Proc. 2001 Australian Conference on Robotics and Automation
Sydney, 14-15 November 2001

Retrofitting Path Control to a Unimate 2000B Robot

S D Donald and G R Dunlop
Department of Mechanical Engineering,
University of Canterbury, Christchurch, New Zealand

Abstract

A large hydraulic Unimate 2000B robot was donated to the University of Canterbury. The original control system was spread across 8 large circuit boards and provided point-to-point control. The new controller uses a single A5 size circuit board, an FPGA and a PC to provide full path control with a fully digital interface. Analog to digital or digital to analog conversions are avoided with the new controller and a very robust interface is obtained. The interface techniques and the software controller implementation are described along with some of the results obtained to date.

1 Introduction

The Unimate 2000B is a serially connected 6-degree of freedom hydraulic robot, and is one of the first robots to be produced by Unimation Industries in the 1970's. The original control hardware has been disconnected and replaced by a PC with some simple hardware and software to take over the control duties. The result is a cheap high performance controller that is easily modified. This implementation provides better control of the robot, as well as a more flexible interface to provide not only the original point to point control, but also the ability to track a user defined path.

The real-time operating system for this application runs Microsoft Windows NT as a client over VenturCom Real-Time Extension (RTX) for Windows NT. This combination of a real-time kernel and general purpose graphics user interface (GUI) allows the use of commercial off-the-shelf technology: a powerful desktop development and test environment, an abundance of proven software development tools, a well-understood Win32 application programming interface (API), and a talent pool of skilled software engineers.

The use of the proven Win32 development tools provides real-time developers with a major advantage. Real-time applications on Windows platforms can be explicitly designed for use with both Win32-based and hard real-time-based processes. This design mix of processes usually requires the use of highly specialized, and often obscure, tools for real-time application development [Dunlop and Yang, 1999]. With RTX, software engineers can now take advantage of the large number of Win32-based development tools for both the Win32 and the real-time processes.

2 The Unimate 2000B

All six axes of the Unimate 2000B manipulator are driven by hydraulic actuators, which are controlled by Moog electro-hydraulic servo valves. The hydraulic actuators for the In-Out and Down-Up motions are connected directly to their respective loads. For the waist rotation, a rack and pinion converts linear travel of a hydraulic ram to rotary motion. The wrist angular motions of Bend, Yaw and Swivel are transmitted by systems of chains, gears and shafts with a ball nut and spline shaft arrangement used to transmit motion to the final wrist axis. Hydraulic power is generated using a vane pump powered by a 7.5 kW induction motor. An accumulator has been charged with dry nitrogen to a pressure of 3.5 MPa to improve the transient performance of the hydraulic actuators. Each of the 6 electro-hydraulic servo valves is a 4-way infinite position valve. The analog control system is spread over 8 A3 size circuit boards, and has been replaced by a single A5 circuit board and a PC. This circuit board provides the pulse width modulation (PWM) system used to drive the valves, and also a readout interface for the encoders on each axis.

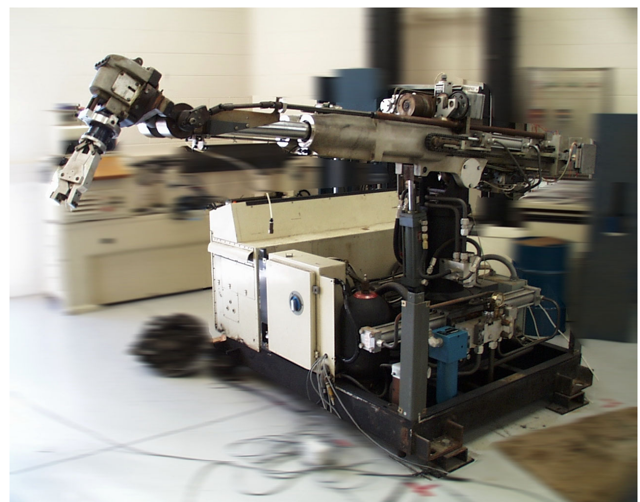

Figure 1: The Unimate 2000B without covers.

Copyright © ARAA 2001

3 Electronic Hardware

The hardware used in this application consists of a Xilinx field programmable gate array (FPGA) and a single custom circuit interface board. The FPGA is plugged into a desktop PC (150MHz Pentium II, 128MB), which runs the RTX and Win32 software applications [Dunlop et al., 1990].

