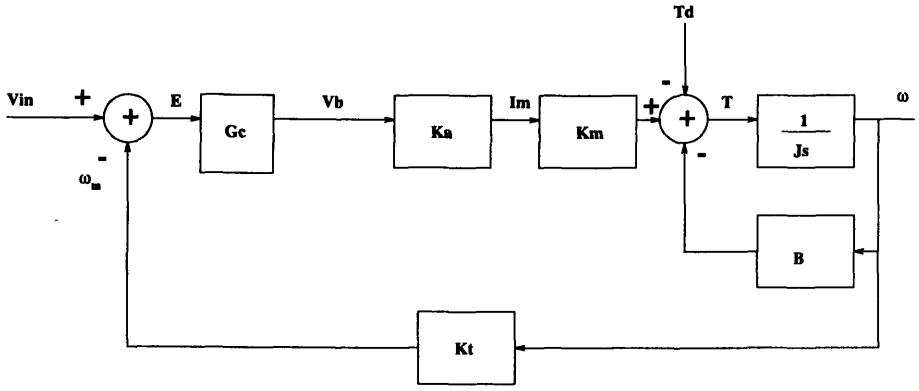

Figure 4-2: Motor Model with Compensator.

The closed loop block diagram is shown in Figure 4-2. Vin is the command input, Gc(s) the compensator and Td is a step disturbance input at the output. The major classical compensator schemes, proportional (P), proportional plus integral (PI), lead and PID, were investigated and implemented for comparison. In the end, P and PI control were chosen as the control methods and subsequently analyzed. The transfer functions relating rotational speed (ω) to the command input (Vin) for P and PI control are,

$$\frac{\omega}{V_{in}} = \frac{(GK_aK_m/J)}{s + \frac{B+GK_aK_mK_T}{J}} \tag{4.10}$$

$$\frac{\omega}{V_{in}} = \frac{GK_{am}}{\tau J} \cdot \frac{(\tau s + 1)}{s^2 + \frac{B+GK_aK_m}{J} s + \frac{GK_aK_mK_T}{\tau J}} \tag{4.11}$$

Appendix E.1 shows the detailed derivations for P and PI analog control. In Equations 4.10 and 4.11, G/τ is the integral gain and G the proportional gain. These two compensation schemes were examined because they provide desirable regulation with minimal computation overhead. The main reason why PI control was ultimately chosen over P control was that PI control can provide zero steady state error to step inputs, which is useful in that the speed is related to the command input, in steady state, by the tachometer constant, which can be estimated quite accurately. Appendix E.1 details other equations relevant in P and PI analog control.

The high valued friction coefficient, B, moves the open loop mechanical pole to the left of the origin (it would be at zero if B = 0). This means that in the root locus for PI control, the closed loop poles do not move into the right-half plane causing stability at certain gains. The value of B helps in implementing a more stable closed loop response. In general though, if there is a pole at the origin, PI control should not be used. In this case, PID might be better for improved stability [4].

In setting the gains and the pole locations, the specifications used were that there be little or no overshoot. Overshoot and ringing are undesirable to reduce the wear and tear of the rotating drum. Placing the poles on the negative real axis will accomplish this. A fast rise time was not necessary as the actions following the start of the drum rotation could be delayed until the drum reached steady state. Thus the poles do not have to be far left of the origin. However for purposes of fast disturbance rejection, it is better to design for poles as far left in the complex plane as is reasonable.

Digital Control

A digital controller was desired in the final implementation for several reasons. Parameter changes can be easily made in software, and the digital controller can be less sensitive to drift and component value variations. Finally, adaptive control can eventually be implemented. Adaptive control is important as some model parameters inevitably change over time. The adaptive controller will change the control parameters to keep the performance at the optimal point. One anticipated parameter change is in the load inertia when polymer solution is dumped onto the drum, or when a new drum is used.

In modeling the plant in the digital domain, a zero-order-hold (ZOH) equivalent model is used [7]. The equivalent plant model block diagram with feedback is shown in Figure 4-3.

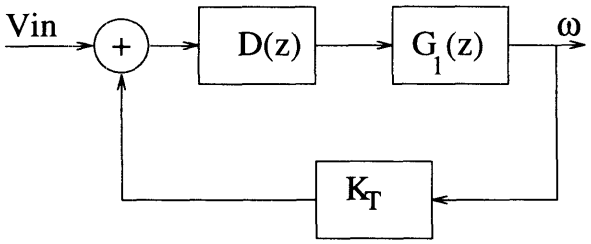

Figure 4-3: Digital Block Diagram.

The function D(z) represents the compensator and $G_1(z)$ the ZOH plant equivalent of the plant model given as[7]

$$G_1(z) = \frac{K_{am}}{B}(1 - e^{-\frac{B}{J}T_s})\frac{z^{-1}}{1 - e^{-\frac{B}{J}T_s}z^{-1}} \tag{4.12}$$

The compensator by

$$D(z) = h_p + \frac{h_i'\Delta T_s}{z-1} \qquad h_i = h_i'\Delta T_s \tag{4.13}$$

Ts represents the sampling interval. The open loop pole is at e^{-T_s/τ_f}.

For the closed loop system with digital PI compensation. the equation relating rotational speed to command input is given by,

$$\frac{\omega}{V_{in}} = \frac{A[h_p z + (h_i - h_p)]}{z^2 + [AK_T h_p - 1 - E]z + E + AK_T(h_i - h_p)} \tag{4.14}$$

where

$$A = \frac{K_{am}}{B}(1 - \exp^{-T_s/\tau_f}) \tag{4.15}$$

and

$$E = \exp^{-T_s/\tau_f} \tag{4.16}$$

The variable *hp* is the proportional gain and *hi* is the integral gain. The theory and equations are developed in more detail in Appendix E.2.

The poles are placed by choosing values for hi, *hp* and Ts. The sample frequency (1/Ts) as a rule is chosen to be at least 5 times the closed loop bandwidth [7]. The sample period was set at 500μs for reasons explained in Section 4.1.4. Using that and a initial rough estimate of the values of integral and proportional gains gave a low overshoot response, these values were tuned in Matlab (see Appendix B) until the closed loop poles resulted in fast responses without overshoot, i.e., poles not imaginary. Matlab simulations were used to verify the pole location calculations. A Matlab function, which determined the pole locations given the plant and compensator parameters, was written because during the course of the project, some parameters changed often, for example K_a, the voltage to current constant changed as different current drives were tried.

Matlab simulations using the estimated parameter values in Section 4.1.2, resulted in coincident poles occur at an *hp* of 22 and *hi* of 0.067. The simulations also show that a decrease in *hp* results in imaginary poles. The experimental starting point would then be to start with the values which result in coincident poles and tune the gains according to the experimental observations. For example, if ringing is observed then increasing *hp* while keeping *hi* constant might move the poles onto the real axis. As in analog control, poles on the real axis are desired to prevent overshoot.

4.1.4 Microprocessor Implementation Issues

The digital control implementation block diagram is shown in Figure 4-4. The command voltage is divided by two to get it in the 80C196KC A/D range of 0-5volts. The same is true for the measured voltage. The measured voltage range is 0-10 volts, and the reference voltage, Vin, is similarly chosen. After A/D conversion, the input data is used to update the command output at sample instants. This command is output as a 16-bit digital value which passes through a D/A. The output of the D/A is Vb. This voltage drives the plant. The drum speed is fed back through the encoder and speed measurement circuitry.

The PI block expanded is shown in Figure 4-5. The error signal, *verr* is fed into an accumulator block and added to previous accumulated values. The variable *cout* is the algorithm output corresponding to adding the a measure of the current error term and previously accumulated value. The algorithm output is subtracted from 32767 before output to the DAC. This is because of the DAC output is effectively "inverted". An input range of 65535 through 32767 to 0 corresponds to an output voltage of -10 through 0 to 10 [3]. The DAC transfer function is

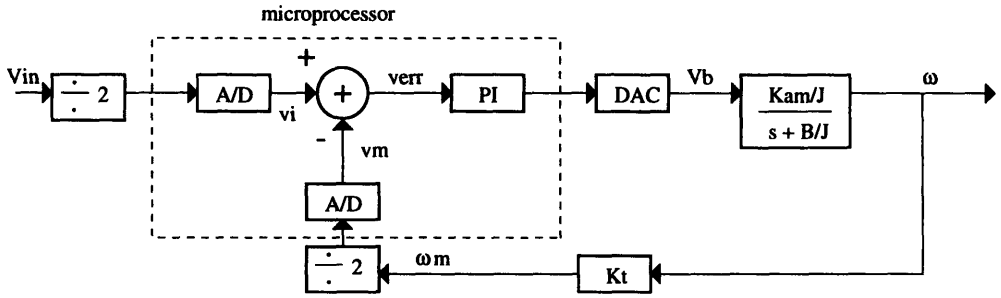

Figure 4-4: Digital Implementation Block Diagram

$$DAC\ output = (32767 - input) * DAC_gain \tag{4.17}$$

By first subtracting cout from 32767, the DAC output will be equal to $cout * DAC_{gain}$, where the DAC gain from "counts" to voltage, is given by $10/2^{15}$.

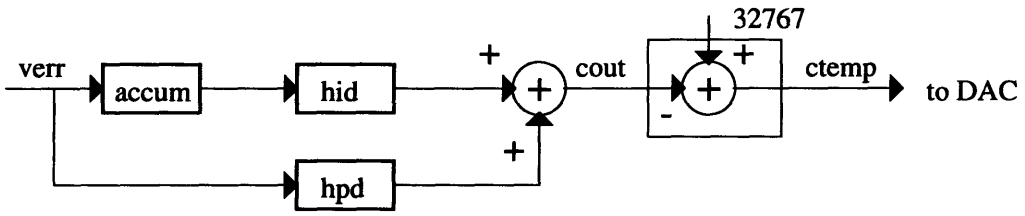

Figure 4-5: PI block

Initial values for the digital gains used were $hp = 6.7$, $hi = 0.067$ and $Ts = 1ms$. The main problem seemed to be excessive noise which was reduced by increasing the sampling rate. The controller might be over-correcting, causing large variations in the drum speed. Faster sampling can reduce the wide swings by "catching" the speed before it overshoots. This leads to lower amplitude noise in the measured speed. The limit for lowering Ts depends on the computation time and on quantization noise. The sample period should be at least as long the time to perform the PI algorithm task and should in fact be longer than because of other non-digital control tasks to be performed in the software timer routine. The PI algorithm time was measured at $50\mu s$ when the software timer interrupt routine handled only speed control. Allowing

for future code expansion, Ts was then set at $500\mu s$. The PI gains were then set at hp = 22, $hi = .067$, after some trial and error. These analytically gave pole locations at $z = 0.9963 \pm .0001j$ and a dominant pole time constant of 0.1332 seconds.

The PI gains in the implementation code are different from the values in simulation, because the A/D and DAC gains are not factored into the calculations. The A/D changes voltages to counts, and its gain is in units of counts/volt, the DAC being the opposite has units of volts/count. The gains are given by

$$F_{AD} = \frac{1023}{V_{refAD}}, \quad F_{DAC} = \frac{V_{refDAC}}{32768} \tag{4.18}$$

The variables F_{AD} and F_{DAC} are the A/D and DAC gains respectively and V_{refAD} = 4.93 volts for the A/D and $V_{refDAC} = 10$ volts for the DAC used. There is also the factor of 1/2 which alters the PI simulation gains. The digital implementation gains, hpd and hid, are related to the original gains by

$$h_{pd} = \frac{2}{F_{AD}F_{DAC}}h_p, \quad h_{id} = \frac{2}{F_{AD}F_{DAC}}h_i \tag{4.19}$$

This is because the gains the implementation adds to the calculations should be accounted for. The gain factor, $2/F_{AD}F_{DAC}$ is about 30. Simulation gains are then multiplied by 30 to set the gains used in the code. The detailed derivation is shown in Appendix E.2.

In the implementation code, an accumulator keeps an update of the the error voltage through verr. The output cout is then

$$cout[n] = h_{id} * acc[n] + h_{pd} * verr[n] \tag{4.20}$$

where

$$acc[n] = verr[n-1] + acc[n-1] \tag{4.21}$$

The forward Euler algorithm was used to compute the successive accumulator values. cout is output after calculating the error signal as well as performing saturation checks on the acc and cout variables. The conditional branch statements inherent in these checks can vary the time between successive outputs. The alternative would be

to calculate the control output and output it at the beginning of the software timer routine. The cost would be an additional pole at z = 1 due to the sample delay. Since for our operation maximum possible the time to output the control command is $50\mu s$, while the minimum cannot be more than $10\mu s$ less, and the sample period $500\mu s$, the variation in command output intervals is not so critical.

The accumulator and cout values are set to maximize at 32000/hid and 32000. cout is limited to avoid saturating problems and the accumulator value is limited to avoid antiwindup which can occur when the drum is first started and the voltage error large. The value 32000 is less than 32767, the maximum value the DAC can handle in its positive range.

4.1.5 Speed Measurement

The measurement of the rotational speed was done using a shaft encoder which has a square wave voltage output whose frequency is a measure of the drum speed. This variable frequency is converted to a variable voltage which is proportional to ω through the tach constant. The early schemes used to measure the speed, such as using another motor in reverse (as a generator), involved having to low pass filter the noisy tach voltage. A frequency-to-voltage converter chip was tried but these use an integrator in determining the voltage output. The low pass filter pole is undesirable because it limits the bandwidth of the speed controller. With P control it will result in oscillatory response if the gain is increased, as well as limiting the settling time above a certain gain. In the case of PI control, the response can be unstable. Figure 4-6 shows a continuous-time example where the tach's low pass filter pole is set at 0.5, the mechanical pole is at 5 and the PI zero at 10.

The DAC mentioned is a different DAC to that used in commanding the motor speed. Here it is used to measure the shaft speed. The shaft encoder is a good choice because it can also be used for position control. Our shaft encoder outputs 300 pulses per revolution. In measuring the speed, one method involves counting the number of pulses in a given time interval, but this time interval would need to be large to accurately detect the velocity and it would indeed end up being another low pass filter tach because of the averaging scheme. The algorithm which was implemented is as follows:

- Count the number of clock pulses within encoder pulses. The clock frequency is much faster than the encoder maximum frequency.
- The count value which is inversely proportional to the speed is the input to EPROMs which act as a lookup table.
- The EPROMs invert the value of their input, with some gain.

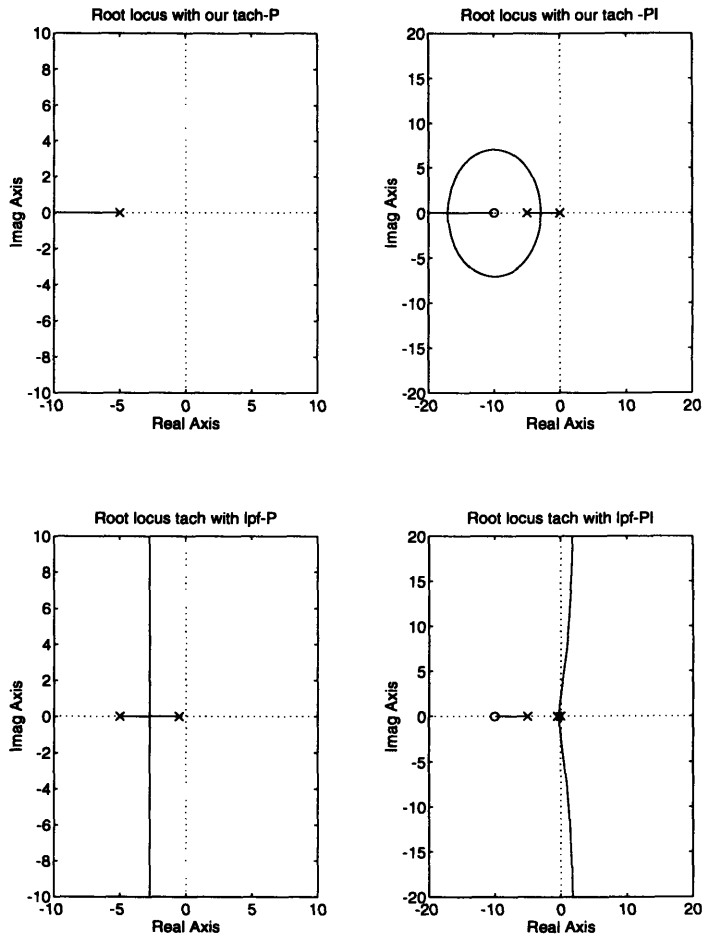

Figure 4-6: Root locus example for different tachometers.

- The inverted output is the input to a DAC.
- The DAC output is the measured rotational speed (volts).

The measured output delay is the propagation delays through the various digital parts. Since these are usually on the order of tens of nanoseconds, this is an improvement over the averaging scheme initially suggested which would need several multiple pulses to produce a speed measurement. This speed measurement is different from those used conventionally, and was used mainly because the sum of parts was cheaper than buying a tachometer with a clean output, and for reasons explained earlier. It should be noted that the DAC input could be as data input to an digital controller which uses digital data anyway. However this would involve using the 80C196KC port pins of which only Port 1 was available and already in heavy use. Also, since there is an analog PI controller as well, the DAC is needed.

The drum operation speed was to range from 20 RPM to 200 RPM, the maximum possible being 357 RPM (the ungeared maximum rated speed of 5000 divided by the gear ratio 14). The DAC (Burr Brown 1600) [3] voltage range is -10 to 10 volts, but we would only be using the positive range since our reference would consist of only positive voltages. 0-10 volts in the DAC maps to 32767- 0 at its input. In summary then, the mapping from encoder frequency to DAC input was 100Hz-1kHz, corresponding to 20-200RPM mapping to 32767-0 counts. It is simpler to think of the DAC input range as 0-32767 and subtract this value from 32767 at the end. So the 100Hz-1kHz can be thought of as mapping to 0-32767.

The encoder pulses are first synchronized with the clock input. The synchronization circuit is basically an R-S latch. The output (ENC) is a pulse output which goes low whenever the encoder input makes a negative transition. The synchronized pulse lasts a clock period long.

