Philippine Engineering Journal (1999) XX (2): 43-65

# NEW CONTROLLER IMPLEMENTATION AND FORCE SENSOR INSTRUMENTATION OF A PUMA ROBOT TOWARDS UNCERTAINTY IDENTIFICATION

**Alvin Chua[1], Edwin Calilung[1], C.J. Sanderson[2], and Jayantha Katupitiya[2]**

[1]Department of Mechanical Engineering
De La Salle University
Manila, Philippines

[2]School of Mechanical and Manufacturing Engineering
University of New South Wales
Sydney, NSW 2052, Australia

## ABSTRACT

*This paper presents a new design and implementation of an interrupt driven control system for a Puma 560 robot, which will provide the facility to implement true real-time control on a PC. In the new controller design, the paper discusses the new hardware configuration, control system adopted, and software design implemented for the Puma robot. Also, a controller tuning procedure was developed for robot joint dc motors to avoid oscillations and overshoot in their response. The inverse kinematics implementation to the software controller will also be explained. Finally, the paper also describes a force sensor integration that will be used together with the new controller for uncertainty (error) identification task. This will include a discussion on the force sensor and its use, the safety system including watchdog timer and crash protector and some experiments to test the effectiveness of the force sensor in uncertainty (error) identification.*

## I. Introduction

Most of the products involved in the manufacturing processes are produced in batch type processes. An increasing number of researches are moving towards the replacement of dedicated workstations by programmable ones. The replacement of the usual pick and place unit to a highly flexible robot is an example of this trend. The robot is usually a single mechanical arm with the movements controlled and programmed through a computer.

Assembly robots with high dexterity are indispensable in highly automated manufacturing systems. Some examples of the assembly robots used in industry are the PUMA, SCARA and the STANFORD robots. The Programmable Universal Manipulator for Assembly (PUMA) robot (see Figure 1) is one of the specialised robots designed for assembly application. General Motors Corporation and Unimation, Inc. developed the PUMA robot arm for the purpose of robotic assembly. Its relatively smaller frame and lighter weight compared to other industrial robots, makes it more adaptable to small and medium scale assembly processes. It could carry a mass of up to 2.5 kg., can apply a static assembly force of 60 N and has a location repeatability of +/- 0.1mm.

One of the outstanding characteristics of the PUMA robot is that its task could be reprogrammed unlike other robots that has a specific task within its lifetime. The current trends leans toward the development of a single station that could assemble different products at the touch of a button. The only barrier to this implementation is that the amount of time spent in programming might make the robot uneconomical.

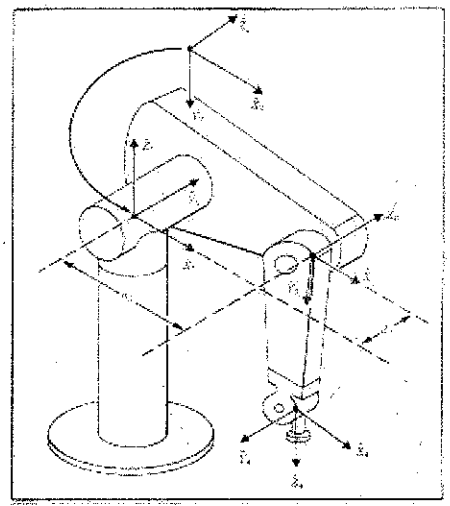

**Figure 1.** PUMA 560 robot (Craig[3])

To reduce the programming time required, the National Bureau of Standards proposed the use of the concept of hierarchical control. With this system, programming time is reduced by creating a modular programming structure to fulfill a given task. The programming scheme is schematically shown in Figure 2.

VAL software was used in the original robot control for the Puma robot. However, there are constraints in the VAL software that makes the implementation of modern control methods difficult. The new control system adopted in the PUMA robot removes this constraint by making use of an IBM-compatible PC to run new control software. First, the new hardware configuration will be described. This is then followed by a discussion of the control system adopted for the joint controls that will also include a new controller tuning procedure to facilitate faster tuning. Finally, the software design implemented for the robot together with the inverse kinematics is explained. To improve the capability of the Puma robot, a force sensor is integrated to the end-effector. This integration process will provide the robot to with force measurements that could be applied in different force control situations. In this paper, the application of the new controller with the integrated force sensor in the robot will be for an uncertainty (error) identification task. An error identification strategy using the Kalman filter and a force/moment sensor is being proposed to solve for the uncertainties.

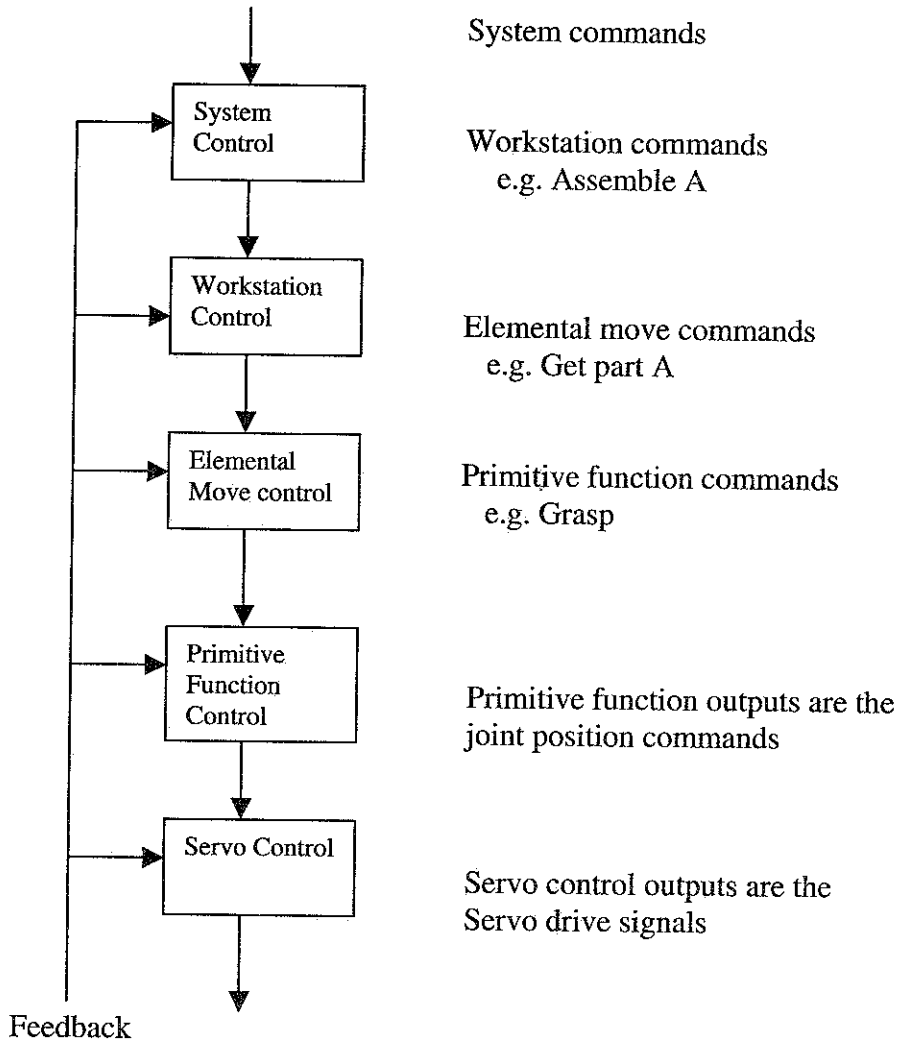


**Figure 2.** Five-level robot control hierarchy (Adapted from A.J. Barbera, National Bureau of Standard Special Publication 500-523, Dec, 1997)

## II. Puma robot controller hardware

An important aspect in the design of the new controller is that it should facilitate flexibility in the implementation of software based control strategies. To conform to this specification, the new system makes a clear distinction between the motor driving hardware and the motor control hardware, whereas the original system contained both hardware devices in the Puma Unimate computer/controller box. The new controller hardware consists of three distinct pieces of hardware (see Figure 3) namely: the original robotic arm; the new motor driving hardware, and the new motor controller hardware, in the form of an IBM-compatible PC.

