An Experimental System for Computer Controlled Mechanical Assembly

PETER M. WILL AND DAVID D. GROSSMAN

Abstract—**This paper describes the software and hardware architecture of a system designed as a research tool for experiments on programming the computer controlled assembly of mechanical objects. The software consists of a real-time control level and a background level in which an on-line interpreter permits interactive programming. The hardware consists of a manipulator with sensory feedback coupled to an IBM System/7. Additional facilities are available through a link to an IBM System/370 Model 145. The application of the system to sample assemblies is also discussed.**

Index Terms—**Experimental system, manipulator, mechanical assembly, robot.**

I. INTRODUCTION

Preface

A RESEARCH project was recently initiated at IBM in the field of computer controlled mechanical assembly. As a first phase of this project, an experimental system was designed and built to act as a research tool for experiments on the assembly of mechanical objects. The hardware and software of this system and its preliminary application to sample assemblies are discussed in this paper.

Background

Broadly, the combination of computer assisted design and manufacturing (CAD/CAM) has become of significant interest in recent years. The output of such a process does not usually extend to the physical control of the primary production capacity of the plant, and the manufacturing cycle is therefore not complete. The outstanding exception is the numerical control of machine tools. Detailed analysis and subsequent implementation of the assembly function would give integrated computer control over a larger portion of the manufacturing process than heretofore has been possible [1], [2].

The current situation in computer controlled mechanical assembly is that a scant beginning has been made on the substantial and complex research problems involved. It is still too early to predict with any reliability whether or not a computer controlled general purpose assembly machine will ever be able to perform satisfactorily. The subject area is sufficiently interesting to be worth examining in detail.

Assembly Functions

Little exists in the technical literature on the broad theory of assembly [3]. The literature deals only with a host of specific details: methods of fastening, machines for wire wrapping and component insertion, devices for orienting, designs of transfer mechanisms, etc. [4]. The lack of theory is understandable because no language exists either for describing parts operationally or for describing motions, especially those required in causing parts to be assembled into a structure.

The problem in this regard is exemplified by the comment of Michie [5] that while the difficulty of Grandmaster chess is easily appreciated by amateurs, we fail to appreciate the subtle complexity of moving, since all of us are Grandmasters at moving, having spent our entire lives in continual practice. A second example is the difficulty of recording choreography and the existence of Labanotation [6] for this purpose. There is evidence [7] that these linguistic and conceptual difficulties may be fundamental in that motion is a right-hand-half brain function and may be basically nonlinguistic, while language, logic, and other formal notions are left-hand-half brain functions which can be discussed, defined, and understood.

The basic procedures, functions, mechanical tolerances, and sensory inputs required for assembly can be determined by considering the operations of any assembly factory or by studying the assembly manuals accompanying many products delivered to the consumer in disassembled form. The assembler requires a workbench, a supply of parts, some hand and power tools, and an assembly procedure or a set of drawings from which a procedure can be deduced. Component parts may be ordered in pallets or magazines, but more customarily they are disordered although sorted in bins. Another important but mostly unappreciated function is that of inspection.

In factory environments the assembly procedure is typically given in verbal rather than in written form and is highly contextual. "Pick up this piece like this and put it there." Unfortunately, this form of description is

Manuscript received September 11, 1974; revised January 30, 1975.

The authors are with the Computer Sciences Department, IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y. 10598.

too loose to be of use currently in programming the complex moves of an assembly operation. Long-term research may eventually determine how closely such a form of description may be approximated.

The only computer languages for motion deal with numerically controlled machine tools, and a typical example is APT [8]. There is a significant difference between the pre-APT environment and the current pre-"Assembly Language" environment. Prior to APT, the machining of parts was already highly systematized in terms of cutting speeds, feed rates, and tolerances. No such application knowledge appears to exist for the assembly function. There is no commonly accepted computer language for the more free-form motions necessary to perform the assembly function and such a language is one possible long-term output of the research in this paper.

State of the Art

The subject of computer controlled manipulators can be viewed as belonging to several fields, each with its own literature, background and history of well defined problems, e.g., prosthetics, teleoperator systems, hand-eye intelligent systems, or industrial robotics. The differences lie in the degree to which computers are integrated into the systems.

The present research emphasis in the field is towards the investigation of systems which attempt to solve a generic and complex job. The example most used is the assembly of mechanical parts. The solution desired is not simply a piece of mechanical hardware plugged into a general purpose computer with the applications research left to the eventual customer. Instead it is a total systems solution in which all issues are fully understood.

The systems approach to the assembly problem has been taken by the three leading groups in the area, the Japanese, the Research Applied to National Needs (RANN) group of the National Science Foundation (NSF), and the leading European groups.

The extent and scope of the Japanese work in computer controlled manipulator systems is clearly shown in the proceedings of the three Symposia on Industrial Robots in 1970, 1972, and 1973 [9]. Over 90 companies in Japan have products in the area, contrasting with 15 in the United States and 32 in Europe [10]. The outstanding technical achievements are exemplified by the work of Ejiri *et al.* [11] in which a program analyzed a line drawing and a disordered scene containing building blocks, developed a strategy for assembling the blocks to form the object in the drawing, and then built it successfully. This work solves in principle all the assembly problems which have become standard in the West in the last 10 years of artificial intelligence (AI) research. What remains is the expansion of the work to handle a real-world environment: nonblock parts, real objects and assemblies, effects of imperfect manipulators, and convenient methods of programming.