In between ENC arrivals, a 15-bit counter counts the clock pulses. The ENC acts as the CLEAR input to the four 4-bit counters. Before the count is cleared and counting restarted, the current count is latched into 27C256 (8x15) EPROMS, The EPROMs have 15 address lines and 8 output lines. Two EPROMs (low and high-byte) are used to output the 16-bit data to the DAC through a second set of latches. Each address location in an EPROM has an output corresponding to:

$$output = 32767 - B/address \tag{4.22}$$

32767 is used to adjust to proper DAC range and B is the gain adjustment which ensures that the full positive DAC range is used. The address being the output of a 15-bit counter, cannot exceed $(2^{15} - 1)$, 32767. Also, in determining the value for

B, the maximum DAC voltage (9.993935 volts) should correspond to the maximum frequency to be measured (1kHz), or to the minimum count. The value of the count, or address is given by,

$$count = \frac{frequency\ of\ clock,\ fclk}{frequency\ of\ encoder,\ fin} \tag{4.23}$$

The DAC transfer equation, can be written as

$$V_{DAC} = (32767 - output)\frac{10}{32768} \tag{4.24}$$

$$V_{DAC} = (32767 - (32767 - B/count))305\mu V \tag{4.25}$$

$$V_{DAC} = \left[\frac{(B)(305\mu V)}{f_{clk}}\right] f_{in} = \left[\frac{(B)(305\mu V)}{2\pi f_{clk}}\right]\omega \tag{4.26}$$

where f_{in} is the frequency of the encoder output and f_{clk} the clock used to increment the 4-bit counters. The tachometer constant is given by,

$$K_T = \frac{B(305\mu V)}{2\pi f_{clk}} \tag{4.27}$$

It should be noted from Equation 4.23 that if we want the maximum count 32767 to correspond to the minimum frequency, 100Hz (effectively considered to be 0Hz), then a clock rate of 3.2767MHz is desired. However, the closest we could obtain was a 3.6864MHz clock. Also, note from Equations 4.25 and 4.26 that at $f_{in} = 0$, the output voltage is <u>not</u> zero because the value for the count is a maximum at 32767, not ∞. Thus for values of f_{in} less than the minimum, the value of V_{DAC} is pegged at to $V_{DAC_{min}}$. This is a consequence of the implementation used and not a desired result. Lastly in calculating B, using the whole positive range of the DAC, and Equation 4.26 with f_{in} at the maximum yields a B of about $1.2X10^8$. This results in a minimum measured voltage of 1.12 volts, and a tach constant of 0.477 [volt-sec/rad].

The implementation circuit is detailed in Appendix F. The EPROM code is given

in Appendix C.7. PALs were used to generate the latch signals and various signals to control the counter operation. The PAL code is given in Appendix C.2.

The initial offset of ω_m is not a problem in the control of the plant, but care must be taken in analog PI control where it is not possible to easily limit the voltage used to set the motor speed to non-negative values. Consequently, when analog PI is used and the initial reference command is zero, the error will be negative and the integrator will cause the amplifiers to go into negative saturation. The voltage used to set the motor current will be saturated and control will be impossible. Thus the analog PI loop's output voltage which is used to command the motor current should be pegged at zero if it tries to go negative. A diode is used to peg the command voltage so that it cannot go below zero. This is shown in the circuit schematic in Appendix F.

4.1.6 Speed Command

The speed command voltage is an input into a LM675 power amp in a non-inverting configuration as shown in Figure 4-7. The 7.8Ω resistor sets the motor current. Using the OP27 operational amplifier in the configuration shown allows us to reduce the value of the resistor in series with the motor, thus reducing power dissipation as well as keeping to a minimum, the voltage drop across the resistor.

The LM675 is chosen for its current delivering capabilities and because it will enable the implementation of position control by allowing the drum to spin in both directions.

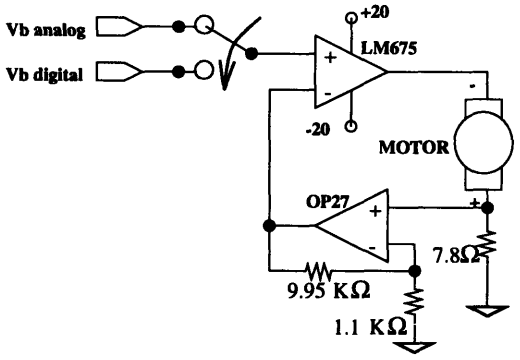

Figure 4-7: Speed Command.

There is a switch which determines whether the drum is being driven by the analog PI or the digital PI. The analog PI loop can be chosen for debugging purposes. Each of these control methods have a potentiometer to set the command reference voltage. The current in the motor is set by Vb and the 7.8Ω resistor. The current-to-voltage ratio, or K_a, is 1/10*7.8Ω, or 0.0128 [amps/volt].

4.2 Drum Position Control

Position control is essential for collecting and bundling the gel fibers. The drum must be accurately positioned so that the arm can collect successive rows of fiber. The shaft encoder used in speed control was also used in position control. This Section discusses the details of the position controller.

4.2.1 Position Control

There are different ways of measuring the position of the drum. For example, a potentiometer is often used as a position sensor. We chose to use the shaft encoder because of its existence in the system. The Datametrics encoder outputs a home pulse and 300 pulses in one revolution. There are two multiple pulsed outputs that are in quadrature, i.e., 90° out of phase with each other. The encoder also supplies all three outputs in the inverted form.

The home pulse can be used to provide a reference from which the subsequent multiple readings can be used to give an angular position measurement. We constructed our own homing pulse because the encoder home pulse will probably not coincide with the reference we desire. We want the reference to lie in line with one of the glass rods. Since the drum is initially inserted without reference to the encoder home pulse position, and since this adjustment might entail complicated mechanical adjustments, we decided to instead make our own homing pulse. This home pulse consists of a optical emitter-detector pair that lies across the bottom of one of the end plates. When a drum is constructed, it is fitted with a thin black tab which is lined up with one of the rods. This tab then breaks the emitter-detector connection when it passes across it. The break is converted to an electrical signal that becomes the homing pulse.

Two encoder quadrature outputs, labeled A and B can be used to determine the drum rotation direction. If normal is defined as the clockwise direction when looking into the drum from the shaft motor end, then in the normal direction, A lags B by 90° and in the opposite direction A leads B by 90°. This information is used by a PAL to determine when the direction changes and the current direction value (see Appendix C.1). Logic 0 corresponds to the normal direction and 1 the opposite. Figure 4-8 shows the scheme used.

The arrows labeled "check" signify when the PAL begins to check for the next positive transition from either $\overline{A}$ or B. If from $\overline{A}$, a flip-flop is reset otherwise it is set. The flip-flop output is an input to the microprocessor and is used to interrupt it whenever the direction has changed, at which point the microprocessor reads in the new direction value.

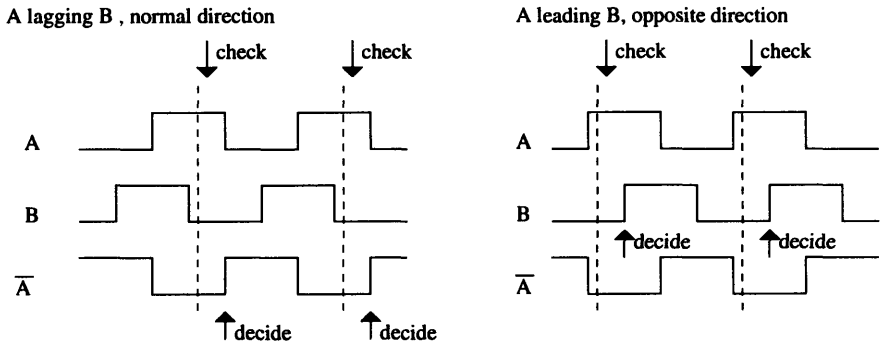

Figure 4-8: Timing Waveforms used in Direction Detection.

4.2.2 Position Control Algorithm

The control algorithm is described as follows:

- Find the home pulse.
- Stop the drum and reset the count (n) which measures the angular position, to zero.
- Wait for a signal to start rotation.
- Every $500\mu s$, compute the error signal measuring the difference between the desired position and the current position.
- When the error (> 5) is large, bang-bang control is used.
- Proportional control is used when the error is small, 5 multiple pulses away from the desired position.
- Direction changes are noted and the voltage command output adjusted accordingly, i.e., positive voltages for normal direction and negative if the drum is to be rotated in the opposite direction.
- After some time, the drum should have settled to the desired position, at which point, a signal to go to the next position is awaited.

Bang-bang control is used initially because of the large command that is needed to overcome friction. It is important that direction changes are noted so that the count variable n is updated in the right direction. The position count is incremented in the normal direction and decremented in the opposite. There are 300 multiple

pulses per revolution, so the angular position resolution is 1.2° per count. To increase the position resolution, n is incremented at every transition in the multiple pulse input, leading to a resolution of 0.6°. This resolution is an important parameter when calculating the clearing distance of the rods from the collecting arm's fingers.

4.3 Stepper Motor Speed Control

The stepper motors will be operated at low speeds. Thus the motors are relatively unlikely to skip steps. The circuit used produce the logic signals is the same for both the extruder and the carriage. The detailed circuit is shown in Appendix F. The stepper signals are generated using a circuit taken from [8] which detailed a simple way to drive a unipolar stepper motor.

The four phase signals are generated by this circuit. The peripheral circuits are the stepper drive circuits and the clock rate generator. The stepper driver in both cases uses the LM18293 4-phase Push-Pull Driver. It is capable of delivering up to 1 amp of current per channel. The carriage needs more current than this. The LM18293 driving four power MOSFETs which are powered by a 10 amp power supply is used to drive the carriage's stepper motor.

The rate generator has at its heart an Intel 8254 Timer Interval chip [9]. This chip can be configured to function as a baud rate generator, an event counter, a complex waveform generator, a programmable rate generator, among other things. It can independently control three outputs, all operating in different modes. Two outputs were used to provide the clocks for the two steppers. The 8254 has three main registers, the Low byte register, the High byte register and the Control Word register. The Control Word can be used to determine operational mode of the 8254. The Low and High bytes (count value) are used to determine the frequency of the output.

The 8254 was operated in Mode 2 which outputs a square wave whose frequency is the clock reference divided by the count value. At the beginning of the count, the output is high, the 8254 then counts down to half the count if it is even and $(count - 1)/2$ if the count is odd. The clock signal then goes low for the remainder of the count. The count register is then reloaded, the output goes high and the above process repeats. If a new count is available, it is loaded at the end of the existing countdown. A gating signal can be used to disable a clock output. I used a PAL to control the gate signals. When a stepper enable signal is activated, the gating signal for the chosen clock is activated. The PAL also ensures that when a different clock is chosen next, the other one is not deactivated. So once a gate signal is activated it remains active until enable goes low. As a safety concern, the enable signal, which is usually set and cleared by the microprocessor, can be disabled through a switch input. The Control Word has two signals, A0 and A1, which set the mode of operation, it

also has a variable which determines whether count loads will be low byte only or low then high byte . A write signal can be used to load in the count and Control Word values. The 8254 read signal is disabled as read operations will not be used.

In addition to the clock rate, the stepper rate generator is used to generate the enable, reset and direction signals used in the driver circuit. These signals are generated in the sequence shown in Figure 4-9. The line to be enabled is the LM18293's enable lines. The reset line clears the flip-flop and starts the phase signal generation. The direction signal sets the direction of rotation, anticlockwise being 0 and clockwise 1 as seen looking into the stepper shafts.

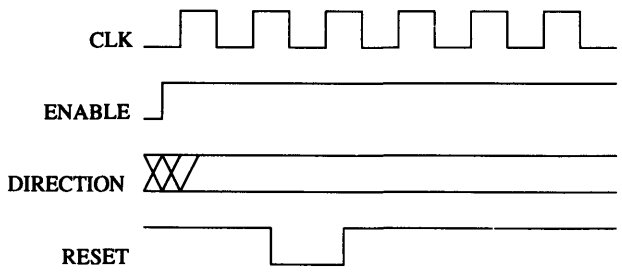

Figure 4-9: Driver Board Stepper Signals.

The basic stepper driver circuit was breadboarded. An opto-isolator isolates the logic inputs. These were included in anticipation of the use of the boards for purposes where isolation was important. For our machine, isolation is not necessary.

The extruder works well with speeds under a 100Hz and the same is true of the carriage stepper. The carriage stepper is mainly limited by the amount of current it has available, but with a 10 amp power supply, this should not be a problem.

4.4 Other Control Features

The remaining components in the machine required sequencing or on-off commands. Examples being the solenoid and the UV lamp.

Control and sequencing circuitry has been or will be wirewrapped and placed in a rack. The machine user is then equipped with a front panel covering the rack which allows for control and monitoring of many machine parts. The digital and analog speed, a digital speed stop and solenoid and UV-lamp manual start/stop switch along with LED's showing the the status of the control lines, are installed to make machine operation and trouble-shooting easier. The measured drum speed is also output for trouble shooting purposes.

Chapter 5

Testing/Results

This chapter discusses the main test results. The speed control measurements are presented and compared with those obtained from simulations. The position control results and the stepper motor are also discussed. There is a brief discussion on the testing of the functional pieces of the machine, the solution spinning, cross-linking and bundling. Finally, the overall coded is discussed. The details of the microprocessor code can be found in Appendix A.

5.1 Test of System Parts

The first parts tested were those used in spinning the polymer gel. This includes testing the speed control of the drum and making sure both stepper motors worked.

5.1.1 Drum Speed Control

An analog PI compensator was designed with a proportional gain of 10 and a break-point at s = -6.85, corresponding to $\tau = 0.146$. The step responses differ from the expected, which were supposed to have no overshoot due to pole placement on the negative real axis. These discrepancies were the result of parameter inaccuracies. The overshoot is 50%, corresponding to a damping ratio, ζ, of 0.2 [5] if we considered the response to be dominated by a complex pair of poles. The peak rise time is 0.62 seconds, which leads to a natural frequency, ω_n, of 5.19 $[rad/s^{-1}]$. If the system response was dominated by a complex pair of poles, then the settling time would be about $4.6/\zeta\omega_n$ or 4.11 seconds, however, the step response settles in about 0.75 seconds which is much faster, but slower than the expected time of 0.6 seconds (see Appendix E.1). Some explanations include an error in the value of one or more of the parameters. The response looks like the result of a zero, close to the dominant pole pair, influencing the step response. A zero close left of the complex pair can cause

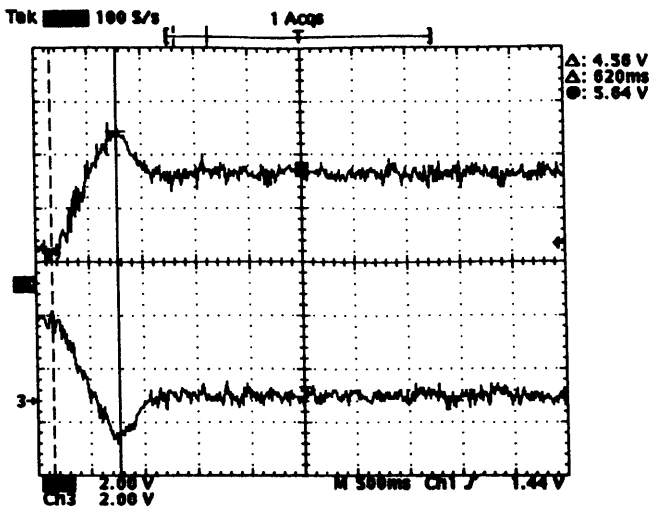

Figure 5-1: Analog PI speed and error response to step input.

the kind of overshoot shown in Figure 5-1.

Another point of note from Figure 5-1 is the high frequency noise observable on the tach voltage. This noise should not be a problem in the closed loop control because it is much faster than the closed loop bandwidth, that is, the drum cannot respond to the fast noise.

Digital control was next implemented. It will be used in the polymer spinning machine. Figure 5-2 shows three step response plots for different proportional gains. The first has a digital gain of 660, or hp of about 22, the second, hp of about 20, and the third, hp of 13.33. In all cases, the integral gain was held at 0.067. The rise speed increases with decrease in hp. The overshoot which might be present, for lower hp, could be getting clipped by the accumulator which avoids windup. Wind up can cause actuator saturation and cause the system to stop responding to closed loop control. However, closer examination of the step response for hp of 13.33 shows a very slight overshoot. It is expected that decreasing hp further will result in overshoot, that is, a response dominated by closed loop imaginary poles. The simulations (Appendix B) predicted a 0.4 second settling time for $hpd = 22$, $hid = 2$, however, the plotted response shows a settling time of 1.32 seconds, which is over three times as slow. The simulations were performed with the same parameters used to predict the analog PI response. The analog PI response did not agree with simulations either. The error can probably be traced to inaccurate model parameter estimation, especially, J, whose value was difficult to measure. Even though the step response for the gains did not match reality, it was adequate for our needs so the gains were not changed.

One last thing to note is the voltage of the tachometer output at zero speed, is not zero, but is about 1.16 volt, which is about the expected value.

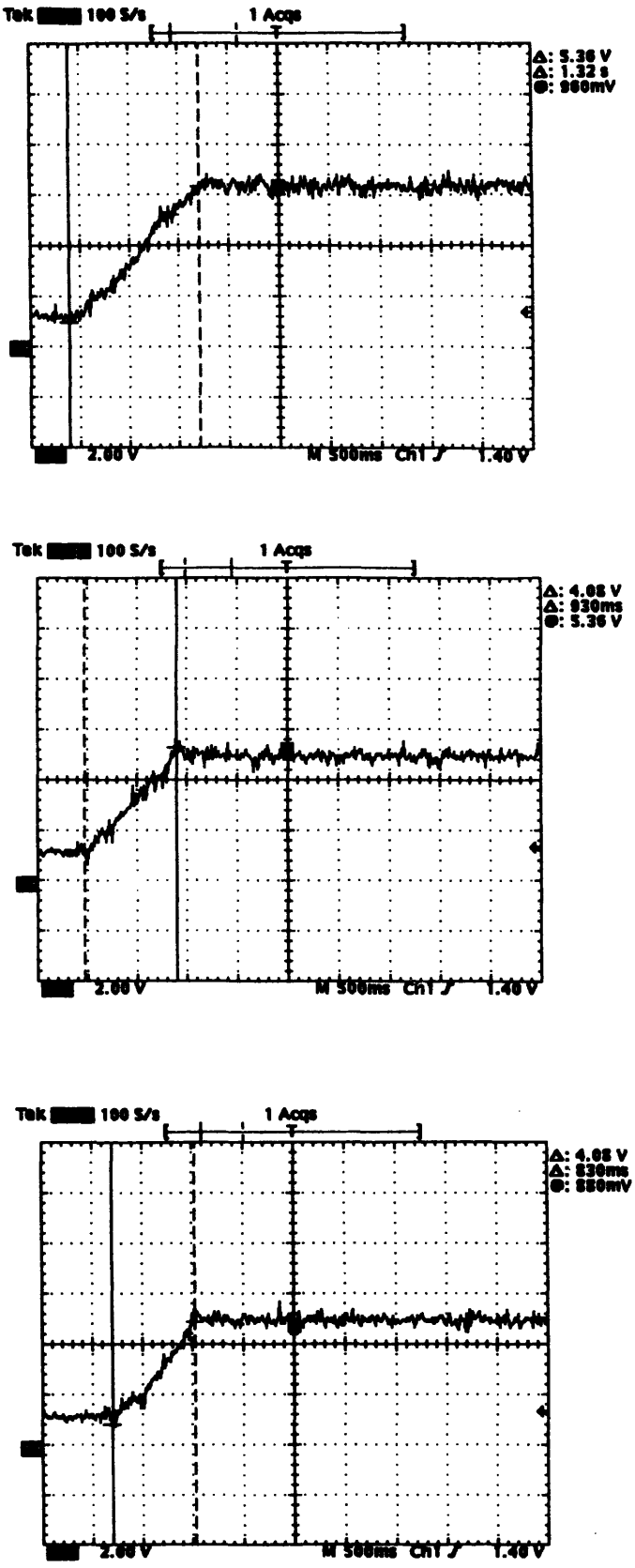

Figure 5-2: Digital PI responses for hpd = 660, 600, 400 and hid = 2.

