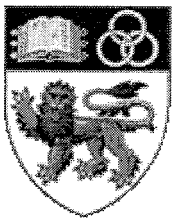
Control of an industrial robot

Technical University of Eindhoven
Department of Mechanical Engineering

TU/e technische universiteit eindhoven
faculteit werktuigbouwkunde

National University of Singapore
Department of Electrical Engineering



Control of an Industrial Robot

Tutor: Dr. Ir. M.J.G. v.d. Molengraft

M.J. v.d. Bosch 441983
B.H.M. Bukkems 428119
A.P. Teerhuis 443879

Ee Cheng Tien
Paul Ng
James Leong

Eindhoven/Singapore, August 25 2000

INDEX

Introduction

This project is conducted by two different disciplines in two different countries: the department of mechanical engineering at the Technical University of Eindhoven in The Netherlands (TUE) and the department of electrical engineering at the National University of Singapore (NUS).

The goal of the project is to design and make a control box for a pick & place robot. This robot is situated at the TUE and the control box will be manufactured at the NUS. This means the two universities have to co-operate with each other.

The students at the NUS will build a control box that can control the robot with given setpoints. The students at the TUE will supply the PID-controller layout and provide extra information for the NUS students.

This report starts with the description of the robot, after which the setpoint generation is described. Then the structure of the used controller will be described, followed by the system identification. Next the final control of the robot is presented after which the creation of the control box and additional program will be described. Finally, the performance of the control box with the robot will be elaborated.
Also a discussion will be included, to make comments on and suggestions for this international co-operation project.

Chapter 1: The robot

1.1: The robot

The robot (Philips CFT, VRS) that will be controlled is a so-called pick and place robot (see figure 1.1). The robot has three axes that can move separately and simultaneously.

At the lower end of the z-module, an electromagnet is situated. This magnet is used to pick the product.

To control the robot, a computer is used. This computer uses a Simulink model that will be downloaded into a dSPACE system. With the use of that dSPACE system, the motors are controlled via an amplifying unit. The position of an axis is measured by an encoder, which will deliver a block signal. This blocksignal is processed by dSPACE and converted to a voltage. This voltage is linear to the position of the axis.

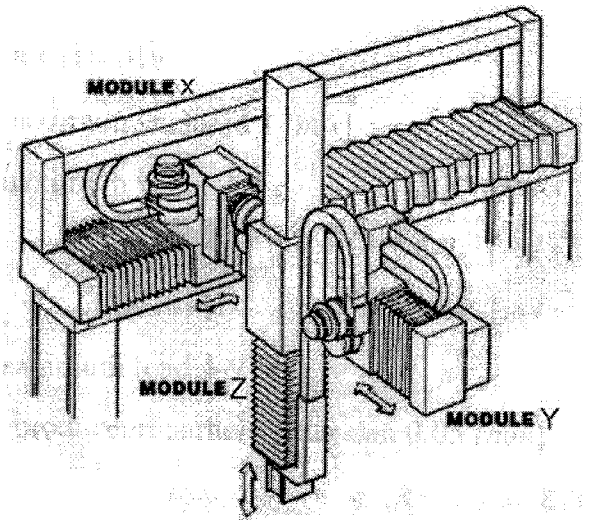


Figure 1.1: The robot with the three axes

1.2: dSPACE

As mentioned above, the Simulink model is downloaded into the dSPACE system. With the use of particular dSPACE software (dSPACE Control Desk), it is possible to change certain parameters of the model, e.g. turning the robot on/off.

The students at the NUS do not possess a dSPACE system and have to create their own: '*the control box*'. The design and creation of the control box will be elaborated in chapter 6 - 9.

The students of the TUE possess a dSPACE system and will use it to conduct experiments on the robot, e.g. to determine the controller and feedforward parameters.

1.3: Performance of the robot

A summary of the maximum velocity and acceleration and the movement range of the three axes [6] will be given in table 1.1. These values have to be taken into account, when designing a trajectory (see chapter 2).

Table 1.1: Performance of the three axis of the robot

	v_{max} [ms^{-1}]	a_{max} [ms^{-2}]	x_{min} [mm]	x_{max} [mm]
x-axis	1	4	80	690
y-axis	1	6	20	-370
z-axis	1	6	-20	-230

Chapter 2: Setpoint generation

2.1: Basic equations

To let the robot move from A to B, a route has to be designed. This route has to comply with a couple of requirements. To ensure smooth movement, sudden steps in velocity or acceleration are not allowed.

A sloped sinus is chosen as the signal for placement. Via differentiation, the signals for the velocity and acceleration are calculated [4]:

$$x(t) = - a \cdot \sin(b \cdot t) + c \cdot t + d \tag{2.1}$$

$$v(t) = - (a \cdot b) \cdot \cos(b \cdot t) + c \tag{2.2}$$

$$a(t) = (a \cdot b^2) \cdot \sin(b \cdot t) \tag{2.3}$$

The variables a, b, c and d are calculated in appendix 1. The results are given here:

$$dt = \frac{\pi \cdot v_{max}}{a_{max}},\ a = \frac{dt \cdot v_{max}}{4\pi},\ b = \frac{2\pi}{dt},\ c = \frac{v_{max}}{2} \text{ and } d = x_0$$

The time varies between '0' and 'dt'. In this time increment, the velocity builds up to maximum and then descends to zero. With the restrictions for the maximum velocity and maximum acceleration, it is not possible to reach every setpoint. For setpoints that don't need maximum velocity or acceleration, the maximum velocity or acceleration is corrected. For setpoints that need more than the maximum velocity and acceleration, a plateau of maximum velocity is inserted (see figure 2.1). Whether the maximum velocity needs correction or not, the distance that is travelled in time 'dt' (Δx_{dt}) is calculated and compared with the desired distance Δx:

$$\Delta x_{dt} = x(dt) - x(0) = c \cdot dt = \frac{v_{max} \cdot dt}{2} = \frac{\pi \cdot v_{max}^2}{2 \cdot a_{max}} \tag{2.4}$$

If: $\Delta x_{dt} > \Delta x \quad \Rightarrow \quad v_{max} = \sqrt{\frac{2 \cdot \Delta x \cdot a_{max}}{\pi}}$

If: $\Delta x_{dt} < \Delta x \quad \Rightarrow$ The time length of the plateau: $t_{pl} = \frac{\Delta x - \Delta x_{dt}}{v_{max}}$

2.2: Equalising the axes

The path that takes minimum time to move from x_0 to x_1 can be calculated for every axis (x, y and z). There will always be one axis, which needs the most time to complete the movement. This gives two possibilities for the other axes: they move as quick as they can, or they go slower and finish at the same time as the axis that takes longest.

Looking at precision [4], it will be better if the axes slow down. This is the option that is chosen for.

That means the maximum velocity has to change. To calculate the change in velocity, first the longest time has to be known and the relation from a chosen time to a maximum velocity:

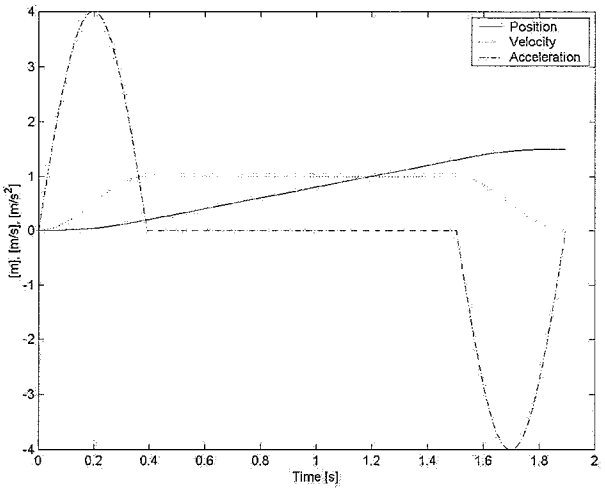


Figure 2.1: The movement, velocity and acceleration curves as generated by the setpoint program. A constant velocity part is inserted to reach the desired setpoint.

$$t_{total} = dt + t_{pl} = \frac{v_{max} \cdot \pi}{a_{max}} + \frac{\Delta x - \Delta x_{dt}}{v_{max}} \tag{2.5}$$

Substituting formula (2.4) into formula (2.5) gives:

$$t_{total} = \frac{v_{max} \cdot \pi}{a_{max}} + \left(\frac{x - \frac{\pi \cdot v_{max}^2}{2 \cdot a_{max}}}{v_{max}} \right) \tag{2.6}$$

Simplifying formula (2.6) gives:

$$t_{total} = \frac{v_{max} \cdot \pi}{2 \cdot a_{max}} + \frac{\Delta x}{v_{max}} \tag{2.7}$$

To give an expression for v_{max} as a function of t_{total}, formula (2.7) is multiplied by v_{max} and solved, leading to the following expression for v_{max}:

$$v_{max} = \frac{t_{total} \pm \sqrt{t_{total}^2 - \frac{2 \cdot \pi}{a_{max}} \cdot \Delta x}}{\frac{\pi}{a_{max}}} \tag{2.8}$$

If there is no constant velocity plateau, the velocity is first corrected (see paragraph 2.1) and the completion time is calculated by the definition for 'dt':

$$dt = \frac{\pi \cdot v_{max}}{a_{max}} \equiv t_{total} \tag{2.9}$$

Then a new v_{max} equals:

$$v_{max} = t_{total} \cdot \frac{a_{max}}{\pi} \qquad (2.10)$$

This can be done for every axis (except of course for the one that takes longest). In this way all the axes finish their movements at the same time.

The setpoints will be inserted by supplying some parameters. The only parameters that are necessary are the maximum velocity and the maximum acceleration for every axis, the start and end points, the magnet status and the desired time.

The desired time is used to make a movement longer than is calculated by the program. For example: if the robot must not move ($\Delta x = \Delta y = \Delta z = 0$) for a certain time to turn on the magnet and pick up something, the program calculates that move will cost 0.000 [s] and after 0.000 [s] it will start calculating the next move. So if a desired time is inserted, the program waits for that certain time, before continuing.

2.3: Practical problems

The method mentioned above and the way it is executed has some practical problems. The first one is the programming of the programs that will compute the trajectory. Those programs (see appendix 3) will calculate the values of the place, velocity and acceleration every time increment (e.g. 0.001 [s]). The problem here is to fit the different phases of the movement together, so that the transition is smooth and all the axes move the same amount of time.

This problem has been dealt with, by defining a maximum length of the matrix (e.g. 2.3 [s] gives a length of 2300 [-]) and when a matrix is too long, deleting the last values, when the matrix is too short, adding the last value several times.

The second problem is, by using those programs, every 0.001 [s] several values have to be send to the robot. That goal is not reachable, so another solution has to be found. That solution is to approximate the trajectory as described in 2.2 by simple equations and only send certain values for the movement to the robot.

2.4: Other method

The acceleration, velocity and position profiles will now be defined as shown in figure 2.2.

The steepness of the first part of the acceleration is called the jerk, J [ms^{-3}]. The equations of the different parts are given in appendix 2. To calculate the minimum time, in which a movement can take place, the following

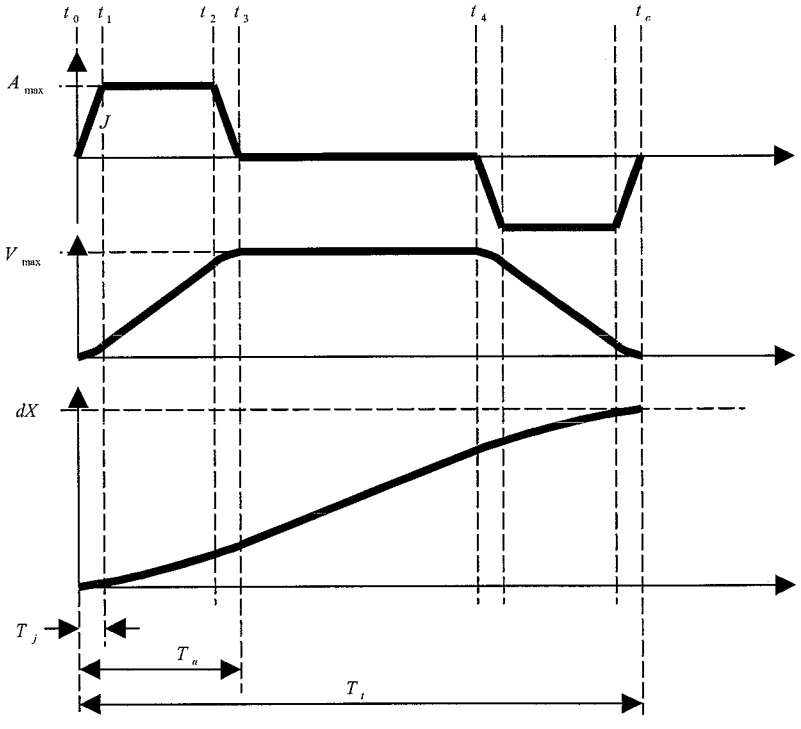


Figure 2.2: The new acceleration, velocity and placement profiles.

parameters are required: the distance (dx [m]), the jerk (J [ms^{-3}]), the maximum acceleration (a_{max} [ms^{-2}]) and the maximum velocity (v_{max} [ms^{-1}]). With use of appendix 2 follows:

$$T_j = \frac{a_{max}}{J} \Rightarrow T_a = \frac{v_{max}}{T_j J} + T_j \Rightarrow T_t = \frac{dx - J(T_j^2 T_a - T_a^2 T_j)}{J(T_a T_j - T_j^2)}$$

To make the three axes finish together, every axis uses the maximum T_a to accelerate. Then every axis uses the maximum time with constant velocity and finally every axis uses the same maximum deceleration time.

When $T_{a,max}$, $T_{j,max}$ and $T_{t,max}$ are known, the jerk, v_{max} and a_{max} for every axis can be calculated:

$$J_i = \frac{dx_i}{(T_a T_j - T_j^2)T_j + T_j^2 T_a - T_a^2 T_j} \quad , \quad a_{max,i} = JT_j \quad \text{and} \quad v_{max,i} = J(T_a T_j - T_j^2)$$

(with i = x, y or z)

Using this procedure, the only parameters that have to be send to the robot, are J_i , $a_{max,i}$, $v_{max,i}$ and dx_i.

Because of the relative simplicity, this is the way the movements of the robot are controlled.

Chapter 3: Controller structure

3.1: Introduction

In this case, the primary goal is to control a robot (Philips VRS). The trajectories (the path which the 'head' of the robot is supposed to follow) must be defined. The 'head' of the robot has to follow the designed path as fast and as accurate as possible. This fact shows the needs for a good controller. To control the robot, both feedforward and feedbackward are used. This chapter deals with the common structure of this controller. Later on, the different parts will be discussed more detailed in separate chapters.

3.2: Feedforward & Feedbackward

To actually make a motor follow a desired path, this motor has to be guided correctly. One could choose to provide the motor exactly with the current needed to perform the desired movement. Therefore a detailed model of the process has to be present. This method is known as *feedforward*. It is presented schematically in figure 3.1.

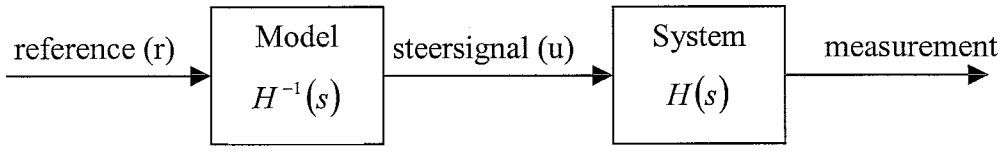


Figure 3.1: Blockdiagram of a feedforward system.

In practice, however, this method is almost never usable. A model is never perfect enough, and disturbances during the process will cause the motor to deviate from the given path. This is the reason that most systems have a *feedback*. The state of the motor (e.g. position) is being measured during the process and compared with the state it has been told to be at that time. When there is a difference, the PID-controller causes to steer the process back to the desired state.

This method is presented schematically in figure 3.2.

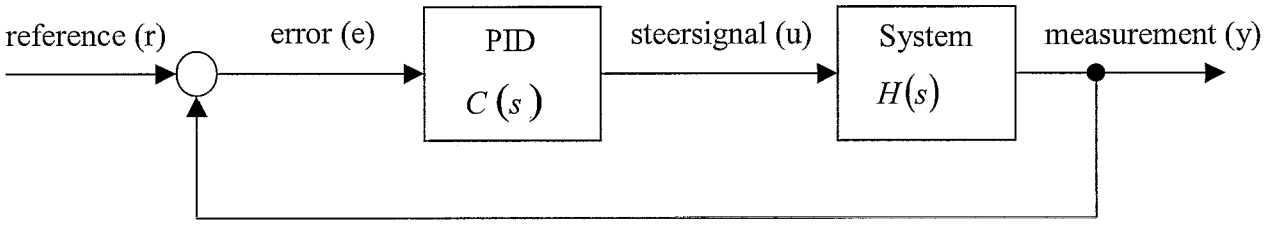


Figure 3.2: Blockdiagram of a feedback system (no feedforward)

Both methods discussed are not optimal. The feedforward method doesn't take into account any disturbances. If a position error has occurred it can't be recovered, because you don't measure, so you don't know how big the error is. The feedback method seems a lot better. It is, but also this method is not as optimal as one would like. When a strongly varying reference signal is given, the errors will be big all the time, which means that the PID-controller is always very busy controlling the system. It is better to use both methods. The feedforward method will be used to estimate a steering signal, based on the knowledge of the system dynamics. The errors, which will be present, can then be handled by the PID-controller. This means that a much

simpler PID-controller can be used. Also the feedforward doesn't need a perfect model. When it does have one, then the PID-controller will hardly be used. But practice proofs that even with little knowledge of the system dynamics, good results can be achieved [2]. Figure 3.3 shows the combined methods.

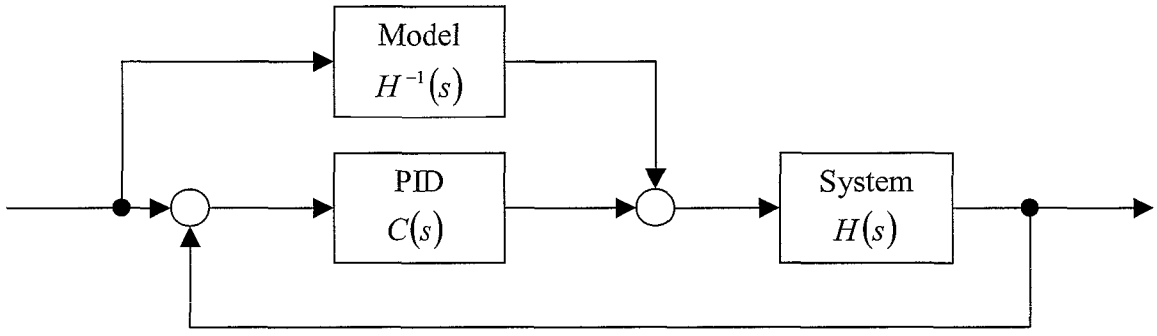


Figure 3.3: Blockdiagram of a combined feedforward-feedbackward system.

3.3: Common features of PID-controllers

The PID-controller is used to control the movements of a dynamic system. Input and output signals will be compared and a steering signal will be calculated, based on the difference between the desired reference and the actual (measured) state of the process.

Mechanical dynamic systems will always have friction in it. Friction corresponds with low frequencies in the frequency-response diagram. So, if one wants to compensate for friction, the amplitude gain at those low frequencies must be set high. In the bode-diagram of the controller, that will be seen as a descending line at low frequencies. A differential part will also be necessary in the controller structure. This part takes care of any stability problems. This action can be seen as an ascending line at high frequencies in the bode plot. When multiplying the differentiated error signal by a larger numbers for larger frequencies, the noise of the measurement will play a significant part. One wants to have the influence of any noise to be as low as possible. Therefore a low-pass filter will be highly advisable to get rid of high frequency noise signals before the controller comes into action. After all this we want the error in the input and output to be as low as possible. That means that in between the integral and the differential frequencies, another, proportional, action has to take place. When presenting the controller graphically, by means of a bode-plot, it will look like figure 3.4.

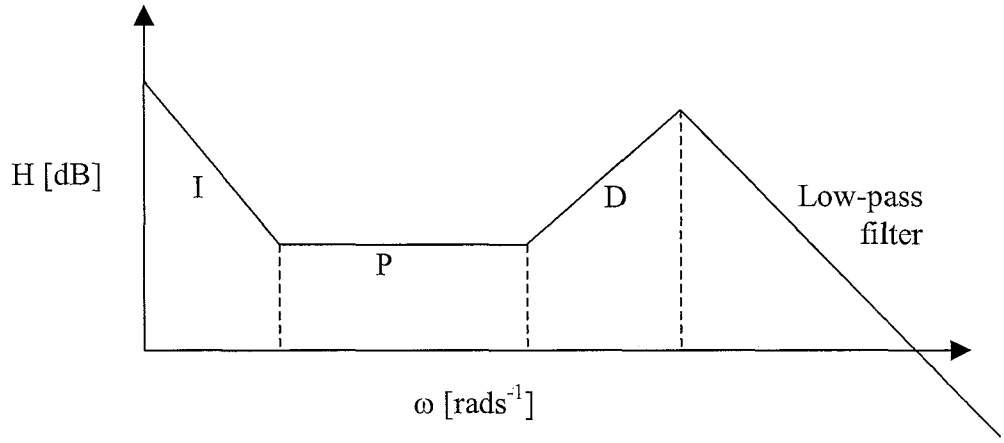


Figure 3.4: Bode plot of a controller.

