Computer Controlled Multi-Axis Robot

By

Jeffery L C Yang

A thesis submitted in
Partial fulfilment of the requirements for
The Degree of Masters of Engineering

In the

Department of Mechanical Engineering
University of Canterbury
Christchurch, New Zealand

August 2000

To the loving memories of My Grand Father, Ah Gong

ABSTRACT

The Real-Time Linux operating system was evaluated for implementing real-time digital controllers. Real-Time Linux is flexible and easy to program compared to ordinary digital controllers such as Digital Signal Processor (DSP) and micro-controller. Real-Time Linux was successfully used to control a single-axis hydraulic test rig and a Unimate 2000B six-axis hydraulic robot. Real-Time Linux proved capable of running a six-axis digital controller using floating-point calculations. On a 486-66 MHz Personal Computer (PC), a controller sampling time of 1 ms required less than half the available computation time.

The direct and inverse robot kinematics was based on the Denavit-Hartenberg parameters and a straightforward PID six-axis controller proved capable of basic path control. While more sophisticated robot controllers and dynamic compensation would require additional computation power, experiments have shown that they can still be run on the Real-Time Linux system.

ACKNOWLEDGEMENT

First of all, I would like to thank my supervisor Dr. Reg Dunlop for suggesting the use of Real-Time Linux for digital control. I am extremely grateful to Andy Cree, for his invaluable technical assistance and insight into hardware and software.

Many thanks to Ra Cleave for his help and advice on the Linux operating system and to Dr. Andrew Lintott for pointing out my problem on the kinematics of the Unimate 2000B. Thank you to Dr. Ian Huntsman who installed the first Real-time Linux kernel on my machine and to David Haywood for his insights into the C programming language.

To my brother Terence and flatmate Carolyn, thank you for your support in every way and for being the stable part of my life for the pass two years. Thank you to Tony for being there and overall, being such a great friend.

Finally, My utmost admiration to the Real-Time Linux and Linux community. Thank you for your hard work into creating such a brilliant FREE computing environment.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Symbol	Meaning	Units
α	Rotation around the z axis	degrees/ rads
β	Rotation around the y axis	degrees/ rads
γ	Rotation around the x axis	degrees/ rads
L_1, L_B	Length of wrist bend	mm
L_2, L_Y	Length of swivel	mm

A/D	Analogue to Digital
C	A high level programming language
C++	Object orientated C
DC	Direct Current
D-H	Denavit-Hartenberg
DSP	Digital Signal Processor
fifo	First in first out
fpu	Floating point unit
GNU	GNU is Not Unix
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
I/O	Input/ Output
Matlab	Matrix Laboratory (A mathematical software that has both numerical and analytical analysis.)
P.I.D.	Proportional Integral Derivative
PC	Personal Computer
PWM	Pulse Width Modulation
RAM	Random Access Memory
RTAI	Real-Time Application Interface
rtf	Real-time fifo (no different from a fifo)
RTiC-Lab	Real-Time Control Laboratory

RTL	Real-Time Linux
RTLT	Real-Time Linux Target
Simulink	A simulation toolbox in Matlab.
UI	User Interface
UPP	Universal Pulse Processor

CHAPTER 1- INTRODUCTION

1.1 BRIEF DESCRIPTION OF PROJECT

The objective of this project was to assess the suitability of Real-Time Linux (RTL) for robotics and machine control. First RTL was used to run controllers of varying sophistication for a single-axis hydraulic test rig and later it was used on a six-axis hydraulic manipulator (robot). The reason for using RTL on the single-axis hydraulic test rig was to become familiar with RTL before using it on a more complex manipulator. Details of the single-axis hydraulic test rig are discussed in chapter 3 while the manipulator work is discussed in chapters 4&5.

RTL was utilised for its fast response capability. When run on a 486-66 PC, it is capable of responding to an interrupt within a maximum of 55μs. This allows RTL to control machines very well since the control algorithms need to be serviced in a fast and timely manner. Details of RTL are discussed in chapter 2.

Both projects utilised a Universal Pulse Processor (UPP) card for interrupt generation, position signal detection for and generating PWM waves. RTL is used to control the UPP card. The positions of the machines' read by the UPP card and then passed back to RTL so that position error and the size of the control signal can be calculated. This control

signal size determines the duty cycle of PWM wave generated by the UPP card to control the machines.

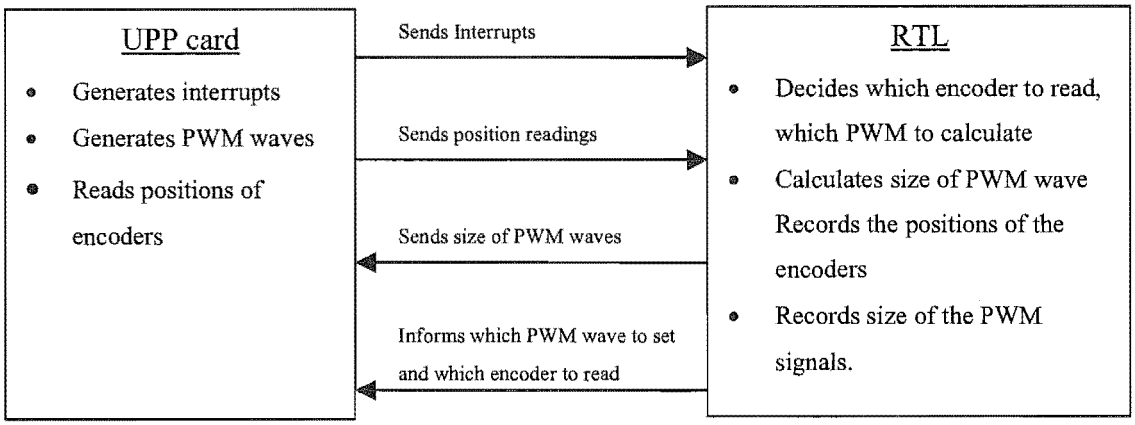

Figure 1.1 Communication between RTL and UPP card

In this thesis, the word "robot" will sometimes be used for "manipulator" and vice versa. Also, the word "ram" may sometimes be used to describe a hydraulic "actuator" or vice versa.

1.2 WHY USE REAL-TIME LINUX?

Traditionally, DSP (Digital Signal Processors) systems have been used for sophisticated controller implementation. DSP RAM is extremely limited due to the high cost of high speed RAM. Consequently, logging of variables during an extensive control run is certainly not possible. For example, most DSP systems have memory sizes of at most 32 to 64kB. The control and monitoring of a six-axis robot at 1 kHz sampling rate for 12 seconds with the requirement to monitor both position (float) and control signal (float) takes up 96kB. 486 machines running RTL can have a shared memory of up to 1MB with Pentiums going up to 4MB of shared memory.

The design cycle of a DSP based embedded hardware system also limits controller growth. First, the embedded system design company selects the chip from the existing

ones in the market. Second, it designs and manufactures a board and a set of software routines to go along with it. Third, the vendor promotes and markets the embedded system. By the time the DSP is available on the market, the chip that was originally used in the embedded system has been superseded as newer and faster chips are already in the market. Vendors of DSP embedded systems rarely make provision to interface older boards with newer ones and the I/O features might also be different from its predecessor.

RTL is easy to program, as there is no need to understand the architecture of the x86 chip. All that is needed is a good grasp of the C programming language, the software architecture of RTL, and communication between RTL and Linux. PCs always have the same I/O features no matter what generation of processor is used.

Furthermore, DSPs are capable of very high sampling rates. In the case of most mechanical control systems, a sampling rate of only 1kHz is considered very high speed, and RTL is more than capable of handling this.

1.3 UNIVERSAL PULSE PROCESSOR CARD

A general purpose UPP circuit can be used to control hydraulic positioning systems and DC motor drives. A printed circuit board developed for use in an IBM-PC contains two UPP chips [4]. It is capable of controlling eight positioning systems each of which provides two quadrature position signals plus an absolute zero position signal, and is driven by a PWM signal. When two zero signals are shared, then ten servo systems can be controlled.

The capability of the UPP card in controlling eight positioning systems would allow it to control the six-axis hydraulic manipulator. The board was originally developed to record speeds of 16 anemometers, but it was designed to be useful in other areas such as servo control, and precision temperature measurements.

Technician A.G. Cree wrote the driver files for the UPP card. These commands (functions) are collections of assembly codes for the UPP card. They make it easy for control programmers to set up encoders, PWM waves and interrupts.

1.3.1 Pulse Width Modulation

PWM was used to control the valves of the hydraulic rams in both the single axis hydraulic test rig and the manipulator.

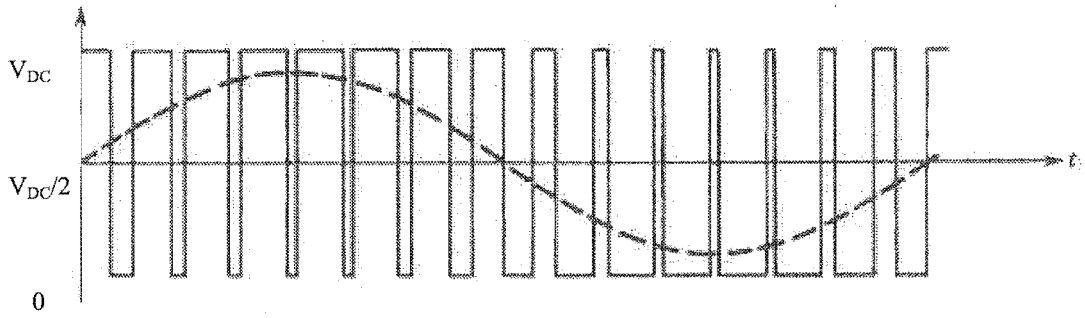

Figure 1.2 Pulse Width Modulation

Figure 1.2 shows how PWM is used to vary the size of a signal (broken line). A constant DC voltage amplifier is supplied a constant voltage V_{DC} as well as PWM signal from the UPP card. The output voltage from the amplifier controls the size of the valve opening in the actuator. The UPP card controls the magnitude of this voltage to the valves by changing the duty cycle of the PWM wave. The PWM wave has a fixed period, however, the wave might not be symmetric. The magnitude of the voltage is varied by the degree of symmetry of the wave. As to say:

$$V = \frac{Time_{on}}{Time_{off}}$$

Therefore, if ¼ of the voltage of V_{DC} is needed, $Time_{on}$ will occupy ¼ of the period and $Time_{off}$ occupies ¾ of the period. The voltage therefore can be varied to be anywhere in between 0-V_{DC} but there would still be discrete intervals in the signal. The smallest discrete interval is fixed by the smallest time difference the PWM wave can be switched by the UPP card (3.25 μs). The PWM wave was set at periods of 100 times the smallest

time difference, which therefore is 325 μs. Therefore the discrete interval of the resultant signal is 1% of V_{DC}.

The use of PWM eliminates the need for a D/A (Digital to Analogue) converter. Since the pulse processor is already used for interrupt generation, it might as well be used for generating PWM waves. The advantage of PWM waves over smooth DC current is that the oscillations in the PWM wave sets up a fluctuating flow through the hydraulic valves. This fluctuating flow makes it easier to dislodge any obstructions in the flow caused by impurities in the hydraulic fluid.

1.3.2 Quadrature Encoder

The UPP card recognises quadrature encoders. Quadrature encoders have only two changing bits. This allows changes in position and direction to be identified.

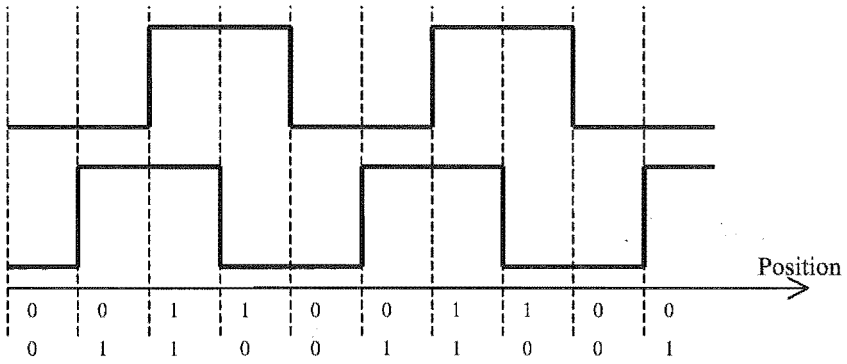

Figure 1.3 Signal from Quadrature Encoder

Figure 1.3 clearly shows how the quadrature encoder allows for direction recognition. At any position of the encoder, changing it by one single interval changes only one bit of the encoder. However, moving forward changes a different bit to moving backward. Therefore, the direction in which the encoder is moving is recognised by which of the two bits changes. When the pattern in Figure 1.3 is made to wrap around so that its end joins on to the start, it becomes an encoder that can wound around forever.

The single-axis hydraulic test rig utilises such an encoder. Since a quadrature encoder has only two changing bits, the absolute position cannot be found from it if there is no initial value to start from. Therefore, for any movement, it has to be given the value of the starting position so that subsequent positions can be measured from that. The manipulator uses an absolute grayscale encoder (see chapter 5 for details) so the positions are read as binary bits.

CHAPTER 2- REAL TIME LINUX

2.1 INTRODUCTION

The use of commodity priced PCs for control is attractive from a cost viewpoint, but has been problematic when window based operating systems are used for digital control. The essential problem is that "real-time" for most commodity operating systems means that the system responds to an operator input with only a small delay. The commonly available software disables the interrupts for periods of several milliseconds resulting in "soft" real-time operation. This is unsatisfactory for "hard" real-time systems where responses (or latency) of around 10 μs or less are required in order to carry out control functions in a timely manner. The real-time operating system Real Time Linux (RTL) developed at the New Mexico Institute of Technology was evaluated for instrumentation and machine control as it is designed to handle hard real-time processing of digital signals [17], [18].

The Posix-like Linux operating system coexists with a small real-time kernel to form RTL. The object is to use the sophisticated services and highly optimised average case behaviour of a standard time-shared Linux based computer system while still permitting real-time functions to operate in predictable low-latency environment. A standard operating system that offers a rich set of services was modified to act as a base kernel in a system where control is shared with a real-time kernel. The modification consists of

emulation code that intercepts commands to enable and disable interrupts. The emulation supports the synchronisation requirements of the base kernel while preventing the base kernel from delaying hardware interrupts. Interrupts that are handled by the base kernel are passed through to the emulation software after any needed real-time processing completes.

If the base kernel has requested that interrupts be disabled, the emulation software simply marks the interrupts as pending. When the base kernel requests that interrupts be enabled, the emulation software causes control to switch to the base kernel handler for the highest priority pending interrupt. Thus a very small modification of the base kernel allows it to execute without imposing a latency penalty on the real-time code. The resulting system can be viewed as a dual kernel operating system with the real-time kernel as the higher priority task.

The current implementation uses the x86 architecture version of the Linux POSIX-like operating system as the base kernel. In this system the real-time executives schedules and runs real-time tasks at a relatively high level of time precision and with low latency and overhead. The Linux kernel supports network services, GUI, development tools and a standard programming environment. One of the most compelling advantages of this method is that it requires very little modification to a reasonably designed base operating system. For Linux version upgrades are limited to the low level interrupt "wrappers" and the routines to disable and enable interrupts. As a result, advantage is taken of the rapid development of Linux and Linux tools. On the other hand, the system has been designed to allow real-time programmers to make nearly full use of the available hardware and processing power, without paying the price normally associated with more sophisticated operating systems.

An ISA bus card containing 2 Universal Pulse Processor (UPP) integrated circuits [4] has been developed to interface control plant to PCs. The UPP can be configured to accept quadrature position signals and to generate pulse width modulated (PWM) signals that set the actuator force/acceleration (or velocity). RTL sets up the digital controller sample

time by programming an interrupt to be generated by the UPP card. This interrupt generates hardware calls to the interrupt controller in RTL, and this in turn calls the interrupt function that reads the control system position feedback before calculating the value of the PWM pulse to be generated by the UPP card. On completion, control is returned to the program that was interrupted.

2.2 COMMUNICATION BETWEEN LINUX AND RTL

RTL is used to handle the real-time digital signal processing and Linux is used to communicate with RTL. In the case of machine control, Linux is used to generate the machine path requirements and RTL calculates the PWM pulse values required for the machine actuators. The actual path is measured in the RTL program and sent to Linux. There are two ways in which communication between RTL and Linux can be set up. One is using FIFOs (first in first out) buffers and the other is using shared memory. FIFOs are point-to-point queues of serial data analogous to Unix character devices or pipes. FIFOs have the following characteristics:

- FIFOs queue data, therefore no protocol is required to prevent data overwrites.
- Boundaries between successive data writes are not maintained. Applications must detect boundaries between data, particularly if the data is of varying size. For instance, a C++ program containing rtf_get(4, &data, sizeof(data)) tells the application in RTL to read the data variable of length size.
- FIFOs support blocking for synchronization. Processes need not poll for data.
- FIFOs are a point-to-point communication channel or pipe. FIFOs do not support the model of one writer and several readers.

The maximum number of FIFOs is declared in rt_fifo_new.c as:

#define RTF_NO 64

and these appear as devices /dev/rtf0-63 in the file system. This limit can be changed and the rt_fifo_new.o recompiled. The number of FIFOs is only limited by practical memory limits.

Shared memory is a portion of the PC's RAM that is shared by both RTL and Linux. This memory has to be allocated by a /etc/lilo.conf file in the Linux boot file. Shared memory is very similar to the common data block structure in FORTRAN and it has the following characteristics:

- Shared memory does not queue data written to it. Applications requiring handshaking must define a protocol to assure data is not overwritten – usually by using a semaphore flag.
- Because data is not queued, individual items in data structures that are several megabytes in size can be quickly updated.
- Shared memory has no point-to-point restriction. Shared memory can be written to or read from by any number of Linux or RTL processes.
- The number of independent shared memory blocks is only limited by the size of the physical memory.
- Blocking for synchronization is not directly supported. To determine if data is new, the data must contain a count that can be compared against previous reads.
- Mutual exclusion of Linux and RTL processes is not guaranteed. However, interrupted reads and writes can be detected.

Shared memory is a good choice for communication when control applications execute periodically on the expiration of an interval timer, and where data queuing is the exception rather than the rule.

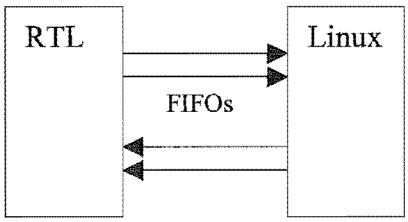

Figure 2.1 Communication between RTL and Linux using FIFOs.

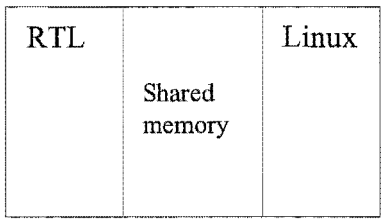

Figure 2.2 Communication between RTL and Linux using shared memory.

2.3 INTERRUPT LATENCY FOR FLOATING POINT PROCESSING

There are some limitations with RTL version 1.0 interrupt handling, especially when floating-point numbers are used for the digital controller calculations. When doing floating-point calculations in RTL on a 486-66 processor, the time taken to respond to an interrupt (the latency) and to read a UPP timer averages 12.5 μs with a maximum of 55 μs and a minimum of 7.5 μs. Therefore, interrupts must be at least 60 μs apart for such a processor doing floating point calculations. If lengthy calculations are required within the interrupt function, then the interrupt period must be increased so that interrupts over-runs do not occur. The latency times are reduced on Pentium processors, but a small amount of extra latency time dither is introduced by processor cache flushing.

The UPP card used for the interrupt latency test executes with a clock cycle of 3.25 μs (the exact cycle time depends on the number of functions active in the UPP), hence the interrupt occurs at multiples of 3.25 μs. The interrupt interval was set to 1000 clock cycles (3.25 ms), and each time the interrupt is generated, a function in RTL is called. This function starts by saving the FPU (Floating-Point Unit) registers and clearing the task switch flag in the CR0 register so that the processor ignores the exceptions that are generated when the FPU is used later. The current time from a UPP counter (that went to zero and generated the interrupt) is then read. The value read is proportional to the time that has lapsed since the interrupt occurred so this is proportional to the latency (note that the time taken to save the FPU registers is included in these latency measurements). At the end of the interrupt handling function, the floating-point registers are restored to the FPU and the flag in CR0 register switched back on for normal Linux operation. The interrupt function running under RTL has the following form:

```
int SaveFpuRegs[28];                                    // allocate memory for the FPU
registers
int Flags;
void IntFunc(void)                                      //Interrupt function
{
long linuxCR0;
  rtl_no_interrupts(Flags);                             // disable interrupts
  __asm__ __volatile__("movl %%cr0,%%eax":"=a"(linuxCR0)::"ax");   // inline assembly coding
  __asm__ __volatile__("clts");                         // clear task switch flag
  __asm__ __volatile__("fsave %0" : "=m" (SaveFpuRegs));   // save floating point regs
```

```
        IntTime=UPPReadData(card,18);                                   // get clock value
        if (Running)                                                    // when the test is running
        {
                rtf_put(1, &IntTime, sizeof(IntTime));      // puts clock value into FIFO 1 to send to
Linux
        }
        IntCount++;
        UPPIntStatClear(card,1);                                        // clear interrupt flag
 __asm__ __volatile__("frstor %0" : "=m" (SaveFpuRegs));                // restore floating point
regs
 __asm__ __volatile__("movl %%eax,%%cr0"::"a"(linuxCR0):"ax");          // restore cr0
  rtl_restore_interrupts(Flags);                                        // restore interrupt enable
}
```

Note the inline assembly code required to manipulate CR0. The UPP interrupt service has been programmed to save and restore the FPU each time there is an interrupt so that floating-point calculations done within the interrupt runtime do not overwrite the FPU calculations being carried out under Linux when the interrupt occurs.

Latency (μs)	Number of Interrupts
0-5	0
5-10	0
10-15	93148
15-20	6721
20-25	227
25-30	66
30-35	61
35-40	59
40-45	5
45-50	0
50-55	3
55-60	0
60-65	0
65-70	0
70-75	0
75-80	0
Total	100290
Maximum	54μs
Minimum	11.5μs
Average	13.97μs

Table 2.1 Floating point interrupt latency

The test program written to test interrupt latency is in Appendix B.

Table 2.1 is a typical interrupt latency check performed on an IBM 486-66 MHz. This particular run was done while the machine was networking with another Linux box. Notice that most of the latencies occur between 10-20 μs.

CHAPTER 3- SINGLE AXIS HYDRAULIC TEST RIG

3.1 INTRODUCTION

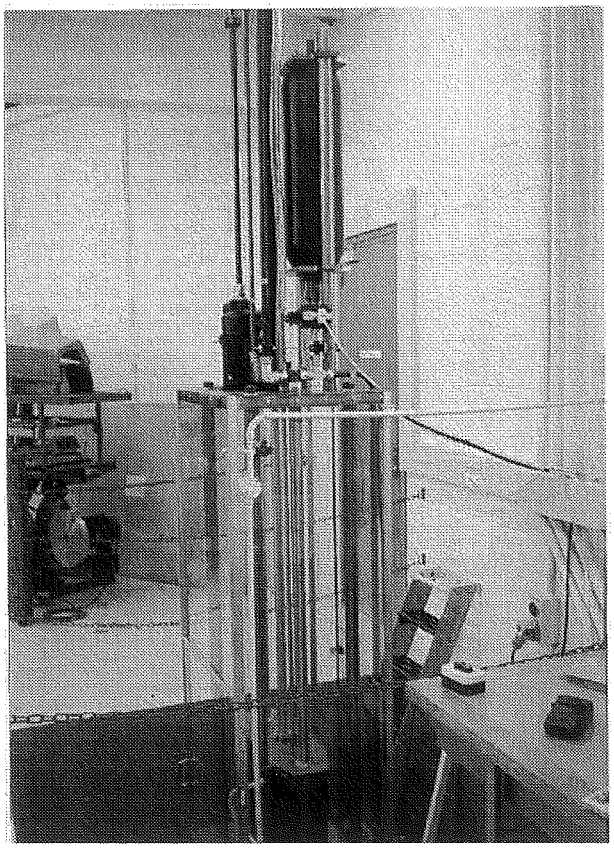

Figure 3.1 Single axis hydraulic test rig

Control of a single axis hydraulic test rig has been successfully implemented on a PC running under DOS [3]. However while a DOS based computer can handle interrupts in a timely manner, it lacks the ease of using a window based multitasking operating system for control system development and system response data display. It also lacks memory space for data storage. These missing features are available under RTL, and several different control algorithms (controller designs) for this test rig have been tested experimentally with the data being displayed at the same time. The results of experimental performance

evaluation have been obtained by observing the time responses for various test inputs (steps, ramps, and sine waves) and some step response results are presented here.

3.2 HARDWARE OF THE SINGLE-AXIS TEST RIG

3.2.1 Hydraulic Actuator

The hydraulic actuator is a single ended, double acting actuator controlled by a spool valve. The cylinder is mounted in a vertical position with an adjustable mass of 50 to 300kg suspended below which is connected to the rod of the actuator.

The cylinder has a stroke of 1.2 meters with the actuation speeds of up to 0.8 m/s possible. The internal diameter of the cylinder is 50.8mm (2") in which the piston and rod move while the rod has an outer diameter of 44.45mm (1.75"). Having the rod on only one side of the piston reduces the area of the piston that the fluid pressure can exert on the opposite side. Hence the force that can be exerted on the opposite side is less that that of the fully exposed side of the piston. With the exposed area at the top being 2.027×10^{-3} m^2 and the area on the rod side being 0.887×10^{-3} m^2, the area ratio is 2.285. Since the rod extends out from the bottom of the cylinder, the force available to move the piston upwards is 2.285 times less than the force available to move it down. For more information on this system, see [13].

3.2.2 Spool Valve

Control of the cylinder is via an electro-servo spool valve in which the PWM signal is from a UPP card within a PC.

3.2.3 Position Transducer

The position transducer for the test rig uses a incremental quadrature encoder. For details about quadrature encoders, refer to section 1.2.2. The resolution for the encoder is set to 0.1 mm. The encoder is encased in plastic tubing to prevent contamination by dust or moisture.

3.2.4 Load

The load is housed in a pexiglass cabinet and is constrained to move in the vertical direction by a pair of guide rods. The purpose of the rods is to prevent the load from twisting and damaging the test rig. The weights have specially designed oil-impregnated bearings where they touch the guide rods, as it is crucial to minimise friction in the system for accurate control of the rig. The weight can be adjusted in steps of 17kg and they are accessed through a door with a mechanical override, which prevents operation of the actuator when the door is open.

3.2.5 Hydraulic Pump

At the base of the test rig is a large electric powered hydraulic swash pump that generates a constant pressure that can be set between 14 and 20Mpa. The pump feeds hydraulic oil under pressure into an accumulator attached to the cylinder. This accumulator is a pressure reservoir that allows the actuator to operate beyond the flow capacity of the pump for short periods of time.