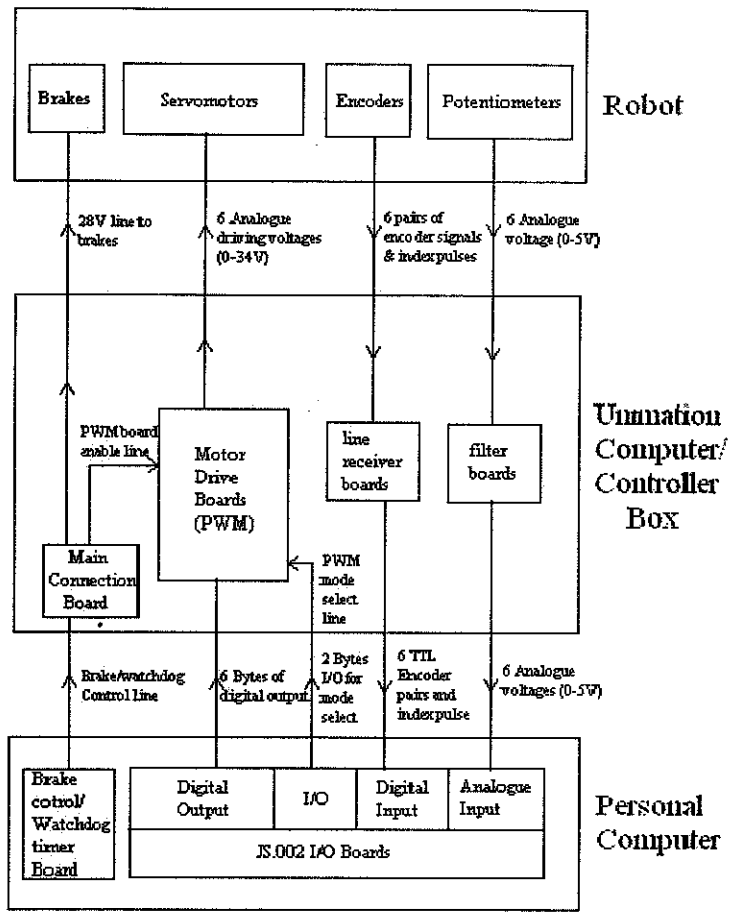


**Figure 3.** Schematic diagram of new system hardware

The controller was implemented entirely on a PC and the original Unimation Controller box contains various interfacing devices like: brake actuation, PWM amplifiers, line receiver boards, filter boards and power supply. This is a total replacement of the original controllers. Two-pulse width modulation (PWM) Motor Drive Boards are used as the new motor driving hardware with each board capable of driving four DC servomotors (four channels, one motor per channel). The first board is used to power joint 1, 2, and 3, the second board powers joints 4, 5, and 6. The PWM is a form of digital voltage control made popular by its simplicity in drive electronics and computer interfacing. In general terms, the voltage across the output terminal of one channel is controlled by electronically adjusting the duty cycle, i.e., the width of the pulse. The line receiver boards were used to validate the encoder signals so that noise glitches can be eliminated. While the filter boards were installed to filter the potentiometer signals coming from the robot joints.

### 2.1 The new PC controller hardware

The implementation of control software routines for each joint was done through the use of a personal computer (PC). The software routines can be written in any appropriate language (in this case, Borland C++). Inside the PC, two I/O boards are installed to allow communication between itself and the controller box, together with the watchdog timer/brake control board to safety actuate the joint brakes.

#### 2.1.1 JS.002 PC I/O Boards

The JS.002 is a multi-functional I/O card used to communicate with the PC. The board has the following facilities: a) Two 24 bit quadrature counter inputs, b) One 16 bit quadrature counter input, c) Six digital index pulse inputs, d) Six bytes of TTL digital output, e) Three bytes of digital I/O, f) Eight channels of analogue input to A/D converter, and g) Timer

#### 2.1.2 Watchdog Timer/Brake control boards

The Watchdog Timer/Brake control board has the purpose of notifying the controller box in the event of the PC crashing. It also controls the enabling and disabling of the brakes. This board sends one control line from the PC to the main connection board in the controller box. This line will only be activated if both the watchdog and the brake are enabled.

The watchdog timer has the function of periodically checking whether a particular memory location has been repeatedly read by the PC's microprocessor. The board was configured to check the memory location is at least read every 20 milliseconds. If this memory location is not read, the control line will be disabled. This will cause the controller hardware to apply the brakes and remove the power for the PWM boards.

The 'brake enable' function of the board controls the releasing of the brakes. The main connection board produces the 28 V when the control line from the PC is activated. This will then release the brakes when applied across the brake connections on the arm cable board. The watchdog timer and the brake enable should both be enabled for the control line from the PC to be activated.

## III. Puma robot control system

The use of the PC for the control and monitoring of this robot has been adopted on the grounds of flexibility, and the relative simplicity involved with the design of a very robust, high performance controller.

### 3.1 Position and velocity controllers of the Puma robot

Due to the variation in load conditions of the robot arm, the DC servomotor controls for each joint call for a control loop that will enable the motor to adjust. Two main loops are incorporated in the control algorithm of the servomotors of the Puma robot. The inner loop is a *velocity feedback controller*, which is used to ensure the motor follow a particular velocity profile. The outer loop, which is the *position feedback loop*, calculates the velocity profile to be followed based on motor shaft position error.

### 3.1.1 Velocity controller

The inner loop of the control algorithm is a Velocity Feedback Loop (see Figure 4) which attempts to make the motor follow a given velocity profile. The controller compares the velocity at which we want the motor to travel, with its actual velocity. The difference between these values is used by the controller to calculate the Control Effort required to drive the motor at the required velocity.

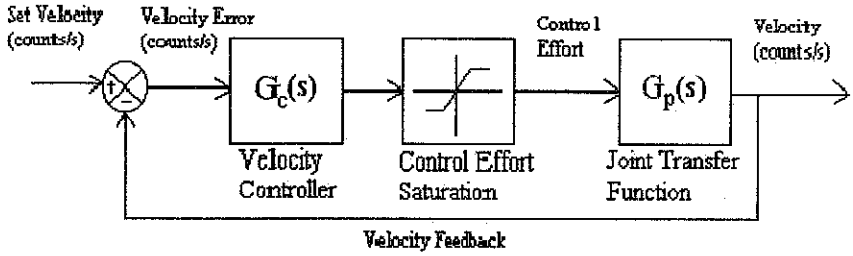


**Figure 4.** Block diagram of the velocity feedback loop

where $G_c(s)$ is the transfer function of the velocity controller and $G_p(s)$ is the joint transfer function (dc motor). The control effort saturation ensures that the controller does not ask for a control effort, which is physically unattainable.