Chapter 4: System Identification

4.1: Introduction

In order to implement the feedforward structure, one has to know some of the most important properties of the used system. The resonances for example are important to know. Also the friction could be an important issue. Because a feedforward is used in combination with a feedback loop, the knowledge needed in order to represent the system can be diminished to only the most important properties.

4.2: Theory

According to Koster [2], most mechanical systems can be thought of as 4^{th} order models. Friction, then, hasn't been taken into account. Later on, friction can be implemented in a model, which then will be in the form of "different" models for the different cases (i.e.: static & dynamic friction).

To estimate the parameters for the model used, one (of course) has to measure. In fact, the measurement solely is based on finding a transfer function of the system. In other words: an experimental *Bode plot* is to be found. Using one of the following methods [1], one can do that:

1) **Transient response** (using a step or impulse)
2) **Frequency response data** (using many different sinusoidal inputs)
3) **Stochastic steady-state information** (using information obtained during normal working operations)
4) **Pseudorandom-noise data** (noise has all frequencies)

The third method is not usable is this case, because it only can be used when the system is already running, and needs improvement by refining (upgrading) the existing model.

The second method seems a good one, only the time needed to find a good transfer function is too long when one needs a good model. When little damping is present, a lot of frequencies around the resonance peek have to be measured.

The first method also seems very easy and simple to perform. But when the damping is too big, one can get a bad output, and a good model then, is hard to get.

So the one remaining method is the fourth. A lot of experiments involving frequency-response data use this method. The equipment used in this case is MatLab/SigLab. SigLab is a piece of hardware that can be connected with the robot on the one side and with a personal computer on the other side. It generates a white-noise signal that is passed to the robot. Then the output is measured directly by the same hardware, and the provided software calculates transfer functions immediately. Once the transfer functions are known, a model can be fitted.

4.3: Used methods

The system can be represented as done in figure 4.1.

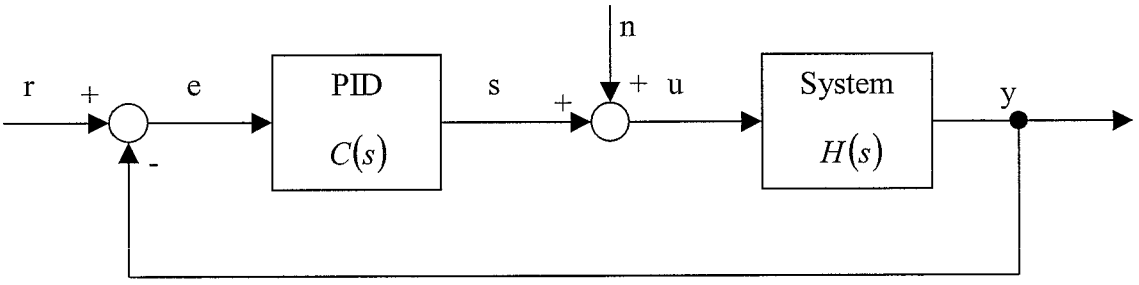


Figure 4.1 Block diagram of the system, used to obtain a transfer function.

The PID-controller used in the experiments is a simple, far from optimal controller. It is only used to perform calculations on the system. Those calculations will be discussed right now.

SigLab produces the noise signal n. The signals s, u and y can be measured. The reference r will be a straight line in space, to get rid of present (static) friction. The measurements will be used in order to determine $H(s)$. The following equations can be helpful in obtaining $H(s)$ [3].

$$\frac{y}{u} = H \tag{4.1}$$

$$\frac{y}{n} = \frac{H}{1+CH} = H_{ps} \tag{4.2}$$

$$\frac{u}{n} = \frac{1}{1+CH} = S \tag{4.3}$$

$$\frac{s}{n} = -\frac{CH}{1+CH} = -H_{cl} \tag{4.4}$$

The method presented with equation (4.3) is called *sensitivity measurement*. To obtain the $H(s)$ the following equation can be used:

$$CH = S^{-1} - 1 = H_{ol} \tag{4.5}$$

When performing *closed-loop measurements* like presented in equation (4.4), the calculation of $H(s)$ will go like the next formula:

$$CH = \frac{1}{H_{cl} - 1} = H_{ol} \tag{4.6}$$

4.4: Parameter estimates

After the measurements have been done, it is time to estimate the parameters. In MatLab there are several tools to do so (see: *System Identification Toolbox*). In appendix 4, the used MatLab scripts are given. In figure 4.2, the measured bode plot is fitted with a 12^{th} order model. This figure only shows the fit of one axis (the x-axis), the other axes use the same method.

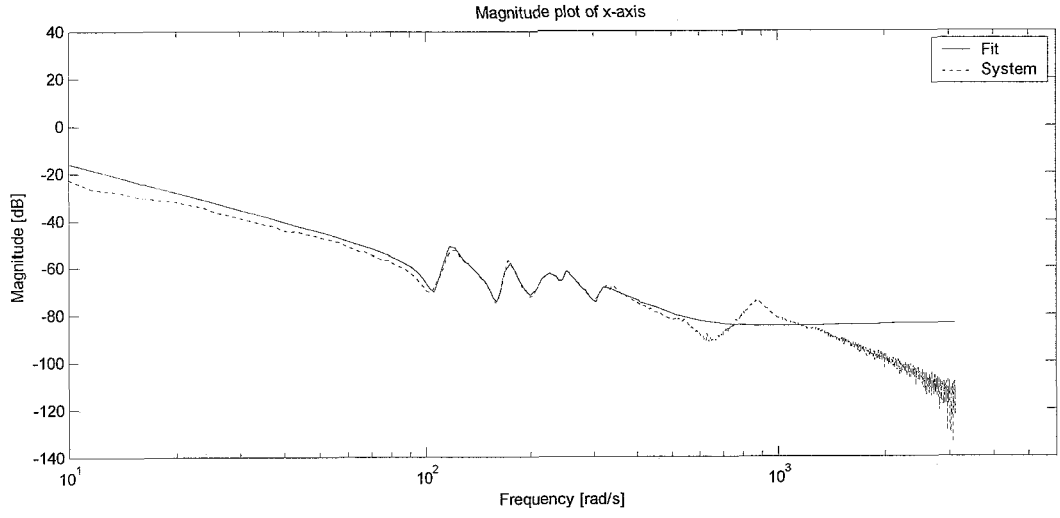


Figure 4.2: Magnitude plot of the x-axis. Both the system and the fit are drawn in the plot.

The program FITTEN.M (appendix 4) produces the fit. This program loads the bode plot obtained from the measurement and uses FRFIT.M [5] to fit. The output are two matrixes which possess the numerator and denominator of the fitted transferfunction (see also figure 4.2).

With this transfer function, the values of the parameters in the PID-controller can be calculated or found using Diet. More about this is written in chapter 5.

Due to lack of time to implement the found 12^{th} order model into hardware, a decision has been made to use a simple mass feedforward.

Instead of fitting a 12^{th} order model onto the frequency-response data, we now will fit the experimental data to the following simple equation:

$$H(s) = \frac{1}{ms^2} \tag{4.7}$$

This method is used for all the three axes. The obtained values for m are nothing else than representations of the mass of each robot axis. The mass for x, y and z are: 0.058 [kg], 0.044 [kg] and 0.023 [kg].

Chapter 5: Final control of the robot

5.1: Introduction

The dynamics of the robot are known, so the controller can be designed. Together with the feedforward loop, the controller forms the basis of the movement of the robot. First, theoretical calculations of the P, I and D parameters of the three axes are discussed, followed by the tuning of controller of the VRS-robot and finally the quality of the controller.

5.2: Controller basics

Stability of a system is an important issue. So the first thing to do is to research whether the system is stable. A general system can be represented by:

$$Y(s)=\frac{C(s)H(s)}{1+C(s)H(s)}R(s)+\frac{1}{1+C(s)H(s)}W(s) \qquad (5.1)$$

Where $R(s)$ is the reference signal, $W(s)$ the disturbance and $Y(s)$ the output signal. $C(s)$ is the transfer function of the controller and $H(s)$ the transfer function of the system.

For minimal placing error, the disturbance must be as low as possible. By making $C(s)$ very large, the value in front of $R(s)$ will be almost 1, so the desired reference will be obtained. With this knowledge, a proportional controller will be chosen. But if the gain P is enlarged, the disturbance at high frequencies (see figure 4.2, above 1000 [rad s^{-1}]) is also amplified. That is not desired. A problem of a different kind becomes clear when looking at figure 5.1.

In figure 5.1, a bode plot of two open loop $C(s)H(s)$-systems is plotted.

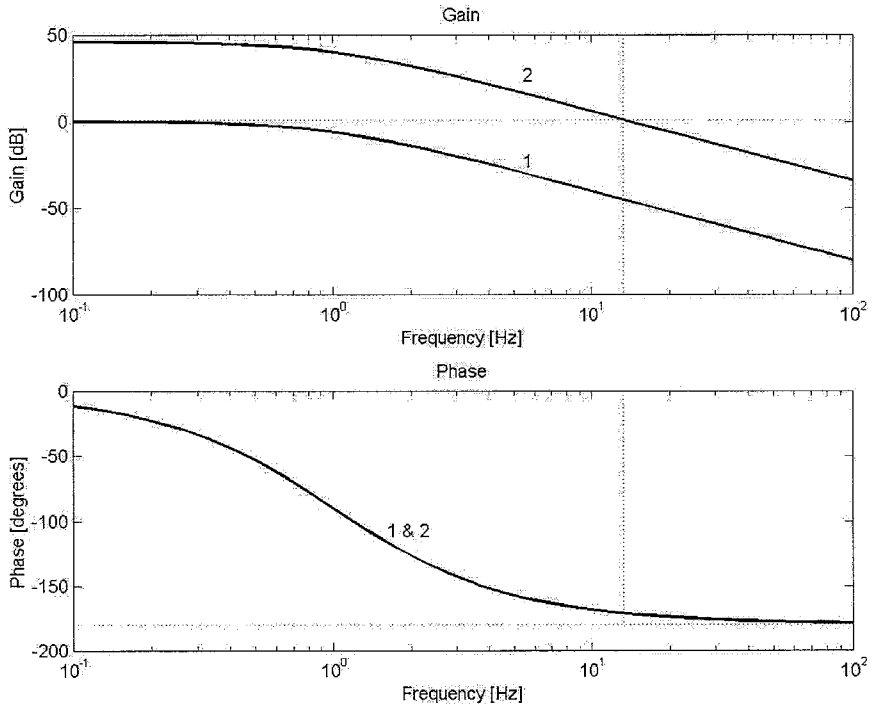


Figure 5.1: Bode plot of a 2nd order system (1) with a P-controller (2).

The P value of the second one is 200 times bigger than the P value of the first one. A problem arises when the gain equals 1 (= 0 [dB]) and the corresponding phase equals -180^0. Because then, the value of $C(s)H(s)$ equals -1. If this value is inserted in formula 5.1, the denominator goes to zero and the error goes to infinity! To prevent this from happening, a couple of rules have been set up. If the phase is smaller than -135^0 and the gain must be smaller than 0.5 (*gain margin*). If the gain is larger than 0.5, the phase must be larger than -135^0 (*phase margin*). If the gain and/or phase exceed these margins, the system becomes unstable. To check whether a system will be unstable, a Nyquist plot can be made.

So the value of the parameter P cannot be enlarged unlimited. To ensure the P value can be enlarged at least a bit, another controller has to be chosen.

A PID-controller is chosen, using the example $H(s)$ of figure 5.1, $C(s)H(s)$ now equals:

$$C(s)H(s)=\frac{Ds^2+Ps+I}{s}\cdot\frac{1}{s^2+2s+1}=\frac{Ds^2+Ps+I}{s^3+2s^2+s+0} \tag{5.2}$$

So rewriting equation (5.1) with use of the example, gives:

$$Y(s)=\frac{Ds^2+Ps+I}{s^3+(2+D)s^2+(P+1)s+I}R(s)+\frac{1}{s^3+(2+D)s^2+(P+1)s+I}W(s) \tag{5.3}$$

When looking at low frequencies (s $\approx$ 0) and a large value for I (D=0), formula (5.3) changes to:

$$Y(s)=\frac{I}{I}R(s)+\frac{1}{I}W(s)=R(s) \tag{5.4}$$

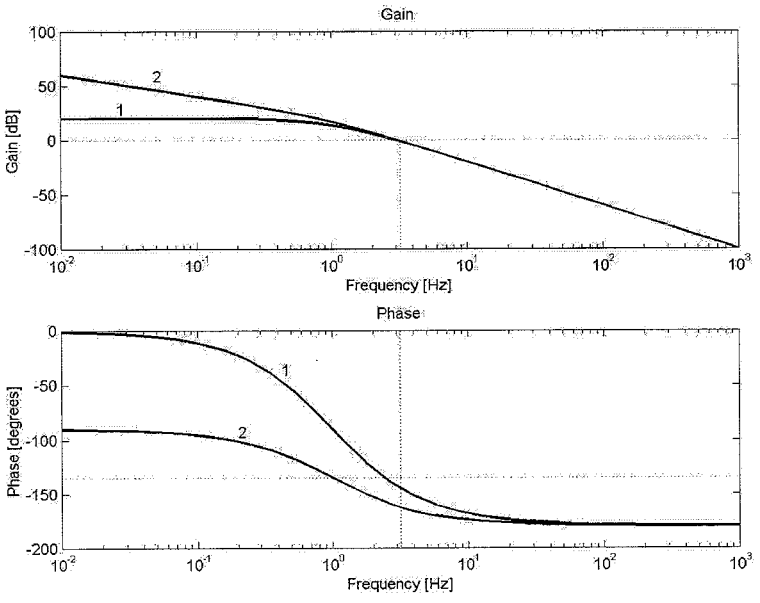


Figure 5.2: A bode plot of the 2nd order system(1) with a PI-controller(2)

Concluding from (5.4), a largest possible value of I is desired to neutralise the error caused by $W(s)$ for low frequencies. There is a limit to I. Because if I is too large and the gain of 1 is reached, the phase is more unfavourable (figure 5.2).

To compensate this, a differential action can be implemented in the controller. By using a D-action, the gain at higher frequencies will increase and also a positive phase jump will occur (see figure 5.3).

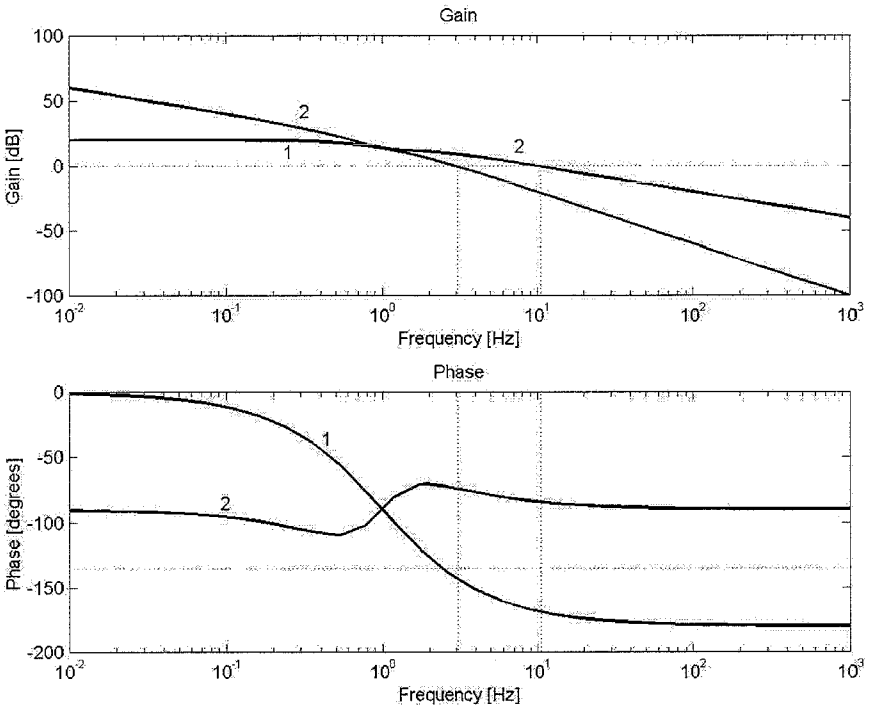


Figure 5.3: A bode plot of the 2nd order system(1). With a PID-controller(2)

Like all other parameters, the D cannot be increased unlimited, because the high frequencies are also increased and those frequencies are mostly noise.

5.3: Determination of the control parameters

The optimum feedback controller layout, will be a PID-controller with a lowpass filter. The block scheme of the controller is drawn in figure 5.4. The error in the position is fed to the controller. The signal of the controller is then passed trough a lowpass filter.

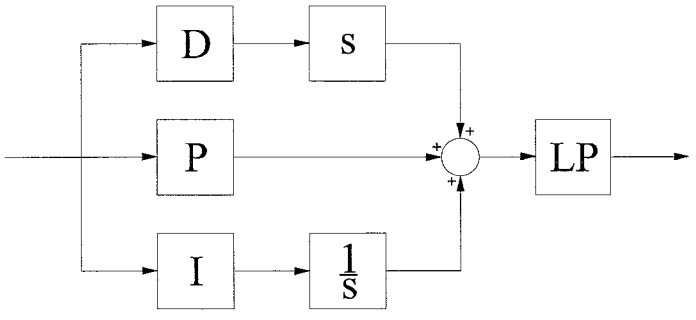


Figure 5.4: Blockscheme of the controller.

The starting frequency of the lowpass filter is not very hard to determine (see figure 4.2). Chosen is for a lowpass filter starting at 1200 [rad s^{-1}], with a damping of 0.5[-]. To see the influence of certain changes in the parameters of the controller, a program, called *Diet* [5] is used. To determine the other three parameters, some designer rules are followed [6, ASML]. Those rules assume a second order

system is used. Two values have to be known, the bandwidth (f_{bw} [Hz]) and the mass (m) of the structure, then [5]:

$$K_P = \frac{m \cdot (2\pi f_{bw})^2}{a} \quad , \quad f_d = \frac{f_{bw}}{a} \quad and \quad f_i = \frac{f_{bw}}{a^2} \quad with \quad a = \sqrt{10} \qquad (5.5)$$

The parameters P, D and I are determined by [5]:

$$P = \frac{K_P \cdot \tau_d}{\tau_i} + K_P, \; D = K_P \cdot \tau_d \; and \; I = \frac{K_P}{\tau_i} \; with: \tau_i = \frac{1}{2\pi f_i} \; and \; \tau_d = \frac{1}{2\pi f_d} \qquad (5.6)$$

The only thing to do when determining the control parameters is to determine the frequency at which the magnitude of the system response equals 0 [dB] and the mass of the system. This will be done for the x-axis, and the results for the y- and z-axis will be presented. The mass follows from chapter 4: m_x = 0.058 [kg]. A 2nd order system is described by:

$$\frac{1}{ms^2} = gain \Rightarrow \frac{1}{0.058 \cdot \omega^2} = 1 \Rightarrow \omega = 4.2 \text{ [rad/s]} \Rightarrow f_{bw} = 0.66 \text{ [Hz]}$$

This gives: P = 0.42, I = 0.13 and D = 0.24. Using these values for the controller, a terrible system is generated. So, assumed is an error is present in the formulas. The control parameters for the y-axis and z-axis are also calculated, but these parameters also gave terrible results. The error is not clear, so this method will not be used.

5.4: The controller in practice

As mentioned above, the calculated values of P, I and D for the three axes give terrible results. To tune the controller, first the values of the control-parameters of the x-axis are to be found, followed by the values of the parameters of the y-axis and z-axis. These values are found, looking at minimum error and stability. Once these values are obtained, one can check whether they are suitable for the overall trajectory.

Finding the values of the parameters is done using the trial and error method and using the knowledge of the influence of the different parameters.

First the x-axis is moved, using the same velocity as used in the final trajectory. It seemed that the value of P was much to low. The final value of P for this axis is 600. The value of the D-parameter also was to low. The final value is 4. Finally, the value of I was much too high, the system became unstable. The final value of I was 0.001. With these values of the control parameters, the maximum error in the position of the x-axis is about 5e-4 [m]. This error appeared when the velocity of the axis changed sign.

The same strategy is used for the y-axis. The final values of the control parameters are the following: P = 1200, D = 7 and I = 0.005.

The maximum error in position in the beginning of the movement was over 5e-4 [m]. When the axis moved a bit longer, the maximum error was 5e-4 [m]. A possible explanation for this may be that during the movement the axis is greased better. This maximum error also appeared when the velocity of the axis changed sign.

The final control parameters of the z-axis are the following: P = 750, D = 2 and I = 0.04. The maximum error in position is about 1e-4 [m].

Finally, the final trajectory is tested with the values of the parameters just found. The maximum error in position of the x-axis is 3e-4 [m]. The maximum error in position in the y-axis is 5e-4 [m] and the maximum error in position of the z-axis is 1e-4 [m].

5.5: Quality of the controller

As mentioned above, the maximum errors of the axes vary from 1e-4 [m] to 5e-4 [m], measured at the axis. Comparing this to the resolution of the encoders, placed on the axis (about 5e-6 [m]), it can be said that our PID-controller combined with a simple mass-feedforward works well. Better results can be obtained using a better feedforward, e.g. including the friction and of course finding better control parameters for the feedbackward part.