The work at the Electrotechnical Laboratory in Japan covers the fields of manipulator design, analysis, control, object recognition, strategic planning, etc. [12]. Mitsubishi and other companies are also known to have significant research activities.

Work in the United States [13]-[15] was formerly funded by the Advanced Research Projects Agency (ARPA) at the major AI research centers under the subject of hand-eye coordination. The present emphasis is a great deal more practical and is funded by the RANN office of the NSF. Three major efforts are underway: The Charles Stark Draper Lab [16] is working on programmable assembly machines with emphasis on good low-level control and tactile sensing. Stanford Research Institute [17] is working on packaging applications and is using vision. Stanford University [18] is working on languages and the theory of assembly. They have successfully demonstrated the assembly of a modified automobile water pump by computer controlled manipulator.

The above citations are to work funded by national agencies. There are signs that private industry is also sponsoring work in the general area of computer controlled assembly. Two examples are General Motors Research [19] where the emphasis is on vision as an input and General Telephone and Electronics Research [20] where a command language for automation is under development.

Work in Europe is led by the University of Edinburgh Department of Machine Intelligence [21], a major AI center in which significant assembly work has been done. The University of Nottingham [22] has concentrated on mechanical expertise and is working in the direction of increasing the pattern recognition content of their work.

In parallel with this effort going on in the West, there have been developments in the USSR. The state of the art at least in the Leningrad Institute [23] shows both breadth and depth. The work appears to be motivated by problems in the design and control of undersea manipulators and articulated vehicles for traversing rough terrain. The research includes hardware development, command language design, and question answering systems for path planning. Ignatiev also gives a comprehensive set of references to other Soviet work.

Against this background, the Research Division of IBM is a relative newcomer in the field of computer controlled manipulators. The system described in the following sections is designed for the class of assemblies which have a size less than a 30 cm cube, a total weight less than 3 kg, and contain up to 50 parts weighing between 1 g and 500 g. Such assemblies require force sensing and are not possible with simple pick and put devices.

The initial hardware and software described in this paper comprise an experimental system which will be the common denominator in future discussions of mechanical and electronic hardware, systems software, and applications results. The system is currently being used to study some highly complex real-world assemblies.

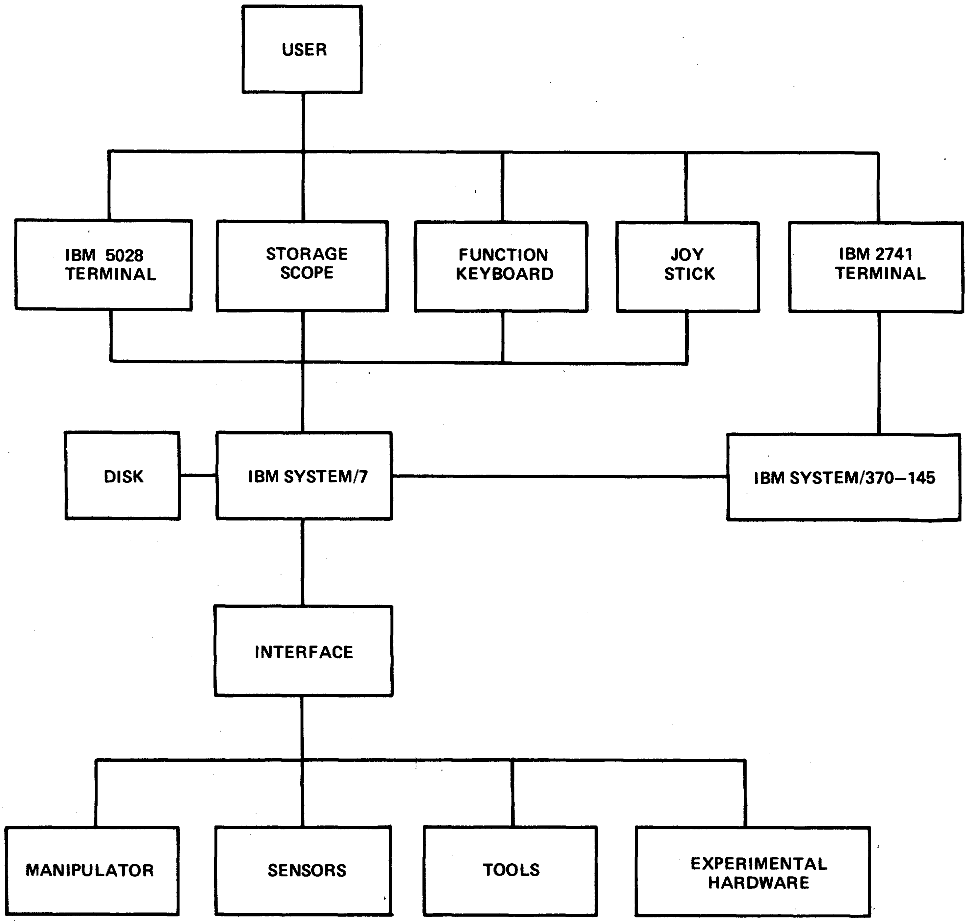


Fig. 1. Block diagram of the hardware system.

II. HARDWARE SYSTEM

Overview

A block diagram giving a general overview of the hardware system is shown in Fig. 1. The central role in this system is played by an IBM System/7 computer. The user station consists of a keyboard terminal, storage scope, function keyboard, and joystick. A second terminal connects to an IBM System/370 Model 145 which provides host support. The System/7 also interfaces to the motors and sensors of the manipulator and to power tools and other devices at the work station.

