13th National Conference on Mechanisms and Machines (NaCoMM07),
IISc, Bangalore, India, December 12-13, 2007 **NaCoMM-2007-072**

Intelligent Modular Controller for Robotic Machines

Yogesh Kumar[1*], Dr. S.K. Srivastava[2], and Dr. A.K. Bajpai[3],
[1]M.Tech. Student, [2]Assistant Professor, [3]Professor,
Department of Mechanical Engineering,
Madan Mohan Malaviya Engineering College,
Gorakhpur (U.P.) -273 010, India.

[*] Corresponding author (email: cim.gkp@gmail.com)

Abstract

The present work is focused on the design and implementation of an Intelligent Modular Controller (IMC) architecture designed to be reconfigurable over a robust network. The design incorporates novel communication, hardware, and software architectures. This was motivated by current industrial needs for distributed control systems. It incorporates the growing demand for less complexity, more processing power, flexibility, and greater fault tolerances.

This paper demonstrates hardware/software co-design as a contribution to the growing discipline of mechatronics. The IMC consists of a motion controller board designed and prototyped in-house, and a Java microcontroller.

An IMC is mapped to each machine/robot axis. An additional IMC can be configured to serve as a real-time coordinator. The entire architecture is implemented in Java. The results show the potential of the flexible controller to meet medium to high performance machining requirements such as surface finish, precision improvement etc.

Keywords: Robot, Intelligent Machines, Mechatronics, Hardware Interfacing.

1. Introduction

The paper presents the design of an intelligent modular reconfigurable controller for machine tools. The design breaks new ground by incorporating embedded technology using commercial-off-the-shelf (COTS) components, flexible architectural design patterns based on object-oriented technology, plug-and-play, and web-design into a distributed control system. The treatise takes a tour through the different design stages from concepts to production, and also gives a rich background of the related state-of-the-art. Experimental results and future research directions wrap up the main paper body.

2. Problem formulation and Objective

The Mechatronics approach for the design of modern control systems inevitably involves embedded technology. This approach is gaining a lot of attention due to the growing demand for distributed real-time systems. Indeed, the gradual paradigm shift from centralized systems is justified in many ways. The most compelling reasons are the needs for less complexity, more processing power, flexibility, and greater fault-tolerance. A typical distributed control system (DCS) consists of several processing nodes connected by a communication network. The network presents a duo of both convenience of connectivity, and inconvenience of dealing with real-time situations. Designs based on such systems have limited flexibility in terms of scalability, unlike serial communication systems where a few serial lines could connect many elements. The objective of the present work is:

1. A generic framework for modular reconfigurable control architecture. The framework addresses software and hardware requirements, and also the communication structure.
2. A small and simple design that fits into embedded low-cost platforms.
3. A working prototype of the architecture.
4. An operational software architecture based on modularity and reusability.

3. Robot Control Architecture Design Concepts and Review

Robot control architecture involves several different notions and implications, particularly architectural styles and structures. Architectural structure shows how a system is decomposed into the subsystems, and how the subsystems interact to each other. The computation and communication underpinnings of a given system invariably reflect a style. For example, one system might use a publish-subscribe message passing style of communication, while another may use a more synchronous client-server approach. Most often the holistic architecture is realized only at the working stage. This is unfortunate, since a well-conceived architecture can have many advantages in the specification, execution, and validation of robot systems. This helps in developing a robot control architecture framework. Generally, a framework refers to the structure external to an architecture which organizes information about the architecture and its application [3].

3.1 Architectural Properties

The architectural style employed has a direct impact on the performance of the overall system. For example, the pipe-and-filter software style supports components reusability and configurability of the application by applying generality to its component interfaces. However, components are constrained to a single interface type. The desired architecture properties are dealing with complexity, execution, openness, performance, scalability, simplicity, modifiability, portability and reliability [5].

3.2 Software Architecture Styles

A software architecture is an abstraction of the runtime behavior of a software system during some phase of its operation. The architecture of a software system defines that system in terms of components and of interactions among those components [5].