5.1.2 Drum Position Control

The algorithm for position control was discussed in Chapter 4. The implemented code is part of Appendix A. A proportional gain of 4000 was chosen. This gain is low enough to avoid overshoot, which can cause the drum to go unstable. The gain is also high enough to be able to quickly respond to positional errors. If the gain is made too low, then the control output is too weak to drive the drum, which then results in a residual error.

The position control was tested by simulating the gel collecting process without involving the linear table and solenoid. The drum is first rotated and the home pulse found. The drum then waits at this position until a switch is toggled, then the drum is rotated by a preset angle and waits again for the switch to be toggled. During this time, the ECM can be used to check the value of the variable used to the measure the angular displacement. The drum rotation sequence is repeated a specified amount of times. To test the drum currently being used, the sequence was repeated 8 times as there are 8 rods on the drum circumference. The corresponding rotation angle was 45° or 75 *multiple* pulses per rotation. The position control was found to be accurate to ± 3 *multiple* pulses in moving from one rod position to the next. Since there are 600 *multiple* pulses per drum revolution, the position control is accurate to 1%.

5.1.3 Stepper Motors

The detailed circuit (Appendix F), was constructed and tested with the microprocessor controlling the various parts. The extruder stepper was found to work well for clock frequencies of 100 Hz and less. The clock frequency is set by the 8254 Timer Interval chip. The linear table can operate at clock frequencies upto 450 Hz, after which it start to miss steps. At the extruder maximum clock frequency of 100 Hz the entire syringe is emptied before the linear table, operating at 250 Hz clock frequency, has moved the carriage half way along the drum. So the extruder clock frequency range is satisfactory for our design needs.

5.1.4 UV Lamp and Solenoid

These parts were tested by first testing the control signals into the gates of the driving MOSFETs or 555 without connecting the lamp or solenoid. Then with the lamp or solenoid in place, the devices were turned on. The lamp turns on and off as expected. When the solenoid is activated, the arm is pushed out and held in position until the switch is turned off. The solenoid has a big kick when returning back. If this is a problem, some way of cushioning the return should be used, for example, using some damping action to oppose the action of the solenoid's return spring.

5.2 Test of Functional Parts

Once it was established that the parts of the machine were working well, the parts performing a specific function were tested together. This involved writing code for the 80C196KC which will eventually run the process.

Code was written for the spinning, the cross-linking and finally, the gathering processes. This code is available on the IBM PC and is a helpful tool for debugging.

5.3 Overall Code

The code used to implement the entire sequence is given in Appendix A. It uses the code written for the various stages. Each code is called in turn and the interrupt procedures are tailored to be as general as possible so as not to have to change too much when a different function is called. Various predefined constants at the beginning of code can be used to set such parameters as the IR emitter which starts and stops the solenoid, the time to cross-link, the time between drum rotations during cross-linking, the stepper clock frequencies, and the drum angular rotation during the cross-linking and the gathering stages.

The polymer gel solution is first spun. The rotating drum is started, followed by a pause to let it it reach the desired speed. Then the linear table stepper motor is started. It moves towards the left of the apparatus, i.e., the side closest to the stepper motor. When the carriage hits the limit switch on that end of the rail, it changes direction, moving to the right. When the carriage detector crosses the path of the first IR emitter, the extruder stepper motor is activated and the syringe contents pushed out. When the rail crosses the 4th emitter, The drum motor as well as the linear table and extruder stepper motors are stopped. The next step is the cross-linking step. For test purposes, the cross-linking was set for 3 minutes. During that interval, the UV lamp is turned on. It does not come on unless a manual switch is enabled. The manual switch can be replaced by the interlocking switch, when this part of the machine has been constructed. After UV cross-link begins, the drum is rotated every minute to expose the gels to more light. After the time is up, the UV lamp is turned off. The next step is to bundle the fibers. The following sequence is repeated m times, where m is the number of rods on the drum circumference. First the drum is rotated until the home pulse is found. Then the linear table is moved alongside the drum, and when the 1st IR emitter is found, the solenoid is turned on. The solenoid is turned off on the 4th emitter detection. The drum then repositions itself for collection of the next line of fibers. The linear table then moves toward the limit switch, and the procedure repeated m times.

There was difficulty in making the overall code work with the solenoid in place. The high currents circulating when the solenoid is activated caused ground problems.

Specifically, the ground potential at certain parts of the machine circuitry, such as the 80C196KC ground, increased by as much as 0.5 volts when the solenoid was activated. This lead to the speed command latches being loaded with unwanted values from IOPORT1 and interfering with speed control. A suggested solution is to minimize parasitic inductance the solenoid current loop. This entails moving the solenoid circuitry much closer to the solenoid, which decreases the inductance due to the long wires being used and reduces voltage spikes during solenoid activation. Another solution entails isolating the solenoid circuitry with a separate ground, power supply, or optoisolator.

The code was tested by disconnecting the solenoid and watching the LED signaling solenoid activation instead. The machine executed the steps specified in the sample run described above. Thus, if the grounding problem is resolved, then all the parts are in place for executing a run involving the polymer solution. The next step would then be calibrating the machine setpoints, such as drum speed, with fiber dimensions.

Chapter 6

Conclusions

This chapter summarizes the main concepts and results of the machine.

6.1 Summary

The research goal was to design and build a polymer gel spinning machine. The machine was needed to quickly produce the fibers to be used in characterizing the polymer fibers which will be used to make linear actuators.

The thesis began by reviewing the need for the machine. The importance of the gel fibers was discussed next, laying the ground work for the importance of the machine. Thus, the project aim was defined.

The design of the machine was presented in Chapter 2, which discussed the main functional parts of the machine. The machine was designed to first spray the polymer gel solution onto a rotating drum. The next step was to cross-link the fibers using UV light. The last step was to collect the fibers from the drum by picking up the fibers from between the drum rods. The various mechanical and electrical components that combine to perform the required functions such as spinning were outlined in this chapter. An 80C196KC microprocessor was chosen to control the various stages of production as well as to perform digital speed and position control. Speed control was used to maintain the drum speed during the spinning of the polymer solution into fibers, and position control was used to maintain the drum position when the fibers were picked and bundled. The digital hardware, including the microprocessor, used to implement the various tasks, was discussed in Chapter 3.

The control implemented was then discussed in Chapter 4. A mathematical model was developed for use in speed control of the drum before discussing the choice of analog and digital PI gains needed to satisfy the step response specifications. The most important specification was a low overshoot in the step response. The simulated and empirical step responses were found to differ. The analog PI gains chosen resulted

in an overshoot in the step response, where it had been designed for none. The response speed was also slower, 0.75 seconds instead of 0.6 seconds. The digital PI gains also resulted in a slower step response than expected. However, the digital PI step response had no overshoot, and since it had been determined that a fast settling time was not too important, the 1.32 second settling time was considered adequate. Position control was also discussed in Chapter 4, where the control used was proportional except when the error was larger than a specified value, $EMIN$. When the error was bigger than $EMIN$, bang bang position control was used. The position control was found to be accurate to $\pm 0.5\%$. The stepper motors' speed control was then discussed. With open loop control, the operational speeds had to be low, so that the stepper motors would not begin to skip steps. The linear table starts to skip steps at step rates exceeding 450 Hz, and the extruder above 100 Hz. The extruder maximum speed of 100 Hz was found to be more than adequate for our design needs.

C Code was written for various procedures to test various parts of the machine. The overall code, presented in Appendix A, can be used to run the entire process. Unfortunately, problems in grounding made it impossible to test the code in a real situation using a polymer solution. The code was observed to do the correct thing if the solenoid was disconnected, and an LED used to signal that it would have been turned on.

6.2 Suggestions for Future Work

Much remains to be done to complete the machine so that it will be able to automatically produce gel fibers.

An improved interface from machine to user should be developed. This includes wire-wrapping all the circuitry and placing the wire-wrap boards in a rack mount. Currently less than half of the circuitry remains on protoboards. An additional feature should be calibrating the speed setting potentiometers used in analog and digital speed controls.

The fiber dimensions could not be calibrated to machine setpoints because of the solenoid problem. Once this has been resolved then the machine can undergo several polymer spinning runs and calibration performed. The most important fiber dimension is the diamter. The diameters resulting with different drum speeds should be investigated. In addition, the effect, if any, of the linear table and extruder stepper motor step rates on fiber dimensions should be investigated.

There were no experiments using a polymer solution as the starting point. So there might be some unforeseen problems requiring machine design re-evaluations. One envisioned problem might be the arm kick back which occurs when the solenoid is deactivated. Damping of the arm's return motion has been presented as a possible

solution.

Another question is the sensitivity of the machine's digital PI speed controller to anticipated parameter changes, such as the value of the load inertia when a different drum is used. Adaptive control can be used to decrease the controller's sensitivity, making the machine more robust.

