RISC-based architectures for multiple robot systems

G Jimenez, J L Sevillano, A Civit-Balcells, F Diaz and A Civit-Breu outline several multi-layered approaches to multiple robot system control

Several approaches to multiple robot system control are discussed. In order to simplify the study a multilayered model is proposed: a control layer which directly acts on the dynamics of the manipulators, a coordination/ communication layer which makes all the manipulators work together and a programming layer which interfaces with the user. For the first layer two architectural alternatives are studied: a centralized single processor system and a distributed multiprocessor with static task assignment. For the second case an implementation based on the i960 family of RISC processors is introduced. For the second layer three possibilities are considered: serial interface, parallel bus and local area network. The latter is carefully studied and a low cost alternative to the standard deterministic network MAP is introduced. This cell network is based on the CSMA/DCR protocol implemented on the i82596 coprocessor. Two alternatives are discussed for the programming layer: a parallel programming language based on a scene approach and a C extended language used to program elementary tasks in a robot independent way coupled with an intelligent scheduler used to assign these tasks to the robot arms at run time.

multiple robots multiprocessor control systems RISC LANs intelligent task scheduling

As industrial processes grow more complex and competitiveness increases the use of multiple robots in work cells (which also include sensors, feeders, machine tools etc.) is becoming more common. The main problem in these systems is the precise synchronization of the elements that make up the cell. Thus the robot arms must be coordinated not only among themselves but also with the information that comes from vision and other sensors and with the rest of the machines in the system. The controllers that are currently used for robots do not include facilities to perform these functions. This is mainly caused because these controllers are designed for single robot systems or very lightly coordinated systems. Although I/O signals are readily accesible from user programs they are wholly insufficient if any coordination that requires a certain ammount of data communication is necessary. It is a fact, and in our opinion one element that strongly opposes robot growth, that many controllers only permit the programming of a robot in a machine-specific language. The situation is so bad that is not uncommon that the same manufacturer uses one language for some of its robots and a different one for others[1].

Turnkey solutions[2] include the use of serial RS232 (or similar) channels usually available in robots to implement the coordination of complex systems. This architecture may not have the bandwidth required to support the coordination traffic in the cell but, what is more important, it limits the design of flexible systems which could be used in many different applications.

These reasons have led several universities and research centres to develop controllers that allow the implementation of sophisticated multiple robot systems[2,3]. An example of these is the Robotic Instruction Processing System (RIPS) from the University of California, Santa Barbara (Figure 1).

In general the implementation of a multiple robot system requires:

- The design of new controllers that permit a better monitoring of robots as well as effective dynamic control which improves system performance.
- The development of coordination and communication systems between the individual robot controllers. Several altematives can be considered according to the required degree of coordination. Among these are:
  - Serial communication. This is the simplest alternative and should be used whenever its performance is acceptable
  - Parallel bus interface (e.g. IEEE 488, VME, Multibus II)
  - Network interface (e.g. MAP, Ethernet)
- The development of software packages that manage the set of robots and their interface to the rest of the cell in an efficient way.

Arquitectura de Computadores, Facultad de Informatica, Universidad de Sevilla, Avda, Rei·a Mercedes s/n, 41012, Sevilla, Spain
Paper received: 2 March 1992

0141-9331/92/040177-09

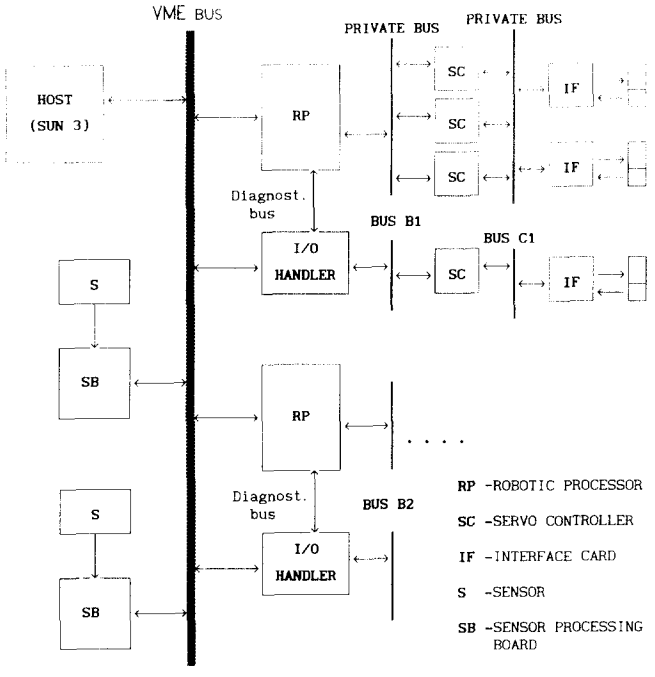


Figure 1

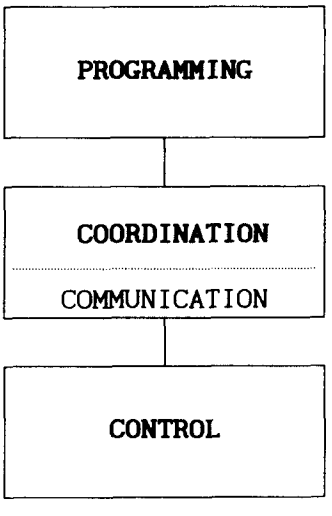


Figure 2

It is convenient to divide the study of multiple robot systems into several layers as is done in other areas of computer science[4,5]. The main advantage of this approach is that every layer can be studied and designed separately from the others. Figure 2 shows our division of a multiple robot system. In this paper each layer is studied considering its interfaces with the ones above and below it.

