Volume 1 • Number 1 • 1981
Coden: DSIADY ISSN: 0196-559X

Digital Systems *for* Industrial Automation

An International Journal

Igor Aleksander, *Editor*

Crane Russak • *New York*

An Example of a Distributed Intelligence Discrete Process Controller

RICCARDO CASSINIS
Milan Polytechnic Artificial Intelligence Project

Abstract This paper describes the scope and the current status of the research on industrial robotics that is being conducted at Milan Polytechnic Artificial Intelligence Project.

The research, started about three years ago, has led to important results both for hardware and software architectures.

This paper is mainly concerned with the hardware architecture of the developed system, that is, a hardware architecture based on an array of microcomputers, driven by a minicomputer, that implements an original distributed intelligence system and that may be used to control a wide variety of processes.

Special emphasis is given to the possibility of driving more than one robot with the same control system, in order to obtain completely automatic assembly lines.

No restrictions are placed on the kind and number of robots and special-purpose machines employed.

An overview of the software structure of the machine, and of the methods that may be used to program it, is also given.

Introduction

The dramatic development that industrial robots have undergone in the last few years has led many scientists to the conclusion that robots represent one of the most interesting applications of technology, and that their impact on manufacturing processes will be very strong in the immediate future.

Digital Systems for Industrial Automation, Volume 1, Number 1
0196-559X/81/010045-00 $02.00/0

The aim of this paper is to present the results of a three-year research project conducted at the Milan Polytechnic Artificial Intelligence Project (MP-AI). The results mainly concern control systems for sophisticated machines, such as assembly robots.

The outstanding results of this research were that a distributed intelligence system, implemented on an array of loosely connected microcomputers driven by a supervisor unit, offers very good performance, and undoubtedly greater than that obtainable by traditional control systems.

Although the system was designed to drive robots, it may also be used for a wide variety of industrial control problems.

The working principles of this system will be described, together with some experimental results obtained on the MP-AI SUPER-SIGMA robot.

The implementation of the system showed that its cost-to-performance ratio is comparable to that of a traditional control system, and improves as the complexity of the process to be controlled increases.

The next section contains an overview of the problems that had to be solved both from hardware and software points-of-view. Mechanical problems will not be discussed, since they depend upon the robot used and are not strictly related to the other problems.

The following section states how intelligence must be split in order to ensure the best utilization of the system's computing power, while succeeding sections explain how the system was actually implemented.

Overview of Problem

Many different mechanical structures may be found among industrial robots.

The most common is perhaps the spherical-coordinates or cylindrical-coordinates structure but, especially for assembly robots such as the OSAI SIGMA [1], the Cartesian coordinates structure has proved to be very effective.

This is because assembly operations consist mainly of three kinds of operation: picking up pieces from dispensers, orienting them, and inserting them in their final positions. The analysis of some typical cases has led to the conclusion that the first and the third kind of operation are usually performed by moving the hand of the

robot along straight lines, and that these lines often coincide with the axes of an orthogonal Cartesian space.

It can be said that, if the robot has a Cartesian structure, many "important" operations may be carried out by moving one motor only.

On the other hand, nearly all other movements require a high degree of precision with respect to their final position, although the path followed from the initial to the final position is often irrelevant.

This means that movements of the motors in a Cartesian robot need not be strictly correlated. For instance, a movement between the points P1 (X1, Y1, Z1) and P2 (X2, Y2, Z2) may be divided into three separate movements:

Axis X: From X1 to X2
Axis Y: From Y1 to Y2
Axis Z: From Z1 to Z2

These three movements may take place one at a time or simultaneously without affecting the final result.

The control system was therefore designed without taking into account the need of coordinating movements of different motors. (Linear interpolation is, however, possible, in order to obtain straight movements in any direction.)

This obviously limits the field of application of the control system. It is, however, possible to tailor the system to other kinds of robot by implementing more hierarchical levels than the two described in this paper [2].

Although the mechanical devices employed in robots are nearly always very good, and offer great accuracy and working speed, their control is usually quite poor, thus limiting top performance of the mechanics. For instance, many robots that could be used for complex and delicate tasks are driven by simple NC and CNC systems, thus making them act as special numerical control machines, rather than as true robots.

Moreover, differences between manufacturers' standards make it often difficult to interface robots of different kinds when coordinated jobs are required.

The main goal of the research was to design a system that would allow top performance from the mechanics and that could be used

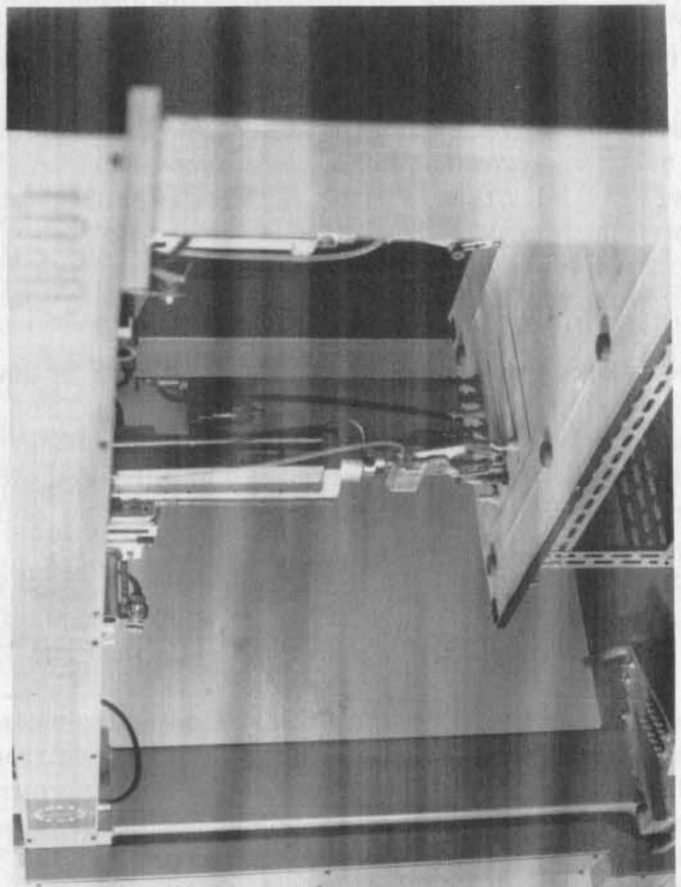
Figure 1. SUPERSIGMA robot mechanical part

without difficulties for several kinds of robots, thus allowing easy interfacing between different machines.

The research began when the feasibility model of the Olivetti SIGMA robot was donated to us in March, 1976. When the machine (see Fig. 1) first came to our laboratory, the control system, based on standard circuits employed in NC machines, severely limited the performance of the mechanics (mainly motors), and no significant improvements were possible.

The first decision was then to completely rebuild the electronics using new concepts, keeping in mind that the control system should enhance the mechanical performance rather than limit it.

Another consideration was that the new control system should be economical, simple, and reliable, with a high degree of modularity and of flexibility.

The global system, that is, the mechanical part and the control electronics, was named SUPERSIGMA.

In order to build a machine with such characteristics, it was necessary to split the global control problem into many sub-problems in such a way that the various sections of the resulting machine would not interfere with each other, and that connections between them would be as simple as possible.