Motors

A manipulator needs three nonredundant motors to have the capability of reaching an arbitrary position in three-dimensional space. In order to achieve an arbitrary gripper orientation it needs three additional motors. Finally, the gripper itself must have a motor. Thus altogether at least seven degrees of freedom of motion are required. Adding additional motors beyond seven makes it possible to bring the gripper to a desired orientation and position while still having degrees of freedom which offer the possibility of avoiding obstacles.

A photograph of our manipulator is shown in Fig. 2. The rectilinear geometric configuration was chosen to allow modularity in motor drives and to minimize the geometrical transformations required in motion control. Given the fact that motion in the direction of gravity is a convenient primitive, a case can be made for a Cartesian frame. The machine has a vertical size of about 1.8 m and a mass of about 300 kg. The machine was originally designed to be an articulated structure with 12 motors driving 12 joints. The convention was adopted to number these motors in the order of their articulation, starting at the base of the manipulator. To date, motors 5–7 have not yet been implemented, so in fact there are only nine degrees of freedom. Some of the joints are rotary, some are prismatic, and some involve more complicated linkages.

The first motor at the base of the machine is a rotary hydraulic motor. It is detented in integral multiples of 2.5 degrees. Peak speed is approximately 90 degrees/s.

The next three motors in the order of articulation are the z, y, and x prismatic joints. All three of these motors are identical linear hydraulic motors of a novel design [24]. A photograph of the motor is shown in Fig. 3. The only moving parts are four pistons which push 90 degrees out of phase from one another against a periodic cam seen

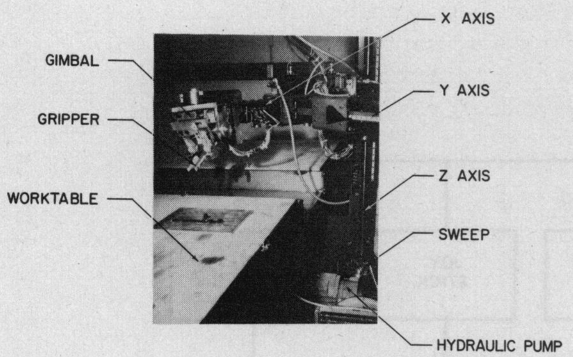


Fig. 2. Photograph of the computer controlled manipulator.

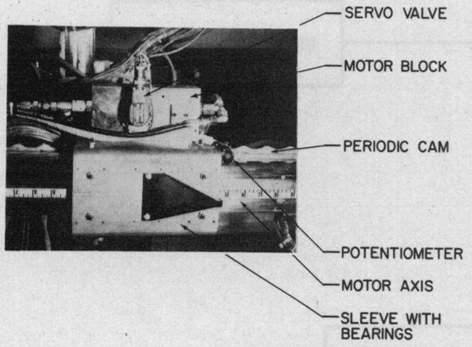


Fig. 3. Photograph of the linear hydraulic motor.

in the picture. Because each piston contains a spool valve, the motor is self commutating. The motor is therefore similar in behavior to a single hydraulic cylinder while not being subject to any corresponding limitation in travel. Each of the three axes has a travel of approximately 51 cm with a repeatibility of about 0.38 mm. For safety reasons the peak speed has deliberately been held to only 20 cm/s.

Motors 5–7 do not exist. These numbers have been reserved for fine z, fine y, and fine x motors, should they prove desirable in the future.

Motors 8–10 are the rotary joints of a roll, yaw, pitch gimbal. The first of these is hydraulic and the other two are electric. Speeds are on the order of 90 degrees/s.

Motors 11 and 12 drive the gripper through a complicated linkage which permits the opening angle of the fingers to be varied as well their separation. The gripper is shown in Fig. 4.

Sensors

The manipulator is provided with sensors in addition to the potentiometers which measure motor positions. The signals from these sensors are fed back to the System/7 as analog or digital inputs, so that they are available to the software system for the strategic control of motors.

The first of these sensors is referred to as the wand. This is a thin probe, built into one of the fingers of the gripper, which gives a contact signal when it is deformed slightly by touching an object. The wand and its housing can be seen in Fig. 4.

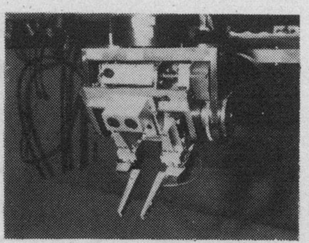


Fig. 4. Photograph of the gripper.

A second type of sensor is a strain gauge array which is built into the base of each finger. In the current manipulator each array measures three components of strain, although designs are complete for a full six degree of freedom array. The resolution is better than 5 g over a range of 1.5 kg.

An ultrasonic proximity sensor is mounted on the gripper just above the fingers as shown in Fig. 4. This sensor can measure the distance from an object with an accuracy of about 2.0 mm over a range of about 0.5 m.

Other sensors are under active development and will be incorporated into the manipulator at a later date. These include fluidic, optical, and area tactile devices.

Computers

The manipulator hardware described above is controlled by an IBM System/7, which is a 16-bit word length machine intended for sensor based applications. The configuration used here has 16K words of memory and a single disk with 1.2M words of storage. There are 21 digital input words, 3 process interrupt words, 16 digital output words, 64 analog input points, and 8 analog output points.

The machine is basically a data acquisition and control device and the host communication is via a 50K baud asynchronous communications module. I/O is also possible to a terminal which supports keyboard interaction as well as paper tape. Initial program load (IPL) is normally performed from the host, but in the present application in which the System/7 is being used primarily in a stand alone minicomputer role, a local bootstrap loader was written to allow IPL from the disk in about 1 s rather than the usual 15 s.