CONTROLLERS FOR MULTIPLE ROBOT SYSTEMS

As robots have been assigned more difficult tasks and the performance constraints have grown their control units have become increasingly sophisticated[6]. The first industrial robots were controlled by simple mono-processors and their dynamic behaviour and user interface were very poor. Controllers have evolved toward multiprocessors as, for example, Silma's Adept One, which is controlled by two M68000[7], and those developed at Stanford[8], the University of Pennsylvania[9], MIT[10,11], Brown University[12], IBM[13] etc. The main motive for this evolution has been the possibility of increasing the computational performance of the controller and thus more powerful algorithms can be implemented in them. Nowadays the trend is to use advanced processors (mainly RISC[14,15], DSP[16-19] or specific VLSI robotic processors[3,20]) in these controllers so that the most complex control policies can be implemented in low cost controllers.

In general multiple robot controllers are extensions of single robot controllers but with a greater throughput and I/O capabilities. These systems can be distributed or centralized.

Centralized multiple robot control systems

In this kind of system the control algorithms for all the robot arms are implemented in a single processing system, mono or multiprocessor, without a fixed assignment of processors to the arms. The centralized control systems can be divided into those that use a single global control algorithm and those that have separate processes for the control of each arm. This last case is, from the software point of view, equivalent to a distributed system but with the maximum number of arms limited by the computing capabilities of the processing system. If a single CPU is used then an upper limit on the number of arms is the quotient of the required sampling period and the time taken by the processor to implement the control algorithm for each arm. If a multiprocessor is used the load balancing algorithms can represent an important overhead and if the system distributes the processing dynamically among CPUs then execution times are difficult to predict and thus it is difficult (or impossible) to use this configuration in real-time controllers. As a consequence, global control with a single processor or several processors with a static task assignment is the only option where centralized control becomes attractive.

We define a completely centralized control as a system where all the arms are considered as a single robot whose degree of freedom is the sum of those of the individual arms. A good term for this type of system is multiple arm robot. The controller for this robot can be, in principle, like that of a traditional single arm but with two important characteristics: it must be capable of interfacing with a great number of actuators and it must be powerful enough to run both the control and the coordination algorithms. The first requirement implies an important I/O handling capability while the second imposes a lower limit on the processing throughput of the system for a certain number of arms.

A system where these requirements are fulfilled is described in Reference 21. This system, which is currently under development, is a heterogeneous RISC multiprocessor. An i860 with 80 MFLOPS single precision peak performance is used to implement the control and coordination algorithm. This processor is not intended for real-time applications and, thus, its I/O capabilities are very poor. A good companion for this processor is the high performance I/O oriented i960CA superscalar microcontroller with very short interrupt response times. This system has been simulated running multiple copies of a single arm adaptive controller[22] and it has been shown

that, in this way, up to eight SCARA arms (4 DOF) can be easily controlled. In the test a sinusoidal trajectory with a 1 ms sampling period has been used. The vector capabilities of the i860 processor are well suited for this algorithm which does not use matrix inversion and where the inner loop is a matrix product in which the i860 can reach its peak FP performance[23]. An interesting fact is that a C vectorizer is not available for this processor (at least to the authors' knowledge) and this requires writing part of the code in Fortran (using the VAST-2 Fortran Vectorizer for the i860[24]) or coding the calls to the vector maths library by hand.

The main difficulty for experimenting with global control is that there are no computationally efficient algorithms available.

In the case of completely centralized control there is little independence between the layers in Figure 2. Coordination and control can be considered as a single layer and the communication sublayer is unnecessary.

Distributed multiple robot control systems

In these systems there is a controller for each arm. One of the advantages of these systems is that they are easily expandable. From the hardware point of view adding a robot only requires the addition of a new controller. From the software side it is usually a very simple task also. This type of system permits the implementation of a centralized or a distributed control policy. In the latter case coordination requires the transmission of the status of every controller to all the others. As we will see later centralized coordination is easier to implement, although the hardware that we will introduce is equally suited for the distributed approach.

The multiple arm controllers can reuse an important part of the design of traditional single arm controllers, but two important aspects must be modified:

- Control software should allow supervision by the coordination layer
- The hardware implementation should include an adequate communication channel.

The first requirement makes the implementation of a multiprocessor for the control of each arm attractive (as in Figure 1). In our case we have adopted a biprocessor structure. One of the processors implements the control algorithm while the other handles the communications with the coordination layer. This processor interprets the received messages and translates trajectories into a form that can be easily used by the control processor.

The kind of algorithms that should be implemented is an important factor in the selection of the processors for the controller. We decided that the control processor should be able to execute advanced algorithms such as adaptive, learning control or mixed techniques[21]. This led us to choose RISC processors that were specially suited for embedded applications such as those in the i960 family. It is important to point out that the task of the controllers requires 'environment sensitive' processors. This means that interrupt latency and processing overhead have to be kept to a minimum. A microcoded CISC requires several cycles to complete its current instruction and save its environment. The RISC philosophy of very short instructions and a larged framed register set makes it possible to finish the instruction and save the state in one or, at least, very few cycles. In an early design the control processor was implemented with an i960KB and an i960CA was used to handle communications with the coordinator. The controllers were connected to the coordinator through a MultibusII (Figure 4). The system was used to control four-axis SCARA robots.

The i960CA superscalar processor is optimized for data communication and is very well suited to control the MBII interface. This RISC is interesting mainly for two reasons: its great processing power and its quick interrupt mechanism. The processing power is due to a simple but highly efficient instruction set together with a micro-architecture that permits up to three instructions/clock (up to 120 native MIPS). An internal 1k instruction cache together with a 1k internal RAM from which four 32-bit words can be read in a single cycle contribute to the speed of this processor.

