Contents

List of Figures

A special mention...

We would like to gratefully acknowledge the contribution of the Sarai/CSDS foundation, New Delhi.

Our project was selected for their FLOSS (Free/Libre Open Source Software Initiatives) Independent Fellowship Programme.

The fellowship entitled us to a grant which was instrumental in covering the cost of our project.

A brief introduction to Sarai and their activities is given below, in their very own words.

The Sarai programme of the Centre of the Study of Developing Societies, is committed to developing a ***public architecture for creating knowledge and creative communities.*** *In keeping with this commitment, we seek to develop a* ***distributed network*** *of scholars, writers, practitioners and programmers who are motivated to make the materials and outcome of research available for public access and circulation, with the understanding that an imaginative engagement with social experience will be fostered by a sharing of information, ideas, research materials and resources. We see our system of independent fellowships as a resource that will be built on by many people working either individually or in groups, but with a sense of collective endeavor and public purpose.*

Further information about the Sarai initiative can be found at

`http://www.sarai.net`

Chapter 1

Introduction

A SCARA robot is a 4 DOF horizontal-jointed robot. SCARA stands for Selective Compliance Assembly Robotic Arm. These robots are generally used for vertical assembly and other operations in parallel planes. Selective compliance is a characteristic feature which is extremely useful in assembly operations requiring insertion of objects into holes (e.g. pegs or screws). The SCARA is extremely stiff in the vertical direction but has some lateral "give" (i.e. compliance), thereby facilitating the insertion process. Commercial SCARA robots include the Adept One robot, the IBM 7545 robot, the Intelledex 440 robot and the Rhino SCARA robot.

This report presents detail of the design and implementation of a SCARA robot which we have built as our final year project. However, before plunging into the technical detail, it will be worth the while to examine some preliminary issues.

1.1 Our design objectives

The goal of our project was to develop an industrial strength SCARA with an optimum balance of economy and performance. Robotics as an application oriented technology has yet to make major inroads in India. We believe that with the availability of low cost, flexible automation this scenario is bound to change for the better.

Although performance is critical, we have, at times assigned a greater priority to the economic constraints. This is consistent with our belief that once the control theory is fully developed, performance varies directly with the quality of hardware components used. Thus, we have concentrated on developing the control aspects of the robot to the maximum extent. This approach, while agreeing excellently with our monetary limits, has resulted in an eminently usable SCARA robot with a highly modular design. To give an immediate example, we have used stepping motors in the joint actuators, although the software deals equally well with servo, brush-less DC or even other 3 phase motors with assorted electronics.

All the controls have been implemented using a standard personal computer. A desktop PC is an incredibly powerful (and an incredibly under-utilized) machine and a significant design goal was to take full advantage of this preexisting power. The parallel port interface would be exclusively utilized.

Another issue was standardization. We have exclusively used free[1] software for this project and we conform to the relevant open standards as closely as possible. Continuing in the same vein, we did not intend to write a new programming language for our robot. The EIA-RS274D commonly known as G-codes (or Gerber codes) are already an established standard in the CNC machine tool industry. Naturally, an important design goal was to ensure that the robot could be programmed using G-codes. This implies *continuous path* control.

Finally, the SCARA configuration, although ideally suited for assembly tasks, need not be limited to it. Our robot should work with a variety of end effectors and be able to carry out many other tasks within its configuration limits.

1.2 What this report is

This report details the complete design and implementation of our SCARA robot. We present everything from the design considerations to the hardware used, the choices we had to make along the way, the problems we faced and how we overcame them. Relevant figures, full-length diagrams, algorithms and mathematics have been included wherever necessary. This report should act as a complete reference and guide to understanding the constructional and functional aspects of our robot.

1.3 What this report is not

This report is not a general purpose reference on robotics. We have not attempted to explain every single detail here, nor are basic concepts covered. We (justly) assume that the reader is already familiar with fundamentals of robotics. Hence, only those aspects that are pertinent to our robot are dealt with. We would direct the new and eager reader to references[1, 2, 3, 4] since they cover the basic theory far better than we can hope to.

[1]When we say "free", we are referring to *freedom*, not price.

Chapter 2

Design Considerations

In this section, we present some of the preliminary thought that went ahead of the actual design.

2.1 Performance requirements of an industrial robot

Robots are designed to be highly accurate, precise and flexible machines. Robots in general and SCARA robots in particular, are used as replacements for human operators. This can be for a variety of reasons, but an important feature of using robots is that they almost always do the job better than a human operator. In order to achieve this important goal, the robot needs to conform to certain minimum standards of performance. To give an idea, some of the specifications of the Adept One XL SCARA robot are described below. This particular robot was chosen because it enjoys wide commercial success.

Reach 800mm

Joint Range

- Joint 1: $\pm 150^o$
- Joint 2: $\pm 140^o$
- Joint 3: 203mm
- Joint 4: $\pm 270^o$

Maximum Joint Speed

- Joint 1: 650^o / sec

- Joint 2: 920^o / sec
- Joint 3: 1,200mm / sec
- Joint 4: 3300^o / sec

Repeatability

- (x,y): ± 0.025 mm
- (z): ± 0.038 mm
- Theta: $\pm 0.05^o$

Maximum Payload 12 kg.

The reader will now be able to better appreciate the following sections.

2.2 Mechanical hardware

The SCARA is a standard configuration among robots. Furthermore we did not design the robot strictly for a specific application. Hence at the preliminary phase, we only had the following points in mind:

- The robot should be as light and rigid as possible, within our economic constraints.
- Load capacity of 3 kg.
- A reach of 0.5 m.
- A control resolution of 0.5 mm.
- Maximum tip velocity of .
- Maximum tip acceleration of .

Also, we intend the robot to be used for more than just assembly tasks. This necessitates analog control of the prismatic axis. Hence, against the industrial norm of using pneumatic actuators, we opted for a ball-screw as the joint actuator.

2.3 Electronics hardware

The electronics involved depends totally on the type of motors being used. In order to cut costs, we decided to use stepper motors in the joint actuators.

The major part of the electronics hardware therefore involves the design of the appropriate stepper motor control circuits. Apart from the control software used, this is the single most important factor, which can drastically affect the performance of the robot. Here, we concentrate exclusively on the stepper motor drivers.

Some of the initial choices faced were

- The type of motor: Unipolar or Bipolar.
- The drive principle: L/R or PWM
- The mode of operation: Half-step or Full-step.
- Torque – speed characteristics.

Further details are given in the subsequent subsections.

2.3.1 Unipolar or Bipolar

A stepper motor moves one step when the direction of current flow in the field coil(s) changes, reversing the magnetic field in the stator poles. The difference between unipolar and bipolar motors lies in the way in which this reversal is achieved.

A bipolar motor has one field coil and two change over switches that are switched in the opposite direction. A unipolar motor has two separate field coils and a single change over switch. See fig. 2.1

The advantage of the bipolar circuit is that there is only one winding, with a good bulk factor (low winding resistance). The main disadvantages are the two changeover switches because in this case more semiconductors are needed.

Unipolar circuits need only one changeover switch. The enormous disadvantage, however, is that a double bifilar winding is required. This means that at a specific bulk factor the wire is thinner and the resistance is much higher.

Although the bipolar circuit is more complicated, we opted for it due to two overriding reasons

1. The bipolar circuit can drive both, bipolar as well as unipolar motors.
2. The bipolar circuit delivers more torque.

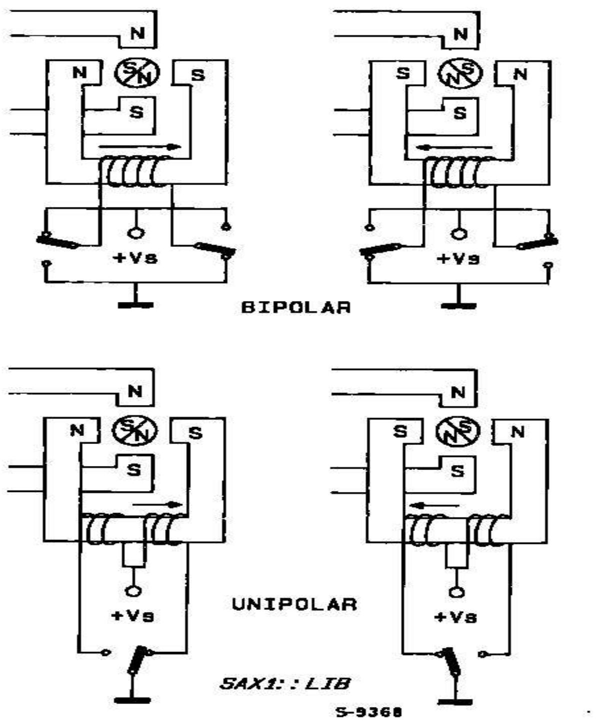


Figure 2.1: Bipolar and Unipolar configurations

The second point needs further explanation. The torque of a stepper motor is proportional to the magnetic field intensity of the stator windings. It may be increased only by adding more windings or by increasing the current.

A natural limit against any current increase is the danger of saturating the iron core. Much more important is the maximum temperature rise of the motor, due to the power loss in the stator windings. This is another advantage of the bipolar circuit, which, compared to the unipolar systems, has only half of the copper resistance because of the double cross-section of the wire. The winding current may be increased by a factor of $\sqrt{2}$ and this produces a directly proportional effect on the torque. At their power loss limit, bipolar motors thus deliver about 40% more torque (fig. 2.2) than unipolar motors built on the same frame.

2.3.2 L/R or PWM

The speed of a stepper motor depends on the rate at which the coils are turned on and off, and is termed as the "step rate". The maximum step-rate and hence the maximum speed depends on the inductance of the stator coils. Fig. 2.3 shows the equivalent circuit of a stator winding and the relation between current rise time and winding inductance. With higher inductance, it takes longer time for the rated current to build up

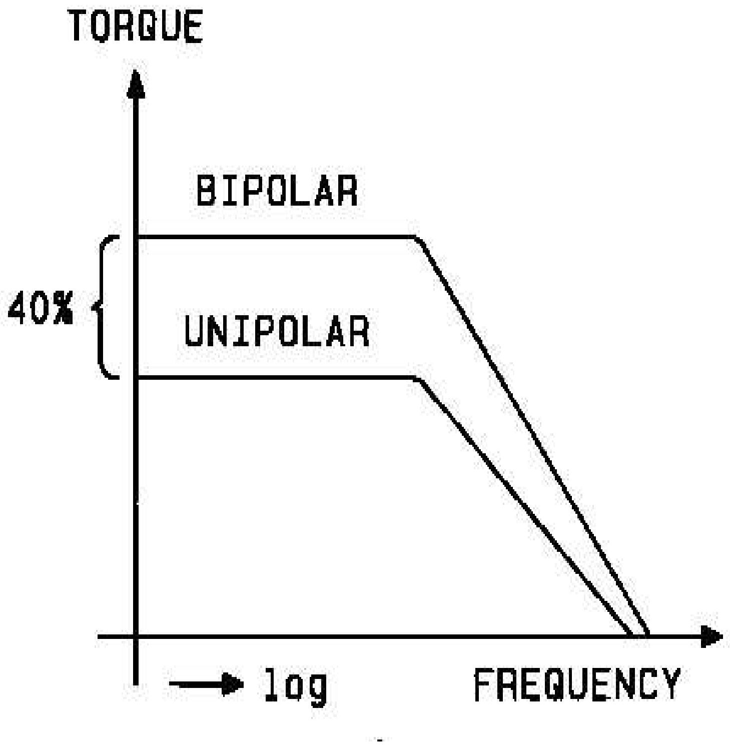


Figure 2.2: Torque in bipolar and unipolar systems

in a winding.

If the time between two step commands is lower than the current build up time, then the motor misses a step. Thus, motor current may never reach full-rated value, especially at high speeds, unless the voltage (Vs) across the terminals is high. In the simplest L/R drive (fig. 2.4), a transistor sequentially activates the windings to drive the motor. This type of drive performs poorly because the supply voltage must be low so that the steady state current is not excessive. As a result, the average winding current — and hence the torque — is very low at high motor drive speed. These problems can be overcome by introducing a series resistance, thereby increasing the overall value by a factor of four — giving an L/4R ratio — and also by increasing the supply voltage (fig. 2.4). Although this approach improves torque at high step rates, it is inefficient, because the series resistance constitutes a substantial waste of power. Thus, the L/R method, although conceptually simple, is not an elegant technique. It was thus rejected in favor of the PWM technique.

A constant current pulse-width-modulation (PWM) is an elegant solution to improve the motor's efficiency and torque speed characteristics. This method comprises of introducing a feedback loop to control the winding current. Now, linear constant current control is possible, but is rarely used because of high losses in the power stage.