3.3 Controller Hardware Architecture

Hardware architectures usually emphasize the computation platforms, their interfaces, and interconnections. In effect the hardware architecture should help readers to understand the underlying execution mechanics and dataflow through different execution stages. The Controller hardware architecture is evaluated on the basis of:

1. Throughput (speed)
2. Communication between modules
3. Functional coherence
4. Robustness
5. Hardware Modularity

The following controller hardware architectures are already available for the present study:

1. Open Modular Architecture Controller (OMAC) API
2. UBC Open Architecture Control System
3. NRC Tripod
4. CLARAty (Coupled Layered Architecture for Robotic Autonomy)Architecture

3.4 Control Architectures

A control architecture refers to the actual operational software used to run a machine (e.g. robot), and may also include intelligence to handle interactions with the environment or system, and optimization procedures necessary to enhance performance. This section reviews some well-known control strategies and some new paradigms in system-level control reconfiguration.

3.4.1 Classic Control

Robot control for contour following operations can be classified into gross motion control and fine motion control. The classification of robot control is given in the Figure 1. In gross motion control, an end-effecter or machine tool follows a prescribed path as closely and as quickly as possible; the task is to find a control law that governs its velocity and position. On the contrary, in fine motion control, the objective is to control position and force simultaneously by a technique called hybrid control. Gross motion control may be implemented in joint or Cartesian space depending on whether the desired path is specified in the joints or Cartesian space. Fine motion control may be passive or active compliance control. The former may be achieved through Remote Compliance Center devices to compensate for disturbances. In active compliance control, there is force (torque) feedback to correct errors [4].

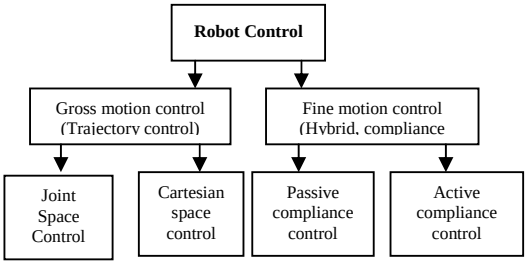

Figure 1: Classification of Robot Control

3.4.2 Reconfigurable Control Architectures

Generally, there are two issues regarding reconfiguration: Distributed system reconfiguration and Control reconfiguration. The former is related to a structural or system-level software reconfiguration while the latter deals with modifying control law in its structure to maintain a certain performance. Reconfigurable control for fault tolerance is an exceptionally challenging control design problem. Failure detection and identification, parameter estimation and controller redesign have to be carried out on-line and completed within tight time boundaries. Some methods that address reconfigurable control include linear-quadratic (LQ) control methodology, adaptive control systems, knowledge-based systems, and Eigen-structure assignments. Considering adaptive control, there are several approaches but in general perspective control reconfiguration is attempted by using a continually adapting nonlinear model. At this point we make a distinction between fault-tolerance, robust control, and reconfiguration control strategies. In the first situation, when a fault appears in one peripheral element and the plant is still observable and controllable, the controller uses a fault accommodation strategy to achieve its original objectives by adapting control parameters to fault conditions [4].

A robust controller (also called adaptive controller in some literature) aims at providing suitable system performance if the parameters and conditions vary within given domains. Parameters and conditions include uncertainties in the mathematical model of the plant, and strong non-linear interconnections (e.g., between the joints of a robot); for distributed systems, robust controllers accommodate network-induced jitters and sometimes computation delays [7].

4. Intelligent Modular Controller (IMC) Architecture

Each axis controller (IMC) is designed such that there is minimal need for global data interchange. Therefore most hard real-time functions are confined within the IMC domain. Each IMC communicates with the central coordinator through a network while communication with the mechanical axis is through an embedded motion controller and a set of I/O signals. The IMC architecture allows a completely distributed or a hierarchical architectural structure in order to accommodate different demands imposed by higher application layers.

Another feature of the IMCs is the ability for any one of them or an additional IMC to serve as a real-time coordinator. This is especially important in two different applications: The first is the situation where the global workstation or computer which hosts most of the higher layer application software runs on a non real-time operating system (OS). While many functions such as path- planning and NC program parsing may be abstracted from non real-time computing, real- time transactions are a necessity for coordinating coarse or finely interpolated data. The architecture allows an IMC to be selected for this purpose. In this case, the IMC is relieved of its motion control activities in order to conserve computing resources for real-time coordination. The architecture, therefore, does not predispose the user to any particular operating system which is a key advantage.