First of all, the components of the robot were analyzed in order to determine their control needs, and also to see if it was possible to classify them accordingly.

This analysis led to the conclusion that the mechanical part of a robot is made up of only two classes of device: actuators and sensors.

In order to avoid any confusion, two definitions will be given.

An actuator is any device that, in response to electrical stimuli applied to its inputs, causes physical modifications of the space in which the robot operates. This definition applies not only to motors and similar devices, but also to lamps, warning bells, etc.

A sensor is any device whose electrical output depends–according to a given law–upon one or more physical parameters of the world surrounding the robot or of the robot itself.

Of course, both actuators and sensors may be divided into a great number of sub-classes, but the fact that no devices may be added to the robot that are neither actuators nor sensors is extremely important for the principles being discussed.

High-Level and Low-Level Control: The CCU Concept

Once the components of the mechanical part have been defined, it must be seen how they can be connected together to achieve the expected performance from the robot.

To do this, a comparison will be made with the human body. It should be pointed out that this comparison is only done to clarify some concepts, and that the robot control system was not designed as an imitation of the organization of man or of other animals.

First, there must be a "brain" that controls the whole system according to its "will," that is, a program that describes the tasks the robot must perform. This brain has the complete control of the machine, since no other device may undertake any action without having been authorized by the brain. For this reason, the brain will be referred to as CCU (Central Control Unit).

Since the robot must be programmable, it is most convenient that the brain be implemented on a computer, rather than on random logics.

SUPERSIGMA employs stepping motors whose characteristics will be detailed in the next section; it will only be said here that it is impossible for a single computer to control all six motors of SUPERSIGMA, plus the other devices (hands, force sensors, etc.).

On the other hand, controlling a stepping motor is not a difficult task; it only requires a great amount of quite trivial processing in order to obtain suitable accelerations and decelerations.

For this reason, some "nervous ganglions" were inserted into the system. These receive concise commands from the CCU and process them in order to obtain signals suitable for driving the motors.

The same thing happens for all the other devices of the system. For instance, force sensors applied to the wrists of the robot require an analog to digital conversion and some processing of the resulting signal. All this work is done by a specialized "ganglion"; the CCU receives only the useful information that can be extracted from sensors.

The correspondence between human body organs and robot components is shown in Fig. 2.

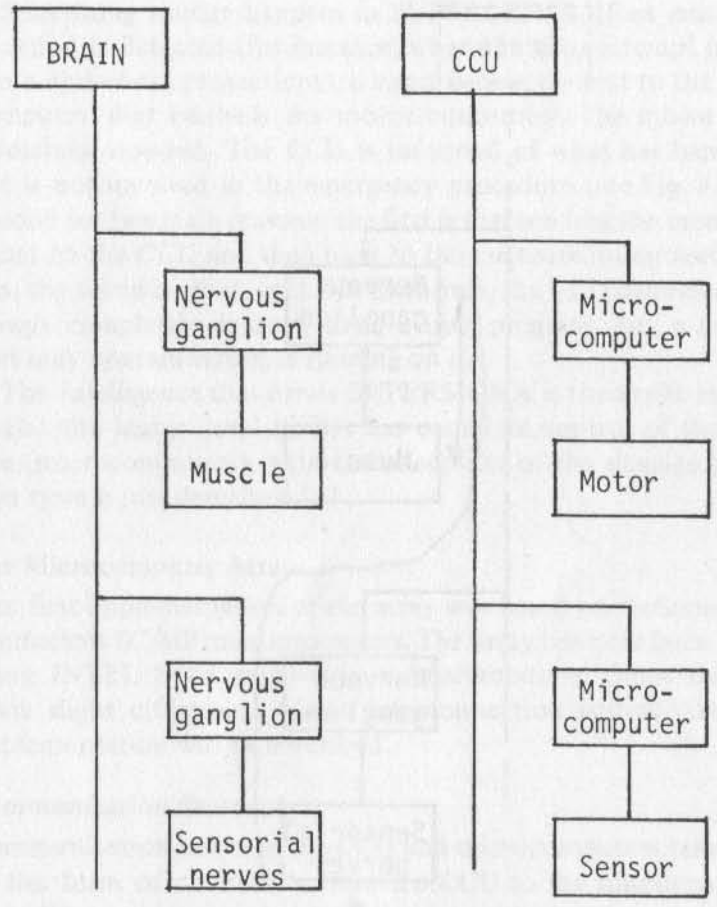

Figure 2. Correspondence between body and robot organs

The nature of the jobs that must be committed to ganglions is such that, at this level also, the use of computers is mandatory. Since each job is quite simple (even if it requires many computations), microcomputers are most suitable for this application.

The communication path between the CCU and microcomputers was called MICROBUS.

There is an important exception to the stated rule that no ganglion may undertake actions without having been authorized by the CCU.

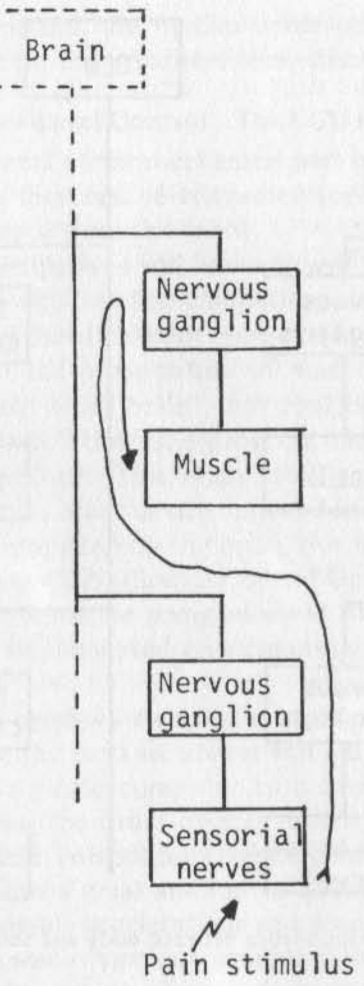

Figure 3. Reflex arc phenomenon

This exception is very similar to the reflex arc phenomenon that takes place in many animals, including man. If, for instance, something hot is touched, the hand is often retracted before pain is felt. This is because stimuli from sensorial organs in the hand cause an immediate contraction of arm muscles. The brain is not involved in this process, which takes place at the spinal cord level.

Something similar happens in SUPERSIGMA. If an emergency situation is detected (for instance, when the arms attempt to overcome end-of-rail protections), a signal is directly sent to the microcomputer that controls the motor concerned. The motor is immediately stopped. The CCU is informed of what has happened, but is not involved in the emergency procedure (see Fig. 3.). This is done for two main reasons: the first is that sending the emergency signal to the CCU and then back to the microcomputer seems useless, the second is that, as it will be shown, the CCU behavior is not always completely known, since a user program, i.e., a program that may contain errors, is running on it.

The intelligence that drives SUPERSIGMA is then split into two levels: the higher level (CCU) has complete control of the lower one (microcomputers), with the exception of the damage prevention system just described [3].

The Microcomputer Array

The first implementation of the array was based on National Semiconductors SC/MP microprocessors. The array has now been rebuilt using INTEL 8748 single-chip microcomputers. Since there are some slight differences in the interconnection system, the latter implementation will be described.