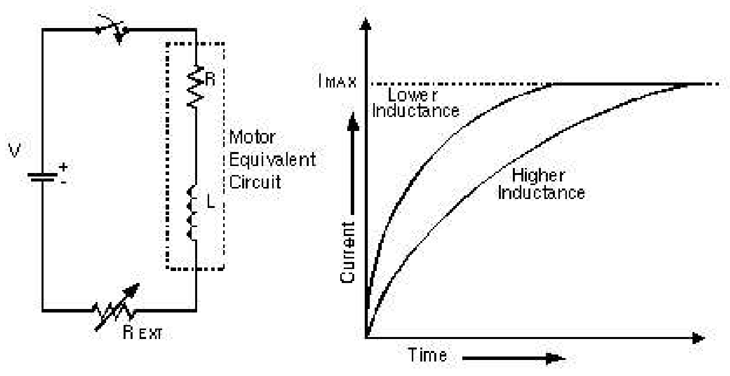


Figure 2.3: Current buildup in a motor winding

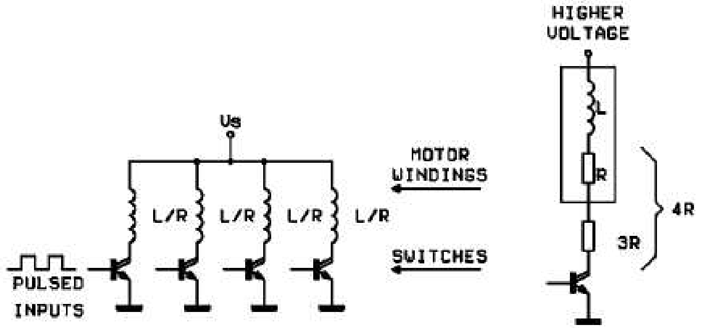


Figure 2.4: L/R drive

However, with the PWM scheme — a chopper circuit — not only is the L/R time constant problem solved, but power dissipation is reduced too. The conceptual diagram of a PWM circuit is shown in fig. 2.5.

2.3.3 Half-step or Full-step

Half-stepping is a technique by which finer positioning can be obtained from a motor. Consider fig. 2.6. When current flows in only one winding, the rotor aligns with the stator poles in positions 0,1,2 and 3. When current flows in both windings, the rotor aligns itself between two stator poles in positions $\frac{1}{2}$, $1\frac{1}{2}$, $2\frac{1}{2}$ and $3\frac{1}{2}$. Thus, compared to full step, the number of steps are doubled.

An essential advantage of a stepper motor operating at half-step conditions is that its position resolution is increased by a factor of two. From a 1.8 degree motor, we obtain 0.9 degrees, which means 400 steps per revolution.

This is not the only reason the prefer half-step mode. It is often essential to avoid resonance in the motor. The rotor of the motor and the changing magnetic field of the stator form an equivalent spring-mass system, which may be stimulated to vibrate. During resonance, the motor torque drops to zero and the motor may lose position

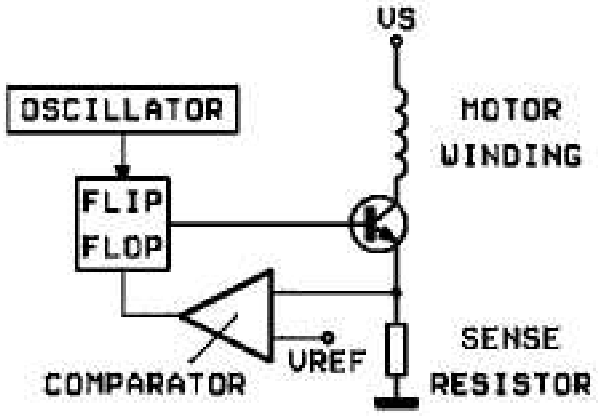


Figure 2.5: PWM control

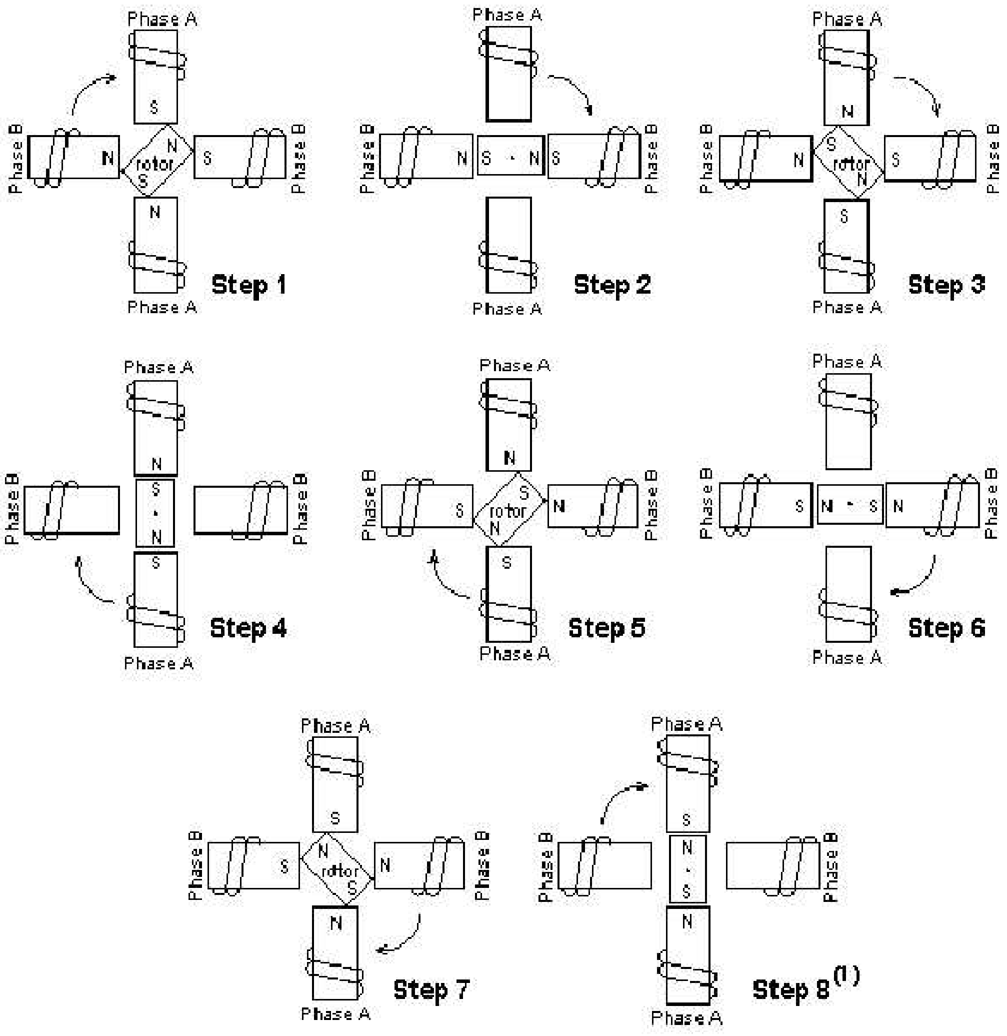


Figure 2.6: Half-step mode

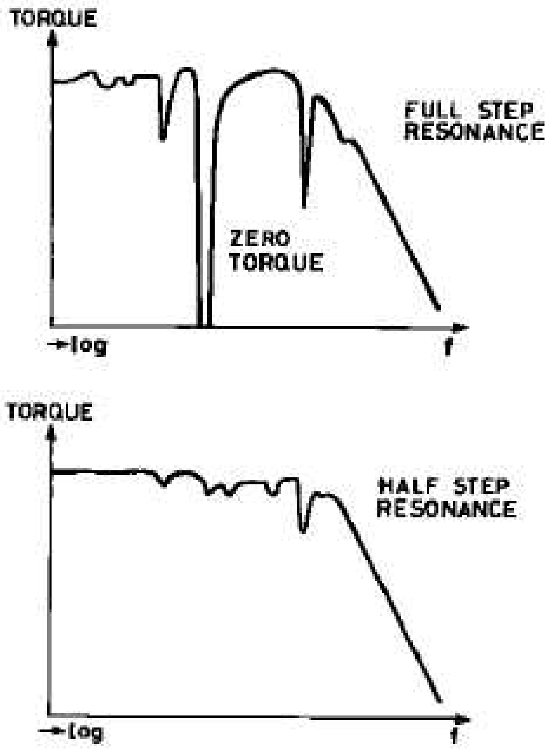


Figure 2.7: Resonance

completely. In practice, the load might deaden this system, but only if there is sufficient frictional force. Half-step mode has decreased chances of resonance, due to the fact that the rotation is only half as long resulting in less stimulation. Fig. 2.7 shows the response curves in half- and full-step modes.

However, the half-step mode has two major disadvantages.

- Twice as many clock pulses are needed, as compared to a full-step system.
- In the half-step position, the motor has only about half the torque of the full-step.

Despite these disadvantages, the half-step mode was chosen, for the sake of better positional accuracy. The justification is that modern computers are fast enough to generate high frequency clock pulses. Also, with proper mechanical gearing, the lost torque can be recovered. Of course, higher operating speeds will be needed, which is actually a help because at high speeds, the loss of torque decreases due to averaging effects.

2.3.4 Torque – Speed characteristics

An important point to be realized when working with steppers is that the torque is a function of speed, and as a general rule, the torque *decreases* as the speed increases.

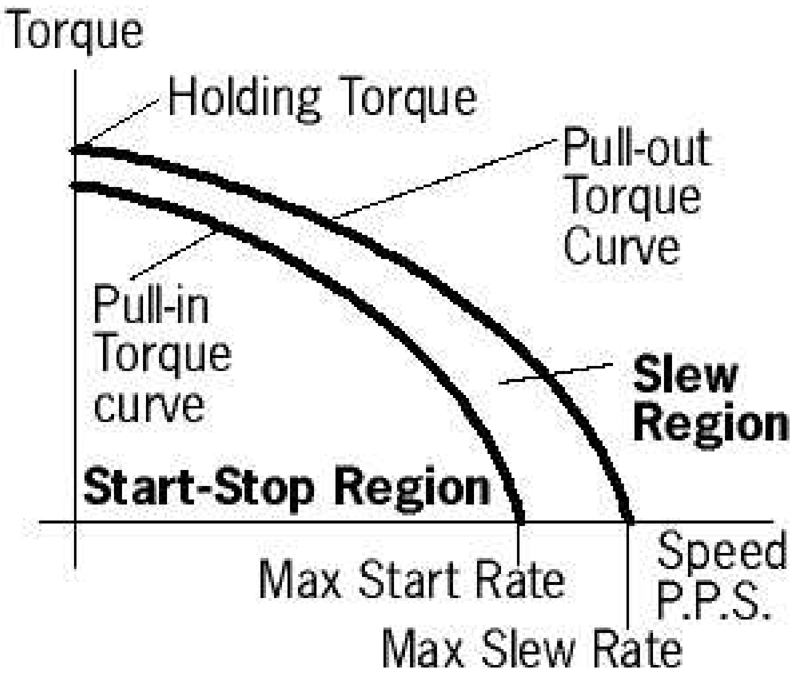


Figure 2.8: Torque – Speed characteristics

Fig. 2.8 shows the torque-speed relation for a typical stepper motor.

"Pull-in" torque is the maximum load torque at which the motor can start or stop without mis-stepping. "Pull-out" torque is the torque available when the motor is continuously accelerated to the operating speed. Max. start rate is the step rate at which the motor can start instantaneously at no load, without mis-stepping.

2.4 Software and control

2.4.1 The need for "real time"

A system is said to be real time when the correctness of its output depends not only on its logical result, but also on the time at which the result is obtained. Thus, a real time operating system is a system which, apart from executing programs, must also see to it that those programs are executed at and within a specified time frame.

Consider that you are running a motor connected to your computer, through a program which generates a steady stream of pulses. Now, if you start another program on the same computer, you will observe that the motor no longer runs smoothly. The motor might even stop running. Why does this happen?

The answer has to do with the way in which the operating system runs programs.

When the operating system is multitasking, it runs each program for a small period of time. This is called a time slice. Whenever a program exceeds its time slice, it is put to sleep and another program is run in its place. Now, if the program which is put to sleep is the one which drives the motor, how will the motor run?

This is just one example of a number of problems that can occur when a program is running. As another example, consider what happens when a program needs to perform an action at time T. The program goes to sleep after telling the operating system to wake it up at the required time. Now if, at time T, another program is executing a system call, or a higher priority process is being executed, our program will not be woken up. If the program controls a machine, that machine might have gone out of control by the time the program is woken up and executed.

Thus, we see that a normal operating system, which is optimized to give good average performance is not at all useful to control real-time tasks.