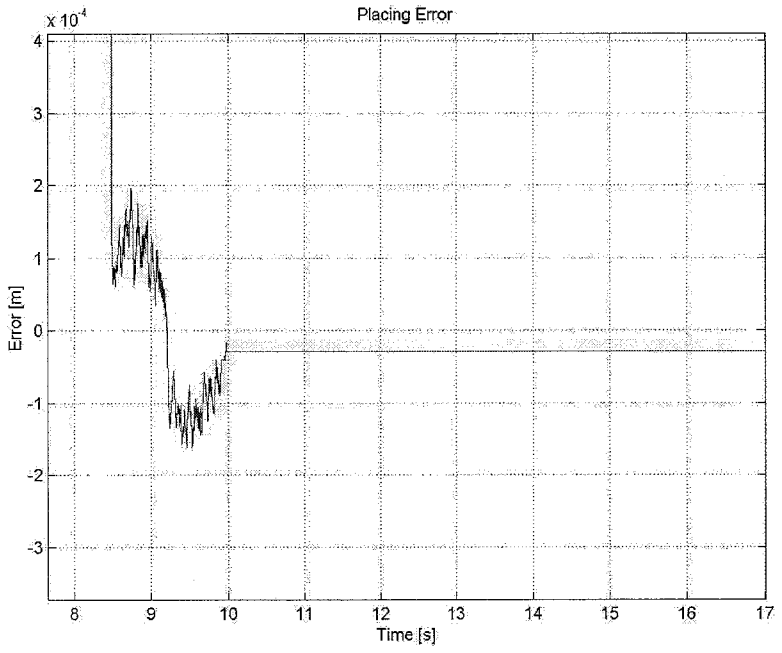


Figure 5.5: A representation of a placing error. The vertical line left is produced by the fact that the reference has been calculated, but the robot is still initialising.

Chapter 6: Hardware introduction

Planning for the project took a full week as we sourced for the type of micro controllers and other peripheral chips to employ in the implementation of the control box. Finally, we settled on using the ATMEL 89C52 as our micro controller, the AD1674 as a single input channel 12-bit ADC, and the AD667 as a single channel 12-bit output DAC. Our choice was much limited by the availability of the chips and their support.

Upon the completion of planning, we moved on to the breadboard implementation of our project. Once the design is verified to be working on 12-bit significance on the breadboard, we proceeded to fabricate the PCB. The Windows Interface Programme was being developed concurrently with the hardware developments.

The final stage of the project involved the interfacing of the micro controller with the Windows programme running on the PC. The workability of our project is also validated through the testing of the proportional and integral control.

Workload of the project was evenly split between all 3 members of the NUS team. Ee Cheng Tien undertook the hardware implementation of the Control Box, Paul Ng coded the Windows Interface Programme, while James Leong worked on the programming of the micro controller and the execution of the PID control.

Chapter 7: Hardware

7.1: Introduction

In many control applications, a hardware section is necessary to serve as an interface between the machine and the user input stage, which may be a LCD screen and a keypad, or which may be a personal computer. This section allows for translation of user intent into machine language that gives the machine instructions to actualise the intention.

7.2: Objectives

The objectives of the hardware section is as follows:

1. Translate physical position of robot arm from a given voltage value into a numerical value while having 12-bit accuracy

2. Microcontroller onboard receives ideal setpoints from personal computer via parallel port, compares with physical positions feedback, implements feedforward control, and determines output voltage using control equations

3. Translate digital output values into voltage at 12-bit accuracy

4. Allow for communications with a personal computer via the parallel port

5. To provide for a stable and reliable power supply for the resulting circuit, taking into account the mains characteristics at the TUE side.

7.3: Overview of Hardware

Figure 7.1 shows the schematics of the control box.

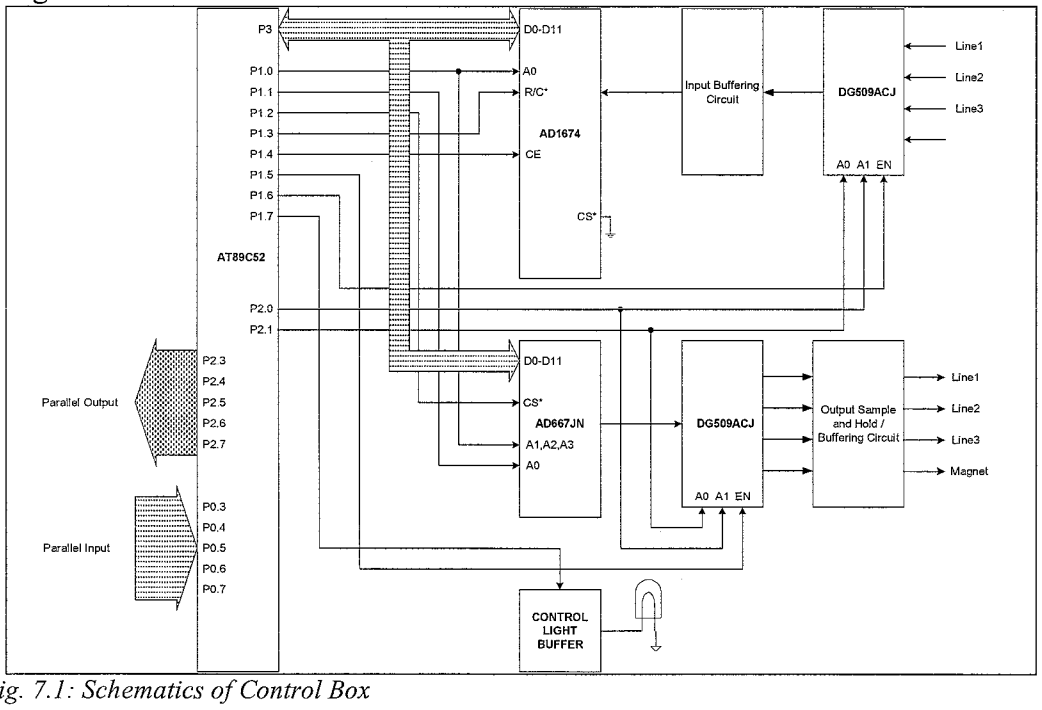


Fig. 7.1: Schematics of Control Box

A brief description of the operation (per cycle) is as follows:

1. The microcontroller selects the appropriate input channel, which feeds the analogue-to-digital converter (ADC) with a particular voltage value.

2. The controller then initiates the conversion, and reads in the 12-bit value given by the ADC.

3. After implementing the control equations, the next desired voltage value (in 12-bit form) is fed to the digital-to-analogue converter (DAC)

4. Conversion from the 12-bit value to an actual voltage is initiated by the controller, which also selects the appropriate channel to output the voltage.

The detailed description of the different sections is available in the following chapters.

7.4: Input of Robot Position in Voltage

Figure 7.2 gives the basic schematics of the input stage.

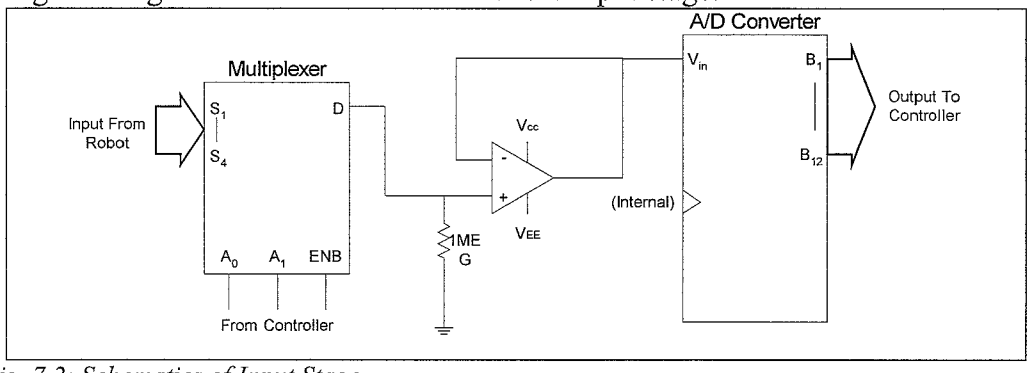


Fig. 7.2: Schematics of Input Stage

Figure 7.3 provides the detailed setup of the input stage.

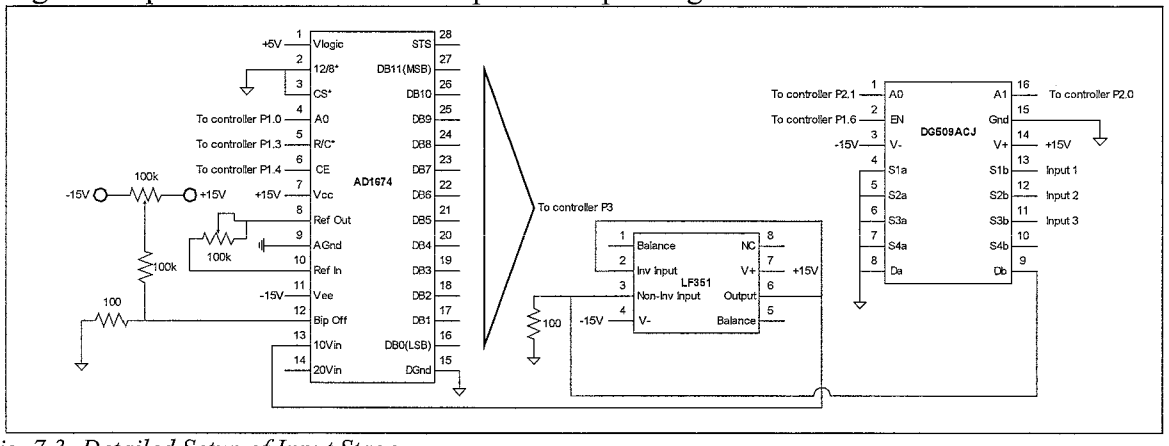


Fig. 7.3: Detailed Setup of Input Stage

The analogue multiplexer (DG509ACJ) allows the controller chip to select the correct voltage to be converted by the ADC (AD1674). This is done via the address pins A0 and A1 as shown in Table 7.1 below:

Table 7.1: Channels Selected Using A0, A1

A0	A1	Channel
0	0	1
0	1	2
1	0	3
1	1	4

Channel 4 is not in use. DG509ACJ, being a CMOS multiplexer, is bi-directional.

An operational amplifier, LF351, is used as an input buffer to the circuit. The high set point frequency and the possible large differences in voltage representations of the different axes meant that the voltage input to the ADC might be subject to large swings within short periods of time. The LF351, as a voltage follower, can successfully serve its purpose, as its slew rate is high (13V/μs).

AD1674 has an accuracy of 12-bits. By setting pin 2 (12/8*) to low the output of the ADC is in byte format, thus allowing it to interface with the controller using a byte-wide data bus. The controller retrieves the desired data output (most significant byte or least significant nibble) by pulling pin 4 (A0) to high or low.

The voltage range is selected to be from 0 to 10V by inputting the voltage at pin 13 (10Vin) whilst leaving pin 14 (20Vin) floating. Conversion is initiated using pin 5 (R/C*). AD1674 has an inbuilt voltage reference, which can be adjusted for maximum accuracy using external resistors and trimmers as shown in Figure 7.3.

7.5: Microcontroller

The controller used is Atmel's AT89C52. It is an 8-bit controller with 32 IO pins, and the inbuilt FLASH memory removes the need for external EPROM. The chip is set up as shown in figure 7.4.

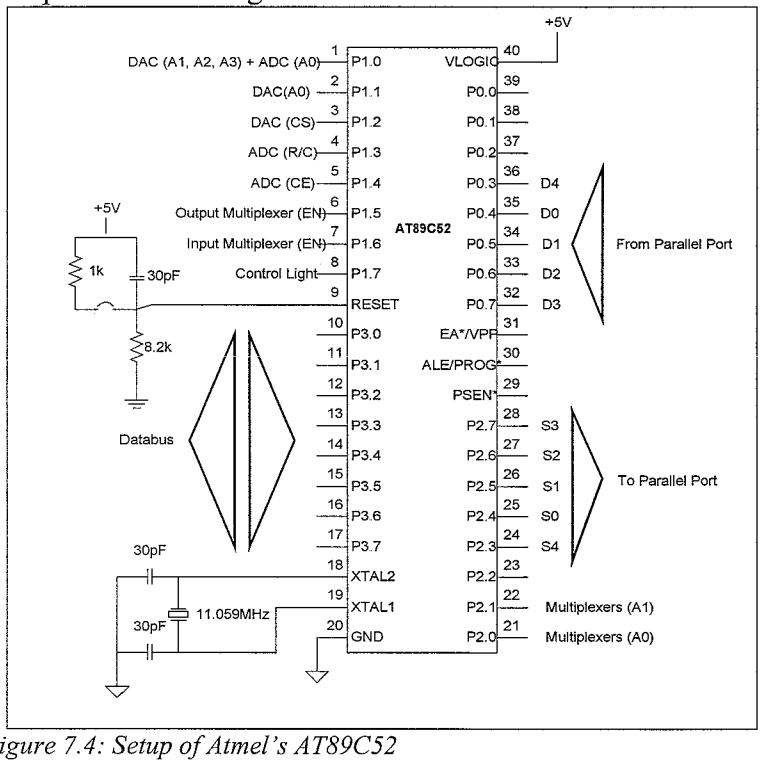


Figure 7.4: Setup of Atmel's AT89C52

Port 1 of the chip serves mainly as the "control bus" of the system. It is a bi-directional IO port that can receive input digital values as well as output data. All the pins used in this "control bus" is pulled up using external 10kΩ resistors, as they are not always able to drive their respective receiving pins, which may be more than one per output pin.

Ports 2 and 3 are bi-directional like Port 1. Port 3 acts as the "data bus", providing values for conversion into analogue voltages at the DAC and receiving values for the outputs from the ADC. 5 IO pins of Port 2 communicate with the PC through the parallel port; they output data.

Port 0, on the other hand, when serving as output pins, enter the high-impedance state. They are therefore unable to drive any output without the use of external pull-up resistors. It is for this reason that 5 of them are chosen to receive inputs from the parallel port.

7.6: Output in Voltage

The overview of the output section is as shown in figure 7.5.

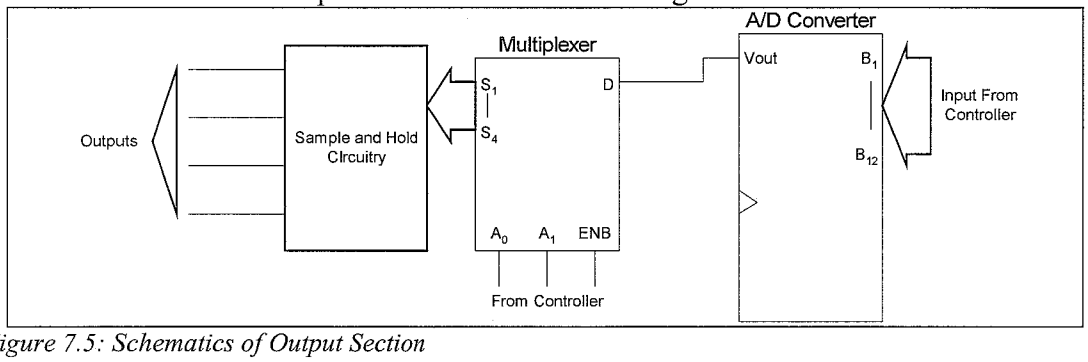


Figure 7.5: Schematics of Output Section

Figure 7.6 provides the detailed setup of the output stage:

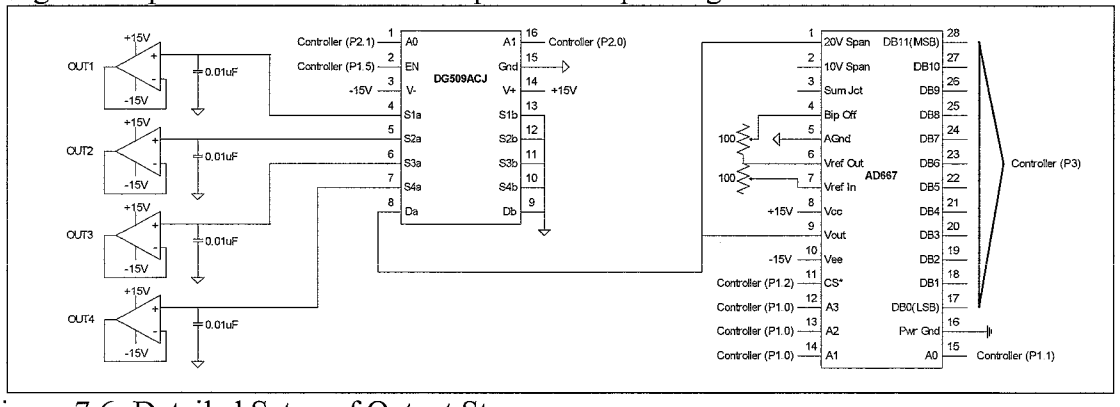


Figure 7.6: Detailed Setup of Output Stage

The digital-to-analogue converter (DAC) AD667 receives 12 data bits from the controller in 2 cycles. In the first cycle, A0 is set at logic 0 while A1, A2 and A3 are at logic 1. This enables the 4 LSBs of to be latched onto the chip. In the next cycle A0 is set to 1, while A1, A2 and A3 are set at 0. The 8 MSBs are then latched. Conversion is initiated during the second cycle.

As an output of ±10V is required, pins 1 and 9 are connected together, while pins 4 and 6 are connected via a 100Ω trim-resistor. The internal voltage reference of the AD667 is used and is adjusted using the 100Ω trim-resistor between pins 6 and 7.

The output multiplexer, like its input counterpart, has 4 channels that are controlled by the AT89C52. Since the output of an inactive channel is left floating, it will not be able to sustain the voltage level when it was active. Hence a sort of sample and hold circuit is required and is provided by the 4 op-amps. When the particular op-amp's channel is selected, the corresponding capacitor at the non-inverting input of the op-amp will be charged up to the current voltage. This capacitor serves to "maintain" the voltage level when the multiplexer switches to other output channels. The setpoint frequency is sufficiently high enough to bring about insignificant capacitor discharge during the times when the channels are inactive.

The op-amps also serve as output buffers to allow for maximum voltage drop across the load. They come in the form of TL074, which has a high slew rate as well as low offset voltage errors.

7.7: PC Parallel Port Connection

A laplink cable is used to connect the control board to the personal computer. Table 7.2 shows the connections between the pins of the micro controller and the parallel port pins. The pin layouts follow that of the DB-25 standard.

Table 7.2: Microcontroller, Laplink Cable and Parallel Port Connections

Controller Pin	Laplink Pin (Control Box End)	Laplink Pin (PC End)	PC Parallel Port Pin Description
P0.3	11	6	Data Pin 4
P0.4	15	2	Data Pin 0
P0.5	13	3	Data Pin 1
P0.6	12	4	Data Pin 2
P0.7	10	5	Data Pin 3
P2.3	6	11	Status Pin 7
P2.4	2	15	Status Pin 3
P2.5	3	13	Status Pin 4
P2.6	4	12	Status Pin 5
P2.7	5	10	Status Pin 6

Trials were carried out to determine if the controller and parallel port outputs can drive the other's inputs directly. Since it was found that there were no problems doing so there was no need for buffers between the two.

7.8: Power Supply

The power supply was designed with the following considerations:

- It must be able to supply the requirements of the circuit
- It must function as intended to at the TUE side
- It does not need to be mobile

The following figure (Fig. 7.7) illustrates the general concept of the power supply:

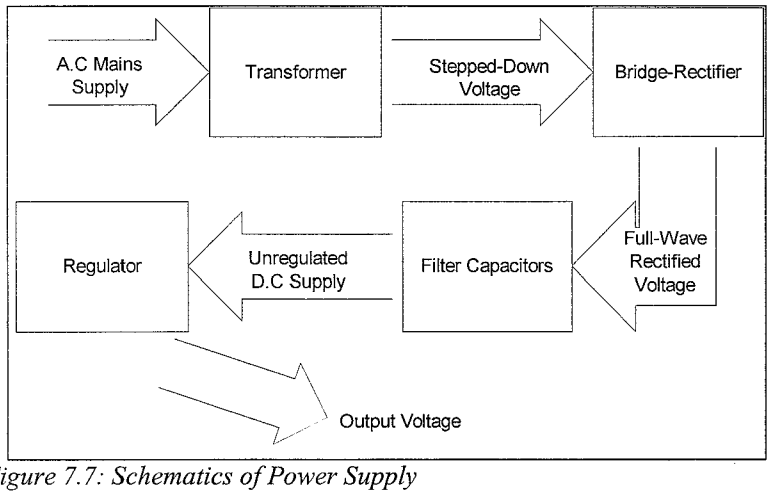


Figure 7.7: Schematics of Power Supply

Figure 7.8 provides the detailed setup of the power supply:

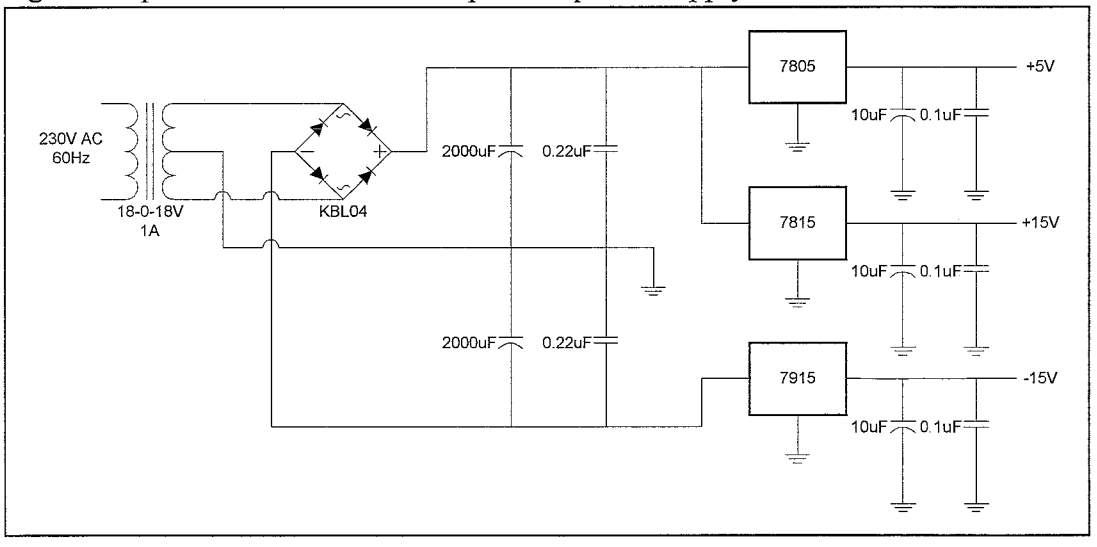


Figure 7.8: Detailed Setup of Power Supply

Using a laboratory's D.C. supply, the following currents were drawn from their respective sources (Table 7.3):

Table 7.3: Current Drawn From Respective Sources

Supply / V	Current Drawn / A
+5	0.044
+15	0.039
-15	-0.060

Thus, a rough gauge of the total power required is:

$$\text{Total Power} = 5 \times 0.044 + 15 \times 0.039 + (-15) \times (-0.060)$$
$$= 1.705 \text{ W}$$

The transformer, which has an output of 18-0-18V, 1A, is therefore able to supply the required power.