### 3.1.2 Position controller

The outer loop of the controller is a Position Feedback Loop (see Figure 5). The purpose of this part of the controller is to create the desired velocity profile. The velocity feedback loop then attempts to follow this velocity profile. The input to the loop is the Set Point and is the joint's destination in encoder counts. The position feedback loop works by calculating the position error.

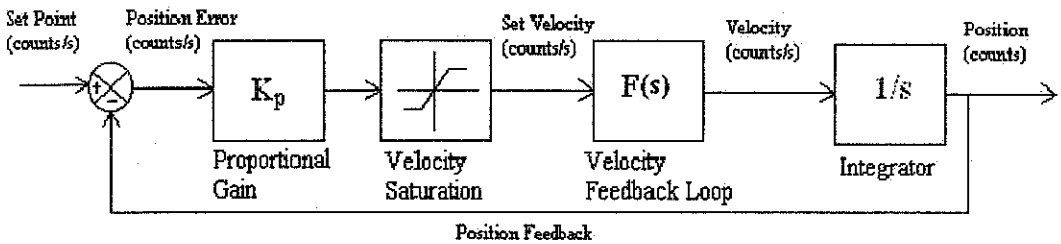


**Figure 5.** Block diagram of the position feedback loop

where $K_p$ is the proportional gain constant which has the effect of increasing the rate at which the motor renders its maximum velocity and F(s) is the transfer function of the whole velocity feedback loop discussed earlier.

## 3.2 Velocity controller tuning procedure

In the course of studying the controller of the Puma robot servomotors, it was found that the proper knowledge of controlling the position and velocity of a DC motor is important in many

applications. Although we could find some design guidelines on controlling the motors in literature [2], they are limited. A PI controller is used in the present Puma robot controller. A direct way of determining the tuning coefficients of the PI controller will lead to a more efficient controller design.

This section determines the bounds of the $K_p$ and $K_I$ coefficients for velocity control of a given DC motor. This will make the controller design approach a more bounded approach rather that a trial and error procedure presently implemented in systems. To aid the designer in the procedure, Matlab codes will be included in the paper. At the same time, the use of Matlab as a controller design tool will be implemented to test the effectiveness of the method in the design of the controller.

### 3.2.1 Modeling the DC motor

According to Golten [3] and Ogata [4], although a DC motor is not exactly a purely linear system, it could be modeled as if they were. In the continuous time domain (s-plane), it could be represented by the mathematical equation:

$$v(t) = v_0(1 - e^{-t/\tau}) \tag{1}$$

where v(t) is the velocity at t, $v_o$ is the steady state velocity of the motor, and $\tau$ is the mechanical time constant of the motor

The input (driving function) is the applied voltage across the servomotor armature in volts. The output (response function) is the motor's angular velocity in encoder counts per second. The Laplace Transform of equation (1) is of the form found in Figure 6:

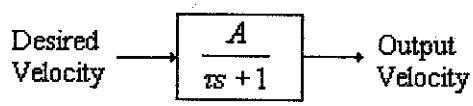


**Figure 6.** Transfer function of a DC motor

The motor is subjected to step input and the response of each motor was observed. This was achieved by applying a step voltage to motor and allowing the corresponding joint to move. Comparing the actual response to the simulated response verified the validity of the model. When designing a controller, the objective is to design it such that the physical system responds in a desired manner. Based on the model, it is possible to predict how each of the motors will respond to different input conditions in the continuous time domain.

### 3.2.2 Design of a velocity controller

The velocity controller attempts to make the motor follow a given velocity profile as specified by the designer. The controller compares the desired velocity with the actual velocity. The difference is used in working out the Control Effort required to drive the motor at the required velocity.

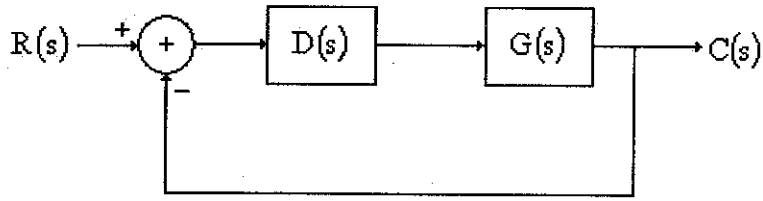


**Figure 7.** Block Diagram of the Velocity Feedback Loop

where D(s) is the transfer function of the controller (PI) in the s-plane, G(s) is the transfer function of the plant (DC motor) in the s-plane, R(s) is the system input, and C(s) is the system output. Following the modeling process outlined by Golten [3], the experimentally determined transfer function of a DC motor is given by

$$G(s) = \frac{0.2341}{1.25s + 1} \tag{2}$$

The overall transfer function of a motor control system with PI controller can be defined as follows:

$$T(s) = \frac{C(s)}{R(s)} = \frac{0.1873K_D s^2 + 0.1873K_p s + 0.1873K_I}{(1 + 0.1873K_D)s^2 + (0.8 + 0.1873K_P)s + 0.1873K_I} \tag{3}$$

The above transfer function can be further analyzed by determining the location of poles and zeros for different values of $K_p$ and $K_I$ Figure 8 and 9 show the poles and zero location of T(s) for different values of $K_p$ and $K_I$ . It can be observed that for poles that lie on the real axis, the pole near the origin is approximately on the same location as the zero. For large values of $K_p$ one of the poles cancels out with the zero. If controllers will be designed in such a way that one of the poles cancels out with the zero, the time constant, setting time and rise time will simply depend on the pole that is far from the origin.

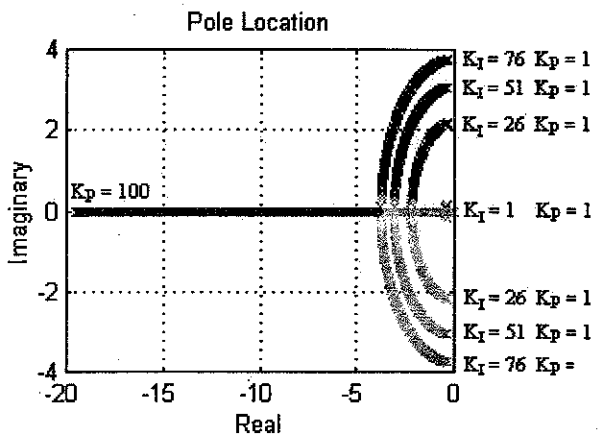


**Figure 8.** Pole location of T(s) for different values of $K_p$ and $K_I$

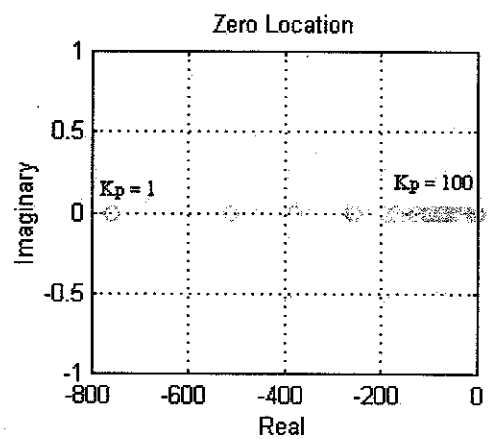


**Figure 9.** Location of zeros of T(s) for $K_I$=76 and for different values of $K_p$

To ensure that oscillation is avoided, the dominant poles should lie on the real axis. The chart shown in Figure 10 shows the minimum $K_p$ for a given $K_I$ that will ensure the poles to lie on the real axis. This figure is generated using the Matlab code given below.

```
k=[];
for ki=1:25:100
   clear r;
   r=[];
   z=[];
   inc=0.1;
   for kp=0:inc:100
      den=[1 0.8+0.1873*kp 0.1873*ki];
      r=[r;roots(den)];
   end
   x=real(r);
   y=imag(r);

   [val,indx]=min(abs(y));
   k=[k;(indx-1)*inc*0.5];
end;

plot(1:100,k);
```

For example, for $K_I = 20$ the minimum $K_p$ to ensure that poles lie on the real axis is 16.4.