3.1 The FPGA

The FPGA has been interfaced to the 16-bit PC ISA bus so that the Win32 and real-time software system (RTSS) processes can interact directly with the FPGA. The FPGA is software configurable and the gate assignment has been designed to provide several functions:

- Configure a timer to interrupt (1 ms) at 6 times the digital control system sample rate (6 ms).
- Generates the positive and negative PWM signals needed to drive the electro-hydraulic valves.
- Provide a digital interface for reading the actuator position Gray scale encoders (14 bits).
- Selects which of the 6 encoders to read.

The FPGA board has a 12MHz crystal clock, which is counted to generate the interrupt time. The FPGA sets an internal register flag when the counter reaches a preprogrammed value (specified by the program at run time). The RTSS process interrupt is connected to this register, and is triggered when this flag is set. At the conclusion of the interrupt service routine (ISR) the register flag is reset, and the interrupted Windows program processing resumes until the flag is set again.

The FPGA also generates the PWM signal required to drive a solenoid valve. It accepts the required PWM duty cycle (0 - 100%) and direction for a valve, and then converts these to 2 digital signals, PWM+ and PWM- that are sent to the external circuit board for amplification. Note that it is also possible to set up the FPGA to accept a 2's complement number and then convert it to the 2 required PWM signals.

3.2 Custom Circuit Board

The external custom A5 size circuit board provides optical isolation between the computer/FPGA circuits and Unimate 2000B circuits. It also provides amplification of the PWM signals generated by the FPGA board. When the PWM is on, an 80mA current is passed through the electro-hydraulic valve [Dunlop and Hampson, 1995] as is shown in fig.2. The board has a number of safety relays that shut down the robot if someone enters the robot workspace and breaks an infrared beam, or if the computer fails to respond to an interrupt i.e. it has crashed and no longer provides control.

4 Software

The advantage of using a PC to control the robot is that custom control and user interface software can be created and run easily. All of software has been created in C++ using Microsoft's Visual C++ environment. The real time

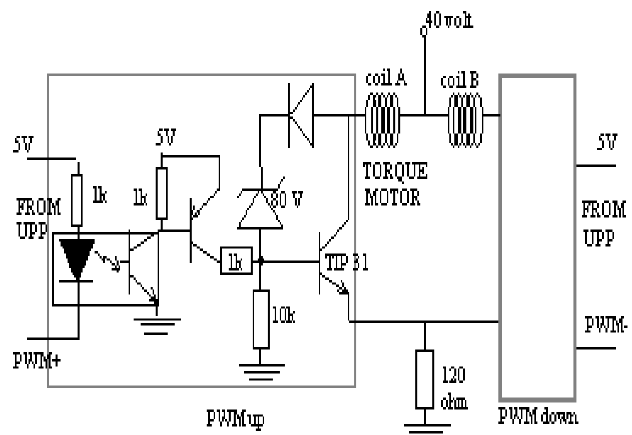

Figure 2: The isolated interface for driving an electro-hydraulic valve. Valve current is proportional to the PWM.

code is restricted to being written in C/C++ but the user interface could have been created in a number of different ways, including Java, Visual Basic, or Matlab. However the use of these languages would still require the use of Win32 dynamic link libraries written in C++ to allow communication with the real time process. The RTX API does not include support for any GUI related calls. Hence the GUI is created as a Win32 process and communicates with the RTSS process through shared memory.

4.1 Graphical User Interface/Operation

A GUI has been developed using Microsoft Visual C++ and is shown in fig.3. Through the GUI the user is able to drive the solenoid PWM values directly, or to track a path. With direct PWM control, a static PWM value can be entered or a sinusoidal PWM signal can be generated. This is useful for obtaining frequency response data. The path can also be a step, a ramp, or a complex user defined path. A graph of the desired actuator position, actual position, and the PWM effort can all be displayed while the robot is in operation.

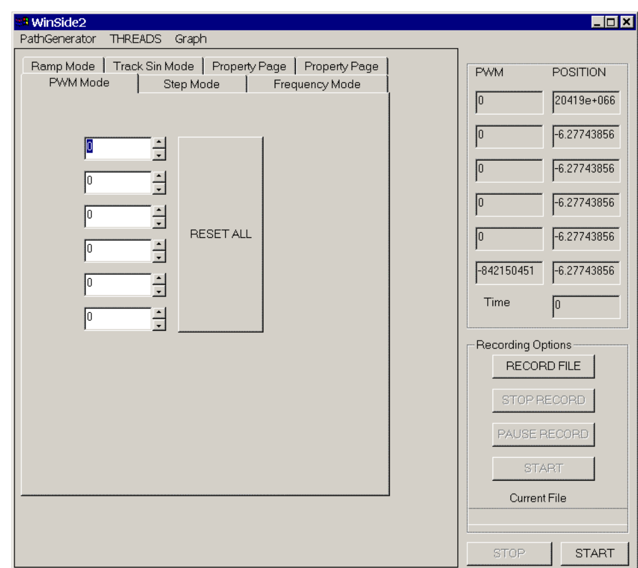

Figure 3: The main graphical user interface.