Appendix A

Microprocessor Code

This appendix details the overall code for polymer spinning machine. The code sequences the entire proces.

```
/*********************************************************/
/*  C Code for Overall Process                           */
/*  First polymer gel solution is spun:                  */
/*   - Start motor spinning, after a bit, start linear   */
/*     table towards limit switch. When limit switch     */
/*     hit, start table in opposite direction. When IR   */
/*     detector lines up with 1st IR emitter, start      */
/*     extruder. When detector lines up with the 4th     */
/*     detector, stop both stepper motors, then stop drum. */
/*                                                       */
/*  Second fibers are cross-linked:                      */
/*   - Find home pulse, after a wait, start carriage     */
/*     towards stepper. When limit switch is hit, change */
/*     carriage direction. On 1st IR detect, solenoid out, */
/*     on 4th pull solenoid in. Stop stepper, tell drum to */
/*     proceed to next position, then repeat the process a */
/*     preset number of times.                           */
/*                                                       */
/*  Lastly fibers are gathered:                          */
/*   - Find the home pulse, stop drum at this point. After */
/*     a while, stop position control. Start the UV lamp. */
/*     Continually check if time interval between drum   */
/*     rotations is up. If up, rotate drum, wait a bit,  */
/*     stop position control. In background, continually */
/*     check if cross-linking time is up. If up, stop UV */
/*                                                       */
/*********************************************************/

#pragma model (kc)
```

```
#pragma interrupt (home = 29)
#pragma interrupt (t2_overflow = 28)
#pragma interrupt (software_timer = 5)
#pragma interrupt (multiple = 4)
#pragma interrupt (hsi_done = 2)
#pragma interrupt (analog_done = 1)

#include<80C196.h>
#include<stdlib.h>

register char apple[15];
#pragma locate(apple = 0x30)

/* Predefined constants.... */

#define Ts (int) 500          /* Sample time in microseconds */
#define STEPS  75             /* No. of steps in one travel */
#define REV_STEPS 2           /* No. of travels */
#define REV 600               /* No. of steps in one revolution */
#define SPEED1 (long) 23000   /* Speed in bang-bang position control */
#define SPEED2 (long) 20000   /* Speed on second pass of home pulse */
#define SPEED3 (long) 28000   /* Speed on first search of home pulse */
#define MAXI (int) 2000       /* Time waste variable's maximum count */
#define MAXJ (int) 1000       /* Time waste variable's maximum count */
#define EMIN (int) 5          /* Minimum error for bang-bang control */
#define SMCLK0 200            /* Linear table clock rate Hz - spin */
#define GMCLK0 200            /* Linear table clock rate Hz - gather */
#define MCLK1 50              /* Extruder clock rate in Hz */
#define ESTART 1              /* IR emitter on which arm pushed out */
#define EEND 2                /* IR emitter on which arm pulled in */
#define CLHOURS 0             /* Cross-linking no. of hours */
#define CLMINS 2              /* Cross-linking no. of minutes */
#define ROTMINS 1             /* Time between drum spins (minutes) */

/* Flag variables, showing status of events of interest */

volatile unsigned char check;     /* AD conversion channel checker */
volatile unsigned char check2;    /* HSI input status checker */
volatile unsigned int when;       /* Timer1 value when hsi occurs */
volatile unsigned char check3;    /* P2.2 status checker */
volatile unsigned char flag;      /* manual on switch flag */
volatile unsigned char flag2;     /* flag for ref input below min. */
volatile unsigned char ch_switch; /* switch check flag */
volatile unsigned char stop;      /* ECM can use to stop drum, steppers */
volatile unsigned char wait;      /* variable while waiting for software */
                                  /* timer interrupt to occur */

/* IOPORT1 loading variables */
```

```
volatile long count;              /* stepper count – two bytes */
volatile long cout;               /* command output – two bytes */
volatile long ctemp;              /* temporary two byte variable */
volatile long ctemp1;             /* temporary two byte variable */
volatile unsigned char ct_lo;     /* low byte output */
volatile unsigned char ct_hi;     /* high byte output */
volatile unsigned char lcontrol;  /* Control latch */

/* Digital PI control variables */

volatile unsigned int vi;         /* digital command input */
volatile unsigned int vm;         /* digital velocity measurement */
volatile unsigned int v2;         /* intermediate variable in AD routine */
volatile int int_vi;              /* digital input – integer */
volatile int int_vm;              /* digital velocity – integer */
volatile long verr;               /* current digital error */
volatile long verr_old;           /* past digital error */
volatile long acc;                /* current accumulator value */
volatile long acc_old;            /* past accumulator value */
volatile long hpd;                /* PI proportional digital gain */
volatile long hid;                /* PI integral digital gain */

/* Position control variables */

volatile unsigned char dflag;     /* direction flag */
volatile unsigned char dflag2;    /* software direction flag */
volatile unsigned char homing;    /* set when looking for home pulse */
volatile int m;                   /* count of number of stop–starts */
volatile int n;                   /* step count variable */
volatile int p;                   /* counts number of home pulse crossings */
volatile int gain;                /* proportional control gain */
volatile int dist;                /* one travel distance */
volatile int error;               /* position error */
volatile int errmax;              /* max position error */

/* Stepper, limit switch, and IR detector variables */

volatile unsigned char do_step;   /* stepper control flag */
volatile unsigned char done0;     /* flag for stepper0 loaded */
volatile unsigned char done1;     /* flag for stepper1 loaded */
volatile unsigned char detects;   /* counts no. of IR detects */
volatile unsigned char limit;     /* set when limit switch hit */
volatile unsigned char limiting;  /* set when looking for limit switch */
volatile unsigned char sol_on;    /* solenoid on flag */
volatile unsigned char detstart;  /* detects no. when solenoid activated */
volatile unsigned char detend;    /* detects no. when solenoid de–activated */
volatile long count;              /* stepper count – two bytes */
```

```
volatile long dmin0;            /* Minimum count for stepper0 rate */
volatile long dmin1;            /* Minimum count for stepper1 rate */

/* UV variables */                                                    130

volatile unsigned char clstop;   /* cross-link stop flag */
volatile unsigned char rotate;   /* flag for drum rotation */
volatile long r,s;              /* Timer2 overflow count variables */
volatile long rmax, smax;        /* Count variable maxima */
volatile long ovpermin;          /* No. of Timer2 overflows per min */

/* Process stage variables */

volatile unsigned char spin, cross, gather;                       140

/* Miscellaneous */
unsigned char k;                 /* Time waste variables between..   */
unsigned char a;                 /* loading of hso commannds, for.. */
unsigned char b;                 /* ioport1 writes */

/* AD conversion complete Interrupt Service Routine (ISR) */
/* - check the channel from which conversion completed */
/* - if ACH0, read in 10 bit value into vm, then sample ACH1 */
/* - if ACH1, read in 10 bit value into vi. */                    150

void analog_done(void)
{
      check = ad_result_lo;
      if((check&7) == 0)     /* AD conversion from channel 0 */
            {
            vm = (unsigned int) ad_result_hi;
            vm = vm << 2;
            v2 = (unsigned int) ad_result_lo;
            v2 = v2 >> 6;                                         160
            vm = vm + v2;
            ad_command = 0x09; /* sample channel 1 immediately */

      }
      else if((check&7) == 1) /* AD conversion from channel 1*/
            {
            vi = (unsigned int) ad_result_hi;
            vi = vi << 2;
            v2 = (unsigned int) ad_result_lo;
            v2 = v2 >> 6;                                         170
            vi = vi + v2;
      }

}
```

```
/* HSI hold register loaded ISR. */
/* - check the HSI STATUS register to determine interrupting line */
/* - if HSI.1 read direction value */
/* - if HSI.2 and pin = 1, then stop drum motor flag set */                    180
/* - if HSI.3 and pin = 1, then IR emitter crossed */

void hsi_done(void)
{
 check2 = hsi_status;        /* read status and time info */
 when = hsi_time;            /* to allow new data loading */
 if ((check2 & 4) == 4)       /* HSI.1 event occurred */
   {
    dflag = check2 & 8;  /* read in new direction value */
    dflag = dflag >> 3;                                                    190
   }
 else if ((check2 & 16) == 16)     /* HSI.2 event occurred */
   {                           /* manual stop switch */
    flag = check2 & 32;
    flag = flag >> 5;
   }
 else if((check2 & 64) == 64) /* HSI.3, IR detector */
  if((check2 & 128) == 128) /* detector value =? 1*/
    detects++;
}                                                                          200

/* HSI.0 positive transition ISR */
/* - depending on the drum direction, increment or decrement */
/*   angular position variable                                  */

void multiple(void)
{
 if(dflag == 0)        /* multiple encoder interrupt */
   n++;          /* direction normal, increment */
 else                                                                      210
   n--;          /* direction towards observer, decrement */
}

/* Software timer ISR */
/* - set up interrupt for next sample instant */

/* - IF SPINNING */
/* - compute digital PI control output */
/* - update PI variable, error, accumulator variables */
/* - perform limiting of accumulator and output, where necessary */          220
/* - check if drum should be stopped */
/* - output command */
```

```
/* - check if steppers need to be started */
/* - if so, output count for 8254 */
/* - .. if for stepper0 (linear table), set up Control Latch */
/* - ..... enable stepper, set direction for 0 and 1 as well */
/* - .. if for stepper1 (extruder), set c0 to inform PAL */

/* - IF CROSS-LINKING */
/* - compute position control output */                                    230
/* - perform output value limiting where necessary */
/* - check if drum should be stopped */
/* - output command */

/* - IF GATHERING */
/* - compute position control output */
/* - perform output value limiting where necessary */
/* - check if drum should be stopped */
/* - output command */
/* - check if stepper0 needs to be started */                              240
/* - if so, output count for 8254 */
/* - .. check if solenoid needs to be activated, deactivated */
/* - .. check if time to search for limit switch. */

void software_timer(void)
{
 hso_command = 0x18;       /* setup interrupt on software timer 0 */
                           /* using timer 1. */
 hso_time = timer1 + Ts;   /* event at end of sample time period */
                                                                        250
 if(spin == 1)
  {
   ad_command = 0x08;              /* sample channel 0 immediately */

   /* Using forward algorithm to compute accumulator values */

   int_vi = (int) vi;
   if(int_vi <= 121)     /* if vi below some minimum, drum stops */
        flag2 = 0;                  /* vi <= .58 volts i.e. ..    */
   else flag2 = 1;                /* 1.16/2*1023/4.93  min "vin"  */      260
   int_vm = (int) vm;

   /* Conditions for drum stop, reference input falls below minimum */
   /* (flag2) or software flag (stop) or manual switch on (flag).    */

   if ((flag2 == 0) || (stop == 1) || (flag == 1))
        {
         verr = 0;                 /* reset error */
         acc = 0;                  /* reset accumulator */
        }                                                               270
```

```
    /* Otherwise carry out PI algorithm */

    else {
        verr  = (long) (int_vi - int_vm);
        acc   = verr_old + acc_old;
    }
    cout  = hid*acc + hpd*verr;

    /* Limit values cout, acc can take */                              280

    if(cout >= 32000)
        cout = 32000;    /* Max allowable cout = 32000 */
    if(cout < 0)
        cout = 0;        /* Min allowable cout = 0 */
    if(acc >= 16000)
        acc = 16000;     /* Max allowable acc = 16000 */
    if(acc < 0)
        acc = 0;     /* Min allowable acc = 0 */
                                                                       290
    verr_old = verr;
    acc_old  = acc;
    ctemp    = 32767 - (int) cout;
  }

else if((cross == 1) || (gather == 1))
  {
    if(dist > n)              /* check for overshoot, in which case */
        {                      /* software changes its direction flag */
          error = dist - n;                                            300
          dflag2 = 0;
        }
    else
        {
          error = n - dist;
          dflag2 = 1;
        }

    if((p >= 2) && (error > errmax)) /* try to eliminate excess */
        if(dflag2 == 0)              /* overshoot by going in opp.*/   310
          dflag2 = 1;                /* direction */
        else dflag2 = 0;

    if(p >= 2)                  /* check error size after first */
        {                       /* pass of homing pulse */
          if (error < EMIN)       /* Error small */
            {
              cout = (long) gain*error; /* proportional control */
```

```
            ch_switch = 1;          /* error small flag */
          }                                                          320
        else
          cout = SPEED1;       /* Error large, open loop control */
      }

  if(stop == 1)
      cout = 0;
  if(cout > 32000)
      cout = 32000;           /* Limit mazimum cout value */

  if(dflag2 == 0)                                                    330
      ctemp  = 32767 - cout;    /* Positive DAC output */
  else
      ctemp  = 32767 + cout;    /* Negative DAC output */
}

/* Ouput control for either speed or position control */

ctemp1 = ctemp;
ctemp  = ctemp & 255;
ctemp1 = ctemp1 & 65280;                                             340
ct_lo  = (unsigned char) ctemp;
ct_hi  = (unsigned char) (ctemp1 >> 8);

/* Output to ioport1 in two byte loads */

ioport1 = ct_lo;

hso_command = 0x20;       /* set HSO.0 */
hso_time = timer1 + 2;
                                                                     350
k = a + b;
k = a + b;

ioport1 = ct_hi;

hso_command = 0x00;       /* clear HSO.0 */
hso_time = timer1 + 2;

k = a + b;
                                                                     360
if((spin == 1) || (gather == 1))
 {
  if ((do_step == 1) && ((done0 == 0) || (done1 == 0)))
      {                       /* Stepper enable etc.. */
        hso_command = 0x21;    /* set HSO.1, to chose stepper */
        hso_time = timer1 + 2;  /* latches for ioport1 writes */
```

```
k = a + b;

ctemp    = (int) count;
ctemp1   = ctemp;                                                    370
ctemp    = ctemp & 255;          /* ANDed with 00FF */
ctemp1   = ctemp1 & 65280;       /* ANDed with FF00 */
ct_lo    = (unsigned char) ctemp;
ct_hi    = (unsigned char) (ctemp1 >> 8);

/* Output to ioport1 in two byte loads */

ioport1 = ct_lo;

hso_command = 0x20;      /* set HSO.0 */                             380
hso_time = timer1 + 2;

k = a + b;
k = a + b;

ioport1 = ct_hi;

hso_command = 0x00;      /* clear HSO.0 */
hso_time = timer1 + 2;
k = a + b;                                                           390

hso_command = 0x03;          /* clear HSO.3 (c0 variable) */
hso_time = timer1 + 2;
k = a + b;

if(spin == 1)
 {
  if(done0 == 0)   /* starting on drive, stepper0 */
       {
         hso_command = 0x03;  /* clear HSO.3 (c0 variable) */        400
         hso_time = timer1 + 2;
         k = a + b;

         ioport1 = lcontrol;
         hso_command = 0x22;      /* set HSO.2 */
         hso_time = timer1 + 2;
         k = a + b;
         hso_command = 0x02;      /* clear HSO.2 */
         hso_time = timer1 + 2;
         k = a + b;                                                  410
         done0 = 1;
       }

  if(done1 == 0)  /* starting on extruder, stepper1*/
```

```
        {

          hso_command = 0x23;    /* set HSO.3 (c0 variable) */
          hso_time = timer1 + 2;
          k = a + b;
          done1 = 1;                                                    420

        }
    }

  else if(gather == 1)
    {

      if((sol_on == 1) && (detects >= ESTART) && (detstart == 0))
          {
           lcontrol = 0x09;      /* start solenoid, step dir0=0 */      430
           detstart = 1;
          }
      else if((sol_on == 1) && (detects >= EEND) && (detend == 0))
          {
           lcontrol = 0x00;       /* stop stepper, solenoid */
           sol_on = 0;
           detend = 1;
          }

      if(stop == 1)                                                     440
          lcontrol = 0x00;            /* stop stepper motors */

      /* Load Control Latch */
      ioport1 = lcontrol;
      hso_command = 0x22;      /* set HSO.2 */
      hso_time = timer1 + 2;
      k = a + b;
      hso_command = 0x02;      /* clear HSO.2 */
      hso_time = timer1 + 2;
      k = a + b;                                                        450

      if(limiting == 0)
          { ipend1 = 0;     /* clear home pulse pending ints. */
            limiting = 1;   /* ready for stepper limit switch */
          }

      done0 = 1;          /* stepper variables have been loaded */
    }

  hso_command = 0x01;  /* clear HSO.1 disable stepper loads */           460
  hso_time = timer1 + 2;
}
```

```
  }

 if(wait == 1) /* This variable is set outside this ISR as a way  */
   wait = 0;   /* of checking that this ISR is run. Setting wait  */
               /* to 0 tells outside routine, ISR has taken place */
}

/* Timer2 Overflow ISR */                                              470
/* - increment rotation count varibale and UV stop count variables */
/* - set flags where necessary */

void t2_overflow(void)
{
 r++;
 if(r >= rmax)
   rotate = 1;
 s++;
 if(s >= smax)                                                         480
   clstop = 1;
}

/* Home pulse or limit switch hit ISR */
/* - if searching for home pulse, on first pass reduce speed, */
/*   set position distance to aim for equal to one revolution */
/*   set max error to a little over one revolution value. */
/*   On second pass set max error to the smaller steps */
/* - if looking for stepper limit switch, read ioport2 and check */
/*   the value of the external interrupt pin */                       490

void home(void)
{
 if(homing == 1)
   {
        if(p == 0)
          {
            cout = SPEED2;
            n = 0;
            dist = REV;                                                500
            errmax = REV+5;
          }
        else if(p == 1)
          errmax = STEPS;

        p++;
   }
 else if(limiting == 1)
  {
    check3 = ioport2;          /* Read Port 2 value */                 510
```

```
    if((check3 & 4) == 4)
        limit = 1;          /* Limit switch hit. */
   }
}

main()
{

  /* main driver for code. Spinning, cross-linking and gathering */
  /* are done in turn. */                                                    520

  int i,j;          /* time waste variables */
  char first;

  hpd = 600;          /* = hp*32768*2*VREF_AD/(1023*VREF_DAC) */
  hid = 2;           /* = hi*32768*2*VREF_AD/(1023*VREF_DAC) */
  a = 1;
  b = 1;
  acc_old = 0;
  verr_old = 0;                                                              530
  acc = 0;
  verr = 0;
  ioport1 = 0;
  flag = 0;
  flag2 = 1;
  done0 = 0;
  done1 = 1;
  limiting = 0;
  homing = 0;
  limit = 0;                                                                 540
  detects = 0;
  stop = 0;
  do_step = 0;
  wait = 0;

  dmin0 = 1000000/SMCLK0;          /* min divisor for max clk0 rate */
  dmin1 = 1000000/MCLK1;           /* min divisor for max clk1 rate */

  ioc0 = 0x50;                  /* HSI pins 2,3 enabled as inputs */
  ioc1 = 0x02;                        /* P2.2 provides EXTINT1 */          550
  hsi_mode = 0x70;          /* HSI.2 Event on every edge transition */
                            /* HSI.3 on positive transitions */

  /* Initialize control latch */

  lcontrol = 0x00;
  ioport1 = lcontrol;
  hso_command = 0x22;       /* set HSO.2 */
```

```
hso_time = timer1 + 2;
k = a + b;                                                          560
hso_command = 0x02;      /* clear HSO.2 */
hso_time = timer1 + 2;

/* SPINNING.....*/

spin = 1;
cross = 0;
gather = 0;

imask1 = 0x20;          /* detector inputs interrupt enabled */     570
ipend1 = 0;
int_mask = 0x26; /* Interrupts on software timer, AD and hsi */
int_pending = 0;
hso_command = 0x18;         /* set up first sampling instant */
hso_time = timer1 + Ts;
ad_command = 0x08;           /* sample channel 0 immediately */
enable();                       /* enable interrupts */

for(i=0;i<MAXI;i++)          /* wait a little for drum to get */
  { for(j=0;j<MAXJ;j++);};  /* up to speed. */                    580

count = dmin0;                 /* value for clk0 16-bit data */
lcontrol = 0x03;            /* stepper0 starts <-- direction */
do_step = 1;                          /* start stepper0 */
wait = 1;                /* wait for stepper0 to get started */
while(wait == 1) {};
limiting = 1;              /* start looking for limit switch */
while(limit == 0) {};           /* wait to hit limit switch */
lcontrol = 0x45;   /* dir0=0, dir1=1,  set up change in dir0 */
done0 = 0;                     /* change stepper0 direction */      590
detects = 0;                      /* reset detector count */
while(detects < 1) {};          /* wait for 1st IR detection */
count = dmin1;                 /* value for clk1 16-bit data */
done1 = 0;                            /* start extruder */
wait = 1;                /* wait for stepper1 to get started */
while(wait == 1) {};
detects = 1;
while(detects < 4) {};          /* wait for 4th IR detection */
wait = 1;                   /* get ready to stop everything */
while(wait == 1) {}; /* right after out of software timer ISR */   600
lcontrol = 0x00;             /* set up disabling of steppers */
done0 = 0;                              /* stop steppers */
wait = 1;
while(wait == 1) {};

stop = 1;                                   /* stop drum */
```

```
wait = 1;
while(wait == 1) {};
disable();                                                    610

for(i=0;i<MAXI;i++)          /* relax .....*/
  { for(j=0;j<MAXJ;j++);};

/* CROSS-LINKING .............*/

spin = 0;
cross = 1;
gather = 0;
                                                              620
homing = 1;
limiting = 0;
ovpermin = 916;
r = 0;
s = 0;
rotate = 0;
clstop = 0;
smax = (60*CLHOURS + CLMINS)*ovpermin;
rmax = ROTMINS*ovpermin;
first = 1;                                                    630
stop = 0;
ch_switch = 0;
n = 0;
m = 0;
dflag = 0;
dflag2 = 0;
dist = REV;
gain = 4000;          /* ((int) SPEED1)/(EMIN - 1); */
errmax = 2*REV;
p = 0;                                                        640
ioport1 = 0;

ioc0 = 0x04;          /* HSI pin 1 enabled as input */
ioc1 = 0x02;          /* HSI data avail int. on holding reg load */
                   /* also, P2.2 provides EXTINT1 */
ioc2 = 0x00;          /* Timer2: normal mode, count up, FFFF/OOOO */
wsr = 1;
t2control = 0x01;      /* Timer2 clocked internally */
wsr = 0;
hsi_mode = 0x0C;       /* HSI.1 event on every transition */  650

imask1 = 0x20;          /* EXTINT1 looking for home pulse */
ipend1 = 0;
```

```
int_mask = 0x34;    /* Interrupt on software timer, hsi & hsi.0 */
int_pending = 0;
hso_command = 0x18; /* generate interrupt on soft. timer0 */
hso_time = timer1 + Ts;

cout = SPEED3;                                                    660

enable();

while(ch_switch==0)        /* wait for homing pulse */
  {};

for(i=0;i<MAXI;i++)        /* Little waste of time, so drum */
  { for(j=0;j<MAXJ;j++);}; /* settles. */

stop = 1;                  /* stop the drum, and position control */  670
disable();
imask1 = 0x10;             /* enable Timer2 overflow interrupt */
ipend1 = 0;
int_mask = 0x34;
int_pending = 0;

lcontrol = 0x10;           /* start UV lamp */
ioport1 = lcontrol;
hso_command = 0x22;        /* set HSO.2 */
hso_time = timer1 + 2;                                            680
k = a + b;
hso_command = 0x02;        /* clear HSO.2 */
hso_time = timer1 + 2;

ioc0 = 0x06;               /* reset Timer2, hsi1 still enabled */
enable();

while(clstop == 0)
  {
    rotate = 0;                                                   690
    while(rotate == 0)
      {};
    r = 0;
    dist = STEPS;
    n = 0;
    dflag = 0;
    ch_switch = 0;
    stop = 0;              /* can start drum */
    if(first == 1)
          {                                                       700
          hso_command = 0x18;/* interrupt on soft. timer 0 */
          hso_time = timer1 + 2; /* immediately */
```

```
        first = 0;
        }
   while(ch_switch==0)        /* wait for positioning */
     {};
   for(i=0;(i<MAXI) && (clstop == 0);i++)     /* drum settling */
     { for(j=0;(j<MAXJ) && (clstop == 0);j++);};
   stop = 1;              /* stop the drum, and position control */
 }                                                                    710


lcontrol = 0x00;          /* stop UV */
ioport1 = lcontrol;
hso_command = 0x22;       /* set HSO.2 */
hso_time = timer1 + 2;
k = a + b;
hso_command = 0x02;       /* clear HSO.2 */
hso_time = timer1 + 2;
                                                                      720
wait = 1;
while(wait == 1) {};
disable();

for(i=0;i<MAXI;i++)          /* relax .....*/
  { for(j=0;j<MAXJ;j++);};

/* GATHERING ..... */

spin = 0;                                                             730
cross = 0;
gather = 1;

stop = 0;
ch_switch = 0;
detects = 0;
limit = 0;
homing = 1;
limiting = 0;
do_step = 0;                                                          740
done0 = 0;
done1 = 1;
detstart = 0;
detend = 0;
sol_on = 0;
n = 0;
m = 0;
dflag = 0;
dflag2 = 0;
dist = REV;                                                           750
```

```
errmax = 2*REV;
p = 0;
dmin0 = 1000000/GMCLK0; /* Minimum clock count */
ioport1 = 0;

ioc0 = 0x44;          /* HSI pins 1,3 enabled as inputs */
ioc1 = 0x02;          /* HSI data avail int. on holding reg load */
                   /* also, P2.2 provides EXTINT1 */
hsi_mode = 0x4C;      /* HSI.1 event on every transition */
                   /* HSI.3 on positive transitions */                    760


imask1 = 0x20;          /* EXTINT1 looking for home pulse */
ipend1 = 0;

int_mask = 0x34;    /* Interrupt on software timer, hsi & hsi.0 */
int_pending = 0;
hso_command = 0x18; /* generate interrupt on soft. timer0 */
hso_time = timer1 + Ts;
                                                                          770
cout = SPEED3;

enable();

while(ch_switch==0)       /* wait for homing pulse and switch */
  {};
homing = 0;               /* home pulse found */

for(i=0;i<MAXI;i++)       /* Little waste of time, so drum */
  { for(j=0;j<MAXJ;j++);}; /* settles. */                                   780

count = dmin0;

while (m < REV_STEPS)
  {
    m++;
    lcontrol = 0x03;          /* stepper0 goes <-- */
    do_step = 1;              /* start the stepper */
    done0 = 0;
    limit = 0;                                                            790
    while(limit == 0) {};       /* until limit switch */
    detects = 0;
    lcontrol = 0x01;          /* stepper0 goes --> */
    done0 = 0;                /* change stepper direction */
    while(detects < ESTART) {}; /* wait for IR ESTART*/
    sol_on = 1;               /* activate solenoid */
    done0 = 0;
    wait = 1;
```

```
    while(wait == 1) {};        /* wait for solenoid action */
    while(detects < EEND) {};   /* wait for IR EEND*/                    800
    done0 = 0;                  /* deactivate solenoid */
    wait = 1;
    while(wait == 1) {};        /* wait for sol, stop stepper */
    stop = 1;                   /* stop motor for a little while */

    for(i=0;i<100;i++)          /* Little waste of time, for arm */
        { for(j=0;j<500;j++);};  /* to pull back */

    detstart = 0;
    detend = 0;                                                          810
    dist = STEPS;
    n = 0;                      /* reset step variable */
    dflag = 0;                  /* reset direction flag */
    ch_switch = 0;              /* motor is started */
    stop = 0;
    while(ch_switch == 0);      /* positioning...
    for(i=0;i<100;i++)          /* Little waste of time, so drum */
        { for(j=0;j<500;j++);};  /* settles. */
  }
                                                                         820
  wait = 1;
  while(wait == 1) {};
  spin = 0;
  cross = 0;
  gather = 0;

  while(1==1) {};

}
                                                                         830
```

Appendix B

Matlab Code

This chapter provides the Matlab code used to simulate digital PI control. Appendix B.1 details the program written to calculate the poles locations of the closed loop system. Appendix B.2 does a little more, calculating the z plane poles and zeros as well as the equivalent s-plane poles, damped frequency, and overshoot to a step response. The overshoot is returned as a variable because we want low overshoot. Appendix B.3 calculates the Z-transfer function of both the speed output, disturbance and the error voltages, these can be used to plot the step responses which will be compared with the observed step response. Appendix B.4 was written to simulate the action of the digital controller. This is useful in modeling such effects as anti-windup, fixed point arithmetic and the effect of the peripheral circuitry such as the analog circuitry, A/D and DAC.