The other interesting feature of the IMC architecture is that they are designed from the onset to be embedded mobile computing elements. This provides the ability for them to be integrated or embedded in the mechanical platform. The concept can therefore be used to control mobile equipment such as AGVs or mobile robots. There are a few architectural permutations that can address such situations: One arrangement is to have one IMC inside each host mechanical axis; another variation is to map several motion controllers to one IMC microcontroller.

4.1 Java for Real-time System Design

The Java Virtual Machine (JVM) loads, verifies and executes the byte code of a Java program. Execution speed is hindered by interpreting byte codes, and this has been one of the setbacks of Java in real-time applications until recently. One solution to this problem is a JVM with a just- in-time (JIT) compiler designed for desktop and server systems. However these require large memory footprints and have to be ported for different processor architectures.

An excellent candidate for real-time embedded system designs is a Java processor (hardware) that implements the JVM as a native machine. This avoids the slow execution model of an interpreting JVM and the memory requirements of a compiler, thus making it suitable for embedded systems. There are two approaches to Java bytecode execution by hardware. In the first approach, a Java coprocessor is placed in the instruction fetch path of a general purpose microprocessor and translates Java bytecodes to sequences of instructions for the host CPU or directly executes basic Java bytecodes. In the second approach, a Java chip replaces the general purpose CPU. All applications therefore have to be written in Java.

4.2 Microcontroller Hardware Selection

The criteria for the selection of an appropriate computing platform are as follows:

1. The hardware must be capable of supporting at least 60 MHz of computing with 2 Mbytes of storage space.
2. The underlying operating system must be capable of multi-tasking synchronously and/or asynchronously.
3. There should be provision for a real-time serial bus and bi-directional Ethernet.
4. Computing platform based on Java.
5. There should be an appreciable number of input/output pins for digital or analog interfacing, serial and parallel interfacing.

Even though there is a wide variety of processing platforms available, but many of them do not meet the aforementioned requirements of an embedded technology with native support for real-time, object-oriented programming and communication network. For this reason, we selected the aJile microprocessor (see Table.1). The chipset is only US$25 (2004 price). The next section gives a brief description of the processor with emphasis on the features of interest to us.

Table 1: Micro-Controller Chips

Product	Type	Chip Technology	Speed (MHz)	Java Standard
JIFFY	Translation	FPGA		
Jazelle	Co-Processor	ASIC 0.18 *µs*	200	
ZSTAR	Co-Processor	ASIC 0.18 *µs*	104	J2ME CLDC
picoJava	Processor			Full
aJile	Processor	ASIC 0.25 *µs*	103	J2ME CLDC
Cjip	Processor	ASIC 0.25 *µs*	67	J2ME CLDC
Komodo	Processor	2600 LCs	20	subset

4.2.1 The aJile Processor

The aJile architecture uses JEM2 as a direct-execution Java processor that is available as both an IP core and a stand alone processor (aJ-100, 2001).The data path is made up of a 32-bit ALU, a 32-bit barrel shifter and the support for floating point operations (disassembly/assembly, overflow and NaN detection). The control memory is a 4K by 56 ROM to hold the microcode that implements the Java bytecode. An additional RAM can be used for custom microcode to implement new instructions. The aJile inventors report that this feature can increase the efficiency of frequently used algorithms by 5–50 times by decreasing execution overheads. This feature is also used to implement basic synchronization and thread scheduling routines in microcode to yield context- switching of 1 µs. A Multiple JVM Manager (MJM) supports two independent, memory

protected JVMs, which can execute with a deterministic schedule and full memory protection (Figure 2) [1].

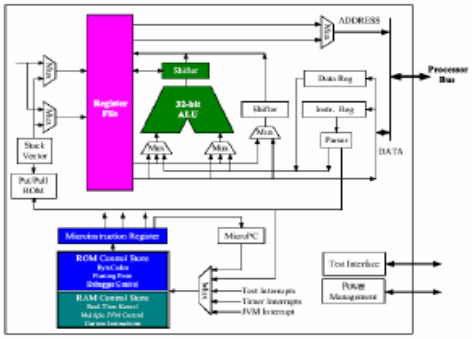

Figure 2: The aJile JEM2 Processor [1]