This corresponds to pole location - 1.9753 and -1.8964 and zero at -1.2195.

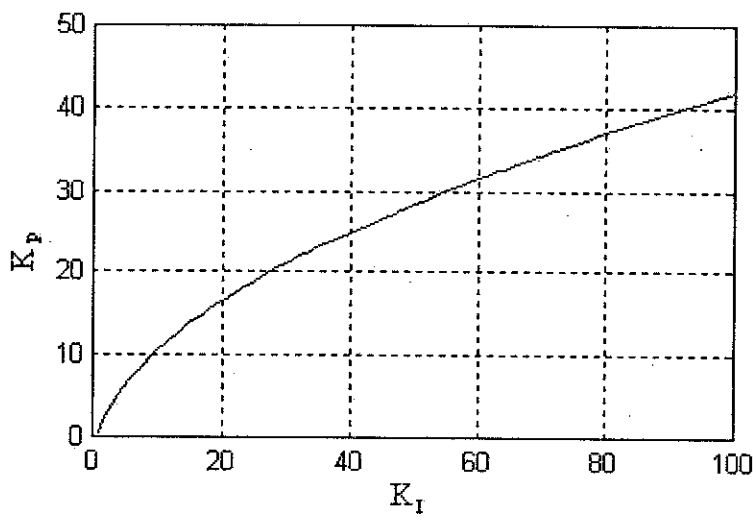


**Figure 10.** Relationship between $K_p$ and $K_I$ that will ensure that poles are real

### 3.2.3 Experimental Result

The previously assumed model is the experimentally determined model of the motor. To improve the steady state error, a PI controller is utilized. To verify the validity of the chart shown in Figure 10, the coefficients $K_p$ and $K_I$ were varied and the corresponding output response are compared to the simulated response. For $K_p = 10$ and $K_I = 40$, the chart shown in Figure 10 shows that the output response will have an overshoot and this is evident in the actual/simulated response shown in Figure 11.

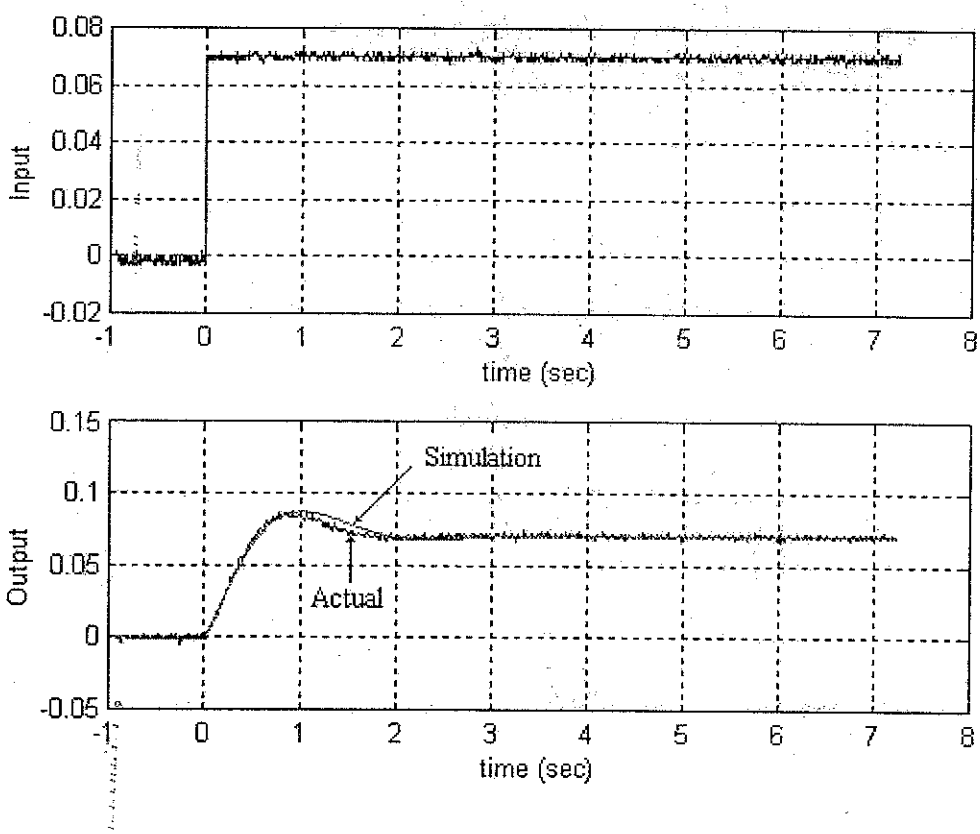


**Figure 11.** Input/Output response for $K_p = 10$ and $K_I = 40$

To avoid overshoot the chart shown in Figure 10 tells the controller designer to select values of $K_p$ and $K_I$ above the curve. Setting $K_p = 20$ and $K_I = 20$, the corresponding output response is shown in Figure 12. It can be noted that the steady state error is approximately equal to zero. Given a velocity profile shown in Figure 13, the output response is plotted on the same figure. It can be noticed that the output (actual/simulation) response is very closed to the given velocity profile.