The basic cycle time of the System/7 is only 400 ns, and, because of the interrupt level structure, interrupts can be serviced in less than 2 μs. However, all input–output is by direct programming with the exception of the disk which is provided with a cycle steal feature. In addition, multiply and divide and floating point arithmetic are software functions.

The timing of the system is dominated by the response of the hydraulic components of the hardware. The motors in the manipulator had as a design goal an average of one second per operation. The bandwidth of the hydraulic system is about 50 Hz. A sampling rate of 100 Hz is used in the control cycle. In this 10 ms period, the state of the system must be sampled, a correction must be computed, and each control variable must be output. Thus, every 10 ms, all motor positions and sensor readings are taken and commands are given to all motors.

These real-time constraints in addition to normal system overhead resulted in the decision to design closed-loop servo system outboard of the System/7 and supply only set points to them. Since the actual position of all motors is monitored in the System/7 anyway, direct digital control may be retrofitted at a later date as time permits.

At present the sampling, computation, and control cycle of 10 ms consumes about 80 percent of the System/7 power. A dramatic change in utilization can of course be made by reducing the bandwidth of the servos and increasing the sampling time. So far there has been no need to take this step.

III. SOFTWARE SYSTEM

General Description

The software system described in this paper is solely the low-level system which was operational about a year ago. The principal improvement since that time has been the provision of an off-line symbolic assembler language which will not be described here. It is clear that higher level languages are necessary if manipulators are ever to perform highly complex assemblies with only moderate programming effort.

The software system which controls the manipulator hardware will be referred to as the manipulator operating system (MOS). MOS has two main functional levels, the higher priority of which is concerned with the tactical problems associated with monitoring sensory input and providing real-time control of motors. The lower priority level is concerned with the strategic problems of assembly as they are represented in the form of programs written in manipulator language (ML). On this lower level, MOS interpretively executes ML programs and interacts with the user at the terminal. The user is also provided with a function keyboard and joystick which can be used to enter geometric data to the manipulator by guiding it through a sequence of points. From his terminal, the user may write and debug ML programs, may direct MOS to execute these programs normally or step through them slowly, and may inspect or modify control tables.

MOS is designed to allow two independent manipulator arms to run concurrently at the same work station. Up to the present, only the "right" arm has been installed, although there are plans to add a "left" arm of very dissimilar design in the near future. The system provides the capability of synchronizing the two arms at the background level, but there is no provision for them to cooperate at the real-time level. There is also no provision for collision avoidance.

The System/7 normally operates in a stand alone mode although MOS is linked to a program running under VM on an IBM System/370 Model 145. This link provides certain functional capabilities which are not essential to manipulator operation. Thus although the stand alone ML is at a very low level, the provision exists for a much higher level language in VM to invoke any of the ML functions and for ML to be the target language for a VM resident assembler or compiler. These VM functions will be discussed only very briefly in this paper.

Safety

Safety is of paramount importance, since an errant manipulator could harm people, equipment, or itself. As a safety precaution, the manipulator operates inside an interlocked room with Lexan windows, and the work tables shown in Fig. 2 contain structural fuses and interlocks.

Unfortunately, formulation of adequate safety rules is almost impossible since exceptions which would cause trouble can almost always be found. The set of safety rules chosen is simple and reasonably complete. The fundamental assumption built into the software is that it is always safe for the manipulator to remain stationary.

A second highly important safety provision is that a fail–soft system supplies electric power to the hydraulic pump which drives all the hydraulic motors. When this electric power is turned off, the manipulator is in a relaxed state.

When the hydraulic pump is enabled, the manipulator can actively maintain its position or move to a new position. In the enabled state, possible emergency conditions include class interrupts on the System/7 (program check, machine check, power–thermal warning), certain programming errors detected by MOS, the user pushing a panic button or tripping an interlock, and unexpected collisions of the manipulator with its environment. When an emergency condition is encountered, MOS immediately commands all motors to freeze at their current positions and turns off the hydraulic pump.

MOS Tables

In order to permit easy debugging and make efficient use of memory, MOS is a table driven system. Naive users need not concern themselves with the nature and content of these tables. Commands are provided in ML, however, which permit knowledgeable users to interact directly with the internals of MOS.

There are altogether about 30 tables, each having a distinct two-character table name. Within any particular table the format consists of a fixed number of lines, each containing a fixed number of words.

The tables are divided into two classes called internal and external. Internal tables are those which are used by the real-time level of MOS. For reasons of performance, these tables must remain permanently resident in main memory. External tables are those which are used only by the background level of MOS and are therefore accessed infrequently. The external tables are maintained on disk and are swapped into page frames in main memory as needed. These tables are few in number but large in size, so that the software paging in effect creates a virtual machine environment of about 100K words.

The three tables of greatest importance in MOS are termed the motor, sensor, and task tables. The first two of these are internal and the last one is external.

The motor table contains data relevant to each motor in the manipulator. These data include the state of the motor in recent sample times as determined by feedback potentiometer, and strategic and tactical goals for the motor, as well as control, and input–output parameters.

The sensor table contains data relevant to each logical sensor in the manipulator. Each logical sensor is either a physical analog or digital input or it is computed by adding the ON bits in a masked digital input word. This count is useful for implementing threshold functions on some types of digital sensors. The sensor table also includes emergency limits for each sensor, upper and lower expected limits, the current reading, and a flag which tells whether or not the expected limits have ever been exceeded, as well as input and conversion parameters.