Currently, there are two silicon versions of JEM2 microcontrollers: the aJ-80 and the aJ-100. The aJ-100 shown in Figure 3 (aJ-100, 2001), provides a generic 8-bit, 16-bit or 32-bit external bus interface and can be clocked up to 103 MHz, while the 66-MHz aJ-80 only provides an 8-bit interface. Both versions are made up of the JEM2 core, the MJM, 48-KB zero wait state RAM and peripheral components, such as timers, I/O's, a real- time serial communication bus (SIP) and UART. 16KB of the RAM is used for the writable control store and 32 KB for storage of the processor stack. Both microprocessors are bundled with J2ME-CLDC Java runtime system, optimizing application builder, and a very basic debugging tool. Complete implementations for real-time networked embedded Java applications are available (aJ-100, 2001 and Systronix, 2003) [1].

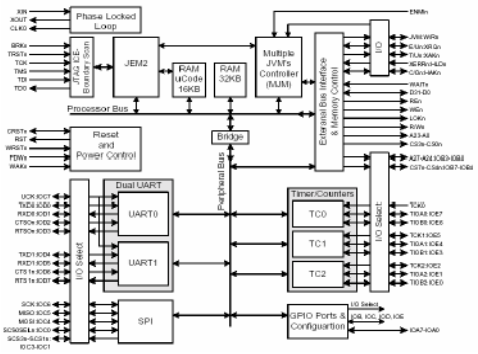

Figure 3: aJ-100 Architecture [1]

4.2.2 The JStick Platform

Systronix Inc. (2003) produces about five different boards with the aJile microprocessor. Their JStick microcontroller meets most of our preferences such as small form factor, I/O input/output) interfaces, and network support. JStick is a Single Board Computer (SBC) with a SIMM30 (Single In-Line Memory Module) format. It features a host of facilities to provide most controller functionalities. The following is summary of the portfolio of this device [2]:

1. Processor: aJ100 (100 MHz).
2. Memory: 2-Mbytes SRAM and 4-Mbytes flash.
3. Power: switching power converters which provide 5 V and 3.3 V for peripheral devices.
4. Power supply: unregulated DC of 9-14 volts or regulated DC power of 5V.
5. Communication: serial I/O, 10BaseT Ethernet (including the RJ45 jack).
6. Bus Interface: A high speed I/O expansion bus.
7. Real-time network: SPI (Serial Peripheral Interface).
8. Multiple timers, counters, PWM, interrupt inputs, etc.

4.3 Motion Controller Hardware

A motion controller is the most important element in a motion control system. Next to choosing a proper motor, the selection or design of a motion controller is the designer's most important decision. The fundamental function of a controller is to compare two signals: the command signal from the microprocessor and the position feedback signal from an encoder, resolver or tachometer. The position feedback signal is subtracted from the reference position to provide a following error which is converted by a digital-to-analog converter (DAC) to analog voltage for the servo amplifier. The controller's prime duty is to minimize the position error without causing system instability. With an appropriate motion controller in place, the designer can focus on stabilizing and programming the system.

4.3.1 Motion Controller Design Options

The decision to select an appropriate motion control scheme depended on a number of factors. The options considered are as follows;

1. JStick as stand-alone motion controller.
2. Use auxiliary hardware in conjunction with JStick for motion control.
3. COTS motion controller board; Interface an off-the-shelf motion control board to JStick.
4. COTS motion control chip; design a motion controller board based on a COTS motion control chip.

4.3.2 Motion Controller Board Design Criteria

The following features were considered for the motion controller board design:

1. A programmable motion controller chip with a compensator (at least a PID filter), a sampling rate of at least 1 KHz for high performance (Bellini et al., 2003), motion profiling, quadrature decoding and a real-time means of populating an integrated buffer with motion set-points from the JStick host. The chip should also be capable of providing encoder readings to allow monitoring or adaptive control.
2. A Digital to Analog (DAC) chip, which easily interfaces with the motion control chip and provides at least a 12-bit resolution and a voltage output range of +/- 10 V. It is worth noting here

that the chip timing parameters should be compatible with that of the motion control chip.