The peak voltage values at the output terminals of the bridge rectifier are

$$= \pm 18 \times \sqrt{2}$$
$$= \pm 25.45\text{V}$$

Since the maximum input voltage to the regulators 7805 and 7815 is 35V, and the minimum to 7915 is –35V, the transformer will not pose a problem.

With the mains supply at 230V, and the output at 18V, the step-down ratio is thus

$$= 18 / 230$$
$$= 0.0782$$

A mains supply of 220V will therefore result in the output voltage of

$$= 220 \times 0.0782$$
$$= 17.22\text{V}$$

The peak voltages are then

$$= \pm 17.22 \times \sqrt{2}$$
$$= \pm 24.34\text{V}$$

The values are sufficiently high enough in magnitude for the 7805, 7815 and 7915 to function properly.

The filtering capacitors serve to smoothen the ripples to a certain extent, which is required for the regulator chips to function as intended.

The capacitors at the outputs of the regulators reduce the voltage fluctuations due to the loading effect, thus providing a more stable supply. They also serve to reduce the noise level that may have been induced by the circuitry.

7.9: Discussion

7.9.1: Offset Errors

It was noted that there were offset errors incurred due to the op-amps not having perfectly matched input transistors. The errors are of mV magnitude and are within the specifications provided by the manufacturer. Complete reduction of this offset is considered but the following points were noted:

1. Offset trimming does not eradicate the problem completely since the offset drifts with temperature differences.

2. An automatic-adjusting circuit will take up more space than the rest of the circuit, which is impractical.

3. Very high quality CMOS op-amps incur high costs.

7.9.2: Noise

The merging of the analogue and digital +5V supply (i.e. they were not separated at the main board side) had no negative effects on the system. As such, no provisions were made for separate supplies. Furthermore, to reduce the level of noise throughout the system, decoupling capacitors were used near the power supplies to the chips.

7.10: Conclusion

The control board is able to effectively translate the robot arm's position into a binary value at 12-bit accuracy, and accurately convert 12-bit values into voltages. However, it was noted that the overall precision of the system would still depend on the module that provides the input voltages, as well as that which receives and translates the output voltages into torque of the motors.

The power supply was made with a slightly different mains supply (220V, 50Hz instead of 230V, 60Hz) in mind, and was designed to operate at TUE.

Chapter 8: Software For Micro Controller

8.1: Advantages of using ATMEL

After much thought, our group decided on using the ATMEL 89C52 micro controller. This is because the 89C52 possess 8K of programmable FLASH. This allows easy programming of the chip without interfacing it with additional ROM. 8K of programmable memory is also sufficient for our purpose.

Besides FLASH, the 89C52 micro controller also has built in IO ports and timers. This means that we will not need to interface it with external timers IO devices, resulting in easier hardware implementation.

However, the 89C52 being a 8-bit controller does have its limitations. As our ADC and DAC are of a 12-bit nature, it is preferable to have a 16 bit micro controller to store and manipulate the inputs and outputs to these devices. As a matter of fact, we need more than 50 instruction cycles to perform a multiplication instruction of two 16 bits number. As a result, we have decided to do adopt a more efficient solution of solving of the control equations within the PC instead. This will reduce the micro-controllers computational requirements, thus allowing a much faster sampling rate.

8.2: ATMEL 89C52

8.2.1: Pin Configuration

The ATMEL 89C52 is a micro controller with a total of 32 I/O pins in four IO ports(P0-P3). A pin in each port is referenced by its port number followed by its pin number, i.e. the third IO pin of port 3 will be P3.3. Of these 32 pins, we will be using 8 for data transfer to the ADC and DAC, 10 for parallel port and another 10 for control signals. The pin configuration are as below:

Table 8.1: Pin Configuration

Pins	Description
P0.3 - P0.7	Five input pins to receive data from the parallel port
P1.0	A0 of DAC and ADC
P1.1	A1 of DAC
P1.2	Chip Select of DAC
P1.3	Read/Convert of ADC
P1.4	Chip Enable of ADC
P1.5	DAC Multiplexer Enable
P1.6	ADC Multiplexer Enable
P1.7	Light of Control Box
P2.0	A1 of multiplexer
P2.1	A0 of multiplexer
P2.3 - P2.7	Five output pins to transfer data to parallel port
P3.0 – P3.7	Data bus for data transfer with ADC and DAC

8.2.2: Timers and Interrupt

Similar to other micro controllers, ATMEL 89C52 does possess its own set of timers and interrupt. It has a total of 7 interrupts as seen below:

Table 8.2: Interrupt Table

Interrupt	Vector Address
System Reset	0000h
External 0	0003h
Timer 0	000bh
External 1	0013h
Timer 1	001bh
Serial Port	0023h
Timer 2	002bh

Of these seven, we only use the timer 0 interrupt. This is necessary to keep track of the amount of time spent keeping the multiplexer open for the voltage to charge up the required capacitor.

The AT89C52 has a total of 3 timers. All these can be configured to operate as either timers or event counters. As a timer, the register is incremented every machine cycle. Thus the register counts machine cycles. As each machine cycle consists of 12 oscillatory periods, the count rate is 1/12 of the oscillatory frequency. As event counters, a count is incremented in response to a transition from 1-to-0 on a corresponding external input pin. For our case, we will be using the timers solely as a timing device.

By configuring the timer to Mode 2, we have configured the timer register as an 8-bit Counter with automatic reload. This enables the timer to trigger off an interrupt at a specified period of time determined by the 8 bit number that is reloaded and the oscillatory frequency. An interrupt is triggered when the reload value has been incremented to 256. Keeping the multiplexer open for a period of 0.05ms per sample cycle of frequency 1kHz,

$$\text{No of machine cycles required} = \frac{0.05\times10^{-3}}{12 \div OscillatoryFrequency}$$
$$= 0.05\times10^{-3}\times11.059\times10^{6} \div 12$$
$$= 46$$

$$\text{Reload value} = 256 - 46$$
$$= 210$$
$$= d2h$$

8.2.3: Memory Model

AT89C52 has a total of 8K FLASH programme memory. This memory is used to house the programme code for our micro controller. Thus there is no need to interface any additional ROM to our controller. Besides FLASH, AT89C52 also provides a 256 bytes of RAM for data memory. The memory structure is as below:

Figure 7. Internal Data Memory

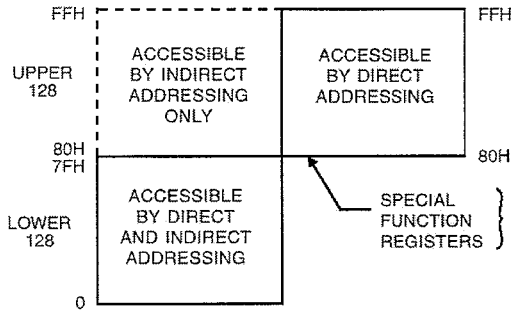


Fig 8.1: Internal RAM diagram

The lower 128 bytes consists of only directly addressable RAM, while the upper 128 bytes consists of both directly addressable and indirectly addressable RAM. Basically, only the lower 128 bytes and the upper 128 bytes(indirectly addressable) are used for data storage. The directly addressable upper 128 bytes generally constitute the Special Function Registers(SFR) of the micro controller and are used to control various devices and functions of the AT89C52 itself, i.e. timer, interrupts, etc.

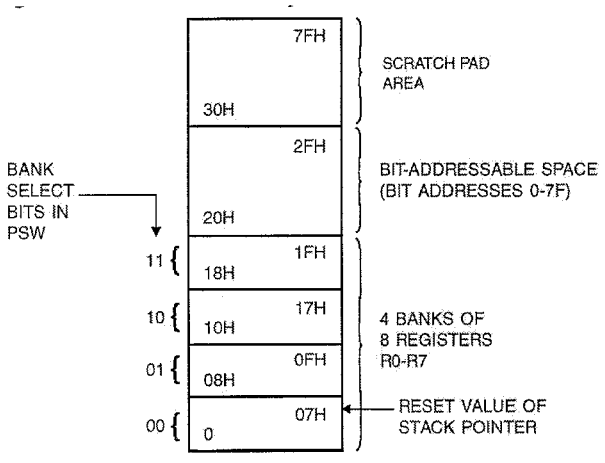


Fig 8.2: Lower Directly Addressable Memory Bank(128 bytes)

In our programme, of the lower 128 bytes, the first 8 bytes(00 to 07h) are used for register bank 0 with registers from R0-R7. These registers are used extensively in the programme. As there is no further need for additional register banks, the stack pointer points to 08h. Memory from 08h to 2fh is specially reserved for the operation of the stack. This will ensure that the micro controller knows where to return after function calls and interrupt requests. Memory from 30h-3bh is used for data storage of output voltage and the location of the robot. As we need 3 sets of values for both output voltage and location values(3 axis), a total of 6 variables need to be stored. As each of these variables have 12-bit significance, each variable will take up 2 bytes. As such 12 bytes(30h-3bh) are allocated for the storage of such data.

8.3: Programming Logic and Flow Chart of Micro Controller

The programme will be appended as appendix 5 in this report. In this section, we will be focussing on the explanation of our programme logic in a general fashion through the use of block diagrams.

8.3.1: Overview of Programme

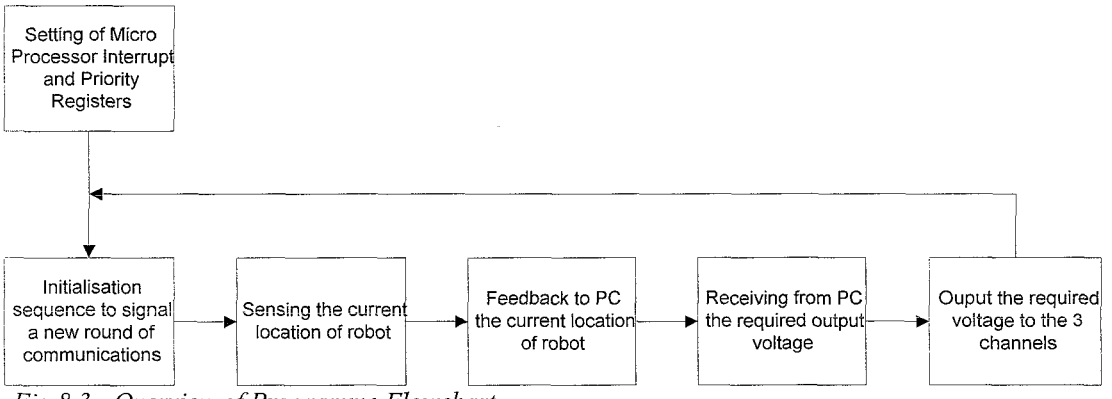


Fig 8.3: Overview of Programme Flowchart

Upon power on, the micro controller jumps to a start up sequence to initialise the interrupt enable and priority registers of timer 0. This is to ensure proper timing for the enabling and disabling of the multiplexer.

An initialisation sequence between the controller and PC then takes place. These sequences ensure that both the PC and micro controller are ready for a new round of communication. It helps to synchronise the data transfer between the controller and the PC. This also serves as a place for the micro controller to stall to await for the next cycle as the PC is keeping the timing of the whole system. The micro controller will stall in this sequence till the PC decides to initialise it for a new sampling cycle.

The location of the robot is then sensed through the ADC and stored in memory location 36h – 3bh. These values will be passed via the parallel port to the PC. The PC upon receiving the positional values will calculate the required output voltage and output it to the micro controller. The micron controller will store the output voltages in memory location 30h-35h, before passing these voltage values to the DAC to be outputted to the 3 channels

8.3.2: Initialisation Sequence

The initialisation sequence generally consists of receiving a sequence of 3 five bit numbers from the PC, 01010, 10101, and 11111. Upon receipt of the 01010, the microprocessor replies with a 01010. Upon receipt of 10101, the microprocessor replies with 10101. Upon receipt of 11111, the sequence is complete, and the microprocessor moves on to do its next stage of sensing the robot's location.

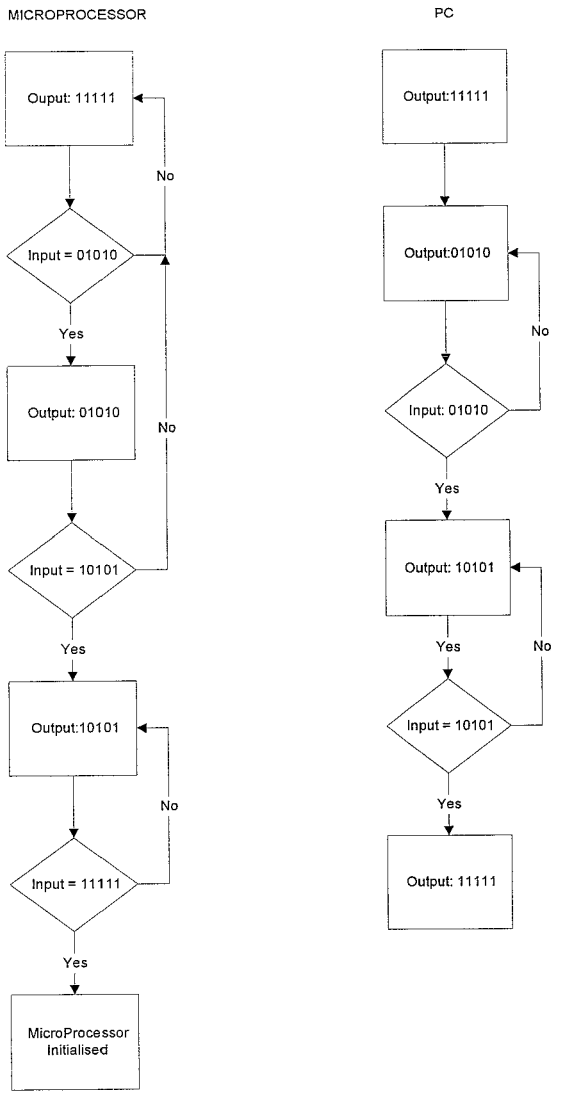


Fig 8.4: Initialisation Sequence Flowchart

8.3.3: Location sensing

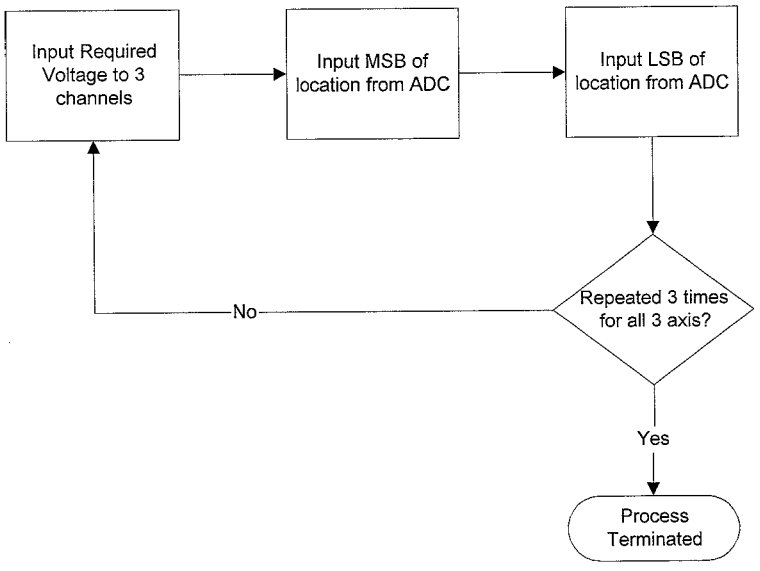


Fig 8.5: ADC Reading of Location Flowchart

The input voltage of each channel is sensed individually by controlling the voltage input into the ADC via the multiplexer. As 8 pins are used for the data transfer, we need read twice from the ADC for each channel. First reading will take in the 8 most significant bit, while the second reading will take in the last 4 bits. The locations are stored in six bytes in memory locations 36h-3bh.

8.3.4: Output to PC

Upon sensing the location of the robot, the micro controller will proceed to feedback these locations to the PC. Output to the PC involves 6 pins. Five output pins and one input status pin. Of the five pins, one pin is used as the output status pin while the other 4 is used to contain the data. As such, each byte is relayed in the form of 2 nibbles to the PC.

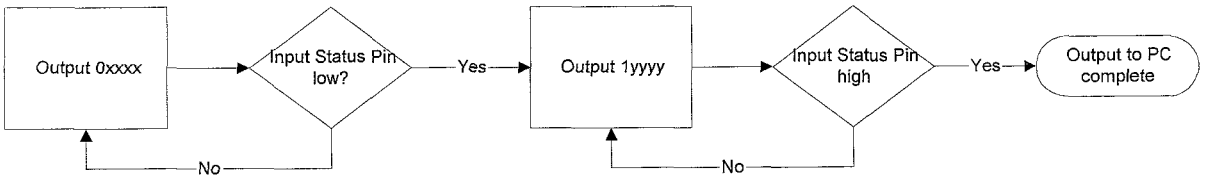


Fig 8.6: Output to PC Flowchart

To signal the beginning of a new byte, the output status pin is toggled low, and the data is put in the other 4 bits(denoted by xxxx in the diagram). The controller will then poll its input status pin continuously. The PC upon receipt of the data will drive the input status pin low. This signifies the transmission of one nibble successfully. The controller will then drive its output status pin high and put the second nibble in the other 4 bits(denoted by yyyy in the diagram). Upon receipt of the second nibble, the PC will drive the input status pin high. This signifies the complete transmission of a byte.

This process is repeated a total of six times, as six bytes(containing the X, Y and Z positions) need to be transmitted over to the PC before it can calculate the output voltage required.

8.3.5: Input from PC

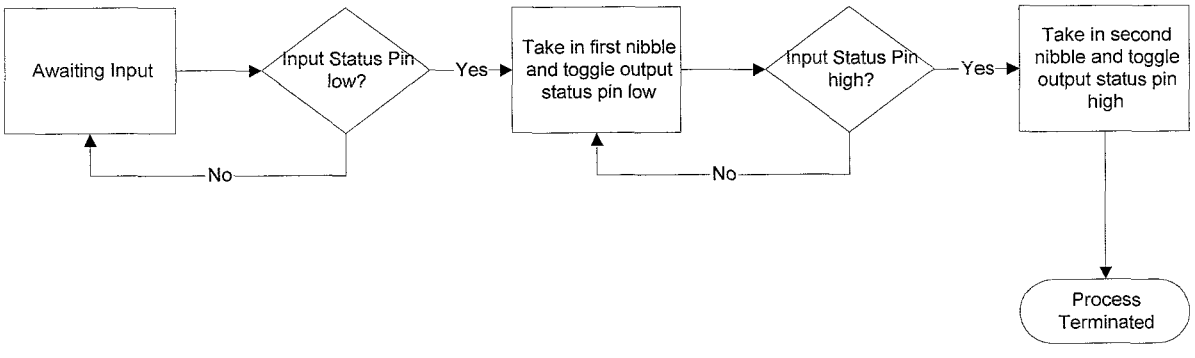


Fig 8.7: Input to PC Flowchart

The input sequence from PC takes in the required output voltage and stores it in memory location 30h-35h. It is similar to the output sequence, involving a total of six pins. 1 output status pin, 4 input data pins and 1 input status pin.

The controller keeps polling the input status pin. When the input status pin turns low, the micro controller takes in the first 4 bits as the more significant nibble and toggles its output status pin low to acknowledge. It will then keep polling the input status pin. When the input status pin turns high, the micro controller will take in the next 4 bits as the less significant nibble and then toggle the output status pin high.

This sequence of inputs is repeated six times, as the voltage output is stored in six bytes.