The task table contains the entire stored ML program of up to 5000 statements. Space in this table may also be used to store points in space to which the manipulator can later be commanded to move.

Real-Time System

Every 10 ms the real-time system samples the state of the world by reading all analog and digital inputs and entering these state data into the motor and sensor tables. The sensor readings are compared to the expected and emergency limits to determine if a force event has occurred. If so, the strategic goal in the motor table is replaced by the current position, and the background level is posted. Next, the actual motor positions are compared to the strategic goal to determine if a goal has been reached. If so, the background level is posted. Finally, control routines compute tactical goals from the strategic goals, and commands are output to all motors. The entire sample and control cycle consumes about 8 ms out of every 10 ms.

ML Syntax

The user inputs ML commands from the terminal keyboard. The basic syntax of each command is

(sequence number) (command code) (parameters).

For example, the following are valid user inputs:

```
250 BRANCH 700
408 MOVE 0 &5 0 5 0 0 0 −50 0 0 8 −&57
827 GOPOINT &17 1

BRANCH 700
MOVE &3 &81 0 5 0 0 0 &120 0 0 8 −7
GOPOINT 28.
```

Sequence numbers may be in the range from 1 to 5000. From the examples above it should be clear that the sequence number is optional. If a sequence number is absent, the command is executed immediately. If a sequence number is present, the command is not executed; instead it is filed as a part of a stored ML program which can be invoked at a later time.

From 0 to 15 parameters are allowed, the number depending on the particular command code. All parameters may be constants or variables. Constants are fixed point integers in the range from −32768 to +32767, these values being determined by the 16-bit word size. Variables are of the form &1 through &256, and they may be assigned any 16-bit fixed point integer.

Chained &'s are permitted to a depth of six. Thus if &17 is assigned 25 and &25 is assigned 700, then the following two commands are equivalent:

```
BRANCH &&17
BRANCH 700.
```

ML Command Codes

There are about 100 ML command codes. These commands can be divided into nine main categories: control, flow, motion, sense, data, arithmetic, edit, test, and host. The various command codes are itemized below.

Control: DISABLE, ENABLE, EXIT, LEFT, PRINT, RIGHT, RUN, TRACE, TRY, TN (where TN is any valid two character table name).

DISABLE and ENABLE turn the hydraulic pump off and on, respectively. LEFT and RIGHT set the context in which all subsequent commands pertain either to the right or left of the two manipulators. The other commands are self explanatory.

Flow: BAL, BRANCH, BRCOMP, BROVFL, DELAY, GOSUB, IF, LOOPI, LOOPJ, NEXTI, NEXTJ, STOP, RETURN.

GOSUB and RETURN offer a very limited subroutine capability. BAL performs a branch and link operation useful for subroutine calls with greater generality. BRANCH performs a branch which may be conditioned on a logical combination of sensors. There are also branches which test for overflow or the result of a comparison. IF performs a branch to any of three addresses depending on whether an argument is less than, equal to, or greater than zero.

Motion: DEFPOINT, DMOTOR, DMOVE, GOPOINT, JOY, HAND, MOTOR, MOVE.

MOVE positions the entire manipulator at a point in 12 space. DMOVE is a differential move which specifies the change in all 12 motors. MOTOR and DMOTOR are the analogous commands when only one motor at a time is moved. HAND moves only the gripper fingers. DEFPOINT and GOPOINT, respectively, store the present position in 12 space and go to this stored position. JOY permits the joystick and function keyboard to be used to guide the manipulator about and define a stored position.

Sense: CALIB, SENSOR, WAND.

CALIB is used to dynamically calibrate sensors. SENSOR sets expected threshold limits for the various sensors, of which the wand is one.

Data: POINT.

This nonexecutable command saves space in the ML program for a 12-space position defined by DEFPOINT or JOY. This POINT data may then be used by a GOPOINT.

Arithmetic: LOAD, DLOAD, ADD, DADD, MULT, DIVD, GET, PUT, GETSEN, GETPOS, GETGOL, LT?, LE?, GT?, GE?, EQ?, NE?, COMPON, COMPOFF, OVFLON, OVFLOFF.

The first six of these commands perform integer arithmetic in single or double precision. GET and PUT are the table access commands. The commands GETSEN, GETPOS, and GETGOL assign to a variable the current value of a sensor, a motor position, or a motor goal, respectively. The remaining commands perform comparisons and set an overflow indicator on or off.

Edit: ERASE, FILE, LIST, QUERY, QPOINT.

ERASE is used to erase sections of an ML program. LIST produces a listing of the ML program on the terminal or storage scope. FILE and QUERY are used for direct user interaction with MOS tables. QPOINT displays the most recent POINT's accessed by DEFPOINT and GOPOINT.

Test: TESTAI, TESTAO, TESTDO.

These commands are used for testing analog input, analog output, and digital output.

Host: CMSLOAD, CMSDUMP, VMCNTL, QUIT, UPROCA, UPROCB, UPROCC, UPROCD, GROT, GTRANS, GOLINE, GOMEAN, DHAND.

CMSLOAD and CMSDUMP can transfer entire ML programs between the System/7 and the host 370/145. VMCNTL is used to start a much higher level language interpreter on the host. The UPROC commands allow the user to provide his own Fortran semantic routines on the host. GROT and GTRANS provide rotation and translation of the manipulator to be specified in the coordinate frame of the gripper. GOLINE moves to the nearest point on a line and GOMEAN moves to the mean of two POINTS. DHAND is a differential HAND command.