3.3 PERFORMANCE SINGLE AXIS TEST RIG

Several years ago, a single axis hydraulic test rig was designed for control research. This test rig is controlled using PWM with a period of 325 μs. The interfacing of the Moog electro-hydraulic servo valve system to the UPP system is described in [4]. The test rig has the following characteristic velocity response when operating at 13.3 MPa with a load of 300 kg:

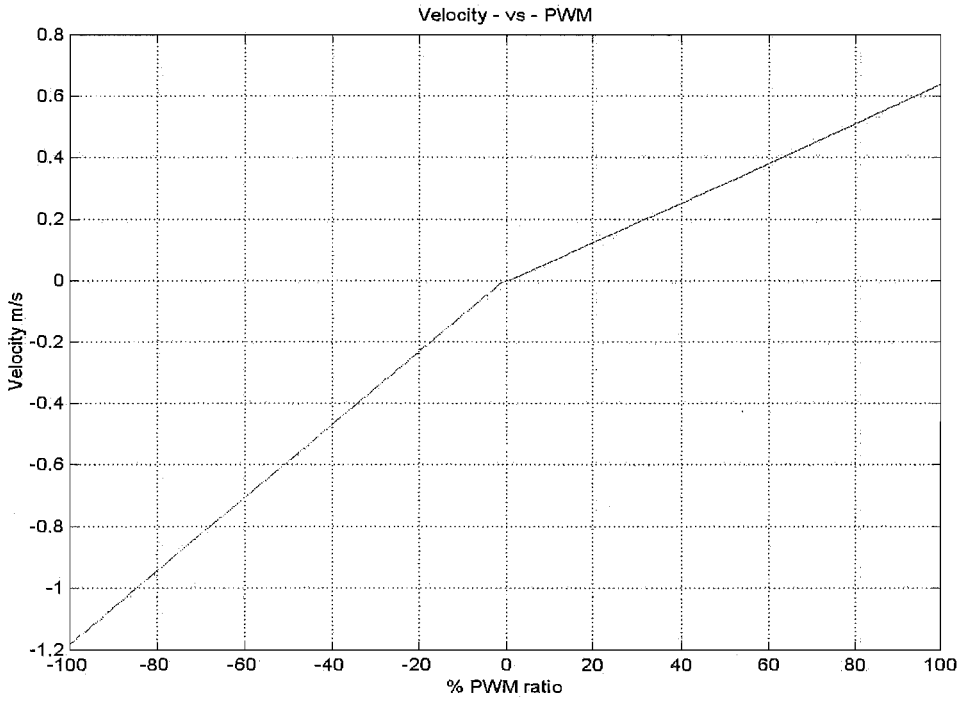

Figure 3.2 Response of the asymmetric hydraulic cylinder to the PWM signal from the UPP.

The velocity on the graph shows the steady state velocity as a function of the PWM ratio. The equations for the positive and negative velocities are respectively:

$$V^{+} = 0.64*PWM - 0.0056 \quad \textbf{(3.1)}$$

$$V^{-} = 1.19*PWM + 0.0056 \quad \textbf{(3.2)}$$

The constant offset of 0.0056 is a function of the electro-hydraulic servo valve spool overlap, and the ratio of the positive and negative gains is 0.64/1.19 = 0.538. The single ended hydraulic cylinder has a 50.8mm (2") bore and a 44.45mm (1.75") rod, the pressure needed to support the 300 kg load is 3.3 Mpa. Since the supply pressure is 13.3 Mpa, the effect of the load is quite significant. The different areas on each side of the hydraulic piston also have an effect, and the total effect is clearly shown in Figure. 3.2.

The control of the hydraulic rig is non-linear and the basic asymmetry inherent in the velocity measurements (Figure 3.2) has been included by the values of the PWM signal to be output. It is then possible to design a digital controller on a linear basis [5].

3.4 SOFTWARE DESIGN

The software implementation was in two parts. The controller was implemented in RTL where real-time processes are executed, while the User Interface (UI) was implemented in Linux for sending information to the controller from the user. To control the hydraulic test rig, data communication between RTL and Linux has so far been done via 4 FIFOs. As shown in Figure 3.3, 3 FIFOs send information from Linux to RTL and 1 sends data from RTL to Linux.

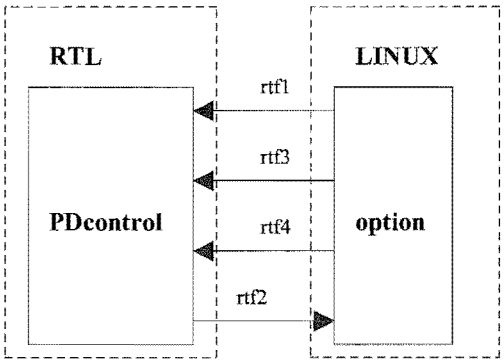

Figure 3.3 Communication between RTL and Linux for the hydraulic test rig

FIFO rtf1 sends commands from Linux to RTL to start and stop interrupts. FIFO rtf4 sends the coefficients to the RTL controller and FIFO rtf3 sends the position commands for the RTL controller to track. FIFO rtf2 sends the actual position and controller output measured by RTL back to Linux. Linux then plots and saves the results in a file. RTL is set up to immediately read any data written by Linux into rtf1. This allows immediate control of the interrupts. The desired tracking positions are queued up in rtf "3" before the interrupt is started in RTL. Once the interrupt has been started, RTL reads one tracking position from rtf "3" at the start of each interrupt.

3.4.1 Software Design in RTL

The controller is started and halted according to "msg.command" send to the controller through rtf 1 from User Interface (UI). There are four possible modes that the controller may function in, these are: START_TASK, SLOW, INITIALISED, STOP_TASK as shown in Figure 3.5. Figure 3.4 shows the different movements that the rig can perform.

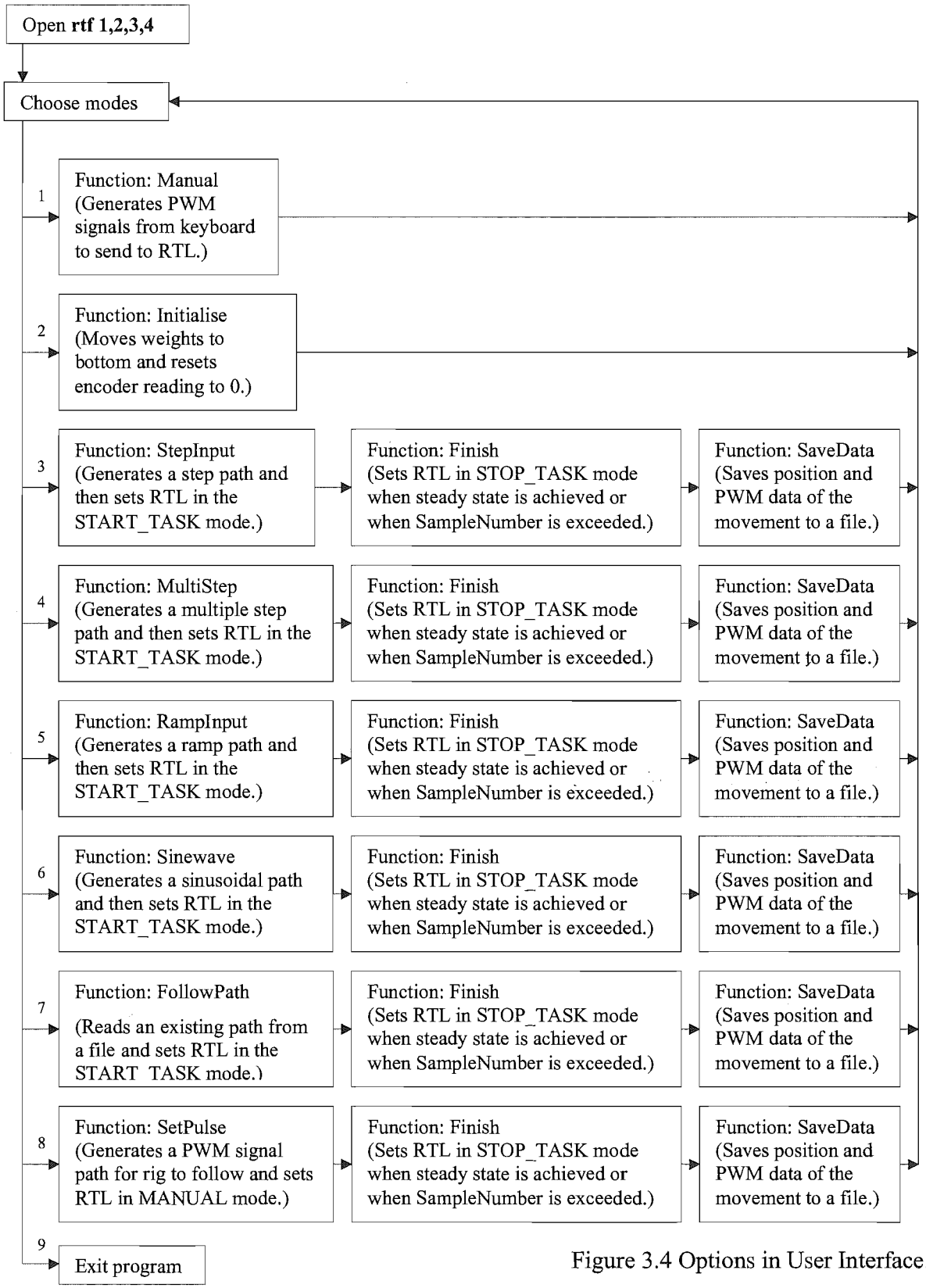

Figure 3.4 Options in User Interface.

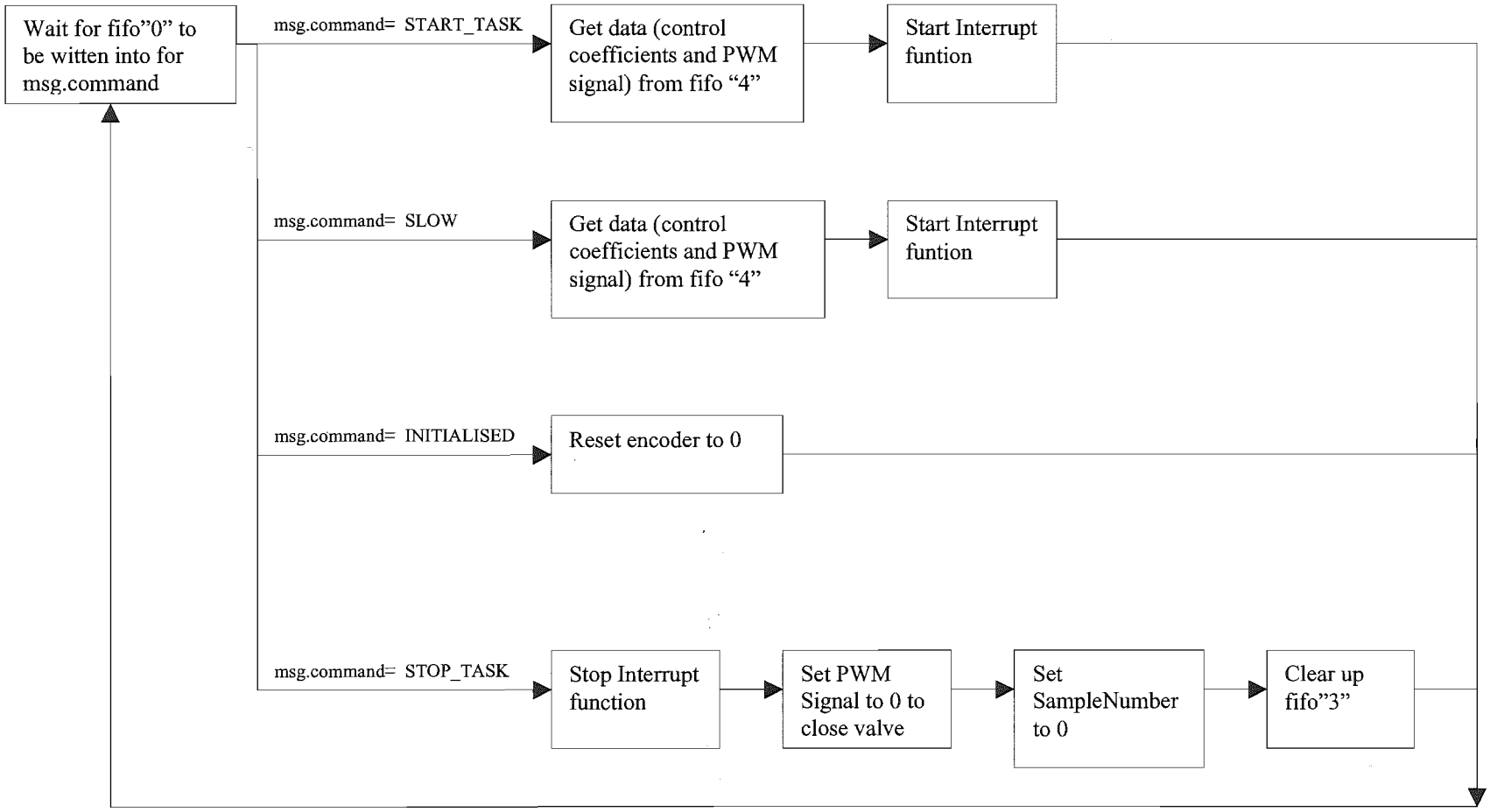

Figure 3.5 Controller modes

The START_TASK mode firstly reads "data" (control coefficient) from rtf4, it then starts the Interrupt function that is called periodically (i.e. every 1ms). The SLOW mode is basically the same as START_TASK except that it only performs part of the operation START_TASK performs in the Interrupt function (Figure 3.6). The SLOW mode is used for manual control of the rig where position error need not be calculated therefore it skips the calculation of the difference equation.

The INITIALISED mode is used to initialise the encoder to 0 when the rig is positioned at the bottom. The STOP_TASK stops the Interrupt function, it then sets the PWM signal to 0 to close the hydraulic valve and resets the "SampleNumber" to 0 and finally, it clears up rtf "3" by reading every integer out of it. The STOP_TASK mode is implemented when the movement of the rig has reached a steady state or when the "SampleNumber" exceeds a predefined number "NumSample".
The Interrupt function is only implemented in the START_TASK mode or the SLOW mode. At each interrupt for the START_TASK mode, an integer (desired position) is read from rtf 3. Rtf 3 has been queued up with thousands of points making a path, which the rig has to track. This desired position is then compared to the position of the encoder in the difference equation to calculate a PWM signal to send to the hydraulic valve. When in the SLOW mode, the interrupt function skips the PWM signal calculation and proceeds straight to sending a defined signal send from UI to the hydraulic valve.

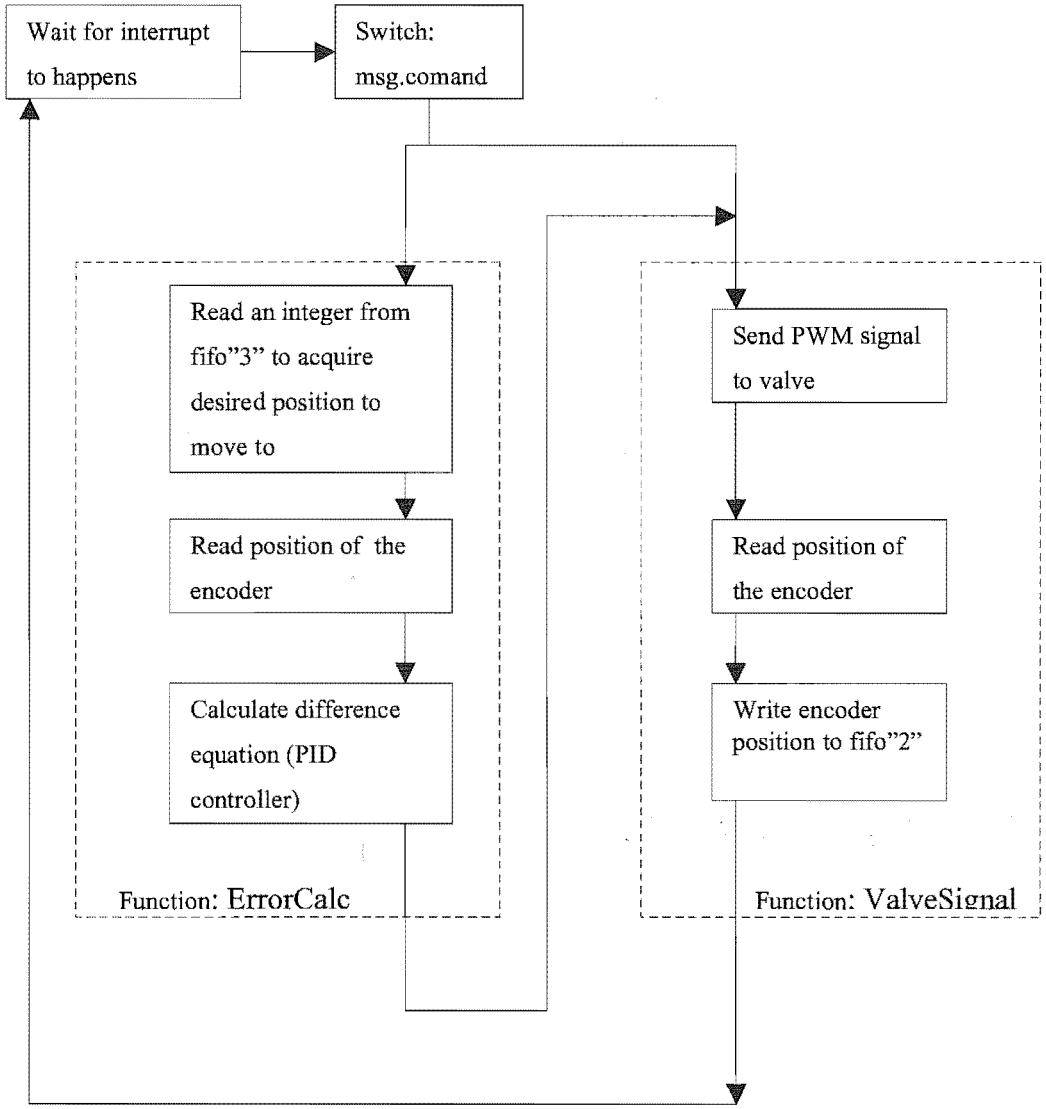

Figure 3.6 Interrupt Handler

3.4.2 Compiling the Software

All the software needed to control the test rig is specified in the "makefile" (See Appendix C). When "make" is typed at the prompt in the directory where the "makefile" is kept, all the files in the "makefile" is compiled using the GNU C compiler.

In the case of the test rig, "option.c", "function.c" are compiled into "option" and "function.o" respectively. The real-time software "PDcontrol.c" is compiled into "PDcontrol.o". Notice that compiling "PDcontrol.c" uses many real-time flags that "option.c" and "function.c" do not require. "PDcontrol.o" functions in RTL however "option" and "function.o" functions in Linux. Also, notice that "option.c" is compiled with "function.o" linked to it because "option" accesses functions from "function.o".

3.4.3 Header Files

"MyFifo.h" is a header file included in both "option.c" and "PDcontrol.c" as it initialises variables that both these programs use. This ensures that these variables are of identical size (float, int, char etc) since information is transferred between these two programs via fifos.

"getkey.h" is a header file that contains a function "getkey(int [3]) that allows "option" to read keys from the keyboard directly without having to hit the "Enter" key. This function uses the termios interface to change the terminal settings [10].

The header files PWM.H and Upp.h are both included in "PDcontrol.c" for accessing commands to control the UPP card.

3.5 USING THE SOFTWARE

In RTL, there are 5 modules that need to be inserted. One of the modules is "PDcontrol.o" and the rest are "PWM.o", "UPP.o", "rt_prio_sched.o" and "rt_prio_fifo.o". "PWM.o" and "UPP.o" are modules from which "PDcontrol.o" access the commands for the UPP card. "rt_prio_sched.o" and "rt_prio_fifo.o" are modules that

come with the RTL kernel. It is from these two modules that "PDcontrol.o" access commands to set up timers and fifos respectively.

To insert the modules, the user has to log into an "xterm" terminal as a super user. On the "xterm" prompt, type in "su" and hit "Enter". The "xterm" will then prompt for a password. When logged on as a super user, use the "insmod" command to insert the modules individually. At the prompt, type in the commands:

- insmod rt_prio_sched.o
- insmod rt_prio_fifo.o
- insmod UPP.o
- insmod PWM.o
- insmod PDcontrol.o

"UPP.o" has to be inserted before "PWM.o" as "PWM.o" has functions that refer to "UPP.o". To remove a module, simply type "rmmod" followed by the name of the module. A super user has authority to change almost everything on the Linux operating system; therefore the "xterm" where the super user role is assumed is only used for inserting and removing modules.

The next step in to launch the "option" program as a normal user in another "xterm" window. The user is then presented with a number of movement choices for the rig: "Initialise", "Manual", "Step", "Ramp", "Sine", "Multiple Step", "Exit". "Initialise" move the weights down to the very bottom and initialises the encoder. "Manual" allows the user to control the movement of the weights manually using the keyboard arrow keys. "Step" prompts the user to enter a start point and end point for the rig to perform a step function. "Ramp" and "Sine" is similar to "Step" because a start point and end point has to be entered to define the movement. However, a time element has to be additionally entered to define the length of the movement for both cases.

The "Multiple Step" choice allows the user to step the rig a number of times to and from a number of positions one after another. It first prompts the user for the number of steps

that need to be performed and then prompts the user for the positions and periods to stay at these positions. "Exit" merely exits to program.

3.6 CONTROLLER PERFORMANCE

Due to the ratio of positive gain to negative gain (section 3.3), the PWM signal for downward motion is scaled by a factor of 0.538. Hence:

$PWM^{+} = u$

$PWM^{-} = 0.538\ u$

Where u is the computed output signal from the controller.

3.6.1 Implementation of a PID controller on RTL

A PID controller was implemented under RTL. The structure of the controller is as follows:

$$u(k+1) = K_p e(k) + K_d[11e(k) - 18e(k-1) + 9e(k-2) - 2e(k-3)]/6\Delta t + \Sigma^k [K_i e(k)]\Delta t \quad \textbf{(3.3)}$$

where: k is the sampling number starting at 0 and incremented by one at each interrupt,

$u(k)$ is the output in PWM $\in$ {-100%,100%},

$e(k)$ is the error signal (desired position- current position),

Δt is the sampling interval or interrupt interval,

K_p is the proportional gain,

K_d is the derivative gain,

K_i is the integral gain.

and $\Sigma^k [K_i e(k)]\Delta t$ is limited to be less than 50 to prevent integrator windup.

The values $\{K_p, K_d, K_i\} = \{0.75, 0.002, 0.01\}$ were used with an interrupt of 1ms to obtain the step responses shown in Figures. 3.7 to 3.10. The effect of gravity can be clearly seen in the negative steady state offsets visible in the expanded scale graphs shown in Figures 3.8 and 3.10. Essentially the anti windup limit for the integrator part of the controller is set too low to remove the error. Also evident in Figure 3.10 is the effect of the oil compressibility. The mass oscillates against the "spring" formed by the

compressible column of oil in the cylinder as the mass velocity is being halted rapidly near the target position.

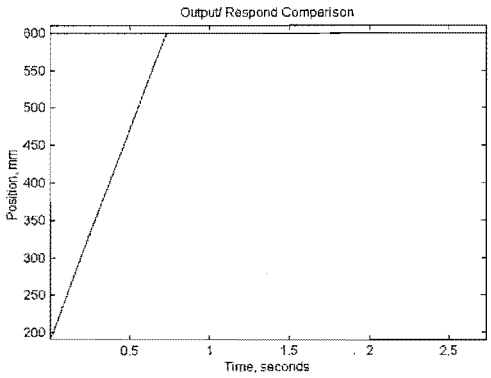

Figure 3.7 Step response from 200mm to 600mm.

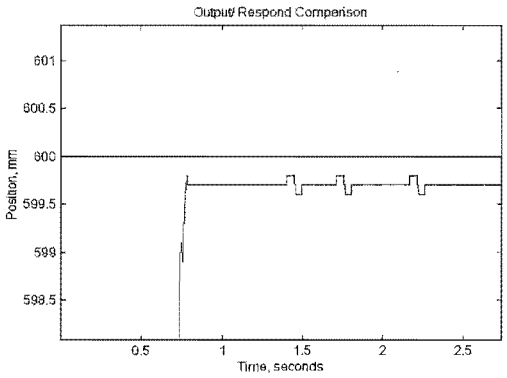

Figure 3.8 Close up of Figure 3.7.

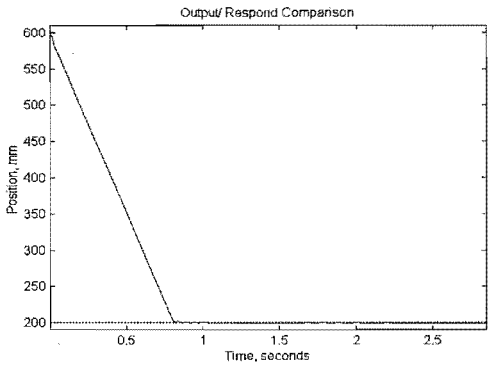

Figure 3.9 Step response from 600mm to 200mm.

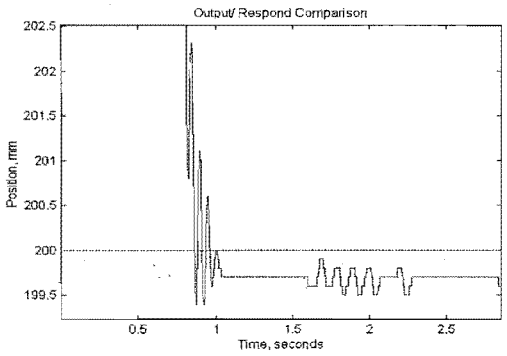

Figure 3.10 Close up of Figure 3.9.

3.6.2 Implementation of an Observer Based controller on RTL

An observer based state space state feedback digital controller was designed using MATLAB and was implemented under RTL. The structure of the resulting controller is:

$$u(k+1) = 95.1u(k) - 11.1u(k-1) + 0.01[8.42e(k) - 7.51e(k-1)] \quad \textbf{(3.3)}$$

Where *k* is the sampling number starting at 0 and incremented by one at each interrupt,
u(k) is the output in PWM (limited to a maximum of 100%),
e(k) is the error signal (desired position - current position).

The following graphs are the step responses for this controller.

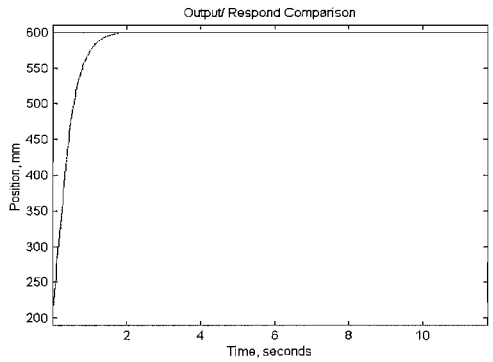

Figure 3.11 Step response from 200mm to 600mm.

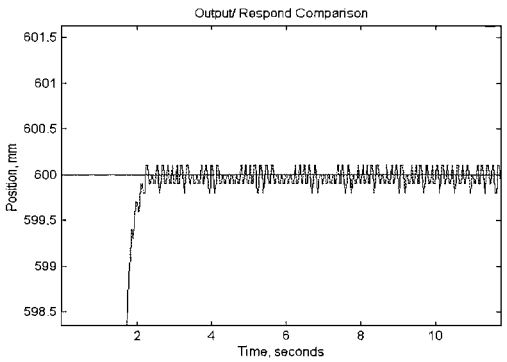

Figure 3.12 Close up of Figure 9.