2.4.2 Choosing a real time OS

The only solution to the above problem is to use a *real time* OS, which guarantees the timing of the processes under its control. Examples of real time operating systems are QNX, RTAI/Linux, RTLinux, VxWorks etc. The last one is a commercial OS which requires two computers to achieve real time capabilities. These factors naturally put it immediately out of consideration. RTAI and RTLinux are real time extensions to the Linux kernel. When patched against the kernel, they ensure real time performance. A similar real time patch for the Microsoft(R) Windows NT(TM) is available from Radisys Inc. However, both products are

1. Commercial
2. Closed Source
3. Costly

Point 2 is of exceptional importance. The closed nature of these products meant that we would be unable to change the system behavior to our liking, nor would we be able to improve it in order to extract better performance. Apart from this, moral, ethical[5] and licensing issues ruled out the possibility of using any proprietary operating system and software.

We are strong supporters of GNU[6] and “Free software”[7]. We also have extensive experience in running and programming GNU/Linux systems in general and the linux[8] kernel in particular. All these factors were decisive in selecting GNU/Linux as the operating system of choice for the project. Since the RTAI and RTLinux API matches closely, it was decided to include support for *both* the real time extensions. Doing this is not as difficult as it sounds, due to the close adherence to standards by both.

2.4.3 Control

Not much is to be said about the control system at this point. The only consideration was that we should be able to achieve

1. Continuous paths in space.
2. Joint-interpolated motion.
3. Circular interpolation.
4. Linear interpolation.
5. Coordinated linear motion.

Coordinated linear motion means that all the axes start and finish their motion segments at the same time. These requirements can largely be handled by the control software. The control strategies are discussed in chapter 5.

Chapter 3

Mechanical hardware design

3.1 Control resolution

For a SCARA robot, the worst case control resolution is at the periphery of the work envelope. This occurs when both the elbow and forearm are in line with each other. Using direct drive with stepping motors, the achievable resolution is found out as follows.

A stepper motor operating in half-step mode is capable of 400 steps per revolution. Hence, it divides the periphery of the work envelope into 400 equal steps. For an elbow length of 0.3m and a forearm of 0.25m, the control resolution, given by equation 3.1 will be 8.639mm.

$$CR = 2\pi(L_1 + L_2)/400 \tag{3.1}$$

To achieve a resolution of 0.5mm, gearing is required. This increases the number of steps into which the periphery can be divided. The gear ratio can be calculated by $8.639/0.5 = 17.27$:1 The closest available standard timing belt ratio is 19.14:1 which is achieved using 2 stage gearing. Each stage consists of a 16 teeth pulley driving a 70 teeth pulley, thus achieving a gear-ratio of 4.375:1. In order to achieve the same resolution for the forearm, 8:1 gearing was selected. This required the length of the elbow to be increased to 0.34m. Recalculating the control resolution with $L_1 = 0.34$ and $L_2 = 0.5$, the control resolution comes out to be 0.484 mm. This value agrees excellently with the value we set out to achieve.

3.2 Material selection

Economy was the prime concern here. The material selected should be low in cost (per kg basis), readily available in the desired shapes and sizes and the procurement time and procurement cost should be low. We initially had a choice of two materials, Aluminum and Mild Steel.

Aluminum provides the necessary rigidity at a low weight. However, using Al meant that we would have to machine the arms from solid billets. This is because, although Al sections are available, they are not *easily* weldable. Al billets of the required size were not available locally and hence procurement costs were very high.

On the other hand, MS is available in a variety of cross-sections, is easily weldable and is more rigid than aluminum. This also allows us to use hollow sections and weld components together to build the arm structure.

Thus MS was exclusively used as the material of choice for the arms.

We chose a rolled steel ball-screw over a ground one, since it is significantly cheaper, yet served our resolution and repeatability requirements. Case hardened and ground shafts were used to support the ball screw against bending.

3.3 Design procedure

The design procedure is essentially iterative in nature. Rather than showing actual calculations, we prefer to elaborate on the design process. This results in a significant loss of clutter in the report, without compromising on clarity.

3.3.1 Design of prismatic axis

We initially started off with a tip load assumption of 3 kg. We had to ensure that the entire assembly was rigid enough to support this load. By “rigid enough”, we mean that the assembly should bear this load and still maintain the control resolution.

The ball-screw design involved initially considering the axial tensile stresses. After this value was obtained, torque required to lift the load was calculated as per the equation

$$T = \frac{d}{2} W \tan(\alpha + \phi) \tag{3.2}$$

where,

T = torque require to lift load W

W = tip load

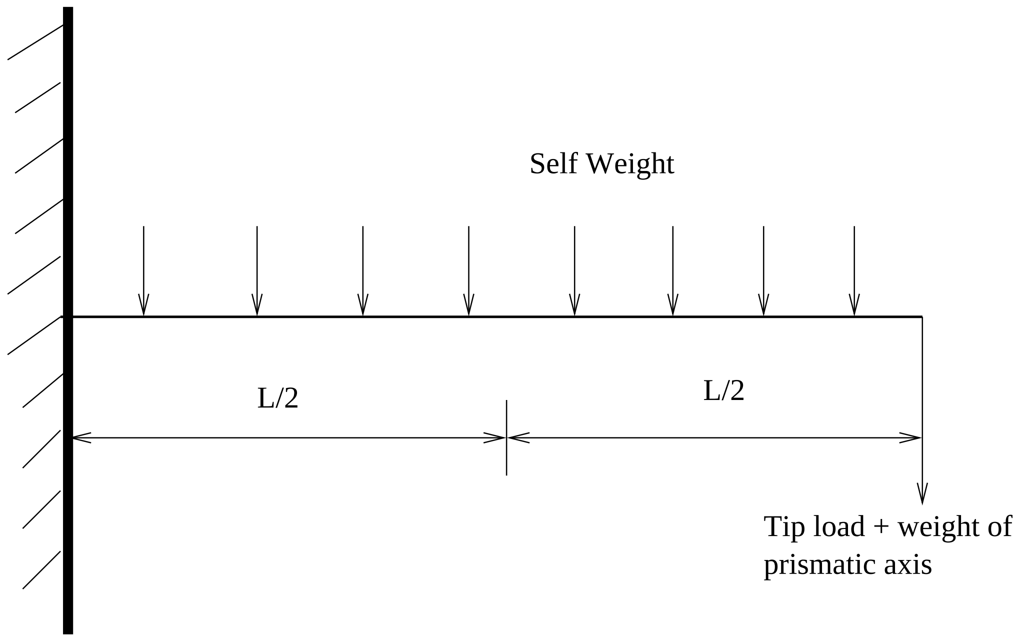


Figure 3.1: Load on the forearm

d = pitch diameter of ball-screw

α = helix angle

$\phi = \tan^{-1} \mu$

μ = coefficient of friction between screw and nut

Once the torque is obtained, then the screw is checked for combined torque and axial load using maximum normal stress theory. After considering the results of this procedure and market availability, we decided on a 14mm pitch diameter ball-screw.

Since the ball-screw is designed to take only the axial load and torque, all bending moments need to be countered by separate elements. We used two 12 mm diameter, case hardened and ground shafts in parallel with the ball screw to achieve this aim. These shafts move along with the ball screw and are supported using linear bearings.

3.3.2 Design of forearm

The forearm needs to be rigid enough to resist the bending moment in the vertical plane generated by the combined weight of the prismatic axis and the tip load, as well as the self-weight of the forearm. This is shown schematically in fig. 3.1

Using bending equation 3.3, we calculated the minimum MI required about the bending axis.

$$\frac{M}{I} = \frac{\sigma}{y} \tag{3.3}$$

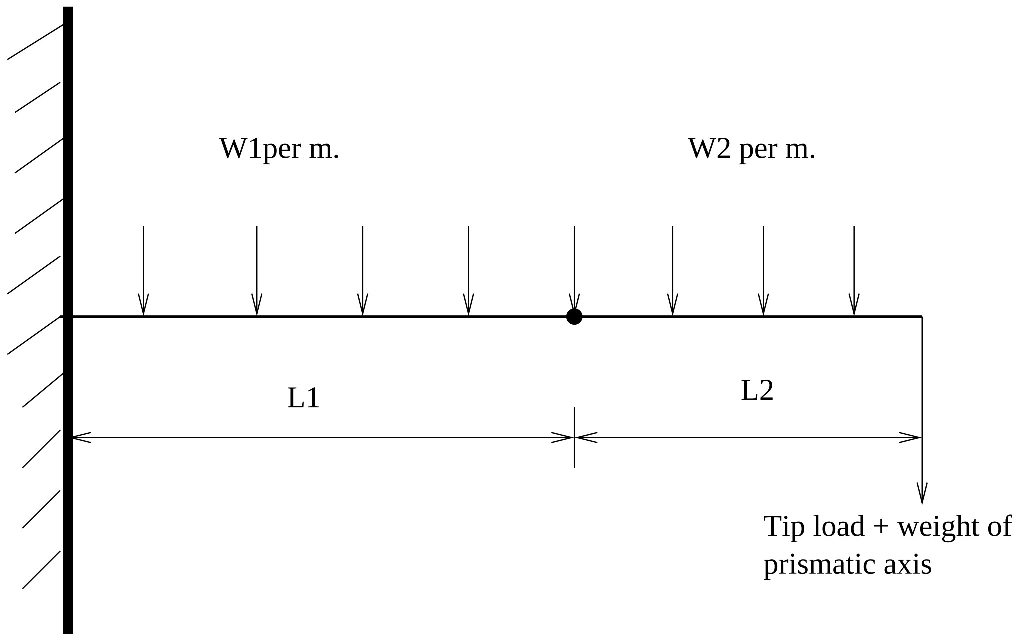


Figure 3.2: Load on elbow

where M is the bending moment generated by the combined load on the forearm, as shown in fig. 3.1. Once the MI was known, a 40x40 mm MS square hollow section of 3 mm thickness was selected for the forearm. The actual deflection of the forearm was then calculated using equation 3.4

$$\delta = \frac{wL^3}{8EI} + \frac{WL^3}{3EI} \tag{3.4}$$

where ,

δ = deflection of forearm tip

w = self-weight of forearm/meter

W = load at tip of forearm

The actual deflection came out to be less than 0.01mm i.e far below the control resolution of the prismatic axis.

3.3.3 Design on elbow

The design of the elbow follows that of the forearm. The only addition is the of self-weight of the elbow and the increase in length. Equation 3.3 remains the same, while equation 3.4 is modified to

$$\delta = \frac{w_1 L_1^3}{8EI_1} + \frac{W(L_1 + L_2)^3}{3EI_1} + \frac{w_2 L_2 (L_1 + \frac{L_2}{2})}{8EI_1} \tag{3.5}$$

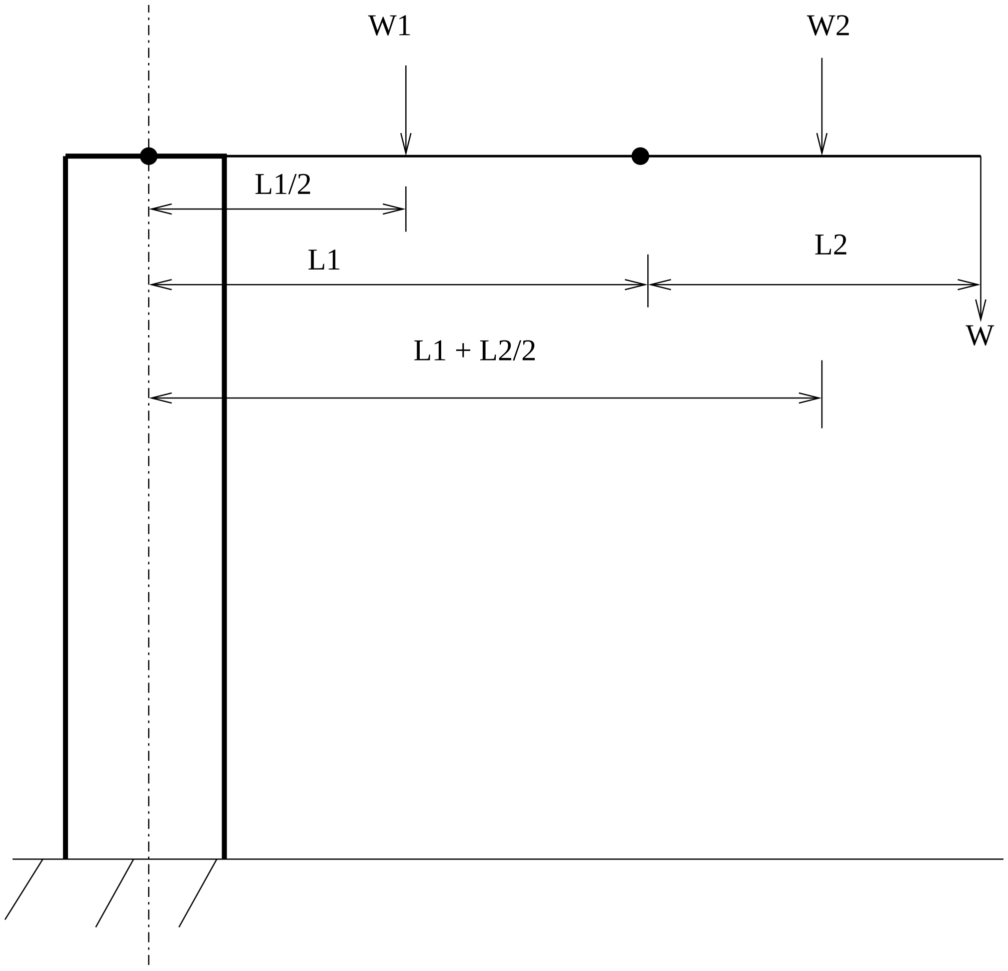


Figure 3.3: Load on column

where

w_1 = self weight per meter of elbow

w_2 = self weight per meter of forearm

A 75 mm channel section was used for the elbow arm. The actual deflection was found to be less than the control resolution.