Guarded Moves

A superficial analysis of dozens of ML programs written by several users for various experimental applications indicates that there is a basic sequence of ML statements. This sequence can be characterized as a guarded move, i.e., a move until some expected sensory event occurs. In this respect ML is similar to the Mantran language [25]. The sequence of ML statements which corresponds to a guarded move consists of multiple SENSOR commands followed by multiple motion commands followed by multiple BRANCH commands. The guarded move is illustrated by the ML program below:

```
200 SENSOR 5 −178 450
210 SENSOR 8 0 1000
215 SENSOR 14 −800 −600

280 GOPOINT 5
285 MOVE 0 0 2000 0 0 0 0 0 4000 −9000 650 1900
288 MOTOR 3 4000
295 DMOTOR 2 −100

400 BRANCH 500 410 5 14
410 BRANCH 550 600 8

500 etc.
550 etc.
600 etc.
```

Statement 200 establishes for logical sensor number 5 a lower expected limit of -178 and an upper expected limit of 450, and it clears a flag associated with this sensor. MOS will set this flag on when sensor 5 first deviates from the interval

$$-178 < \text{logical sensor number } 5 < 450.$$

In a similar fashion, statements 210 and 215 clear flags and establish lower and upper expected limits for logical sensors numbers 8 and 14.

Motion is permitted only as long as all sensor flags are off. The moment any sensor flag comes on, MOS overrides any motion command in execution and commands the manipulator to freeze at its current position. In addition, all subsequent motion commands become nullified, and ML program control falls right through these motion statements.

Statements 280 through 295 are motion commands. Statement 280 tells the manipulator to move to point number 5 which must have been previously defined. Once this motion is completed, statement 285 tells the manipulator to move its motors according to the parameter vector. As mentioned earlier, the numerical ordering of the motors corresponds to the order of their articulation. Linear dimensions for the prismatic joints are given in units of 0.001 in and angular dimensions for the rotary joints are given in units of 0.01 degrees. Thus, for example, statement 285 commands motor 3 to move to the absolute position 2.000 in. Next, statement 288 commands only motor 3 to move to the new absolute position 4.000 in. When this motion is completed, statement 295 commands motor 2 to move differentially a distance -0.100 in from its current position.

Suppose that during the execution of statement 288, logical sensor number 5 went out of its expected range just as motor 3 was passing the position 2.750 in. MOS would immediately override statement 288 and directly command motor 3 to remain at position 2.750 in. All other motors are also commanded to remain at their current positions, but in this particular instance they were stationary anyway. Because a sensor event has

```
**********************************************************
* INITIALIZE VARIABLES TO DEFINE FORWARD DIRECTION   *
**********************************************************
   1 LOAD 1 -1000
   2 LOAD 2 0

************************************************************
* GUARDED MOVE IN FORWARD DIRECTION UNTIL FIRST HIT   *
************************************************************
   3 SENSOR 1 -1 1
   4 DMOVE 0 0 &1 &2
   5 BRANCH 13 4 1

***************************************************************
* GUARDED MOVE DIFFERENTIAL STEP IN FORWARD DIRECTION *
***************************************************************
   6 SENSOR 1 -1 1
   7 DMOVE 0 0 &1 &2
   8 BRANCH 13 9 1

************************************************************
* IF NO HIT THEN TURN LEFT FOR NEW FORWARD DIRECTION  *
************************************************************
   9 LOAD 3 &2
  10 LOAD 2 &1
  11 LOAD 1 -&3
  12 BRANCH 6

********************************************************
* IF HIT THEN BACK OFF ONE DIFFERENTIAL STEP        *
********************************************************
  13 SENSOR
  14 DMOVE 0 0 -&1 -&2

*************************************************************
* GUARDED MOVE DIFFERENTIALLY ONE STEP TO THE RIGHT   *
*************************************************************
  15 SENSOR 1 -1 1
  16 DMOVE 0 0 &2 -&1
  17 BRANCH 18 6 1

***********************************************************
* IF HIT THEN TURN RIGHT FOR NEW FORWARD DIRECTION    *
***********************************************************
  18 LOAD 3 &2
  19 LOAD 2 -&1
  20 LOAD 1 &3
  21 BRANCH 13
```

Fig. 5. ML curve follower program.

occurred, statement 295 is nullified, and execution falls through directly to statement 400.

Statement 400 asks whether logical sensors 5 or 14 have ever gone out of range. If the answer were no, control would branch to statement 410. However, in this particular example it is assumed that sensor 5 has triggered, so control will branch to statement 500.

Host ML Commands

A subset of the ML commands invoke communication with a program running under VM on an IBM System/370 Model 145. Because these commands are not essential for MOS operation, MOS can be thought of primarily as a stand alone system.

There are three main functions which are obtained through the link to VM. First, the link is being used for development work on a language for manipulator control which is on a much higher level than ML. Secondly, through CMSLOAD and CMSDUMP commands it is possible to transfer entire ML programs between machines. This facility permits the loading of ML object code generated offline by the symbolic assembler. It also permits users to prepare ML programs offline with the full power of the CMS editor. Thirdly, VM can be used to circumvent the limitation on program size and speed inherent in the 16K word System/7. Thus, for example, motion in the coordinate frame of the gripper is provided by the VM program. Such computations are sufficiently compute bound in ML to be worth invoking the communications overhead of being shipped to VM. Finally, individual users are permitted to code their own special user procedures which are already in ML as commands UPROCA,···, UPROCD.