The i960KB is not superscalar and thus it is limited to one instruction/clock (up to 33 native MIPS). The reason for selecting this processor for control is that it includes an internal FP unit which is essential for some control algorithms. The FP instructions as well as the memory references can be overlapped with other instructions, thus contributing to program speed up.

For the implementation of our system two commercially available boards have been used:

- XCRC/KB, a modification of an Intel i960KB board specially adapted to SCARA robot control.
- MIB II 960/110 (RISC Development Board), an MBII board based on the 960CA with a prototyping area for user expansion.

Both processors have to communicate in an efficient form. Our approach is to use a dual-port memory for this purpose. In this memory the i960KB leaves trajectory data and the i960CA checks for possible conflicts. All this must be done in real time. The arm controller is shown in Figure 3.

If the i960CA has to run with zero wait states (not really necessary in our application) the memory access time must be under 40 ns. Dual port 32k memories with 35 ns access times are currently available[25] and thus interprocessor communications are very simple from the hardware side. These communications are completed by the use of interrupts.

This system has been designed for centralized coordination where the coordinator sends, among other messages, trajectory information. This information is received by the arm communication controller (i960CA) and passed to the arm dynamic controller (i960KB). In this case there are six types of messages between the controllers[26].

From KB to CA:

1) STOP_D: Tells the communications controller that the trajectory cannot be completed (usually due to an obstacle interrupt).
2) TRAJECTORY_END: Tells the communications controller that the trajectory has ended.
3) TRAJECTORY_POINT: Tells the communications controller that a trajectory point has been calculated and must be checked for possible conflicts.

From CA to KB:

4) STOP_C: Tells the dynamic controller to stop the arm. This can be due to an emergency stop (sent as a

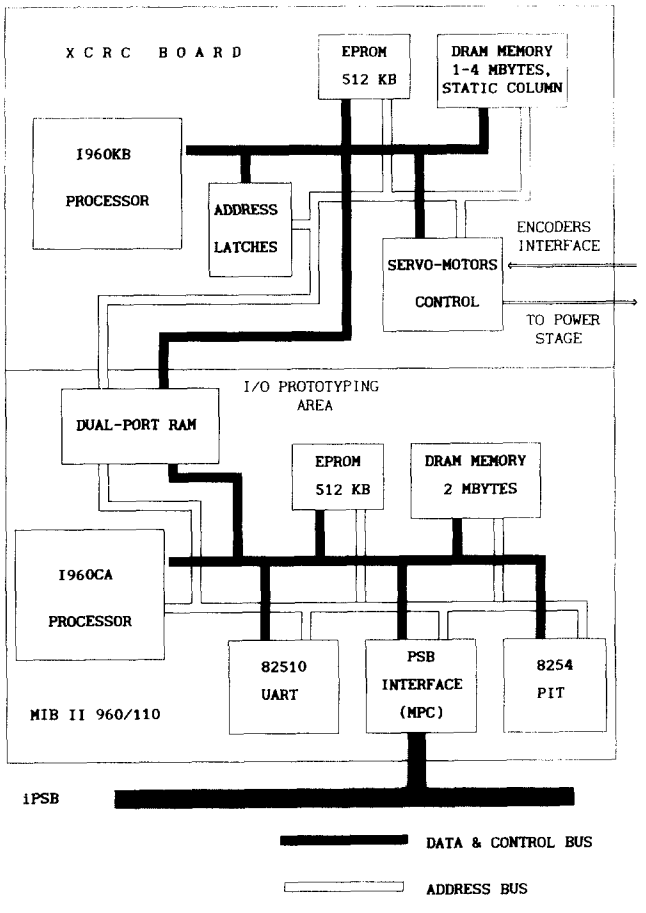


Figure 3

broadcast message) or the detection by the communication controller of the entrance into an unavailable interference zone.

5) TRAJECTORY: Tells the dynamic controller that a trajectory description is available in a mail box in shared memory.
6) CONTINUE: Tells the dynamic controller to continue a trajectory that was stopped by STOP_C.

These messages can be implemented through structures in shared memory and interrupts.

The overall controller process is as follows. Once a trajectory is received by the KB it starts calculating points and leaving them in shared memory. For each point a type three interrupt is generated, telling the CA to check for possible conflicts. The calculation/checking/control is pipelined i.e. when the KB is calculating point *i*, the CA is checking point *i*-1 and the KB control is acting over point *i*-offset. The value of the offset variable depends on the maximum speed, the inertial parameters of the robot and the characteristics of the actuators (the robot cannot stop instantaneously). The points involved in the pipeline are stored in a circular queue of length offset +2.

COORDINATION

Several degrees of coordination are possible[27]. Applications usually require a certain degree of coordination but this degree can also be limited by system characteristics. We now consider two main cases: tight and loose coordination.

Tight coordination

A tightly coordinated system requires that the kinematic variables of the arms (position, velocity and acceleration) should be coordinated at every instant. This means, as stated, that the system can be considered as a multiple-arm robot. In such a system a trajectory is defined as the temporal evolution of the coordinates of all the arms in the system. The control should use a centralized approach and the processing system should be able to handle matrix operations, where the dimension of the matrices is the sum of the degrees of freedom in the system.

The biggest advantage of tight coordination is that it permits adequate modelling of all the dynamic couplings in the system and, thus, the flexibility is a maximum. As an example, two arms could manipulate objects by establishing a constraint in the system matrix or this matrix could be used to determine the physical interference of any arms.

The problem with this type of coordination is the huge amount of calculations that must be carried out in real time. One must always bear in mind that in the robot environment the only processors that can realistically be used are single or clustered microprocessors.

Loose coordination

In this case the object of coordination is to prevent physical interference between the arms, thus permitting their cooperation by sharing resources (tools or working zones). In our approach the global working zone (i.e. the total of all the robot working zones) is divided into logical subzones. These subzones can change with time but we impose the condition that this change will only take place when all the robots are stopped. All zones that may be used are defined before the system starts operating and later they are activated or deactivated at run time.