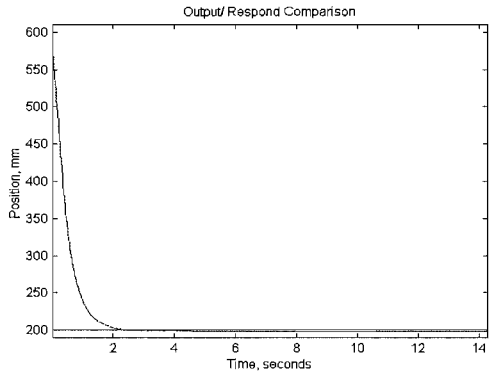

Figure 3.13 Step response from 600mm to 200mm.

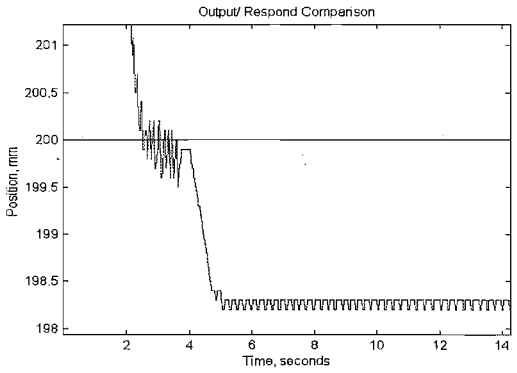

Figure 3.14 Close up of Figure 11.

The interrupt latency has been tested on a 486-66 MHz PC and found to be less than 70 μs. A major advantage, other than cost, of using a PC operating under the RTL system is that controllers are relatively easy to develop, implement and test thus allowing easy programming of real-time machine control applications. So far, RTL has been successfully used to control a single axis hydraulic test rig at sampling rates of up to 1000Hz.

CHAPTER 4- MANIPULATOR KINEMATICS

4.1 INTRODUCTION

The Unimate model 2000B is a serial link robot. It has six degrees of freedom, with five of these being revolute joints and one being a prismatic joint. The Denavit-Hartenberg notation was used as a systematic method of describing the kinematics of the robot [1]. It uses a minimum number of parameters to completely describe the kinematics relationship between links.

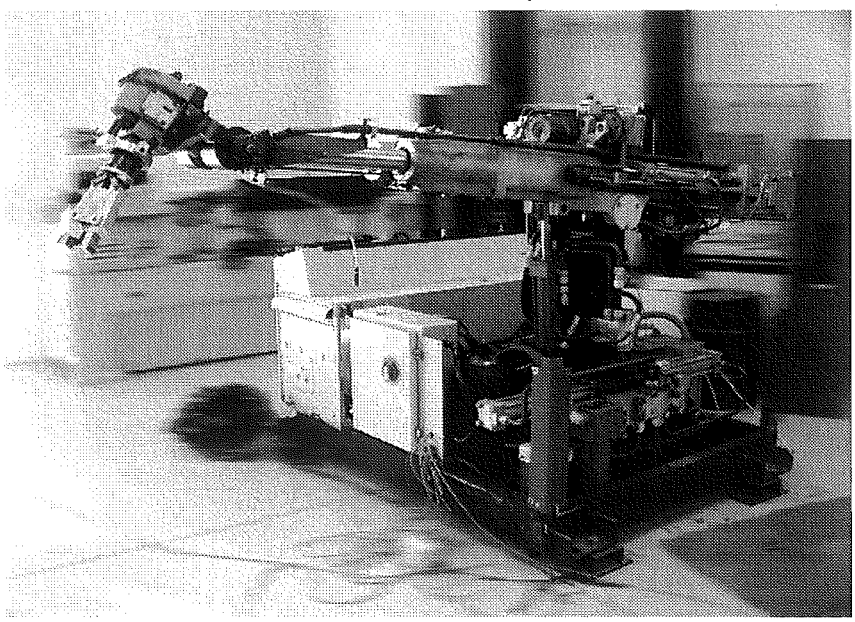

Figure 4.1 Unimate 2000B

4.2 DENAVIT-HARTENBERG (D-H) NOTATION

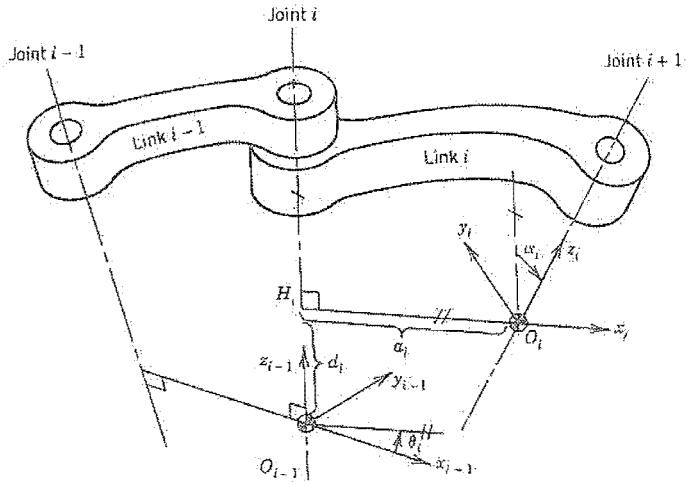

Figure 4.2 Denavit-Hartenberg notation

Figure 4.2 shows a pair of adjacent links, link i-1 and link i, plus their associated joints, joints i-1, i, i+1. Line H_iO_i in the Figure is the common normal to joint axes i and i+1. The relationship between the two links is described by the relative position and the orientation of the two coordinates frames attached to the two links. In the D-H notation, the origin of the i-th coordinate frame O_i is located at the intersection of joint axis i+1 and the common normal between joint axes i and i+1. Note that the frame of link i is at joint i+1 rather than at joint i. The x_i axis is directed along the extension line of the common normal, while the z_i axis is along the joint axis i+1. Finally, the y_i axis is chosen such that the resultant frame O_i-$x_iy_iz_i$ forms a right hand coordinate system.

The location of the frame O_i relative to frame O_{i-1} is completely determined by the following four parameters:

a_i the length of the common normal

d_i the distance between the origin O_{i-1} and the point H_i

α_i the angle between the joint axis i and the z_i axis in the right hand sense

θ_i the angle between the x_{i-1} axis and the common normal H_iO_i measured about the z_{i-1} axis in the right hand sense

The parameters a_i and α_i are constant parameters that are determined by the geometry of the link: a_i represents the link length and α_i is the twist angle between the two joint axes. One of the other two parameters d_i and θ_i varies as joint i moves.

The are two types of joint mechanisms used in manipulator arms: revolute joints in which the adjacent links rotate with respect to each other about the joint axis, and prismatic joints in which the adjacent links translates linearly to each other along the joint axis. For a revolute joint, parameter θ_i is the variable that represents the joint displacement, while parameter d_i is constant. For a prismatic joint, on the other hand, parameter d_i is the variable representing the joint displacement, while θ_i is constant.

$$X^{i-1} = A_i^{i-1} X^i \qquad \textbf{(4.01)}$$

where

X^i and X^{i-1} are 4x1 position vectors in O_i-$x_i y_i z_i$ and O_{i-1}-$x_{i-1} y_{i-1} z_{i-1}$.

$$A_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(4.02)}$$

The matrix A_i^{i-1} represents the position and orientation of the frame i relative to frame i-1. The first three 3x1 column vectors of A_i^{i-1} contain the direction cosines of the coordinate axes of frame i, while the last 3x1 column vector represents the position of the origin O_i. In other words, the former represents the rotation of frame X^i to frame X^{i-1} and the latter represents the translation of frame X^i to frame X^{i-1}. The derivation of formula A_i^{i-1} can be obtained from [1].

Therefore, equation (4.01) can be used to define the end point (end-effector) of a manipulator with n number of joints from the base frame when given the D-H parameters.

i.e. $$X^{i-n} = A_{i-n+1}^{i-n} A_{i-n+2}^{i-n+1} \ldots A_{i-1}^{i-2} A_i^{i-1} X^i \qquad \textbf{(4.03)}$$

4.3 X-Y-Z Fixed Angles

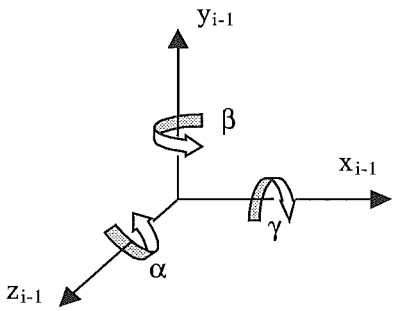

Figure 4.3 X-Y-Z fixed angles

As mentioned in the previous section, the first three 3x1 column vectors of A_i^{i-1} contain the direction cosines of the coordinate axes of frame i in frame i-1. These three 3x1 column vectors describe the orientation of the i-th reference frame to the i-1 reference frame. It can be more simply defined as angles γ, β, α and they are the angles around fixed axis x_{i-1}, y_{i-1}, z_{i-1} respectively [2]. This x-y-z fixed angles method was the one chosen for this thesis. However, there are in total, 12 ways of describing the three rotations about fixed axes including the one currently discussed.

Some of the other methods are rotation around e.g. (x_{i-1}, y_{i-1}, x_{i-1}), (z_{i-1}, y_{i-1}, x_{i-1}), (z_{i-1}, y_{i-1}, z_{i-1}). There are also 12 other ways of describing the three rotations using Euler angles where the axes of rotations are rotated with each rotation [2].

In the case of x-y-z fixed angles method, the rotations are done in the order of γ, β then α. The directions of x_{i-1}, y_{i-1} and z_{i-1} do not change as the rotations are performed.

Below is the matrix representing the rotation of frame X^i to frame X^{i-1}.

$$ {}^{i-1}_{i}R_{XYZ}(\gamma,\beta,\alpha) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha c\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma - c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix} \qquad \textbf{(4.04)} $$

where c and s are short for cos and sin. See [2] the derivation of (4.04). Equation (4.04) allows angles γ, β and α to be found. Below are the equations to solve all three angles[1].

$$ \begin{aligned} \beta &= A\tan 2(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}), \\ \alpha &= A\tan 2(r_{21}, r_{11}), \\ \gamma &= A\tan 2(r_{32}, r_{33}) \end{aligned} \qquad \textbf{(4.05)} $$

[1] Atan2(Y,X) is the four quadrant arctangent of the real parts of the elements of X and Y. $-\pi \le \text{Atan2}(Y,X) \le \pi$.

where:

$$ {}^{i-1}_{i}R_{XYZ}(\gamma,\beta,\alpha) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad \textbf{(4.06)} $$

Equation (4.05) describes only the rotation of frame X^i to frame X^{i-1} without translation. When translation is added, it is of the form

$$ {}^{i-1}_{i}A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(4.07)} $$

In the form above, it fully describes the movement from one reference frame to another with rotation described in the first 3x3 matrix and translation described in the last column. The 4th row is merely put in to make the matrix square so that it is commutable for multiplication when changing from one reference frame to another, i.e. the homogenous coordinate in (4.01) remains homogenous.

4.4 KINEMATICS OF UNIMATE MODEL 2000B

The Unimate Model 2000B consist of an arm segment and a wrist segment (see Figure 4.4). The arm segment is made out of the Rotary, Down-Up and In-Out joints with the last being a prismatic joint. The wrist segment is made out of the Bend, Yaw and Swivel joints.

The arm segment is used to position the end-effector of the manipulator to a certain point (x, y, z) in the workspace. The wrist segment is then used to rotate the end-effector into a certain orientation (γ, β, α).

Six degrees of freedom is the minimum requirement for a manipulator to be able to move its end-effector to any point and orientation within the manipulator workspace.

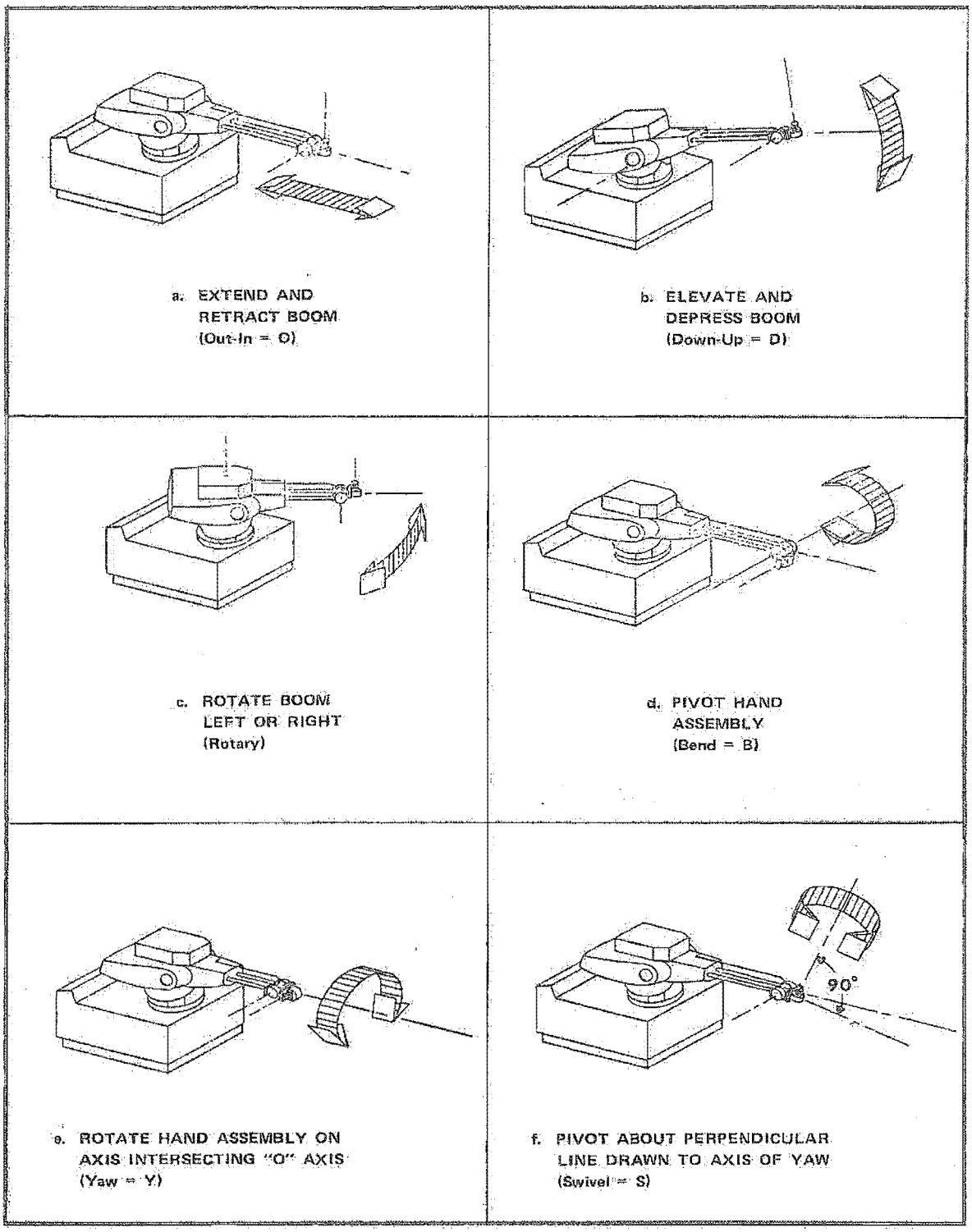

Figure 4.4 X-Y-Z fixed angles

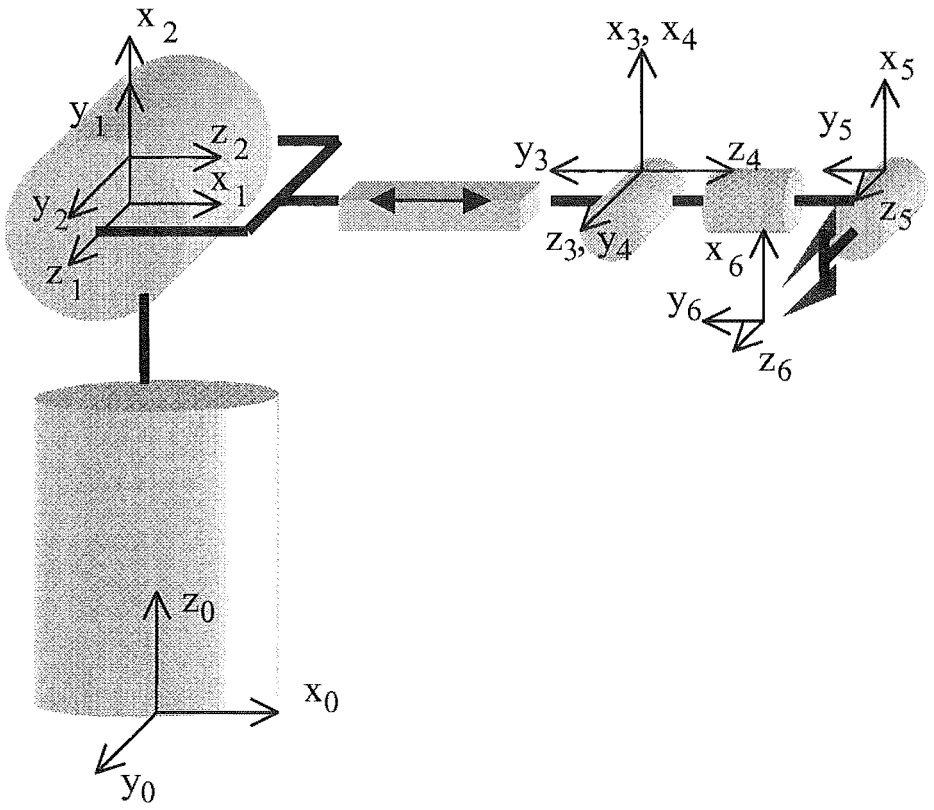

Figure 4.5 Reference frame on each joint

The above Figure depicts the reference frame at each joint. All joints are positioned at their origins; therefore θ_1, θ_2, θ_4, θ_5 and θ_6 are zero when the manipulator is in the position as shown in the Figure. The origins of frame 3&4 coincide.

The table below shows the four variables needed for the D-H convention.

Link number	α_i (degrees)	a_i (mm)	d_i (mm)	θ_i (degrees)
1	+90	0	1066.8	θ_1
2	+90	120.65	0	θ_2
3	-90	0	d_3	0
4	90	0	0	θ_4
5	-90	0	200 (L_b)	θ_5
6	0	0	440 (L_y)	θ_6

Table 4.1 Link Parameter for Unimate Model 2000B

$$A_1^0 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_2^1 = \begin{bmatrix} -\sin\theta_2 & 0 & \cos\theta_i & -a_2\sin\theta_2 \\ \cos\theta_2 & 0 & \sin\theta_2 & a_2\cos\theta_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_4^3 = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 & 0 \\ \sin\theta_4 & 0 & -\cos\theta_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5^4 = \begin{bmatrix} \cos\theta_5 & 0 & -\sin\theta_5 & 0 \\ \sin\theta_5 & 0 & \cos\theta_5 & 0 \\ 0 & -1 & 0 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_6^5 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The derivation of A_2^1 was done through two steps of D-H instead of one. Unlike the other transformations, A_2^1 from A_1^0 requires first a rotation about z_1 then about x_1' (the rotated x_1).

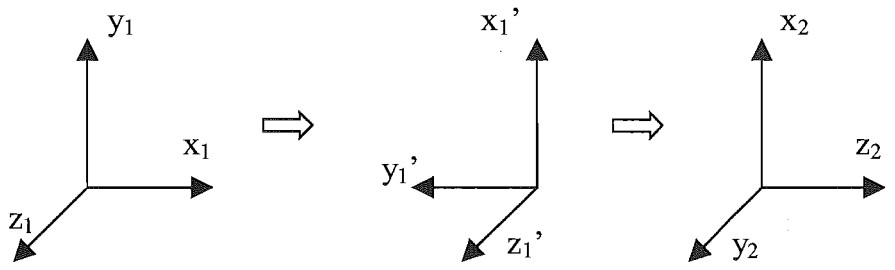

Figure 4.6 Transformation from frame 1 to frame 2.

Link number	α_i (degrees)	a_i (mm)	d_i (mm)	θ_i (degrees)
1'	0	0	0	90
2	90	0	0	θ_2

Table 4.2 Link Parameter for A_2^1

Hence:

$$A_{1'}^{0} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_{2}^{1'} = \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 & 0 \\ \sin\theta_2 & 0 & -\cos\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Since:

$$A_{2}^{1} = A_{1'}^{0} A_{2}^{1'} \qquad \textbf{(4.08)}$$

Therefore:

$$A_{2}^{1} = \begin{bmatrix} -\sin\theta_2 & 0 & \cos\theta_i & -a_2\sin\theta_2 \\ \cos\theta_2 & 0 & \sin\theta_2 & a_2\cos\theta_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.5 INVERSE KINEMATICS

The direct kinematics equation (4.03), establishes the functional relationship between the joint variables and the end-effector position and orientation. The inverse kinematics consist of the determination of the joint variables corresponding to a given end-effector position and orientation. The solution of this problem is fundamental to transform the motion specification, assigned to the end-effector in the operation space, into the corresponding joint space motions that allow execution of the desired motion.

The Unimate 2000B has six degrees of freedom therefore it has a finite number of solutions for inverse kinematics. A manipulator arm must have at least six degrees of freedom in order to locate its end-effector at an arbitrary position with an arbitrary orientation in its workspace. Manipulator arms with less than 6 degrees of freedom are not able to perform such arbitrary positioning. On the other hand, if a manipulator arm has more than 6 degrees of freedom, there exist as infinite number of solutions to the inverse kinematics equation.

4.5.1 Solving the Inverse Kinematics equation

Equation (4.03) states that

$$X^{i-n} = A_{i-n+1}^{i-n} A_{i-n+2}^{i-n+1} \ldots A_{i-1}^{i-2} A_i^{i-1} X^i$$

Let

$$X^{i-n} = TX^i \qquad \textbf{(4.09)}$$

Where

$$T = A_{i-n+1}^{i-n} A_{i-n+2}^{i-n+1} \ldots A_{i-1}^{i-2} A_i^{i-1} \qquad \textbf{(4.10)}$$

T is directly derived from equation (4.09)

For a 6 degree of freedom manipulator arm

$$T_6^0 = A_1^0 A_2^1 A_3^2 A_4^3 A_5^4 A_6^5 \qquad \textbf{(4.11)}$$

Therefore, closed form solutions exist for an arbitrary end-effector location T. The above equation can be written in many different forms. For example, post multiplying both sides by the inverse of A_6^5 yields

$$T_5^0 = T_6^0 (A_6^5)^{-1} = A_1^0 A_2^1 A_3^2 A_4^3 A_5^4 \qquad \textbf{(4.12)}$$

Further pre-multiplying both sides by $(A_1^0)^{-1}$,

$$T_5^1 = (A_1^0)^{-1} T (A_6^5)^{-1} = A_2^1 A_3^2 A_4^3 A_5^4 \qquad \textbf{(4.13)}$$

The left and side of equation 4.12 is only the function of θ_6, while the right-hand side involves all the other joint displacements. Similarly, equation 4.13 has θ_1 and θ_6 on the left-hand side and the remaining joint displacements on the right-hand side. Since T is known (from the orientation and position of the end-effector), θ_6 could be worked out from equation 4.12 and hence θ_1 from equation 4.13.

The symbolic math toolbox in the mathematics program Matlab was used to help solve the inverse kinematics of the Unimate 2000B. The program was used to expand the matrices so that 6 independent equations could be found to solve for θ_1, θ_2, d_3, θ_4, θ_5, θ_6. "function inverse" in Appendix E shows the program in Matlab.

The further mathematical manipulation of the equations found are presented in Appendix A.

The equations to solve for the 6 joint variables are:

$$\theta_1 = \tan^{-1}(r_{23}-p_y/L_y \, , \, r_{13}-p_x/L_y) \quad \textbf{(4.14)}$$

$$\theta_6 = \tan^{-1}(c\theta_1 r_{22} - s\theta_1 r_{12} \, , \, s\theta_1 r_{11} - c\theta_1 r_{21}) \quad \textbf{(4.15)}$$

$$\theta_5 = \text{Atan2}(c\theta_6 s\theta_1 r_{11} - c\theta_6 c\theta_1 r_{21} - s\theta_6 s\theta_1 r_{12} - s\theta_6 c\theta_1 r_{22} \, , \, s\theta_1 r_{13} - c\theta_1 r_{23}) \quad \textbf{(4.16)}$$

Numerically solve for θ_2:

$$L_b r_{31} s\theta_6 + L_b r_{32} c\theta_6 - L_y r_{33} + p_z - d_1 =$$
$$t\theta_2\left((L_b r_{11} s\theta_6 + L_b r_{12} c\theta_6 - L_y r_{13} + p_x)/c\theta_1 \right) + t\theta_2 a_2 s\theta_2 + a_2 c\theta_2 \quad \textbf{(4.17)}$$

$$\theta_4 = \text{Atan2}\left((-r_{31} s\theta_6 - r_{32} c\theta_6)/ (c\theta_5 c\theta_6 r_{31} - c\theta_5 s\theta_6 r_{32} - r_{33} s\theta_5) \right) - \theta_2 \quad \textbf{(4.18)}$$

$$d_3 = (L_b r_{31} s\theta_6 + L_b r_{32} c\theta_6 - r_{33} L_y + p_z - d_1 - a_2 c\theta_2)/ s\theta_2 \quad \textbf{(4.19)}$$

4.5.2 Multiple Solutions to Inverse Kinematics

The six degrees of freedom manipulator has a fixed number of joint solutions to any given end-effector position and orientation. Therefore, it is solvable unlike

Equation 4.14 shows that θ_1 has 2 solutions due to the property of $\tan^{-1}$ from regions of 0 to 2π. Due to the fact that the Rotation joint is limited from 110° to -110°, it was decided for simplicity to limit the θ_1 to 90° ($\pi/2$) to -90°($-\pi/2$) so that the extra solution do not propagate into 4 solutions through equation 4.15. The atan function in the C programming language is also limited in the range of 90° ($\pi/2$) to -90°($-\pi/2$).

θ_6 also have 2 solutions as can be deduced from equation 4.15. If the extra solution from θ_1 in the quadrant of $\pi/2$ to $3\pi/2$ was retained, equation 4.15 would resolved in 4 solutions for θ_6. Once again, for simplicity sake, θ_6 were limited to 90° ($\pi/2$) to -90°($-\pi/2$) even though the Swivel joint can rotate from π to $-\pi$. This allows for simple computation when solving the inverse kinematics.

There is only one solution for θ_5 because Atan2 (equation 4.16) was used resulting from knowing both $\sin\theta_5$ and $\cos\theta_5$. There is also one solution for θ_2 because the Up-Down joint only rotates from 30° to -27°. A numerical approach was used to solve for θ_2 as it cannot be isolated in equation 4.17 using any trigonometric relationship.
Lastly, d_3 (equation 4.18) has only one solution because no inverse trigonometric function is required to solve for it.

However, the flexibility of the manipulator is limited by fixing the solution of θ_6 between π and $-\pi$. If θ_6 were not limited, there would be two different orientation of the manipulator for any end-effector position in the workspace.

4.6 MANIPULATOR HARDWARE

4.6.1 Manipulator Hydraulics

All six axis of the Unimate 2000B manipulator are powered by hydraulic actuators, which are under control of servo valves. Hydraulic actuators for In-Out and Down-Up motions are connected directly to their respective loads. For rotation, a rack and pinion converts linear travel of the hydraulic rams to rotary motion. The motion of Bend, Yaw and Swivel are transmitted by systems of chains, gears and shaft since the wrist is moved by the arm and the wrist has to be kept at a minimum weight. A ball nut and spline shaft arrangement was used to transmit motion to the wrist axes.