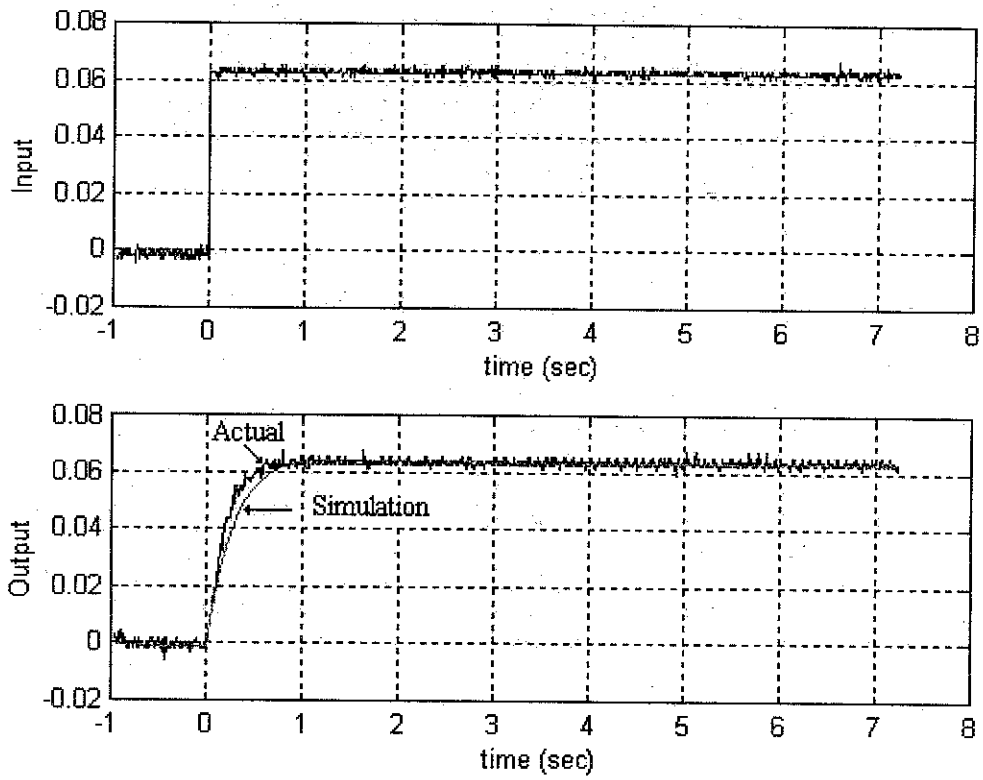


**Figure 12.** Input/Output response for $K_p$= 20 and $K_1$ = 20

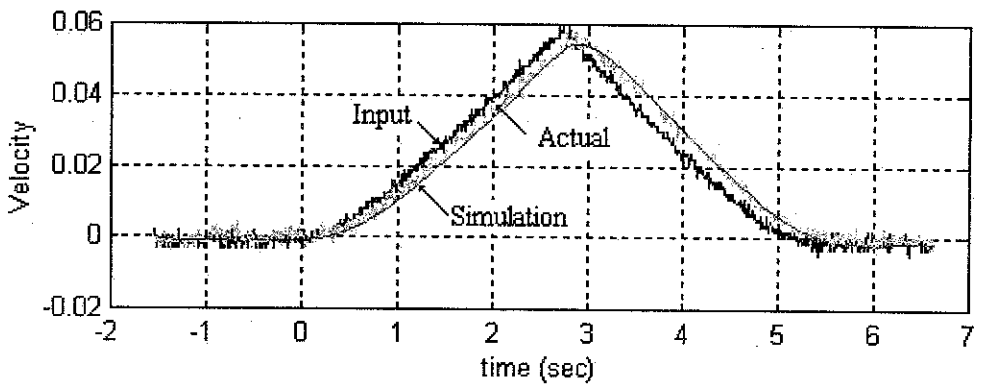


**Figure 13.** Output response given an increasing input

# IV. Puma robot control software design

Implementation of a real-time controller on a PC requires the use of timer interrupts. The software design process centers on the use of interrupts to implement the sampling time needed for the controller. The foreground tasks are the inverse kinematics and screen update while the background task is the Interrupt Service Routine (ISR).

## 4.1 The use of interrupts

Approximately every 55 milliseconds, the PC timer generates a hardware interrupt. It is connected to the highest priority line of the Intel 8259A (Interrupt Controller), and has the interrupt number 0x08. The Interrupt Service Routine (ISR) 0x08 is pre-programmed to do the following: a) Update the system timing, b) Jump to a dummy interrupt, and c) Send the end of interrupt (EOI) back to the interrupt controller.

The ISR can be replaced by a new ISR defined by the programmer. This has the effect of executing the new ISR code every time interrupt is generated. The following three functions written in C++ accomplish this task.

First, we need a function that will store the address of the old ISR and direct the program to run the new ISR. This process is known as installing the new ISR.

```
void install (int inum, void interrupt far (*new_isr)(...))
{
      disable();
      OldIsr = getvect(inum);
      setvect(inum,newisr);
      enable();
}
```

The code for the new interrupt routine needs to be written. This code is executed every time the timer interrupt is generated.

```
// New interrupt subroutine
void interrupt far NewIsr(...)
{
      // Place body of new interrupt subroutine here
      outportb(0x20,0x20);
}
```

Just before exiting the robot control program, the original interrupt service routine must be restored. This is achieved by executing the routine given below.

```
void unistall (int inum, void interrupt far (*old_isr)(...))
{
      disable();
      setvect(inum,old_isr)
      enable();
}
```

The main program that uses these three functions is outlined in the following program fragment. *Quit* is a variable defined in the program that sets to 1 to exit the program.

```
#DEFINE INTR 0x08
void interrupt far (*OldIsr)(...);
```

```
main()
{
…
        install (INTR, NewIsr);
        while (!Quit());
        uninstall(INTR,OldIsr);
…
}
```

This program installs the new ISR at the position of ISR 08. This new interrupt subroutine will be run every 55 milliseconds until the keyboard is hit. Once the keyboard is hit, the new ISR is uninstalled and the old ISR will be returned to its original position. We can also change how often the PC generates this timer interrupt and so the sampling time can be altered. Figure 14 shows a flowchart of the main program for the joint control with the ISR. The new ISR run every FIVE ms.

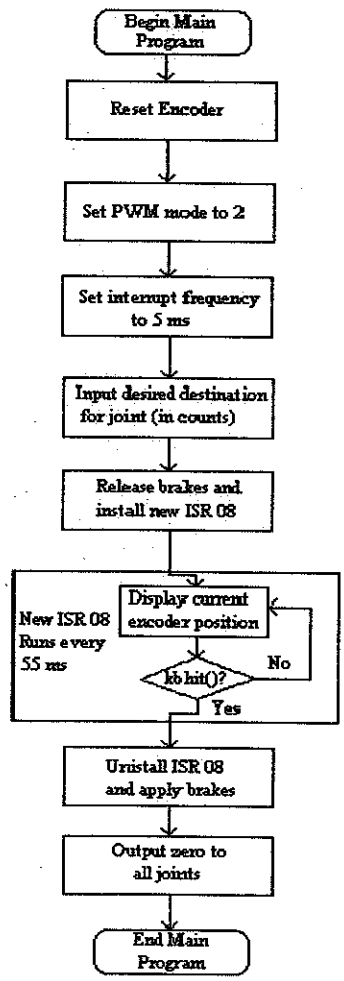


**Figure 14.** Flowchart of the main program for joint control w/ISR

## 4.2 Implementation of controller for simultaneous control of all joints

There are two types of motion for all six joints of the Puma robot namely: Fastest Motion and Linearly Interpolated Motion. The fastest motion will move all the joints at their top speeds while the linearly interpolated motion ensures that each joint arrives at its destination at the same time.

## 4.3 Calibration Software

The calibration software establishes a reference point or initial absolute position of the robot for the controller. Therefore, the calibration routine should be executed before any other action is performed. The calibration software performs two tasks: reading of analogue inputs from the potentiometers, and the detection of index pulses from the shaft encoders. The robot joints are slowly turned until the potentiometer-reading fall within a prescribed range after which the encoder index pulse is searched with that range. There is only one encoder index pulse that can be found within that range and when the index pulse is found, the encoders are reset to 0, thereby establishing the calibrated zero point for that particular joint.

## 4.4 Implementation of Inverse Kinematics

*Kinematics* is the relationship between the positions, velocities and accelerations of the links of a manipulator, where a manipulator is an arm, finger, or leg [1].