There are two main types of shared resource management:

- Exclusive use: In this case resources can only be used by one robot at a time. Thus a robot would not be allowed to enter a zone that is already owned by another. Of course, it would also be impossible for a robot to use a tool if it was already being used by another.
- Simultaneous use with restrictions: In this case several robots can share a zone but only one of them at a time can move inside it. Tools are, of course, always exclusive resources.

This division can be simplified if we redefine the concept of resource and consider that the movements inside a zone can be treated as one.

Each of the cases has associated problems. In the first all the necessary conditions for deadlock[28] hold so it is necessary to consider this possibility and handle it adequately. In the second case logical deadlock is not possible because if a robot that has the ownership of movement in its zone wants to enter another and it is not granted the resource it will stop at the border of the zones and liberate the resource that it already owned. Thus it is not necessary for any robot to hold two movement resources at a time and so one of the conditions for deadlock does not hold. Of course this system can block for other reasons:

- Tool deadlock: Tools are still exclusive use shared resources.
- Geometric deadlock: A robot that has movement permission may not be able to move using the desired trajectory if there are stationary robots that interfere with it. In this case an alternative trajectory should be calculated. This approach is suitable for most assembly tasks but is completely inadequate for welding or gluing.

In loose coordination robots are considered as independent units and thus it is necessary to establish which information they must share for the synchronization and coordination of the system. This information depends on the coordination structure that is used. There are two possibilities:

- Centralized coordination, in which every element in the system demands information from a central coordinator which makes the decisions on resource allocation and global synchronization.
- Distributed coordination, in which every element needs to know the information about those that may interfere with it.

In any case communications are through messages that include trajectories and coordination traffic.

Even though both options can be justified, distributed coordination is more complex. The causes for this are:

- It requires higher coordination traffic. In case of conflict every robot asks its neighbours for their status.
- It is more difficult to alter in run time the tasks that have been sent to robots.
- The cost and complexity are higher because functions that would otherwise be centralized have to be repeated in all elements.

In any case, a system with enough communication bandwidth can easily support distributed coordination. From now on we will limit the discussion to the easier centralized coordination scheme. It is important to note that having a central coordinator does not mean that control cannot be distributed in the system. Both layers (Figure 2) are independent. There could also be different levels of centralization according to the functions assigned to the coordinator.

For all these reasons we propose a system based on centralized loose coordination which considers movement in the interference zones as a shared resource. Thus, in this system, the coordinator may have to compute an alternative trajectory if the originally programmed one would cause a collision. Experience from other areas of computer science (like operating systems and multiprocessors) in resource sharing can be applied in this type of system. Communications will be handled by a small set of messages[21]. These include:

From robot controller to coordinator:

- Request to move in a shared zone (includes zone selection).
- Position information. This message is transmitted by robot controllers when they stop inside a shared zone in a position different from the end point of their assigned trajectory.
- Zone release. Robots send this message when they leave a conflict zone.
- Trajectory end.
- Alarm.

From coordinator to robot controller:

- Trajectory description and start command. Sends the definition of a trajectory that should be immediately executed.
- Sleep. Tells the controller that the zone movement requested is not available and thus the robot should stop at its border. If a robot is already in a zone and a trajectory is sent to it by the coordinator this implies that it has permission to move within that zone.
- Wake up. Tells a robot that has received a stop message that it can continue its trajectory.
- Shared zone grant. Tells a robot that it can move in the requested zone.
- Active zone grant. This message is used to change the active zones.
- Alarm.

COMMUNICATIONS

In this part of the paper we will study the three alternatives presented before for multiple robot communications.

Serial interface

In the simplest systems serial lines can be used for the robots to communicate with the central coordinator[2]. Its simple implementation using commercial controllers and its low cost make this type of system the most common in current industrial applications. Two types of organization can be considered:

- Star topology. In this approach a dedicated serial line connects each robot to the coordinator. This is the most common case and is suitable for systems with a small number of arms and low speed and response time requirements.
- Bus topology. This is much less common because not all serial interface devices permit this configuration. The bandwidth of the serial channel (usually around 10 kb s^{-1}) is shared among all the devices and thus this approach is not suitable for any type of multiple robot system.

Parallel interface

Nowadays very high performance multiple robot systems use parallel buses. Although the most popular is VME (Figure 1), MultibusII is, in our opinion, the best alternative for this type of message based application[29]. MultibusII has a separate address space for message passing and the standard interface chip for this bus (MPC, message passing coprocessor) handles communicatins using this space with very little assistance from the processor. Other important characteristics of this bus are its synchronous nature and its high bandwidth (40 Mbyte s^{-1}). An interesting possibility is that several operating systems can be run on a single bus. There is a standard message passing protocol (Intel transport protocol) that is used by all system software in the multibus system and thus messages can be easily passed between processors running different operating systems, or kernels[30, 31].

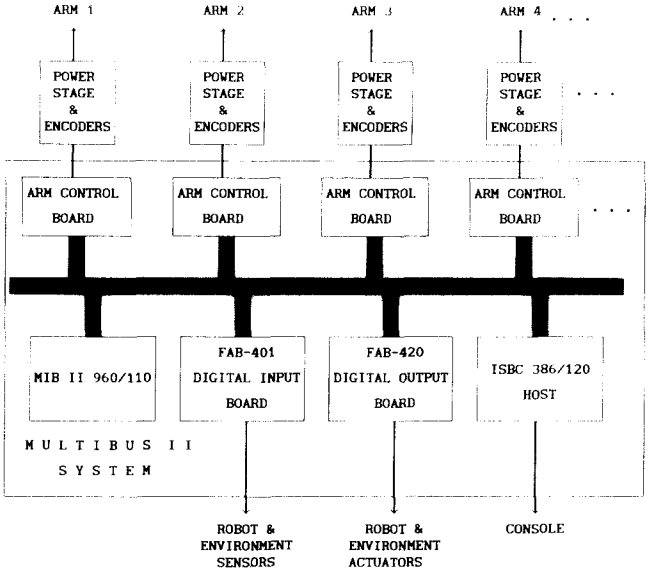


Figure 4

A Multibus II based multiple robot system is shown in Figure 4 and described extensively in Reference 26. This system uses the boards that have already been discussed. Although the system was originally designed for SCARA robots there is no reason why it could not be used with other types of system.