The hydraulic power is generated using a type of vane pump that is powered by a 10 hp (7.5 kW) electric motor. The accumulator has been charged with dry nitrogen to a pressure of 525 psi (3.5 Mpa). This extra pressure ensures that the system pressure will be maintained when flow demand exceeds pump output flow capacity. Each of the 6 servo valves has four way infinite position valves, which are controlled by electric signals. The polarity of the signal controls the direction of the movement, while the magnitude of the signal controls the size of the valve opening. A larger valve opening means a higher flow rate, therefore a faster joint movement. A servo valve directs the

hydraulic fluid to one side of a hydraulic actuator and opens the opposite side for the return flow. When moving in the opposite direction, the connections are reversed.
The Rotary and In-Out actuators are slightly different. The rotary actuator consists of two actuators at opposite ends of the rack. Hydraulic fluid is admitted from the piston side of each actuator just as though it was a single piston in a single actuator. The In-Out actuator has fluid at system pressure at the rod side of the piston at al times. The servo valve controls fluid flow to or from the piston side only of the Out-In actuator. The Swivel motion utilises a hydraulic motor, not a linear actuator. See [6] for more information on the manipulator hardware.

4.6.2 Manipulator Safety Measures

A physical barrier in the form of a chain has been installed across half of the hydraulics lab to prevent people from going into the manipulator workspace. There are also infrared sensors in front and behind the manipulator. When the sensor beam is broken, a relay turns off the hydraulic pump.

A start and stop relay is used to start and stop the hydraulic pump. However, this is not enough to stop the manipulator as hydraulic pressure is still accumulated in the accumulator. Another remote emergency stop button was also installed in the circuit between the UPP card and manipulator. When this circuit board emergency stop button is activated, all PWM signals to the manipulator are stopped thus closing all actuator valves. This effectively stops the movement of the manipulator even when the accumulator is still fully charged.

Therefore, when required to stop the manipulator instantly, the circuit board emergency stop button should be pressed first, then the stop button for the hydraulic pump relay.

4.6.3 The Circuit Board

The manipulator originally was controlled by discrete electronics. It was decided in an earlier project to use none of the onboard electronics except for the solenoid, encoder and

some wiring. The function of the circuit board is to amplify the PWM signals for the valves, produce the correct voltage for the encoder scan output, and provide a suitable medium for the encoder signals to the computer.

The major inputs and outputs for the circuit board are listed on the following page.

- 32 bit link to the computer
- 15 input bits, 13 bits for the encoder, leaving 2 for other inputs
- 6 output bits to scan the encoders
- 6 positive amplified PWM signals for each of the valve
- 6 negative amplified PWM signals for each of the valve
- Common for PWM
- 24V power output (positive and earth)
- 15V output
- 5V output

Originally, an external 24V power source was going to be applied to the control circuit board to power the PWM amplifiers. The control circuit board steps down the voltage to 15V and 5V to power the encoder lamps and logic respectively. The robot also has a built in 28V power source as well as existing wiring to power the 15V encoder lamp and the 5V logic signals for the encoder output. Fortunately, the external control circuit board is able to run on the 28V power source therefore eliminating the need for an external power source for the control circuit board.

4.6.4 The Absolute Grayscale Encoders

The encoders return an absolute value that defines where the joints are orientated or positioned. The encoders consist of an encoder disk, a light source, a photocell cathode and related circuitry for each of the available 15 bits. The disk has 15 concentric rings of grayscale-patterned slots across its surface. The 15 photocells are able to sense light transmitted through the 15 rings, and send 15 bits for the light pattern sensed. Of the 15 available signals, all encoders on the manipulator use only 13.

There are 6 encoders; these encoders are multiplexed so that only one encoder is read at one time. Therefore, only 20 wires are needed instead of 84 wires. Each encoder has a 13 bits output that joins up to the same wires from the other encoders. Only one encoder is active driving the sensed bits down the wires at any one time. The active encoder is selected one at a time so that 6 select and read operations are required to determine the manipulator position.

CHAPTER 5- MANIPULATOR CONTROL

5.1 SOFTWARE DESIGN

The software implementation was in two parts as was done for the single axis hydraulic test rig. The controller was implemented in RTL where real-time processes are executed where as the User Interface (UI) was implemented in Linux for sending information to the controller from the user. The program in RTL was given the name "RT" and the UI in Linux was given the name "option".

The RTL part for the manipulator varied from the RTL part in the single axis hydraulic test rig in the way the Interrupt function is controlled. The Interrupt function in the single axis hydraulic test rig was started and halted from the UI where as the Interrupt function of the manipulator is started as soon as it is inserted and only halted when it is removed. There is also a separate "visual" program where information on the manipulator joints (position and PWM signal) is updated every 300 interrupts or 50 complete position readings. This "visual" program is automatically started when "option" is launched.

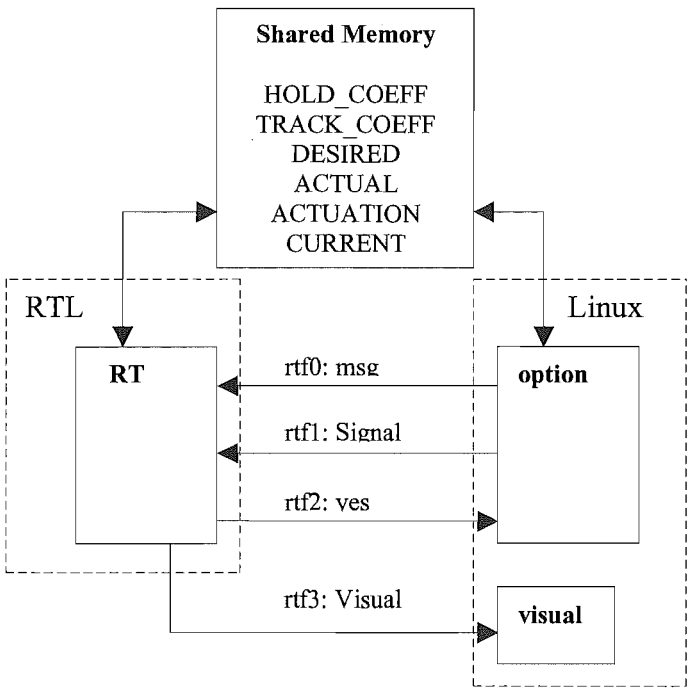

Figure 5.1 Communication between RT, option and visual.

Figure 5.1 shows the whole software set up on the PC. There are altogether 4 fifos used in the set up. rtf0 is used to transfer to RTL the "msg" structure that determines the mode required for RTL to function in. rtf1 transfers the "Signal" structure to set the PWM signals when it is in the manual control mode. rtf2 is used to send an integer variable "yes" to inform the UI when a movement has been completed. rtf3 sends the "Visual" structure to the "visual" program in Linux that inform the user about the current states of the joints (PWM, position).

The shared memory between RTL and Linux starts at the 38^{th} Mb of RAM out of the 39Mb in the PC. This 1Mb of RAM has been allocated for information storage accessible only by programs "RT" and "option". "HOLD_COEFF" and "TRACK_COEFF" are both structures of controller coefficient. "DESIRED" is a structure of paths for the joints to track during a movement, while "ACTUAL" is a structure of paths and PWM signals that the joints have carried out during a movement. "ACTUATION" is a structure of PWM signals to be performed by the actuators in a movement. Finally, "CURRENT" is a structure of current joint positions that RTL writes to when "SampleNumber" is at 1. Fig. 5.2 shows the different movements the manipulator can perform.

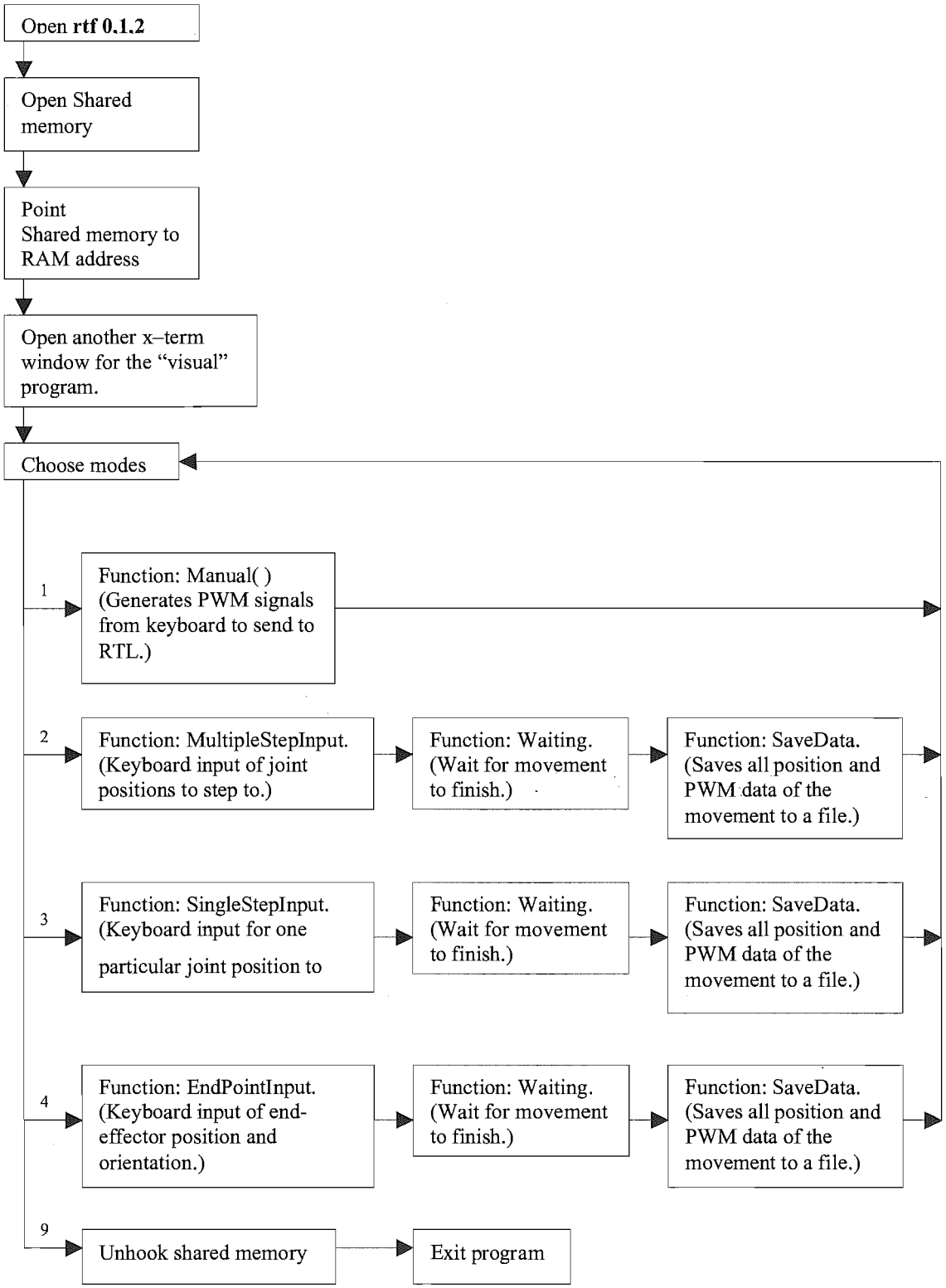

Figure 5.2 Choices of different movements in "option"

5.1.1 Software Design in RTL

5.1.1.1 Fifo Handler

There are four modes in which the RTL program "RT.o" may function. The program RT.o can only be in one of these mode at any one time. These modes are decided by variable "msg.command" sent through rtf 0 from the UI. They are: "hold", "follow_track", "manual_actuate" and "tracking_actuate". The default mode is "hold" because it starts at this mode and the modes "follow_track" and "tracking_actuate" revert back to "hold" once "SampleNumber" exceeds a predefined "NumSample". "SampleNumber" is incremented every six interrupts because each interrupt services one joint and there are six joints on the manipulator.

The function "my_handler" is executed only once every time something is written into rtf0 (see Figure 5.3). Different operations are done depending on the mode rtf 0 receives from the UI. When the "hold" mode is executed, RTL firstly copies the "Hold_Coefficient" (coefficients for the controller) from shared memory. It then sets "holding" to the hold position for all six joints at the positions it was previously at, one "SampleNumber" before. Finally it resets the "SampleNumber" back to 0.

When the "follow_track" mode is implemented, RTL copies "Desired_tracking" (joint positions for the joints to follow) from shared memory. It then copy "Track_coefficient" (coefficients for the controller) from shared memory. Lastly, it resets "SampleNumber" back to 0 for the start of the tracking movement. There are two different coefficients ("Hold_Coefficient" and "Track_coefficient") because it gives the user flexibility of having two different controllers for when it's stationary and when it's moving.

The "manual_actuate" mode is used for manual control of the manipulator and all it does is read the PWM signal for all joints from rtf 1.

The last possible mode is the "tracking_actuate" mode where the manipulator tracks PWM signals instead of joint positions. It is meant for the use in system identification

where the system response (position, velocity, acceleration) is compared to signal input (PWM) so that a model can be derived from the system. When the "tracking_actuate" mode is sent through rtf 0, the "my_handler" function copies "Actuate_Tracking" (PWM signal for the actuators at each joint) from shared memory and then resets "SampleNumber" to 0.

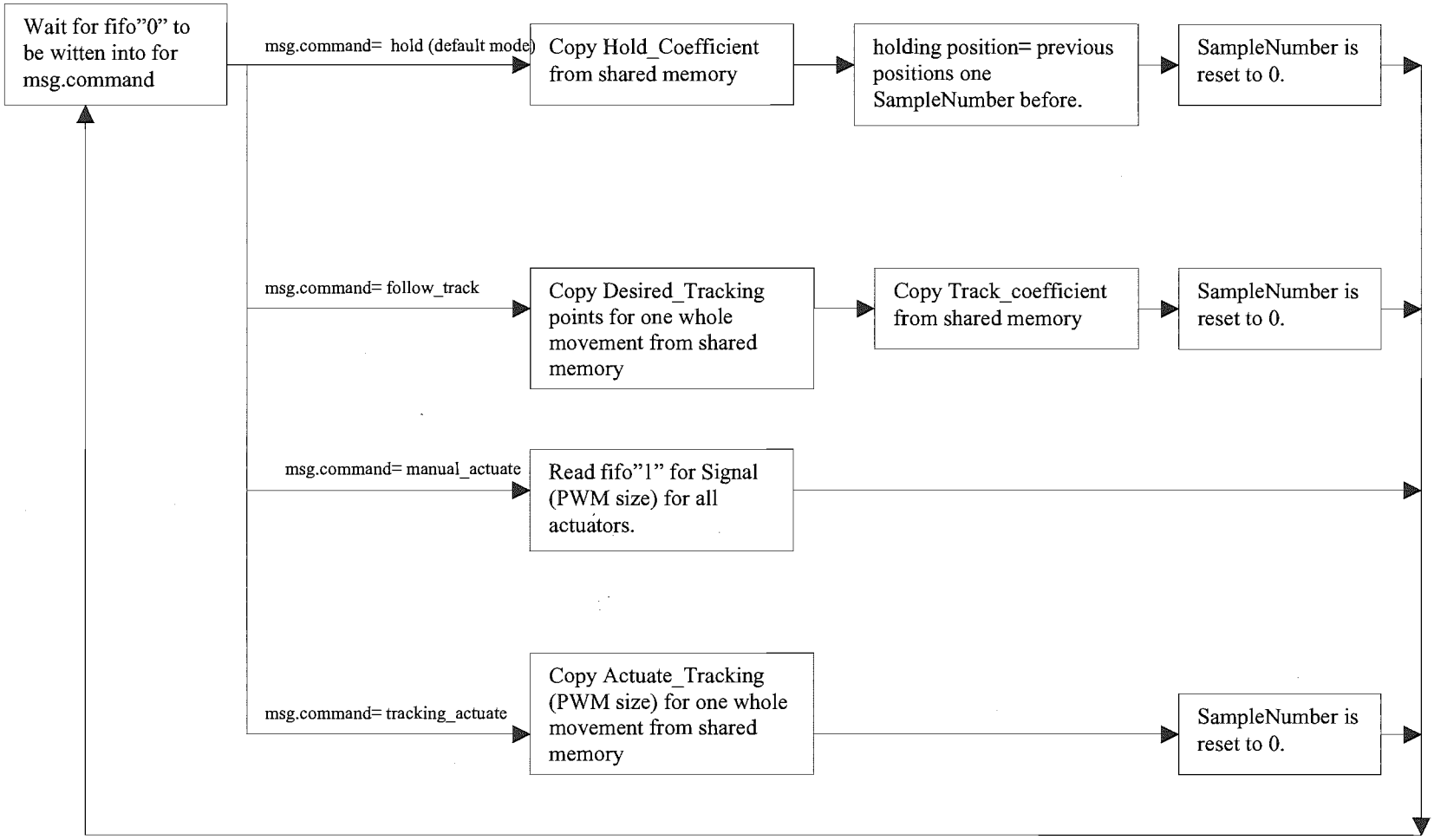

Figure 5.3 Different modes for "RT" operation.

5.1.1.2 Interrupt Function

The Interrupt function is always running and the variable "Actuator_No" is incremented near the end of each interrupt. Therefore, each interrupt services one joint and the next joint is serviced when the next interrupt happens.

At the start of an interrupt, the encoder is read and this reading is then converted from grayscale to normal binary. It then checks whether "Actuator_No" is 4 or 5 (Yaw or Swivel) because the positions of Yaw depends partly on the position of Bend and the position of Swivel depends partly on the position of Yaw. After the joint dependencies have been compensated for, the Interrupt function splits into four possible paths depending on which of the four modes it is running (see Fig. 5.4).

When in the "hold" mode, the "ErrorCalc" function is called to calculate the PWM signal and then this calculated signal is send to the "ValveSignal" function to set the hydraulic ram valve. When in the "follow_track" mode, it goes through the same procedure as "hold" except that it copies the joint positions of the movement to shared memory at the end of the movement (NumSample ≥ SampleNumber-1).

For the "manual_actuate" mode, only the "ValveSignal" function is called to set the hydraulic ram valve. The "tracking_actuate" mode is the same as "manual_actuate" mode because PWM signals are send to RTL and the controller is not used to calculate it. The difference is that the "manual_actuate" mode copies a predefined PWM signal path "ACTUATION" from shared memory for one movement and copies the joint positions of the movement to shared memory at the end of the movement (NumSample ≥ SampleNumber-1). This mode was added to allow system identification to be done on any actuator of the manipulator. System identification gauges the system response (position, velocity, acceleration) to different signals so that a mathematical model can be found to describe the system. This mathematical model can be used to aid controller design.

The rest of the interrupt function applies to all modes. After "SampleNumber" increments by a certain set amount i.e. 50 complete position readings (300 interrupts), the information of the joints at that instant is send through rtf "3" to be printed on the "visual" program. This is to inform the current status of the joint positions as well as the PWM signals send to each joint. It then increments "Actuator_No" or resets it to 0 if the current "Actuator_No" is 5 so that the joint serviced after Swivel is Rotate. Once "Actuator_No" has been switched to the next joint, that particular encoder (next joint) is turned on so that when the next interrupt happens, it would have been on long enoughfor the encoder output signal to be stable.

Next in the Interrupt function, "SampleNumber" is checked to see if it exceeds "NumSample-1" to check if a movement has been completed. If that is the case, RTL switches back to "hold" (default) mode only if it is not in the "manual_actuate" mode to make sure that a user can manually control the manipulator as long as possible. It also resets "SampleNumber" to 0 if "SampleNumber" exceeds "NumSample-1".

The whole Interrupt function finished but restarts when the UPP card generates the next interrupt.

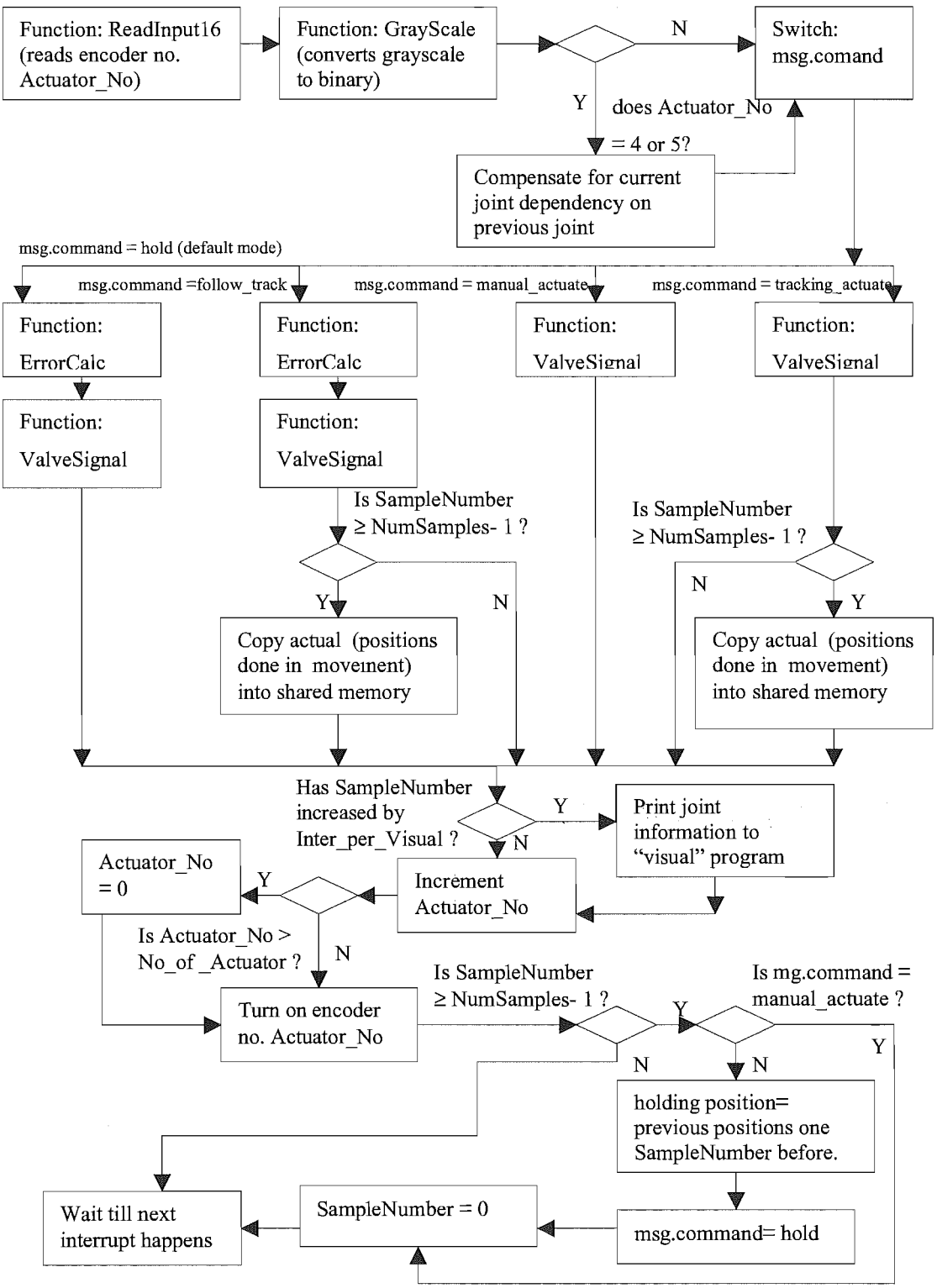

Figure 5.4 Interrupt Handler of "RT"

5.1.2 Compiling the Software

All the software needed to control the test rig is specified in the "makefile" (See Appendix D). When "make" is typed at the prompt in the directory where the "makefile" is kept, all the files in the "makefile" is compiled using the GNU C compiler.

In the case of the manipulator, "option.c", "visual.c" are compiled into "option" and "visual" respectively. The real-time software "RT.c" is compiled into "RT.o". Notice that compiling "RT.c" uses many real-time flags that "option.c" and "visual.c" do not require. "RT.o" functions in RTL however "option" and "visual" function in Linux.

5.1.3 Header Files

"rtshare.h" is a header file included in both "option.c" , "visual.c" and "RT.c" as it initialises variables that both these programs use. This ensures that these variables are of identical size (float, int, char etc) since information is transferred between these two programs via fifos and shared memory.

"kinematics.h" is a header file included in "option.c" that contain functions to solve for forward and inverse kinematics of the manipulator.

The header files PWM.H and Upp.h are both included in "RT.c" for accessing commands to control the UPP card.

5.2 USING THE SOFTWARE

In RTL, there are 5 modules that need to be inserted. One of the modules is "RT.o" and the rest are "PWM.o", "UPP.o", "rt_prio_sched.o" and "rt_prio_fifo.o". "PWM.o" and "UPP.o" are modules from which "PDcontrol.o" access the commands for the UPP card. "rt_prio_sched.o" and "rt_prio_fifo.o" are modules that come with the RTL kernel. It is from these two modules which "RT.o" access commands to set up timers and fifos respectively.

To insert the modules, the user must log into an "xterm" terminal as a super user. On the "xterm" prompt, type in "su" and hit "Enter". The "xterm" will then prompt for a password. When logged on as a super user, use the "insmod" command to insert the modules individually. At the prompt, type in the commands:

- insmod rt_prio_sched.o
- insmod rt_prio_fifo.o
- insmod UPP.o
- insmod PWM.o
- insmod RT.o

"UPP.o" has to be inserted before "PWM.o" as "PWM.o" has functions that refer to "UPP.o". To remove a module, simply type "rmmod" followed by the name of the module. A super user has authority to change almost everything on the Linux operating system; therefore the "xterm" where the super user role is assumed is only used for inserting and removing modules.

The next step in to launch the "option" program as a normal user in another "xterm". The user is then presented with a number of movement choices for the manipulator:

- Manual
- Multiple Step Input
- Single Step Input
- End Point and Angle Input

The "Manual" choice allows the user to control each actuator manually using the keyboard directly without having to hit "Enter". The "Multiple Step Input" choice prompts 6 positions for all six actuators to step to while "Single Step Input" prompts for which actuator to step and a position to step to. The "End Point and Angle Input" choice allows the user to define an end point and orientation for the end-effector of the manipulator to step to.

5.3 GRAYSCALE CONVERSION

The encoders on the manipulator are absolute encoders. Therefore, there is no need to initialise the encoders in any situation. However, the encoders are grayscale devices. Grayscale is a binary scale that changes 1 bit at a time for each position increment. Below is a table of normal 3 bits binary compared to their grayscale counterpart.

Decimal	**Binary**	**Grayscale**
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Table 5.1 Decimal, Binary and Grayscale conversion

It can be seen in table 5.1 that the binary sometimes changes multiple bits for an increment. For instance, the single increment from "001" to "010" changes both the first and second bit at the same time. This introduces the possibility of large errors if any of the changing bits were misread. Using the change from "001" to "010" as an example, "000" could be read if only the change in the first bit out of the two is detected. On the other hand, if only the change in the second bit is detected, it could read "011".

The advantage of grayscale is that it eliminates the possibility of large errors by limiting each increment to only one changing bit at any time. However, grayscales need to be converted to normal binary so that computers could recognise them for their value.

The conversion from grayscale to normal binary is defined as follow:

Binary$_i$= Grayscale$_i$ XOR Binary$_{i+1}$

The i-th bit of the binary scale is simply the ***XOR*** operation of the i-th bit of the grayscale with the (i+1)th bit of the binary scale. For any grayscale-binary conversion of ***n*** number of bits, the (n+1)th bit of the binary scale is "0" , to kick start the conversion process.