3.3.4 Design of vertical column

Fig. 3.3 shows the loading on the vertical column. Its design essentially comprises of applying equation 3.3.

In this case,

$$M = W(L_1 + L_2) + W_1\frac{L_1}{2} + W_2(L_1 + \frac{L_2}{2}) \tag{3.6}$$

We selected a hollow MS pipe of external diameter 60 mm and wall thickness 4 mm.

3.3.5 Design of shafts

The various shafts were designed to sustain the torsional stresses. The equation used was

$$\frac{T}{J} = \frac{\tau}{r} \tag{3.7}$$

3.4 Selection of motors

The torques required to drive the elbow and forearm for a SCARA robot are given by

$$\begin{aligned}
\tau_1 &= \left[(\frac{m_1}{3} + m_2 + m_3)a_1^2 + (m_2 + 2m_3)a_1a_2C_2 + (\frac{m_2}{3} + m_3)a_2^2 \right] q_1^{''} \\
&- \left[(\frac{m_2}{2} + m_3)a_1a_2C_2 + (\frac{m_2}{3} + m_3)a_2^2 \right] q_2^{''2} + b_1(q_1^{'}) \\
&- a_1a_2S_2 \left[(m_2 + 2m_3)q_1^{'}q_2^{'} - (\frac{m_2}{2} + m_3)q_2^{'2} \right]
\end{aligned} \tag{3.8}$$

$$\begin{aligned}
\tau_2 &= -\left[(\frac{m_2}{2} + m_3)a_1a_2C_2 + (\frac{m_2}{3} +)m_3)a_2^2 \right] q_1^{''} + (\frac{m_2}{3} + m_3)a_2^2 q_2^{''} \\
&+ (\frac{m_2}{3} + m_3)a_1a_2S_2q_1^{'2} + b_2(q_2^{'})
\end{aligned} \tag{3.9}$$

The terms involving $q^{''}$ are inertia forces while those involving $q_1^{'}q_2^{'}$ are the Coriolis force components. Those involving $q^{'2}$ are the centrifugal velocity coupling. The term $b(q^{'})$ is to account for the joint friction. Since we used anti-friction bearings, this term has been ignored. In the above equations,

m_1 = mass of elbow
m_2 = mass of forearm
m_3 = effective mass on forearm tip
a_1 = length of elbow
a_2 = length of forearm
C_2, S_2 = $\cos(\theta_2)$ and $\sin(\theta_2)$ respectively
$q_1^{'}, q_2^{'}$ = angular velocities of the elbow and forearm respectively
$q_1^{''}, q_2^{''}$ = angular accelerations of the elbow and forearm respectively

Using these equations, 13 kg-cm motors for were found to be suitable for both the elbow as well as joint axes.

Chapter 4

Electronics hardware design

Preliminary thoughts on design have been covered in section2.3. We will now concentrate our attention on the actual electronics used in the robot.

From chapter 3 we know that the maximum torque requirement is 13 kg-cm. We had a means of easily procuring stepper motors manufactured by Sanyo Denki, and so after looking up the relevant catalog, the following models were selected:

103H7126-5040: This is a 13 kg-cm 2 phase, bipolar motor with 1.8 degrees/step and 2A current rating.

103H548-0440: This is a 2.7 kg-cm unipolar motor with 1.8 degrees/step and 1.2A current rating.

It was decided to build a circuit capable of driving either of these motors. Four prototypes were built, one for each motor.

4.1 The power driver

Most control circuits are low power devices, which generally operate within a range of 0 to +5 Volt and a current rating of a few milli-ampere. The motor, on the other hand, is a high power device requiring 2 Ampere current. Thus, the control circuits are clearly incapable of driving a motor directly. What is needed is a power stage that can accept low power input and provide sufficient output to drive the motor. There are standard circuits to drive motors, and considering our requirements, we opted for the L298 driver from SGS-THOMSON.

The L298 is a high current, dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads like relays, solenoids, DC and stepping motors. The internal block diagram of the IC is shown in fig. 4.1

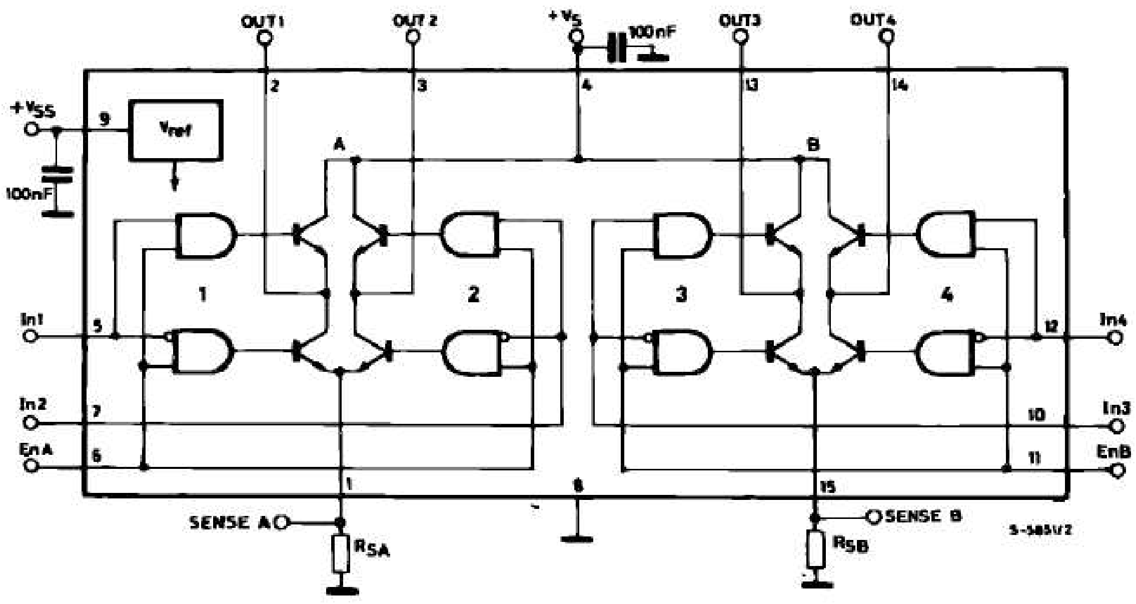


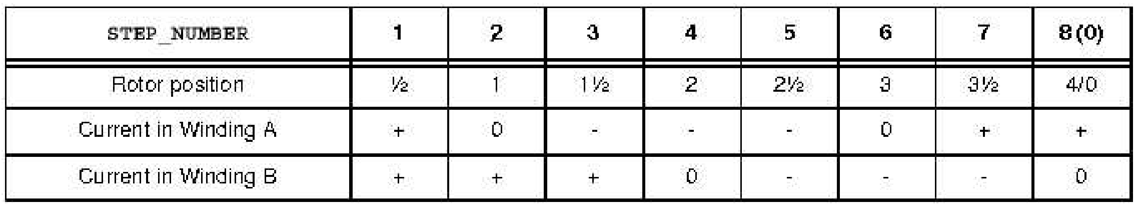
Figure 4.1: Internal block diagram of the L298

Two enable inputs are provided to enable or disable the device independently of input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage. The main characteristics of the L298 are

- Operating supply voltage up to 46V.
- Total DC current up to 4 A.
- Low saturation voltage.
- Over-temperature protection.
- Logical '0' input voltage up to 1.5V (High noise immunity).

The L298 integrates two power output stages. An output stage is a bridge configuration that can drive an inductive load in common or differential mode, depending on the state of the inputs. The current that flows through the load comes out from the bridge at the sense output: an external resistor here will allow to detect the intensity of this current.

Each bridge is driven by means of four gates, the inputs of which are In1, IN2, EnA and In3, In4 and EnB. The In input sets the bridge state when the En input is high. A low state of the En input inhibits the bridge. All inputs are TTL compatible.



STEP_NUMBER	1	2	3	4	5	6	7	8(0)
Rotor position	½	1	1½	2	2½	3	3½	4/0
Current in Winding A	+	0	-	-	-	0	+	+
Current in Winding B	+	+	+	0	-	-	-	0

Figure 4.2: Half-step sequence.

It should be noted that an external bridge of diodes is required when the inputs of the IC are chopped. Fast Schottky diodes from the FR series are preferred. These diodes are shown on the output side in fig. 4.6 and fig. 4.7.

4.2 The phase translator

A stepper motor needs a specific sequence of pulses in its windings in order to rotate. For half-step mode, the currents in each winding at a given step number and rotation fraction, is given in fig. 4.2

There are two options to generate this phase sequence.

1. Sequence generation through the computer itself.

2. Using a separate sequence generation chip.

Since the sequence is nothing but a combination of 1's and 0's, it can be very well generated from the computer itself. However, there is a major limitation to this technique when driving a large number of motors. Specifically, 4 output lines per motor are necessary. The standard parallel port of a computer has only output lines. Although the 4 control lines can be used for output, it still amounts to a total of 12 lines. This implies that simultaneous operation of only three motors is possible. Since the SCARA requires coordinated linear motion of 4 axes, this mode of sequence generation falls short of the requirement.

A better approach is through the use of a phase translator chip. These IC's contain internal translators that generate the stepping sequence from a minimum input of clock and direction. Thus it is possible to drive 4 motors simultaneously from the parallel port. These IC's also sport other attractive features like enable/disable inputs, half/full step sequence generation, reset and sync possibilities etc. For our project, we used the L297 stepper motor controller chip from SGS-THOMSON.

The L297 IC is specifically designed for use with the L298 and thus integrates well with the rest of our setup. It receives control signals from an external source (in our

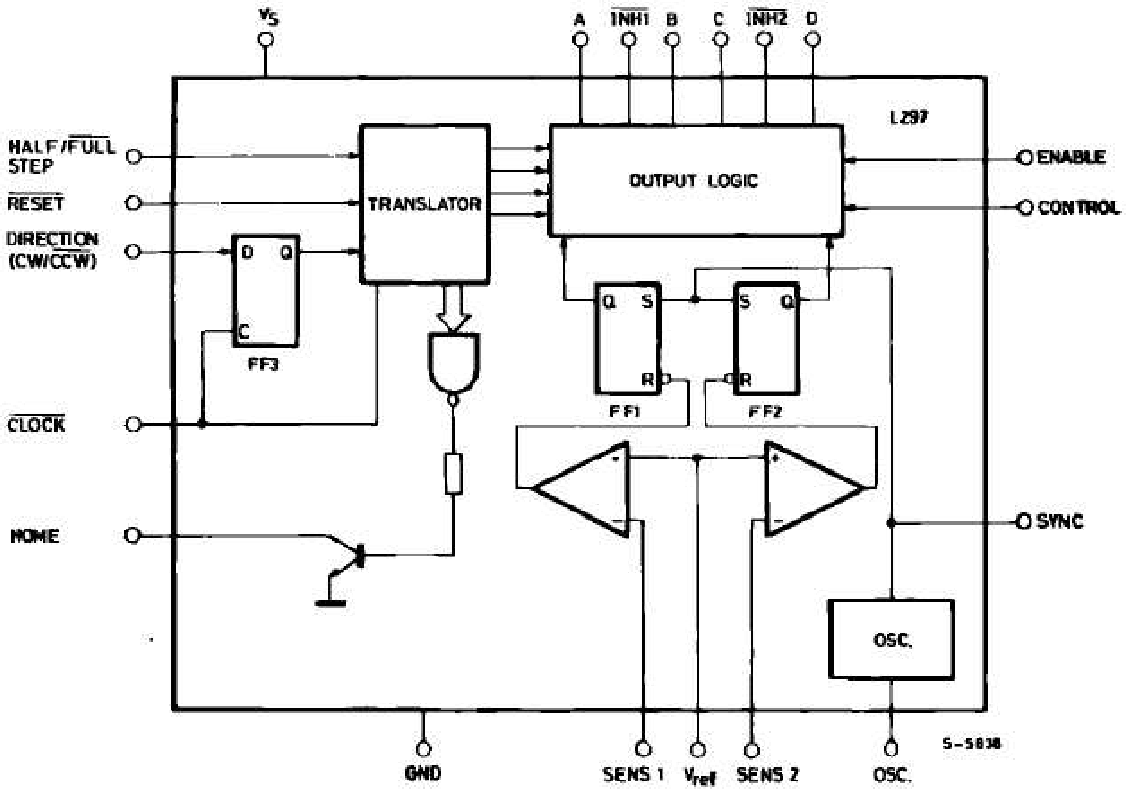


Figure 4.3: Internal block diagram of the L297

case, this source is the computer) and provides all the necessary drive signals for the power stage. Additionally, it includes two PWM chopper circuits to regulate the flow in the current windings. It also handles normal, wave drive and half-step drive modes. Other advantages are

- Very few components are required. So assembly costs are low, reliability is high and little space is required.
- Software development is considerably simplified.
- Computational burden on the computer is significantly reduced.
- The L297 can be used with any power stage, including discrete power devices.