B.1 Closed Loop Pole Locations

```
function [x,y,z,b,c,r1,r2,tau1,tau2] = rcalc(Ka,Km,Kt,B,hp,hi,Ts,tf)
%
% [x,y,z,b,c,r1,r2,tau1,tau2] = rcalc(Ka,Km,Kt,B,hp,hi,Ts,tf)
% calculates the paramters of digital control characteristic
% equation. Example for my parameters....
% [x,y,z,b,c,r1,r2,tau1,tau2] = rcalc(.01172,.61,.477,4.61e-2,
%                                     (600/30),(2/30),500e-6,.175);

z = exp(-Ts/tf);
x = Ka*Km*Kt*(1-z)/B;
y = -1-z;
b = x*hp + y;
c = x*(hi-hp) + z;
```

```
[rts] = roots([1 b c]);
r1 = rts(1);
r2 = rts(2);
zmag1 = abs(r1);
zmag2 = abs(r2);
tau1 = -Ts/log(zmag1);
tau2 = -Ts/log(zmag2);
```

The above function was used to find the pole locations in Section 4.1.3.

B.2 Closed Loop Response Parameters

The values in the parentheses are the values that were chosen.

```
function [zp,zz,mag,sp,wd,po] = dmr(tf,Ts,Kt,K,hp,hi)
% [zp,zz,mag,sp,wd,po] = dmr(tf,Ts,Kt,K,hp,hi);
%
% This function returns the values of the closed loop poles,
% zeros and magnitudes in the z-plane. Also returned are
% the s-plane poles, oscillation frequency and overshoot.
% tf is the motor damping time constant (.175)
% Ts is the sample time (5e-4)
% Kt is the tach constant (.477)
% K is the value of Ka*Km/B, where Ka is the amplifier current
% to voltage gain, Km is the motor constant and B is the damping
% coefficient. (.155 i.e. .01172*.61/4.61e-2)
% hp is the proprotional gain and hi the integral gain. (20,.067)
% e.g....[zp,zz,mag,sp,wd,po] = dmr(.175,5e-4,.477,.155,20,.067);

E    = exp(-(Ts/tf));
A    = K*(1-E);
G1n  = [0 A];
G12n = Kt*G1n;
G1d  = [1 -E];
h3   = hi - hp;
Dn   = [hp h3];
Dd   = [1 -1];
Bn   = [Kt];
Bd   = [1];
[Fn,Fd] = series(Dn,Dd,G1n,G1d);
```

```
[Tn,Td] = feedback(Fn,Fd,Bn,Bd,-1);
zp = roots(Td);
zz = roots(Tn);
[mag,wn,rho] = ddamp(Td,Ts);
alpha1 = wn(1)*rho(1);
alpha2 = wn(2)*rho(2);
if rho(1) ~= 1
   po = 100*(exp((-pi*rho(1))/(sqrt(1-rho(1)*rho(1)))));
   wd = wn(1)*(sqrt(1-rho(1)*rho(1)));
   sp = [(alpha1+i*wd);(alpha1-i*wd)];
else
   po = 0;
   wd = 0;
   sp = [alpha1; alpha2];
end
```

For the suggested values which correspond to the PI gains used and the parameters estimated, the results are: z-plane poles at $0.9965 \pm 0.0013j$, z-plane pole radius of 0.9965, s-plane poles at $7.0828 \pm 2.56562j$ and damped oscillations at 2.5626 $[rad/s]$ and an overshoot of 1.7%, which are reasonable values.

B.3 Closed Loop Transfer Functions

The closed loop, command output, as well as the error transfer functions, are calculated in this Matlab function. The results can be plotted using the Matlab function dstep(num,den), which takes the numerator and denominator coefficients and plots the step response.

```
function [Tn,Td,En,Ed,Cn,Cd] = dmt(tf,Ts,Kt,K,hp,hi)
% [Tn,Td,En,Ed,Cn,Cd] = dmt(tf,Ts,Kt,K,hp,hi);
%
% This function calculates the Z-transfer function for the
% closed loop of the motor with PI control. The returned
% variables are the numerator and denominator coefficients
% in decreasing order of z.
% tf is the motor damping time constant (.175)
% Ts is the sample time (5e-4)
% Kt is the tach constant (.477)
% K is the value of Ka*Km/B, where Ka is the amplifier current
% to voltage gain, Km is the motor constant and B is the damping
```

```
% coefficient. (0.155, = .01172*.61/4.61e-2)
% hp is the proprotional gain and hi the integral gain. (20,.067)
% e.g...[Tn,Td,En,Ed,Cn,Cd] = dmt(.175,5e-4,.477,.155,20,.067)
E    = exp(-(Ts/tf));
A    = K*(1-E);
G1n  = [0 A];
G12n = Kt*G1n;
G1d  = [1 -E];
h3   = hi - hp;
Dn   = [hp h3];
Dd   = [1 -1];
Bn   = [Kt];
Bd   = [1];
[Fn,Fd] = series(Dn,Dd,G1n,G1d);
[Tn,Td] = feedback(Fn,Fd,Bn,Bd,-1);
[Ln,Ld] = series(Dn,Dd,G12n,G1d);
[En,Ed] = feedback([1],[1],Ln,Ld,-1);
[Cn,Cd] = feedback(Dn,Dd,G12n,G1d,-1);
```

The step responses are shown in Figure B-1. The output settles in about 800 samples or 0.4 seconds, and there is no overshoot.

B.4 Digital Controller Simulation

This code can be used to observe the effects of the DAC, A/D gains, and the limiting of the PI variables cout (command) and acc (accumulator). It can also be used to observe the effect of the initial measured voltage offset of 1.12 volts. The vector results returned can be plotted to see the evolution of their states.

```
function [vel,ve,vpi,cout,acc] = pisim(tf,Ts,Kt,K,h1,h2,Vin,
                                       Vref,mult,vm1,pts)
% [vel,ve,vpi,cout,acc] =
% pisim(tf,Ts,Kt,K,h1,h2,Vin,Vref,mult,vm1,pts);
%
% This function simulates the action of the digital controller
% and its interaction with the peripheral components. The
% parameters passed in:
% tf is the motor damping time constant (.175)
% Ts is the sample time (5e-4)
% Kt is the tach constant (.477)
```

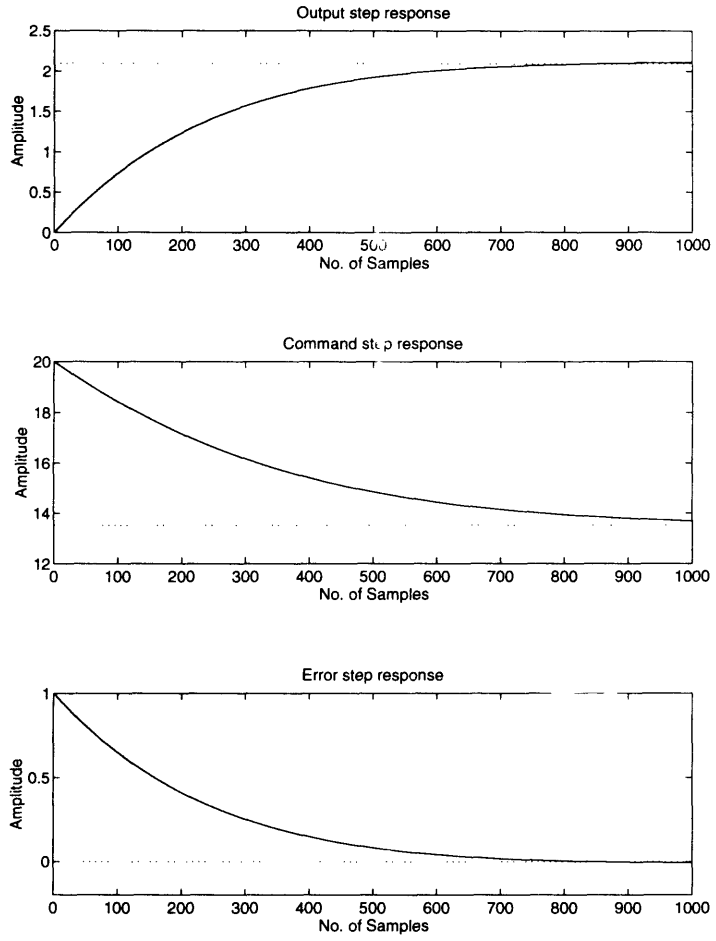

Figure B-1: Speed, control command and error step responses from transfer function.

```
% K is the value of Ka*Km/B, where Ka is the amplifier current
% to voltage gain, Km is the motor constant and B is the friction
% coefficient (0.155, = .01172*.61/4.61e-2)
% h1 is the proprotional gain and h2 the integral gain (20, .067)
% Vin is the command voltage (1)
% Vref is the A/D reference voltage (4.93)
% mult is what the PI gains are multiplied by to be used by the
% microprocessor (1)
% vm1 is the initial measured velocity (0)
% pts is the number of samples (1500)
% The results returned are vectors of the velocity, error, PI control
% output voltages and cout and accumulator variables.
% e.g...[vel,ve,vpi,cout,acc] =
% pisim(.175,5e-4,.477,.155,20,.067,1,4.93,1,0,1500);

E    = exp(-(Ts/tf));        % open loop pole
A    = K*(1-E);              % ZOH  plant numerator
Fad  = 1023/Vref;            % A/D gain
Fdac = 10/32768;             % DAC gain
gain = round(2/(Fad*Fdac)); % Digital gain
h11  = h1*mult*gain;         % adjusting PI gains
h21  = h2*mult*gain;

vm(1)   = vm1;
mvel(1) = vm1/2;             % analog circuitry (divide by 2)
vel(1)  = vm1/Kt;            % tachometer reading
Vinr    = Vin*.5;            % analog circuitry (divide by 2)

for n=1:pts                    % similar to controller algorithm
  int_vi(n) = round(Fad*Vinr); % in 80C196KC code (Appendix A.1)
  int_vm(n) = round(Fad*mvel(n));
  verr(n)   = int_vi(n) - int_vm(n);
  ve(n)     = verr(n)/Fad;

  if n == 1
    acc(n) = 0;
  else acc(n) = verr(n-1) + acc(n-1);
  end

  cout(n)   = h21*acc(n) + h11*verr(n);
```

```
  if cout(n) >= 30000          % limit control output
  cout(n) = 30000;
  end
  if cout(n) <= 0
        cout(n) = 0;
  end

  if acc(n) >= 15000           % antiwindup
  acc(n) = 15000;
  end
  if acc(n) <= 0
acc(n) = 0;
  end

  ctemp(n)  = round(32767 - cout(n));   % output of microprocessor
  vpi(n)    = (32767 - ctemp(n))*Fdac;  % DAC transfer function
  if n ~= pts
     vel(n+1)  = A*vpi(n) + E*vel(n);   % plant action
     vm(n+1) = Kt*vel(n+1);             % speed measurement
     mvel(n+1)   = vm(n+1)/2;
  end
end
```

Referring to Figure B-2, the simulated values for the values above show that the speed response differs from that obtained in the transfer function. The settling time is increased to about 0.7 and the dc gain is decreased from about 2 to about 1.5. The decrease in dc gain can be attributed in part the fact that Vref (4.93 volts) is less than the 5 volts anticipated when calculating the PI gains. The increased settling time could be due to the saturation of the control output at 30,000, and the accumulator at 15000.

These Matlab simulations are compared with empirical results, in Chapter 5.

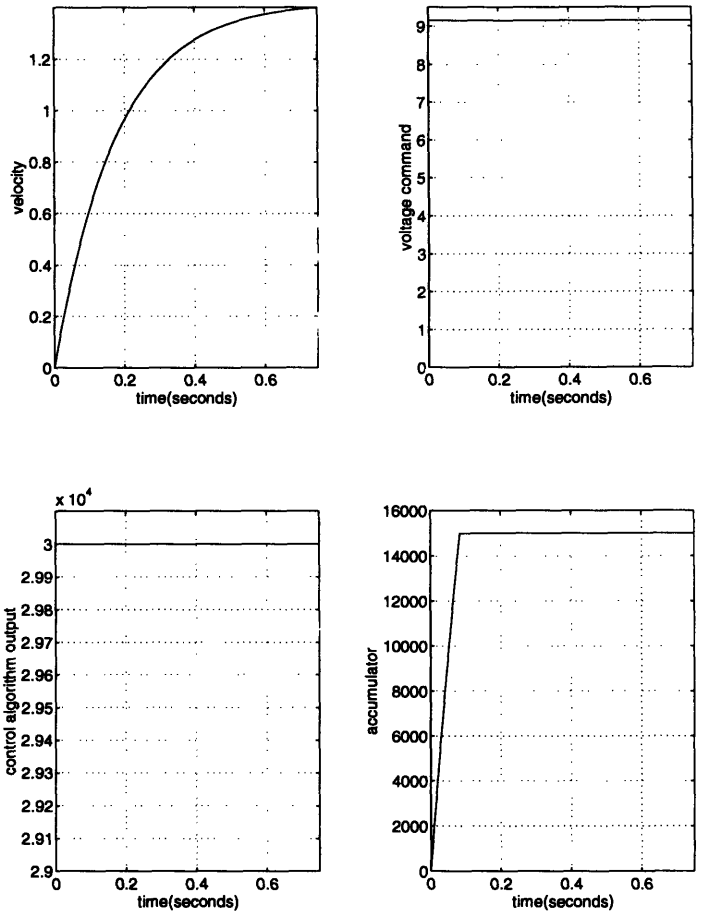

Figure B-2: Speed, control command and error step responses from simulation.

Appendix C

PAL and Logic Code

C.1 Drum Speed Command PAL

This PAL is used in speed measurement. It monitors the encoder and the ripple carry overflow (RCO4) of the 4th counter. When the 15-bit counter is about to overflow, and the RCO4 input is high, then the PAL freezes the counter so that it does not overflow. The counter value is held at 32767 ($2^{15} - 1$), until a new encoder pulse is detected. Otherwise, every time a new encoder pulse is detected, the counter value is latched and then the counter cleared. The PAL sends out the clock signals for the latches at the input and output of the EPROMS used to help measure the speed (see Appendix F).

```
/**  Inputs  **/

Pin  1        =  clk          ;      /* clock input, 1 MHz */
Pin  2        =  rco4         ;      /* RCO of counter 4 */
Pin  3        =  encoder      ;      /* synchronized encoder input */
Pin  4        =  !reset       ;      /* manual reset signal  */

/**  Outputs  **/

Pin  14       =  enp          ;      /* count enable                */
Pin  15       =  !clear       ;      /* clear count                 */
Pin  16       =  !load4       ;      /* load couner 4 with value 8 */
Pin  17       =  latch1       ;      /* latch into EPROM            */
Pin  18       =  latch2       ;      /* latch out of EPROM          */
Pin  19       =  stop         ;      /* stop cont flag              */
Pin  [20..22] =  [b0..2]      ;      /* state count                 */
```

```
/** Declarations and Intermediate Variable Definitions **/

FIELD state_count =  [b2..0];
enp               =  !rco4 & !stop;

/**  Counting Machine  **/
sequence state_count {

PRESENT 0
        if  reset # (!rco4 & !encoder)   NEXT 0;
        if  rco4 & !encoder              NEXT 3 OUT stop OUT latch1;
        if  rco4 & encoder               NEXT 1 OUT stop OUT latch1;

PRESENT 1
        if reset                         NEXT 0;
        DEFAULT                          NEXT 2 OUT clear;

PRESENT 2
        if reset                         NEXT 0;
        DEFAULT                          NEXT 0 OUT latch2 OUT load4;

PRESENT 3
        if reset                         NEXT 0;
        if encoder                       NEXT 6 OUT clear;
        DEFAULT                          NEXT 4 OUT stop;

PRESENT 4
        if reset                         NEXT 0;
        if encoder                       NEXT 6 OUT clear;
        DEFAULT                          NEXT 5 OUT stop OUT latch2;

PRESENT 5
        if reset                         NEXT 0;
        if !encoder                      NEXT 5 OUT stop;
        DEFAULT                          NEXT 6 OUT clear;

PRESENT 6

        DEFAULT                          NEXT 0 OUT load4;
```

```
PRESENT 7                                   NEXT 0;

                }

/**  Logic Equations  **/

[clear,load4,latch1,latch2,stop].ar = 'b'0;
[b0..2].ar                          = 'b'0;
[clear,load4,latch1,latch2,stop].sp = 'b'0;
[b0..2].sp                          = 'b'0;
```

C.2 Drum Direction PAL

The drum direction PAL uses two quadrature outputs of the shaft encoder to determine the direction of drum spin. It also outputs a signal called *multiple* which is toggled whenever one of the shaft outputs, *ain*, undergoes a transition. *multiple* is thus twice the frequency of *ain*.

```
/**  Inputs  **/

Pin  1         =  clk           ;      /* clock input, 1 MHz */
Pin  2         =  ain           ;      /* encoder quadrature output !A */
Pin  3         =  bin           ;      /* encoder quadrature output !B */

/**  Outputs  **/

Pin  14        =  set             ;      /* set flipflop */
Pin  15        =  !reset          ;      /* reset flipflop */
Pin  16        =  multiple        ;      /* ain freqeuncy doubled */
Pin  17        =  wait1           ;      /* delay variable */
Pin  18        =  wait2           ;      /* delay variable */
Pin  [19..20]  =  [b0..1]         ;      /* state variables  */


/** Declarations and Intermediate Variable Definitions **/

FIELD state_count =  [b1..0];
```

```
/**  Counting Machine  **/

sequence state_count {

PRESENT 0
        if  ain & !bin                  NEXT 1;
        DEFAULT                         NEXT 0;

PRESENT 1
if !ain                         NEXT 2 OUT reset;
if bin                          NEXT 2 OUT set;
DEFAULT                         NEXT 1;

PRESENT 2
if !ain                         NEXT 0;
        DEFAULT                         NEXT 2;

}

/**  Logic Equations  **/

wait1.d = ain;
wait2.d = wait1;
multiple = wait1 $ wait2;

[set,reset,wait1,wait2,b0,b1].ar = 'b'0;
[set,reset,wait1,wait2,b0,b1].sp = 'b'0;
```

C.3 Stepper PALs

Three PALs, timer1, timer2 and timer3, are used to help set the *clock*, *reset*, *enable*, and *direction* variables for a selected stepper motor.