3. An encoder receiver chip (if the motion controller needs one) and circuitry for filtering out noise. Encoder signals are very susceptible to noise, especially single line encoders. This is compounded by long transmission lines and the use of relatively cheap encoder technology like totem-pole and open-collector types.
4. Digital I/O with interrupts for enable-signal on motor driver (amplifier) and mechanical switches such as limit and home position switches.
5. Logic chips such as flip-flops for buffering signals and signal inverters.
6. Power converter chip for devices requiring power levels which are unavailable on the JStick.

4.4 Motion Controller Chip Selection

There are a number of commercial motion controller chips on the market. A few of these were identified as potential candidates for the motion controller board design. The National Semiconductor's LM628 (LM628/LM629, 2003) was eventually selected based on the criteria outlined above, including its low price, simplicity, and proven capabilities.

5. Motion Controller Board Design

This section focuses on the hardware design of the motion controller board for the IMC node. The schematic of the board is shown in Figure 4. The hardware architecture is a multi-tier system with independent controllers for each joint of the robot.

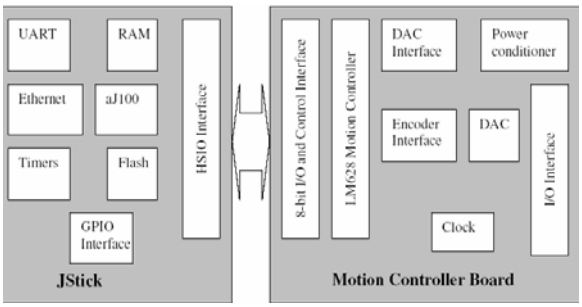

Figure 4: Peripheral Motion Controller Board Architecture

5.1 JStick's Peripheral Interface Signals

In order to interface this board with peripheral I/O devices, JStick uses a synchronous, memory-mapped High Speed I/O Interface (HSIO). The HSIO provides byte-wide (8-bit) data, twelve address bits, read and write strobes, two chip selects, 3.3 VDC power and ground. More chip selects can be easily decoded from the 12-bit address. HSIO signals interface directly to 3.3V TTL and CMOS or 5V TTL devices (Systronix, 2001). In order to configure the HSIO for a peripheral device one needs to properly select timing parameters to match the timing specifications of the peripheral device. Fortunately, the bus timing can be varied to support most peripheral speeds. The bus timing is controlled by three features; a Phase Locked Loop (PLL), the aJile CPU external bus interface settings and the HSIO memory mapped address bits [2].

5.2 Clocking

The hardware architecture provides two means of clocking the LM628 chip; static clocking and dynamic clocking. There is an onboard 6 MHz clock which provides very accurate clocking for the chip. The clock can be enabled or disabled in software but the output frequency is fixed.

5.3 LM628 DAC Output

The LM628 precision motion controller outputs data to a DAC (Digital to Analog Converter) on its ports DAC0-DAC7. Its output port can be configured for either a latched 8-bit parallel output or a multiplexed 12-bit output. While the 8-bit output can be directly connected to non-input-latching DAC (Digital to Analog Converter), the 12-bit output has to be demultiplexed using an external 6-bit latch. The IMC motion controller board uses the 12-bit output mode for better resolution.

5.4 Servlet Technology

Servlets are reusable Java applications which run on response/request-oriented web server. The server loads and executes the servlets, which accept zero or more requests from clients and return data to them. Functionally, they are similar to CGI scripts, but more platform-independent. A few of the many applications of servlets include the following;

1. Servlets can process data posted over Https using an HTML form.
2. Servlets can accommodate multiple requests concurrently to allow collaboration between multiple users.

Servlets can forward requests to other servers and servlets in order to balance load among servers or partition a single logical service over several servlets.

6. Results

System evaluation (and fine-tuning) is a very broad exercise which could pass for another major research projects. Distributed reconfigurable controller is evaluated based on the following criteria:

1. Sampling Time and Communication Latency
2. Block Processing Time and Real-Timeliness
3. Synchronicity
4. Positioning Accuracy
5. Architectural Flexibility.

6.1 Sampling Time and Communication Latency

The PID motion controller chip (LM628) on the IMC performs high speed trajectory generation (including ramping and slewing) at a maximum sampling speed of 341µs. For jitter and processing time, the maximum latency is approximately 4 µs.