An internal block diagram of the L297 is shown in fig. 4.3

The 8 step master sequence of the translator, corresponding to half-step mode is shown in fig. 4.4. Clockwise rotation is indicated.

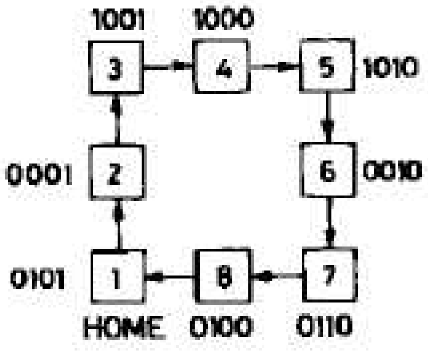


Figure 4.4: Half-step sequence

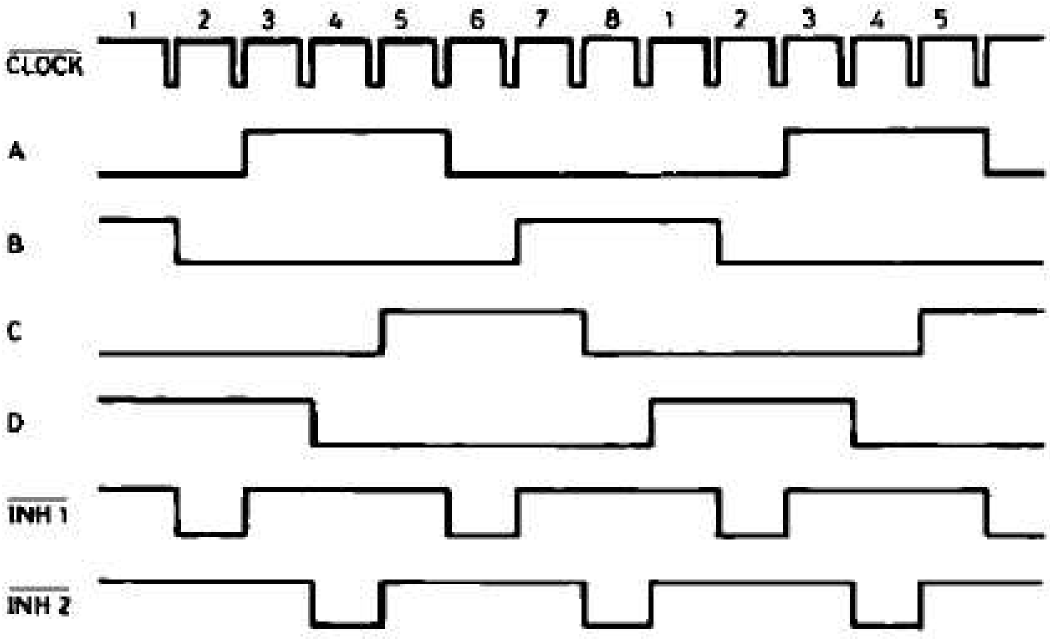


Figure 4.5: Output of the L297

4.3 How everything hangs together

In this section, we show how both the stages discussed above are tied together to give the required result. Fig. 4.5 shows the output waveforms of the L297 corresponding to half-step mode. The chopper action is not indicated, in order to keep things simple.

Fig. 4.6 gives a clear idea of the working of the circuit at the block diagram level.

The actual circuit is indicated in fig. 4.7

Finally, the voltage waveforms for half-step controls in the individual windings are shown in fig. 4.8.

The aim of this section was to shed light on the circuits required to drive the stepper motors. The reader should note that assorted circuitry for noise filtering and isolation

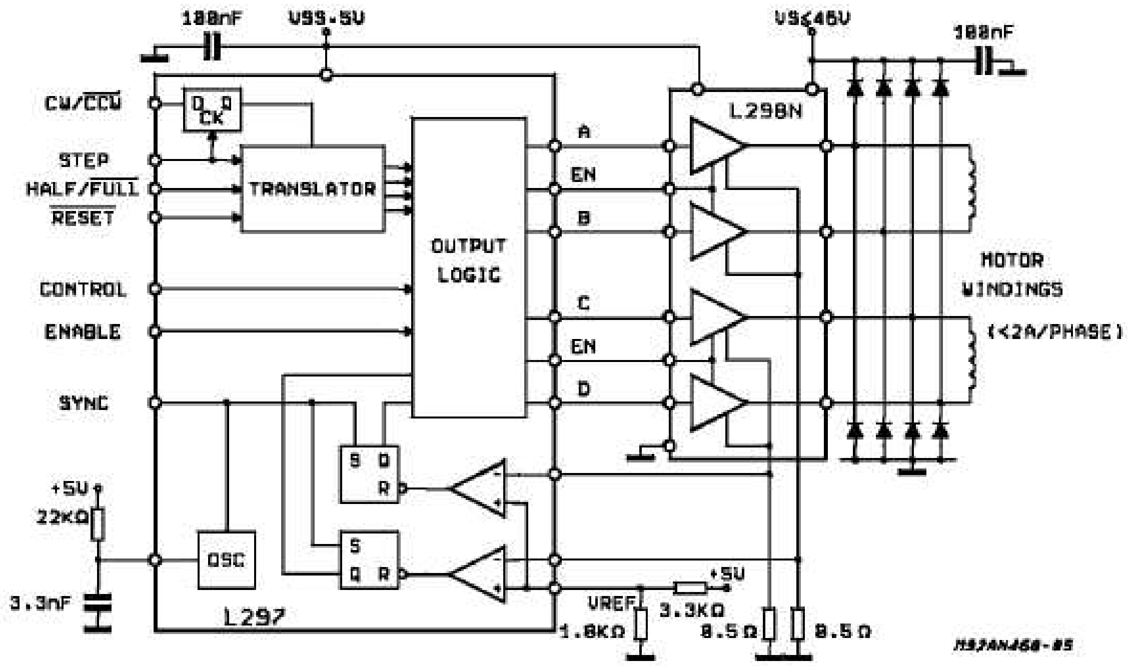
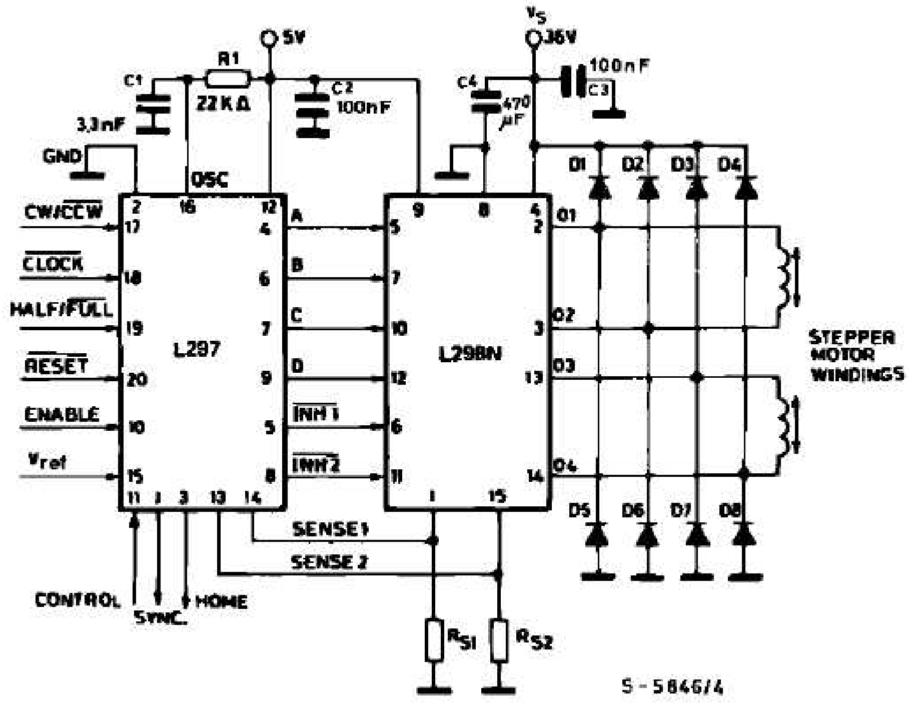


Figure 4.6: Block diagram of the entire circuit



$R_{S1} = R_{S2} = 0.5\ \Omega$

D1 to D8 = 2 A Fast diodes $\left\{ \begin{array}{l} V_F \le 1.2\ V\ @\ I = 2\ A \\ t_{rr} \le 200\ ns \end{array} \right.$

Figure 4.7: The actual circuit

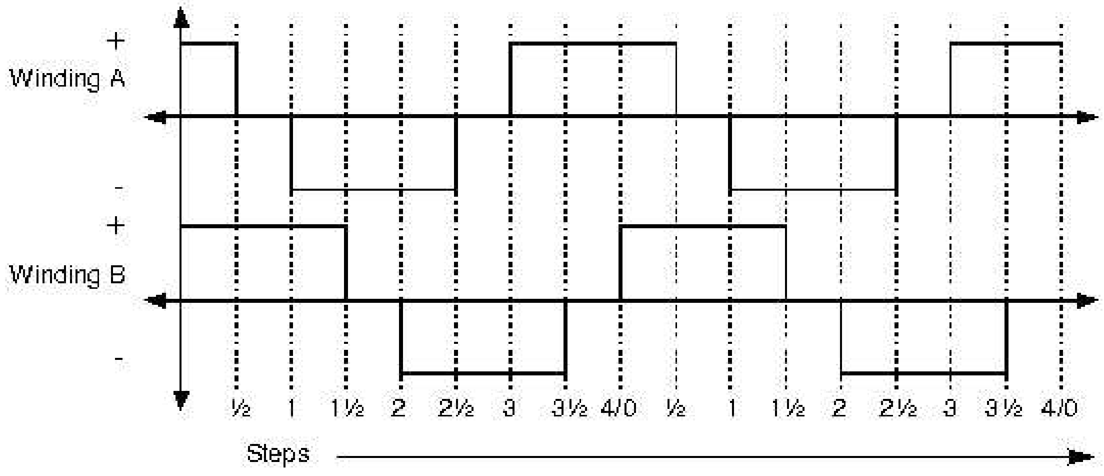


Figure 4.8: Winding voltage waveforms

at various points is also required. These circuits are banal and trivial and will not be discussed here. They can be found in any good book on basic electronics. However, for the sake of completeness, the implemented power and drive circuit diagrams are included as inserts in this report.

Chapter 5

Control

5.1 The link coordinate diagram

Before any control theory can be developed, a consistent and systematic representation of the robot has to be developed. This involves assigning coordinate frames to the manipulator and assigning the Denavit-Hartenberg link parameters. Applying the Denavit-Hartenberg algorithm to the link chain results in the link coordinate diagram shown in fig. 5.1

The kinematic parameters of the robot are summarized in table 5.1.