8.3.6: Voltage Output

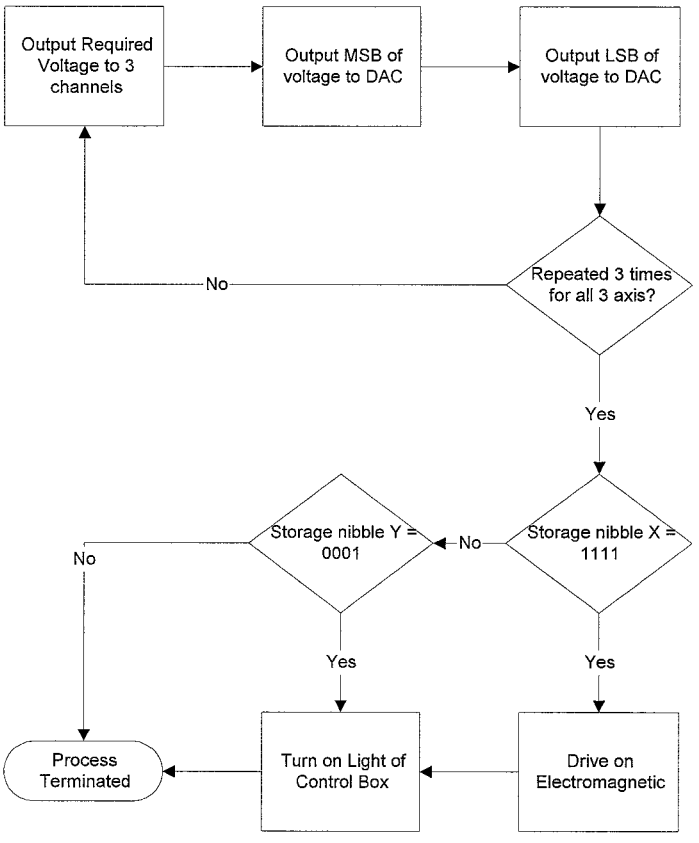


Fig 8.8: Output Voltage Flowchart

The voltage is sent out to the DAC in 2 bytes. The most significant 8 bits are sent in the first byte before the least significant 4 bits in the second byte as seen below. This process will be repeated 3 times till all three channels are fed with the appropriate output voltage.

Byte 1		Byte 2	
Data Bits			Control Bits
1111	1111	1111	0001

Fig 8.9: Byte Arrangement Diagram

As the data is only of a 12-bit nature, there remains 4 unused bits in the second byte(see figure 8 above). These remaining 4 bits are used for the control of the light in the control box and the electromagnet. Should it be 1111, both the electromagnet and the control light will be turned on. Should it be 0001, only the control light will be turned on. Any other signals will turn off both the magnet and the control light.

Thus the PC through the six bytes of output can control not only the output voltages, but also the solenoid and the control light.

8.4: Future Improvements

Upon the completion of the project, it is felt that the system could better be improved had the channels been worked on separately, one by one. This is so as it will enable the multiplexer to be opened for a longer time to allow the capacitor to charge up properly. Currently each channel's capacitor is only allocated a time of 0.05ms to charge.

As all channel are processed simultaneously, the multiplexer cannot be opened while the controller is in its communication phase with the PC. It can only do so in the one dedicated phase(highlighted in yellow) to output the voltage. This means that we cannot allow the multiplexer to open for too long a time as we have to complete the whole process in under 1 ms to satisfy the 1kHz frequency requirement.

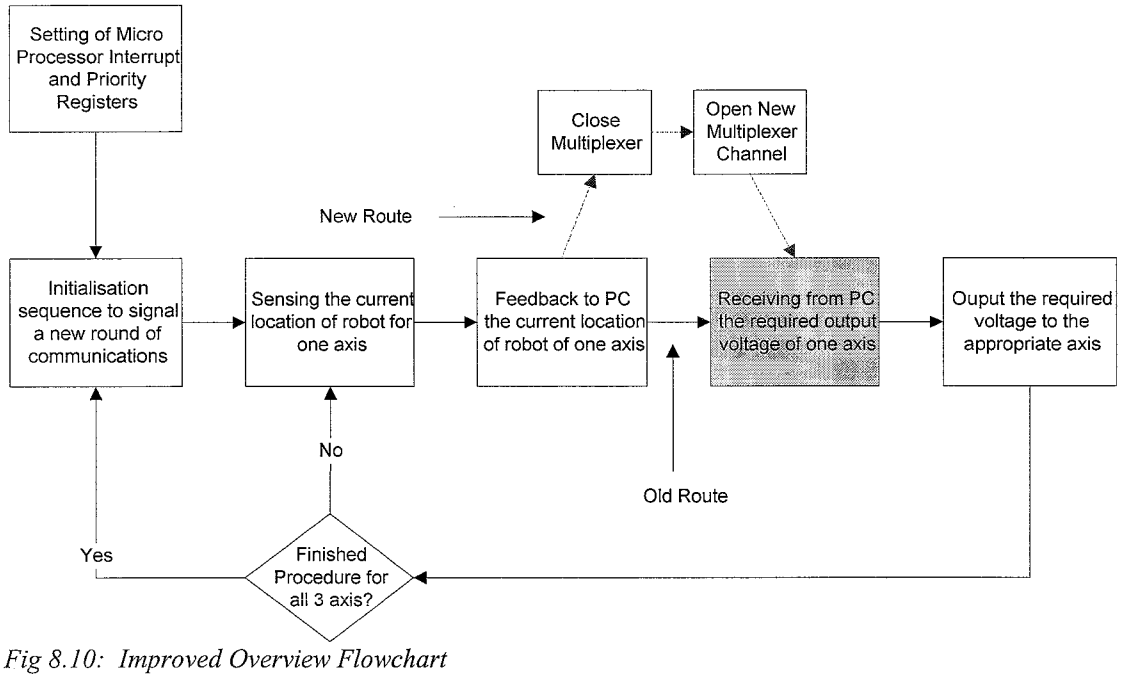


Fig 8.10: Improved Overview Flowchart

By processing the channels individually, we can keep the output voltage of the previous channel open while the controller is communicating with the PC. This means that while the controller and the PC are communicating, the capacitor is at the same time being charged up to its required voltage. Only when the communication is over, do we close the multiplexer and open another new channel. Thus, in this new method, the multiplexer is kept open almost all the time.

Chapter 9: Software For PC

9.1: Introduction

An interface is required to provide a means of interaction between the human user and the micro-controller. In this control application, the personal computer is the medium of communication. As such, a program has to be developed to provide a user-friendly graphical interface for the convenience of the user. As the Microsoft Windows Operating System is widely used, it has been decided that this program has to run in Windows. The program was written using Microsoft's Visual C++ Integrated Development Environment and is aptly named "The Industrial Robot Controller" (IRC), in view of the objectives for its development.

9.2: IRC Objectives

9.2.1 To provide an event-driven Windows graphical environment for user-controller interaction.

9.2.2 To implement a 2-way communication process between the personal computer and the micro-controller.

9.2.3 To implement feed-forward control equations for effective robot control based on parameters from user input.

9.2.4 To provide setpoint generation based on parameters from user input.

9.2.5 To allow visual inspection of the robot's ideal and actual trajectory characteristics through the means of graphs.

9.2.6 To provide a means of testing of certain aspects of the hardware, specifically the analogue-to-digital converter and the digital-to-analogue converter.

9.3: Description

The "Industrial Robot Controller" (IRC) communicates with the micro-controller by the parallel port of a personal computer. The personal computer is linked to the control-box via a lap-link cable. Input from the micro-controller is read by the status-port. For a parallel port, the status port only accepts inputs for the 5 most significant bits. Output to micro-controller is via the data-port. In this case, only the 5 least significant bits are used for transmission purposes.

9.3.1: Concept of Communication

It may be noted that the communication sequence consists of 4 main stages. They are the checking of the presence of the micro-controller, the receiving of the current location from the controller, the processes of the data via the control equations and finally the sending of the next predicted set point to the micro-controller.

Figure 9.1 shows the overall functional block diagram of the 2-way communication process between the personal computer and the micro-controller, as implemented by IRC.

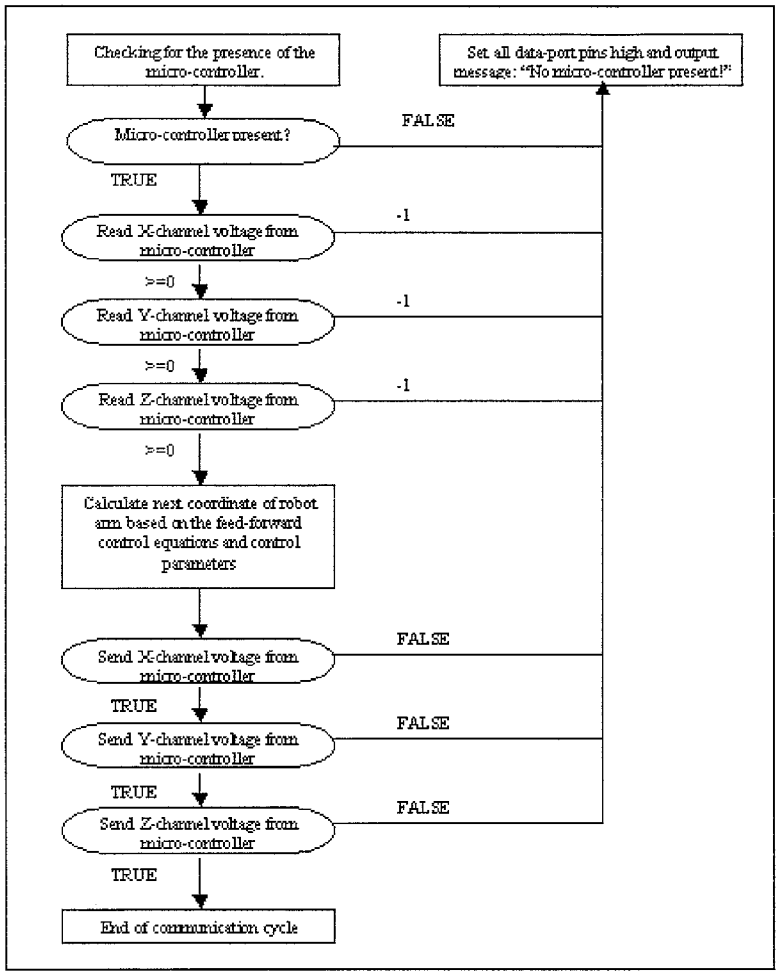


Figure 9.1: Overall Communication Sequence Between PC and Micro-Controller

Below, figure 9.2 describes the initialisation process of the communication sequence, which is the check for the presence of the micro-controller.

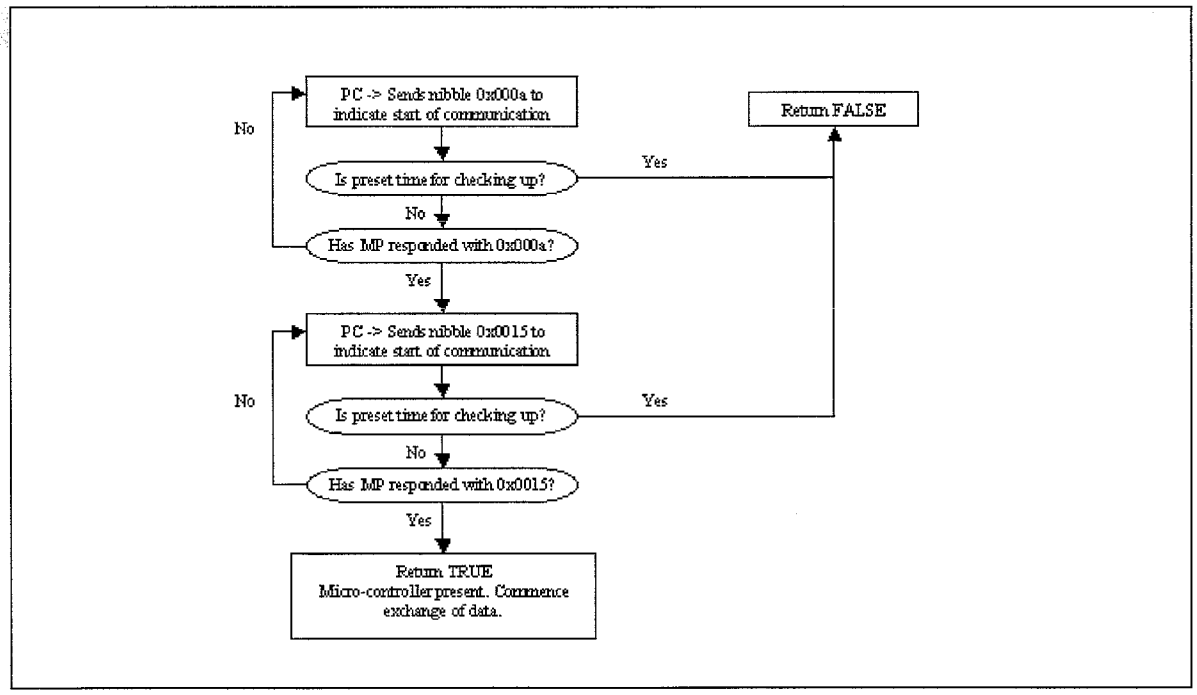


Figure 9.2: Initialisation Sequence

In the next page, figure 9.3 describes the reading of data from the micro-controller. Each of the below sequence reads in the position of the robot arm provided in terms of volts. Each sequence is executed 3 times, as there are 3 axes. For each sequence, a total of 20 bits is read in sequentially in packets of 5 bits. The first 3 packets describe the location of the robot arm. The most significant bit of each packet is toggled to facilitate the communication process. The fourth packet is just to set all the pins of the data-port high, and serves no purpose in this read sequence. It will be of useful purpose in the send sequence described in Figure 8.4.

The processing of the data is based on equations provided by the Dutch counterparts. Its implementation will be described in detail in Section 9.3.2.

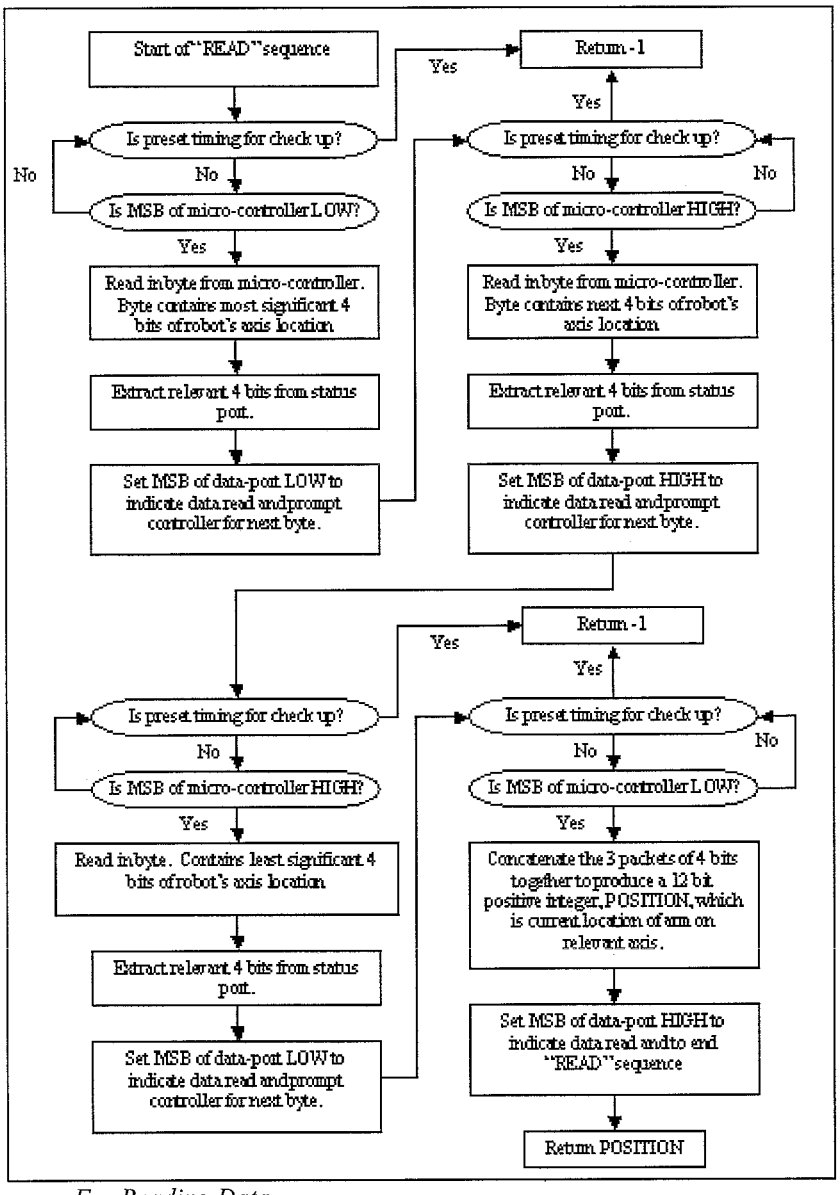


Fig. 9.3: Sequence For Reading Data

Figure 9.4 describes the sequence for sending the predicted location back to the micro-controller. Each of the below sequence sends the predicted position of the robot arm provided in terms of volts. Again, each sequence is executed 3 times, as there are 3 axes. For each sequence, a total of 20 bits is sent sequentially in packets of 5 bits.

The first 3 packets describe the location of the robot arm. The most significant bit of each packet is toggled to facilitate the communication process. The fourth packet is now used to control the control light of the control-box as well as, the magnet.

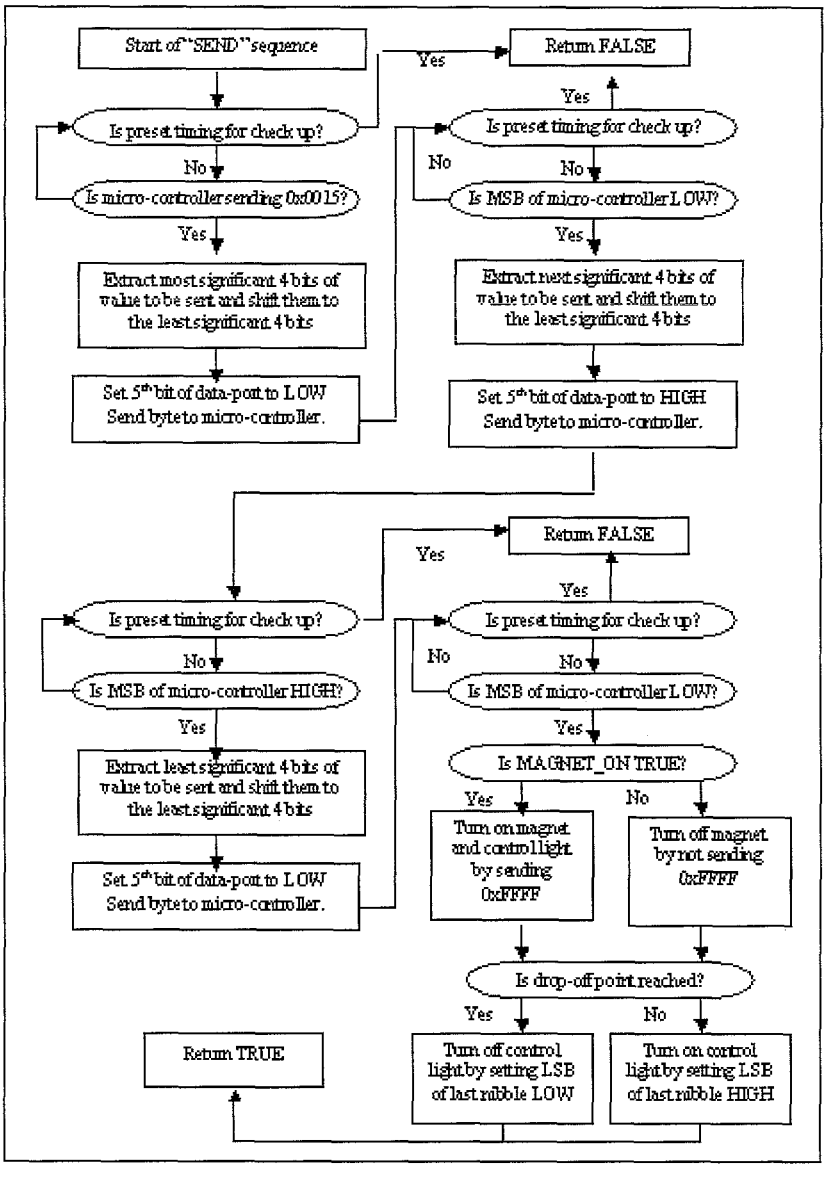


Fig. 9.4: Sequence For Sending Data

9.3.2: Implementation of PID Control

9.3.2a: Input of Control Parameters

Figure 9.5 shows the flow-chart for the input of the control parameters.