6.2 Block Processing Time and Real-Timeliness

The interpolator runs on a computing platform (JStick) dedicated for real-time coordination. The JStick platform executes real-time threads in a cyclic deterministic manner. Thread context switch takes only 1 µs, thereby providing acceptable real-time outputs. ISO G-code programs are loaded to the interpolator's flash memory by Ethernet from a GUI program running on a regular PC. The JStick's flash speed is 90 ns and its execution speed is 15 Mega bytecodes/s. To measure the average block processing speed, G1 and G2 codes are executed on the platform. An average time of 1.5 ms is obtained for 3-axis linear interpolation, and 2.0 ms for 2-axis circular interpolation. However, the block processing time is constrained by the hardware limitation of the motion controller chip to 10 ms as mentioned in the previous section.

6.3 Synchronicity

Synchronization is a critical issue with decentralized controllers. The worst case delay, as shown in Figure 5 is about 1 ms, and over 20% of the data fell in this region. In the next, the clock Synchronization scheme developed for the architecture is implemented. The worst case delay fell appreciably to 0.1 ms (Figure 6). For even finer synchronization, the real-time coordinator is hardwired to the IMC controllers by an interrupt line. The delay results are illustrated in figure 7. Delays are highly repeatable in this case. This illustrates the versatility of the architecture.

.

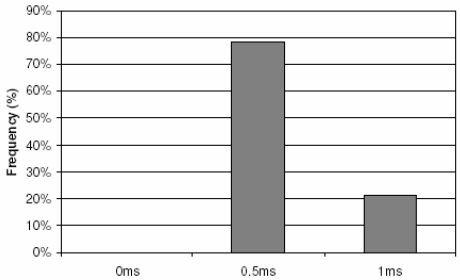

Figure 5: Timing Variations-Uncompensated Delays

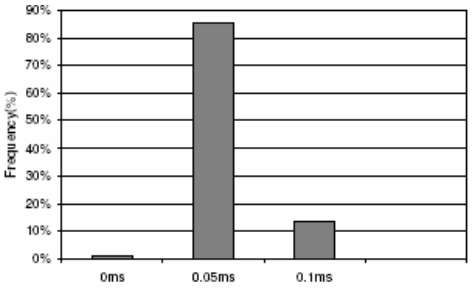

Figure 6: Timing Variations- Compensated Delays

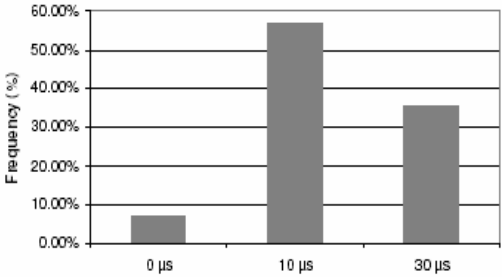

Figure 7: Timing Variation- Synchronization by Interrupts

6.4 Positioning Accuracy

Some position measurements are presented below to illustrate the performance of the controller on a 3-axis machine. Figure 8 shows a zigzag pattern move realized with a feed-rate of 900 mm/s. Figure 9 illustrates circular interpolation with a feed-rate of 350 mm/s and a radius of 5 mm (1440 counts). The trajectory errors associated with a 1-mm radius (288 counts), 5-mm radius and 25-mm radius circular paths are shown in figure 10 to 12. The error patterns are very similar and bound by a maximum of about 11 counts. Two linear and one circular interpolations followed by another linear interpolation are executed in one trajectory path in Figure 13.