5.2 The arm matrix

Next, we calculate the arm matrix for the robot.

$$T_{base}^{tool} = T_0^1 T_1^2 T_2^3 T_3^4 \tag{5.1}$$

where,

$Axis$	θ	d	a	α	$Home$
1	q_1	d_1	a_1	π	0
2	q_2	0	a_2	0	0
3	0	q_3	0	0	100
4	q_4	d_4	0	0	$\frac{\pi}{2}$

Table 5.1: Kinematic parameters

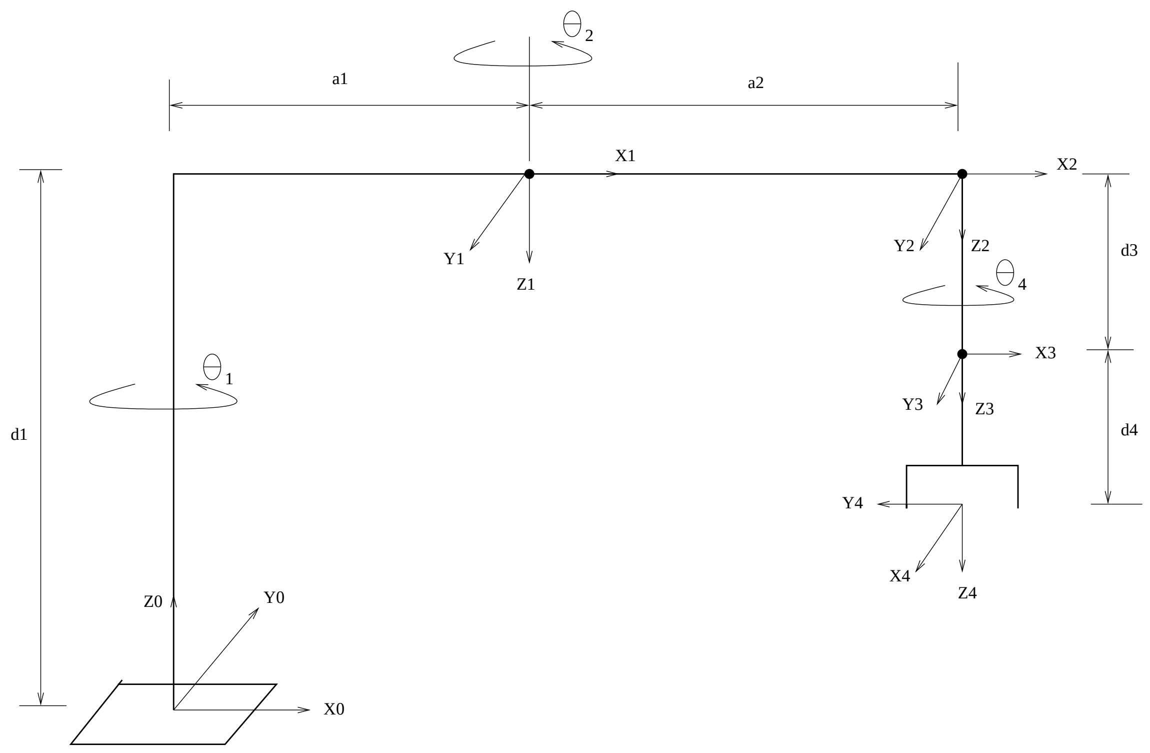


Figure 5.1: The link coordinate diagram

$$T_0^1 = \begin{bmatrix} C_1 & S_1 & 0 & a_1C_1 \\ S_1 & -C_1 & 0 & a_1S_1 \\ 0 & 0 & -1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^4 = \begin{bmatrix} C_4 & -S_4 & 0 & 0 \\ S_4 & C_4 & 0 & 0 \\ 0 & 0 & 2 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After multiplication and simplification using trigonometric identities, we get the following arm matrix

$$T_{base}^{tool} = \begin{bmatrix} C_{1-2-4} & S_{1-2-4} & 0 & a_1C_1 + a_2C_{1-2} \\ S_{1-2-4} & -C_{1-2-4} & 0 & a_1S_1 + a_2S_{1-2} \\ 0 & 0 & -1 & d_1 - q_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

Here, the notation C_{1-2-4} denotes $\cos(q_1 - q_2 - q_4)$, and similarly, S_{1-2-4} denotes $\sin(q_1 - q_2 - q_4)$. Note that the approach vector is *fixed* at $r_3 = -i^3$, independent of the joint variables. This is a characteristic of SCARA robots, since they are designed to manipulate objects from directly above.

5.3 Forward kinematics

The forward kinematics problem involves the mapping of a vector from joint space to tool-configuration space. Thus, when a joint vector is known, using forward kinematics equations, it is possible to calculate the world coordinates of the tool[1] tip. In our software, forward kinematics is used for purposes like jogging the individual axes in joint mode.

In order to derive the forward kinematics equations, we make use of two properties of the arm matrix.

1. The columns of the arm matrix give the coordinates of the tool frame in terms of the base frame.

2. For a give 4x4 homogeneous arm matrix, the matrix comprising $A_{i,j}$, $i, j \leq 3$ denotes the tool orientation matrix, while the matrix comprising $A_{i,j}$, $j = 4$ and $1 \leq i \leq 3$ denotes the position vector of the wrist.

Thus, from equation 5.2 we can conclude that for a given joint vector $J(\theta_1, \theta_2, d_3, \theta_4)$ the following equations hold true.

[1]By *tool* we mean any end effector attached to the robot's wrist.

$$X = a_1 C_1 + a_2 C_{1-2} \tag{5.3}$$

$$Y = a_1 S_1 + a_2 S_{1-2} \tag{5.4}$$

$$Z = d_1 - d_3 \tag{5.5}$$

Note that the equation for Z is slightly modified. Specifically, we do not consider the length d_4 because we intend to program the robot using G-codes. Naturally, d_4 will be specified as the tool length offset.

Note that if the sign conventions are set such that all counter-clockwise rotations are positive, then C_{1+2} and S_{1+2} will have to be used in equations 5.3 and 5.4 respectively, instead of C_{1-2} and S_{1-2}.

The magnitude of roll i.e θ_4, remains the same.

5.4 Inverse kinematics

Inverse kinematics involves finding an inverse mapping from tool-configuration space back to joint space. Thus, we can determine the necessary joint vector to achieve a given tool orientation and position in space.

Inverse kinematics is also used to check that the joint limits of the robot are observed. Joints 1 and 4 do not have any limits. Joint 2 is limited to angles between $\pm 150^o$. Joint 3, the prismatic joint, is limited to movement not exceeding its length (0.15*m*). No solutions to the inverse kinematics problem are possible if any joint overrides its limits.

Working back wards from equation 5.2, we obtain the following general inverse solution, once given a vector $W(X, Y, Z, \theta_4)$ in the tool-configuration space.

$$\theta_2 = \pm \arccos\left(\frac{X^2 + Y^2 - a_1^2 - a_2^2}{2a_1 a_2}\right) \tag{5.6}$$

$$\theta_1 = \arctan\left(\frac{Y}{X}\right) - \arctan\left(\frac{\pm a_2 S_2}{a_2 C_2 + a_1}\right) \tag{5.7}$$

$$d_3 = d_1 - Z \tag{5.8}$$

As before, the value of roll i.e θ_4, remains the same!

Inverse kinematics equations are also used in other tasks like trajectory generation and interpolation.

5.5 Motion planning

Motion planning deals with trajectory and path generation. The term *path* refers to the locus of points in space that the tool tip traverses in a given motion cycle. *Trajectory* is a more specific term and it includes the time-frame and time-scale within which the motion cycle is completed. As mentioned in section 1.1, the robot is programmed using the RS274NGC[2] language. Motion control includes all of the following functions:

- Sampling the position of the axes to be controlled.
- Computing the next point on the trajectory.
- Interpolating between these trajectory points.
- Computing an output for the motors.
- Programmable software limits.
- Interfaces to hardware and limit switches.
- PID servo compensation with zero, first and second order feed forward.
- Maximum following error.
- Selectable velocity and acceleration values.
- Individual axis jogging (continuous, incremental and absolute).
- Queued blended moves for linear and generalized circular motion.
- Programmable forward and inverse kinematics.

Efficient motion planning makes all the difference between a crude point-to-point robot and a highly sophisticated, contouring one. An overview of the trajectory planning algorithm used for the robot control software is given below.

During each trajectory planning cycle, the trajectory planner computes the next point on the path, based on the type of motion segment (line, circle, arc etc.), speed and the acceleration/deceleration specified. The time period of the trajectory cycle is

[2]This is the NGC dialect of RS274. NGC stands for Next Generation Controller.

typically 10 milli-seconds.[3] After every time period, the trajectory point is run through the inverse kinematics functions to generate the corresponding axes points. Thus, at the end of each trajectory cycle, a new set of axes points is obtained.

There is an interpolator for each axis and each point from the new set is appended to its associated axis interpolator. This is because the axis controller runs at a higher frequency than the trajectory planner. The cycle time for the axis controllers is typically 1 millisecond.[4] Thus, the axes controller runs 10 times faster than the trajectory cycle. Since the trajectory planner and axis controllers all run in one piece of code, what happens is that the whole loops runs at the fastest rate, and the trajectory calculations are done once every Nth time through the loop. All the axis calculations are run at the rate of the fastest, instead of trying to slow some of the slower ones down by only running them every other cycle.

The axis controller is where PID servo control is done, or the frequency generator for steppers is loaded. In every one of the non-trajectory cycles, the interpolator generates an in-between point from the last axis point to the most recent axis point.

Many types of interpolation can be done.

Zeroth-order interpolation is one in which the same point is used over and over again, until the next trajectory point is computed. Then the new one is used over and over again. At the transition, there are jumps in position, speed and acceleration.

First-order interpolation is one in which the in-between points fall proportionally between the end points. At the transition, position is continuous but there is a jump in speed, and acceleration has large spikes.

Second-order (quadratic) interpolation is one in which the in-between points fall on a curve between the end points. At the transition, position is continuous, and at one side (just preceding or just following the transition), speed can be continuous. Since there is no point in only having speed continuity half of the time, second-order interpolation is not of practical use.

Third-order (cubic) interpolation is one in which the in-between points fall on a curve between the end points. At the transition, position and speed are continuous, but there is a jump in acceleration and jerk (change in acceleration per change in time) has large spikes.

Fourth-order (quartic) interpolation is one in which the in-between points fall on a curve between the end points. At the transition, position and speed are continuous, and at one side (just preceding or just following the transition), acceleration can be continuous. Since there is no point in only having acceleration continuity half of the

[3]This value can be changed.

[4]This value can be changed.

time, fourth-order interpolation (like second-order interpolation) is not of practical use. Neither are any other even-order interpolators.

Fifth-order (quintic) interpolation, is one in which the in-between points fall on a curve between the end points, like the cubic. At the transition, position, speed and acceleration are continuous. Jerk is bounded and doesn't have spikes. This is the smoothest from the group, but although have code for this, we haven't implemented it yet.

Each type of interpolation introduces a delay, and the higher the order, the longer the delay. This is because the interpolating curve parameters need to fit more input points, and they can't be constructed until all the necessary points have been given to the interpolator. For zeroth order interpolation (no interpolation), there is no delay and it can be used once a single point is obtained. For first order, two points are needed in order to begin interpolating. For third (cubic), four points, for fifth (quintic), six points, etc.

Currently, cubic interpolation is being utilized in the robot control software.

Chapter 6

Software design

We now turn to some of the software aspects of our project. We have filtered down all principles to their lowest common denominator. Thus, this chapter should be understandable by everybody with a basic familiarity with computers. We will *not* be jumping into all the gory details here. Those are already well documented within the source code and other help files on the accompanying CD. We have no interest in reprinting thousands of lines of code.

6.1 Programming the parallel port

A quick overview of programming the parallel port is presented in this section.

The programming interface to the parallel port consists of 3 registers. These are the Base register, Status register and the Control register. Each register has an address in memory. The addresses of the Status and Control registers depend on the address of the Base register, which is simply called the base address. In a computer, the base address is either of 0x3bc, 0x378 or 0x278.

The addresses of the other registers are calculated as follows:

$$Status\, register\, address = Base\, address + 1$$

$$Control\, register\, address = Base\, address + 2$$

The Base register is used for data output, status register is used for data input and the control register is used for setting operating modes, interrupt enable/disable etc.

Data written to the base register appears as voltage at the output pins of the parallel port. Voltages applied to the input pins appear as data in the status register. These voltages correspond to the TTL standard i.e. in the ideal case, a logical HIGH (1) is

+5V and a logical LOW(0) is 0V.

Pins 2 through 9 are data outputs, pins 10 through 13 and 15 are used for data input. Thats about all the information we require to get started. Further information can be found in references[9, 10, 11, 12].

6.2 Driving a stepper motor

A stepper motor requires a train of pulses in order to rotate. The rotation of the motor is a function of the number of pulses sent and their frequency. There are various techniques of producing a pulse train at the output pins of the parallel port. The algorithm for the simplest technique is

```
1. Send logic 1[1] to the pin.
2. Wait for some time.
3. Send logic 0[2] to the pin.
4. Wait for some time.
5. GOTO step 1.
```

Unfortunately, this technique cannot be used for driving multiple motors simultaneously. There are other ways to accomplish this. The best[3] method is discussed below.