Communication Protocol

Communication between the CCU and microcomputers takes place in the form of commands from the CCU to the microcomputers and of data and status signals in the opposite direction.

The communication protocol was chosen in such a way that it would minimize hardware requirements. Speed is not a problem, since the mechanical nature of the hardware used to drive implies that the actuation of each command is always slow compared to the transmission time.

The need to minimize hardware suggests the use of a bus structure reaching all microcomputers (see Fig. 4.).

Two assumptions were necessary in order to have a simple and straightforward communication protocol.

The first is that only "free" microcomputers may receive commands. This means that microcomputers that are executing a

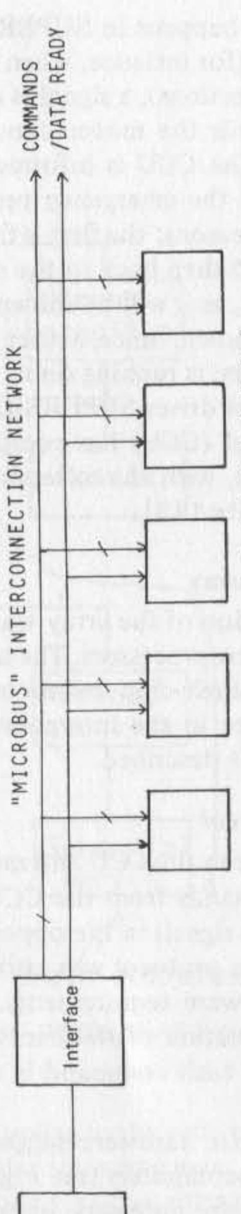
CCU
Interface
"MICROBUS" INTERCONNECTION NETWORK
M I C R O C O M P U T E R S
COMMANDS
/DATA READY

Figure 4. MICROBUS block diagram

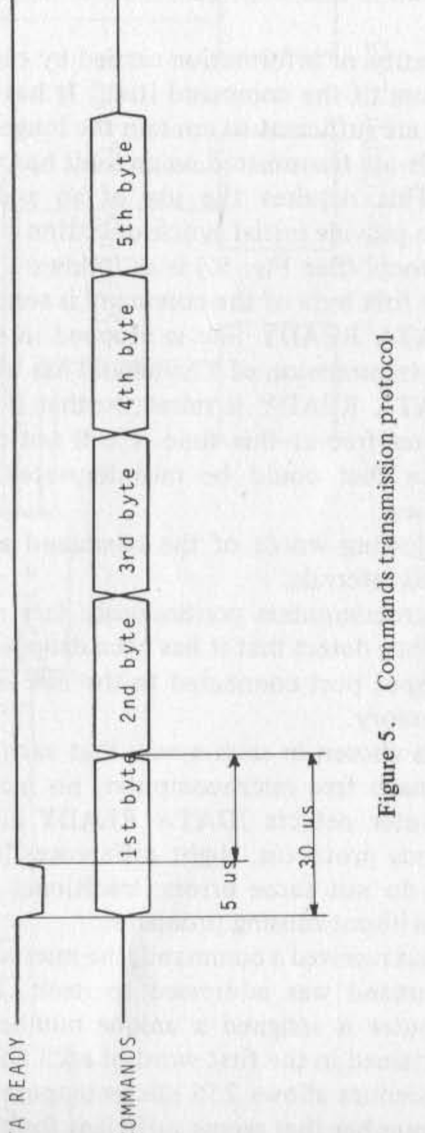

Figure 5. Commands transmission protocol

previously received command will be virtually disconnected from the array until the execution of the command has come to an end.

The second is that each command is directed to a single microcomputer.

The quantity of information carried by each command depends on the nature of the command itself. It has been found that five 8-bit words are sufficient to contain the longest possible command. These words are transmitted on an 8-bit bus with an asynchronous protocol. This requires the use of an additional line (/DATA READY) to provide initial synchronization.

The protocol (See Fig. 5.) is as follows:

1) The first byte of the command is sent on the bus.
2) /DATA READY line is dropped in order to indicate that the transmission of a command has been initiated.
3) /DATA READY is raised, so that if a microcomputer becomes free at this time it will not receive the command, since that could be misinterpreted, if the first byte is missed.
4) Following words of the command are sent on the bus at fixed intervals.

Free microcomputers continuously test /DATA READY line. As soon as they detect that it has been dropped, they get five words from the input port connected to the bus and store them in their internal memory.

Timing is chosen in such a way that each command may be received by each free microcomputer, no matter at what time the microcomputer detects /DATA READY signal. As happens for asynchronous protocols, slight differences in the speed of microcomputers do not cause errors; traditional crystal oscillators are employed without causing problems.

Once it has received a command, the microcomputer must decide if the command was addressed to itself. For this reason, each microcomputer is assigned a unique number, and this number is always contained in the first word of each command.

This procedure allows 256 microcomputers to be connected to the bus: a number that seems sufficient for any application.

In order to let the CCU know if microcomputers are free or busy, a certain number of "status lines" (see Fig. 6.) were added

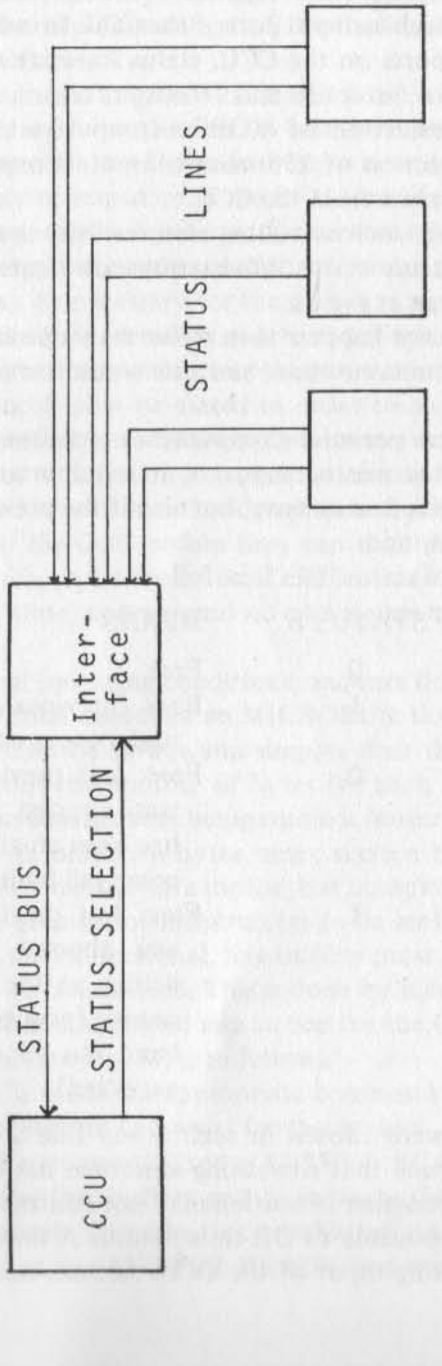

Figure 6. Status lines

to the system. These lines (typically two per microcomputer) should directly reach an input port of the CCU. In order to minimize wiring and I/O ports on the CCU, status lines are multiplexed in such a way that 16 input bits and 16 output bits on the CCU allow the complete connection of 80 microcomputers to MICROBUS. The full connection of 256 microcomputers requires only two additional output bits from the CCU.