There are two problems present in the analysis of the position kinematics of a given manipulator, namely: Forward kinematics and Inverse kinematics. In Forward kinematics, the forward transformation equations ($^{R}\mathbf{T}_{H}$) are solved to find the location of the robot end point in terms of the angles and displacements between the links. The Inverse kinematic involves solving the inverse transformation equations ($^{R}\mathbf{T}_{H}^{-1}$) to find the corresponding joint coordinates from the Cartesian coordinates of the robot end point in space (Figure 15). The angles and displacements between the links are called joint coordinates and are described with link variables, while the location of the robot end in space is described with Cartesian coordinates.

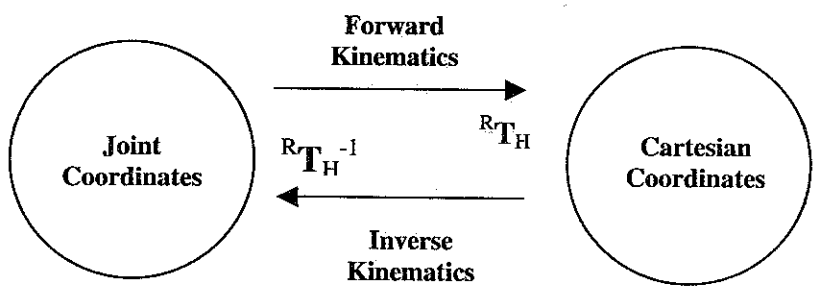


**Figure 15.** Joint and Cartesian space kinematic mapping

The inverse kinematic equations incorporated in the controller of the Puma robot was taken from the derivations given in Craig [1]. There are several solutions proposed fort he inverse kinematics problem. The method used here is the inverse transforms technique for Euler angles.

The Cartesian coordinate values of the robot are known and the inverse kinematics will convert them to its equivalent joint coordinates. The Cartesian coordinate values are given in matrix form below:

$$ {}^{0}_{6}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [4] $$

The value of $r_{11}$, $r_{12}$, $r_{13}$, $r_{21}$, $r_{22}$, $r_{23}$, $r_{31}$, $r_{32}$, $r_{33}$, $p_x$, $p_y$, and $p_z$ are given values. The dimensional parameters (a3, d3, a2, d4) are given by the type of Puma robot considered. Below are the equations of the transformation from the Cartesian to joint coordinates.

**Table 1**
Inverse kinematics equations

**Joint 1:** (5)

$$\theta_1 = A\tan 2(p_y, p_x) - A\tan 2(d_3, \pm\sqrt{p_x^2 + p_y^2 + d_3^2})$$

**Joint 2:** (6)

$$K = \frac{p_x^2 + p_y^2 + p_z^2 - a_2^2 - a_3^2 - d_3^2 - d_4^2}{2a_2}$$

$$\theta_3 = A\tan 2(a_3, d_4) - A\tan 2(K, \pm\sqrt{a_3^2 - d_4^2 - K^2})$$

**Joint 3:** (7)

$$c_1 = \cos(\theta_1)$$

$$s_1 = \sin(\theta_1)$$

$$c_3 = \cos(\theta_3)$$

$$s_3 = \sin(\theta_3)$$

$$a = (-a_3 - a_2 c_3)p_z - (c_1 p_x + s_1 p_y)(d_4 - a_2 s_3)$$

$$b = (a_2 s_3 - d_4)p_z + (a_3 + a_2 c_3)(c_1 p_x + s_1 p_y)$$

$$\theta_{23} = A\tan 2(a, b)$$

$$\theta_2 = \theta_{23} - \theta_3$$

**Joint 4:** (8)

$$c_{23} = \cos(\theta_2)\cos(\theta_3) - \sin(\theta_2)\cos(\theta_3)$$

$$s_{23} = \cos(\theta_2)\sin(\theta_3) + \sin(\theta_2)\cos(\theta_3)$$

$$\theta_4 = A\tan 2(-r_{13}s_1 + r_{23}c_1, -r_{13}c_1 c_{23} - r_{23}s_1 c_{23} + r_{33}s_{23}$$

(9)

**Joint 5:**

$$c_4 = \cos(\theta_4)$$

$$s_4 = \sin(\theta_4)$$

$$s_5 = -(r_{13}(c_1c_{23}c_4 + s_1s_{23}) + r_{23}(s_1c_{23}c_4 - c_1s_4) - r_{33}(s_{23}c_4))$$

$$c_5 = r_{13}(-c_1s_{23}) + r_{23}(-s_1s_{23}) + r_{33}(-c_{23})$$

$$\theta_5 = A\tan 2(s_5, c_5)$$

**Joint 6:** (10)

$$s_6 = -r_{11}(c_1c_{23}s_4 - s_1c_4) - r_{21}(s_1c_{23}s_4 + c_1c_4) + r_{31}(s_{23}s_4)$$

$$c_6 = r_{11}(c_1c_{23}c_4 + s_1c_4)c_5 - c_1s_{23}s_5) + r_{21}((s_1c_{23}c_4 - c_1s_4)c_5 - s_1s_{23}s_5) - r_{31}(s_{23}c_4c_5 + c_{23}s_5)$$

$$\theta_6 = A\tan 2(s_6, c_6)$$

### 4.4.1 Kinematic Calibration of the robot

Before the implementation of the inverse kinematics program to the Puma robot controller, a calibration of centres was done. The calibration of centres will determine the robot kinematic constants and the proper *home position*. The *home position* is the initial joint setting of the Puma robot based on the deviation of the inverse kinematic equations.

The calibration was done using a theodolite to guarantee the levelness and accuracy of the robot while turning at each joint. The calibration was mainly done on the first three main joints of the Puma robot. An approximate starting point was chosen as the position for calibration of the robot. From the initial reference position, the theodolite was used to indicate the correction needed to find the joint centres of the Puma robot. In joint 1, an initial marking was done then the joint was turned, if the mark stays in the location as indicated by the theodolite then centre is found.

With all the centers found in each joint, the kinematic constant are determined by measuring the displacement from each of the joints. The kinematic constants are shown in Table 2. (Refer to these constant in the very diagram of the PUMA robot).

**Table 2**
Puma robot constants

| **Puma 560 robot kinematic constants** | |
|---|---|
| a3 | 19.1 mm |
| d3 | 125.4 mm |
| a2 | 431.8 mm |
| d4 | 431.8 mm |

The home position of the Puma robot was found by moving the robot to the position shown in Figure 1. Please take note that the home position setting was based on the calibration position of the Puma robot. The calibration position initializes the position of the joint motors before the robot does any task. The center point found at each joint was used as the basis to assure the levelness of the robotic home position. From this experiment, the joint counts necessary to go to the home position were found. The values for each joint are given in Table 3.

**Table 3**
Values of the Puma Home Position

| Home Position (counts) |
|---|
| SetPoint1=0;<br>SetPoint2=27360;<br>SetPoint3=-12517;<br>SetPoint4=0;<br>SetPoint5=-499;<br>SetPoint6=0; |

### 4.4.2 Forward and Inverse Kinematic Program Simulation Results

In order to check the inverse kinematics program, a simulation test was done. This test made use of the corresponding forward kinematics to check the output values of the inverse kinematic program. The results are shown in Table 4.