Unsolicited packets (intelligent interrupts) are used to transmit coordination messages. These messages have 20 bytes for user information, which is more than enough for our purpose. Trajectories have to be sent using solicited messages due to the large amount of data that they require (type, parameters, initial and final point, linkage conditions etc.). For information from sensors and I/O devices the type of message would depend on the amount of information that must be included in the message. For alarms, unsolicited broadcast messages are used.

Network interface

The local area network that is commonly used for industrial applications is the standard IEEE 802.4 'Token Bus' which is the protocol specified in the manufacturing automation protocol (MAP)[32]. However, its cost and complexity make it very difficult to use in a multiple robot system. The simplified version MiniMAP, in which the delay is limited by eliminating the high level protocol layers, is still inadequate: the problem is that the underlying token bus access protocol is too slow in itself[33]. Another widely used network, Ethernet[34], is non-deterministic and therefore it is not suited for these applications.

We consider that a good solution is the use of a new access protocol (CSMA/DCR deterministic collision resolution) which is implemented on the Intel i82596 family of LAN coprocessors[35]. This protocol is equivalent to that of Ethernet (CSMA/CD IEEE 802.3) but with a different approach to collision resolution. In Ethernet a probabilistic algorithm (exponential backoff) is used to compute the retransmission delays, while in CSMA/DCR the collision resolution period is divided into slots (units of time) and each station is assigned one of them (Figure 5). After a collision is detected, and the jam period is over, every station transmits only in its corresponding slot. This means that stations transmit in a predefined order in a way that is very similar to that used in token passing nets (Figure 5). Of course, this method is not optimal in the sense that mean delays are higher than in the probabilistic Ethernet approach. What happens is that these delays are upper bounded because, in the worst case, collision resolution periods follow one another with no idle period between them.

In a collision the time between the first transmission and the moment when all stations sense the channel idle is $\gamma + \tau, \gamma = 2\tau + jam + \varepsilon$, with τ representing the end to end propagation delay in the bus and ε the estimated collision detection time in every station[36]. Thus the maximum delay is:

$$D_{max} = N\left(\frac{X}{R} + \tau + IFS\right) + IFS + \gamma + \tau$$

where N is the number of stations, X the length of the packets and R the data rate of the network.

This enables calculation of the maximum delay of a message in our system. Also, from a hardware point of view, the great experience with design and support of Ethernet networks can be used with the CSMA/DCR protocol. As a result this low cost, fully deterministic network is a very interesting alternative for multiple robot systems. In order to verify this we must check the maximum delay and required bandwidth for any system configuration. As a worst case we could consider a system made by 10 stations, one coordinator and nine robots connected by a 100 m bus. This is not a realistic multiple robot system nowadays but it can be used as a worst case to see if the network suits our requirements. The longest possible message transmitted by the coordinator is a trajectory, while in the case of a robot it is a point. Supposing this situation we have:

$$D = \left(\frac{X_t}{R} + \tau + IFS\right) + (N-1)\left(\frac{X_p}{R} + \tau + IFS\right) + IFS + \gamma + \tau$$

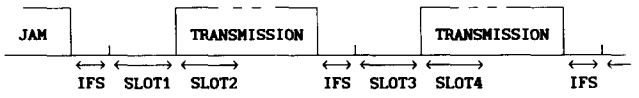
where X_t is the length of a 'trajectory information and start order' while X_p is the length of a 'position information' message. For this last message we need to send:

- 1 byte for message type
- 24 bytes (six degrees of freedom * four bytes) for robot position
- 26 bytes overhead.

This means that $X_p = 51$ bytes. In the case of the 'trajectory information and start order' message we need to send:

- 1 byte for message type
- 1 byte for trajectory type, velocity and continuity information (to inform the controller whether it should stop at the end of the trajectory)
- A field that depends on the type of trajectory and which in the most favourable case would only include the end point (24 bytes). For our system the least favourable case is a five-point interpolated trajectory (120 bytes)
- 26 bytes overhead

Thus $X_t = 148$ bytes and $D_{max} = 604\ \mu s$, with the standard network parameters, which is acceptable since the typical



IFS ≡ Interframe Spacing.

Figure 5

sampling period of robotic systems is about 1 ms. This delay can be substantially reduced by using non-standard parameters. For example, the slot-time must be higher than *Jam* + 2τ, the higher collision fragment. The standard value, 51.2 μs, equals the transmission time of the shortest message (64 bytes excluding the preamble) and it is slightly higher than the maximum *Jam* + 2τ for a 2500 m bus length (49.8 μs). However, a much lower value can be used in a shorter bus where τ is much shorter. Interframe spacing can be reduced to 3.2 μs and the data rate can be increased to 20 Mb s^{-1}.

These considerations have led to a modification in the former controller (Figure 4). The MIB II 960/110 board has been substituted by an EVQT960E20 with an i82596 LAN coprocessor (Figure 6). In Figure 7 the complete system configuration is shown.