Other solutions, such as coding statuses into messages, did not seem feasible because status informations are treated by the CCU as flags, rather than as events.

Moreover, it may happen that more than one microcomputer change state at the same time, and this would make coding more difficult.

Two status lines per microcomputer were chosen instead of one because, for motor microcomputers, it is important to state not only if the device is free or busy, but also if the previous command was correctly executed.

The meaning of status lines is as follows:

STATUS A	STATUS B	MEANS
0	0	Free
0	1	Busy (previous command still under execution)
1	0	Free, but previous command was aborted because there has been an attempt to overcome rail limits
1	1	Free, but previous command was aborted because force acting on the hand was excessive (possibly because the hand has hit something unexpected)

Status codes were chosen in such a way that a logical "1" on status line A means that something abnormal has happened, and that the normal program execution may not continue.

This made it possible to OR-tie all status A lines from motors into an interrupting input of the CCU. Actions undertaken upon

reception of such interrupt depend on the program stored in the CCU. This is the implementation of the "reflex arc" described in the previous section.

We have seen how commands are transmitted to microcomputers from the CCU, and how the CCU is informed about the status of each microcomputer. It will be shown now how data may be transmitted from microcomputers to the CCU (see Fig. 7.).

This latter is specially significant for sensors microcomputers, to be discussed later, but applies also to motor microcomputers. For instance, it may be necessary for the CCU to know the position of one motor at the time a crash was detected (Status 1-1).

Since data are transmitted over the same bus that carries commands, a protocol must be stated in order to avoid conflicts (for the sake of simplicity, no hardware bus arbiter was implemented).

This protocol is based on a "speak when you are requested to" principle. In other words, microcomputers must receive a special command from the CCU before they can send data over the bus. It is then the responsibility of the CCU to interrogate one microcomputer at a time, and to send no commands until the answer is complete.

Under normal operating conditions, answers flow is only a small fraction of the total data flow on MICROBUS; the communication protocol may then be slower and simpler than that used for commands. Moreover, the number of bytes for each answer is not yet known, since new sensors are being studied. Motor microcomputers may answer with only two bytes, since sixteen bits are sufficient to represent the position of a motor, but complex sensors may require a greater amount of information to be sent for each answer.

Since the bus is bidirectional, line buffers must be reversed when an answer is to be transmitted. This is done by means of /ANSWER READY line, that is also used as a strobe for the CCU (see Fig. 7.).

The protocol (see Fig. 8.) is as follows:

1) The CCU sends the appropriate command to the concerned microcomputer and waits for the answer.
2) The microcomputer drops /ANSWER READY line in order to reverse line buffers and to indicate that a byte is being transmitted. To make this possible, all microcomputers are connected to /ANSWER READY line through a port that

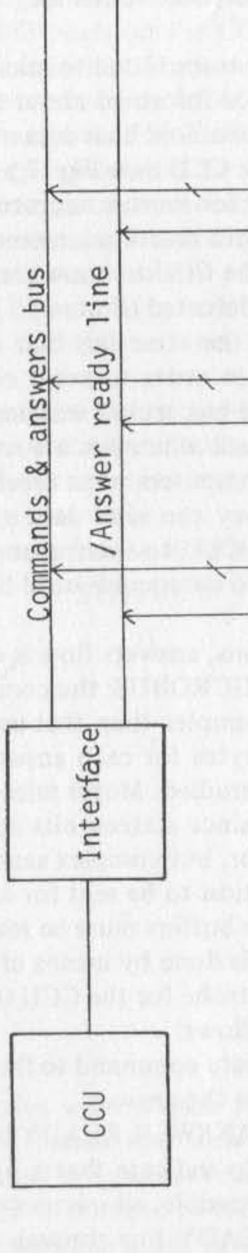

Figure 7. Answers data path

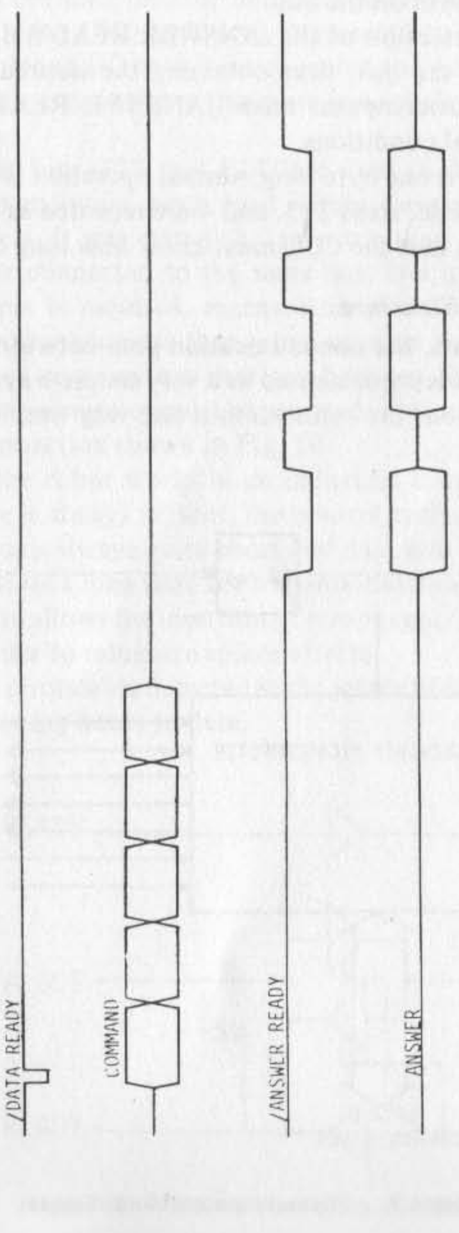

Figure 8. Answers transmission protocol

is normally in the high impedance state.

3) At the same time, the microcomputer sends the appropriate answer byte on the bus.

4) Upon detection of the /ANSWER READY signal, the CCU samples the bus, thus obtaining the desired information.

5) The microcomputer raises /ANSWER READY to restore the initial conditions.

If the answer is one byte long, normal operation is resumed after this step; otherwise, steps 2, 3, and 4 are repeated as required. This of course means that the CCU must know how long each answer is.

Microcomputer Structure

As already shown, the communication path between the CCU and microcomputers is implemented in a very simple way. From a hardware point-of-view, the connection is also very simple (see Fig. 9.).