There is a structure associated with each motor that contains actual and commanded values of speed, position and acceleration. The control loop consists of a single fast periodic task that drives all the motors. To move the motor move from point X to point Y, this is what happens: The motor thinks it is at point X. It gets a message to move to point Y. From this, it figures the direction of movement. It then uses an accumulator algorithm for acceleration/deceleration. Each time through the task, the speed is increased as per the equation

$$speed = speed + acceleration * TIME\_PERIOD$$

The increment to distance in each period is smaller than the step size. Hence, a position accumulator, *accumPos,* is incremented each time through the cycle, as per the equation

$$accumPos = accumPos + speed * TIME\_PERIOD$$

[1]1 means HIGH. Which means that +5V appears at the pin.

[2]0 means LOW. Which means pin voltage drops to 0V.

[3]By "best", we mean the best technique for *our* purpose.

Next, the value of *accumPos* is compared against the motor step-size. If

$$accumPos \geq stepSize$$

then,

$$\begin{aligned} distance &= distance + stepSize \\ accumPos &= accumPos - stepSize \end{aligned}$$

Naturally, both values are corrected for direction also. Thus, the motor accelerates as long as

$$speed < targetSpeed$$

When the target speed is reached, it stops accelerating and a cruises at a steady velocity. To slow down, the motion code calculates the stopping distance given its current speed and the constant deceleration value. It checks this as part of the main loop and sets target speed to zero. The same cycle repeats and the motor slows down.

6.3 The Enhanced Machine Controller[4]

The Enhanced Machine Controller[13] is a free and open source program that can control servo motors, stepper motors, relays and other devices related to machine tools. We have made extensive use of the EMC, adapting it for our control purposes. The SCARA forward and inverse kinematics were separately written and re-integrated with the main stream code. This section gives an overview of the EMC.

There are four components to the EMC software, as shown in fig. 6.1: a motion controller (EMCMOT), a discrete I/O controller (EMCIO), a task executor which coordinates them (EMCTASK) and a collection of text-based or graphical user interfaces.

6.3.1 The motion controller EMCMOT

The motion controller is written in C for maximum portability to real-time operating systems. Motion control includes sampling the position of the axes to be controlled, computing the next point on the trajectory, interpolating between these trajectory points

[4]This section is available courtesy of John Kasunich.
http://home.att.net/~JMKasunich/EMC_Docs/EMC_Home.htm

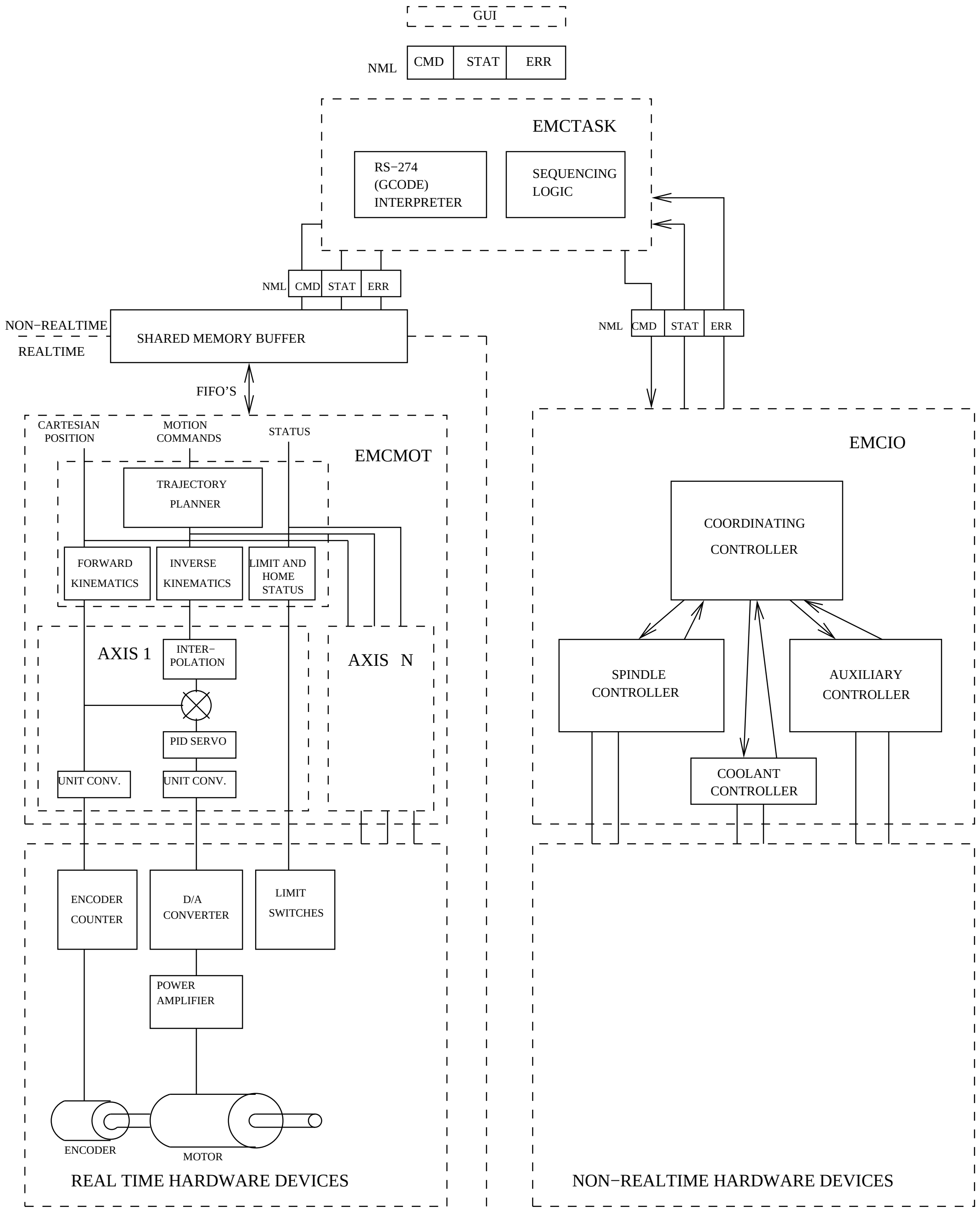


Figure 6.1: EMC architecture

and computing an output to the motors. For servo systems, the output is based on a PID compensation algorithm. For stepper systems, the calculations run open-loop, and pulses are sent to the steppers based on whether their accumulated position is more than a pulse away from where their commanded position should be. The motion controller includes programmable software limits, interfaces to hardware limit and home switches, PID servo compensation with zero, first, and second order feed forward, maximum following error, selectable velocity and acceleration values, individual axis jogging (continuous, incremental, absolute), queued blended moves for linear and generalized circular motion, and programmable forward and inverse kinematics. The motion controller is written to be fairly generic. Initialization files (with the same syntax as Microsoft Windows INI files) are used to configure parameters such as number and type of axes (e.g., linear or rotary), scale factors between feedback devices (e.g., encoder counts) and axis units (e.g., millimeters), servo gains, servo and trajectory planning cycle times, and other system parameters.

Complex kinematics for robots can be coded in C according to a prescribed function interface and linked in to replace the default 3-axis Cartesian machine kinematics routines. A C language application programming interface (API) between the motion controller and the external world is provided so that specific hardware can be integrated into the EMC without modifying any of the core control code. Programmers must implement these API functions for each board.

6.3.2 Discrete I/O controller EMCIO

The discrete I/O controller (bottom right in fig. 6.1) is written in C++, using the NIST RCS Library. It is based on a hierarchy of C++ controller classes derived from the NML_MODULE base class, each communicating using NML[5]. Discrete I/O controllers are highly machine-specific, and are not customizable in general using the INI file technique used to configure the more generic motion controller.

6.3.3 Task Executor EMCTASK

The Task Executor (second from top in fig. 6.1) is coded similarly to the discrete I/O controller, using the NML_MODULE base class and the RCS Library. It is less machine specific than the discrete I/O controller, as it is responsible for interpreting G and M code programs whose behavior does not vary appreciably between machines.

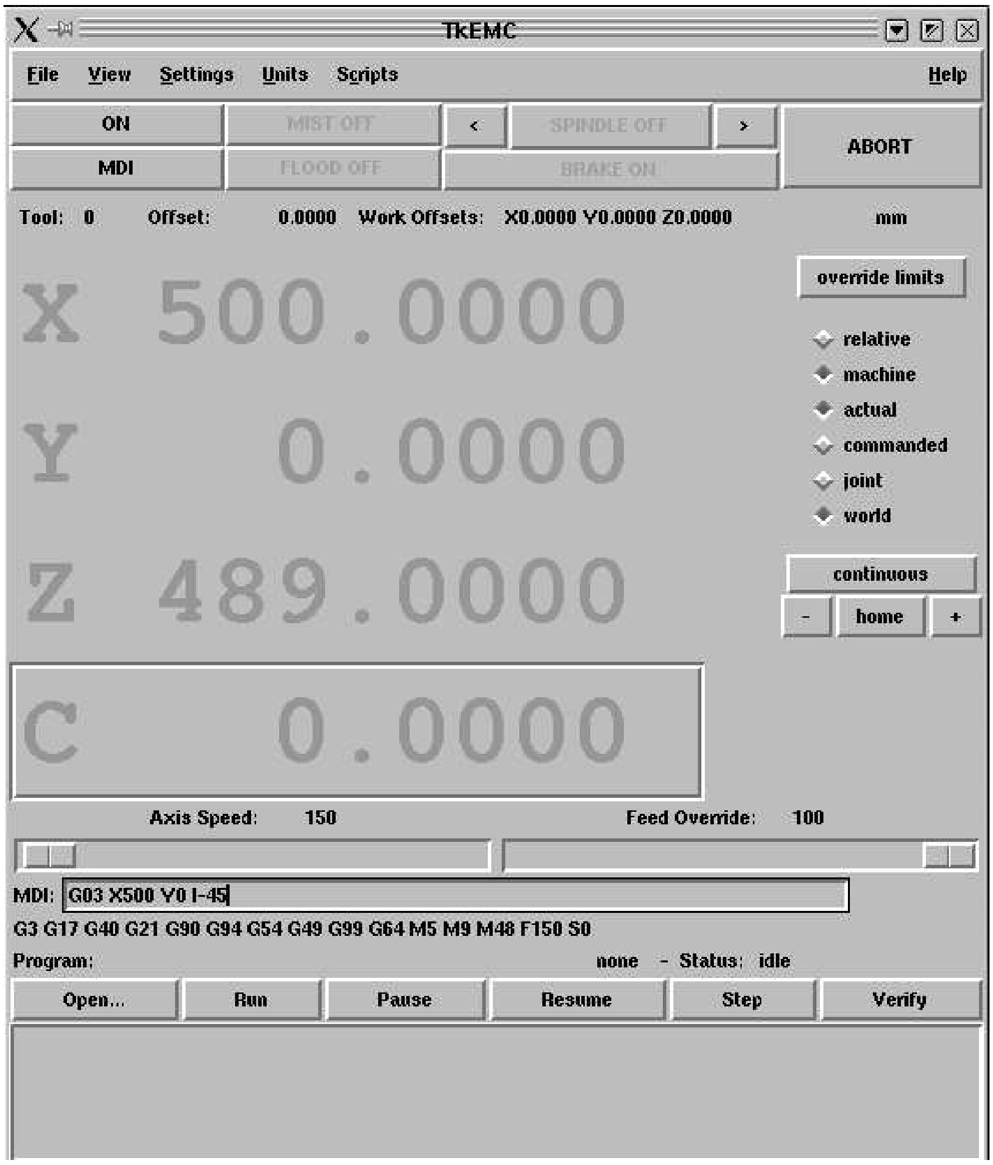
[5]NML stands for Neutral Messaging Language. It is platform independent.

6.3.4 Graphical User Interfaces

Indicated at the very top in fig. 6.1 is the fourth component of the EMC software, the graphical user interface (GUI). The architecture of EMC supports other components instead of the GUI, such as flexible manufacturing systems or remote GUI's. Whatever component is at the top has high level control of the EMC. It communicates using NML, sending messages such as power on, enter automatic mode, run a program, or power down. GUI's may send manual commands initiated by the operator, for example jogging machine axes in manual mode or homing each axis.

The EMC comes with several types of user interfaces: an interactive command-line program emcpanel, a character-based screen graphics program keystick, an X Windows program xemc, a Java-based GUI emcgui, and a Tcl/Tk-based GUI TkEmc. A screen shot of TkEMC is shown in fig. 6.2.

TkEmc is most well-supported, and runs on Linux and Microsoft Windows. The Windows version can connect to a real-time EMC running on a Linux machine via a network connection, allowing the monitoring of the machine from a remote location. TkEmc comes with the Linux distribution of the EMC.