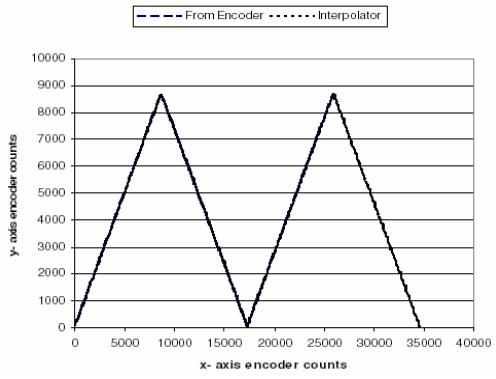

Figure 8: Linear Trajectory

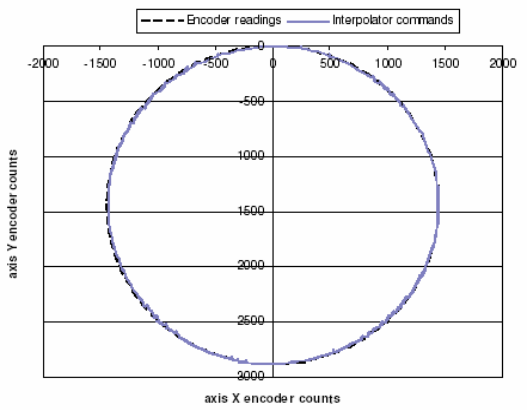

Figure 9: Circular Trajectory – 5-mm radius

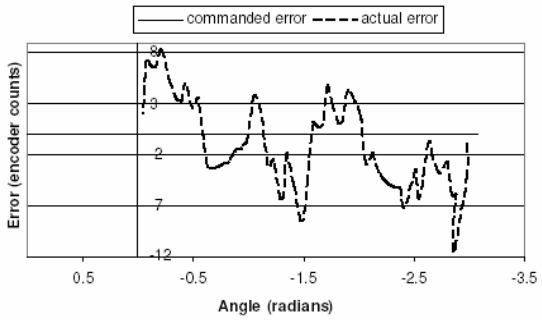

Figure 10: Radial Error – 1-mm radius

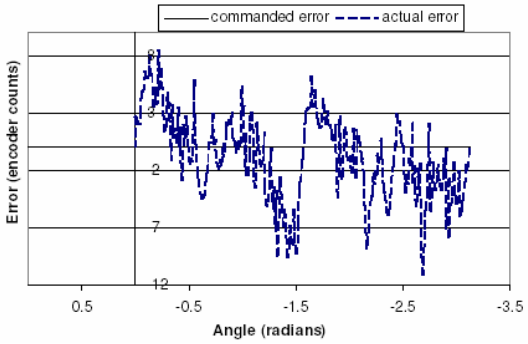

Figure 11: Radial Error – 5-mm radius

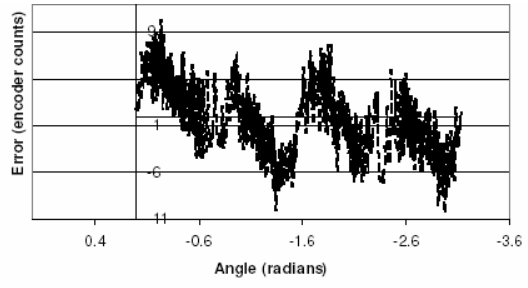

Figure 12: Radial Error – 25-mm radius

The error patterns are very similar and bound by a maximum of about 11 counts. Dry friction in the mechanical drives and interpolation approximations contribute significantly to these errors. Blended moves are also very acceptable. In Figure 13, two linear and one circular interpolations followed by another linear interpolation are executed in one trajectory path.

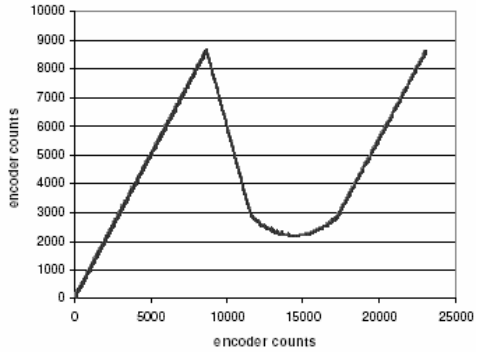

Figure 13: Combined Linear and Circular Paths

6.5 Architectural Flexibility

By virtue of the intelligent communication protocols and the modular nature of our hardware and software architecture, flexibility is greatly enhanced. When machine (or workstation) configurations such as the number of axes need to be scaled, control modules are added to the network to match the number of axes. The protocols automatically configure the network for such changes. The modular nature of the software architecture makes it easy to add modules to accommodate changes. For example, through software interfacing new algorithms for kinematics or interpolation can be readily integrated. Figure 14 illustrates the simulation of the guide (slider) movements of a parallel kinematic mechanism on a 3-axis table.