5.4 CONTROLLER DESIGN

The controller is implemented in RTL. The controller is used in every mode except "manual", where the actuation is controller manually by the user.

5.4.1 Controller Structure

For each of the 6 actuators, the controller is of the following form:

$$u(k+1) = \sum_{i=k-n+1}^{i=k}[w(i)u(i)] + \sum_{i=k-m+1}^{i=k}[K_p(m)e(i)] + K_d[\sum_{i=k-p+1}^{i=k}e(i)f(i)\ ]/p\Delta t + \sum^{k}[K_i e(k)]\Delta t \quad \textbf{(5.1)}$$

where:

k is the sampling number starting at 0 and incremented by one every 6th interrupt,

$u(k)$ is the output in PWM $\in$ {-100%, 100%},

$e(k)$ is the error signal (desired position- current position),

$f(p)$ is the filter coefficient to filter the error signal,

$K_p(m)$ is the proportional gains,

$w(n)$ is the weights for previous PWM outputs,

K_d is the derivative gain,

K_i is the integral gain,

Δt is the sampling interval or six interrupt interval.

A controller of such a form allows it to be either a P.I.D controller or an Observer Based controller. An Observer Based controller needs information on previous output, which a P.I.D. controller does not need. All the control coefficients are sent as one single structure from U.I. to RTL. This allows the controller to be changed at any time of operation to

make testing controllers flexible. It also allows adaptable or self-tuning controllers to be implemented.

Friction exists in all six joints. Therefore, a PWM of greater than zero is needed to start any of the six joints moving. It is therefore a requirement that the PWM signal send to the actuators has a minimum value that just starts it moving. The manipulator was tested in the "manual" mode to find these values and they are:

In/Out actuator: PWM ≥ 26, PWM ≤ 17;
Rotate actuator: PWM ≥ 22, PWM ≤ 18;
Up/Down actuator: PWM ≥ 14, PWM ≤ 26;
Bend actuator: PWM ≥ 25, PWM ≤ 28;
Yaw actuator: PWM ≥ 21, PWM ≤ 16;
Swivel actuator: PWM ≥ 20, PWM ≤ 20;

There are two controllers for the manipulator. One operates when in the 'hold' mode, and a different one for all the other modes. This is to ensure that the manipulator is able to stay stationary when in the 'hold' mode regardless of the capability of the controller being tested.

5.4.2 Control Responses

Figure 5.5 and Figure 5.6are step responses to a P.I. controller of values:
In/Out actuator: $K_p(n-1)= 20$, $K_i=200$;
Rotate actuator: $K_p(n-1)= 25$, $K_i=20$;
Up/Down actuator: $K_p(n-1)= 30$, $K_i=90$;
Bend actuator: $K_p(n-1)= 3$, $K_i=3$;
Yaw actuator: $K_p(n-1)= 4$, $K_i=20$;
Swivel actuator: $K_p(n-1)= 20$, $K_i=80$;

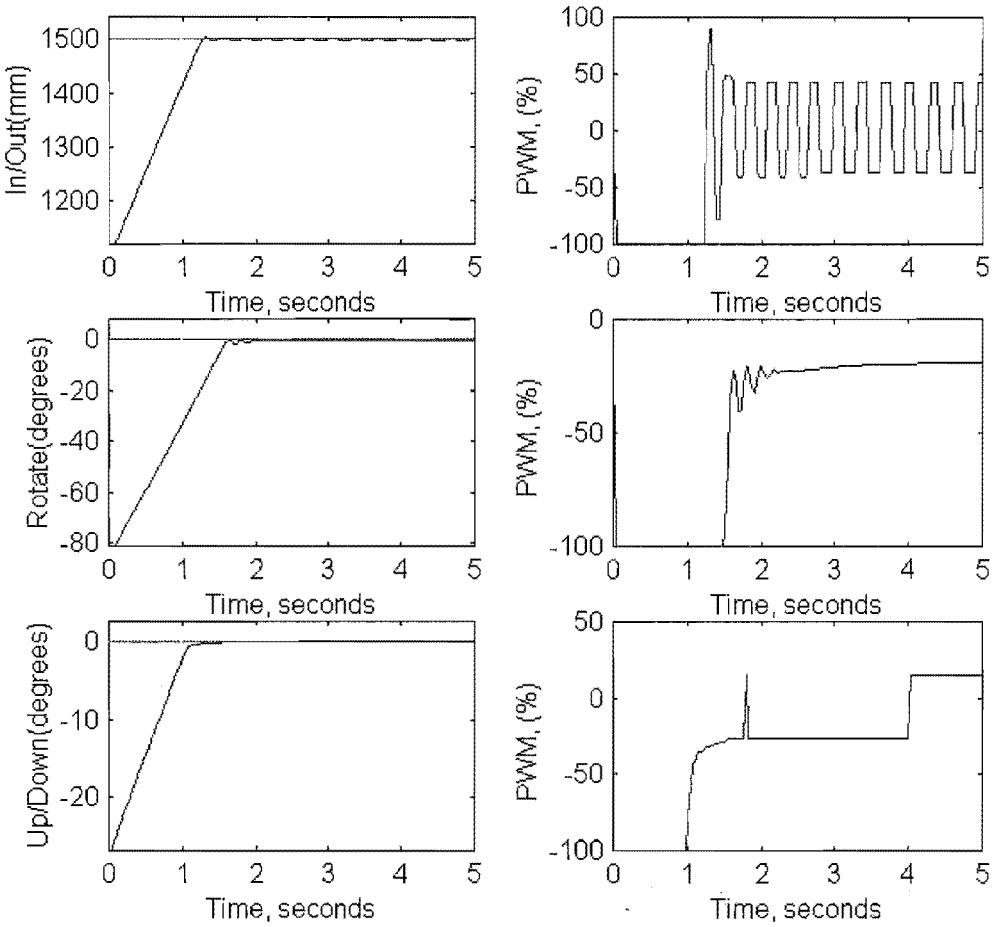

Figure 5.5 Step response for the arm of Unimate 2000B

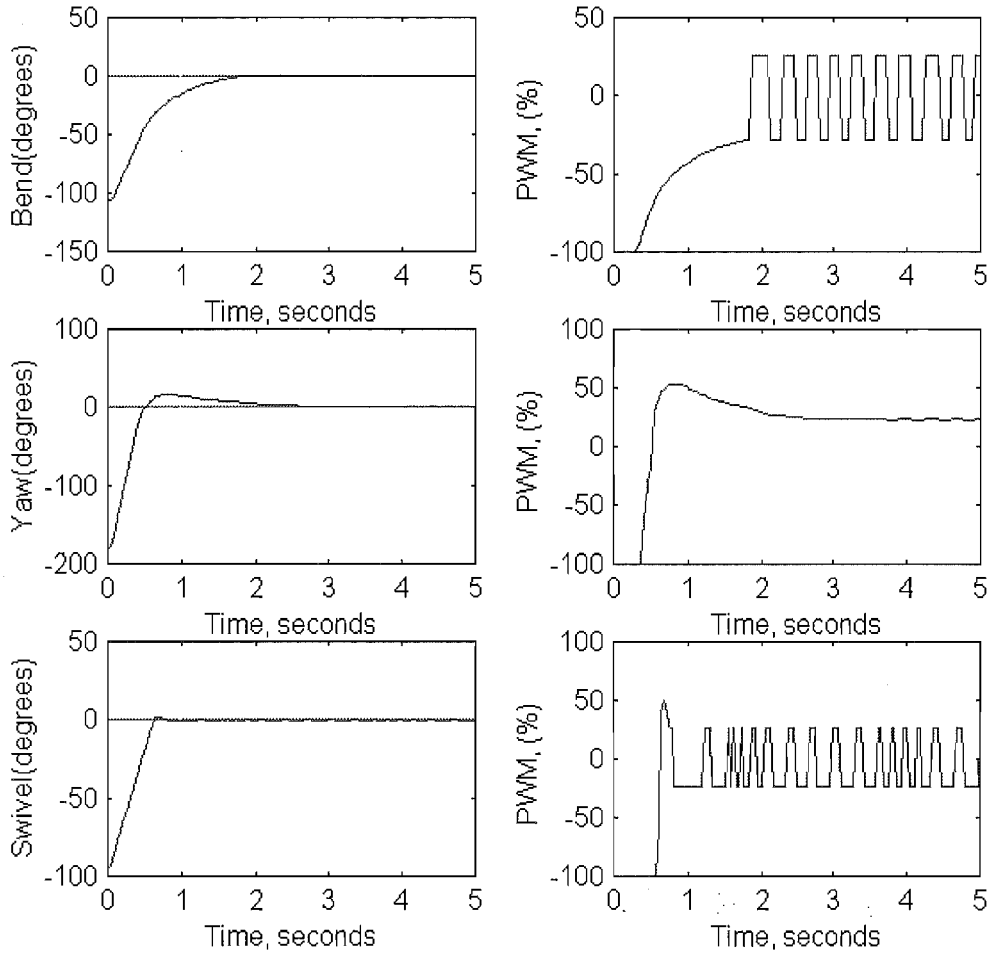

Figure 5.6 Step response for the wrist of Unimate 2000B

The abrupt switching in the PWM signal apparent in the In/Out, Bend, Swivel joints are due to the minimal absolute PWM settings discussed in the later part of section 5.4.1. The P.I. controller proves adequate in achieving simple path tracking for the manipulator.

CHAPTER 6- DISCUSSION and RECOMMENDATON

6.1 SOFTWARE IMPROVEMENTS

6.1.1 Real-Time Linux versus Real-Time Application Interface

There is another version of real-time kernel for Linux called RTAI (Real-Time Application Interface) developed by The Aerospace Engineering Department of Milan Polytechnic in Italy (Dipartimento di Ingegneria Aerospaziale Politecnico di Milano-DIAPM). The RTAI plug-in helps Linux to fulfil some real time constraints (few milliseconds deadline, no event loss). It is based on a RTHAL: Real Time Hardware Abstraction Layer. This concept is also known in Windows NT. The HAL exports some Linux data & functions closely related to the hardware. RTAI modifies them to get control over the hardware platform. That allows RTAI real time tasks to run concurrently with Linux processes. The HAL defines a clear Interface between RTAI & Linux [12].

A feature of RTAI implementation is that interrupt handlers preambles take care of the task switch (TS) flag within RTAI. Thus floating point operations can be freely used in interrupt handlers, without causing a trap fault from Linux processes. On the other hand, RTL has a function that executes when an interrupt is generated by the UPP card. This function starts by saving the FPU (Floating-Point Unit) registers and clearing the task

switch flag in the CR0 register so that the processor ignores the exceptions that are generated when the FPU is used later. See section 2.3 for more details. This capability of RTAI is extremely useful as floating-point calculations are used extensively in robotics and control applications. Having a real-time kernel that saves the floating-point registers in the FPU reliably by itself allows it to interface to any hardware. The programming of modules in RTAI is similar to RTL because fifos and shared memory are used. Therefore, it is easy for an RTL programmer to switch to RTAI.

6.1.2 Extensions to RTL for control applications

There are free software-extensions developed by other technical institutes to run on Linux. These have built in functions to simplify the development and tuning of controllers in RTL. However, these do not support the UPP card but the source codes of these programs could be changed to accommodate the UPP card. Two such extensions are:

6.1.2.1 RTiC-Lab

The Real Time Controls Laboratory (RTiC-Lab) was developed by the Rotating Machinery and Controls (ROMAC) Laboratories at the University of Virginia. It is a semi-detached open source software designed to run on Linux and RTL. It is designed as an easy to use controls prototyping tool. It gives the controls engineer real time access to:

- Plant states,
- Plant I/O,
- Controller states,
- Controller parameters (scalar or matrix), and
- Hard real time environment for plant modelling

Run time data can currently be saved to:

- stdout, and data files

Although modules are now being coded to add plotting capabilities. Most importantly, RTiC-Lab is intended to be extensible by creating a simple to use interface in Linux so that users can add their own modules. RTiC-Lab is released under the Free Software

Foundation's General Public License (GPL), allowing the source codes to be obtained and modified.

6.1.2.2 RTLT

Real-Time Linux Target (RTLT) is a software package developed by Quality Real Time Systems that gives the user the ability to implement a Simulink block diagram on a standard Intel PC in hard real-time. Specifically, RTLT is a set of source files, device driver libraries, a template makefile, and an MEX-file interface that uses RTW (Real-Time Workshop) to automatically generate C code from a user defined Simulink block diagram. The C code is first generated and compiled on a PC running RT-Linux. A target for running the generated code is then built on the same PC.

During the execution of a Simulink block diagram, RTLT captures sampled data from one or more input channels using standard I/O boards (*e.g.*, A/D channels, digital lines, and encoder lines, *etc.*). RTLT then provides the data to the block diagram model. The Simulink block diagram model then processes the data accordingly. RTLT then outputs the processed data via one or more output channels (*e.g.*, D/A channels). A custom Simulink block library and four different hardware I/O board drivers are also provided. The user can also observe the behaviour of any signal during or after the real-time run via the Simulink Scope blocks. If the user builds the Simulink code in the external mode, the user can perform on-line parameter tuning during real-time execution.

6.1.3 Graphical User Interface (GUI)

The user interface (UI) at the moment are just prompts on the xterm windows. A GUI ought to be developed on the Linux side. This can be done using the Java programming language, which is now supported on Linux. A development tool kit for Java can be obtained through their web site for free. The advantage of using Java is that it has reliable graphics library and is constantly improved by a dedicated group of people at Sun Micro Systems.

Java is also a web-enabled language that allows it to run through any web browsers like Internet Explorer and Netscape Navigator. The Java interpreter unique to each operating system also allows for easy portability between different operating systems.

6.2 HARDWARE PROBLEMS

It was noted in section 4.6.4 the absolute encoders have to be switched on before reading. When interrupts of less than 6 ms were used to switch between actuators, there can be 2 or 3 erroneous readings from the encoders due to the fact that the encoders are not fully switched on when the reading occurs. Figure 6.1 shows a stepping response for all actuators with an interrupt time of 1ms. Notice the two erroneous readings for the In/Out, Up/Down and Bend joints individually. Also, notice the three erroneous readings for the Yaw joint.

Therefore, the encoders need to be improved physically or the interrupt time will have to be kept at 6 ms and above to ensure enough time to switch on the encoder. An interrupt between actuators of 6 ms would results in a complete position reading of the manipulator in 36ms.

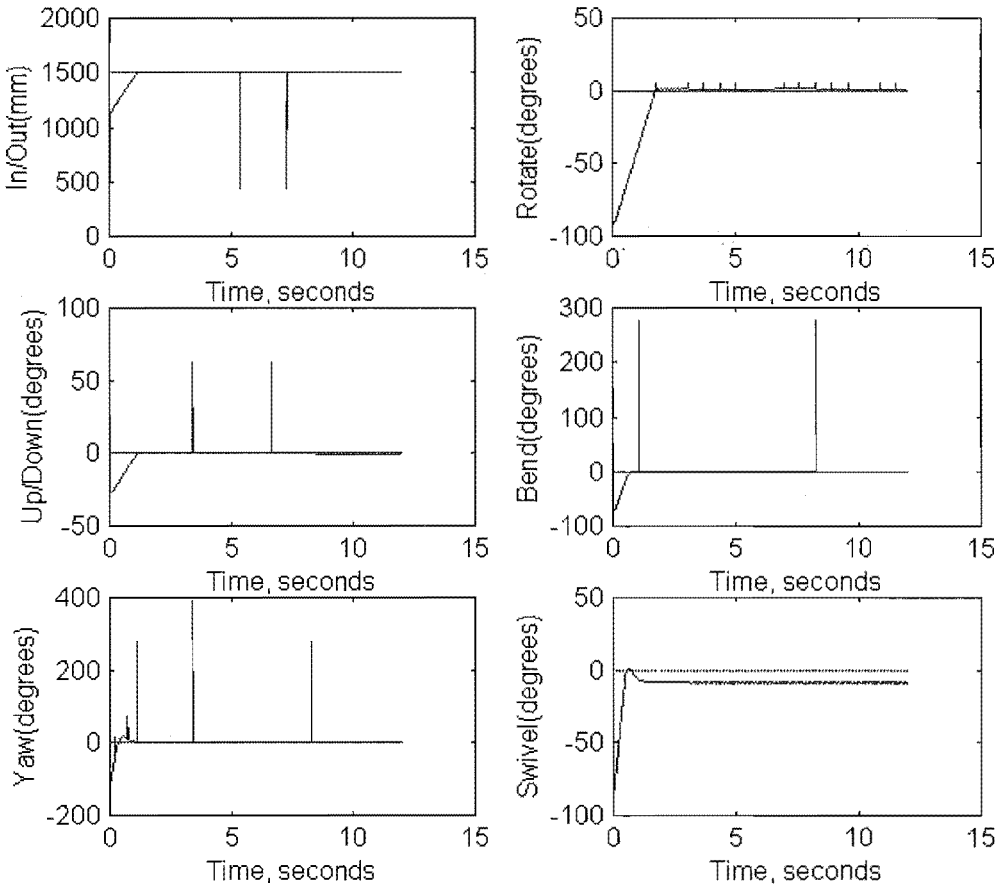

Figure 6.1 Erroneous readings at an interrupt time of 1ms per axis.

6.3 CONTROLLER IMPLEMENTATION

The software design has been implemented in such a way that the user can change the controller coefficients while the robot is moving (refer to section 5.4). If an adaptive algorithm is to be utilised for identifying the parameters of the robot, the algorithm can be implemented on Linux ("option") and when the latest estimates have been calculated, new controller coefficients can be determined and sent to the RTL ("RT.o") side. This way, RTL can be saved from doing extensive processing and left to run the controller reliably.

System identification could also be easily extended into the software, as there is already a "tracking_actuate" mode in RTL ("RT.o") that allows the robot actuators to follow PWM signals send to it by Linux ("option"). This allows the user to gauge the system response from tailored inputs so that a mathematical model of the system can be identified. The System Identification toolbox in Matlab correlates signal input to system response (position, velocity and acceleration) and also allows different types of system models to be derived (State Space, ARX, etc).

6.4 PATH GENERATION

The software has not been written to perform path tracking, as not enough information on the parameters of the robot is known for a feed-forward controller. A function can be easily written to generate a path from a minimum of three points. A path has to have a starting and ending point with at least one intermediate point between the former two points. A polynomial or cubic spline interpolation can be used to generate the path [14].

The end-effector position is defined by 6 variables (x, y, z, γ, β, α), see section 4.3 for more details. It would be troublesome to have to define 6 variables for each point as there are at least three points needed to generate a path. In a pick and place movement of a manipulator, it is often desired that the end-effector keeps a constant orientation. Imagine a manipulator picking up a cup; it would be desirable that the contents of the cup do not get tipped out along the path i.e. keep the cup level.

Therefore, it is vital that the γ and β angles do not change along the path. The α angle has to change in accordance to the Rotate joint angle in a 1:1 relationship. If the end-effector is to move from point "A" to point "C" through point "B", with the knowledge of the starting position (x_A, y_A, z_A) and orientation (γ_A, β_A, α_A), all we need now is the intermediate position (x_B, y_B, z_B) and ending position (x_C, y_C, z_C) to define the path. This is due to the fact that the γ and β angles are constant throughout the path. The α_B and α_C can be found using the (x_B, y_B, α_A) and (x_C, y_C, α_A) variables respectively.

Since the software has functions to calculate the inverse kinematics of the manipulator, it is only a matter of using appropriate interpolation algorithm to generate a path.

CHAPTER 7- CONCLUSION

The Unimate 2000B is in adequate working order and can still be used for further research into robotics. With its inverse kinematics problem solved, it is ready for research in robotics and controller design (control algorithm). In the case of the current software, a feedback controller was implemented with no feed-forward. Feed-forward is a controller design based on the mathematically derived dynamics of a manipulator. It linearises the non-linear dynamics of a manipulator to allow trajectory following. Since the dynamic model of the Unimate 2000B is not known, a feed-forward controller was not implemented.

RTL has been found to be excellent for the control of the Unimate 2000B. RTL would also be more than adequate in the control of most mechanical systems where a sampling rate of 1 kHz is considered very fast. A 486-66MHz PC running RTL using the FPU has a latency of not more than 70 μs. This would mean a maximum latency of 7% for interrupts of 1 ms, see section 2.3 for more detail. For faster computers such as high end Pentiums, the latency could be a lot less, but cache flushing takes extra time.

In the research of digital control of mechanical systems, DSPs are frequently used. Most mechanical engineers have poor knowledge of digital hardware. The use of real-time operating systems saves mechanical engineers from having to familiarise themselves with the hardware architecture of a DSP. DSPs are notorious for their fast paced changes on

the market and vendors of DSP embedded systems rarely make provision to interface older boards with the newer ones and the I/O features might vary too. There is also the problem that DSP embedded systems have limited RAM for data logging.

Therefore, RTL is ideal for the research on mechanical control systems. Once a good control algorithm has been developed through the use of RTL, the algorithm can then be ported to a DSP. With a clear understanding of the control requirements after implementation on RTL, a suitable DSP embedded system can be chosen for the task.
Real-Time operating systems will play an increasingly important role in the research of digital controllers for mechanical systems. The research into robotics and real-time operating systems reported in this thesis has shown that a comprehensive system capable of meeting the current future needs of robotics research has been developed. That this development is freely available and supported by academic institutions worldwide is a great benefit.

REFERENCES

[1] Asada, H. and Slotine, J. J. E. ***Robot Analysis and Control.*** *John Wiley and Sons, Inc.* 1986.
TJ 211.A798

[2] Craig, John. J. ***Introduction to Robotics.*** *Addison-Wesley, Publishing Company.* 1989.
TJ 211.C886

[3] Dunlop, G. R. and Murphy, J. D. J. ***A universal pulse processing board for instrumentation and machine control, Proc.*** *NELCON'93*, Auckland, New Zealand **1**, 171-178. 1993

[4] Dunlop, G. R. and Hampson, S. P. (1995) ***An integrated digital control system for hydraulic actuators.*** *Innovations in Fluid Power* (Seventh Bath International Fluid Power Workshop), Eds. Burrows C R and Edge K A, R.S.P. John Wiley & Sons Inc., ISBN0 471 95688 0 Ch 13 pp177-186.

[5] Dunlop, G. R. and Hampson, S. P. (1995) ***Bond graphs for nonlinear modeling of an electro-hydraulic system.*** *Active Control in Mechanical Engineering*, Paris, France, Ed. L. Jézéquel, Editions Hermès, ISBN 2-86601-450-2, pp57-68.

[6] Donald, S.D. ***Commission of a Hydraulic Powered Robot.*** *Christchurch, University of Canterbury.* 1998. Project No: 30. (3rd pro project report: Mechanical Engineering)

[7] Hampson, S.P. and Dunlop, G. R. ***Digital controlled hydraulic rig suitable for teaching advanced control theory.*** *IPENZ Trans* **19**,1, 28-33.

[8] Hampson, S.P.; Sirisena, H. R. and Dunlop, G. R. (1995) ***Model predictive control of a hydraulic actuator***, Proc. *SICICI'95*, IEEE Singapore International Conf. on Intelligent Control, Singapore, July 3-7, **1**, 396-401.

[9] Hilton, E.F. ***RTiC-Lab*** (Real-Time Controls Laboratory). http://128.143.47.231/~efh4v/rtic-lab.html

[10] Johnson, M.K. and Troan E.W. ***Linux Application Development.*** *Addison-Wesley.* 1998.

[11] Kelly, A. and Pohl, I. ***A Book on C.*** *Addison-Wesley.* 1995

[12] Mantegazza, P.;Bianchi, E.;Dozio, L. ***DIAPM-RTAI*** (Dipartimento di Ingegneria Aerospaziale Politecnico di Milano- Real-Time Application Interface). http://www.aero.polimi.it/projects/rtai/

[13] Palmer, C.N. ***Computer Control of a Hydraulic Test Rig***. *Christchurch, University of Canterbury.* 1998. Project No: 32. (3rd pro project report: Mechanical Engineering)

[14] Press, W.H.; Teukolsky, S.A.; Vetterling, W.T. and Flannery, B.P. ***Numerical Recipes in C.*** *Cambridge University Press.* 1988.