Tool: 0 Offset: 0.0000 Work Offsets: X0.0000 Y0.0000 Z0.0000 mm

X 500.0000

Y 0.0000

Z 489.0000

C 0.0000

Axis Speed: 150 Feed Override: 100

MDI: G03 X500 Y0 I-45

G3 G17 G40 G21 G90 G94 G54 G49 G99 G64 M5 M9 M48 F150 S0

Program: none - Status: idle

Figure 6.2: TkEMC

Chapter 7

Robot capabilities

Some of the robot's capabilities are as follows:

1. Programmable using standard G and M codes.
2. Point-to-point control possible.
3. Continuous path control possible.
4. Remote operation of the robot is possible.
5. The GUI front-end can run on any operating system. For example, you can connect the robot to a computer running GNU/Linux, while the GUI can run on a separate computer running Microsoft Windows.
6. Back plotting is possible. This means, you can view the tool tip path in a separate graphical window, as the robot goes through its motions.
7. Verification of the input programs is possible. Thus, one can ensure that the input program is correct, before executing it on the robot.
8. Control resolution is 0.5 mm.

The work envelope of the robot is shown by the hatched region in fig. 7.1

The maximum outer radius is 590 mm, while the inner radius of the work envelope is 176 mm.

A screen-shot of the back plot in action is shown in fig. 7.2.

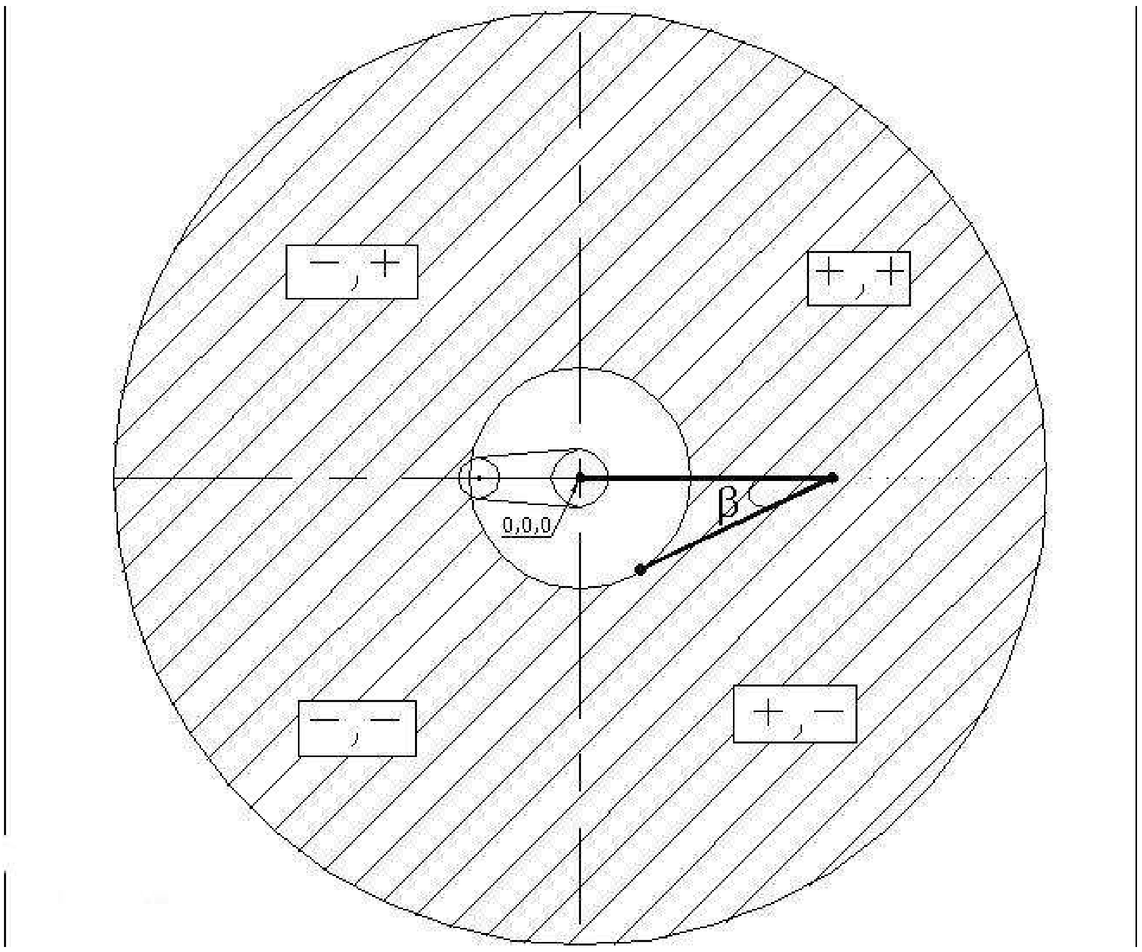
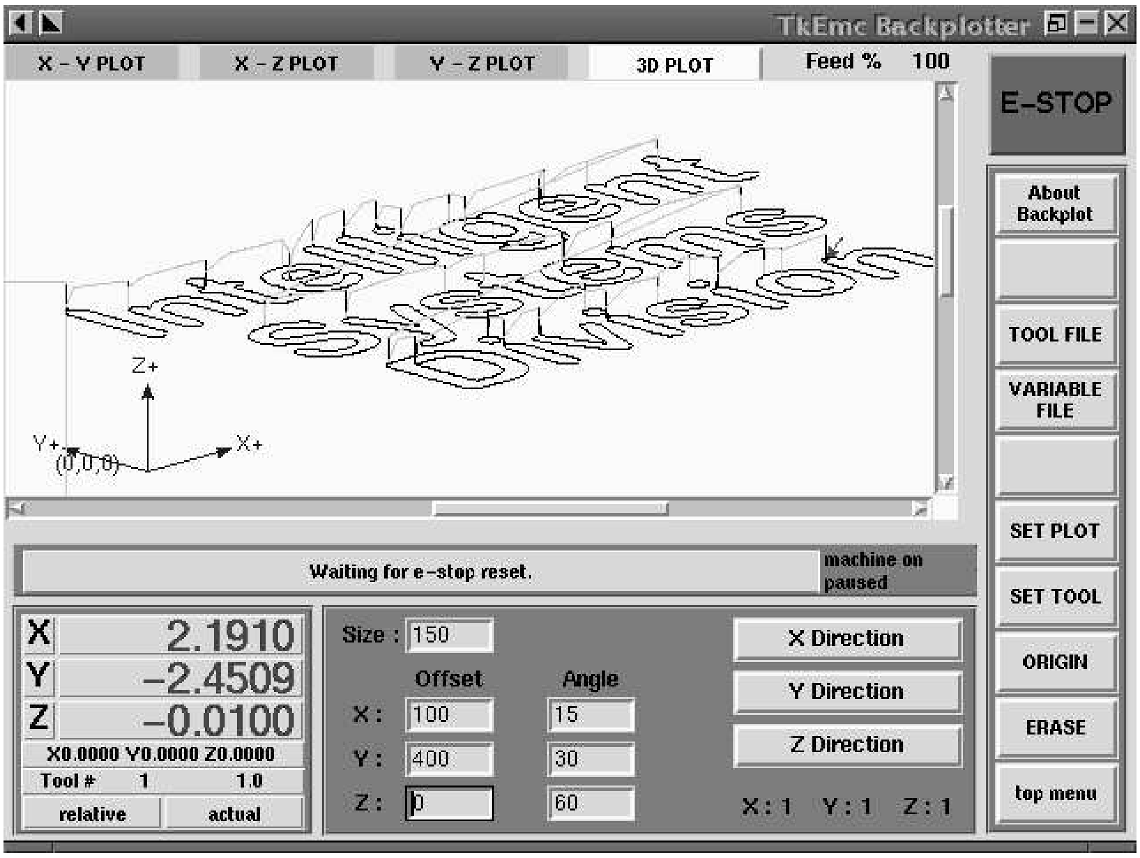


Figure 7.1: Work envelope



Waiting for e-stop reset.

machine on
paused

X 2.1910
Y -2.4509
Z -0.0100
X0.0000 Y0.0000 Z0.0000
Tool # 1 1.0

Size : 150

	Offset	Angle
X :	100	15
Y :	400	30
Z :	0	60

X : 1 Y : 1 Z : 1

Figure 7.2: The backplotter

Chapter 8

Scope of application

Our robot can be used for a wide variety of applications. The wrist of the robot ends in a flange assembly, to which different types of end-effectors can be clamped. Thus, apart from vertical assembly, some of the applications that it can easily perform are

- Spot welding
- Arc welding
- Engraving
- Pick-and-place operations
- Material handling and unhandling
- Certain types of machine loading and unloading
- Application of glue, sealants etc. in various manufacturing processes
- Painting and coating applications on $2\frac{1}{2}^o$ surfaces

Chapter 9

Costs involved

A rough cost calculation of the robot is given below:

Cost of motors Rs. 9000

Cost of electronics hardware Rs. 14000

Cost of raw material Rs. 20000

Machining cost Rs. 12000

Miscellaneous Rs. 6000

Total: Rs. 61000

Note that these costs do not include the cost of the computer and the software development costs.

Chapter 10

Conclusion

We are of the belief that *real* engineers should have wide interdisciplinary knowledge. This project involved a substantial amount of mechanical, electronics, computer and control engineering. Tackling it has exponentially increased our knowledge base.

Both of us are fascinated by machine automation and robotics, and making our very own robot has been a long standing dream. It is gratifying to see it converted to reality. Working on the robot has been a source of joy and inspiration to us throughout the year. So much so, that we are almost sorry that the project is over. It was one of those rare instances, where not only the result, but also the *process* was a source of continued pleasure.

However, in conclusion, we can only say that building the robot tested our design skills to the limit. Its success, therefore, is our greatest reward!

Acknowledgments

Building a complete robot from scratch is an extremely challenging process. We wouldn't have pulled through, without the help of several wonderful people. We take this opportunity to express our gratitude.

Our project guide, **Prof. J. S. Jadhav** granted us a remarkable degree of independence. Working on the project has been fun and rewarding due to the freedom, support and respect he has given us.

Dr. D. R. Pangavhane, the Head, Dept. of Mechanical Engineering has helped us in innumerable ways during the past four years. We take this opportunity to thank him.

Mr. Prakash Pujare is single-handedly responsible for ensuring that our project efforts took off the ground. He patiently advised us and entertained many wild ideas when we were initially deliberating over several project options.

Mr. Prakash Bhide was an immense help with the motors and electronics aspects of the project. We mechies would have never been able to develop the electronics without his assistance from time to time.

Andrew Mustun went to the extraordinary step of giving us an unlimited license to his **CAM Express** software. This is a wonderful software and proved invaluable in giving us better understanding and insight into CNC programming.

Paul Corner and **Ray Henry** are two really great people, directly responsible for helping us with the EMC. Sagar often sat down at nights to discuss his problems with **Paul**, only getting up when he realized that the Sun had risen and was peeping through the curtains. **Ray** was primarily responsible for setting the EMC juggernaut rolling. He helped with everything, from the concept to setting up the first configuration file.

Mr. Bhambre of Jogeshwari Industries generously let us use his machine shop to manufacture our robot. He also helped us from time to time with the manufacturing aspects.

Ashley Lima lent us his excellent printer, due to which printing the report was a breeze.

Finally, we would like to thank our families for pretending to believe that we were sane, while we went about our project.

Bibliography

[1] SCHILLING, R. J. (1990). *Fundamentals of Robotics: Analysis and Control,* Prentice-Hall, Inc.:Englewood Cliffs, N.J.

[2] FU, K. S., R. C. GONZALEZ, AND C. S. G. LEE (1987). *Robotics: Control, Sensing, Vision and Intelligence,* McGraw-Hill: New York.

[3] GROOVER, M. P., M. WEISS, R. N. NAGEL AND N. G. ODREY (1986). *Industrial Robotics: Technology, Programming and Applications,* McGraw-Hill: New York.

[4] KLAFTER, R. D., CHMIELEWSKI, T. A., M. NEGIN (1989). *Robotic Engineering: An Integrated Approach,* Prentice-Hall, Inc.:Englewood Cliffs, N.J.

[5] `http://www.gnu.org/philosophy/why-free.html`

[6] `http://www.gnu.org/`

[7] `http://www.gnu.org/philosophy/free-sw.html`

[8] `http://www.linux.org`

[9] `http://www.beyondlogic.org`

[10] `http://www.repairfaq.org/filipg/LINK/PORTS/F_PARALLEL1.html`

[11] `http://www.lvr.com`

[12] `http://et.nmsu.edu/~Eetti/`

[13] `http://www.linuxcnc.org`