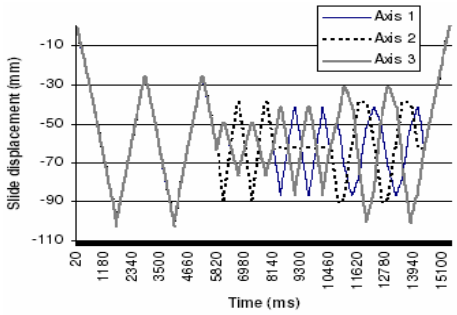

Figure 14: Tripod Slider Displacements

7. Conclusion:

The main contributions of the present work are as follows:

1. An Ethernet-based real-time communication architecture with implicit clock synchronization. The technology also enables the controller subcomponent design to incorporate embedded web-servers for remote monitoring and system configuration.
2. The development and demonstration of a protocol for automatic configuration (i.e., PnP) of controllers and other embedded shop floor devices.
3. The design and implementation of a medium-performance flexible controller incorporating the above on a homogenous Java software and hardware processor environment.

Table 2: Performance Evaluation Results

Evaluation Criteria	**Value**
Maximum servo loop cycle	341 µs
Communication latency	4 µs
Block Processing	3-axis linear interpolation 2-axis circular interpolation
Synchronicity	Synchronization through network: maximum jitter = 1µs. Synchronization by interrupt line: maximum jitter = 30 µs
Positional Accuracy	Max. radial error on 25 mm circle: 0.03 mm (BLU = 0.003 mm)
Architectural flexibility	Easy to add/remove axis Quick adaptation to different mechanical platforms.

References:

[1] aJile-100™ Manual version 2.1, aJile Systems, Inc., 6 December, 2001.

[2] JStick (Real-time Native Java™ Network Module), Systronix®, 939 Edison Street, Salt Lake City, Utah, USA 84111, www.systronix.com, www.jstick.com.

[3] Rodney Atta-Konadu, Sherman Y. T. Lang, Chris Zhang and Peter Orban, "Design of a Robot Control Architecture", Proceedings of the IEEE International Conference on Mechatronics & Automation Niagara Falls, Canada, July 2005, pp:1363-1368.

[4] Birla S., D. Faulker, J. Michaloski, S. Sorenson, G. Weinert and J. Yen, "Reconfigurable Machine Controllers using the OMAC API", Proceedings of the CIRP 1st International Conference on Reconfigurable Manufacturing , Ann Arbor, MI - May 01, 2001.

[5] Eve Coste-Maniere, and Reid Simmons, "Architecture, the Backbone of Robotic Systems", Proceedings of the 2000 IEEE International Conference on Robotics & Automation San Francisco, CA, volume 1, April 2000, pp: 67-72.

[6] Divelbiss A. W. and J. T. Wen, "A Path Space Approach to Nonholonomic Motion Planning in the Presence of Obstacles", IEEE Transactions on Robotics and Automation, Vol. 13, No. 3, June 1997, pp. 443-51.

[7] Feng-Li L., J. R. Moyne and D. M. Tilbury, "Implementation of Networked Machine Tools in Reconfigurable Manufacturing Systems", Proceedings of the 2000 Japan-USA Symposium on Flexible Automation, Ann Arbor, MI, July 2000.

[8] Guttman E., "Autoconfiguration for IP Networking: Enabling Local Communication", IEEE Internet Computing, May-June 2001, pp. 81-88.

[9] Keum-Shik Hong, Jeom-Goo Kim, Chang-Do Huh, Kyung-Hyun Choi, and Suk Lee, "A PC-Based Open Robot Control System: PC-ORC", Robotics and Computer Integrated Manufacturing 17 (2001) 1901–1906.

[10] James J. and McClain R., "Tools and Techniques for Evaluating Control Architecture", Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design, Kohala Coast-Island of Hawai'i, Hawai'i, USA, August 22-27, 1999.

[11] Mitch Pryor, Chetan Kapoor, Rich Hooper, and Delbert Tesar, "A Reusable Software Architecture for Manual Controller Integration", Proceedings of the 1997 IEEE international Conference on Robotics and Automation Albuquerque, New Mexico - April 1997, pp: 3583-3588.

[12] Kim K. H., Im C. and Prasad A., "Realization of a Distributed OS Component for Internal Clock Synchronization in a LAN Environment", Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Washington, D.C., April. 2002, pp. 263-270.

[13] Lee C. J. and C. Mavroidis, PC-Based Control of Robotic and Mechatronic Systems Under MS-Windows NT Workstation" IEE/ASME Transactions on Mechatronics, Vol. 6, No. 3, 2001, pp. 311-321.