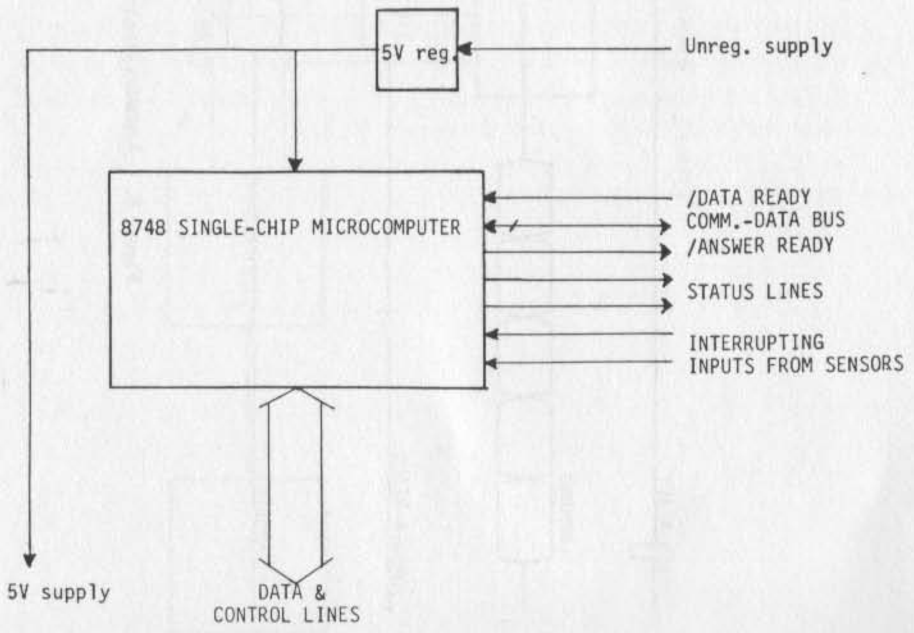

Figure 9. Microcomputers block diagram

One of the three on-chip bidirectional ports of 8748 is used for the MICROBUS connection. The port is bidirectional, and no interface is needed between the chip and the transmission line.

A testable input (TO) is directly connected to /DATA READY line, and one bit of I/O port 3 is connected to /ANSWER READY line. As already pointed out, this port is normally in the high impedance state.

Two output bits (P23 and P24) are used to drive status lines. In order to maintain ports load within fan-out capabilities of microcomputers, it was stated that no more than 10 microcomputers should be connected to the same bus. If a higher number of microcomputers is required, regenerators must be placed on the bus. It is important to note that, when answers are being transmitted, only those regenerators that are between the CCU and the answering microcomputer must be reversed. This is simply obtained by the OR connection shown in Fig. 10.

Although the robot works in an industrial environment, where electrical noise is always present, the control system is not affected because the bus is always quite short and data rate is comparatively small. This allows a long time for overshoot settling before a line is tested, and also allows the insertion of strong capacitive loads where required in order to minimize spikes effects.

In fact, no errors were detected in the whole life of the machine—about 600 working hours to date.

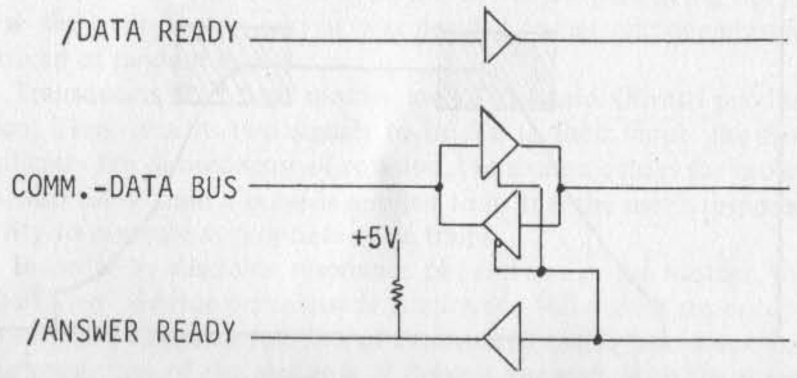

Figure 10. Regenerators block diagram

Microcomputer Interfaces

Interface circuitry depends upon the task each microcomputer must perform. The two most significant interfaces that have been built will now be described.

The first is motor's interface.

SUPERSIGMA employs stepping motors for moving the arms: although these motors do not require feedback, and stability problems do not arise, these motors are quite difficult to drive because they need to be carefully accelerated to their working speed in order to avoid loss of position synchronism between the rotor and the statoric magnetic field. The same care must be taken for decelerating and stopping them.

In order to obtain the best performance from each motor, the speed-versus-time curve should have a very smooth connection between acceleration and constant speed parts, since this is a very critical point, where loss of synchronism may occur (shaded area in Fig. 11).

Moreover, it is desirable to have the option of using various acceleration and deceleration curves, in order to accommodate, for instance, different loads on the hands of the robot.

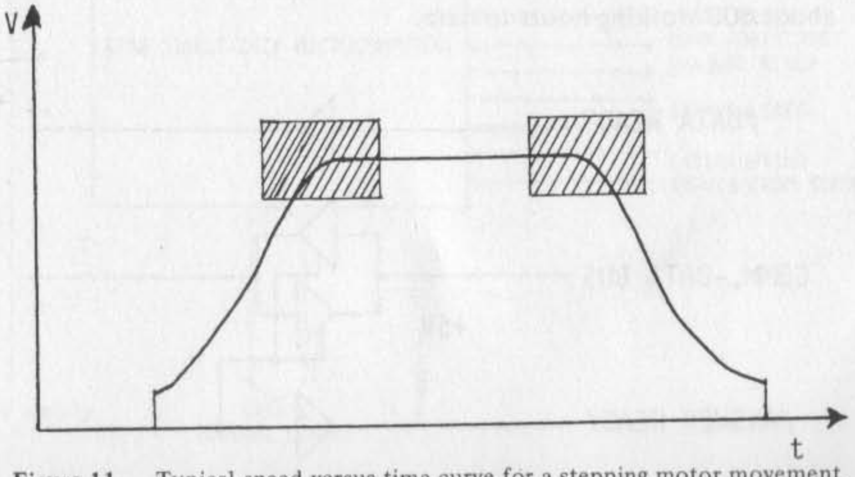

Figure 11. Typical speed-versus-time curve for a stepping motor movement. Shaded areas indicate critical points.

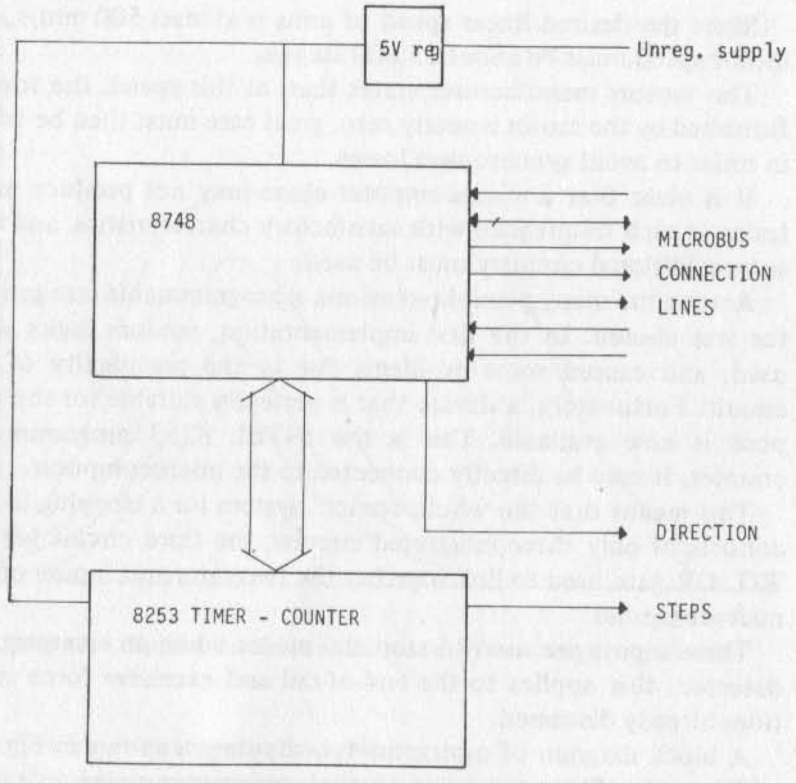

Figure 12. Motor microcomputers block diagram

The need to obtain such complex and often parametric curves was the main reason why it was decided to use microcomputers instead of random logics.