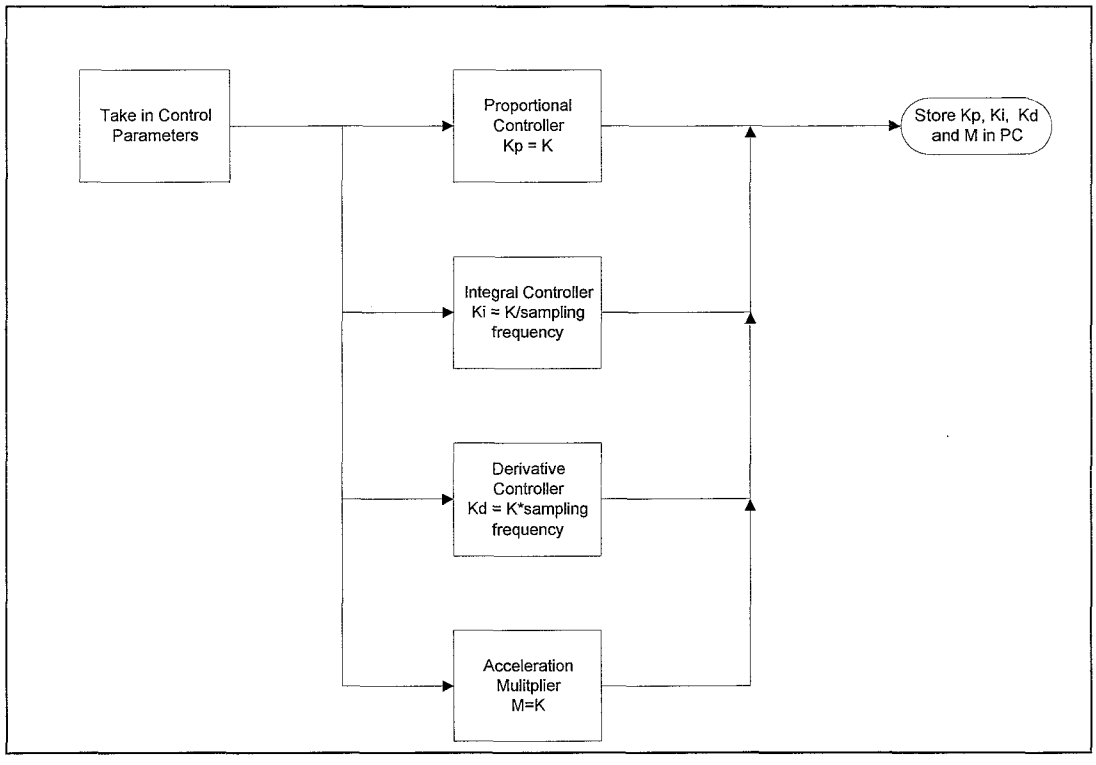


Fig. 9.5: Input of Control Parameters

Users are prompted to enter the control parameters via a customised dialog box. The proportional constant, Kp, and acceleration multiplier constant, M are stored as variables in the PC memory. For the integral constant, Ki, the constant entered is divided by the sampling frequency before storage. For the derivative constant, Kd, the constant entered is multiplied by sampling frequency before storage. This adjustment will simplify the control equations to K'_I and D'_d as seen below:

$$V = K_i \times I \times Period$$
$$= K_i \div SamplingFrequency \times I$$
$$= K_i^{'} \times I$$

$$V = K_d \times D \div Period$$
$$= K_d \times SamplingFrequency \times D$$
$$= K_d^{i} \times D$$

9.3.2b: Calculation Of Output Voltage

Figure 9.6 shows the flowchart for the calculation of output voltage to the micro-controller.

The calculated output voltage is a result of the sum of all four components: proportional controller (Kp), integral controller (Ki), differential controller (Kd) and the feed forward loop (M).

For the proportional controller, the error is directly multiplied by Kp to generate the output voltage.

For the integral controller, an integral variable, I, is updated by adding the current error to it. This integral variable, I, keeps track of the sum of all the errors. This integral variable will then be multiplied by the integral constant, Ki to generate the output voltage.

For the differential controller, the error is subtracted from the previous error. The result is then multiplied by Kd to generate the output voltage. It should be noted that the current error would now be updated to be a previous error for the next round of calculation.

Lastly, the feed forward loop is implemented by just multiplying the calculated acceleration required with the acceleration constant, M.

The result of all four components is then summed up to generate the required output voltage.

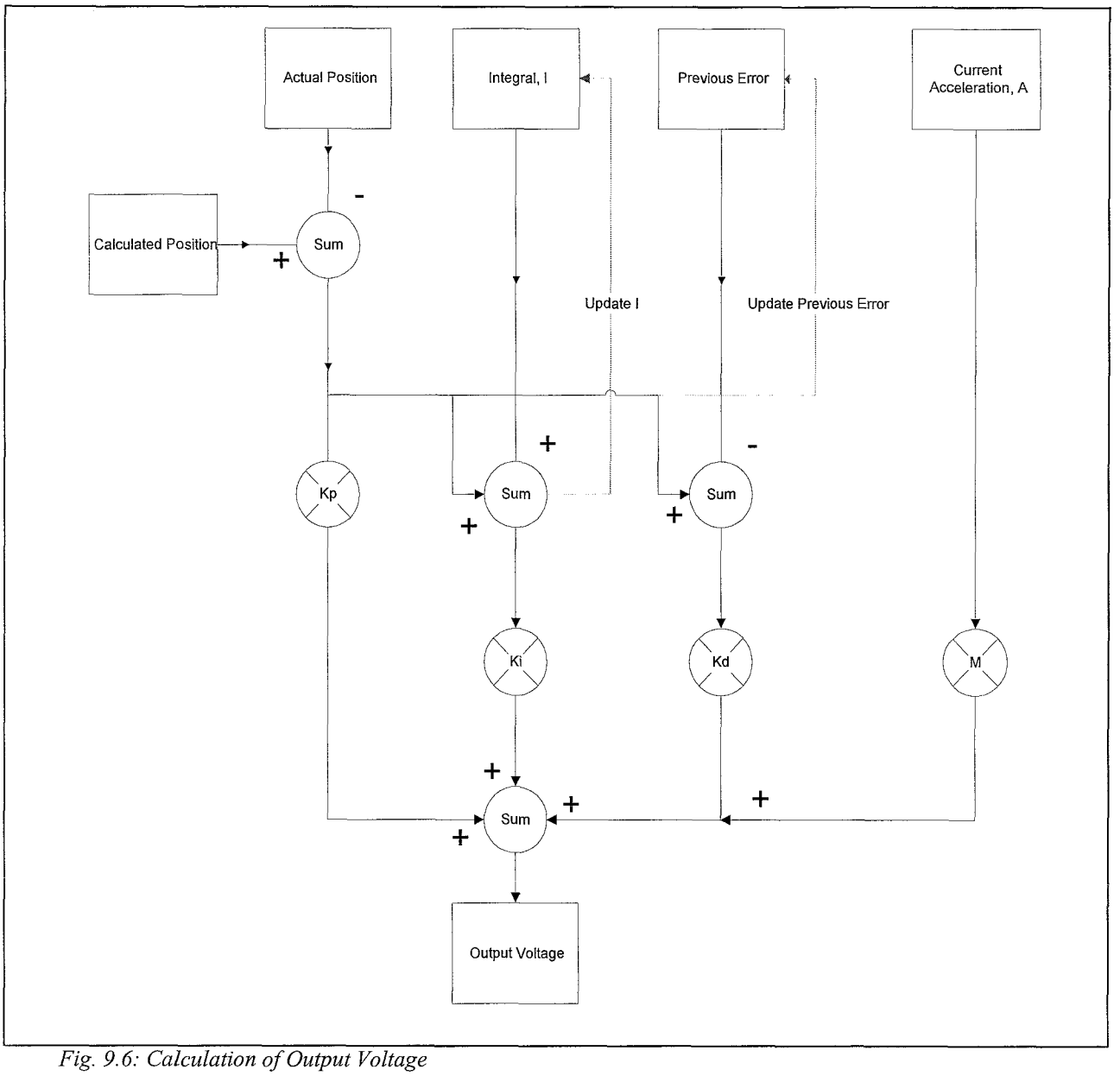


Fig. 9.6: Calculation of Output Voltage

9.3.3: Description of IRC Features

The "Industrial Robot Controller" is an event-driven object-oriented program written in Microsoft's Visual C++. It provides a Windows tabbed pane graphical

interface for the convenience of the user. It consists of 3 panels, namely the Control Box panel, the Graph Plots panel and the Options panel.

9.3.3a: The Control Box Panel

Below, Figure 9.7 shows a screen-shot taken of the Control Box panel.

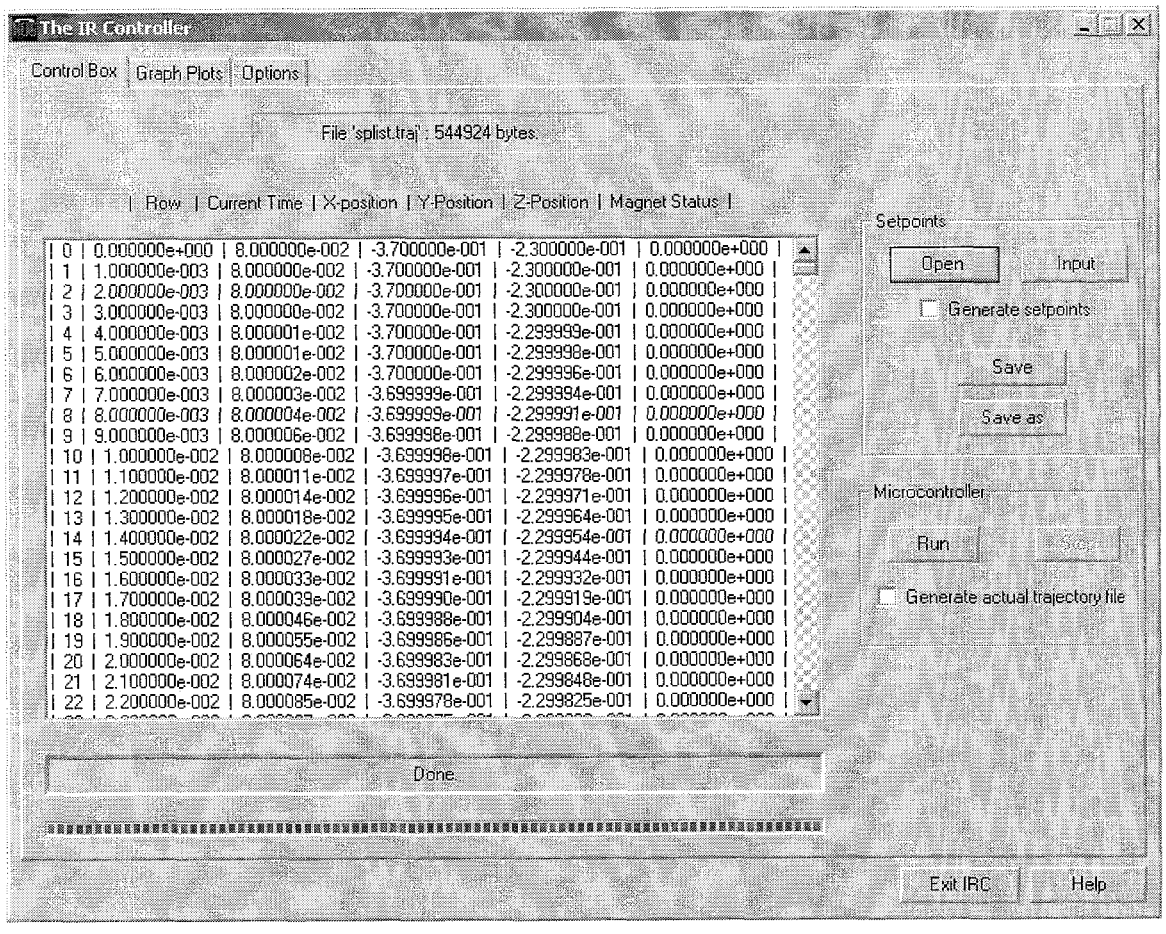


Fig. 9.7: The Control Box Panel

The Control Box panel is the main panel, which the user uses to control the robot. It provides an interface for the following functionalities.

Set-point Generation

IRC allows the user to generate set points based on the parameters provided by the user. The basic generation of the set points is based on a Mat-Lab algorithm developed by our Dutch counterparts, which we converted to C++ code.

IRC permits the user to generate set points whether he is connected to the micro-controller or not. When connected, IRC will detect the current location of the robot from the microprocessor. It will then prompt the user for the required parameters, specifically: load pick-up point, load drop-off point, maximum velocity, maximum acceleration, desired time and sampling frequency. Input is done via a pop-up dialog box shown in Figure 9.8.

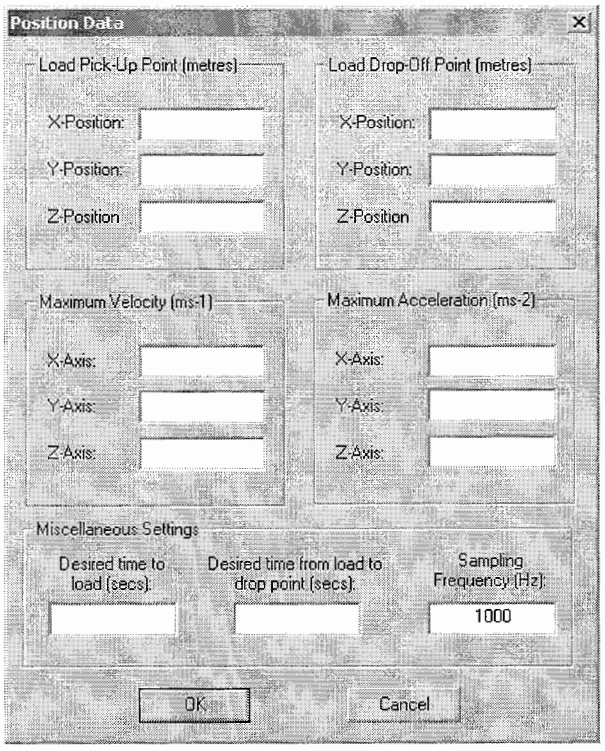


Fig. 9.8: Input Dialog

If the user is not linked to the microprocessor, a separate dialog will request the user to input a starting position for the robot. In this case, the "Generate Set Points" check box has to be enabled. Set point generation will then proceed based on the 3 points: start point, pick-up point and drop-off point. These will be displayed in the list box.

During set point generation, other trajectory characteristics like velocity, acceleration will be concurrently generated. The user may choose to examine these characteristics by simply double-clicking on the relevant row, which will bring up the data in a dialog box, as shown in Figure 9.9.

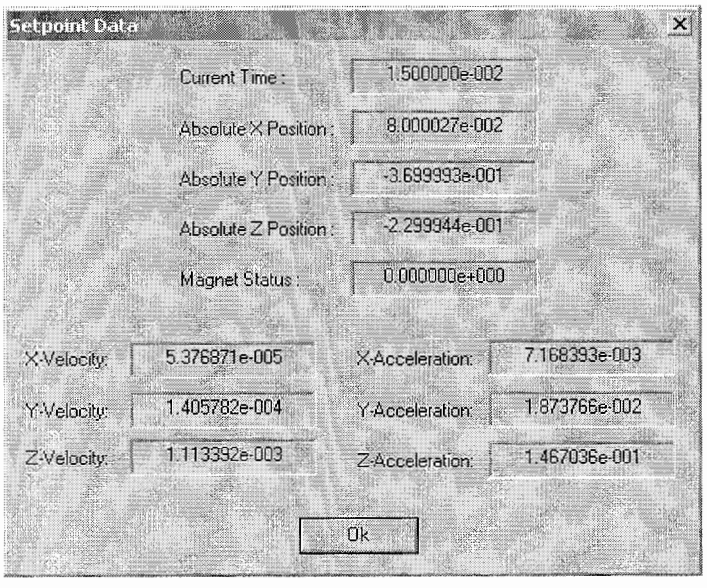


Fig. 9.9: Data Dialog

For all inputs, there is error checking to ensure that the parameters entered by the user is within the practical bounds of the robot, and that there are no complex roots arising from the algorithm provided. Should such errors occur, pop-up dialog boxes will warn the user.

Saving and Opening of Text Files

IRC allows the user to save all generated set points and actual trajectories in files for future use/reference. To save actual trajectories, the "Generate Actual Trajectory File" checkbox has to be enabled before saving.

For each trajectory saved, 2 files are generated. The set points are saved as *.traj files, while the other trajectory characteristics are saved as *.dat files. The pick-up point and drop-off point are marked in the set point file at the beginning of the relevant row, with a 'p' and a 'd' respectively.

The user is also able to open pre-generated trajectory files for usage. For this process to occur without errors, both the *.traj file and the *.dat file must be present.

Upon the clicking of the "Run" button to begin the robot motion, IRC will detect the current position of the robot and generate set points from this position leading up to the first starting position of the opened file. It will then concatenate the 2 sets of points together into one trajectory, and commence the motion.

Commencing and Halting Robot Motion

The clicking of the "Run" button will commence the motion of the robot.
It will start a separate periodic thread that will execute the communication sequence described in 9.3.1), in accordance to the sampling frequency entered.

A multimedia thread is used as it is documented as having the highest precision. Despite taking up much of the processor's computing resources, it will still allow other tasks to be executed, while the program is running. This allows for the termination of the communication sequence prematurely by clicking on the "Stop" button.

Help File

A help file in HTML format, describing the functionalities and usage of IRC may be accessed upon clicking the "Help" button. This button is visible in all panels and will open the help file in Microsoft's Internet Explorer.

9.3.3b: The Graph Plots Panel

Below, Figure 9.10 shows a screen-shot taken of the Graph Plots panel.

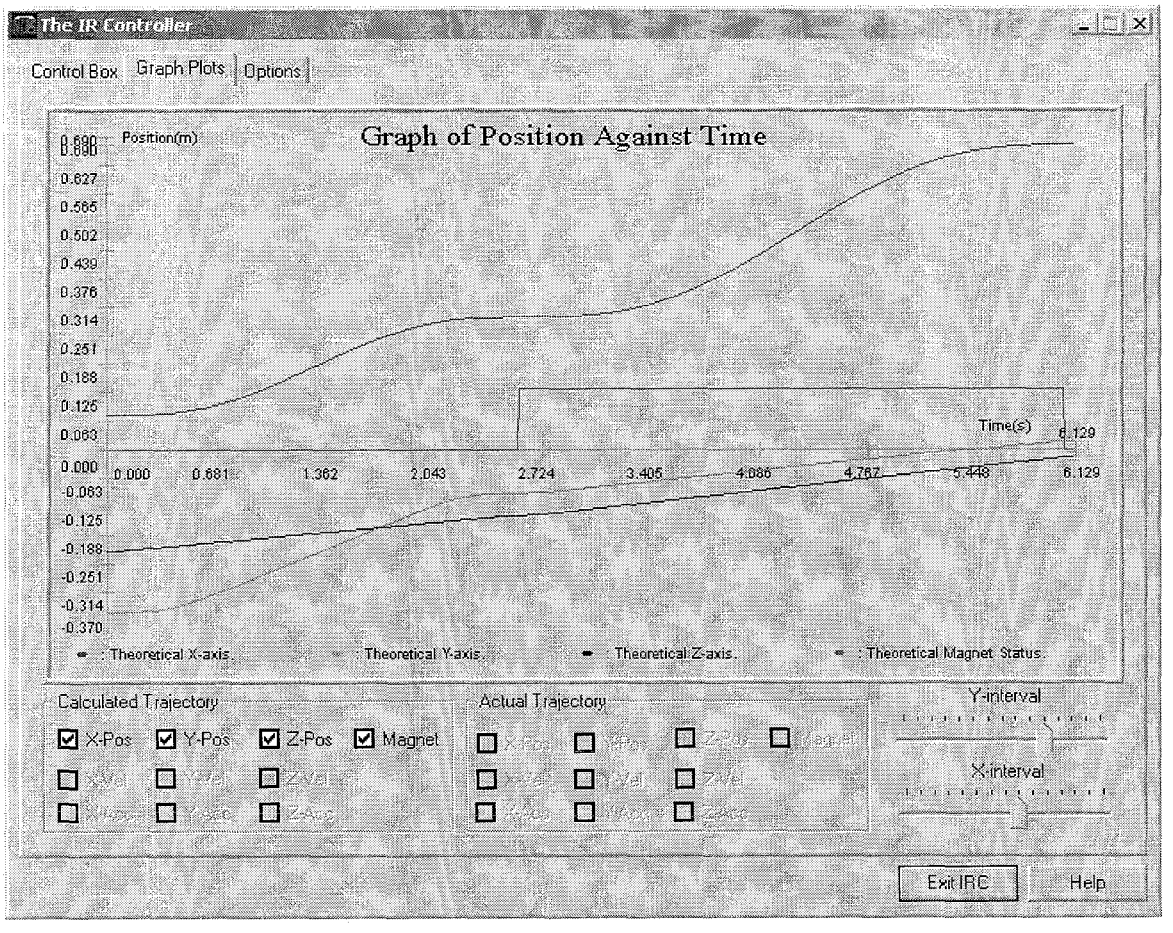


Fig. 9.10: The Graphs Plots Panel

The Graph Plots panel allows for visual inspection of both the generated and actual trajectory data in the form of graphs. The set points, velocities and accelerations of all 3 axes may be plotted out against time and compared with each other.

The actual instantaneous velocity graph for each axis is approximated from the actual positions, using the below equation:

$$V(t) = [P(t) - P(t-1)] / ts$$

Where V(t): velocity of a particular axis at time t
P(t): position at time t
ts : sampling period

The actual instantaneous acceleration graph for each axis is approximated from the actual velocities, using the below equation.

$$A(t) = [V(t) - V(t-1)] / ts$$

Where A(t): acceleration of a particular axis at time t
V(t): velocity at time t

ts : sampling period

The X and Y axis of the graph change dynamically according to the data set plotted. The intervals displayed on each axis may be adjusted using the slider bars provided.

9.3.3c: The Options Panel

Below, Figure 9.11 shows a screen-shot taken of the Options panel.