Inter-process communication between the GUI Win32 and the RTSS processes is achieved through shared memory. The shared memory consists of two circular buffers, one to send data from the RTSS process to the GUI, and the second to send path information from the GUI to the real-time process. A third shared memory block is created for a matrix data structure that contains joint data calculated from the path points (or knots) provided by the path planning software. These points control the behavior of the robot manipulator. The joint path generation program uses cubic spline interpolation between the points to generate real-time path demand coordinates for the joint control systems to track.

4.2 Path Generation

The offline path generation software was created as a C++ application as shown in fig.4. A series of way points (or knots) along the path are created from a series of end effector matrices which may be defined relative to the base frame or relative to the previous frame. The end effector matrices may be entered directly in the form of Euler (or RPY) angles and the gripper position. The gripper or end effector matrix can also be measured from the robot's current position if required.

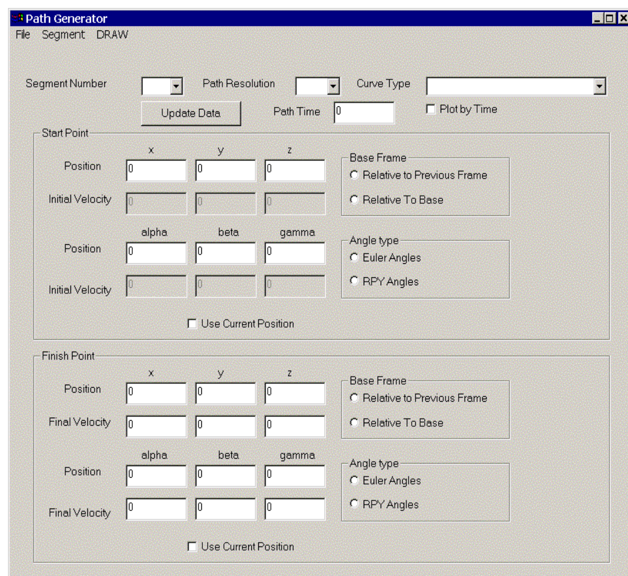

Figure 4: The path planning user interface.

A cubic spline is fitted through the knots to obtain the end effector path. The acceleration and velocity limits are applied to this interpolated path to determine further waypoints for which the inverse kinematics are solved in order to generate the waypoints for the joint coordinates. These joint coordinates and times are placed in the data array shared with the real-time control system.

$$\begin{bmatrix} a_{0,0} & a_{1,0} & a_{2,0} & a_{3,0} & t_0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{0,n} & a_{1,n} & a_{2,n} & a_{3,n} & t_n \end{bmatrix}$$

A cubic spline is fitted to each joint variable array in real-time to generate the demand x for the joint control system.

$$x = a_{0,k} + a_{1,k}t + a_{2,k}t^2 + a_{3,k}t^3$$

where $0 < t < t_k - t_{k-1}$ and all times are integer multiples of the control system sample time Δ (6 ms).

4.3 Interrupt Service Routine Considerations

The design of real time software requires careful consideration of a number of factors, the most important being timing issues. The robot controller is relatively uncomplicated and makes use of a single ISR that is activated at 6 times the control system sample rate. The operation of the ISR is shown in fig.5. The main issue is

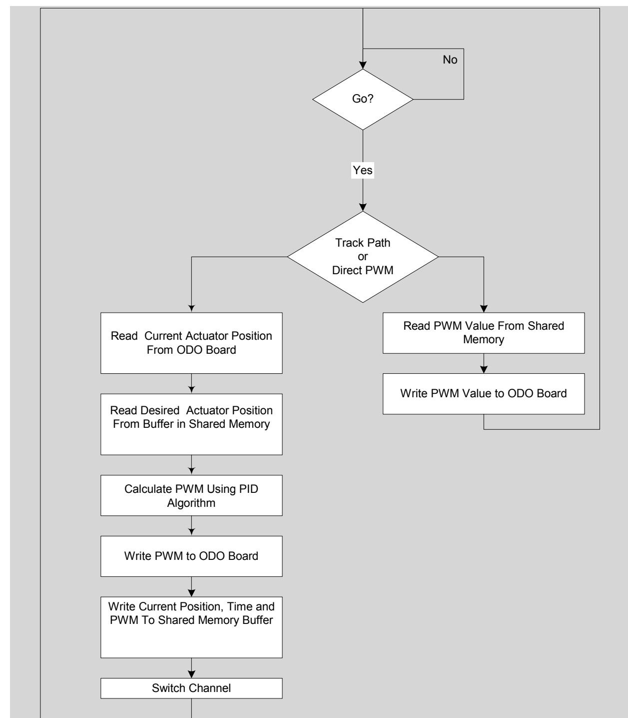

Figure 5: The interrupt service routine.