Transducers that feed motors are of standard Olivetti production. They require two signals to be fed to their input: the first indicates the desired sense of rotation, the second causes the motor to step every time a pulse is applied to it. It is the user's responsibility to generate appropriate pulse trains.

In order to eliminate resonance phenomena in the motors, the "half step" driving technique is employed. 400 pulses are needed to obtain a complete rotation of the motor's shaft; this means that the resolution of the motor is .9 degrees per step. With the actual gear system, this leads to a linear resolution of .0375 mm per step.

Since the desired linear speed of arms is at least 500 mm/s, top motor speed must be about 15,000 steps/s.

The motors manufacturer states that, at this speed, the torque furnished by the motor is nearly zero; great care must then be taken in order to avoid synchronism losses.

It is clear that a microcomputer alone may not produce pulse trains at such frequencies with satisfactory characteristics, and that some additional circuitry must be used.

Among the many possible solutions, a programmable rate generator was chosen. In the first implementation, random logics were used, and caused some problems due to the complexity of the circuit. Fortunately, a device that is perfectly suitable for the purpose is now available. This is the INTEL 8253 programmable counter. It may be directly connected to the microcomputer.

This means that the whole control system for a stepping motor consists of only three integrated circuits, the third circuit being a TTL OR gate used to link together the two interrupt inputs of the microcomputer.

These inputs are used to stop the motor when an emergency is detected; this applies to the end-of-rail and excessive force situations already discussed.

A block diagram of motor microcomputers is shown in Fig. 12.

From the firmware point-of-view, program running in motor microcomputers is rather intricate, due to the need to generate transcendental functions and their reciprocals.

The normal command sent to motor microcomputers contains only two units of information: the number of steps to be performed and the direction of the movement.

The microcomputer must then decide, on the basis of the length of the movement, the maximum speed to be reached and the shape of the acceleration and deceleration curves. If the movement is very short, it is performed at a constant low speed; if longer, the motor is accelerated until a computed speed has been reached, and then decelerated until it stops. If the computed speed exceeds the maximum allowable, a constant speed movement is inserted between the acceleration and deceleration segments (see Fig. 13.).

If an interrupt is sent to the microcomputer while the motor is running, the movement must be stopped as soon as possible, that

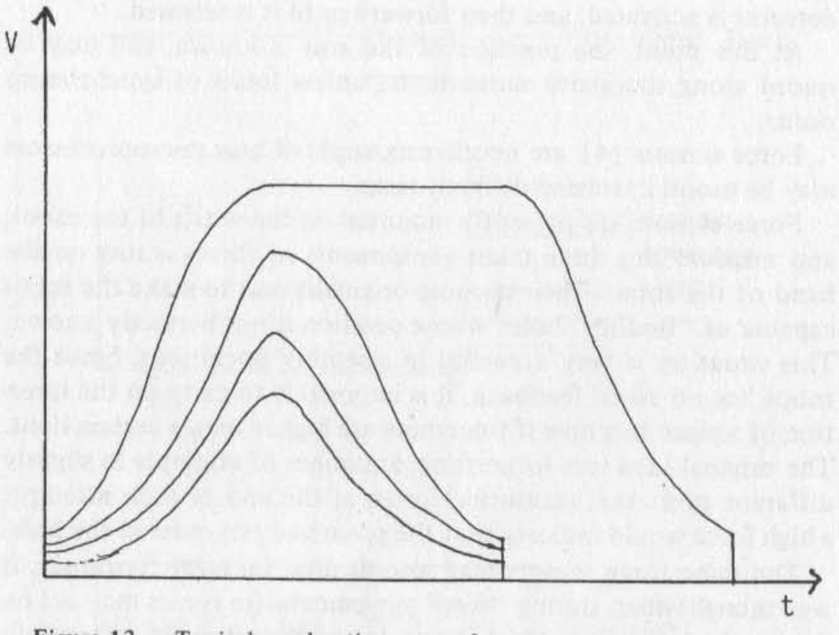

Figure 13. Typical acceleration curves for various length movements

is, following the steepest allowable curve.

It should be remembered that a stepping motor may not be stopped simply by suspending the pulse train, since this would most probably cause a loss of synchronism that would increase the stopping time, rather than reduce it.

Some other functions may be performed by motor microcomputers. For instance, a movement may be performed at constant speed, regardless of its length, or speed limits may be imposed. These special functions are indicated by appropriate op-codes in the command.

Another important function is the origination. Since no positional feedback is provided in SUPERSIGMA, it is impossible to know the position of the arms when the machine is first turned on. The only way to know this is to bring each axis into a known position. This operation is automatically performed by motor microcomputers upon reception of a special command that causes the motor to be moved in the backward direction until the end-of-rail detector is activated, and then forward until it is released.

At this point, the position of the arm is known, and may be traced along successive movements, unless losses of synchronism occur.

Force sensors [4] are another example of how microprocessors may be useful in solving difficult tasks.

Force sensors are presently mounted on the wrists of the robot, and measure the three main components of forces acting on the hand of the robot. Their purpose originally was to make the robot capable of "finding" holes whose position is not perfectly known. This situation is very common in assembly operations. Since the robot has no visual feedback, it is impossible to carry on the insertion of a piece in a hole if tolerances are higher than a certain limit. The original idea was to perform a number of attempts in slightly different positions, measuring forces at the end of each attempt: a high force would indicate that the piece had not entered the hole.

The same force sensors may also be used for safety purposes. It was thought that, during "free" movements, no forces may act on the hand, except for inertial forces. It was then decided to continuously test force sensors in order to determine if unexpected forces were present. The presence of such forces would mean that the hand had struck something, and the first thing to do would be to stop the motor.

Inertial forces that affect force measurements during accelerations may be eliminated by combining each force sensor with an accelerometer and mixing the two analog signals in an appropriate way.

Since force sensors are analog devices, an analog to digital converter is needed. Signals from force sensors are first multiplexed in order to use only one converter.

The associated microcomputer controls the multiplexer and the converter (see Fig. 14.). Since force sensors must be continuously tested, the microcomputer is always "busy."/DATA READY line must then be connected in this case to an interrupting input of the microcomputer so that is may receive commands.