[15] Sciavicco, L. Siciliano, B. ***Modeling and Control of Robot Manipulators.*** *McGraw-Hill.* 1996.
TJ 211.S416

[16] Yao, Z.G. ***RTLT (Real-Time Linux Target).*** http://www.qrts.com/products/rtlinuxtarget/index.shtml

[17] Yodaiken, V. ***The RT-Linux approach to hard real-time.*** http://luz.cs.nmt.edu/rtlinux.new/documents/papers/whitepaper.html

[18] Yodaiken, V.; Barabanov, M. ***A Real-Time Linux.*** http://www.rtlinux.org/rtlinux.new/documents/papers/lj.ps

APPENDIX A- INVERSE KINEMATICS of UNIMATE 2000B

Inverse Kinematics:

In T_6^1 (3,3)

$s\theta_1 r_{13} - c\theta_1 r_{23} = c\theta_5$ (1)

In T_6^1 (3,4)

$s\theta_1 p_x - c\theta_1 p_y = c\theta_5 L_y$ (2)

(2)

$\Rightarrow$ $s\theta_1 p_x / L_y - c\theta_1 p_y / L_y = c\theta_5$ (3)

L.H.S. of (1) equals L.H.S. of (3)

$\Rightarrow$ $s\theta_1 r_{13} - c\theta_1 r_{23} = s\theta_1 p_x / L_y - c\theta_1 p_y / L_y$

$\Rightarrow$ $t\theta_1 r_{13} - r_{23} = t\theta_1 p_x / L_y - p_y / L_y$

$\Rightarrow$ $t\theta_1 (r_{13} - p_x / L_y) = r_{23} - p_y / L_y$

$\Rightarrow$ $\mathbf{\theta_1 = \tan^{-1}(r_{23} - p_y / L_y , r_{13} - p_x / L_y)}$

In T_6^2 (2,1)

$s\theta_1 r_{11} - c\theta_1 r_{21} = s\theta_5 c\theta_6$ (4)

In T_6^2 (2,2)

$s\theta_1 r_{12} - c\theta_1 r_{22} = -s\theta_5 s\theta_6$ (5)

(5)/(4)

$\Rightarrow$ $-t\theta_6 = (s\theta_1 r_{12} - c\theta_1 r_{22}) / (s\theta_1 r_{11} - c\theta_1 r_{21})$

$\Rightarrow$ $\mathbf{\theta_6 = \tan^{-1}(c\theta_1 r_{22} - s\theta_1 r_{12} , s\theta_1 r_{11} - c\theta_1 r_{21})}$

In T_5^1 (3,1)

$s\theta_5 = c\theta_6 s\theta_1 r_{11} - c\theta_6 c\theta_1 r_{21} - s\theta_6 s\theta_1 r_{12} - s\theta_6 c\theta_1 r_{22}$ (6)

In T_5^1 (3,3)

$c\theta_5 = s\theta_1 r_{13} - c\theta_1 r_{23}$ (7)

(6)/(7)

⇨ $t\theta_5 = (c\theta_6 s\theta_1 r_{11} - c\theta_6 c\theta_1 r_{21} - s\theta_6 s\theta_1 r_{12} - s\theta_6 c\theta_1 r_{22}) / (s\theta_1 r_{13} - c\theta_1 r_{23})$

⇨ **$\theta_5 = \text{Atan2}(c\theta_6 s\theta_1 r_{11} - c\theta_6 c\theta_1 r_{21} - s\theta_6 s\theta_1 r_{12} - s\theta_6 c\theta_1 r_{22} , s\theta_1 r_{13} - c\theta_1 r_{23})$**

In T_3^0 (1,4)

$L_b r_{11} s\theta_6 + L_b r_{12} c\theta_6 - L_y r_{13} + p_x = c\theta_1 c\theta_2 d_3 - c\theta_1 s\theta_2 a_2$ (8)

In T_4^1 (2,4)

$s\theta_2 d_3 - c\theta_2 a_2 = L_b r_{31} s\theta_6 + L_b r_{32} c\theta_6 - L_y r_{33} + p_z - d_1$ (9)

(8)

⇨ $d_3 = (L_b r_{11} s\theta_6 + L_b r_{12} c\theta_6 - L_y r_{13} + p_x + c\theta_1 s\theta_2 a_2) / c\theta_1 c\theta_2$ (10)

(9)

⇨ $d_3 = (L_b r_{31} s\theta_6 + L_b r_{32} c\theta_6 - L_y r_{33} + p_z - d_1 - c\theta_2 a_2) / s\theta_2$ (11)

R.H.S of (10) equals R.H.S of (11)

⇨ $(L_b r_{11} s\theta_6 + L_b r_{12} c\theta_6 - L_y r_{13} + p_x + c\theta_1 s\theta_2 a_2) / c\theta_1 c\theta_2$
$= (L_b r_{31} s\theta_6 + L_b r_{32} c\theta_6 - L_y r_{33} + p_z - d_1 - c\theta_2 a_2) / s\theta_2$

⇨ $L_b r_{31} s\theta_6 + L_b r_{32} c\theta_6 - L_y r_{33} + p_z - d_1 - c\theta_2 a_2$
$= t\theta_2 (L_b r_{11} s\theta_6 + L_b r_{12} c\theta_6 - L_y r_{13} + p_x + c\theta_1 s\theta_2 a_2) / c\theta_1$

⇨ **$L_b r_{31} s\theta_6 + L_b r_{32} c\theta_6 - L_y r_{33} + p_z - d_1$**
$= t\theta_2 ((L_b r_{11} s\theta_6 + L_b r_{12} c\theta_6 - L_y r_{13} + p_x)/c\theta_1) + t\theta_2 a_2 s\theta_2 + a_2 c\theta_2$
(Numerically solve for θ_2)

In T_4^0 (3,1)

$-s\theta_2 s\theta_4 + c\theta_2 c\theta_4 = c\theta_5 c\theta_6 r_{31} - c\theta_5 s\theta_6 r_{32} - r_{33} s\theta_5$ (12)

In T_4^0 (3,3)

$c\theta_2\, s\theta_4 + s\theta_2\, c\theta_4 = -r_{31}s\theta_6 - r_{32}c\theta_6$ (13)

(12)

$\Rightarrow$ $c(\theta_2 + \theta_4) = c\theta_5 c\theta_6 r_{31} - c\theta_5 s\theta_6 r_{32} - r_{33}s\theta_5$ (14)

(13)

$\Rightarrow$ $s(\theta_2 + \theta_4) = -r_{31}s\theta_6 - r_{32}c\theta_6$ (15)

(15)/(14)

$\Rightarrow$ $t(\theta_2 + \theta_4) = (-r_{31}s\theta_6 - r_{32}c\theta_6) / (c\theta_5 c\theta_6 r_{31} - c\theta_5 s\theta_6 r_{32} - r_{33}s\theta_5)$

$\Rightarrow$ $\mathbf{\theta_4 = Atan2((-r_{31}s\theta_6 - r_{32}c\theta_6), (c\theta_5 c\theta_6 r_{31} - c\theta_5 s\theta_6 r_{32} - r_{33}s\theta_5)) - \theta_2}$

(11)

$\Rightarrow$ $\mathbf{d_3 = (L_b r_{31} s\theta_6 + L_b r_{32} c\theta_6 - r_{33} L_y + p_z - d_1 - a_2\, c\theta_2) / s\theta_2}$

APPENDIX B- SOFTWARE FOR TESTING INTERRUPT LATENCY

```
# makefile

all:  intlaten.o  IntCon

# you might have to change this

# the path to the rt-linux kernel

RTL_DIR = /usr/src/rtl
RTLINUX_DIR = /usr/src/linux
INCLUDE= -I/usr/src/linux/include -I//usr/src/rtl/include
CFLAGS = -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -fno-
strength-reduce -D__RT__ -D__KERNEL__ -DMODULE -c

EXECFLAGS = -O2 -Wall

intlaten.o: intlaten.c
      gcc ${INCLUDE} ${CFLAGS} intlaten.c


IntCon: IntCon.c
      gcc ${INCLUDE} ${EXECFLAGS} -g  -o IntCon IntCon.c
clean:
      rm -f intlaten.o
      rm -f IntCon
```

```
/* MyFifo.h */

#ifndef MYFIFO_H_
#define MYFIFO_H_

#define START_TASK      1
#define STOP_TASK 2
#define READ_REG        3

typedef struct my_msg_struct {
      int command;
      int task;
      int period;
}MyMsg;

#endif
```

```
/* intlaten.c */

#define MODULE
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/version.h>
#include <linux/errno.h>
#include <asm/rt_irq.h>
#include <asm/io.h>
#include </usr/src/rtl/include/rtl_sched.h>
#include </usr/src/rtl/include/rtl_nfifo.h>
#include </usr/src/rtl/include/rtl_fifo.h>
#include "Upp.h"
#include "MyFifo.h"
#include </usr/src/rtl/include/rtl_sync.h>

#define Period 1000

static UINT card;

static      int IntCount=0;
static      int Running=FALSE;
static      int IntTime;


int SaveFpuRegs[28];

void IntFunc(void)
{

int Flags;
long linuxCR0;

   rtl_no_interrupts(Flags);             // disable interrupts
   __asm__ __volatile__("movl %%cr0,%%eax":"=a"(linuxCR0)::"ax");
   __asm__ __volatile__("clts");         // clear task switch flag
   __asm__ __volatile__("fsave %0" : "=m" (SaveFpuRegs));


  //IntTime=UPPReadData(card,18);    // get counter value

            IntTime=UPPReadData(card,18);    // get counter value

            if (Running)
      {
            rtf_put(1, &IntTime, sizeof(IntTime));
      }
      IntCount++;
      UPPIntStatClear(card,1);          // clear interrupt flag

   __asm__ __volatile__("frstor %0" : "=m" (SaveFpuRegs));
                                        // restore floating point regs
    __asm__ __volatile__("movl %%eax,%%cr0"::"a"(linuxCR0):"ax");
                                        // restore cr0
   rtl_restore_interrupts(Flags);       // restore interrupt enable

}
```

```
static void LoadUpp(void)
{

/* The following section contains the program that is placed into the
UPP
       microcontroller on the UPP card
*/


static ProgType pwc[]=
{
      {"USCR",0x00},/* Stop The UPP */
      {"MFNR",0x02},/* Set the number of functions to be programmed to
1 */

      {"FNR",0x01},/* Select function 1 for programming     */
      {"CMR",0x50},/* INC command for interrupt*/
      {"RASRA",0x0F2},/* Counter register */
      {"RASRB",0x0F3},/* Compare register  */
      {"IOARA",0x0FF},
      {"IOARB",0x0FF},
      {"IOARC",0x00},   /* 0x0F1 Select internal port for output of
signal */
      {"IOARD",0x0FF},

      {"UCER2",0x00},// disable upp contact lines
      {"UCER1",0x00},/* disable all UPP contact lines       */
      {"DDR1",0xFF},
      {"DDR2",0xFF},
      {"NDER",0x00},/* Disable the next data register       */
      {"USCR",0x02},/* Start The UPP            */
      {"",0}
};

      card=0x230;      // address for pwm stuff
      UPPLoadProgram(card,pwc);

      UPPWriteData(card,19,Period);             // set compare register
      UPPWriteData(card,18,0);               // reset counter
      UPPIntStatClear(card,1);              // clear interrupt flag
      UPPIntEnableWrite(card,1,0x01);           // enable interrupt on
UPP
}
int nummm[]={7,6,5,4,3,2,1,0};
static ProgType pwcC[]=
{
      {"USCR",0x00},/* Stop The UPP */
      {"UCER2",0x0},// disable upp contact lines
      {"UCER1",0x0},/* disable all UPP contact lines  */
      {"DDR1",0x0}, // data dir input
      {"DDR2",0x0},
};
int FifoHandler(unsigned int fifo)
```

```
{
MyMsg msg;
int err;
int Data;
float A=1.2,B=2.3,C;
      while ((err = rtf_get(2, &msg, sizeof(msg))) == sizeof(msg))
      {
            switch (msg.command)
              {
              case START_TASK:
                LoadUpp();
                Running=TRUE;
                IntCount=0;
                //  rtf_put(1, nummm, sizeof(int)*8);
C=A*B;
                break;
                case READ_REG:
                Data=12345;
                 rtf_put(1,&C,sizeof(C));
                // UPPWriteData(card,19,Data);
                 //Data=inportb(0x21); //UPPReadData(card,18);
                // rtf_put(1,&Data,sizeof(Data));
                //rtf_put(1,&IntCount,sizeof(IntCount));
                break;
              case STOP_TASK:
                      Running=FALSE;
                UPPIntEnableWrite(card,3,0);      // disable interrupts
                UPPLoadProgram(card,pwcC);      // stop upp

                break;
            default:    return -EINVAL;
            }
      }

      if (err != 0) {
            return -EINVAL;
      }
      return 0;
}
int numm[]={0,1,2,3,4,5,6,7};
int init_module(void)
{
      rtf_create(1, 4000);
      rtf_create(2, 400);           //input control channel
      rtf_create_handler(2, FifoHandler);
            request_RTirq(IRQ7, IntFunc);
      return 0;
}

void cleanup_module(void)
{
      UPPIntEnableWrite(card,1,0);      // disable interrupts

      rtf_destroy(1);
      rtf_destroy(2);
            free_RTirq(IRQ7);
}
```

```
/* IntCon.c */

#include <stdio.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <rtl_fifo.h>
#include <asm/rt_time.h>

#include "MyFifo.h"


int buf[50];


int main()
{
      int fd0, ctl;
      MyMsg msg;
int i,j,n;
int Max=0,Min=1000;
float Total=0;
int g, bar[20]={0};

      if ((fd0 = open("/dev/rtf1", O_RDONLY)) < 0)
      {
            fprintf(stderr, "Error opening /dev/rtf1\n");
            exit(1);
      }

      if ((ctl = open("/dev/rtf2", O_WRONLY)) < 0)
      {
            fprintf(stderr, "Error opening /dev/rtf2\n");
            exit(1);
      }

      /* now start the tasks */
      msg.command = START_TASK;
      msg.period = 1000;

      if (write(ctl, &msg, sizeof(msg)) < 0)
      {
            fprintf(stderr, "Can't send a command to RT-task\n");
            exit(1);
      }
```

```
for (i = 0; i < 100000; )
      {

      msg.command = READ_REG;
      msg.period = 1000;


            {
                  {

                        n = read(fd0, buf, sizeof(buf));
                       n=n/sizeof(int);

                        for (j=0;j<n;j++)
                       {
                             if (buf[j] > Max) Max=buf[j];
                             if (buf[j] < Min) Min=buf[j];
                             for(g=0; g< 20; ++g)
                               {
                                 if( buf[j]>= g*10 && buf[j]<=
(g+1)*10)
                                    ++bar[g];
                               }
                             Total+=buf[j];
                       }
                             i=i+n;
                       }
            }
      }
      msg.command = STOP_TASK;
      if (write(ctl, &msg, sizeof(msg)) < 0)
      {
            fprintf(stderr, "Can't send a command to RT-task\n");
            exit(1);
      }

      printf("\n\nMax %f microsecond\nMin %f microsecond\nAverage %f
microsecond\n",(float)Max/2.0,(float)Min/2.0,(float)Total/(float)i/2.0)
;
      for(g=0; g< 20; ++g)
        { printf("number btwn %d-%d microsecond: %d\n", g*5, (g+1)*5,
bar[g]); }
      return 0;
}
```

APPENDIX C- CONTROL SOFTWARE FOR SINGLE-AXIS HYDRAULIC TEST RIG

```
/* makefile */

all: function.o option PDcontrol.o

# you might have to change this

# the path to the rt-linux kernel
RTLINUX = ../../../linux
INCLUDE = ${RTLINUX}/include/linux

CFLAGS = -O2 -Wall

function.o:  function.c option.c MyFifo.h
      gcc -I${INCLUDE} ${CFLAGS}  -c -g  function.c

PDcontrol.o: PDcontrol.c MyFifo.h
      gcc -I${INCLUDE} ${CFLAGS}  -D__KERNEL__ -D__RT__  -c PDcontrol.c

option: option.c function.c MyFifo.h
      gcc -I${INCLUDE} ${CFLAGS}  -o  option option.c function.o -lm
clean:
      rm -f  PDcontrol.o function.o option option.c~ PDcontrol.c~
getkey.h~ function.c~ makefile~ MyFifo.h~
```

```
/* MyFifo.h */

#define START_TASK      1
#define SLOW            2
#define STOP_TASK       3
#define INITIALISED     4

#define size            200
#define NumSamples      16000
#define MaxSetting      100
#define TerminalCount   5000
#define Pulse           3.25e-6
#define InterPeriod     300
#define SampleTime      Pulse*InterPeriod

struct my_msg_struct {
  int command, Max;
};

struct control_data {
  float PosG, DerG, IntG, Signal;
};

struct array {
            int SetPos;
};

struct my_msg_struct msg;
struct control_data data;
struct array ary;

int End;
int inf1, inf2;
int ctl, fdb, dcs, dta;
int StartPosition, EndPosition;
float InputDerG, InputIntG, InputPosG, Integ;
int SampleNumber, Number;
int ValveSetting[NumSamples], ReadPosition[NumSamples],
SetPosition[NumSamples], RamPosition[NumSamples];

void Write_fifo(void);
void ErrorCalc(void);
void ResetValues(void);
void Manual(void);
void SetPulse(void);
void Input(void);
void move(void);
void Initialise(void);
void Finish(void);
void SaveData(void);
void StepInput(void);
void RampInput(void);
void MultiStep(void);
void Sinewave(void);
void FollowPath(void);
void ValveSignal (float, int);
void Interrupt(void);
```

```
/* PDcontrol.c */

#define MODULE
#include <math.h>
#include <linux/module.h>
#include <linux/errno.h>
#include </usr/src/rtl/include/rtl_sched.h>
#include </usr/src/rtl/include/rtl_fifo.h>
#include "PWM.H"
#include "MyFifo.h"
#include "Upp.h"

#define NumChannels 2
int ArrayPWM[NumChannels];
int TypePWM[NumChannels]= {2, 2};
RTIME begin, end, latent;
int between;

/*-------------------------------------------------------------------------
*/

void Interrupt(void)
{
  end= rt_get_time();
  switch (msg.command)
    {
    case START_TASK:
      ErrorCalc();
      break;

    case SLOW:
      ValveSignal(data.Signal, msg.Max);
      break;

    default:
      ValveSignal(0.0, 20);
      break;
    }
}
/*-------------------------------------------------------------------------
*/
```

```
/*-------------------------------------------------------------------
*/
void ErrorCalc(void)
{
float Distance[4], Deriv, Result;
int i, Intmax= 50, Maxvalve= 100;
//float SampleTime= InterPeriod* 3.25e-6;

SampleNumber++;

if (SampleNumber >= NumSamples) {StopTimerInt();}

rtf_get(3, &ary, sizeof(int));

RamPosition[SampleNumber]= ReadSingleEncoder(0);
Distance[3]= ary.SetPos- RamPosition[SampleNumber];
//LastDistance= ary.SetPos- RamPosition[SampleNumber-1];

Deriv= (11*Distance[3]- 18*Distance[2]+ 9*Distance[1]- 2*Distance[0])/
6*SampleTime;
Integ= Integ+ (data.IntG*Distance[0]*SampleTime);
 if (Integ > Intmax)
     Integ= Intmax;
 if (Integ < -Intmax)
     Integ= -Intmax;

Result= (data.PosG* Distance[3])+ (Integ)+ (data.DerG* Deriv);

ValveSignal(Result, Maxvalve);

 for (i=0; i<4; i++) {Distance[i]= Distance[i+1];}

return;
}
/*-------------------------------------------------------------------
*/
```

```
/*-------------------------------------------------------------------------
*/
void ValveSignal (float Signal, int Max)

{
float PWM1, PWM2;
float AreaRatio= 0.44;

if (Signal>= 0)
  { PWM2=0;
      if (Signal< Max)
      PWM1= Signal;
      else
      PWM1= Max;
  }

if (Signal< 0)
  { PWM1= 0;
     if (Signal>= -Max)
       PWM2= (-(Signal));
     else
       PWM2= Max;
  }
ArrayPWM[0]= PWM1;
ArrayPWM[1]= PWM2* AreaRatio;
SetPWMValues(ArrayPWM);
inf1= ReadSingleEncoder(0);
rtf_put(2, &inf1, sizeof(inf1));
inf2= PWM1- PWM2;
//inf2= end -begin;
rtf_put(2, &inf2, sizeof(inf2));
begin= rt_get_time();
return;
}
/*-------------------------------------------------------------------------
*/
int my_handler(unsigned int fifo)
{
      int err, n, i;

      while ((err = rtf_get(1, &msg, sizeof(msg))) == sizeof(msg) ) {
            switch (msg.command) {
              case START_TASK:
                          rtf_get(4, &data, sizeof(data));
                        StartTimerInt(InterPeriod);
                      break;
                  case SLOW:
                        rtf_get(4, &data, sizeof(data));
                        StartTimerInt(InterPeriod);
                      break;
                  case INITIALISED:
                      SetEncoder(0, 5);
                      break;
                case STOP_TASK:
                        StopTimerInt();
                      ValveSignal(0.0,20);
                      for(i=0; i<2; i++)
```

```
                        { End= -1;
                          rtf_put(2, &End, sizeof(int));}
                      SampleNumber= 0;
                      while((n= rtf_get(3, &ary, sizeof(int))) > 0)
                        ;
                      break;
                default:
                      return -EINVAL;
          }
    }
    if (err != 0) {
          return -EINVAL;
    }
    return 0;
}

int init_module(void)
{
    rtf_create(1, sizeof(msg));
    rtf_create(2, sizeof(int)*NumSamples);
    rtf_create(3, sizeof(int)*NumSamples);
    rtf_create(4, sizeof(data));
    rtf_create_handler(1, &my_handler);
    InstallPWMControl(2, 100, TypePWM);
    //InstallOutputs(8);
    //WriteOutput(0x0ff);
    InstallEncoder(1);
    InstallTimerInt(Interrupt);
    return 0;
}


void cleanup_module(void)
{
    rtf_destroy(1);
    rtf_destroy(2);
    rtf_destroy(3);
    rtf_destroy(4);
    RemoveTimerInt();
    StopUpp();
}
```

```
/* option.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <linux/errno.h>
#include "MyFifo.h"

void main(void)

{
int choice;

system("clear");

if ((ctl= open("/dev/rtf1", O_WRONLY))<0)
  {
    fprintf(stderr, "Error opening /dev/rtf1\n");
      exit(1);
  }

if ((dcs= open("/dev/rtf2", O_RDONLY))<0)
  {
    fprintf(stderr, "Error opening /dev/rtf2\n");
      exit(1);
  }

if ((fdb= open("/dev/rtf3", O_WRONLY))<0)
  {
    fprintf(stderr, "Error opening /dev/rtf3\n");
      exit(1);
  }

if ((dta= open("/dev/rtf4", O_WRONLY))<0)
  {
    fprintf(stderr, "Error opening /dev/rtf4\n");
      exit(1);
  }

do
  {
   printf("    HYDRAULIC ACTUATOR PROGRAM  \n\n");
   printf("(1) Manual Control \n");
   printf("(2) Move Ram to bottom, set position to zero \n");
   printf("(3) STEP Input \n");
   printf("(4) Multipule STEP Input \n");
   printf("(5) RAMP Input \n");
   printf("(6) Sine Wave Input \n");
   printf("(7) Path Input \n");
   printf("(8) Set Pulse \n");
   printf("(9) Quit \n");

   switch(choice= getchar())
    {
      case'1':
      Manual();
```

```
      break;

      case'2':
      Initialise();
      break;

      case'3':
      StepInput();
      break;

      case'4':
      MultiStep();
      break;

      case'5':
      RampInput();
      break;

      case'6':
      Sinewave();
      break;

      case'7':
      FollowPath();
      break;

      case'8':
      SetPulse();
      break;

      case'9':
      break;

        default:
        printf("Choose 1 to 9 only\n");
        break;
    }
   }
while( choice != '9');
return;
}
```

```
/* function.c */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <linux/errno.h>
#include <math.h>
#include <string.h>
#include "MyFifo.h"
#include "getkey.h"

/*-------------------------------------------------------------------------
*/
void Send_error(void)

{
          fprintf(stderr, "Can't send a command to RT-task\n");
          exit(1);
}
/*-------------------------------------------------------------------------
*/
void SetPulse(void)

{
  float inputsignal;

  system("clear");
  getchar();
  printf("Enter percentage of PWM pulse (0- 100)\n");
  scanf("%f", &inputsignal);
  data.Signal= inputsignal;
  if (write(dta, &data, sizeof(data)) < 0) { Send_error(); }

  getchar();
  printf("Press Enter to start ram NOW\n");
  if (getchar()== 10)
    {
     msg.Max= MaxSetting;
     msg.command= SLOW;
     if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
    }

  printf("Press Enter to STOP ram NOW\n");
  if (getchar()== 10)
    {
     msg.command= STOP_TASK;
     if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
    }

  Finish();

  printf("Press Enter to Save Data\n");
  if (getchar()== 10) {SaveData();}

return;
```

```
}
/*-------------------------------------------------------------------
*/

void Manual(void)

{
  int Choice[3], rubbish[10000];

  system("clear");
  getchar();
  printf("Press the UP arrow to move up \n");
  printf("Press the DOWN arrow to move down \n");
  printf("Press Space Bar to close the valve \n");
  printf("Press 'e' to quit \n");
do
  {
      get_key(Choice);

      if (Choice[0]== 'e')
       data.Signal= 0.0;

      if((data.Signal< MaxSetting)&&(Choice[0]== 27 )&& (Choice[2]== 65
))
       data.Signal= data.Signal+ 10.0;

      if((data.Signal> -(MaxSetting)) &&(Choice[0]== 27 )&&
(Choice[2]== 66 ))
       data.Signal= data.Signal- 10.0;

      if (Choice[0]== 32)
       data.Signal= 0.0;

      //data.PosG=0;
      //data.IntG=0;
      //data.DerG=0;
       msg.Max= MaxSetting;
       msg.command= SLOW;

      if (write(dta, &data, sizeof(data)) < 0)
      { printf("dta\n");
        Send_error(); }
      if (write(ctl, &msg, sizeof(msg)) < 0)
      { printf("msg\n");
        Send_error(); }

  }
while ((Choice[0]==32)||(Choice[0]== 27));

msg.command= STOP_TASK;
printf("STOP\n");
if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
read(dcs, rubbish, sizeof(rubbish));
return;

}
```

```
/*-------------------------------------------------------------------------*/
void Input(void)

{
  system("clear");
  printf("Enter starting position.  -  ");
  scanf("%d", &StartPosition);

  printf("Enter end position.  -  ");
  scanf("%d", &EndPosition);

  printf("Enter Position Gain (must be +ve).  -  ");
  scanf("%f", &InputPosG);

  printf("Enter Derivative gain (must be +ve).  -  ");
  scanf("%f", &InputDerG);

  printf("Enter Integral Gain (must be +ve).  -  ");
  scanf("%f", &InputIntG);

  data.Signal= 0;

  printf("finishedInput\n");
  return;

}

/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void StepInput(void)

{
int i;

 Input();
 getchar();
  printf("Press Enter to move ram to initial position\n");
  if (getchar()== 10)
    {
      move();
    }
  for (i=0; i< NumSamples; i++)
    {
      ary.SetPos= EndPosition;
      SetPosition[i]= EndPosition;
      if (write(fdb, &ary, sizeof(ary)) < 0) { Send_error(); }
    }
  printf("generated step array\n");

  data.DerG= InputDerG;
  data.PosG= InputPosG;
  data.IntG= InputIntG;
  msg.Max= MaxSetting;

  if (write(dta, &data, sizeof(data)) < 0) { Send_error(); }

  printf("Press Enter to step ram NOW");
  if (getchar()== 10)
    {
     msg.command= START_TASK;
     if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
    }
  printf("step started\n");

  Finish();

  printf("Press Enter to Save Data\n");
  if (getchar()== 10) {SaveData();}

return;
}
/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void MultiStep(void)

{
  int n, step, dist, StayNum, i=0, TotalNum=0;

  Input();
  getchar();
  data.DerG= InputDerG;
  data.PosG= InputPosG;
  data.IntG= InputIntG;
  msg.Max= MaxSetting;
  if (write(dta, &data, sizeof(data)) < 0) { Send_error(); }

  printf("Enter number of steps to perform. - ");
  scanf("%d", &step);

  for(n=1; n<=step; n++)
    { printf("Enter the absolute distance of the %dst step. - \n", n);
      scanf("%d", &dist);
      printf("Enter the number of time samples to stay at that step
(each sampling is %f. - \n", SampleTime);
      scanf("%d", &StayNum);

      TotalNum= StayNum+ TotalNum;
      while(i< TotalNum && i< NumSamples)
      {
        ary.SetPos= dist;
        SetPosition[i]= dist;
        if (write(fdb, &ary, sizeof(ary)) < 0) { Send_error(); }
        ++i;
      }
    }
  getchar();
  printf("Press Enter to step ram NOW");
  if (getchar()== 10)
    {
     msg.command= START_TASK;
     if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
    }
  printf("step started\n");

  Finish();
  printf("Press Enter to Save Data\n");
  if (getchar()== 10) {SaveData();}

return;
}

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void RampInput(void)

{
int i;
float increment, RampTime;
//float SampleTime= InterPeriod* 3.25e-6;

  Input();

  printf("Enter Ramping Time (seconds).  -  ");
  scanf("%f", &RampTime);
  getchar();
  printf("Press Enter to move ram to initial position\n");
  if (getchar()== 10)
    {
      move();
    }

  increment= (EndPosition-StartPosition)*SampleTime/(RampTime);

  for (i=0; i< NumSamples; i++)
    {
      ary.SetPos= StartPosition+ increment*i;
        if (((increment< 0)&& (ary.SetPos< EndPosition))
|| ((increment> 0)&& (ary.SetPos> EndPosition)))
           {ary.SetPos= EndPosition;}
      SetPosition[i]= ary.SetPos;
      if (write(fdb, &ary, sizeof(ary)) < 0) { Send_error(); }
    }
  printf("generated ramp array\n");

  data.DerG= InputDerG;
  data.PosG= InputPosG;
  data.IntG= InputIntG;
  msg.Max= MaxSetting;

  if (write(dta, &data, sizeof(data)) < 0) { Send_error(); }

  printf("Press Enter to step ram NOW");
  if (getchar()== 10)
    {
     msg.command= START_TASK;
     if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
    }
  printf("ramp started\n");

  Finish();

  printf("Press Enter to Save Data\n");
  if (getchar()== 10) {SaveData();}

return;
}
/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void Sinewave(void)
{
  float Period;
  int   Sinval, Magnitude,i;

  Input();
  printf("Enter Period (seconds).  -  ");
  scanf("%f", &Period);
  getchar();
  printf("Press Enter to move ram to initial position\n");
  if (getchar()== 10)
    {
      move();
    }

  Magnitude= ceil((EndPosition-StartPosition)/2);
  for (i=0; i< NumSamples; i++)
    {
      Sinval=StartPosition+Magnitude*(1-
cos(i*SampleTime*2*3.14159/Period));
      ary.SetPos= Sinval;
      SetPosition[i]= Sinval;
      if (write(fdb, &ary, sizeof(ary)) < 0) { Send_error(); }
    }
  printf("generated sine array\n");

  data.DerG= InputDerG;
  data.PosG= InputPosG;
  data.IntG= InputIntG;
  msg.Max= MaxSetting;

  if (write(dta, &data, sizeof(data)) < 0) { Send_error(); }

  printf("Press Enter to fluctuate ram NOW");
  if (getchar()== 10)
    {
     msg.command= START_TASK;
     if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
    }
  printf("ramp started\n");

  Finish();

  printf("Press Enter to Save Data\n");
  if (getchar()== 10) {SaveData();}

  return;
}

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void move(void)

{
int i;
  printf("in move\n");
  for (i=0; i< NumSamples; i++)
    {
      ary.SetPos= StartPosition;
      if (write(fdb, &ary, sizeof(ary)) < 0) { Send_error(); }
    }
  printf("outof move forloop\n");

  data.PosG= 0.5;
  data.IntG= 0.05;
  data.DerG= 0.0;
  msg.Max= 20;

  if (write(dta, &data, sizeof(data)) < 0) { Send_error(); }

  msg.command= START_TASK;
  if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
  printf("%d %d %f %f %f\n",msg.command, ary.SetPos, data.PosG,
data.DerG, data.IntG);
  printf("move started\n");
  Finish();
return;
}

/*-------------------------------------------------------------------*/
void FollowPath(void)

{
  int count=0, i;
  int Time[NumSamples];
  char line[20];
  char file[15];
  char filename[size];
  char *Datafile="/home/jly16/Pathdata/";
  FILE *fp;

printf("Enter Path filename..ie:path.txt) .  ");
scanf("%s", &file);
strcpy(filename, Datafile);
strncat(filename, file, 15);

if ((fp=fopen(filename, "r"))== NULL)
  {printf("Could not find file. Press ENTER to exit.\n");
    if (getchar()== 10)
      return;
  }

 do {
   fgets(line, 20, fp);
   sscanf(line, "%d\t%d\n", &Time[count], &SetPosition[count]);
   count++;
 } while (!feof(fp));
```

```
  printf("Enter Position Gain (must be +ve).  -  ");
  scanf("%f", &InputPosG);
  printf("Enter Derivative gain (must be +ve).  -  ");
  scanf("%f", &InputDerG);
  printf("Enter Integral Gain (must be +ve).  -  ");
  scanf("%f", &InputIntG);

  StartPosition= SetPosition[0];
  printf("Press Enter to move ram to initial position\n");
  if (getchar()== 10)
    {
      move();
    }

  for (i=0; i< count; i++)
    {
      ary.SetPos= SetPosition[i];
      if (write(fdb, &ary, sizeof(ary))< 0) { Send_error(); }
    }

  data.DerG= InputDerG;
  data.PosG= InputPosG;
  data.IntG= InputIntG;
  msg.Max= MaxSetting;

  if (write(dta, &data, sizeof(data)) < 0) { Send_error(); }

  printf("Press Enter to let ram follow path");
  if (getchar()== 10)
    {
     msg.command= START_TASK;
     if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
    }
  printf("ramp started\n");

  Finish();

  printf("Press Enter to Save Data\n");
  if (getchar()== 10) {SaveData();}

  return;
}


/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void Initialise(void)

{
  data.Signal= -20.0;
  msg.Max= 20;
  msg.command= SLOW;
  if (write(dta, &data, sizeof(data)) < 0) { Send_error(); }
  if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
  Finish();
  msg.command= INITIALISED;
  if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
printf("Initialisation Completed\n");
printf("Press 'ENTER' to return to main menu.\n");
  getchar();
  if (getchar()== 10)
    system("clear");
    return;
}

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------
*/
void Finish(void)

{
 int buf[20000], n;
 int i, j=0;
 int Max= 0, Min= 10000;
 float Total=0;

 Number= 0;
 while  ((n= read(dcs, buf, sizeof(buf))) > 0)
 {
  n= n/sizeof(int);
    for(i= 0; i< n; i++)
      {
      if (j%2 !=0)
        {
          ValveSetting[Number]= buf[i];
          if (Number%100 ==0)
            { system("clear");
            printf("%d\t%d\t%d\n", Number, ValveSetting[Number],
ReadPosition[Number]);}
          /*if (j >1)
            {
            if (ValveSetting[Number] > Max) Max= ValveSetting[Number];
            if (ValveSetting[Number] < Min) Min= ValveSetting[Number];
            Total+= ValveSetting[Number];
            ++Number;
            }*/
          if (j >1)
            {++Number;}
        }

       else
        { ReadPosition[Number]= buf[i];
          if (/*(Number >= TerminalCount &&
            abs(ReadPosition[Number]- ReadPosition[Number-
(TerminalCount/2)                ])  <= 10
                && abs(ReadPosition[Number]- ReadPosition[Number-
TerminalCount]                ) <= 10)*/ ReadPosition[Number]==-1  ||
Number >= NumSamples-10 )
             {
            msg.command= STOP_TASK;
            if (write(ctl, &msg, sizeof(msg)) < 0) { Send_error(); }
            read(dcs, buf, sizeof(buf)); /*clear the buffer.*/
            printf("\nMax%d\nMin%d\nAverage%f\n", Max, Min,
Total/Number);
            return;
             }
        }//close else
      ++j;
      }//close for
    //printf("2n: %d\n", n);
if (ReadPosition[Number]== -1)
        {return;}
 }
```

```
printf("before return\n");
 return;
}

/*-------------------------------------------------------------------------
*/

void SaveData(void)

 {
   int count; //Number;
   //float valve[5000], position[5000];
   char file[15];
   char filename[size];
   char *OutputFile= "/home/jly16/Ramdata/";
   FILE *cp;

   printf("Save data ? (y/n)\n");
   if (getchar() !='y')
     return;
   printf("Enter Filename ..");
   scanf("%s", file);
   strcpy(filename, OutputFile);
   strncat(filename, file, 15);

   if((cp= fopen(filename, "wt"))== NULL)
     {  printf("Could not open file\n");
        return;
     }

   for (count=1; count< Number; count++)
     {fprintf(cp, "%d\t%d\t%d\t%d\n", count, ReadPosition[count],
SetPosition[count]                    ,ValveSetting[count]);
     }

   fclose(cp);
   printf("data stored Press 'Enter' to continue.\n");
   if (getchar()== 10)
     return;
 }
```

```
/* getkey.h */

#include <stdio.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <unistd.h>


static char get_key(ch)
int ch[3];
{
  static struct termios my_kb, orig_kb;

        tcgetattr(STDIN_FILENO, &orig_kb);
        my_kb = orig_kb;
        my_kb.c_lflag &= ~(ECHO | ISIG | ICANON);
        my_kb.c_cc[4] = 1;

     tcsetattr(STDIN_FILENO, TCSAFLUSH,  &my_kb);
     ch[0]= getchar();
     if (ch[0]== 27)
       { ch[1]= getchar();
         ch[2]= getchar();
       }
     tcsetattr(STDIN_FILENO, TCSAFLUSH,  &orig_kb);
     return;

}
```

```
function plotdata %Matlab file for graph plotting

file=('f:\Ramdata\');

data=input('Filename to retrive. in inverted commas. >>');  %gets
filename
filename=strcat(file,data);
sample=input('Sampling time in seconds. >>');
%combine to get the whole path

result= dlmread(filename,'\t');
%read the data from the file

size= length(result)
time= result(:,1).*sample;
actual= result(:,2).*0.1;
desired= result(:,3).*0.1;

PWM= result(:,4);

velocity= 0.1*(actual(size)- actual(ceil(size/2)))/(time(size)-
time(ceil(size/2)))

figure (1)
error= actual- desired;   %calculates error
plot(time, error,'b');     %plots the error curve
line([0 max(time)],[0,0],'color','r');
%the zero line is included for clarity
xlabel('Time,seconds');
ylabel('error, mm');
title(strcat('Error Plot', data));
zoom yon;                          %enable the zoom function
% click the left mouse button to zoom in, right to zoom out

figure (2)
plot(time,desired,'r-'); %plot the input signal
hold on
plot(time, actual,'b');
%axis([min(time) max(time) 190 610])
%plot the system respond curve
xlabel('Time, seconds');
ylabel('Position, mm ');
title(strcat('Output/ Respond Comparison', data));

zoom on;

%title('Output/ Respond Comparison');



figure (3)

plot(time, PWM,'b');

xlabel('Time, seconds');
```

```
ylabel('PWM, (%) ');

title(strcat('PWM/ Respond Comparison', data));


hold off
zoom yon;
```