IV. EXPERIMENTAL APPLICATIONS

Curve Follower

As an example of an ML program, a curve follower program is instructive to the extent that it involves the sort of sensory feedback which is used for pattern recognition in more advanced assembly work, and it is the simplest that can be used in this paper for pedgogical purposes. The ML program shown in Fig. 5 performs very simplistic curve following of a nonconvex object lying in the xy plane. Because of the need to guard every move, even when retracting to a supposedly clear position, this program is more complex than would be required in video curve following.

It is assumed that at the start of execution the manipulator is already at a good starting position in the correct xy plane, the wand has been assigned to logical sensor number 1, and it normally reads 0. The command

SENSOR 1 −1 1

is therefore used to guard a move until the moment that the wand reading deviates from 0, i.e., until the wand strikes anything. The command

SENSOR

sets all sensor limits to plus and minus infinity, effectively unguarding subsequent moves.

Each time it moves forward and bumps into the object the manipulator backs off, moves to the right, and then repeats the cycle. When the forward move fails to produce contact, the program assumes the manipulator has moved too far to the right, so it rotates the search pattern 90 degrees to the left. If, on the other hand, the move to the right ever produces contact, the program assumes the manipulator has encountered a concavity, so it rotates the search pattern 90 degrees to the right. This particular ML program is not able to detect when it has completed a circuit around the object.

Current Work

In the initial learning phase of the project, programs were written to stack toy blocks and assemble a toy train. Work is now concentrating on real and demonstrably useful assembly tasks. The overall assembly function was divided into three subareas, each of which is being actively pursued. The first area deals with the problems of reaching into a pile of disordered but homogeneous parts, extracting one, and performing some superficial inspection. The second area deals with the problem of orienting a known part which is held in the manipulator gripper. The third area deals with the assembly of palletized parts, i.e., parts which are already in known positions and orientations.

Using a symbolic assembler language which generates ML object code, programs have been written which to a limited degree perform the three operations of bin search, orientation, and assembly of a rail support from a typewriter, including the operation of picking up a power screwdriver and inserting screws. This real-world assembly consists of over 20 parts and involves motion control and sensing to the tolerances required in industrial practice. Work is continuing on assemblies which are spatially and kinematically complex.

V. CONCLUSIONS

This paper has described a system built to allow experiments to be performed in the area of computer controlled assembly of mechanical objects. The hardware has sufficient accuracy and freedom of movement to be able to perform all but the most intricate movements required in the application. The software described in the paper covered the operating system and low-level control language only. High-level languages for part description, orientation, trajectory and inspection specification, and procedures for the programming of two cooperative manipulators are under study.

ACKNOWLEDGMENT

Many present and past contributors to this work must be acknowledged. The hardware in the system is due to the contributions of A. Brenneman, E. Carey, N. Diana, R. Garrison, H. A. Panissidi, G. Shelton, S. Wang, and many members of the Central Scientific Services Department. The software owes much to R. Evans and S. S. Soo. The symbolic assembler was written by D. Garnett. The programming of experimental applications was heavily dependent on the efforts of M. Blasgen, J. Griffith, L. Lieberman, and M. Wesley. We also acknowledge the contributions of former group members E. Ayoub and W. Book.

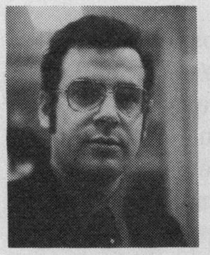
REFERENCES

[1] M. D. Kilbridge and T. O. Prenting, "Assembly: the last frontier of automation," *Management Rev.*, pp. 16–18, Feb. 1965.
[2] T. Alexander, "The hard road to soft automation," *Fortune*, p. 95, July 1971.
[3] W. V. Tipping, *An Introduction to Mechanical Assembly*. London, England: Business Books, 1969.
[4] For example, see *Automated Assembly*, 18 vols. London, England: Institution of Production Engineers, 1970.
[5] D. Michie, "Machines and the theory of intelligence," *Nature*, vol. 241, pp. 507–512, Feb. 23, 1973.
[6] A. Hutchinson, *Labanotation*. New York: Lauglin, 1961.
[7] R. E. Ornstein, *The Psychology of Consciousness*. San Francisco, Calif.: Freeman, 1972.
[8] W. H. P. Leslie, Ed., "Numerical control programming languages," in *Proc. PROLAMAT Conf.*, Rome, Italy, 1969. Amsterdam, The Netherlands: North-Holland, 1970.
[9] *Proc. First Nat. Symp. Industrial Robots*, Res. Inst., Illinois. Inst. Technol., Chicago, 1970; also in *Proc. Second Int. Symp. Industrial Robots*, Res. Inst., Illinois Inst. Technol., Chicago, May 1972; also in *Proc. Third Int. Symp. Industrial Robots*, Zurich, Switzerland, May 1973.
[10] *Industrial Robots—A Survey*, Int. Fluidics Services, Ltd., Bedford, England, 1972.
[11] M. Ejiri, T. Uno, H. Yoda, T. Goto, and K. Takeyasu, "A prototype intelligent robot that assembles objects from plane drawings," *IEEE Trans. Comput.*, vol. C-21, pp. 161–170, Feb. 1972.
[12] *Special Issue of the Bulletin of the Electrotechnical Laboratory of Japan*, vol. 35, no. 3, Mar. 1971.
[13] J. A. Feldman *et al.*, "The Stanford hand–eye project," in *Proc. First Int. Joint Conf. Artificial Intelligence*, London, England, Sept. 1971, pp. 350–358.
[14] N. J. Nilsson and B. Raphael, "Preliminary design of an intelligent robot," in *Computer and Information Sciences*, vol. II. New York: Academic, 1967, pp. 235–260.
[15] Massachusetts Inst. Technol., Cambridge, 1971 MAC Progress Rep. VIII.
[16] J. L. Nevins, D. E. Whitney, and S. N. Simonovic, "Report on advanced automation: system architecture for assembly machines," Charles Stark Draper Lab, Cambridge, Mass., Rep. R764, Nov. 1973.
[17] C. A. Rosen, "The future of automation," *Nav. Res. Rev.*, pp. 1–15, 1973.
[18] R. Bolles and R. Paul, "The use of sensory feedback in a programmable assembly system," Artificial Intelligence Lab., Stanford Univ., Stanford, Calif., Memo 220, Oct. 1973.
[19] *Metalworking News*, Dec. 3, 1973 (discusses the expansion of the Machine Perception Section of the General Motors Computer Sciences Department).
[20] S. J. Wang, "ALFA: a language for automation," presented at the Milwaukee Symp. Automat. Contr., Milwaukee, Wisc., Mar. 28–30, 1974.
[21] A. P. Ambler, H. A. Barrow, C. M. Brown, R. M. Burstall, and R. J. Popplestone, "A versatile computer controlled assembly system," in *Proc. Third Int. Conf. Artificial Intelligence*, Stanford Univ., Stanford, Calif., Aug. 1973, pp. 298–307.
[22] W. B. Heginbotham, P. W. Kitchin, and A. Pugh, "Visual feedback applied to programmable assembly machines," in *Proc. Second Int. Symp. Industrial Robots*, Res. Inst., Illinois Inst. Technol., Chicago, May 1972.