C.3.1 Stepper-Timer1 PAL

Timer1 PAL sends the output enable signals for the three latches which precede the 8254. It also sends out the *write* (WR), *A0* and *A1* inputs to the 8254 [9]. A *ready* signal is output by the PAL to inform the timer2 PAL that it has sent out the inputs the 8254 needs to set up a clock. The *ready* signal is needed by the timer2 PAL only when the initial clock rate is being set.

```
/**  Inputs  **/

Pin  1        =  clk          ;      /* clock input, 1 MHz */
Pin  2        =  rst          ;      /* reset pin */
Pin  3        =  value        ;      /* new 8254 count requested   */
Pin  4        =  counter      ;      /* new clock (stepper) being chosen */
Pin  5        =  done         ;      /* clock rates finished */
Pin  6        =  c0           ;      /* stepper0 variable */
Pin  7        =  c1           ;      /* stepper0 variable */

/**  Outputs  **/

Pin  14       =  !oecw          ;      /* Output enable control word */
Pin  15       =  !oelsb         ;      /* Output enable LSByte */
Pin  16       =  !oemsb         ;      /* Output enable MSByte */
Pin  17       =  a0             ;      /* 8254 Mode information */
Pin  18       =  a1             ;      /* 8254 Mode information */
Pin  19       =  !wr            ;      /* 8254 write signal */
Pin  [20..22] =  [b0..2]        ;      /* State variables */
Pin  23       =  !ready         ;      /* ready signal, can start */

/** Declarations and Intermediate Variable Definitions **/

FIELD state_count =  [b2..0];
a0                =  oecw # !oecw & c0;
a1                =  oecw # !oecw & c1;

/**  Counting Machine  **/

sequence state_count {

PRESENT 0
        if  rst                          NEXT 0;
        DEFAULT                           NEXT 1 OUT oecw OUT wr;

PRESENT 1                                 NEXT 2 OUT oecw;

PRESENT 2                                 NEXT 3 OUT oelsb OUT wr;
```

```
PRESENT 3                                   NEXT 4 OUT oelsb;

PRESENT 4                                   NEXT 5 OUT oemsb OUT wr;

PRESENT 5                                   NEXT 6 OUT oemsb;

PRESENT 6                                   NEXT 7 OUT ready;

PRESENT 7
        if value & !done                    NEXT 3 OUT oelsb OUT wr;
        if counter & !done & !value         NEXT 1 OUT oecw OUT wr;
        if done                             NEXT 0;
        DEFAULT                             NEXT 7;
}

/** Logic Equations **/

[oecw,oelsb,oemsb,a0,a1,wr,ready].ar = 'b'0;
[b0..2].ar                          = 'b'0;
[oecw,oelsb,oemsb,a0,a1,wr,ready].sp = 'b'0;
[b0..2].sp                          = 'b'0;
```

C.3.2 Stepper-Timer2 PAL

The additional signals which need to be sent to the stepper logic sequencer and amplifier, are provided by this PAL. It outputs the *reset* and *direction* for a specified stepper, as well as a common enable output for both steppers.

```
/** Inputs **/

Pin  1      =  clk          ;     /*  clock input, 1 MHz */
Pin  2      =  ready        ;     /*  ready to enable stepper boards*/
Pin  3      =  enable       ;     /*  enable PAL  operation */
Pin  4      =  direction0   ;     /*  direction of linear table */
Pin  5      =  direction1   ;     /*  direction of extruder    */
Pin  6      =  c0           ;     /*  = 0 if stepper0, = 1 otherwise*/

/** Outputs **/

Pin  14     =  counter        ;     /*  c0 value has changed */
```

```
Pin  15       =  rst              ;      /*  output reset for timer2 PAL  */
Pin  16       =  en01             ;      /*  enable stepperes, 0,1 */
Pin  17       =  dir0             ;      /*  direction output, linear table */
Pin  18       =  !reset0          ;      /*  linear table reset output */
Pin  19       =  dir1             ;      /*  linear table direction   */
Pin  20       =  !reset1          ;      /*  extruder reset ouput     */
Pin  21       =  wait             ;      /*  extruder reset output   */
Pin  [22..23] =  [b0..1]          ;      /*  state variables */*/


/** Declarations and Intermediate Variable Definitions **/

FIELD state_count =  [b1..0];
rst.d          = enable;
en01.d         = !ready & !enable;
wait.d         = c0;
counter.d      = c0 $ wait;
dir0           = !c0 & direction0 & (b0 # b1);
dir1           = c0 & direction1 & (b0 # b1);
reset0         = !c0 & b1 & b0;
reset1         = c0 & b1 & b0;

/**  Counting Machine  **/

sequence state_count {

PRESENT 0
        if  !en01                       NEXT 0;
        DEFAULT                         NEXT 1;

PRESENT 1
        if  !en01                       NEXT 0;
        DEFAULT                         NEXT 3;

PRESENT 2
        if  !en01                       NEXT 0;
        DEFAULT                         NEXT 2;

PRESENT 3
        if  !en01                       NEXT 0;
```

```
        DEFAULT                                 NEXT 2;
}

/**  Logic Equations  **/

[rst,en01,wait,counter].ar          = 'b'0;
[dir0,reset0,dir1,reset1,b0,b1].ar  = 'b'0;
[rst,en01,wait,counter].sp          = 'b'0;
[dir0,reset0,dir1,reset1,b0,b1].sp  = 'b'0;
```

C.3.3 Stepper-Timer3 PAL

This PAL controls the gate inputs of the 8254 [9]. When the gate signals are high, the output clock signals are enabled, but if the gate inputs are low, the output frequency is zero. It also sends out an enabling signal to the timer1 and timer2 PALs in reponse to the lcontrol's enable signal. The manual switch has to be on for the enable signal to be sent out to the other PALs.

```
/**  Inputs  **/

Pin  1        =  clk          ;      /* clock input, 1 MHz */
Pin  2        =  c0           ;      /* stepper choice c0=0 stepper0 */
Pin  3        =  renable      ;      /* enable from microprocessor */
Pin  4        =  switch       ;      /* manual switch input  */

/**  Outputs  **/

Pin  14       =  enable          ;      /* enable inputs for other PALs  */
Pin  15       =  gate0           ;      /* gate of output0 of 8254 */
Pin  16       =  gate1           ;      /* gate of output1 of 8254 */
Pin  17       =  wait0           ;      /* delay variable  */
Pin  18       =  wait1           ;      /* delay variable  */
Pin  19       =  a               ;      /* intermediate variables */
Pin  20       =  b               ;      /* intermediate variables */

/**  Logic Equations  **/

enable       =  renable & switch;
gate0        =  a & enable;
gate1        =  b & enable;
```

```
wait0.d      =  gate0;
wait1.d      =  gate1;
a            =  !c0 # wait0;
b            =  c0 # wait1;

[wait0,wait1].ar          = 'b'0;
[wait0,wait1].sp          = 'b'0;
```

C.4 Low and High Byte Enable Sequencer PAL

The two bytes of the DAC as well as the two count bytes of the 8254 are loaded from IOPORT1. This PAL determines the destination of the IOPORT1 data given the value of HSO.1. It also sends out a signal, called *value*, which tells the timer1 PAL that a new value for the current stepper motor is about to be loaded.

```
/**  Inputs  **/

Pin  1        =  clk           ;      /* clock input, 1 MHz */
Pin  2        =  hso0          ;      /* HSO.0  */
Pin  3        =  hso1          ;      /* HSO.1 */
Pin  4        =  val_disable   ;      /* variable can disable value */

/**  Outputs  **/

Pin  14       =  splo              ;      /* Stepper low byte clock input */
Pin  15       =  sphi              ;      /* Stepper high byte clock input */
Pin  16       =  stlo              ;      /* DAC low byte clock input  */
Pin  17       =  sthi              ;      /* DAC high byte clock input */
Pin  18       =  value             ;      /* value is input to timer1 */
Pin  19       =  wait              ;      /* delay variable  */
Pin  20       =  wait2             ;      /* delay variable  */

/** Declarations and Intermediate Variable Definitions **/


/**  Logic Equations  **/

wait.d    =  hso0;
splo    =  !hso1 & hso0 & !wait;
```

```
sphi    =  !hso1 & wait & !hso0;
stlo    =  hso1 & hso0 & !wait;
sthi    =  hso1 & wait & !hso0;
wait2.d   =  sthi;
value.d   =  wait2 & !val_disable;

[wait,wait2,value].ar = 'b'0;
[wait,wait2,value].sp = 'b'0;
```

C.5 EPROM Code

The EPROM is used as a lookup table. It essentially inverts its input value to ensure that our tachometer output is proportional to the encoder input frequency. As stated in Equation 4.22, the output corresponds to

$$output = 32767 - B/address \tag{C.1}$$

The variable *address* is the counter output. The EPROM code, written in C, computes the different output values and places the values in an array. The index into the array corresponds to the *address* value. The content of the array is the *output*. This *output* which is a two byte value, is then divided into low and hgh bytes and stored in the binary format that can be used to program the EPROMS.

```
/* The implemented version, with 3.6864MHz clock */

#include <stdio.h>
#include <math.h>

#define MAXADDR 32768
#define BMULT 120792269
#define OFFSET 32767
#define START 3687

                                                              10
int data[32769];

main()
{
  char c, num;
  long int i;
  FILE *fopen(), *inf, *outf, *outf2;
```

```
/* Blank Array */
for (i = 0; i < MAXADDR; i++)                                    20
        data[i] = 0;

/* Fill array with data upto data[32767] */
for (i = START; i < MAXADDR; i++)
        data[i] = OFFSET - BMULT / i;

data[i] = OFFSET;

inf = fopen("i_hexfunc.dat", "w");
for (i = 0; i < MAXADDR; i ++)                                   30
        fprintf (inf,"%.4x\n",data[i]);

fclose(inf);


inf = fopen("i_hexfunc.dat", "r");
outf = fopen("i_highbyte.dat","w");
outf2 = fopen("i_lowbyte.dat","w");
for (i = 0; i < MAXADDR; i ++) {
                fscanf(inf,"%c",&c);                             40
                fprintf(outf,"%c",c);
                fscanf(inf,"%c",&c);
                fprintf(outf,"%c \n",c);

                fscanf(inf,"%c",&c);
                fprintf(outf2,"%c",c);
                fscanf(inf,"%c",&c);
                fprintf(outf2,"%c \n",c);
                fscanf(inf,"%c",&c);
                                                                 50
}

fclose(inf);
fclose(outf);
fclose(outf2);

inf = fopen("i_highbyte.dat", "r");
outf = fopen("i_high.bin","w+b");
for (i = 0; i < MAXADDR; i++) {
        fscanf(inf,"%x",&num);                                   60
/*      printf ("%d\n",(int) num); */
        fwrite(&num,sizeof(char),1,outf);
}
fclose(inf);
fclose(outf);
```

```
  inf = fopen("i_lowbyte.dat", "r");
  outf = fopen("i_low.bin","w+b");
  for (i = 0; i < MAXADDR; i++) {
          fscanf(inf,"%x",&num);                                          70
/*        printf ("%d\n",(int) num); */
          fwrite(&num,sizeof(char),1,outf);
  }
  fclose(inf);
  fclose(outf);

}
```

Appendix D

Parameter Calculation Data

Parameter measurement results are summarized in this appendix.

Table D.1: Current Amplifier Gain Measurements

Vb (volts)	Im (mA)	Vb (volts)	Im (mA)
-3.47	-45.2	5.79	59.8
-4.46	-58.7	6.28	66.9
-5.06	-66.4	6.49	70.4
-5.82	-76.9	6.58	72.5
-6.09	-79.8	7.04	79.6
-6.79	-89.2	7.57	89.1
-7.31	-96.7	8.12	100.3
-7.68	-100.1	8.49	107.9
-8.14	-102.9		
-8.73	-109.9		
-9.07	-113		
-9.9	-112.9		

Table D.2: Motor Torque Constant Measurements

Vb(volts)	Im (amps)	speed (RPM)
5	0.066	53.333333
7	0.075	85
9	0.08	113
10	0.084	127
12	0.093	161

Table D.3: Tachometer Constant Measurements

speed (rad/s)	tach voltage (volts)
3.959	1.88
5.384	2.62
6.464	3.06
7.272	3.48
8.378	4.06
9.226	4.4

Appendix E

Derivations

Appendix E.1 presents the analog P and PI control equations in greater detail. Appendix E.2 derives the digital PI control equations. Appendix E.3 details the derivation of the tachometer constant used in Chapter 4.

E.1 Analog P and PI Control

The equations developed will be made with reference to Figure 4-2 shown in Chapter 4.

Analog P Control

In proportional control, the compensator $G_c(s) = G$. The closed loop speed response is given by

$$\frac{\omega}{V_{in}} = \frac{(GK_{am}/J)}{s + \frac{B+GK_{amT}}{J}} \tag{E.1}$$

where

$$K_{am} = K_a K_m \tag{E.2}$$

The closed loop pole moves from $\tau_f = \frac{J}{B}$ to $\tau' = J(B + GK_{amT})$, that is, the closed loop moves further into the negative real axis, speeding up the step response. The new time constant, τ', was used to estimate the value of J in Section 4.1.2. The response to the step disturbance input shown in Figure 4-2 is

$$\frac{\omega}{T_d} = -\frac{1/J}{s + \frac{B+GK_{amT}}{J}} \tag{E.3}$$

Comparing Equations E.1 and E.3, shows that the disturbance rejection can be improved by increasing the gain G. The steady state error to a step input is not zero, but

$$\frac{E}{V_{in}}_{ss} = \frac{B}{B + GK_{amT}} \tag{E.4}$$

If the damping coefficient is zero, the error is zero. This is because the closed loop contains an integrator which produces a zero steady state error to step inputs. It should be noted that the steady state error to a step disturbance is given by

$$\frac{E}{T_d}_{ss} = \frac{K_T}{B + GK_{amT}} \tag{E.5}$$

which is not zero if B is zero, but is very close to zero if the gain G is made very high.

Proportional control was studied because of its ease of implementation. It is useful in estimating some parameter values. It is also a good way to verify the plant model developed. For example, if P control produced a ringing response then it would be clear that there was an unmodeled pole in the system.

Analog PI Control

The controller used in analog PI control is

$$G_c(s) = \frac{G(\tau s + 1)}{\tau s} \tag{E.6}$$

The low frequency gain is G/τ and the high frequency gain is G with the break-point occurring at $1/\tau$ $[rad/s]$. The circuit which was used to implement this block is shown in Figure E-1. G equals $(1.2 + 10)/1.2$ or about 10 and τ equals $(178.6K\Omega * 0.82\mu F)$ or about 0.146. Matlab was used to find a satisfactory closed loop response by choosing values for G and τ which produced no overshoot in the step response.

The closed loop response given by Equation 4.9 corresponds to the characteristic

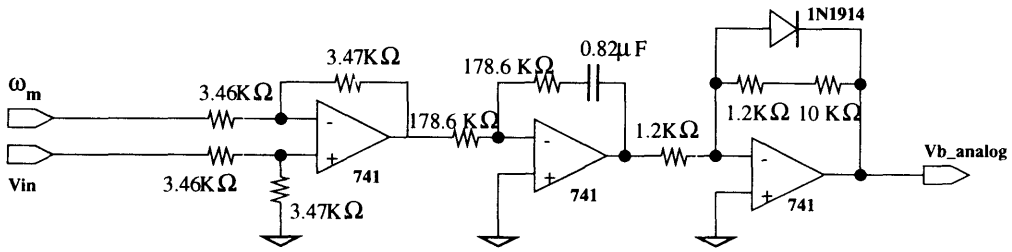

Figure E-1: Analog PI Compensator Circuit.

equation

$$s^2 + \frac{B + GK_{amT}}{J}s + \frac{GK_{amT}}{\tau J} \tag{E.7}$$

The disturbance response is given by

$$\frac{\omega}{T_d} = \frac{-1}{J} \cdot \frac{s}{s^2 + \frac{B+GK_{amT}}{J}s + \frac{GK_{amT}}{\tau J}} \tag{E.8}$$

and the steady state values by

$$\frac{\omega}{V_{in}} = \frac{1}{K_T} \quad , \quad \frac{\omega}{T_d} = 0 \tag{E.9}$$

and

$$\frac{E}{V_{in}} = 0 \quad , \quad \frac{E}{T_d} = 0 \tag{E.10}$$

The advantage of PI control is therefore, a zero steady state error to step inputs, including disturbance inputs. An added advantage is the zero steady state disturbance response.

Matlab was used to plot the analog PI closed loop response for the parameters chosen in Chapter 4. The closed loop velocity and error responses are plotted in Figure E-2 for $G = 10$ and $\tau = 0.146$. The root locus is also shown.

The root locus shows that for low gains, the roots are on the negative real axis.