These commands may be of two kinds. The first is "tell me the value of force acting on axis . . . ," and is quite simple to understand. The second command is used when operations that require force

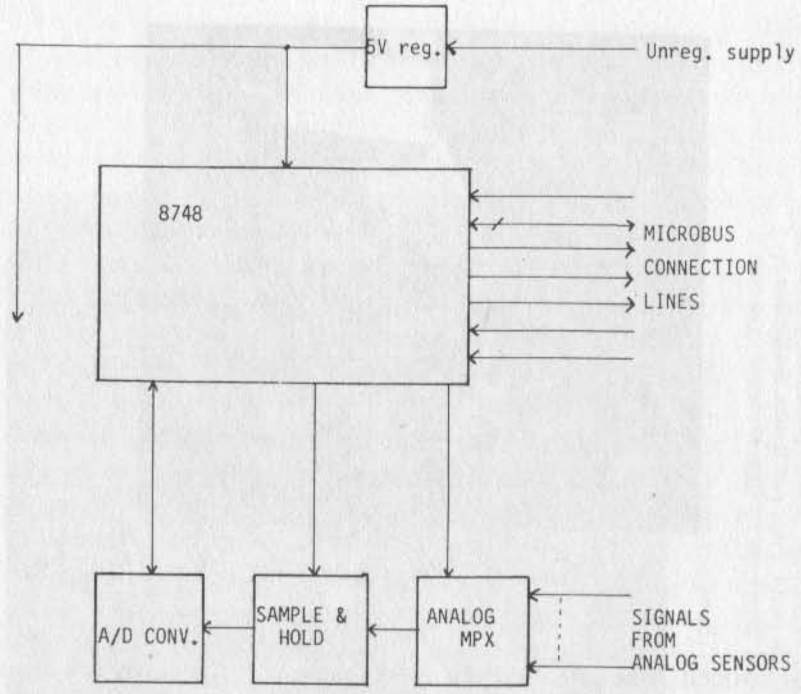

Figure 14. Sensors microcomputers block diagram

to be issued from the robot's hands are to be performed. Under normal conditions, the force threshold that causes an interrupt to be sent to the corresponding motor microcomputer is quite low, so that crashes may be detected as soon as they occur. If force has to be issued, these emergency thresholds must be changed, otherwise the motor would stop as soon as the operation is initiated. The command is then used to change thresholds for concerned axes, and to restore them to standard values when the operation ends.

Object Detector

An interesting device that has been added to the robot is a very simple object detector that may be used for a variety of purposes.

A lamp-photodiode couple was mounted in the fingers of the robot (see Fig. 15.). Its original purpose was to test if an object was present in the hand at some points of assembly operation. It

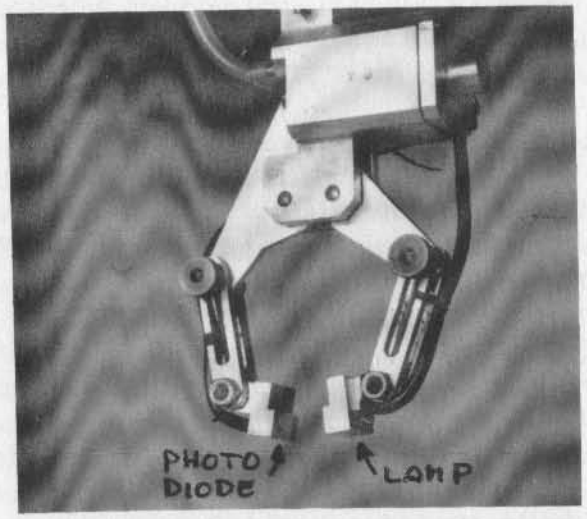

Figure 15. Object detector

was noticed, however, that the same sensor could be used for more sophisticated operations, such as the lookup of pieces, or even to measure the dimensions of such pieces. This is possible because the coordinates of the hand are always known. Thus, by combining a few program instructions the task may be easily solved.

As an experiment, a program was written that looked for chess pieces randomly placed along some lines on the working board of the robot, measured their height, and placed them accordingly on other lines (See Fig. 16.).

The most obvious application of this sensor is to eliminate some parts feeders, or to substitute them by simpler ones, since for many objects an exact initial position is not required.

Programming the CCU

The problem of programming robots has often been neglected by manufacturers.

Besides those methods that are quickly becoming obsolete (such as using pinboards or similar devices), three basic methods are presently used: teaching-by-doing, console programming, and

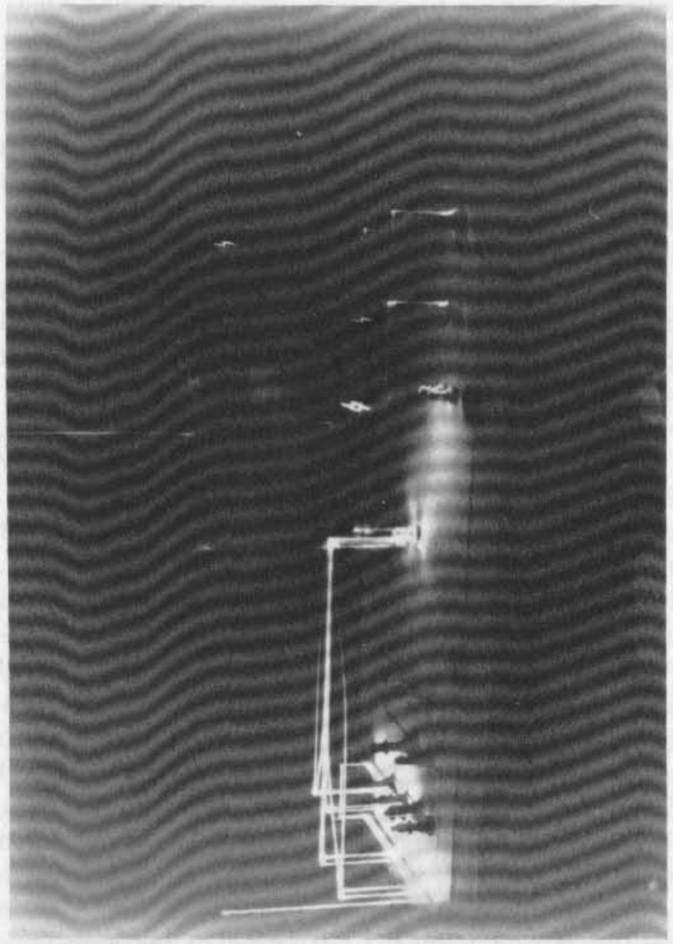

Figure 16. Experimental setup for object detectors testing.

Note: This is a long-exposure picture taken during the program execution. The only light used is that furnished by object detectors lamps. Light beams show movements of the arms. Pieces to be searched are on the left of the table; they are brought to the intermediate position by the left arm, and to the final position by the right arm.

language programming.

The first method is extremely useful when complex movements are to be described, as in painting or welding robots. It is obvious that with this method only straightforward programs may be generated. The impossibility of implementing conditional branches severely limits the possibilities of this method.

Console programming is based on a specialized console that allows programs to be inserted in the robot by pressing appropriate pushbuttons or similar devices. This method allows full programmability of the robot, since conditional and unconditional branches may be described, but the language level is rarely higher than that of a small programmable calculator.

The third method implies normal computer programming techniques. This method was chosen by us for many reasons, among which the most important was that the robot must not be considered as a computer peripheral, but that the whole machine be regarded as a special purpose computer.

This allows a great simplification of all programming problems, because such problems may be solved with the techniques commonly used in normal programming [5], [6].

Since no computer exists with the appropriate instruction set for driving the robot, the first step was to design such a computer. In order to avoid the use of microprogrammable computers, it was decided to use a virtual machine implemented on a normal minicomputer. The definition of the instruction set of this virtual machine was quite a difficult task, since its level has to be low enough to allow complete control of the machine, yet high enough to be efficient and to simplify the design of compilers for high-level languages. Furthermore, it should provide multiprocessing facilities, since the only way to carry on parallel processes is to describe them as separate tasks and to use a set of synchronizing primitives to provide correct execution sequences. In the first implementation of the system we started with the definition of the high level language, then designed a virtual machine that could interpret this language in the simplest possible way.