**Table 4**
Results of the Inverse and Forward kinematics programs

| Input to program | Cartesian coordinates | Inverse kinematics | Forward kinematics |
|---|---|---|---|
| Rot x-axis = 30° | r11 = 0.492404 | $\theta_1$ = 0.0218654 rad | r11 = 0.492404 |
| Rot y-axis = 40° | r12 = -0.456826 | $\theta_2$ = 0.0337389 rad | r12 = -0.456826 |
| Rot z-axis = 50° | r13 = 0.740843 | $\theta_3$ = 0.0149781 rad | r13 = 0.740843 |
| Px = 5 mm | r21 = 0.586824 | $\theta_4$ = 3.02453 rad | r21 = 0.586824 |
| Py = 10 mm | r22 = 0.802872 | $\theta_5$ = 2.27754 rad | r22 = 0.802872 |
| Pz = 15 mm | r23 = 0.10504 | $\theta_6$ = 0.54031 rad | r23 = 0.10504 |
| | r31 = -0.642788 | | r31 = -0.642788 |
| | r32 = 0.383022 | | r32 = 0.383022 |
| | r33 = 0.663414 | | r33 = 0.663414 |
| | Px = 5 mm | | Px = 5.00002 mm |
| | Py = 10 mm | | Py = 9.99999 mm |
| | Pz = 15 mm | | Pz = 15 mm |

As could be seen in Table 4, the inverse kinematics program worked since the values of the Cartesian coordinates in column 2 corresponds to the result of the forward kinematics as found in column 3 with inverse kinematics results as input. The corresponding joint coordinates for the Puma robot using the gear ratios found in Table 5 will be: Joint 1 = -217 counts, Joint 2 = -577 counts, Joint 3 = -127 counts, Joint 4 = 36680 counts, Joint 5 = 21604 counts, and Joints 6 = 6595 counts.

**Table 5**
Gear Ratios and Encoder counts/revolution

| Joint | Gear Ratio | Counts/revolution of joint | Counts/motor revolution |
|---|---|---|---|
| 1 | 62.6 | 62600 | 1000 |
| 2 | 107.5 | 107500 | 1000 |
| 3 | 53.5 | 53500 | 1000 |
| 4 | 76.2 | 76200 | 1000 |
| 5 | 59.6 | 59600 | 1000 |
| 6 | 76.7 | 76700 | 1000 |

# V. Force Sensor Integration

## 5.1 Force Sensor and Data Acquisition Program

A JR3 force-moment sensor (see Figure 16) was used to measure the necessary forces and moments for the Puma end-effector. It weights 0.4lb (180g) and can carry a maximum load of 200 N in the z-axis and 100 N for the x-axis and y-axis. It can also carry a maximum moment of 12.5 N-m in all axes. The force sensor comes with an interface card that links the PC with the sensor itself. A data gathering and monitoring program was created to gather data from the six-axis force sensor. The program was written in Borland C++ and the corresponding built-in commands in the interface card were use to gather the force data.

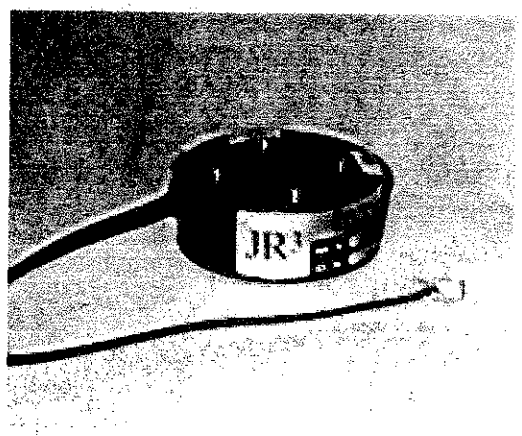


**Figure 16.** JR3 force sensor

A force/moment data acquisition program was designed for the robotic contact experimentation. The data acquisition program of the force sensor consists of four main tasks.

- The first task initializes the measurement of the force/moment data to a zero initial value. It cancels out the noise inherent to the system and sets the initial zero condition for the force/moment data measurement. This function is usually activated before actual measurements are acquired using the force sensor.
- The second task displays the different data available in the force sensor interface card at any point in time. It shows the configuration of the card and displays the results of the force/moment data at different filtering conditions.

- The third task saves a specified number of force/moment data for a given time. It stores the data to a file specified by the user. These data could then be used for data analysis and testing.
- The final task takes care of range of force/moment data acquired. Based on the specifications of the force sensor used, the maximum values of the six force/moment data are the following Fx=200N, Fy=200N, Fz=400N, Mx=12.5N-m, My=12.5N-m, and Mz=12.5N-m.

## 5.2 Force sensor effectiveness testing

To test the accuracy and effectiveness of the force-moment sensor in getting the uncertainties or state variables, a simple set-up was fabricated as shown in Figure 17. The uncertainties/state variables are quantities to be identified through the force/moment sensor measurements. Since the force/moment data gathered are noisy we used a filtering technique like the Kalman filters to compensate for the noise and identify the state variables. The setup consists of an inclined block and some weights. The force sensor was mounted on the inclined block while weights were placed on top of the force sensor. Three different state variables were to be found namely, the angle of inclination relative to the horizontal (theta), the angle of rotation of sensor (alpha), and the moment arm (r) (Figure 17).

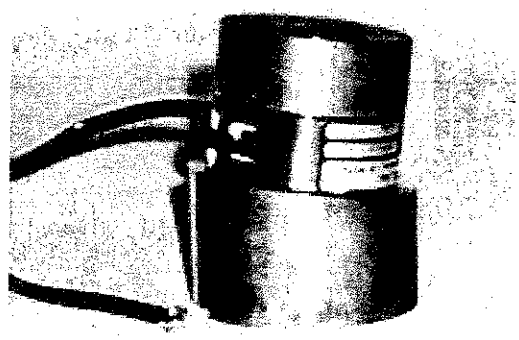

**Figure 17.** Force Sensor and Kalman Filter Effectiveness Set-up

A program was created using Matlab to find the uncertainties or state variables. In creating the program, the measurement model of the set-up was solved using basic statics. The measurement model relating the state variables and measurement data (forces and moments) are given below.

$$
\begin{aligned}
Fy &= -mg\,Sin\theta\,Sin\alpha \\
Fx &= mg\,Sin\theta\,Cos\alpha \\
Fz &= -mg\,Cos\theta \\
Mx &= mgr\,Sin\alpha\,Sin\theta \\
My &= mgr\,Cos\alpha\,Sin\theta \\
Mz &= 0
\end{aligned}
\qquad (11)
$$

The equations show a non-linear relationship between the variables so the Extended Kalman filters were used to solve the uncertainties. In the graph, the straight lines indicate the true values of the state variables used (theta = 7 degrees (0.122 rad), alpha = 45 degrees (0.785 rad), r = 25 mm).

Experimental data were then taken and passed through the Kalman filter program. It showed a very good estimate with a certain degree of error. The final estimates of the state variables are shown in Figure 18.