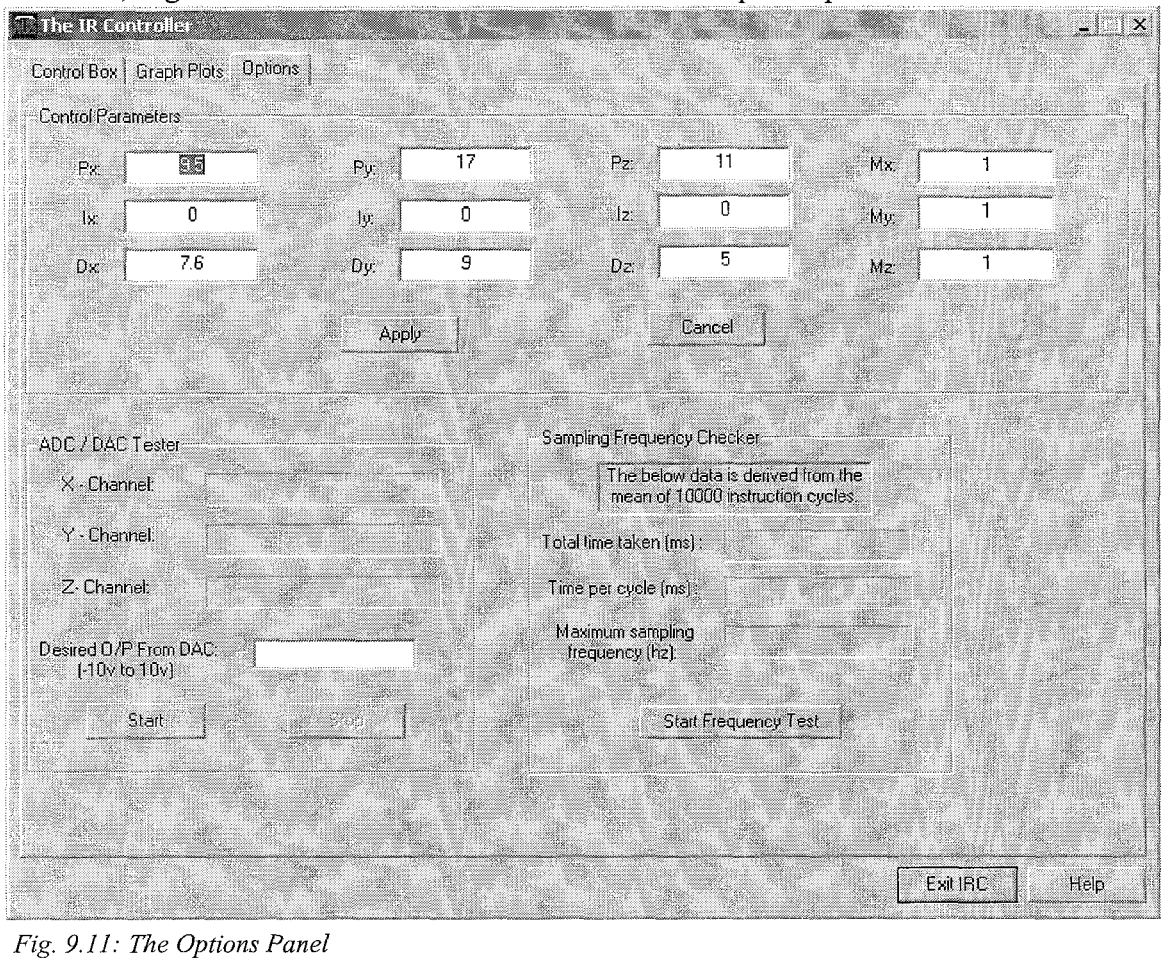


Fig. 9.11: The Options Panel

The Options Panel carries with it three tools.

Input of Control Parameters

The user is able to input the control parameters required for the feed-forward control equations. The 9 parameters make up that of a PID controller for each of the 3 axes.

The Options Panel is serialisable and all previous user inputs are retained.

ADC/DAC Tester

This is a simple tool to test certain portions of the hardware, namely the analogue-to-digital converter and the digital-to-analogue converter.

Upon the clicking of the "Start" button, IRC will constantly read in input from the micro-controller and display the voltage applied to the 3 input channels via the analogue-to-digital converter.

IRC will at the same time, output a voltage as specified by the user, to all three channels of the micro-controller. The micro-controller will in turn provide the necessary signals for the digital-to-analogue converter to output an analogue signal via the output channels of the control box.

This voltage may then be verified using a simple voltmeter.

Sampling Frequency Tester

This is a tool to test the personal computer's processing speed, and thus determine the maximum sampling frequency allowed, before overlapping occurs.

The function, which encapsulates the communication sequence described earlier, is executed 10000 times, and the mean execution time is displayed. As the sampling period cannot be smaller than the execution time of the function, the maximum sampling frequency is calculated and displayed.

9.4: Program Considerations

Throughout the development of IRC, several important issues were considered and will be highlighted.

9.4.1: Communication With Micro-Controller

There was a need to agree on a protocol by which the micro-controller communicated with the personal computer via the lap-link cable. Such a protocol had to be robust and efficient enough for reliable and quick communication between the 2 computers. The protocol agreed upon was described earlier in Section 9.3.1).

9.4.2: Sampling Frequency

It was decided that the timing should be initiated and kept by the personal computer. In a Windows environment, there are many background processes being carried out and conventional timers may not be accurate enough for the 1khz sampling frequency specified by the Dutch.

Finally, a multimedia timer was used. It is a very high priority timer that demands much system resources. Besides using a reliable timer, the code executed within the sampling period had to be as short as possible to prevent overlapping, which may cause instability in the program.

9.4.3: Compatibility with Windows NT and Windows 2000

IRC was developed and tested in a Windows 9.x environment. Later tests revealed that security features in WinNT and Win2000 did not allow the direct access to the printer port by IRC.

An IRC version able to be used under WinNT and Win2000 has been developed. For this version of IRC to run correctly, a driver has to be installed. This driver was

obtained from the Internet and provided C++ instructions for accessing the parallel port.

However, this version of IRC executes much slower than the original and with this version, a high sampling frequency is not possible. The maximum sampling frequency may be obtained by running the "Sampling Frequency Tester" in the Options Panel.

9.5: Testing Procedure And Observations

As the actual robot to be controlled is not available, there is no foolproof way to test the "Industrial Robot Controller" and the control box.

The communication between the control box and the personal computer was tested in this manner. Variable voltage supplies were connected to the input channels of the control-box to simulate the signals provided by the robot. The output voltages of the control box were monitored using voltmeters.

The entire sequence was then stepped through using the Integrated Development Environment's debugging tool. Changing the inputs and monitoring the outputs verified the control sequence and equations.

Upon verifications of the equations and sequences, the entire process was executed in real time. Again, manually applied voltages simulated the inputs from the robot, till the load pick-up point and load drop-off point was reached. The graphs were then verified for their correctness.

It was observed that for a sampling period of 1ms, the computer tended to slow down dramatically and even stall after a short interval. It may have been a result of a new process overlapping a previous instance, by commencing before the end of the previous one. This could be a result of other background processes, which may have slowed down the main process (initiated by the high priority thread), causing it to take more than 1ms to execute.

For a sampling period of 2ms, the program executed properly, with the expected results.

9.6: Conclusion

Using Microsoft Visual C++ for the first time to develop a complex Windows application, which interfaces with hardware, is a challenging but enriching task.

The "Industrial Robot Controller" was developed to provide a graphical user interface for the control of an industrial robot in Holland. It communicates with the control box effectively and implements the feed-forward control equations necessary to control the robot. It provides functionality for set point generation, visual inspection of trajectory data via graphs and hardware testing.

However, as it is running in a Windows environment, where there are numerous background thread processes executing, it was unable to attain the initially specified sampling frequency of 1khz. But it could execute properly at 500hz, which is sufficient for the objectives of the project, which is the control of a mechanical system.

Chapter 10: Achievements of the control box

10.1: Introduction

This chapter will describe the performance of the control box in reality. The security box will be mentioned. This is a dSpace system, which will prevent the robot from damaging itself.

10.2: The testing procedure

First of all, the AD and DA converters have to be checked. That is very easy accomplished by using the IRC program. A power supply is connected with the input of the control box. The program gives the voltage that is delivered by the power supply. If they match, the AD converter is working well. The program allows letting the control box deliver a specific voltage, this voltage can be checked with voltage meter. If they match, the DA converter is also working well.

The inputs of the control box work between 0 [V] and 10 [V]. So, the security box must also deliver that voltage range. That is not the case, so the Simulink model has to be changed. Every output of the Simulink model (x, y and z) has to be checked and changed. For example, the minimum x-axis position is around 0.1 [m], the security box delivers approximate 1.0 [V], instead of 0 [V]. The maximum x-axis position is around 0.65 [m], the security box delivers approximate 6.5 [V], instead of 10 [V].
To adapt the signal to the right one, 1 [V] has to be subtracted of the signal and than the signal must be multiplied by 1.84. Now 0.10 [m] equals 0 [V] and 0.65 [m] equals 10 [V].
For the y-axis and z-axis the same can be done. Only the security box gives the 10 [V] when the axis is at the minimum (-0.23 [m] for the z-axis) and the IRC program demands that is 0 [V]. So the values produced by the y-axis and z-axis is subtracted with 10 [V] and multiplied by –1.
The outputs of the control box (except for the magnet) have to be calculated by the inverse of the above mentioned calculations.

To check whether the controller of the control box works well, a test can be performed. The setpoint that is inserted in the security box will be the same as is inserted in the control box. Both boxes will control the robot. The outputs of the control box however, are not connected with the robot. So only the security box will control the robot for real. If the control signals match (within a reasonable margin) the control box and the Simulink model will be approved, otherwise something is wrong.

10.3: Reality

Attempts to make the above mentioned method work failed. So the controlbox is directly connected to the robot. After some alternations in the Simulink model and adding some tricks, the controlbox is able to control the robot.

To make the controlbox work better, an update of the IRC has been made. This new program will not check the initial position of the robot and thus will not create new setpoints when it is activated. Also the program will start the controlsignals, only after a specific signal is send to the three inputs.

The new program doesn't work very well, some unspecified errors occur. So the old IRC program is used instead.
After the robot has finished its initialisation procedure, the control signal will be set to 0.000 [V]. Then the 'run' button in the IRC program is pressed and the program begins initialising. When the IRC program gives 'connected to microcontroller' the user must press a button which will transmit the controlsignal of the controlbox to the robot.

Also the IRC program must not be quit, because the next time the user starts the program, it will conclude the robot is at a other position than the first time (maybe 0.01 [mm] off) and will produce new setpoints, which results in a program error.

Running the IRC program gives an error that the 'ircdata\parameters.dat' cannot be found. After experimenting the program wants that file in the same directory as the setpoint files. So now it is possible to use controlparameters defined by the user.

Finally, the controlparameters which are used in the IRC program differ very much from the ones used in the Simulink / dSpace controller. The output signal of the controlbox is also multiplied by 0.1 to make the robot move more relaxed.
Now the robot moves relatively smooth form one point to the next.

Chapter 11: Discussion

11.1: Communication

First of all it must be made clear that this project adds a whole new dimension to the study. It is, of course, obvious that communication is a major subject in this project. In order to prepare the students for their future career this communication part that couldn't be found in earlier projects is very educational. But on the other hand it was also the most difficult part of the project. Understanding each other turned out to be very difficult.

Also the fact that the two groups of students came from different faculties was a source for some minor miscommunication. The NUS-students were studying on the electrical engineering faculty, whereas the TUE-students follow their courses at the mechanical engineering faculty.

Later on during the project, it became clear that the video conferencing sessions were not the best source of communication. That role was taken in by letters via E-mail. On the other hand those video conferencing sessions were certainly *not* redundant.

The meetings via the Internet, using NetMeeting, however, *were* redundant. Due to low bandwidth on the Internet and the bad sound and video quality, those meetings didn't add anything constructive during this project.

11.2: Recommendations

The start of the project was the toughest. Both groups of students (NUS & TUE) wanted to work on it at once. But as the NUS-students were depending on the information given to them by the TUE-students, the last ones had to do double work.

First, they had to find out what the 'problem' was and how the system worked, and at the same time they had to provide the NUS-students with all the information they needed to start with their part of the project. In future similar projects, this can be avoided by letting the TUE side first find out how everything works. Then, when they have made a working controller the NUS-students can join. From that point on both parties can work. The TUE-side can improve their controller and also they can provide the NUS-side accurately and fast with the information on the system.

Another point which can use improve at TUE side, is the planning of the project. Now the three students worked most of the time together. Much more productive it will be, if the students would work parallel to each other. That is not possible all the time, because every one has to understand certain parts of the project. But the planning could have been better.

Even when the planning is better, the time for the project is too short. Previous years, teams of four persons worked on the project. Now, a three-person team works on the project and also has to maintain communication with the other team.

Due to the lack of familiarity with micro controllers, digital to analogue and analogue to digital converters, digital PID control and the Microsoft Visual C++ programming language a lot of time had to be spent in understanding, learning and experimenting before the controller and its software could be successfully implemented.

Conclusion

The joint project between the Mechanical Engineering department at the Technical University of Eindhoven and the Electrical Engineering department at the National University of Singapore is quite successful.

It is very informative and useful to work together with students from another country and with a different technical background. Good planning and communication are two very important skills needed to successfully complete this project. Those skills have certainly improved since the start of the project. Also writing a report in English is useful.

The project itself is more complicated and costs more time than was foreseen. After a couple of months of hard work, it's uplifting to conclude the control box does almost work the way it should. Some errors and minor bugs still exist, but the goal to make the robot move is achieved.

Literature list

[1] G.F. Franklin, J.D. Powell & A. Emani, Feedback control of dynamic systems. 3rd Edition, Eddison-Wesley, 1994

[2] M.P. Koster, W.T.C. van Luenen & T.J.A. de Vries, *Mechatronica*, 5th Edition, dictaat, Technische Universiteit Twente, 1998

[3] M. Steinbuch & M. Vervoordeldonk, *Control System Tuning*, Philips CFT/PRLE, 1996

[4] Dictation Constructieprincipes 1, *Bedoeld voor het nauwkeurig bewegen en positioneren*, Eindhoven University of Technology, Mechanical Engineering, Precision Engineering, march 1998

[5] H. Butler, Stage Control in PAS5500-based Step & Scan Systems, ASML Control Course TUE 1999, 21-10-1999

[6] R. v.d. Molengraft, Sheets belonging to the project *Control of a Industrial Robot*,

Symbolic List

Quantity	Symbol	Unit	Symbol
Time	t	Seconds	s
Position	x	Meters	m
Velocity	v	Meters per second	ms^{-1}
Acceleration	a	Meters per squared second	ms^{-2}
Length	l	Meters	m
Force	F	Newton	N
Current	I	Ampere	A
Voltage	V	Volts	V
Frequency	f	Hertz	Hz
Frequency	ω	Radians per second	$rads^{-1}$
Amplitude	H	Decibels	dB
Phase	φ	Degrees	^{o}
Resistance	R	Ohm	Ω
Power	P	Watt	W
Jerk	J	Meters per 3^{rd} power of second	ms^{-3}

Appendix 1: The calculation of constants

In this section the determination of constants used in chapter 3 is executed. The formulas (3.1)-(3.3) are repeated here:

$$x(t) = -a \cdot \sin(b \cdot t) + c \cdot t + d \qquad \text{(A1.1)}$$

$$v(t) = -(a \cdot b) \cdot \cos(b \cdot t) + c \qquad \text{(A1.2)}$$

$$a(t) = (a \cdot b^2) \cdot \sin(b \cdot t) \qquad \text{(A1.3)}$$

Using formula (A1.1)-(A1.3):

$0 \le b \cdot t \le 2\pi$ with: $0 \le t \le dt$,so:

$$b \cdot t \equiv 2\pi \ (t=dt) \Rightarrow b = \frac{2\pi}{dt}$$

$$a \cdot b = 0{,}5 \cdot v_{max} \Rightarrow a = \frac{0.5 \cdot v_{max}}{\frac{2\pi}{dt}} = \frac{v_{max} \cdot dt}{4\pi}$$

A constant term is added to formula (A1.2) to let the velocity walk from '0' to 'v_{max}' instead of '$-0{,}5 \cdot v_{max}$' to '$0{,}5 \cdot v_{max}$'. This term equals:

$$c = 0{,}5 \cdot v_{max}$$

The last term from equation A1.1 is founded by:

If: $t \equiv 0$, then $x(0) = x_0 \Rightarrow d = x_0$

Calculating the minimum time to accelerate and decelerate 'dt', equation (A1.2) and (A1.3) are used:

$$(a \cdot b) = 0{,}5 \cdot v_{max} \text{ and } a \cdot b^2 = a_{max}$$

Figure A1.1: Course of velocity

gives:

$$0{,}5 \cdot v_{max} \cdot b = a_{max}$$

Using the definition for 'b':

$$\frac{2\pi \cdot v_{max}}{dt} = a_{max} \Rightarrow dt = \frac{\pi \cdot v_{max}}{a_{max}}$$

Appendix 2: Equations new trajectory

Table A2.1: Definition of the different zones used in chapter 2, figure 2.2.

Zone	time
A	$t_0 < t < t_1$
B	$t_1 < t < t_2$
C	$t_2 < t < t_3$
D	$t_3 < t < t_4$

Table A2.2: The equations belonging to the zones used in chapter 2, figure 2.2

zone	equation
A	$a = Jt$ $v = \frac{1}{2}Jt^2$ $x = \frac{1}{6}Jt^3$
B	$a = JT_j = a_{\max}$ $v = JT_j t - \frac{1}{2}JT_j^2$ $x = \frac{1}{2}JT_j t^2 - \frac{1}{2}JT_j^2 t + \frac{1}{6}JT_j^3$
C	$a = -Jt + JT_a$ $v = -\frac{1}{2}Jt^2 + JT_a t + \left(JT_a T_j - JT_j^2 - \frac{1}{2}JT_a^2\right)$ $x = -\frac{1}{6}Jt^3 + \frac{1}{2}JT_a t^2 + J\left(T_a T_j - T_j^2 - T_a^2\right)t + J\left(-\frac{1}{2}T_a^2 T_j + \frac{1}{2}T_j^2 T_a + \frac{1}{6}T_a^3\right)$
D	$a = 0$ $v = JT_a T_j - JT_j^2 = v_{\max}$ $x = J\left(T_a T_j - T_j^2\right)t + J\left(\frac{1}{2}T_j^2 T_a - \frac{1}{2}T_a^2 T_j\right)$

Appendix 3: The Trajectory M-files

Name: Setpoint.m

```
% Clearing all figures and variables
clear all;
close all;

% Defining some global variables
global data;                % Used for saving path-
information
global fs;                  % The sample frequency
global t0;                  % The initial starting time
global total_data;    % Total data matrix consist of all
data marixes

global newdatax;            % Used for TUE setpoint
generation
global newdatay;            % Used for TUE setpoint
generation
global newdataz;            % Used for TUE setpoint
generation
global newdatamag;% Used for TUE setpoint generation


fs = 1000;
t0 = 0;
total_data = [];

vmax = [ 1 1 1 ];
amax = [ 4 6 6 ];

% Matrixes for TUE setpoint generation
[x0 x1 vmax amax jerk t0]  en  [on/off t0]
newdatax = [];
newdataxy = [];
newdataxz = [];
newdatamag = [];

% Factor to extend time with
a = 1;

% Square in xy-plane with z-movement
p = [0.130 0.010 -.030;
    0.640 0.010 -.220;
    0.640 -.360 -.030;
    0.130 -.360 -.220;
    0.130 0.010 -.030];

% Diamand in xy-plane
p = [0.1000 -.1700 -.1000;
    0.3850 0.0000 -.1000;
    0.6700 -.1700 -.1000;
    0.3850 -.3400 -.1000;
    0.1000 -.1700 -.1000];


% Diagonaal van min naar max
p = [0.130 0.010 -.030;
    0.640 -.360 -.220;
    0.130 0.010 -.030];


 % Haasje-over
%  ?0 is the natural start position
z0 = -.050; z1 = -.100; z2 = -.170; z3 = -.220;
x0 = 0.400; x1 = 0.640; x2 = 0.490; x3 = 0.340; x4 =
0.190;
y0 = -.200; y1 = -.190; y2 = -.180; y3 = -.190; y4 = -.180;
w = 1;
% Beweeg van 'row i' naar 'row i+1' met de parameters
van 'row i'
p = [x0 y0 z0 0 0;
    x1 y1 z2 0 0;
    x2 y2 z1 1 0;
    x4 y4 z1 1 0;
    x4 y4 z3 0 w;
    x4 y4 z3 0 0;
    x0 y0 z0 0 w;
    x0 y0 z0 0 0;
    x4 y4 z2 0 0;
    x4 y4 z3 1 w;
    x4 y4 z3 1 0;
    x4 y4 z1 1 0;
    x2 y2 z1 1 0;
    x2 y2 z3 0 w;
    x2 y2 z3 0 0;
    x2 y2 z2 0 0;
    x3 y3 z2 0 0;
    x3 y3 z3 1 w;
    x3 y3 z3 1 0;
    x3 y3 z1 1 0;
    x1 y1 z1 1 0;
    x1 y1 z3 0 w;
    x1 y1 z3 0 0;
    x0 y0 z0 0 w;
    x0 y0 z0 0 0];

% Computing the paths -> total_data
for( i=1:1:size(p,1)-1 ),
  axcomp( p(i,1:3), p(i+1,1:3), vmax, amax,p(i,4),p(i,5));
end

% Plotting the data
plot(total_data(:,1),
total_data(:,2),'r',total_data(:,1),total_data(:,3),'g',total_da
ta(:,1),total_data(:,4),'b',total_data(:,1),total_data(:,5)/10,'
k:');
axis([min(total_data(:,1)) max(total_data(:,1))
min(min(total_data(:,2:5))-0.10)
max(max(total_data(:,2:5))+0.10)]);
title('x:red  y:green  z:blue  magnetstatus:black');
xlabel('Time [s]');
ylabel('Position [m]');
```