to reduce the interrupt latency (the time from setting the interrupt flag to running the ISR) and ensure that the ISR is as efficient as possible. With a Pentium processor, cache flushing can cause lengthy interrupt latencies. While this cannot be prevented, careful design can reduce problems.

4.3.1 Cache

Fast memory is expensive so it is common to use large blocks of slow (cheap) memory, and a relatively small amount of very fast (expensive) cache memory. The idea is that the processor, as a background activity, keeps filling the instruction and data caches with the information that it expects to need next. If the required information is found in cache memory when required then this is called a cache-hit, otherwise a cache miss occurs. A time-critical instruction loop will run fastest if all its instructions and data fit within the instruction and data caches. Slowest operation occurs when the data cache must be saved (to slow memory) before being loaded (from slow memory)

with the data needed for the ISR. The program cache only needs to be loaded as the current program cache instructions do not change and thus do not need to be written back to slow memory.

Caches are introduced into a system to buffer the mismatch between main memory and processor speeds. The cache is designed so that its access time matches the processor cycle time. Thus, if the processor is running with a 100MHz clock the cache should be able to respond to a memory request in approximately 10ns. The typical size of the first-level (L1) primary caches on the processor is 8kb for program, and 8kB for data or a total of 16kb of cache on the processor chip. Many system designs also include a larger (128kB to 4MB) off-chip cache, which is called the second-level (L2) cache.

While the first-level cache must match the processor speed, the second-level cache can be somewhat slower, but not as slow as the main memory. Thus processor memory requests pass first to the primary L1 cache, and if there is a cache miss the request is forwarded to the L2 cache. If the data is not found in the L2 cache, then the request is finally forwarded to the main memory. When the main memory responds to the memory request, the data item is passed back to the L2 cache and then to the L1 cache.

4.3.2 Program data structures

Caches work well because the primary cache can usually service a memory request. The reason that so many memory requests can be handled by the primary cache has to do with two aspects of program behaviour:

- Temporal locality: If a memory location is referenced, it is very likely that the memory location will be referenced again in the near future.
- Spatial locality: If a memory location is referenced, it is very likely that a nearby memory location will also be referenced in the near future.

Spatial locality is embodied in a cache design by grouping sequential bytes of memory into a single *cache line*: Information transfer between the cache and the memory is in terms of complete cache lines, rather than individual bytes. Thus if the program needs a particular byte, the entire cache line containing that byte is obtained from the memory.

It is important that the critical part of the ISR code (the innermost loops) fits into the program cache. Frequently used pieces of code or routines that are used together should be stored close together. Seldom used branches or procedures should be put at the bottom of the ISR code. If the critical part of the code accesses large data structures or random data addresses, then all frequently used variables (counters, pointers, control variables, etc.) are kept within a single contiguous block of 4 KBytes so that a complete set of cache lines is available for accessing random data. The Pentium PII and above have 16 KBytes for code and 16 KBytes for data.

The PII data cache consists of 512 lines of 32 bytes. Each time a data item, which is not cached, is read, the processor will read an entire cache line from memory. The cache lines are always aligned to a physical address divisible by 32. When a byte is read from an address divisible by 32, then the next 31 bytes can be accessed at minimum time from the L1 cache. Data items which are used near each other are arranged together into aligned blocks of 32 bytes of memory e.g. if a loop accesses two arrays, then interleave the two arrays into one array structure so that data are used together and also stored together. This almost doubled the access speed. Also the separation of the read only and the read plus write data structures is beneficial. Data that has been modified will need to be written back to slow memory when removed from cache, whereas unmodified data blocks will not. Thus separation prevents unnecessary write back of unmodified data.

While the cache memory designed into advanced processors can significantly speed up the average performance of many programs, it also causes performance variations that surprise system designers and cause problems during product integration and deployment. The two main problems are a lack of predictability and a lack of determinacy. Determinacy means that the results of executing a piece of code, including how long it takes to execute, don't change from run to run. With the use of cache memories, determinacy and predictability are badly affected. The problem with poor predictability is that performance of programs can be very difficult to characterise during development, and the problem with poor determinacy is that a program can have latent timing problems that are extremely difficult to detect, reproduce, and correct.