The coprocessor communicates with its associated CPU using shared memory[35]. Part of this memory is used as a command list where the CPU writes the commands that should be executed by the LAN coprocessor. In particular, the most common command TRANSMIT includes the destination station, the packet length and a pointer to the actual data to be sent. Commands can be chained in memory and, thus, long data fields can be broken into smaller packets. Also, when the message queue increases, as many buffers as necessary can be assigned in memory. Using adequate software it is easy to order the TRANSMIT commands in the list according to the priority of their associated packet. It is important to know that the i82596 has a 64 byte transmission FIFO and, thus, part of the first packet in the queue may have been read into this structure. This means that if a new packet is generated it cannot be placed in the first position in the queue but at most in the second, regardless of its priority level. This kind of service is called non-preemptive.

PROGRAMMING/USER INTERFACE

The ideas behind the system software for our multiple arm system changed drastically during the project execution. The high level layers have not been implemented yet and thus further changes will probably be implemented in the future.

The first step was the development of a universal compiler for SCARA robots, 'ALFIR'[37], which permitted us to have a single language interface for all the robots in the system. Later commercial controllers proved to be inadequate for the type of coordination required to implement our system. That meant that all arms had to be fitted with a special controller that directly understood a common intermediate language. With this decision ALFIR was not needed any more.

In a later stage the idea for a complete programming environment for the multiple arm system (EMR[38]) was developed. This environment included:

- ALMA: A language for multiple arms that was supposed to be a Pascal-like robot language with parallel capabilities.
- TDL: A trajectory description language that permitted learned and taught trajectories to be mixed and kept in a robot independent intermediate code.
- SDL: Subzone description language. This permits the description of subzone walls that can be later activated or deactivated at run time.

At this moment TDL and a simplified version of SDL are still under development but ALMA is undergoing extensive modification. The causes for this are:

- ALMA is an oriented language in the sense that any task is explicitly assigned to one arm. This makes the programming task difficult because it has to be decided *a priori* what tasks will be assigned to each robot.

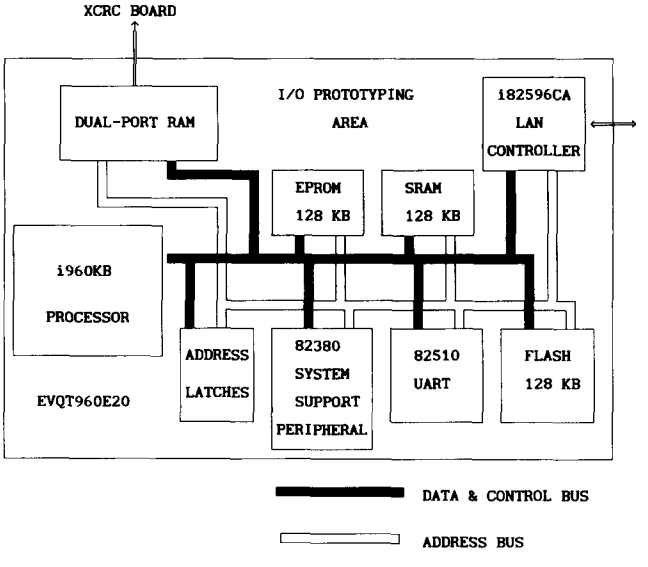


Figure 6

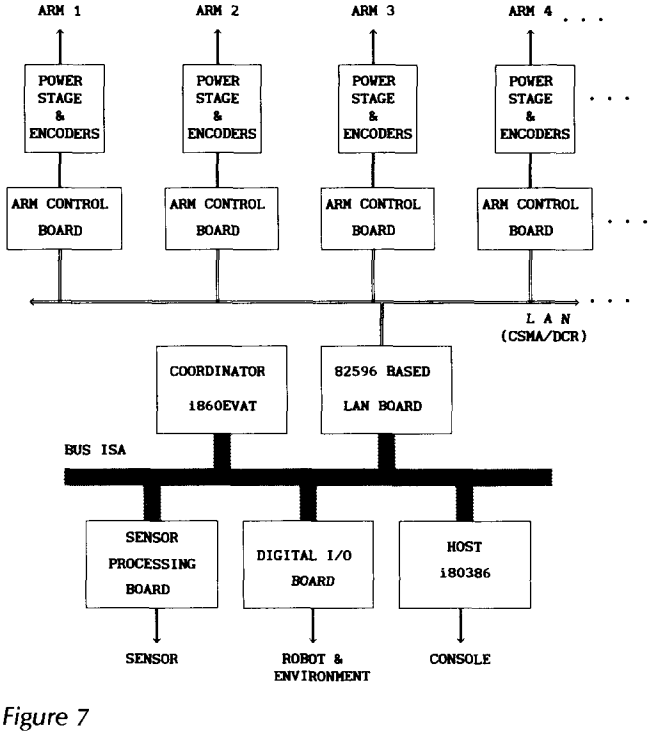


Figure 7

- ALMA includes two possibilities for the description of parallel programs. The first is based in the scene idea where any robot included in a scene when it has finished its task will wait for the others to finish. The second approach is to use elementary tasks and explicitly include dependences in their description.
- ALMA is general while our specific interests relate mainly to assembly. Thus ALMA does not understand about parts and the possibility of waiting for them.
- ALMA is Pascal-like while in our environment C is more common.
- The development of ALMA would require a huge effort and would not bring any significant achievement from a research point of view.

These facts have made us break with our past work and start to develop a new idea in multiple arm programming which replaces the parallel multiple arm compiler with two new tools: a C language library for task description and a multiple arm task scheduler[39]. The C library permits the description of robot independent elementary tasks (tasks that should be performed by a single robot). It also permits the execution of trajectories learned through TDL and the inclusion of task and part prerequisites for starting the task. The scheduler would be the newest and most difficult part of the system since it would be in charge of selecting when a task has to be performed and which robot has to be assigned to the task.