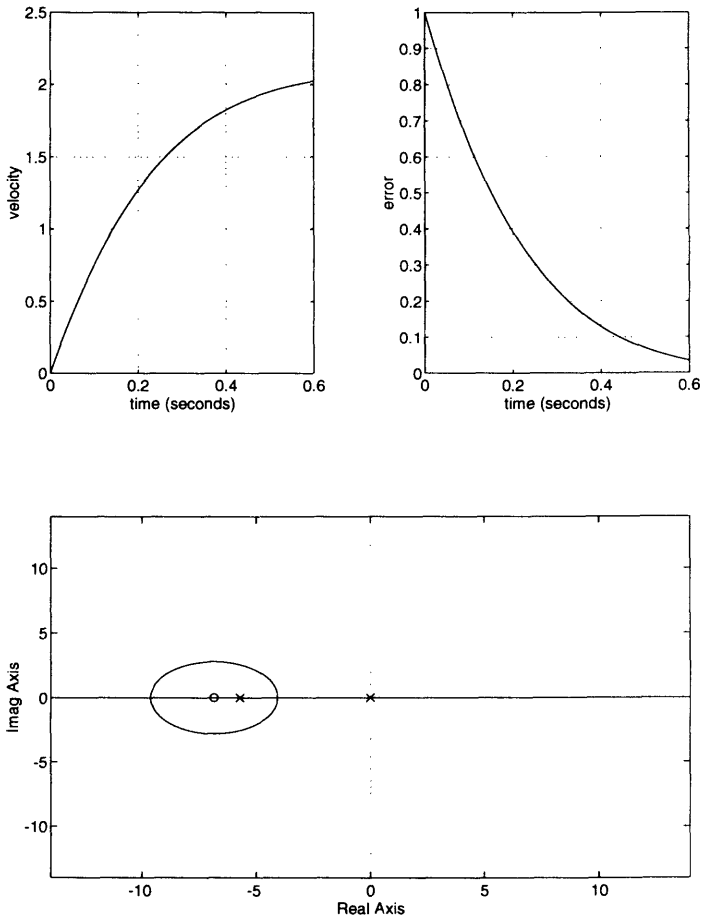

Figure E-2: Speed Response, Error Responses and Root Locus.

If the gain is increased, the poles become imaginary. The step response shows no overshoot and takes about 0.6 seconds to settle, which corresponds to a closed loop dominant pole at $s = -7.67$.

E.2 Digital PI Control

The ZOH equivalent model for the plant model presented in Chapter 4 was given by Equation 4.12, which can be rewritten as

$$G_1(z) = \frac{K_{am}}{B}(1 - \exp^{-\frac{B}{J}T_s})\frac{z^{-1}}{1 - \exp^{-\frac{B}{J}T_s}z^{-1}} \tag{E.11}$$

In the Matlab simulations ran in Appendix B.4, this plant model was used to update the angular velocity variable, i.e.,

$$G_1(z) = \frac{\omega(z)}{vpi(z)} \tag{E.12}$$

Thus ω was updated using

$$\omega[n+1] = A * vpi[n] + E * \omega[n] \tag{E.13}$$

where

$$A = \frac{K_{am}}{B}(1 - \exp^{-T_s/\tau_f}) \tag{E.14}$$

and

$$E = \exp^{-T_s/\tau_f} \tag{E.15}$$

The closed loop characteristic equation can be used to place the poles of the closed loop system. The characteristic equation is given by

$$z^2+[\frac{K_{amT}}{B}h_p(1-\exp^{-T_s/\tau_f})-1-\exp^{-T_s/\tau_f}]z+\frac{K_{amT}}{B}(h_i-h_p)(1-\exp^{-T_s/\tau_f})+\exp^{-T_s/\tau_f} \tag{E.16}$$

Computing the poles using Equation E.16 is time consuming, hence Matlab was used to quickly calculate the roots and determine the continuous time characteristics of the system. To calculate the settling time, natural frequency, and damping ratio, once the z-plane poles have been found, we use the fact that [7]

$$z = e^{sT_s}\,|_{s_1,s_2} \tag{E.17}$$

The z-plane poles can be written in polar form

$$z = re^{\pm j\theta} \tag{E.18}$$

The damping ratio and natural frequency can be derived from Equation E.17 and E.18 to yield

$$\zeta = \frac{-\ln r}{\sqrt{\ln^2 r + \theta^2}} \tag{E.19}$$

$$\omega_n = \frac{1}{T_s}\sqrt{\ln^2 r + \theta^2} \tag{E.20}$$

The settling time can be estimated from the real part of the s-plane poles. The corresponding time constant τ is given by

$$\tau = \frac{-T_s}{\ln r} \tag{E.21}$$

The settling time is about $4.6/\tau$ [7].
Thus using Matlab, the expected system response can be easily calculated. The

results, for the values eventually chosen, using the Matlab function given in Appendix B.2 and B.3, were $\tau = 0.1332$, $z = 0.9963 \pm .0001j$, $r = 0.9965$.

E.3 Tachometer Calculations

The speed measurement was explained in Section 4.1.5. This appendix details the design process, explaining the choice of gain and implementation in further detail.

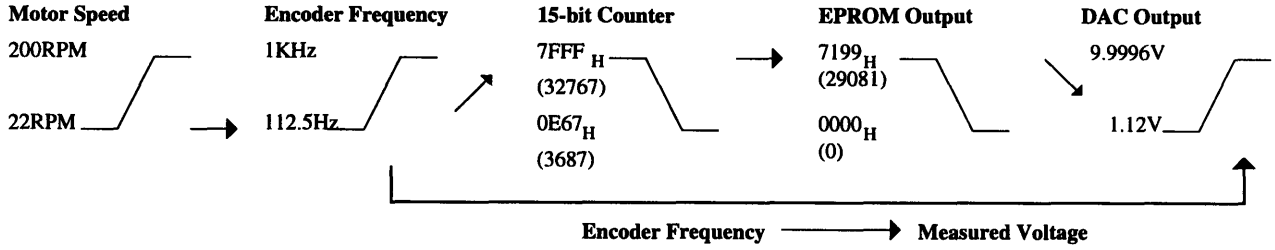

Figure E-3: Ranges at Stages of Tachometer Circuit

Figure E-3 shows signals at various stages of the tachometer circuit as well as their ranges. The motor speed velocity has the desired speed range from about 20 or 22 to 200RPM. This corresponds to a frequency range from the encoder output of 112 Hz to 1 KHz. The conversion is given by dividing RPM by 60 to get revolutions per second, then multiplying by 300 (number of pulses per revolution) to get the encoder "speed" in Hz. The encoder interval is represented by a 15-bit counter. A longer interval corresponds to a bigger count and a slower motor. However, the count cannot exceed 32767, the maximum for a 15 bit counter. By following the arrows in Figure E-3, we can trace the path a 112.5 Hz, which is the slowest one of interest. Speeds below this value are considered zero. The 112.5 Hz encoder pulse causes the counter to reach its maximum value of 32767. Since the counter value is inversely related to the frequency of the input voltage, it is necessary to invert this reading so as to get an output reading proportional to the frequency input. Defining a *newcount* can perform this inversion.

$$newcount = [32767 - B/count] \tag{E.22}$$

The minimum and maximum values of count are:

$$min\_count = \frac{f_{clk}}{f_{max}} = \frac{3.6864MHz}{1KHz} = 3687 \tag{E.23}$$

$$max\_count = \frac{f_{clk}}{f_{min}} = \frac{3.6864MHz}{112.5Hz} = 32767 \tag{E.24}$$

Actually, Equation E.24 is used to set f_{min}, not the other way round. f_{min} Thus, the entire range of the counter is used. After the 112.5 Hz signal passes through the DAC, the measured voltage is found from Equation 4.24

$$measured\ \ voltage = [B/count]305\mu V \tag{E.25}$$

The next step is to calculate the value of B. To use the full range of the DAC it is desired that the maximum frequency, corresponding to the smallest count, should yield the maximum positive DAC voltage. Thus

$$\frac{(B)(305\mu V)}{f_{clk}} f_{max} = 9.993539\ volts \tag{E.26}$$

This leads to $B = 120,792,269$. In Figure E-3, then, the 112.5 Hz signal passes through the EPROM coming out as a 29081 value input to the DAC. This minimum frequency yields a DAC voltage of 1.12 volts, which could be calculated from Equation E.25 by replacing count by f_{clk}/f_{max}. If the drum spins at a speed slower than 22RPM, the encoder frequency will be slower than 112.5 Hz. The counter will count up to 32767 and remain at 32767 (see Appendix C.2), until the speed increase past the minimum of 112.5 Hz. Thus the DAC voltage can never fall below 1.12 volts.

Appendix F

Circuit Schematics

This appendix presents the schematics for the circuits constructed for the spinning machine.

This is the most complete text of the thesis available. The following page(s) were not included in the copy of the thesis deposited in the Institute Archives by the author:

p. 108

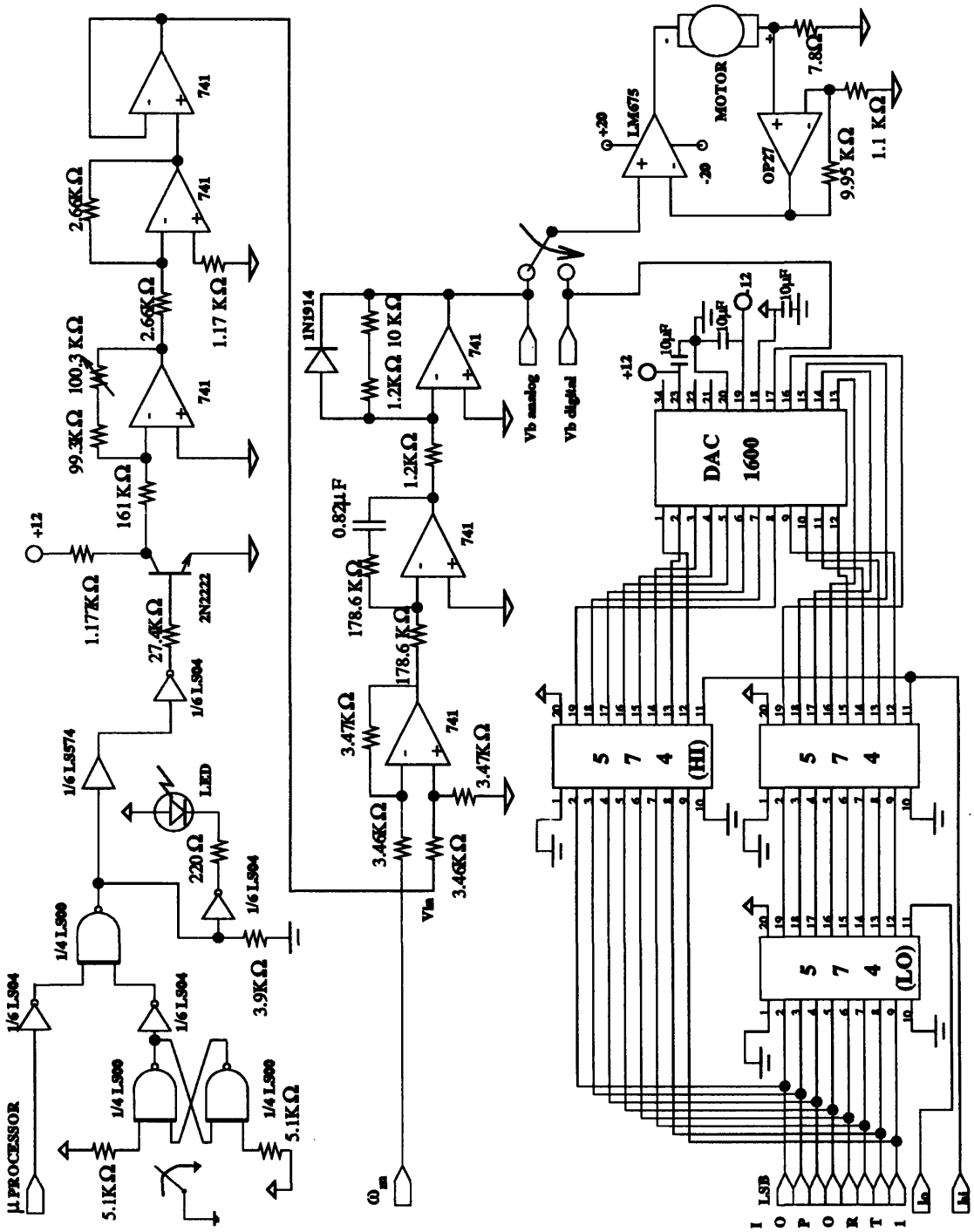

Figure F-2: Motor Voltage Command and Analog PI

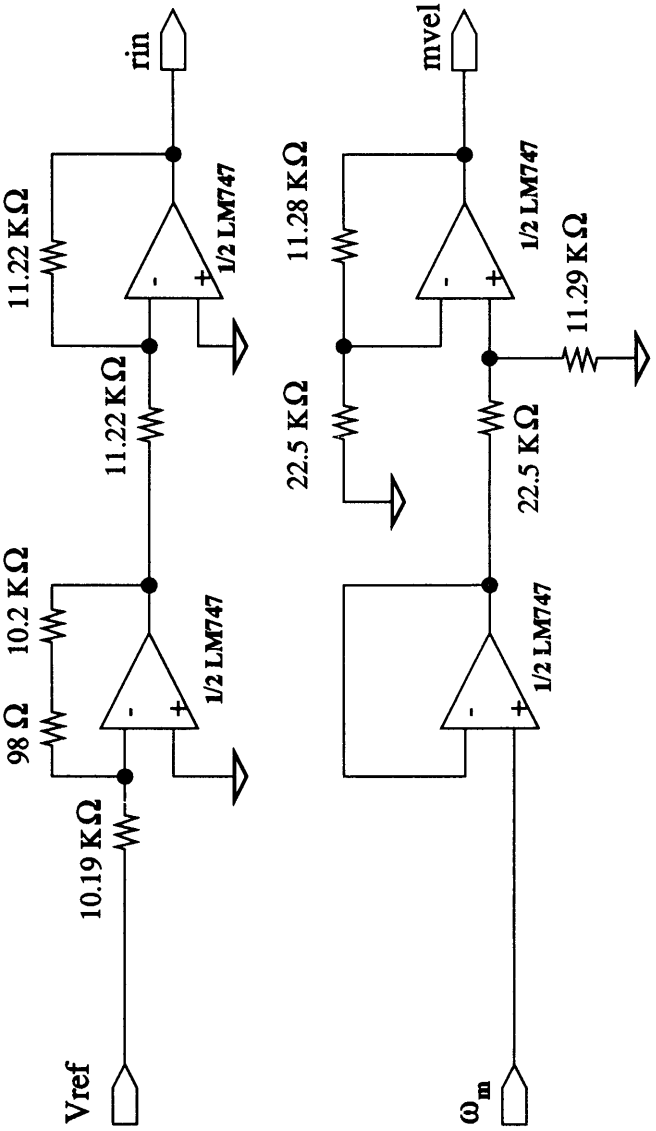

Figure F-3: A/D Channel Input Voltage Circuitry

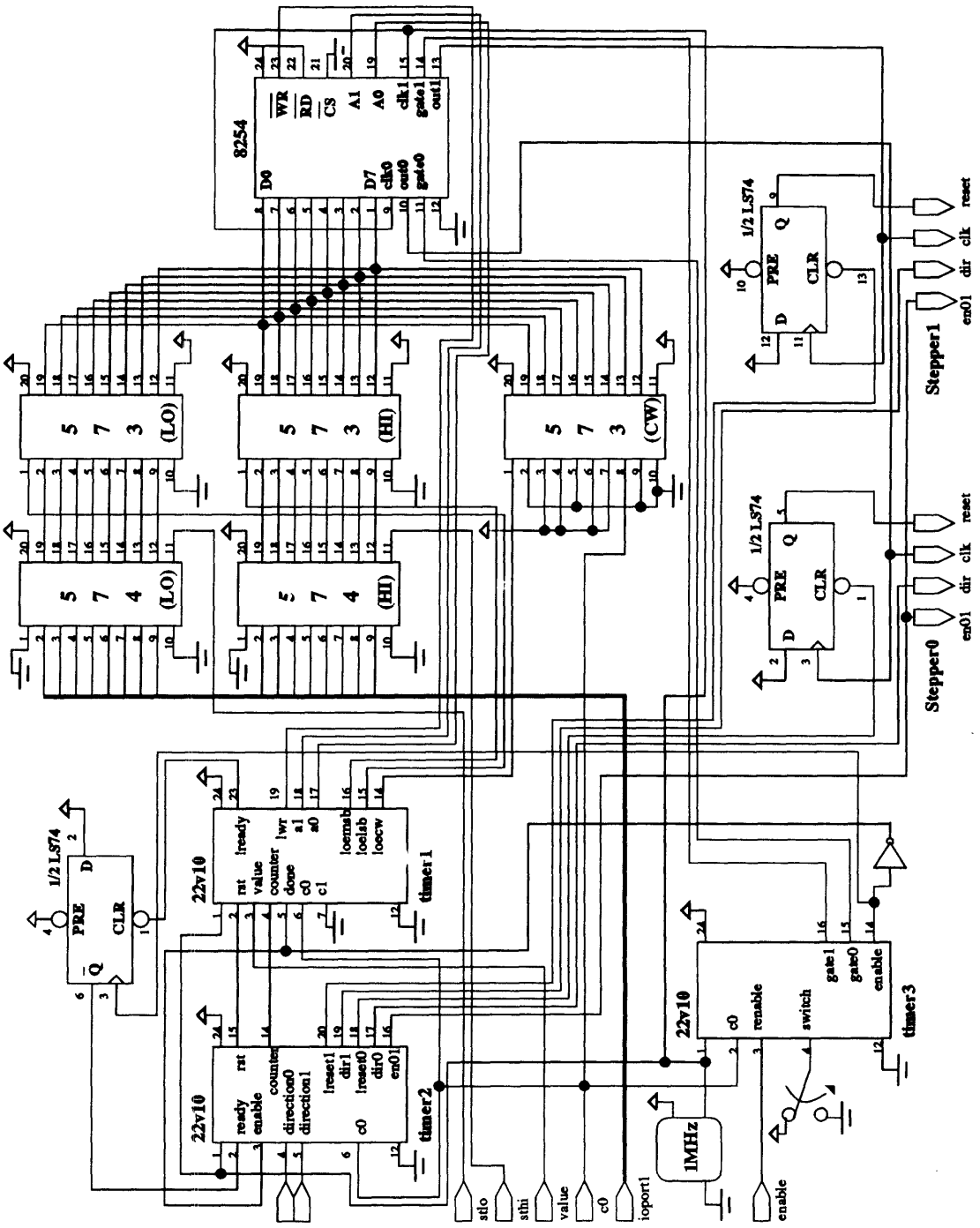

Figure F-4: Stepper Rate and Signals Generator

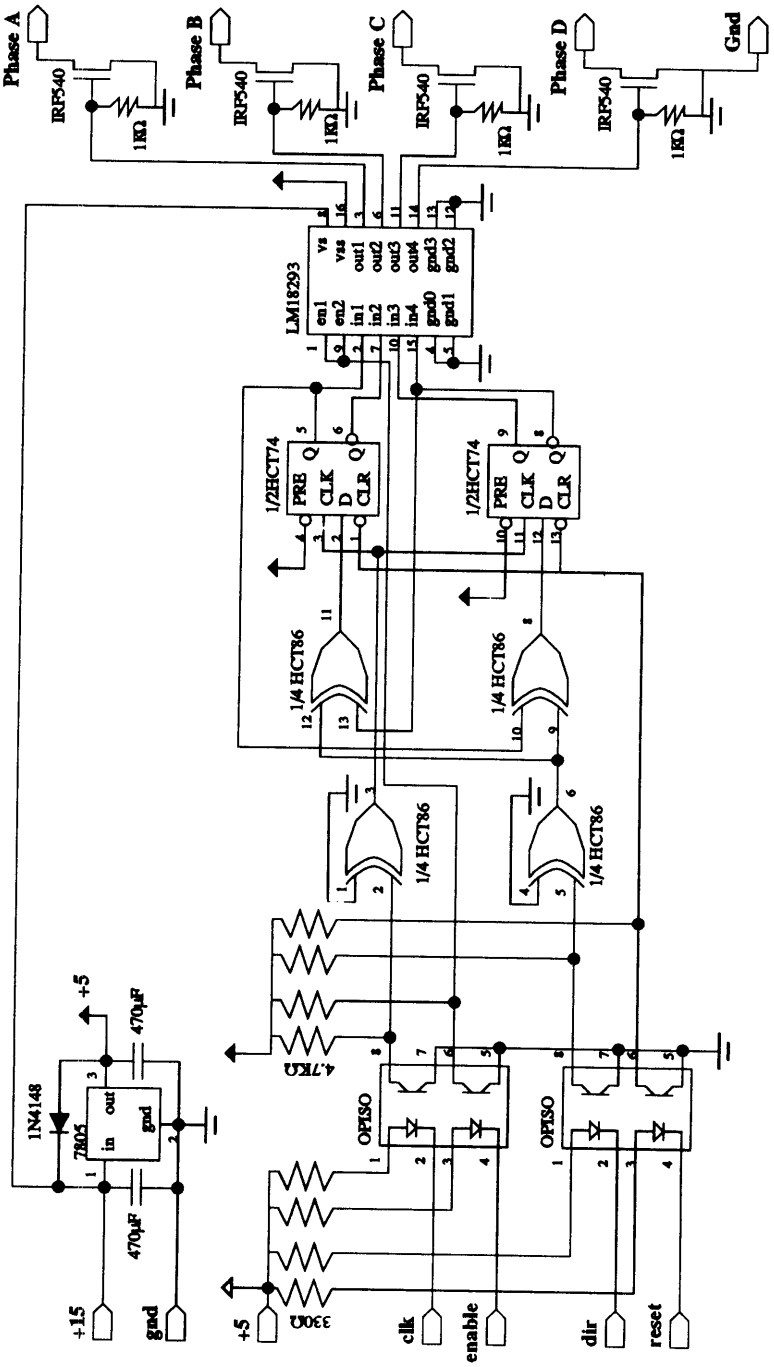

Figure F-5: Stepper Phase Logic Sequencer and Amplifier

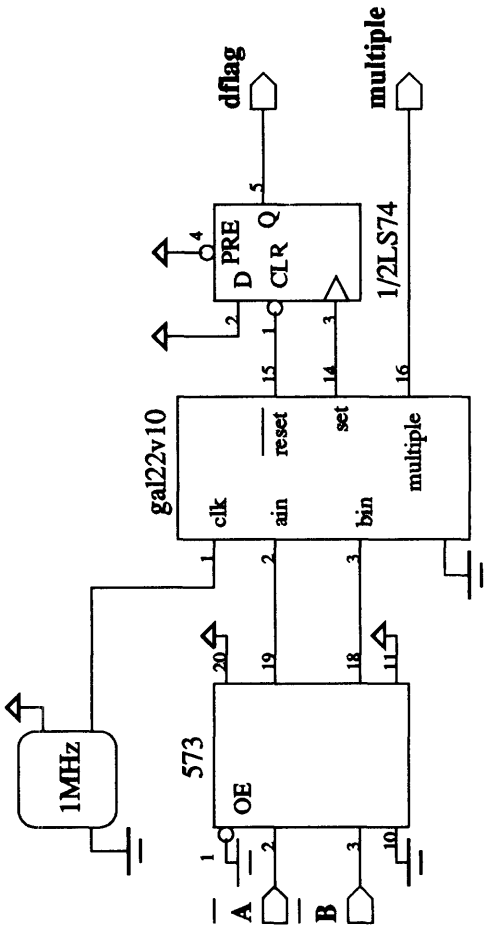

Figure F-6: Drum Rotation Direction Circuitry

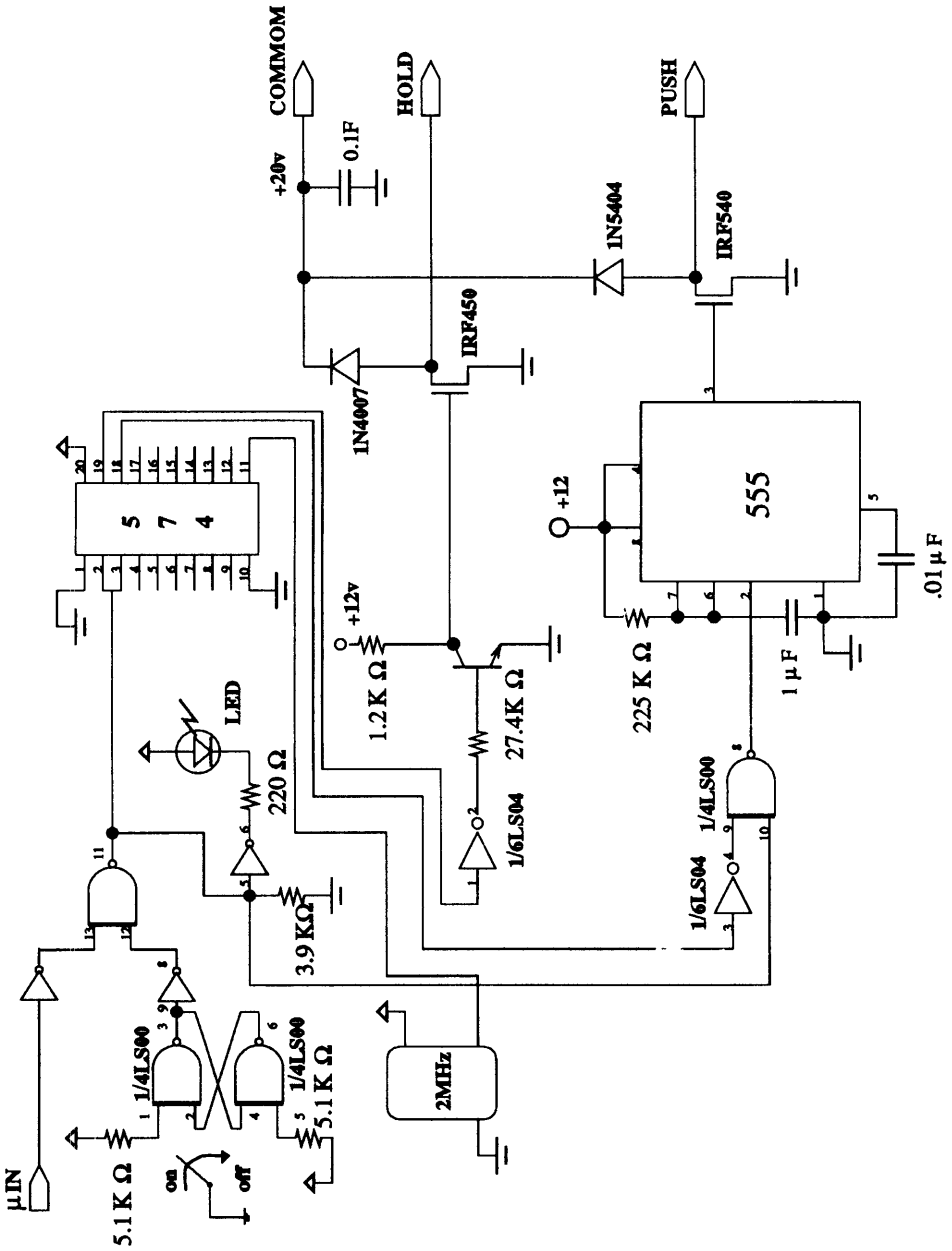

Figure F-7: Solenoid Activation Circuitry

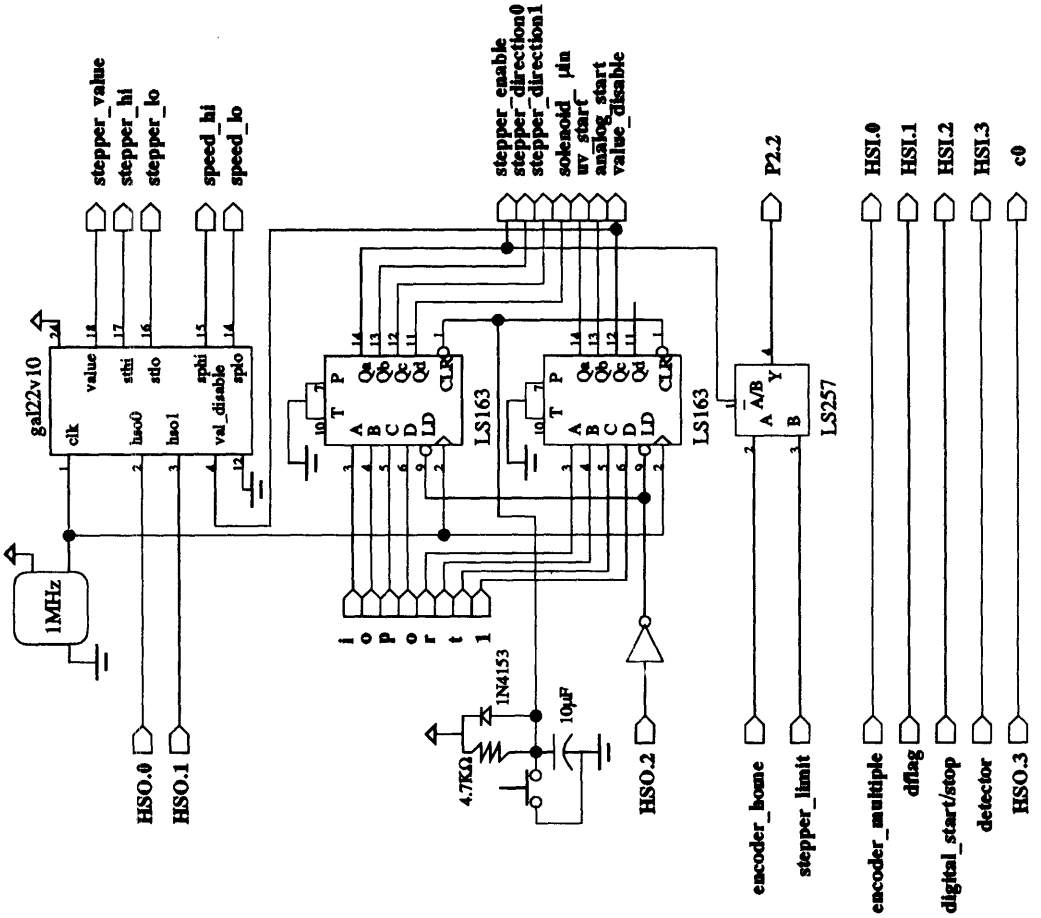

Figure F-8: Handshaking Circuit for I/O Lines

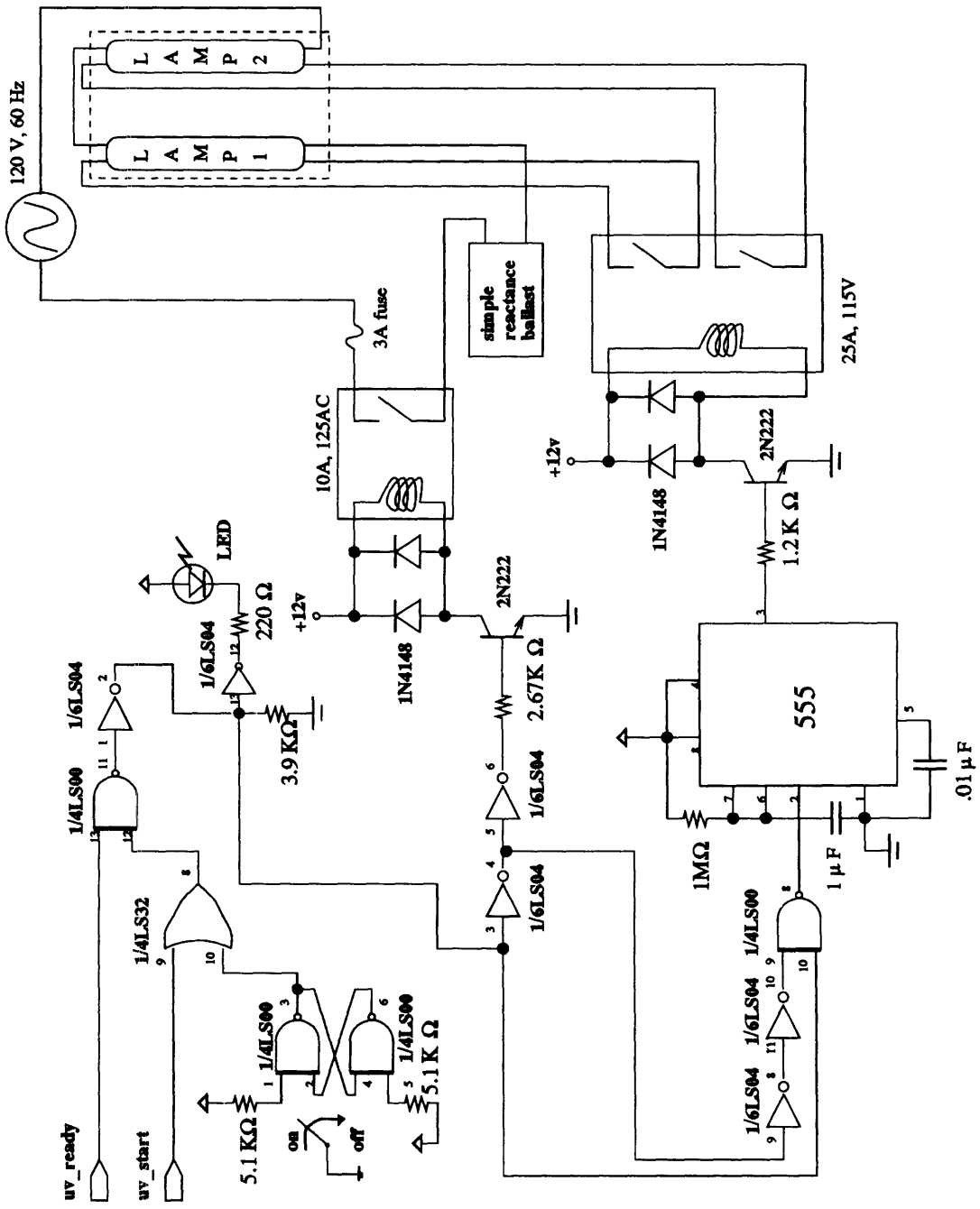

Figure F-9: UV Lamp Circuit

Bibliography

[1] Techno a DSG Company. *Linear Motion Products Catalog.* New Hyde Park, NY, USA, 1989. Part # HL31SBM501060005.

[2] T. Tanaka A. Suzuki. Phase transition in polymer gels induced by visible light. *Nature*, 346(6282), July 1990.

[3] Burr-Brown. *Integrated Circuits Data Book Supplement Volume 33C.* Tuscon, AZ, USA, 1992.

[4] Electrical Engineering Dept. of Aeronautics/Astronautics. 6.302 class notes: Feedback systems. Technical report, Massachusetts Institute of Technology, 1993.

[5] Electrical Engineering Dept. of Aeronautics/Astronautics. 6.302 lab 1: Servo motor modelling. Technical report, Massachusetts Institute of Technology, 1993.

[6] Justin Foreman. The design of a stepper motor controller. MIT Laboratory For Electromagnetic and Electronic Systems, 1993.

[7] Michael L. Workman Gene F. Franklin, J. David Powell. *Digital Control of Dynamic Systems.* Addison-Wesley Publishing Company Inc., Reading, MA, USA, 1990.

[8] Charles Ho. Simplest driver yet for stepper motors. *Electronic Design*, December 1991.

[9] Intel. *Microprocessor and Peripheral Handbook.* Mt. Prospect, IL, USA, 1987.

[10] Intel. *8X196KC/KD User's Manual.* Mt. Prospect, IL, USA, 1992.

[11] Intel. *EV80C196KC Microcontroller Evaluation Board User's Manual.* Mt. Prospect, IL, USA, 1992.

[12] Ahmed Mitwalli. *A Digital Controller for a Unity Power Factor Converter.* Master's thesis, Massachusetts Institute of Technology, June 1991.

[13] Ahmed Mitwalli. 2.165: Multijoint manipulator design project. Technical report, Massachusetts Institute of Technology, 1993.

[14] Micro Mo. *Miniature Drive Systems.* St. Petersburg, FL, USA, 1993. Part #'s 2842S024C Motor, Gearhead Series 23/1.

[15] Allen Mottershead. *Introduction to Electricity and Electronics.* John Wiley and Sons Inc., New York, NY, USA, 1986.

[16] Y. Osada and S.B. Ross-Murphy. Intelligent gels. *Scientific American*, May 1993.

[17] Synchro-Start Products. *Dual Coil D.C. Solenoids.* Niles, IL, USA, 1992. Part # 1757ES.

[18] L. E. Unnewehr S. A. Nasar. *Electromechanics and Electric Machines.* John Wiley and Sons Inc., New York, NY, USA, 1983.

[19] A. Mitwalli S.B. Leeb T. Tanaka U. Sinha. *Polymer Gel Actuators.* To be published in UPEC, September 1994.

[20] Gordon R. Slemon. *Electric Machines.* Addison-Wesley Publishing Company Inc., Reading, MA, USA, 1980.

[21] Toyoichi Tanaka Steven B. Leeb. Gel polymer actuators. MIT, LEES and CMSE, November 1993.

[22] Sterling Instrument Stock Drive Products. *Handbook of Design Components.* Stock Drive Products, Sterling Instrument, New Hyde Park, NY, USA, 1992.

[23] M. Hasebe T. Osada. Electrically actuated mechanochemical devices using polyelectrolyte gels. *Chemistry Letters*, 1985.

[24] Toyoichi Tanaka. Gels. *Scientific American*, 244(1), January 1981.

[25] T. Tanaka I. Nishio S. Sun S Ueno-Nisho. Collapse of gels in an electric field. *Science*, 218(29), October 1982.