[23] M. B. Ignatiev, F. M. Kulakov, and A. M. Pokrovskii, "Robot manipulator control algorithms," Leningrad, USSR, 1972; NTIS Trans. JPRS 59717, Aug. 1973.
[24] H. A. Panissidi and B. Gardineer, private communication, IBM Corp., Yorktown Heights, N. Y.
[25] D. J. Barber, "MANTRAN: a symbolic language for supervisory control of an intelligent remote manipulator," Massachusetts Inst. Technol., Cambridge, Eng. Projects Lab. Rep. 70283-3, June 1967.

Peter M. Will was born in Peterhead, Scotland, on November 2, 1935. He received the B.Sc. degree in electrical engineering and the Ph.D. degree, both from the University of Aberdeen, Aberdeen, Scotland, in 1958 and 1960, respectively.

During 1961 he worked at the Research Laboratory of Associated Electrical Industries, Ltd., Manchester, England, on adaptive control systems and from 1962 to 1965 he was with A. M. F. (in England and the United States) working on industrial electronics and manipulators. Since 1965 he has been a Staff Member at the IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., where his main interest was in image processing. Currently he is manager of the Automation Research Project, and his interests have returned to the problems of manipulators and their application.



David D. Grossman was born in New York, N.Y., on February 2, 1940. He received the B.A., M.A., and Ph.D. degrees, all in physics, from Harvard University, Cambridge, Mass., in 1961, 1962, and 1967, respectively.

Until 1970 he was a Physics Instructor at Princeton University, Princeton, N.J., commuting to research activities at the electron–positron colliding beam machine in Frascati, Italy. In 1970 he switched fields, becoming a Staff Member in the Computer Sciences Department at the IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y. Since that time he has worked on image processing, data compression, optimum placement of records on a disk, and computer networks. As software project leader in the Automation Research Project, he coordinated the development of the initial system used to control the manipulator.

Image Data Processing by Hadamard-Haar Transform

K. R. RAO, SENIOR MEMBER, IEEE, M. A. NARASIMHAN, AND KRISHNAIAH REVULURI

***Abstract*—A hybrid version of the Haar and Walsh–Hadamard transforms (HT and WHT) called Hadamard–Haar transform (HHT)$_r$ is defined and developed. Efficient algorithms for fast computation of the (HHT)$_r$ and its inverse are developed. (HHT)$_r$ is applied to digital signal and image processing and its utility and effectiveness are compared with other discrete transforms on the basis of some standard performance criteria.**

***Index Terms*—Data compression, digital time processing, feature selection, Hadamard–Haar transform (HHT)$_r$, Wiener filtering.**

I. INTRODUCTION

DIGITAL signal and image processing has come into prominence in recent years. This requires, in many cases, utilization of discrete orthogonal transforms [1], [2]. Fourier [3], slant [4], Walsh–Hadamard [5], Haar [1], [6], [7], discrete linear basis [8], rapid [9], [10], slant Haar [11], and discrete cosine [12] have already been utilized in these areas. This utilization is stimulated in part by the rapid development of digital hardware. Efficient algorithms for fast implementation of the orthogonal transforms have further accelerated their effectiveness, leading to the design and development of special-purpose digital processors tailored for specific transforms. Since the linear transformation of image data results in compaction of its energy into fewer coefficients [13], image processing by transform techniques can lead to lower transmission rates with negligible image degradation [4], [14].

Manuscript received August 1974; revised March 1975.
K. R. Rao is with the Department of Electrical Engineering, University of Texas at Arlington, Arlington, Tex. 76019.
M. A. Narasimhan is with Texas Instruments, Inc., Dallas, Tex.
K. Revuluri is with the Department of Information Engineering, University of Illinois, Chicago Circle, Ill. 60680.

II. HADAMARD–HAAR TRANSFORM [15]

The objectives of this paper are to develop a hybrid version of the well-known Walsh–Hadamard (WHT) and Haar transforms (HT) such that the advantages of both of