At this time, the control system was under construction, and its features were not yet completely known. For this reason, a quite general language was chosen. It was named MAL (Multipurpose

Assembly Language), since it was not oriented to any particular application, and its level was similar to BASIC language [7].

Syntax is about the same as for BASIC, with some minor modifications (for instance, variable names may be as long as desired, in order to increase the program's readability); some BASIC features were eliminated (such as array management) because they were useless in robot programming, and some added. Additions included all the instructions that drive the robot (MOVE, ACTUATE, etc.), and provisions for parallel task execution.

The resulting language is quite efficient for assembly operations, since it allows a simple description of the tasks to be performed.

As stated above, MICROBUS capacity is much higher than that required to drive SUPERSIGMA. Moreover, SUPERSIGMA alone is unable to accomplish any task, because it needs devices that bring pieces to it, and that withdraw assembled parts. We then extended the idea of robot to the whole system that is used to perform the process, rather than to the single machine. This is possible, since MICROBUS may drive a great variety of devices, not necessarily robots.

The whole process may then be regarded as a number of tasks, each accomplishing a definite job. The only problem is to synchronize these tasks in order to have a correct execution order. The same technique used in real time systems may be applied here with no modifications. In MAL, we used flags to synchronize processes, but any other method would suit equally well.

The resulting program is a number of "boxes," each describing a different task, tied to each other by means of synchronizing primitives.

Originally, the CCU was implemented on a LABEN 70 minicomputer. In the new implementation, it will be replaced by a Digital LSI 11, in order to have a greater efficiency and a higher computing speed.

Since MAL is written in FORTRAN, it may be easily transferred on the LSI 11. Since our research program includes new, higher-level languages for robot programming, we also decided to implement on the LSI 11 a new virtual machine written in real-time BASIC. This can be done in two ways: the first is to add to real-time BASIC a few functions that provide MICROBUS driving, and to write robot programs in BASIC; the second is to define the virtual machine and to implement it by using BASIC. The second solution is more complex, but allows the system to be exactly tailored to our needs.

Very high-level languages, such as POINTY [8], will be implemented on this new machine.

Experimental Results

Since the first implementation was terminated, SUPERSIGMA has worked for about 500 hours, while many experiments were being carried out.

The behavior of the machine has always been good, in spite of its "home made" appearance. The greatest problems encountered were related to poor power supplies or similar trivial causes.

Some experiments were carried out to measure MICROBUS saturation. As may be seen from Table 1, the execution time of some test programs was compared with the total number of commands issued by the CCU.

Table 1

MICROBUS Saturation for Various Robot Programs

Maximum Commands Rate: 10,000/sec.

PROG.	*Arms*	*Commands*	*Seconds*	*Com/Sec*	*MICROBUS Saturation*
ORD	1	751	167	4.5	.045%
SCP1B	1	328	56	5.86	.059%
SCP2B	2	394	36	10.94	.109%
PRCO	2	1600	260	6.15	.062%

It was seen that in the worst case (program SCP2B, that continuously moves the two arms of the robot over very short distances, so that execution time of every command is very small), MICROBUS was running at only .1% of its maximum capacity.

CCU saturation mostly depends on the number of arithmetic calculations required by each task. No precise measurements were taken, but it was estimated that at least 50% of the total time is

spent idling, because no commands may be issued, since microcomputers are still busy executing previously received commands.

Reliability of the system is good, as the number of components employed is very small. This is especially true of the new implementation, because the introduction of single-chip microcomputers and of programmable rate generators has reduced the total component count by a factor of about 15. The weak part of the system is, from this point-of-view, stepping motor power supplies. Here, high integration is impossible, due to the high powers employed.

Due to the modularity of the system, fault diagnosis is extremely easy. If performance is abnormal, the fault may be located using very simple techniques. Standard test programs may be used for the CCU. If the fault is not in the CCU, it obviously is at a lower level. Should it be in MICROBUS, it is easy to decide whether it is in interfaces or in microcomputers. Any failure of components in interfaces will cause malfunctions of groups of microcomputers, while a faulty microcomputer will only affect the performance of related devices. Faults located at transducer level may be found using suitable techniques that depend on the nature of the transducers.

It is also possible to include self-checking procedures for the CCU and microcomputers. These procedures may be run through, for instance, when the system is first turned on.

Conclusions and Future Developments

The system described is an experimental device that has proved to be suitable for use in industrial robots.

The structure of electronics and of the components employed are such that the engineering work for industrial production of the system would be quite reduced.

Moreover, the system may be employed in a great number of control systems where centralized control and distributed intelligence are necessary.

Besides the main research program, that concerns the hardware and software architecture of the robot, two other lines of research are being carried on: one is of a mechanical nature and will be concerned with the improvement of the existing mechanics and the addition of some degrees of freedom to the hands. (The final number will be 6 degrees of freedom per hand.)

The second is the development of a vision system to be connected to SUPERSIGMA or to other robots. At first this system will be used to recognize the kind and the position of objects being carried to SUPERSIGMA by a conveyor belt, thus eliminating the need of positioners for some classes of objects.

The vision system will later be used as an aid to assembly operations.

In parallel with this work, we are also carrying out a program of research on artificial intelligence. This program is particularly intended as a means of automatically solving emergency problems in robots, and of implementing a natural language comprehension system that would yield a much simpler method for programming the robot.

Acknowledgments

The research on SUPERSIGMA required the cooperation of many people over a period of more than three years.

Credit is due to Prof. Marco Somalvico, and to all those graduates and students whose work and ideas made the realization of the system possible: Paolo Dalla Vecchia, Giovanni Domenella, Alberto Evi, Giuseppina Gini, Maria Gini, Renato Gini, Dario Giuse, Enrico Pagello, Lorenzo Schnickel, and Massimo Tomaini.

This work was partially supported by the National Research Council.

References

1. A. d'Auria, M. Salmon: SIGMA: An integrated general purpose system for automatic manipulation. Proc. 5th I.S.I.R. Chicago, 1975.
2. R. Finkel: Constructing and debugging manipulator programs. Stanford Artificial Intelligence Laboratory. Memo AIM-284. Stanford, 1976.
3. R. Cassinis, L. Mezzalira: A Multi-microprocessor system for the control of an industrial robot. Proc. 7th I.S.I.R. Tokyo, 1977.
4. R. Cassinis: Sensing system in SUPERSIGMA robot. Proc. 9th I.S.I.R. Washington, 1979.
5. R. Bisiani, R. Cassinis: The development of a multi-micro processor system to be used in the control of an industrial robot. Proc. MI-MI '76. Zurich, 1976.

6. M. Salmon: SIGLA – the Olivetti SIGMA robot programming language, Proc. 8th I.S.I.R. Stuttgart, 1978.
7. C. Gini, M. Gini, R. Gini, D. Giuse: Introducing software systems in industrial robots. Proc. 9th I.S.I.R. Washington, 1979.
8. T. O. Binford, et al.: Exploratory study of computer integrated assembly systems. Stanford Artificial Intelligence Laboratory Memo AIM-285.4, Stanford, 1977.