There are many important difficulties for the development of the multiple arm robot scheduler. Among them we will highlight the fact that the robots in the system may have different capabilities and performance. Another important fact is that parts may not come in fixed order, times or positions. These facts, and the huge number of possible alternatives, make the use of many current scheduling tools difficult for this problem. The use of artificial intelligence techniques[40] for the implementation of the scheduler seems to be the only possibility considering the real-time requirements of the problem.

Another approach would be to make the system part driven, in the sense that part descriptions would have associated prerequisites and associated tasks. This approach would be very attractive from a user point of view, making programming quite simple. The problem is that some tasks cannot be associated with a single part but have to be carried out when a set of independent parts has been assembled. Here we say that parts are independent if none of them is a prerequisite to another.

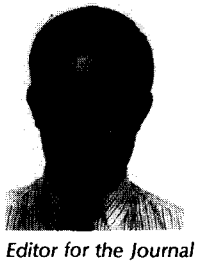
A very important issue is the relation between low level coordination and high level scheduling. The scheduler assigns elementary tasks to arms. These tasks are composed of trajectories. When a trajectory is executed the arm processors continuously monitor it to find whether it will enter an interference subzone. When this happens a message is sent to the coordinator asking for the resource. The coordinator checks whether the zone is available and then checks the original trajectory for any collision with a static robot in the subzone. If the original trajectory led to a collision it is modified to a new collision-free path. Thus the coordinator can send three different kinds of message when an arm asks for the interference zone: 'stop', 'continue' or 'continue with alternative trajectory'. Clearly the result of this interrogation affects task completion times not only of the requesting task but possibly of other tasks in the system. This means that, as the scheduler controls the assignment and execution order of tasks, it should include the coordination policy in its decision making mechanism. In other words, if a task can be assigned to a robot that will not have resource or collision conflicts this is a favourable score for that assignment.

This scheduling mechanism requires calculating robot entering points into collision zones and trajectories before they are actually sent to the robots. What is more important, calculations have to be carried out not only for the final task scheduling but for several other alternatives. If this operation has to be carried out in real time, even with a multiprocessor system, a strong tree-pruning algorithm has to be used to minimize the alternatives and collision detection and alternative trajectory calculations have to be greatly simplified. Simplifying collision detection and alternative trajectory calculation usually means operating with coarser space grids, thus reducing the number of possibilities but, as a side effect, making trajectories possible in the real world impossible for the system. In practice a compromise will be required between high speed calculation and accuracy of representation.

REFERENCES

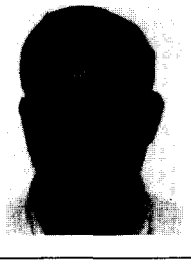
1 **Bonner, S and Shin, K G** 'A comparative study of robot languages' *Computer* (December 1982)
2 **Franke, E** 'Communications and control in robot workcells' Tutorial, Int. Robots & Vision Automation Show and Conference, Detroit, MI (October 1991)
3 **Wang, Y and Butner, S E** 'A new architecture for robot control' Proc. 1987 IEEE Int. Conf. on Robotics and Automation, Raleigh, NC Vol 2 (April 1987) pp 664–670
4 **Day, J D and Zimmermann, H** 'The OSI reference model' *Proc. IEEE* Vol 71 (December 1983) pp 1334–1340
5 **Tanenbaum, A S** *Structured Computer Organization* (3rd edition) Prentice Hall, Englewood Cliffs, NJ (1990)
6 **Graham, J H** 'Special computer architectures for robotics: tutorial and survey' *IEEE Trans. Robotics Autom.* Vol 5 No 5 (October 1989)
7 **Craig, J** *Adaptive Control of Mechanical Manipulators* Addison-Wesley, Reading, MA (1986)
8 **Chen, J B, Fearing, R S, Armstrong, B S and Burdick, J W** 'NYMPH: A multiprocessor for manipulation applications' Proc. 1986 IEEE Int. Conf. on Robotics and Automation, San Francisco, CA Vol 3 (April 1986) pp 1731–1736
9 **Paul R P and Zhang, H** 'Design of a robot force/motion server' Proc. 1986 IEEE Int. Conf. on Robotics and Automation, San Francisco, CA Vol 3 (April 1986) pp 1878–1883
10 **Narasimhan, S *et al.*** 'Implementation of control methodologies on the computational architecture for the Utah/MIT hand' Proc. 1986 IEEE Int. Conf. on Robotics and Automation, San Francisco, CA Vol 3 (April 1986) pp 1884–1889
11 **Narasimhan, S *et al.*** 'Condor: A revised architecture for controlling the Utah/MIT hand' Proc. 1988 IEEE Int. Conf. on Robotics and Automation, Philadelphia, PA Vol 1 (April 1988) pp 446–449
12 **Kazanzides, P *et al.*** 'A multiprocessor system for real-