APPENDIX D- CONTROL SOFTWARE FOR UNIMATE 2000B

```
#makefile

all: visual RT.o option

# the path to the rt-linux kernel
RTLINUX = ../../../linux
INCLUDE = ${RTLINUX}/include/linux

CFLAGS = -O2 -Wall

visual: visual.c kinematics.h rtshare.h
      gcc -I${INCLUDE} ${CFLAGS} -o  visual visual.c  -lm

RT.o: RT.c rtshare.h
      gcc -I${INCLUDE} ${CFLAGS}  -D__KERNEL__ -D__RT__ -c  RT.c

option: option.c kinematics.h rtshare.h
      gcc -I${INCLUDE} ${CFLAGS}  -o  option option.c  -lm -lslang

clean:
      rm -f   function.o option* option.o option.c~  function.c~
makefile~ Non_rt.h~ rtshare.h~ RT.o RT.c~
```

```
/* rtshare.h */

#define Base_Address  ((38 * 0x100000))  //RAM between 38-39Mb is
reserved for shared memory

#define  hold              0
#define  follow_track      1
#define  manual_actuate    2
#define  tracking_actuate  3

#define  NumSamples        250  /*Number of Samples the controller goes
through before resetting to the hold position, also mean that each
movement has 2000 Samples where each sample consist of 6 interrupts*/
#define  MaxSetting        100  //Maximun setting for PWM
#define  Second_per_Pulse  3.25e-6  //clock rate of UPP in seconds
#define  InterPeriod       1846 //308 //InterPeriod x Second_per_Pulse=
1 ms (interrupt period)
#define  SampleTime        (Second_per_Pulse*InterPeriod)
#define  No_of_Actuator    6
#define  Inter_per_Visual  12  //no. of interrupts for each visual
printout into program "visual"

#define  No_of_Deriv  4
#define  IntMax       50

int Actuator_No, SampleNumber, bit;
float holding[No_of_Actuator];
typedef float scalar;
typedef scalar OnebySix[6];  //1x6 matrix
typedef scalar FourbyFour[4][4];  //4x4 matrix

const float metric_gradient[]={-0.2617, -0.0295, 0.01432, 0.06057,
0.08877, -0.09595},
metric_constant[]={2041, 120.277, -27, -110, -169, 243.8};
  /*for the conversion of binary into degress or mm for each joint*/

const float joint_dep[]= {0, 0, 0, 0, 1, 0};  //coefficient for joint
position dependency

const int limit_pos[No_of_Actuator]= {2041, 110, 30, 110, 174,180};
/* maximum limit of joints*/
const int limit_neg[No_of_Actuator]= {976, -110, -27, -110, -169,-180};
/* minimum limit of joints*/

const int pminim[No_of_Actuator]= {26, 22, 14, 25, 21, 20},
  nminim[No_of_Actuator]= {17, 18, 26, 28, 16, 20};
/*minimum value of PWM to start moving the rams*/
```

```
/*-----------------------------fifo's---------------------------------*/

struct my_msg_struct {
  int command;
} msg;  //control message send from user interface to RT.o

struct my_signal_struct {
  int PWM[No_of_Actuator];
} Signal;  //PWM value send through fifo when in "manual" control mode

struct my_visual_number {
  float Points[No_of_Actuator], PWM[No_of_Actuator];
  int msg, SampleNumber;
  float InterSeconds;
} Visual;  //structure of data send to "visual" program for user to see


/*-------------------------shared memory-------------------------------*/

typedef struct  {
  float OutPut[No_of_Actuator][No_of_Deriv],
    PosG[No_of_Actuator][No_of_Deriv],  //proportional gain
    DerG[No_of_Actuator],  //derivative gain
    IntG[No_of_Actuator],  //integral gain
    filter[No_of_Deriv];   //coefficients to smooth out the derivative
} HOLD_COEFF;  //coefficient for difference equation when robot is in
"hold" mode


typedef struct  {
  float OutPut[No_of_Actuator][No_of_Deriv],
    PosG[No_of_Actuator][No_of_Deriv],
    DerG[No_of_Actuator],
    IntG[No_of_Actuator],
    filter[No_of_Deriv];
} TRACK_COEFF;  //coefficient for difference equation when robot is NOT
in "hold" mode

typedef struct  {
  float Points[No_of_Actuator][NumSamples];
} DESIRED;  //desired path for robot defined by user interface

typedef struct  {
  float Points[No_of_Actuator][NumSamples],
PWM[No_of_Actuator][NumSamples];
} ACTUAL;  //the actual path performed by robot

typedef struct  {
  float PWM[No_of_Actuator][NumSamples];
} ACTUATION;  //the PWM signals send to the ram valves

typedef struct  {
  unsigned char inuse;
  float Points[NumSamples];
} CURRENT;  //current positions of the rams at start of each movement
```

```
typedef struct  {

  HOLD_COEFF    Hold_Coefficient;
  TRACK_COEFF   Track_Coefficient;
  DESIRED       Desired_Tracking;
  ACTUAL        Actual_Tracking;
  ACTUATION     Actuate_Tracking;
  CURRENT       Current_Position;

} Shared_Mem;   //summing all structures together for shared memory
allocation
```

```
/* RT.c */

#define MODULE
#include <math.h>
#include <unistd.h>
#include <linux/module.h>
#include <linux/errno.h>
//#include <time.h>
#include </usr/src/rtl/include/rtl_sched.h>
#include </usr/src/rtl/include/rtl_fifo.h>
#include "PWM.H"
#include "rtshare.h"
#include "Upp.h"
#define  AreaRatio  0.44

extern void * vremap(unsigned long offset, unsigned long size);
extern void vfree(void * addr);

Shared_Mem   *ptr;
struct  Coefficient{
  float OutPut[No_of_Actuator][No_of_Deriv],
    PosG[No_of_Actuator][No_of_Deriv],
    DerG[No_of_Actuator],
    IntG[No_of_Actuator],
  filter[No_of_Deriv];
} coeff;
DESIRED      desired;
ACTUAL       actual;
ACTUATION    actuation;
CURRENT      current;

/*-------------------------------------------------------------------*/
int ReadInput16(void)  //read input from encoder
{
  int Result;

  Result= UPPReadPort(0x0230, 1)+ UPPReadPort(0x0230, 2)*256;
  return Result;
}
/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
float GrayScale(int raw_reading)  //converts grayscale to binary
{
  int i,gray_convert_integer=0, array_binary[14]= {0}, array_gray[14]=
{0}, a=8192;  //0010 0000 0000 0000
  float temp;

  for (i= 0; i< 14; ++i)
    {
      if ((raw_reading & a) == 0)
      array_gray[i]= 0;
      else
      array_gray[i]= 1;

      array_binary[i]= array_gray[i]^array_binary[i-1];
      gray_convert_integer += array_binary[i]* a;
      a >>= 1;
    }
  /*converts binary to degrees or mm for each joint*/
  temp= gray_convert_integer* metric_gradient[Actuator_No]+
metric_constant[Actuator_No];

return temp;

}

/*-------------------------------------------------------------------*/
void ValveSignal(float PWM)  //sends the PWM signal to the ram valves
{
  if ( PWM> MaxSetting )
      PWM= MaxSetting;

  if (PWM< -MaxSetting)
      PWM= -MaxSetting;

SetSinglePWMValue(Actuator_No, PWM);
actual.PWM[Actuator_No][SampleNumber]= PWM;
return;
}

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
float ErrorCalc(float SetPos)  //Calculates PWM signal according to a
differential equation
{
  float Deriv=0, Error_Sum=0, Output_Sum=0;
  int i;
  static float e[No_of_Actuator][No_of_Deriv+1],
u[No_of_Actuator][No_of_Deriv+1], Integ[No_of_Actuator];

  e[Actuator_No][No_of_Deriv-1]=
actual.Points[Actuator_No][SampleNumber]- SetPos;
  for (i=0; i<No_of_Deriv; ++i)
    {
      Deriv += coeff.filter[i]*e[Actuator_No][i];
      Error_Sum += coeff.PosG[Actuator_No][i]*e[Actuator_No][i];
      Output_Sum += coeff.OutPut[Actuator_No][i]*u[Actuator_No][i];
      e[Actuator_No][i]= e[Actuator_No][i+1];
      u[Actuator_No][i]= u[Actuator_No][i+1];
    }
  Deriv =Deriv/ (6*SampleTime);

  Integ[Actuator_No]= Integ[Actuator_No]+
coeff.IntG[Actuator_No]*e[No_of_Deriv -1][Actuator_No]*SampleTime;

    if (Integ[Actuator_No]> IntMax)
      Integ[Actuator_No]= IntMax;
    if (Integ[Actuator_No]< -IntMax)
    Integ[Actuator_No]= -IntMax;
    /*limits the integral gain*/

  u[Actuator_No][No_of_Deriv -2]= Error_Sum + Output_Sum +
Integ[Actuator_No]+ (coeff.DerG[Actuator_No]*Deriv);

  if (u[Actuator_No][No_of_Deriv -2]> 0)
    u[Actuator_No][No_of_Deriv -2]= (1-
(pminim[Actuator_No]/MaxSetting))*u[Actuator_No][No_of_Deriv -2]+
pminim[Actuator_No];
      else
      u[Actuator_No][No_of_Deriv -2]= (1-
(nminim[Actuator_No]/MaxSetting))*u[Actuator_No][No_of_Deriv -2]-
nminim[Actuator_No];

  return u[Actuator_No][No_of_Deriv -2];
}
/*-------------------------------------------------------------------
```

```
/*-------------------------------------------------------------------*/
void Interrupt(void)  //Interrupt function that executes at each
interrupt
{
  int  i, input14bit, yes=1;
  float PassingSignal=0;
  static long begin, end;

  end= rt_get_time();
  Visual.InterSeconds= (end-begin)*1000000/RT_TICKS_PER_SEC;
//converts computer ticks to microseconds
  begin= rt_get_time();

  input14bit= ReadInput16() & 32767;  //0111fff masking to read first
14 bit
  actual.Points[Actuator_No][SampleNumber]= GrayScale(input14bit);

  if ( Actuator_No== 4 || Actuator_No== 5)
    {
      actual.Points[Actuator_No][SampleNumber]=
actual.Points[Actuator_No][SampleNumber]+
joint_dep[Actuator_No]*actual.Points[Actuator_No-1][SampleNumber];
    }  /*orientation of joint 5 (yaw) is dependant on joint 4 (bend),
so is joint6 (swivel) on joint 5 (yaw)*/

  if ( SampleNumber==1 && Actuator_No==5 )
    {
      for(i=0; i< No_of_Actuator; ++i)
      current.Points[i]= actual.Points[i][SampleNumber];
      if (0== ptr-> Current_Position.inuse)
      memcpy(&ptr->Current_Position, &current, sizeof(current));
    }  //copies the position of the joints to shared memory at the
start of each movement

  switch(msg.command)  //conditional selector according to message from
fifo 0 (msg)
    {
      case hold:
      PassingSignal= ErrorCalc(holding[Actuator_No]);
      ValveSignal(PassingSignal);
      break;

      case follow_track:
      PassingSignal=
ErrorCalc(desired.Points[Actuator_No][SampleNumber]);
      ValveSignal(PassingSignal);
      if (SampleNumber>= NumSamples-1)
        {
          memcpy(&ptr->Actual_Tracking, &actual, sizeof(actual));
          rtf_put(2, &yes, sizeof(yes));  //tells user interface to
read from shared memnory
        }  //copies the actual path done by robot to shared memory
      break;

      case manual_actuate:
      ValveSignal(Signal.PWM[Actuator_No]);
      break;
```

```
case tracking_actuate:
      ValveSignal(actuation.PWM[Actuator_No][SampleNumber]);
      if (SampleNumber>= NumSamples-1)
        {
          memcpy(&ptr->Actual_Tracking, &actual, sizeof(actual));
          rtf_put(2, &yes, sizeof(yes));
        }  //copies the actual path done by robot to shared memory
      break;

      default:
      ValveSignal(0);
      break;

    }  //end of conditinal selector

  /* (information send back to the visual screen) */
  if (SampleNumber% Inter_per_Visual== 0)
    {
      Visual.Points[Actuator_No]=
actual.Points[Actuator_No][SampleNumber];
      Visual.PWM[Actuator_No]= actual.PWM[Actuator_No][SampleNumber];

      if (Actuator_No== 5)
      {
        Visual.msg= msg.command;
        Visual.SampleNumber= SampleNumber;
        rtf_put(3, &Visual, sizeof(Visual));  /*send to fifo 3 (vsl),
to be send to "visual program*/
      }
    }

  /* (increment counter) */
  ++Actuator_No;
  if (Actuator_No>= No_of_Actuator)
    {
      Actuator_No= 0;
      ++SampleNumber;
    }  //rotate between the 6 joints and increments SampleNumber after
each rotation

  WriteOutput(Actuator_No);  //turn encoder on for reading

  /* (change control mode back to 'hold' after 'follow_track' &
'tracking_actuate') */
  if (SampleNumber>= NumSamples)
    {
      if (msg.command!= manual_actuate)
      {
        for (i=0; i< No_of_Actuator; ++i)
          holding[i]= actual.Points[i][SampleNumber-1];
        msg.command=hold;
      }  //return to "hold" control mode after movement
      SampleNumber= 0;  //reset SampleNumber after each movement
    }

  //begin= clock();
return;
```

```
}

/*-------------------------------------------------------------------*/
int my_handler(unsigned int fifo)  //reads from fifo 0 (msg) to for
control mode instructions
{
int err,i;

  while ((err = rtf_get(0, &msg, sizeof(msg))) == sizeof(msg) ) {
    switch (msg.command) {

      case hold:
      memcpy(&coeff, &ptr-> Hold_Coefficient, sizeof(coeff));/*copies
hold coefficients into controller*/
      if ( SampleNumber != 0)
        for (i=0; i< No_of_Actuator; ++i)
        {holding[i]= actual.Points[i][SampleNumber-1];}
      else
        for (i=0; i< No_of_Actuator; ++i)
        {holding[i]= actual.Points[i][NumSamples-1];}
      /*holds at coordanates of previous Sample*/
      SampleNumber=0;  //resets SampleNumber to start movement
      break;

      case follow_track:
      memcpy(&desired, &ptr-> Desired_Tracking,
sizeof(desired));/*copies desired tracking coordinates from shared
memory*/
      memcpy(&coeff, &ptr-> Track_Coefficient, sizeof(coeff));
      SampleNumber=0;  //resets SampleNumber to start movement
      break;

      case manual_actuate:
      rtf_get(1, &Signal, sizeof(Signal));  /*gets value of PWM signal
in "manual" control mode*/
      break;

      case tracking_actuate:
      memcpy(&actuation, &ptr-> Actuate_Tracking,
sizeof(actuation));/*copies the PWM signals from shared memory*/
      SampleNumber=0;
      Actuator_No= 0;
      break;

      default:
      return -EINVAL;
    }
  }
  if (err != 0) {
    return -EINVAL;
  }
  return 0;
}

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
int init_module(void)  //events executed once RT.o is inserted with
"insmod"
{
int PwmType[]= {1, 1, 1, 1, 1, 1};

 ptr= (Shared_Mem *) vremap(Base_Address, sizeof(Shared_Mem));  //ptr
points to RAM allocated for shared memory
  rtf_create(0, sizeof(msg));
  rtf_create(1, sizeof(Signal));
  rtf_create(2, sizeof(int));
  rtf_create(3, sizeof(Visual)*20);
  rtf_create_handler(0, &my_handler);
  InstallPWMControl(No_of_Actuator, 100, PwmType);
  bit= InstallOutputs(4);  //4 bits installed to
  //UPPPortDir(0x232,2, 0xff);
  SampleNumber= 0;
  Actuator_No= 0;
  InstallTimerInt(Interrupt);
  StartTimerInt(InterPeriod);
  return 0;

}


void cleanup_module(void)  //events executed once RT.o is removed with
"rmmod"
{
  StopTimerInt();
  vfree(ptr);  //free pointer to shared memory
  rtf_destroy(0);
  rtf_destroy(1);
  rtf_destroy(2);
  rtf_destroy(3);
  RemoveTimerInt();
  StopUpp();
}
```

```
/*option.c*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <linux/errno.h>
#include <ctype.h>
#include <slang/slang.h>
#include <sys/mman.h>
#include <math.h>
#include <string.h>
#include "rtshare.h"
#include "kinematics.h"

int cmd, m_act, linux_rd, vsl;
Shared_Mem   *ptr;
HOLD_COEFF   hold_coeff={
  { {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0} },
  { {0,0,0,10}, {0,0,0,10}, {0,0,0,11}, {0,0,0,0.5}, {0,0,0,1},
{0,0,0,2} },
  {0,0,0,0,0,0},
  {200, 29, 100, 3, 14, 80},
  {11, -18, 9, -2}
};  //control coefficients in hold mode
TRACK_COEFF  track_coeff={
  { {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0} },
  { {0,0,0,20}, {0,0,0,20}, {0,0,0,31}, {0,0,0,3}, {0,0,0,2}, {0,0,0,4}
},
  {0,0,0,0,0,0},
  {200, 29, 100, 3, 14, 80},
  {11, -18, 9, -2}
};  //control coefficients in follow_track mode
DESIRED      desired;
ACTUAL       actual;
ACTUATION    actuation;
CURRENT      current;



/*-------------------------------------------------------------------*/
void Send_error(void)  //eror shown when fifo not functioning

{
          fprintf(stderr, "Can't send a command to RT-task\n");
          exit(1);
}

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void Manual(void)  //UI for manual control of robot by keyboard

{
  int i, j=0;
  char ch='\0', decision;


  system("clear");
  getchar();
  printf("Press number 1-9 for size of PWM increment\n\n");
  printf("Press 'a' or 'z' to move '+' or '-' for In/Out\n");
  printf("Press 's' or 'x' to move '+' or '-' for Rotation\n");
  printf("Press 'd' or 'c' to move '+' or '-' for Up/Down\n");
  printf("Press 'f' or 'v' to move '+' or '-' for Bend\n");
  printf("Press 'g' or 'b' to move '+' or '-' for Yaw\n");
  printf("Press 'h' or 'n' to move '+' or '-' for Swivel\n");
  printf("Press Space Bar to stop all axis \n");
  printf("Press 'q' to quit \n");

  SLang_init_tty(-1, 0, 1); //initialise SLang mode (direct reading
from keyboard)

  do
    {
      ch= SLang_getkey();

      if (isdigit(ch) !=0)
      {
        j=isdigit(ch)? ch:'0';
        j=j-48;
      }/*if it is a digit, convert 'ch' into a digit, this would be the
incremental value defined by user*/

    decision= isalpha(ch)? ch:'0';  //decision= 'ch', if ch is an
alphabet
    if (decision== 'a')
      Signal.PWM[0]+= j;  //increment by j, incremental value given by
user
    if (decision== 'z')
      Signal.PWM[0]= Signal.PWM[0]- j;

    if (decision== 's')
      Signal.PWM[1]+= j;
    if (decision== 'x')
      Signal.PWM[1]= Signal.PWM[1]- j;

    if (decision== 'd')
      Signal.PWM[2]+= j;
    if (decision== 'c')
      Signal.PWM[2]= Signal.PWM[2]- j;

    if (decision== 'f')
      Signal.PWM[3]+= j;
    if (decision== 'v')
      Signal.PWM[3]= Signal.PWM[3]- j;
```

```
    if (decision== 'g')
      Signal.PWM[4]+= j;
    if (decision== 'b')
      Signal.PWM[4]= Signal.PWM[4]- j;

    if (decision== 'h')
      Signal.PWM[5]+= j;
    if (decision== 'n')
      Signal.PWM[5]= Signal.PWM[5]- j;

    if (isspace(ch) !=0 || decision=='q')
      for(i=0; i< No_of_Actuator; ++i)
        {
          Signal.PWM[i]= 0;
        }

    for(i=0; i< No_of_Actuator; ++i)
    printf("%d\t", Signal.PWM[i]);
    printf("\n");

    if (write(m_act, &Signal, sizeof(Signal)) < 0) { Send_error(); }

    msg.command= manual_actuate;
    if (write(cmd, &msg, sizeof(msg)) < 0) { Send_error(); }

    } while (ch !='q');  //q for quiting the "manual" control mode

  SLang_reset_tty(); //reset to normal mode
  msg.command= hold;  //reset back to "hold" control mode
printf("STOP\n");
if (write(cmd, &msg, sizeof(msg)) < 0) { Send_error(); }
return;

}

/*-------------------------------------------------------------------*/

void MultiStepInput(void)  //steps each joint to desired position
{
  int i,j;

  for (i=0; i< No_of_Actuator; ++i)
    { printf("Enter the position of Actuator%d to step to\n", i);
    scanf("%f", &desired.Points[i][0]);  //enter 6 positions to step to
      for (j=0; j <NumSamples; ++j)
      desired.Points[i][j]= desired.Points[i][0]; //generate each step
path
    }

  memcpy(&ptr-> Desired_Tracking, &desired,  sizeof(desired));
  memcpy(&ptr-> Track_Coefficient, &track_coeff,  sizeof(track_coeff));

  msg.command= follow_track;
  if (write(cmd, &msg, sizeof(msg)) < 0) { Send_error(); }

  return;
}
```

```
/*-------------------------------------------------------------------*/
void SingleStepInput(void)  //steps only 1 selected joint to desired
position
{
  int i,j, selected_actuator=0;

  ptr-> Current_Position.inuse= 1;
  memcpy(&current, &ptr->Current_Position , sizeof(current));
  ptr-> Current_Position.inuse= 0;

  printf("Actuator Position are: ");
  for (i=0; i< No_of_Actuator; ++i)
    printf("%5.1f\t", current.Points[i]);

  do {
      printf("\nSelect an actuator to step 0-5:\t");
      scanf("%d", &selected_actuator);  //select the actuator for
stepping
      printf("\nEnter position of actuator to step to:\t");
      scanf("%f", &current.Points[selected_actuator]);  //select a
position to step to
     } while(selected_actuator >=No_of_Actuator || selected_actuator
<0);

  for (i=0; i< No_of_Actuator; ++i)
    {
      for (j=0; j <NumSamples; ++j)
      desired.Points[i][j]= current.Points[i];  //generate step path
    }

  memcpy(&ptr-> Desired_Tracking, &desired,  sizeof(desired));
  memcpy(&ptr-> Track_Coefficient, &track_coeff,  sizeof(track_coeff));

  msg.command= follow_track;
  if (write(cmd, &msg, sizeof(msg)) < 0) { Send_error(); }

  return;
}

/*-------------------------------------------------------------------*/
void EndPointInput(void)
{
  int i, j, N=6;
  char *w[6]={"alpha", "beta", "gamma", "x", "y", "z"};
  OnebySix end_point, joint;

  for (i=0; i< N/2; ++i)
    {
      printf("Enter value for %s, the angle around %s axis\n", w[i],
w[N-i-1]);
      scanf("%f", &end_point[i]);  //enter "alpha", "beta", "gamma"
values
    }

 for ( i=N/2; i< N; ++i)
   {
```

```
      printf("Enter value for %s, the distance along the %s axis\n",
w[i], w[i]);
      scanf("%f", &end_point[i]);  //enter "x", "y", "z" values
    }

 Inverse_calc(end_point, joint);  //inverse kinematic function in
kinematics.h

 for (i=0; i< No_of_Actuator; ++i)
    {
      printf("The position of Actuator%d to step to %f\n", i,
joint[i]);
      for (j=0; j <NumSamples; ++j)
      desired.Points[i][j]= joint[i];  //generate tracking path
    }

  memcpy(&ptr-> Desired_Tracking, &desired,  sizeof(desired));
  memcpy(&ptr-> Track_Coefficient, &track_coeff,  sizeof(track_coeff));

  msg.command= follow_track;
  if (write(cmd, &msg, sizeof(msg)) < 0) { Send_error(); }


  return;
}
/*-------------------------------------------------------------------*/
void Waiting(void)  //waiting for end of movement message from fifo"2"
{
  int yes=0;

  printf("Waiting for Movement to Complete ...\n");

     for( ; ; read(linux_rd, &yes, sizeof(yes)))
       {
       if (yes!=0)
         {
           read(linux_rd, &yes, sizeof(yes));
           return;  //exits Waiting() function when able to read from
fifo"2"
         }
       }

return;
}

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void SaveData(void)

 {
   int countdown, countside;
   char file[15];
   char filename[200];
   char *OutputFile= "/home/jly16/RobotData/";  //directory specified
   FILE *cp;

         getchar();
       printf("Save data ? (y/n)\n");
       if (getchar() !='y')
         return;
       printf("Enter Filename .."); //create a new file or overwrite
existing
       memcpy(&actual, &ptr-> Actual_Tracking, sizeof(actual));
       /* copy information out from shared memory */

       scanf("%s", file);
       strcpy(filename, OutputFile);
       strncat(filename, file, 15);  //adding filename to directory

       if((cp= fopen(filename, "wt"))== NULL)
         {  printf("Could not open file\n");
         return;
         }

       for (countdown=0; countdown< NumSamples; ++countdown)
         {
           fprintf(cp, "%d\t", countdown);

           for(countside=0; countside< No_of_Actuator; ++countside)
             {
             fprintf(cp, "%.1f\t%.1f\t%.1f\t",
desired.Points[countside][countdown],
actual.Points[countside][countdown], actual.PWM[countside][countdown]
);
}
           fprintf(cp, "\n");
         }

       fclose(cp);  //close file

   printf("data stored Press 'Enter' to continue.\n");
   if (getchar()== 10)

     return;
 }
/*-------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------*/
void main(void)
{

int choice, fd, visualprogram;

 if ((cmd= open("/dev/rtf0", O_WRONLY))<0)  //open rtf0
  {
    fprintf(stderr, "Error opening /dev/rtf0\n");
      exit(1);
  }

 if ((m_act= open("/dev/rtf1", O_WRONLY))<0)  //open rtf1
  {
    fprintf(stderr, "Error opening /dev/rtf1\n");
      exit(1);
  }

 if ((linux_rd= open("/dev/rtf2", O_RDONLY))<0)  //open rtf2
  {
    fprintf(stderr, "Error opening /dev/rtf2\n");
      exit(1);
  }

 if ((fd= open("/dev/mem", O_RDWR)) < 0)  //open shared memory
    { printf("Can't open /dev/mem\n"); }

ptr= (Shared_Mem *) mmap(0, sizeof(Shared_Mem), PROT_READ |
PROT_WRITE,\
                    MAP_FILE | MAP_SHARED, fd, Base_Address);
/*ptr points to allocated RAM for shared memory*/

  if (MAP_FAILED== ptr)
    { printf("MAP_FAILED\n"); }


visualprogram=fork();
  if (visualprogram== 0)
    if( execl("/usr/X11R6/bin/xterm", "xterm", "-e", "visual", 0) < 0)
      /*starts "visual" program to monitor the robot joints*/

      printf("Can't open program visual\n");
      printf("%d\n", visualprogram);

  memcpy(&ptr-> Hold_Coefficient, &hold_coeff, sizeof(HOLD_COEFF));
  /* copy hold_coeff into shared memory */
do
  {
   system("clear");
   printf("    HYDRAULIC SIX AXIS ROBOT PROGRAM  \n\n");
   printf("(1) Manual Control \n");
   printf("(2) Multipule STEP Input \n");
   printf("(3) SingleRamStep Input \n");
   printf("(4) End Point and Angle Input \n");
   printf("(6) Sine Wave Input \n");
   printf("(7) Path Input \n");
   printf("(8) Set Pulse \n");
```

```
    printf("(9) Quit \n");

    switch(choice= getchar())
     {
       case'1':
       Manual();
       break;

       case'2':
       MultiStepInput();
       Waiting();
         SaveData();
       break;

       case'3':
       SingleStepInput();
       Waiting();
         SaveData();
       break;

       case'4':
       EndPointInput();
       Waiting();
         SaveData();
       break;

       /*case'6':
       Sinewave();
       break;

       case'7':
       FollowPath();
       break;

       case'8':
       SetPulse();
       break;*/

       case'9':
       break;

         default:
         printf("Choose 1 to 9 only\n");
         break;
     }
    }
while( choice != '9');
munmap(ptr, sizeof(Shared_Mem));
return;
}
```

```
/* kinematics.h */

#define pi 3.14159
#define Lb 200
#define Ly 440
#define a2 120.65
#define d1 1066.8
/*-------------------------------------------------------------------*/

void forward_calc (OnebySix raw_relative, FourbyFour trans)
/*calculates the end-effector orientation and position (4x4) from joint
positions (1x6)*/
{
  float c[7], s[7], d3;
  int i;

  d3= raw_relative[0];  /* actuator"0" the prismatic joint(In-Out) is
the 3rd joint in kinematics calculations */

  for (i=1; i<3; ++i)
    {
      s[i]= sin(raw_relative[i]* pi/180);
      c[i]= cos(raw_relative[i]* pi/180);
    }  //converts degrees into radians

  for (i=4; i<7; ++i)
    {
      s[i]= sin(raw_relative[i-1]* pi/180);
      c[i]= cos(raw_relative[i-1]* pi/180);
    }  //converts degrees into radians

  trans[0][0]= -c[6]*c[5]*c[1]*s[2]*c[4]-
c[6]*c[5]*c[1]*c[2]*s[4]+c[6]*s[1]*s[5]+s[6]*c[1]*s[2]*s[4]-
s[6]*c[1]*c[2]*c[4];

  trans[0][1]= s[6]*c[5]*c[1]*s[2]*c[4]+s[6]*c[5]*c[1]*c[2]*s[4]-
s[6]*s[1]*s[5]+c[6]*c[1]*s[2]*s[4]-c[6]*c[1]*c[2]*c[4];

  trans[0][2]= s[5]*c[1]*s[2]*c[4]+s[5]*c[1]*c[2]*s[4]+s[1]*c[5];

  trans[0][3]=
Ly*s[5]*c[1]*s[2]*c[4]+Ly*s[5]*c[1]*c[2]*s[4]+Ly*s[1]*c[5]-
Lb*c[1]*s[2]*s[4]+Lb*c[1]*c[2]*c[4]+c[1]*c[2]*d3-c[1]*a2*s[2];


  trans[1][0]= -c[6]*c[5]*s[1]*s[2]*c[4]-c[6]*c[5]*s[1]*c[2]*s[4]-
c[6]*c[1]*s[5]+s[6]*s[1]*s[2]*s[4]-s[6]*s[1]*c[2]*c[4];

  trans[1][1]=
s[6]*c[5]*s[1]*s[2]*c[4]+s[6]*c[5]*s[1]*c[2]*s[4]+s[6]*c[1]*s[5]+c[6]*s
[1]*s[2]*s[4]-c[6]*s[1]*c[2]*c[4];

  trans[1][2]= s[5]*s[1]*s[2]*c[4]+s[5]*s[1]*c[2]*s[4]-c[1]*c[5];

  trans[1][3]=  Ly*s[5]*s[1]*s[2]*c[4]+Ly*s[5]*s[1]*c[2]*s[4]-
Ly*c[1]*c[5]-Lb*s[1]*s[2]*s[4]+Lb*s[1]*c[2]*c[4]+s[1]*c[2]*d3-
s[1]*a2*s[2];
```

```
  trans[2][0]= c[5]*c[6]*c[2]*c[4]-c[5]*c[6]*s[2]*s[4]-s[6]*s[2]*c[4]-
s[6]*c[2]*s[4];

  trans[2][1]= -c[5]*s[6]*c[2]*c[4]+c[5]*s[6]*s[2]*s[4]-c[6]*c[2]*s[4]-
c[6]*s[2]*c[4];

  trans[2][2]= -s[5]*c[2]*c[4]+s[5]*s[2]*s[4];

  trans[2][3]= -
s[5]*Ly*c[2]*c[4]+s[5]*Ly*s[2]*s[4]+Lb*c[2]*s[4]+Lb*s[2]*c[4]+s[2]*d3+a
2*c[2]+d1;


  trans[3][0]= 0;

  trans[3][1]= 0;

  trans[3][2]= 0;

  trans[3][3]= 1;

  /*for (i=0; i<3; ++i)
    {
      for (j=0; j<3; ++j)
      trans[i][j]= trans[i][j]*180/pi;
      }*/

return;

}

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
//Numerical Solution to solve the Up-Down angle//

float Numerical_Up_Down (float px, float pz, float r11, float r12,
float r13, float r31, float r32, float r33, float s1, float c1, float
s6, float c6)
{
  int i,j;
  float eqn[2],Joint2, k=0.01;

  Joint2= (30.0*pi/180.0);

  eqn[1]= Lb*r31*s6+Lb*r32*c6-r33*Ly+pz-d1-
tan(Joint2)*((Lb*r11*s6+Lb*r12*c6-r13*Ly+px)/c1)-
tan(Joint2)*a2*sin(Joint2)-a2*cos(Joint2);

  for(i=1; i<= 3; ++i)
    {
      j=0;
      Joint2= (Joint2+ k);
      k= pow(0.1, i+1);  //in each subsequent loop, "k" the solution
resolution gets smaller
      eqn[0]=eqn[1];
      while(((eqn[0]> 0 && eqn[1]> 0) || (eqn[0]< 0 && eqn[1]< 0)) &&
j< 10000*k )
      /*when eqn interates pass "0", while loop is exited*/
      {
        eqn[1]=eqn[0];
      printf("eqn[1]:%f\n", eqn[1]);
        Joint2= (Joint2- k);  //decrementing the solution
        eqn[0]= Lb*r31*s6+Lb*r32*c6-r33*Ly+pz-d1-
tan(Joint2)*((Lb*r11*s6+Lb*r12*c6-r13*Ly+px)/c1)-
tan(Joint2)*a2*sin(Joint2)-a2*cos(Joint2);  //calculate eqn for new
solution
        printf("Joint2:%f\n", Joint2);
        printf("eqn[0]:%f\n", eqn[0]);
      printf("j:%d\n", j);
      ++j;
      }
      printf("In for loop\n");
    }
  printf("At Return\n");
  return Joint2;
}
       //End Of Numerical Solution//

/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void Inverse_calc(OnebySix EndPoint, OnebySix Joint)
{

int i, yes=0;
float r11,r12,r13,px, r21,r22,r23,py, r31,r32,r33,pz;
float s1,c1,s2,c2,s4,c4,s5,c5,s6,c6, sum;

for (i=0; i<3; ++i)
      EndPoint[i] = EndPoint[i]*pi/180;

     /** converts 1x6 joint positions into 4x4 end-effector matrix**/
     r11= cos(EndPoint[0])*cos(EndPoint[1]);
     r12= cos(EndPoint[0])*sin(EndPoint[1])*sin(EndPoint[2])-
sin(EndPoint[0])*cos(EndPoint[2]);
     r13=
cos(EndPoint[0])*sin(EndPoint[1])*cos(EndPoint[2])+sin(EndPoint[0])*sin
(EndPoint[2]);
     px= EndPoint[3];

     r21= sin(EndPoint[0])*cos(EndPoint[1]);
     r22=
sin(EndPoint[0])*sin(EndPoint[1])*sin(EndPoint[2])+cos(EndPoint[0])*cos
(EndPoint[2]);
     r23= sin(EndPoint[0])*sin(EndPoint[1])*cos(EndPoint[2])-
cos(EndPoint[0])*sin(EndPoint[2]);
     py= EndPoint[4];

     r31= -sin(EndPoint[1]);
     r32= cos(EndPoint[1])*sin(EndPoint[2]);
     r33= cos(EndPoint[1])*cos(EndPoint[2]);
     pz= EndPoint[5];

     /************************convertion ends*********************/


     /** Inverse Kinematics Calculations STARTS here **/

     Joint[1]= atan((r23-(py/Ly))/ (r13-(px/Ly)));
     s1= sin(Joint[1]);
     c1= cos(Joint[1]);

     i=0;
     Joint[5]= atan((-s1*r12+c1*r22)/ (s1*r11-c1*r21));
     while(yes==0 && i< 2)  //i< 2 makes sure only a maximum of 2 loops
is executed
       {
       s6= sin(Joint[5]);
       c6= cos(Joint[5]);

       Joint[2]= Numerical_Up_Down (px, pz, r11, r12, r13, r31, r32,
r33, s1, c1, s6, c6);
       /*Numerical function to solve for Joint[2]*/
       if (Joint[2]< limit_pos[2]*pi/180  &&  Joint[2]>
limit_neg[2]*pi/180)
         /*make sure Joint[2] is within joint limits*/
         {
```

```
            yes=1;  //if yes=1, while loop is exited
            printf("yes!!!!!!!!!!!!\n");
          }

        i=i+1;
        if ( Joint[5]> 0 && yes==0)
          { Joint[5]= Joint[5]-pi;}
        if ( Joint[5]< 0 && yes==0)
          { Joint[5]= Joint[5]+pi;}
        /*if Joint[2] out of joint limit, iterate again with Joint[5] in
a different quadrant*/

        printf("i= %d\n", i);
        }

     Joint[4]= atan2(c6*s1*r11-c6*c1*r21-s6*s1*r12+s6*c1*r22, s1*r13-
c1*r23);
     s5= sin(Joint[4]);
     c5= cos(Joint[4]);

     sum= atan2((-r31*s6-r32*c6), (c5*r31*c6-c5*r32*s6-r33*s5));

     s2= sin(Joint[2]);
     c2= cos(Joint[2]);

     Joint[3]= sum- Joint[2];
     s4= sin(Joint[3]);
     c4= cos(Joint[3]);

     Joint[0]= (Lb*r31*s6+Lb*r32*c6-r33*Ly+pz-d1-a2*c2)/s2;

     /** Inverse Kinematics Calculations STOPS here **/

     for (i=1; i<No_of_Actuator; ++i)
       Joint[i]= Joint[i]*180/pi;  //convert radians back to degrees

     return;
}
/*-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------*/
void Euler_calc(FourbyFour input, OnebySix orientation)  /*converts 4x4
end-effector matrix into Euler angles*/

{
  orientation[0]= atan2(input[1][0], input[0][0]);

  orientation[1]= atan2(-input[2][0], sqrt(pow(input[0][0],2)+
pow(input[1][0],2)));

  orientation[2]= atan2(input[2][1], input[2][2]);

  orientation[3]= input[0][3];

  orientation[4]= input[1][3];

  orientation[5]= input[2][3];


return;

}

/*-------------------------------------------------------------------*/
```

```
/* visual.c */

#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include <fcntl.h>
#include <linux/errno.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <string.h>
#include "rtshare.h"
#include "kinematics.h"

void main(void)
{
  int vsl, n, i;
  FourbyFour g;
  OnebySix position;

 if ((vsl= open("/dev/rtf3", O_RDONLY))<0)
  {
    fprintf(stderr, "Error opening /dev/rtf3\n");
      exit(1);
  }

 while  ((n= read(vsl, &Visual, sizeof(Visual))) > 0)  //read from
fifo"3"
   {
     printf("msg:%d\t%d\t%6.2fmicro seconds", Visual.msg,
Visual.SampleNumber, Visual.InterSeconds);
     printf("\n");

     for(i=0; i< No_of_Actuator; ++i)
       {printf(" %5.2f\t", Visual.Points[i]);}
     printf("\n\t");

     for(i=0; i< No_of_Actuator; ++i)
       {printf(" %3.1f\t", Visual.PWM[i]);}
     printf("\n");

     forward_calc(Visual.Points, g);  /*calculate end-effector position
and orientation from joint positions, the function is in kinematics.h*/

     Euler_calc(g, position);  //calculate the Euler angles, the
function is in kinematics.h


   printf("\norientation:");
  for (i=0; i<3; ++i)
    {
      position[i]= position[i]*180/pi;
      printf("\t%5.2f", position[i]);
    }

  printf("\nposition:");

  for (i=3; i<6; ++i)
```

```
    printf("\t%5.2f", position[i]);

  printf("\n");
     printf("\n\n\n");

   }

return;

}
```

```
function plotsix %Matlab function to plot response graph

file=('f:\Ramdata\');  %make sure this is the correct directory

actuator= {'In/Out', 'Rotate', 'Up/Down', 'Bend', 'Yaw', 'Swivel'};
data=input('Filename to retrive. in inverted commas. >>');  %gets
filename
filename=strcat(file,data);
sample=input('Sampling time in seconds. >>');
%combine to get the whole path

result= dlmread(filename,'\t');
%read the data from the file

time= result(:,1)*sample;

clf;
figure(1);
title('ARM plot for Rotate, Extend, Up-Down')
for i=1:3

  desired(:,i)= result(:, 2+3*(i-1));
  actual(:,i)= result(:, 3+3*(i-1));
  PWM(:,i)= result(:, 4+3*(i-1));

  subplot(3,2,1+2*(i-1))

  if i==1
    unit= '(mm)';
  else
    unit= '(degrees)';
  end;

  plot(time,desired(:,i),'r-'); %plot the input signal
  hold on
  plot(time, actual(:,i),'b');
  xlabel('Time, seconds');
  ylabel(strcat(actuator(i),unit));
  hold off;
  zoom xon;

  subplot(3,2,(2*i));
  plot(time, PWM(:,i),'b');
  xlabel('Time, seconds');
  ylabel('PWM, (%) ');
  zoom xon;

end

figure(2);
title('WRIST plot for Bend, Yaw, Swivel')

for i=4:6
  j=i-3;

  desired(:,i)= result(:, 2+3*(i-1));
  actual(:,i)= result(:, 3+3*(i-1));
```

```
  PWM(:,i)= result(:, 4+3*(i-1));

  subplot(3,2,1+2*(j-1));

  plot(time,desired(:,i),'r-'); %plot the input signal
  hold on
  plot(time, actual(:,i),'b');
  xlabel('Time, seconds');
  ylabel(strcat(actuator(i),unit));
  hold off
  zoom xon;

  subplot(3,2,(2*j))
  plot(time, PWM(:,i),'b');
  xlabel('Time, seconds');
  ylabel('PWM, (%) ');
  zoom xon;

end
```