4.3.3 Performance

The very worst execution times are instances when there many cache misses while executing the code, as well as DRAM refreshes or other untoward events occurring to further slow down the program. Execution times can vary greatly from run to run, depending on what other programs have done to the cache memory between ISR executions. The FPGA was set up to generate an interrupt at millisecond intervals and the counter was reset to zero when each interrupt was generated. The ISR read the counter to get a value proportional to the interrupt latency and the results from 10,000 ISR latency measurements are

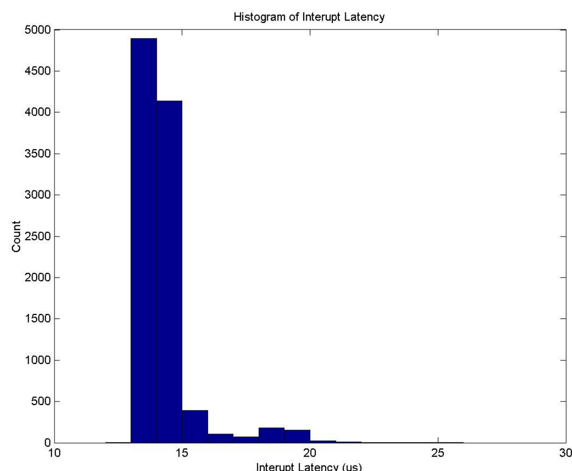

Figure 6: Histogram of interrupt latencies.

given in fig.6. An average interrupt latency of 14 µs was measured with a maximum of 26 µs and minimum of 12 µs. This proved quite adequate for controlling the robot.

5 Control algorithms

The control algorithms have been developed on a simple digital version of the PI controller as shown in fig.7. The delay constant Δ is set to 6 ms, which corresponds to the sample rate of the digital control system. This time is 6 times the interrupt interval as the position feedback encoders require approximately 1 ms to settle after being selected. The 6 Gray scale encoders are multiplexed onto the same 14-bit data bus and must be selected one at a time. Thus after an encoder is read to determine the current position of an axis, the next encoder is selected so that it is ready to be read during the next ISR invocation to control the axis for that encoder.

Figure 7: **The digital controller for 1 axis**

Figure 8: **Bode plot measured for the up/down axis.**

Coupling between the axes has been neglected and, in this case, was found to make little difference to the performance. The frequency response of each axis was measured using the frequency option from main graphical user interface (c.f. fig.3). A typical Bode plot is shown in fig.8 for the up/down axis (second from the base) with the boom extended halfway. The PI constants have been calculated from frequency response plots of the system.

6 Discussion

The real-time part of the software runs a simple control algorithm and joint coordinate path interpolation algorithm. All other aspects of running the robot are taken care of in a Win32 application that communicates with the RTSS through shared memory. The Win32 application has 3 threads:

- The main thread that takes care of the user interface and all user requests.
- A thread writing the path to shared memory.
- The third thread reading information from shared memory and writing data to file as well as plotting graphs showing the system behaviour.

The advantage of running Windows NT on RTX is that a number of methods may be used to write the path planner, user interface etc. Dynamic Link Libraries can be written in C^{++}, which acts as an interface between any Win32 application and the RTSS program. It is then be possible to use programs such as Visual Basic, Java, or Matlab to develop the project in a high level language.

7 Conclusion

An obsolete robot has been refurbished by replacing the old complex controller circuitry with a simple digital interface and using software to implement the various control functions. Normally this would require specialized software running on a specialized computer or DSP, but here adequate performance has been obtained using a Pentium PC for the task. Some care was required to avoid memory cache problems inherent in modern microcomputers. Despite this, more than adequate performance for the control task was obtained from a basic PC and specialized hardware and software were not required.

References

[Dunlop et al. 2000] Dunlop G R, Cree A G, Murphy J D J and Phillips J P Programmable gate arrays for hard real-time control. *Proceedings of the Mechatronics and Machine Vision in Practice, M2VIP, Ed. Billingsley J*, Research Studies Press Ltd, ISBN 0 86380 261 3, Hervy Bay, Australia, Sept. 19-21: 343 – 348, 2000.

[Dunlop G R, Yang J L C, 1999] Dunlop G R, Cree A G, Katzir S and Yang J L C Machine control with real time Linux, *Proc. Control 99*, Auckland, NZ, #32: 1-6, 1999.

[Dunlop and Hampson 1994] Dunlop G R and Hampson S P An integrated digital control system for hydraulic actuators, *Proc. Seventh Bath International Fluid Power Workshop*, Bath, U.K., Sept. 21-23: 2-11, 1994.