Name: Axcomp.m

```
function axcomp(x0,x1,vmax,amax,magnet,gewlengte)


global data;
global fs;
global t0;
global total_data;
global newdatax;
global newdatay;
global newdataz;
global newdatamag;

magnet = magnet;

for(i=1:3),
  % The delta-path
  dx(i) = x1(i) - x0(i);
  % Total minium time to startup and slow down
  dtx(i) = vmax(i)*pi/amax(i);
  % The path which has been covered by startingup and
slowing down
  x_in_dtx(i) = 0.5*abs(dtx(i))*vmax(i);
  % If that path is smaller than the required path, a
constant velocity part is required
  if( x_in_dtx(i) <= abs(dx(i)) ),
     t_total(i) = (0.5*vmax(i)*pi)/amax(i) +
abs(dx(i))/vmax(i);
  else,
    % No constant-velocity-part. This means de vmax
has to be changed:
    % dx = 0.5*dt*vmax = (0.5*pi*vmax^2)/amax
    vmax(i) = sqrt( 2*abs(dx(i))*amax(i) / pi );
    % dt also changes because the total path must be
the same
    dtx(i) = vmax(i)*pi/amax(i);
     t_total(i) = dtx(i);
  end
end

% The maximum minimum-time of the three paths
t_totalmax = max( t_total );


if( t_totalmax < gewlengte ),
  t_totalmax = gewlengte;
end

% If all axes are zero, the length doesn't change
lengte = 0;

for(i=1:3),
  % Calculating new velocities
  if( dx(i) == 0 ),
    vmax(i) = 0;
  else,
    % In case with a constant velocity part:
          % vmax^2*pi/2/amax - t_totalmax*vmax + dx =
0
     a = pi/2/amax(i); b = -t_totalmax; c = abs(dx(i));
        % Opl1 seems to deliver the good solution
opl1 = (-b-sqrt(b^2-4*a*c))/(2*a);
     opl2 = (-b+sqrt(b^2-4*a*c))/(2*a);
        vmaxtemp(i,1) = opl1;
     % No constant velocity part
     % dx = 0.5*dt*vmax
     vmaxtemp(i,2) = abs(dx(i)) / (0.5*t_totalmax);

     % total time in case of cons.vel.part
     tmaxtemp(i,1) = (0.5*vmaxtemp(i,1)*pi)/amax(i) +
abs(dx(i))/vmaxtemp(i,1);
     % total time in case with no cons.vel.part
     tmaxtemp(i,2) = (0.5*vmaxtemp(i,2)^2*pi)/amax(i);
     % trying to find the good solution
     if( tmaxtemp(i,1) - t_totalmax < 0.01 ),
           vmax(i) = vmaxtemp(i,1);
     elseif( tmaxtemp(i,2) - t_totalmax < 0.01 ),
     vmax(i) = vmaxtemp(i,2);
     else,
     fprintf('Failure, this program sucks!!!!\n');
    end
     % Starting a new matrix for the path-data
        lengte = ceil(t_totalmax*fs);
  end
end

if( gewlengte*fs > lengte ),
  lengte = gewlengte*fs;
end

% Computing the paths of the axis
for(i=1:3),
  setgen( x0(i), x1(i), amax(i), vmax(i),i,lengte,magnet);

  % This part is included to compute the jerk, needed for
TUE setpoint generation
  dt2(i) = vmax(i)*pi/amax(i);
  delta(i) = -(vmax(i)/amax(i)) + 0.5*dt2(i);
  if( delta(i) == 0 ),
    jerk(i) = 0;
  else,
    jerk(i) = amax(i) / delta(i);
  end

end

% Updating total_data matrix by placing newest
movement behind it
total_data = [total_data;  data];
% Updating the newest starting time
t0 = total_data( size(total_data,1), 1) + 1/fs;
%fprintf('DONE\n');


% Needed for TUE setpoint generation
newdatax = [newdatax;  x0(1) x1(1) abs(vmax(1))
abs(amax(1)) abs(jerk(1)) t0];
newdatay = [newdatay;  x0(2) x1(2) abs(vmax(2))
abs(amax(2)) abs(jerk(2)) t0];
newdataz = [newdataz;  x0(3) x1(3) abs(vmax(3))
abs(amax(3)) abs(jerk(3)) t0];
newdatamag = [newdatamag; magnet t0];
```

Name: setgen.m

```
function
[data]=setgen(p0,p1,MaxA,MaxV,as,lengte,magnet)

global data;
global fs;
global t0;
global total_data;
global newdatax;
global newdatay;
global newdataz;
global newdatamag;

% Max. velocity and max. accelaration
vxmax = MaxV;
axmax = MaxA;
%as = as;
%lengte = lengte;
% Start and end positions
x0 = p0;x1 = p1;dx = x1 - x0;

if( dx == 0 ),
  % The position does not change
  t = [t0:1/fs:t0+(lengte-1)/fs]';
  x = x1*ones( size(t,1), 1);
  v = 0*ones( size(t,1), 1);
  a = 0*ones( size(t,1), 1);
      % computing the size
      size_t = size(t,1);
      size_x = size(x,1);
else,
      if( dx < 0 ),
        vxmax = -vxmax;
      axmax = -axmax;
      end
      % Computing the minimum time to reach max.
speed with max. acc.
      % NOTE: dt = 2 * minimum time
      dt    = (vxmax*pi)/axmax;
      % End time when maximum 'v' has been reached
      t1 = t0 + 0.5*dt;
      %Computing the moved lenght in t:[t0,t1] and
t:[t2,t3]
      x01 = 0.5*vxmax*(t1-t0);
      x23 = x01;
      % So the lenght to go 'x12' is the total lenght minus
x01 and x23
      x12 = dx - x01 - x23;
      if( (dx > 0 & x12 > 0) | (dx < 0 & x12 < 0)),
            % The time that takes on top speed:
        t12 = x12 / vxmax;
      else,
      vxmax = dx/(t1-t0);
      t12 = 0;
      end
      % So:
      t0 = real( ceil(t0*fs)/fs );
      t1 = real( ceil((t0 + 0.5*dt)*fs)/fs );
      t2 = real( ceil((t1 + t12)*fs)/fs );
      t3 = real( t0+lengte/fs );
      t_constant = t12;
      t_total = t3 - t0;
      % Acceleration phase
      t01 = [t0:1/fs:t1]';
  v01 = - 0.5 * vxmax * cos( (2*pi)/dt * (t01-t0)) +
0.5*vxmax;
      a01 = 0.5 * vxmax * (2*pi)/dt * sin( (2*pi)/dt * (t01-
t0));
      x01 = - (0.5*vxmax*dt)/(2*pi) * sin( (2*pi)/dt * (t01-
t0)) + 0.5*vxmax*(t01-t0) + x0;
if( t12 ~= 0 ),
            % Constant velocity phase
            t12 = [t1:1/fs:t2]';
            v12 = vxmax.*t12./t12;
            a12 = 0.*t12./t12;
    x12 = x01(size(x01,1)) + vxmax.*(t12-(t1+1/fs));
      else,
      v12 = [vxmax];
        a12 = [0];
      x12 = [x01(size(x01,1))];
      t12 = [t1];
      end
  % Deceleration phase
  t23 = [t2:1/fs:t3]';
  v23 = - 0.5 * vxmax * cos( (2*pi)/dt * (t23-t2+0.5*dt)) +
0.5*vxmax;
      a23 = 0.5 * vxmax * (2*pi)/dt * sin( (2*pi)/dt * (t23-
t2+0.5*dt));
  x23 = ( x12(size(x12,1)) - x01(size(x01,1)) + x0 ) -
(0.5*vxmax*dt)/(2*pi) * sin( (2*pi)/dt * (t23-t2+0.5*dt)) +
0.5*vxmax*(t23-t2+0.5*dt);
      % Combining the matrixes
      x = [x01; x12; x23];
      v = [v01; v12; v23];
      a = [a01; a12; a23];
      t = [t01; t12; t23];
      % computing the size
      size_t = size(t,1);
      size_x = size(x,1);
      % For some weird reason the matrix is sometimes
      % smaller/larger than 'lengte'. So this alters
      % the matrix by cutting of some info or adding
      % some info
      t = [t0:1/fs:t0+(lengte-1)/fs]';
      size_t = size(t,1);
      teller =  0;
      while(size_x ~= lengte),
      if( size_x > lengte ),
      %fprintf('Axis %d too big. Correcting...\n',as);
      x( size_x ) = [];
      size_x = size_x - 1;
      end
            if( size_x < lengte ),
        %fprintf('Axis %d too small. Correcting...\n',as);
      for( i=size_x+1:1:lengte),
            x(i) = x(size_x);
          end
        size_x = size(x,1);
        end
      end
end

% Filling of the matrix
if( as == 1),
  data = [];
  % Only the first time the time-table is required and the
magnet
  data(1:size_t,1) = t(1:size_t);
      data(1:size_t,5) = magnet*ones(size_t,1);
end
%The path data
data(1:size_t,as+1) = real( x(1:size_t) );
```

Appendix 4: The M-file to fit a transferfunction

Name: Fitten.m

```
clear all
close all

bepalen = 0;
weergeven = 1;

fb = 10;          % Beginfrequentie
fe = 100;  % Eindfrequentie

wb = 10;          % Begin hoekfrequentie van de plot
[rad/s]
we = 500;         % Eind hoekfrequentie van de plot [rad/s]


if( bepalen == 1 ),

      for( i = 1:3 ),

        if( i == 1 ),
        load x-as1
      load Hpx-asmid
        end
        if( i == 2 ),
        load y-as1
      load Hpy-asmid
        end
        if( i == 3 ),
        load z-as1
      load Hpz-asmid
      end

      % Overdrachtsfunctie bepalen
            H = mean(Hp')';             % Gemiddelde van
alle responsies
            df = hz(2)-hz(1);           % Stapje in de
frequentie

            nb = round(fb/df);    % Begin increment
      ne = round(fe/df);    % Eind increment

            hz = hz(:,[nb:ne])';  % Matrix aanpassen
            H = H([nb:ne],:);           % Matrix aanpassen

            [num,den] = frfit(H, hz, [14,14,2]);          %
Overdracht fitten

            sys = tf(num,den);
      % Systeem opstellen

     if( i == 1 ),
       save overx2.mat den num sys
     end
     if( i == 2 ),
       save overy2.mat den num sys
     end
     if( i == 3 ),
       save overz2.mat den num sys
     end

   end
end

if( weergeven == 1 ),
  for( i = 1:3 ),
        if( i == 1 ),
              load x-as1
       load Hpx-asmid
        load overx2
     end
        if( i == 2 ),
        load y-as1
load Hpy-asmid
      load overy2
     end
        if( i == 3 ),
        load z-as1
      load Hpz-asmid
      load overz2
     end

     H = mean(Hp')';

     w = hz'*2*pi;

     sys2 = frd(H,w);

     figure
     %sys2 = tf([10 650 7676],[1 0 ]);


     % Het uitrekenen van de waarden van de matrixen
     [mag1, phase1, w1] = bode(sys,{1,3142});
     [mag2, phase2, w2] = bode(sys2,{1,3142});
     %[mag2, phase2, w2] = bode( sys*tf([1.6 16.5
30.9],[1 0]),{1,3142});
            %[mag2, phase2, w2] = bode( sys*tf([0.25
0.42 0.12],[1 0]),{1,3142});
            %[mag2, phase2, w2] = bode( sys*tf([10 650
7676],[1 0]),{1,3142});

     % Aanpassen van de matrices, omdat Matlab ze
hierboven
     % ...in een heel raar formaat wegschrijft
     mag1 = reshape(mag1,size(mag1,3),1);
            phase1 = reshape(phase1,size(phase1,3),1);
     mag2 = reshape(mag2,size(mag2,3),1);
            phase2 = reshape(phase2,size(phase2,3),1);

     phase1 = wrap(phase1);
     phase2 = wrap(phase2);

     % Weergeven van de waarden
     %subplot(2,1,1);
     semilogx( w1, 20*log10(mag1),'k',w2,
20*log10(mag2),'k:' );
     ylabel('Magnitude [dB]'); xlabel('Frequency [rad/s]');
title('Magnitude plot of x-axis');
     a = axis; axis([wb we -100 0]);
     legend('Fit','System');
     %grid on

     %subplot(2,1,2);
     %semilogx( w1, phase1, w2, phase2 );
     %ylabel('Fase [graden]'); xlabel('Omega [rad/s]');
title('Fase Diagram');
     %a = axis; axis([wb we a(3) a(4)]);
     %grid on

   end
end
```

Appendix 5: AT89C52 Code

```
;*********************************************************************
;               Initialisation of Constants and Data Storage(RAM)
;---------------------------------------------------------------------


PIN         equ         P0          ;input port from the PC
PINS        equ         P0.3        ;input status pin
POUT        equ         P2          ;output port to PC
POUTS       equ         P2.3        ;output status pin

DATABUS     equ         P3      ;dataport to communicate with DAC and ADC
M0          equ         P2.1    ;multiplexer channel selection pin 1
M1          equ         P2.0    ;multiplexer channel selection pin 2
A0          equ         P1.0    ; ADC and DAC A0
A1          equ         P1.1        ; DAC A1
DAC_CS      equ         P1.2        ; chip select of DAC
ADC_RC      equ         P1.3        ; read/convert of ADC
ADC_CE      equ         P1.4        ; chip enable of ADC
DAC_EN      equ         P1.5        ; DAC multiplexer enable
ADC_EN      equ         P1.6        ; ADC multiplexer enable
LIGHT       equ         P1.7        ; Control box light control pin
TIMER_B     equ         2fh.0       ;Timing bit for multiplexer

X_OUT_1     data        30h         ;stor for output voltage for x axis
X_OUT_2     data        31h
Y_OUT_1     data        32h         ;stor for output voltage for y axis
Y_OUT_2     data        33h
Z_OUT_1     data        34h         ;stor for output voltage for z axis
Z_OUT_2     data        35h

X_POS_1     data        36h         ;storage for location for x axis
X_POS_2     data        37h
Y_POS_1     data        38h         ;storage for location for y axis
Y_POS_2     data        39h
Z_POS_1     data        3ah         ;storage for location for z axis
Z_POS_2     data        3bh

;---------------------------------------------------------------------
;               a. Interrupt Vector Table
;---------------------------------------------------------------------
            org         0000h               ;System reset - jumps to start
            jmp         start
            org         000bh               ;Timer 0 interrupt.  Intialising of
            setb        TIMER_B
            reti


;---------------------------------------------------------------------
;               i. Initialisation of Interrupt Enable Register
;---------------------------------------------------------------------

start:
            mov         IE, #10000010b              ;enable Timer 0
            mov         TMOD, #00100010b    ;setting appropriate modes for Timer 0

            mov         TH0, #0d2h          ;Set timer interrupt to 0.2 ms
                                            ;Configured for a 11.059Mhz crystal
            clr         LIGHT


initialisation:                             ;initialisation loop to signify new sampling phase

            mov         C, PINS
            jc          initialisation
            mov         A, PIN
            anl         A, #0f0h
            cjne        A, #10100000b, initialisation
            mov         POUT, #10100000b

initialisation_2:

            mov         C, PINS
            jnc         initialisation_2
            mov         A, PIN
            setb        POUTS
            anl         A, #0f0h
            cjne        A, #01010000b, initialisation
            mov         POUT, #01011000b

ini_wait:

            mov         A, PIN
            anl         A, #0f8h
            cjne        A, #11111000b, ini_wait


;---------------------------------------------------------------------
;               c. Main Programme Loop
;---------------------------------------------------------------------


main:       call        sensor                  ;to sense the location of robot
            call        output_pc       ;to output to pc the location of robot
            call        input_pc                ;to input from pc the required output voltage
            call        motor                   ;to output the voltage to the appropriate channels
```

```
                jmp         initialisation;

;-------------------------------------------------------------------------------
;input_pc : input output voltage from PC
;-------------------------------------------------------------------------------

input_pc:
                call        parin                       ;input x output voltage
                mov         X_OUT_1, R0
                call        parin
                mov         X_OUT_2, R0
                call        parin
                mov         Y_OUT_1, R0                 ;input y output voltage
                call        parin
                mov         Y_OUT_2, R0
                call        parin
                mov         Z_OUT_1, R0                 ;input z output voltage
                call        parin
                mov         Z_OUT_2, R0

                ret

;-------------------------------------------------------------------------------
;output_pc : output position to PC
;-------------------------------------------------------------------------------

output_pc:

                mov         POUT, #00001000b
                mov         R0, X_POS_1                 ;output x position
                call        parout
                mov         R0, X_POS_2
                call        parout
                mov         R0, Y_POS_1                 ;output y position
                call        parout
                mov         R0, Y_POS_2
                call        parout
                mov         R0, Z_POS_1                 ;output z position
                call        parout
                mov         R0, Z_POS_2
                call        parout

out_wait:
                mov         C, PINS
                jnc         out_wait
                mov         POUT, #01011000b;

                ret

;-------------------------------------------------------------------------------
;parout : byte output from PC from R0
;-------------------------------------------------------------------------------

parout:         mov         C, PINS
                jnc         parout
                mov         A, R0
                anl         A, #0f0h
                mov         POUT, A
                clr         POUTS
powait:         mov         C, PINS
                jc          powait
                mov         A, R0
                anl         A, #00fh
                rl          A
                rl          A
                rl          A
                rl          A
                mov         POUT, A
                setb        POUTS

                ret

;-------------------------------------------------------------------------------
;parin : byte input from PC into R0
;-------------------------------------------------------------------------------

parin:          mov         C, PINS
                jc          parin
                mov         A, PIN
                clr         POUTS
                anl         A, #0f0h
                mov         R0, A

pwait:          mov         C, PINS
                jnc         pwait
                mov         A, PIN
                setb        POUTS
                anl         A, #0f0h
                rr          A
                rr          A
                rr          A
                rr          A
                orl         A, R0
                mov         R0, A

                ret

;-------------------------------------------------------------------------------
;motor : Outputing Analogue voltages,
;-------------------------------------------------------------------------------

motor:
```

```
            mov     R0, X_OUT_1             ;output X voltage to channel 1
            mov     R1, X_OUT_2
            clr     M0
            clr     M1
            call    dtoa;

            mov     R0, Y_OUT_1             ;output Y voltage to channel 2
            mov     R1, Y_OUT_2
            setb    M0
            clr     M1
            call    dtoa

            mov     R0, Z_OUT_1             ;output Z voltage to channel 3
            mov     R1, Z_OUT_2
            clr     M0
            setb    M1
            call    dtoa

            mov     A, X_OUT_2
            anl     A, #0fh;
            cjne    A, #0fh, no_magnet      ;check to see if magnet is to be on from control nibble
magnet:

            mov     R0, #0ffh               ;magnet on sequence
            mov     R1, #0f0h
            setb    M0
            setb    M1
            call dtoa
            jmp     home_magnet

no_magnet:

            mov     R0, #080h               ;magnet off sequence
            mov     R1, #0h
            setb    M0
            setb    M1
            call dtoa

home_magnet:

            mov     A, Y_OUT_2
            anl     A, #0fh
            cjne    A, #00h, lite           ;check to see if light is on


no_lite:
            clr     LIGHT                   ;light off sequence
            jmp     home_lite


lite:
            setb    LIGHT                   ;light on sequence


home_lite:

            ret

;-------------------------------------------------------------------------------------------
;dtoa : Digital to Analogue conversion.  Outputting R0(MSB) R1(LSB)
;-------------------------------------------------------------------------------------------

dtoa:

            mov     DATABUS, R1             ;input 4 bit LSB(left justified)
            setb    A0
            clr     A1
            clr     DAC_CS
            mov     DATABUS, R0
            clr     A0
            setb    A1
            clr     DAC_CS
            setb    DAC_EN

            mov     A, #0d
            setb    TR1
mul_out:                                    ;implementation of 0.05ms delay
            jnb     TIMER_B, mul_out
            clr     DAC_EN
            clr     TR1
            clr     TIMER_B

            ret

;-------------------------------------------------------------------------------------------
;sensor : Storing analogue inputs.
;-------------------------------------------------------------------------------------------

sensor:

            clr     M0                      ;sensing X position
            clr     M1
            call    atod
            mov     X_POS_1, R0
            mov     X_POS_2, R1

            setb    M0                      ;sensing Y position
            clr     M1
            call    atod
            mov     Y_POS_1, R0
```

```
                mov         Y_POS_2, R1

                clr         M0                              ;sensing Z position
                setb        M1
                call        atod
                mov         Z_POS_1, R0
                mov         Z_POS_2, R1

                ret

;-------------------------------------------------------------------------------------------------
;atod : Analogue to Digital Conversion, Storing Input in R0(most sig)
;and R1(less sig)
;-------------------------------------------------------------------------------------------------

atod:
                setb        ADC_EN
                clr         A0                              ;full 12-bit conversion
                clr         ADC_RC
                setb        ADC_CE
                setb        ADC_RC
                clr         ADC_CE
                clr         ADC_EN


                mov         DATABUS, #0ffh
                clr         A0                              ;reading 8 most significant bits
                setb        ADC_RC
                setb        ADC_CE
                mov         R0, DATABUS
                clr         ADC_CE
                clr         ADC_RC

                mov         DATABUS, #0ffh
                setb        A0                              ;reading 4 least significant bits
                setb        ADC_RC
                setb        ADC_CE
                mov         R1, DATABUS
                clr         ADC_CE
                clr         ADC_RC

                ret
end
```