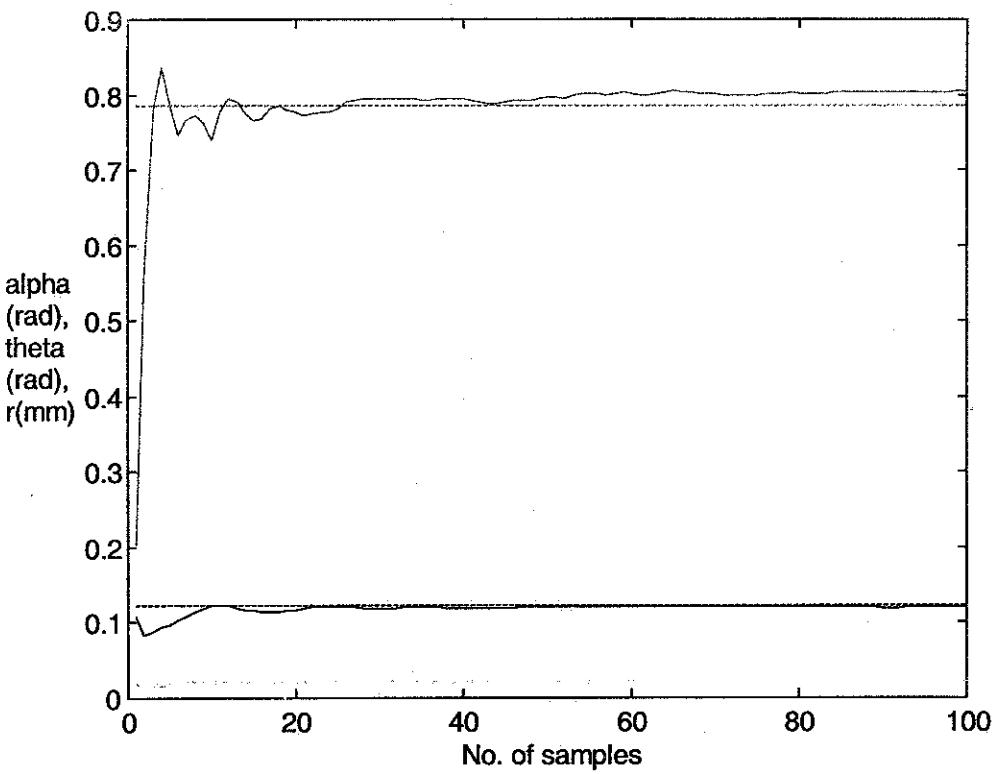


**Figure 18.** Estimation using a force sensor

## 5.3 Crash protection set-up

In order to protect the force sensor in accidental crashes, a crash protection device was installed with a model OPD-MS-2HD (see Figure 19) which mechanically interfaces directly to the JR3 force sensor. To provide a more sensitive adjustment in the crash activation, an electronic module (OPD-EM-12) was installed (see Figure 20).

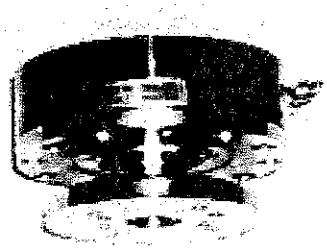

**Figure 19.** Overloaded Protection Device (OPD)

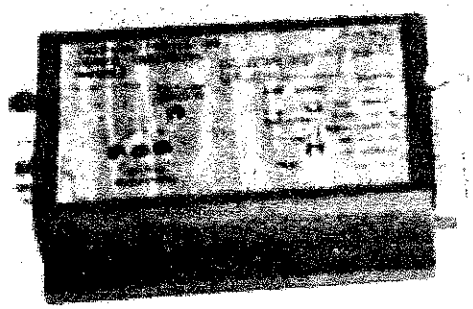

**Figure 20.** Electronic Module (EM) for the OPD

The Electronic module (EM of the crash protection device was incorporated into the Puma robot controller through its output signal. The output of the EM was connected the controller using the circuit given below. The Pin Configuration of the Input/Output Box Headers of JS002 PC interface card is shown together with the connection to the EM terminals for crash detection.

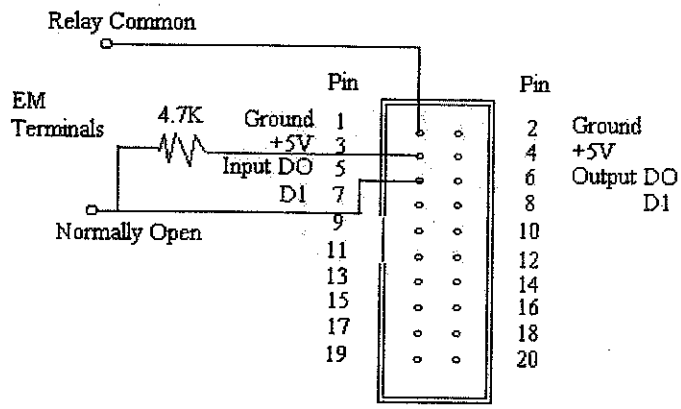


**Figure 21.** Crash Protection Signal Interface

The program for detecting the crash signal by the controller is shown below:

```
int NotCrashed = inportb(BASEA+7) & 0x01;
if(!NotCrashed)
{
        SetPoint1 = SAFE1;
        SetPoint2 = SAFE2;
        SetPoint3 = SAFE3;
        // Fastflag
}
```

The controller looks at an address to determine the NotCrashed condition of the robot. If the NotCrashed variable is activated, it then drives the joints 1 to 3 of the Puma robot to a specified safe position (SAFE1, SAFE2 and SAFE3).

## VI. Conclusion

This paper described a new controller hardware and software for a Puma robot. The new system is shown to be effective in the implementation of new control strategy for robot joints. The position and velocity feedback loops were implemented to compensate for different loadings on the robot arm. Anew tuning procedure was shown to be effective in minimizing the overshoot and oscillations in the response of the dc motors of the robot joints. It could be used in future design of controllers for a more efficient implementation. The inverse kinematics implemented to the control software was shown to be effective in moving the arm to a specific Cartesian space point. Finally, the force sensor system was successfully tested and installed at the end of the robot arm for future experimentation studies (e.g. uncertainty (error) identification, force control). On the whole, the new controller design enhanced the capability of the robot in performing tasks and new control strategies can be tested for further development.

## Reference


1. Craig, J., (1989). *Introduction to Robotics Mechanics and Control* 2nd ed. Addison-Wesley Publishing Company.
2. Nise, N. (1996). *Control Engineering.* Addison-Wesley Publication.
3. Golten, J. and Vewer, A. (1991). *Control System Design and Simulation.* McGraw-Hill, New York.
4. Ogata, K. (1990). *Modern Control Engineering.* Second ed. Prentice Hall, New Jersey.
5. P. McKerrow. (1991). *Introduction to Robotics.* Addison-Wesley Publishing Company.
6. Gelb, A. (1974). *Applied Optimal Estimation.* The MIT Press. London, England.
7. J. Mendel. (1985). *Lessons in Estimation Theory for Signal Processing, Communications, and Control.* Prentice Hall International edition.
8. Bar-Shalom and X. Li. (1993). *Estimation and Tracking, Principles, Techniques, and Software.* Artech House.
9. J. Katupitiya, S. Dutre, S. Demey, J. De Geeter, H. Bruyninckx and J. De Schutter (1996). Estimation Contact and Grasping Uncertainties Using Kalman Filters in Force Controlled Assembly. *In Proc. Int. Conf. On Intelligent Robots and Systems,* pp. 696-703.
10. J. Katupiya, R. Rajedewski, C.J. Sanderson and M.J. Tordon (1997). Implementation of a PC based Controller for a PUMA robot. *Fourth International Conference on Mechanics and Machine Vision in Practice*, Toowoomba, Australia.