APPENDIX E- MATLAB FILES FOR KINEMATICS AND INVERSE KINEMATICS

```
function forward %function to calculate the end-effector matrix frame
syms d3 t1 t2 t4 t5 t6 T1 T2 T3 T4 T5 T6 d1 a2 Lb Ly

T1= [cos(t1) 0 sin(t1) 0;
   sin(t1) 0 -cos(t1) 0;
   0 1 0 d1; 0 0 0 1];

T2=[-sin(t2) 0 cos(t2) -a2*sin(t2)
   cos(t2) 0 sin(t2) a2*cos(t2)
   0     1   0                    0
   0      0   0                       1];

T3= [1 0 0 0; 0 0 1 0; 0 -1 0 d3;0 0 0 1];

T4= [cos(t4) 0 sin(t4) 0; sin(t4) 0 -cos(t4) 0;
   0 1 0 0; 0 0 0 1];

T5= [cos(t5) 0 -sin(t5) 0; sin(t5) 0 cos(t5) 0;
   0 -1 0 Lb; 0 0 0 1];

T6= [cos(t6) -sin(t6) 0 0; sin(t6) cos(t6) 0 0;
   0 0 1 Ly; 0 0 0 1];


T04R= simplify(T1*T2*T3*T4);
T05R= simplify(T1*T2*T3*T4*T5);
T06R= simplify(T1*T2*T3*T4*T5*T6)

%shows end-effector orientation in alpha, beta and gamma angles
tanalpha= simplify(T06R(2,1)/T06R(1,1))
tanbeta= simplify(-T06R(3,1)/sqrt(T06R(1,1)^2+T06R(2,1)^2))
tangamma= simplify(T06R(3,2)/T06R(3,3))
```

```
function inverse %generates equations for solving inverse kinematics
syms d1 a2 d3 t1 t2 t4 t5 t6 T1 T2 T3 T4 T5 T6 Lb Ly r11 r12 r13 px r21
r22 r23 py r31 r32 r33 pz


T01= [cos(t1) 0 sin(t1) 0;
   sin(t1) 0 -cos(t1) 0;
   0 1 0 d1; 0 0 0 1];

T12=  [-sin(t2) 0 cos(t2) -a2*sin(t2)
   cos(t2) 0 sin(t2) a2*cos(t2)
   0     1   0                    0
   0      0   0                       1];

T23= [1 0 0 0; 0 0 1 0; 0 -1 0 d3;0 0 0 1];

T34= [cos(t4) 0 sin(t4) 0; sin(t4) 0 -cos(t4) 0;
   0 1 0 0; 0 0 0 1];

T45= [cos(t5) 0 -sin(t5) 0; sin(t5) 0 cos(t5) 0;
   0 -1 0 Lb; 0 0 0 1];

T56= [cos(t6) -sin(t6) 0 0; sin(t6) cos(t6) 0 0;
   0 0 1 Ly; 0 0 0 1];

T06= [r11 r12 r13 px; r21 r22 r23 py; r31 r32 r33 pz; 0 0 0 1];


Tinv01= simplify(inv(T01));
T16L= Tinv01*T06
T16R= simplify(T12*T23*T34*T45*T56)

Tinv12= simplify(inv(T12));
T26L= simplify(Tinv12*T16L)
T26R= simplify(T23*T34*T45*T56)

Tinv23= simplify(inv(T23));
T36L= simplify(Tinv23*T26L)
T36R= simplify(T34*T45*T56)

Tinv34= simplify(inv(T34));
T46L= simplify(Tinv34*T36L);
T46R= simplify(T45*T56);

Tinv45= simplify(inv(T45));
T56L= simplify(Tinv45*T46L);
T56R= simplify(T56);

Tinv56= simplify(inv(T56));
T15L= simplify(T16L*Tinv56)
T15R= simplify(T12*T23*T34*T45)

T14L= simplify(T15L*Tinv45)
T14R= simplify(T12*T23*T34)

T25L= simplify(Tinv12*T15L)
T25R= simplify(T23*T34*T45)
```