time robotic control' *Inform. Sci.* Vol 44 (1988) pp 225–247
13 **Korein, J U *et al.*** 'A configurable system for automation programming and control' Proc. 1986 IEEE Int. Conf. on Robotics and Automation, San Francisco, CA Vol 3 (April 1986) pp 1871–1877
14 **Buhler, M *et al.*** 'The Yale real-time distributed control system: XP/DCS version 1.0' 2nd Ann. Workshop on Parallel Computing, Portland, OR (April 1988)
15 **Amaya, C *et al.*** 'Risc Multiprocessor for SCARA robot control' IEEE Workshop on Sensorial . . . , Zaragoza, Spain (1989)
16 **Caselli, S *et al.*** 'Performance evaluation of processor architectures for robotics' Proc. of IEEE COMPEURO'91, Bologna, Italy (May 1991) pp 667–671
17 **Ish-Shalom, J and Kazanzides, P** 'SPARTA: Multiple signal processors for high-performance robot control' *IEEE Trans. Robotics Auto.* Vol 5 No 5 (October 1989)
18 **Kanade, T *et al.*** 'Real-time control of CMU direct-drive arm II using customized inverse dynamic' Proc. 23rd IEEE CDC, Las Vegas, NV (Dec. 1984) pp 1345–1352
19 **Wu, C H and Koch, P N** 'Design of robot joint servo development system' Proc. 24th IEEE CDC, Ft. Lauderdale, FL (December 1985) pp 344–349
20 **Olson, D E *et al.*** 'Systolic architectures for computation of de Jacobian for robot manipulators' in **Graham, J (Ed)** *Computer Architectures for Robotics and Automation*, Gordon and Breach, New York (1987)
21 **Jimenez, G *et al.*** 'Development and implementation of advanced multiple robot controllers' 22nd Int. Symp. on Industrial Robots, Detroit, MI (October 1991)
22 **Lin, Guo and Angeles, J** 'Controller estimation for the adaptive control of robotic manipulators' *IEEE Trans. Robotics Autom.* Vol 5 No 3 (June 1989)
23 *80860 Programmer's Reference Manual* Intel Corp. (1990)
24 *Fortran-860 Compiler and VAST-2 Fortran Vectorizer for the i860 Manuals*
25 *MOS Memory Products* Toshiba (1989)
26 **Jimenez, G *et al.*** 'Risc multiprocessor for multiple Scara robot control' Proc. Int. Symposium Mini and Microcomputers and their Applications, Lugano, Switzerland (June 1990)
27 **Civit, A *et al.*** 'Multiple Robot Coordination using Multiprocesisng Primitives' ISMM Symp. Computer Applications in Design, Simulation and Analysis, New Orleans (March 1990)
28 **Maekawa *et al.*** *Operating Systems, Advanced Concepts* Benjamin Cummins (1987)
29 **Hyde, J** 'The Multibus II bus structure' in **Di Giacomo (Ed)** *Digital Bus Handbook* McGraw-Hill, New York (1990)
30 *MULTIBUS II Bus Architecture Specification Handbook* Intel Corp. (1986)
31 *MULTIBUS II Transport Protocol Specification and Designer's Guide* Intel Corp. (1986)
32 **Kaminski, M A** 'Protocols for communicating in the factory' *IEEE Spectrum* (April 1986) pp 56–62
33 **Shin, K G** 'Real-time communications in a computer-controlled workcell' *IEEE Trans. Robotics Autom.* Vol 7 No 1 (February 1991)
34 **Shoch, J F *et al.*** 'Evolution of the Ethernet local computer network' *Computer* (August 1982) pp 10–27
35 *82596 User's Manual* Intel Corporation (1989)
36 **Tobagi, F A and Hunt, V B** 'Performance analysis of carrier sense multiple access with collision detection' *Comput. Networks* Vol 4 No 5 (October-November 1980) pp 245–259
37 **Diaz, E *et al.*** 'ALFIR: A language for industrial robots' Proc. ISMM Int. Symp. Industrial, Vehicular and Space Applications of Microcomputers, New York (October 1990)
38 **Sevillano, J L *et al.*** 'EMR: An environment for multiple robot programming' 22nd Int. Symp. on Industrial Robots, Detroit, MI (October 1991)
39 **Baker, K R** *Introduction to Sequencing and Scheduling* John Wiley, New York (1974)
40 **Yazdani, M (Ed)** *Artificial Intelligence, Principles and Applications* Chapman and Hall Computing, London (1986)
41 **Ryan, D P** 'Intel 80960: An architecture optimized for embedded control' *IEEE Micro.* Vol 8 No 3 (June 1988)
42 **Janetzky, D and Watson, K S** 'Performance evaluation of the MAP token bus in real time applications' in **Kummerle, K, Tobagi, F A and Limb, J O (Eds)** *Advances in Local Area Networks* IEEE Press (1987)



Gabriel Jiménez Moreno received his Masters in physics (electronics) and his PhD from the University of Sevilla, Spain. After working for several months with Alcatel he was granted a fellowship from the Spanish Science and Technology Commission (CICYT). Currently he is Associate Professor of Computer Architecture in the University of Sevilla. He is the author of various papers and research reports on robotics and computer architecture. He is also Assistant Editor for the Journal of Computer and Software Engineering.

José Luis Sevillano Ramos received his Masters in physics (electronics) from the University of Sevilla, Spain, in 1990. From 1989 to 1991 he was a researcher supported by the Spanish Science and Technology Commission (CICYT). He is currently Assistant Professor of Computer Architecture in the University of Sevilla, where he also works towards a PhD degree. He is the author of various papers and research reports on robotics and computer communications.



Anton Civit Balcells received his Masters in physics (electronics) and his PhD from the University of Sevilla, Spain in 1984 and 1987 respectively. After working for several months with Hewlett-Packard he joined the University of Sevilla where he is currently Profesor Titular of Computer Architecture. He is the author of various papers and research reports on computer architecture and robotics.

Fernando Díaz del Río received his degree in electronic physics from Seville University in 1990. From July 1990 to April 1991 he was with Abengoa-Control Data developing a new control system in electrical supply systems. He currently holds a Fellowship from the CICYT at the University of Seville, where he works towards his PhD. His research interests include multiple robot coordination and control.

Antón Civit Breu received his degree in physics from the University of Barcelona in 1951 and his PhD from the University of Madrid in 1956. He has published more than 40 papers, two books and has directed more than 20 PhD theses. He has worked in analogue electronics, medical electronics and train safety systems. His current research interests include robotics, control system design and electric motor control.