The Early Stages of Robotics

Richard P. Paul

ABSTRACT: The early stages of robotics are recalled by tracing the evolution of robot manipulator programming languages. Initial manipulator control was by means of procedures embedded in a high-level language. This evolved into a geometric-based planning and execution system where planning was time-independent and execution was time-efficient. With the addition of sensor feedback and the need to develop the ad hoc procedures involved in assembly, an interpretive system evolved. Finally, we have come full circle with manipulator control once again embedded in a high-level language. This is now possible because of the development of high-level languages, the simplification of manipulator-control eliminating the need to plan trajectories and to precalculate dynamics, and the increasing computer power available to provide for the economic real-time control of the manipulator.

Introduction

The present-day industrial robot has its origins in both teleoperators and numerically controlled machine tools. The teleoperator is a device to allow an operator to perform a manual task from a distance. The numerically controlled machine tools shape metal automatically, based on digitally encoded cutting data.

The teleoperator was developed during the second world war to handle radioactive materials [1]. An operator was separated from a radioactive task by a concrete wall with one or more viewing ports through which the task could be observed. The teleoperator was a substitute for the operator's hands; it consisted of a pair of tongs on the inside (the slave) and two handles on the outside (the master). Both tongs and handles were connected by linkage mechanisms to provide for arbitrary positioning and orientation of the master and slave. The mechanical linkage caused the slave to replicate the motion of the master.

In 1947, the first servoed electric-powered teleoperator was developed. The slave was servo-controlled to follow the position of the master. Force information was, however, no longer available to the operator, and tasks requiring parts to be brought into contact were difficult to perform. To quote Goertz, “The general-purpose manipulator may be used for moving objects, moving levers or knobs, assembling parts, and manipulating wrenches. In all these operations, the manipulator must come into physical contact with the object before the desired force and movement can be made on it. A collision occurs when the manipulator makes contact. General-purpose manipulation consists essentially of a series of collisions with unwanted forces, the application of wanted forces, and the application of desired motions. The collision forces should be low, and any other unwanted forces should also be small” [2]. In 1948, a servo system was introduced in which the force exerted by the tongs could be relayed to the operator by back driving the master; the operator could once again feel what was going on.

In 1948, faced with the need to procure advanced planes whose parts were designed to be machined rather than riveted, the Air Force sponsored research in the development of a numerically controlled milling machine [3]. This research was to combine sophisticated servo system expertise with the new, developing digital computer techniques. The pattern to be cut was stored in digital form on a punched tape, and then a servo-controlled milling machine, using the tape as input, would cut the metal. The MIT Radiation Laboratory was awarded a subcontract and demonstrated such a machine in 1953.

The Industrial Robot

In the 1960s, George Devol demonstrated what was to become the first Unimate industrial robot, a device combining the articulated linkage of the teleoperator with the servoed axis of the numerically controlled (NC) milling machine. The industrial robot could be taught to perform any simple job by driving it by hand through a sequence of task positions that were recorded in digital memory. Task execution consisted in replaying these positions by servoing the individual joint axes. Task interaction was limited to opening and closing the tongs or end-effector and to either signaling external equipment or waiting for a synchronizing signal. The industrial robot was ideal for pick and place jobs, such as unloading a diecasting machine (see Fig. 1). The part would appear in a precise position, defined with respect to the robot; it would be grasped, moved out of the die, and dropped on a conveyor. The success of the industrial robot, like the NC milling machine, relied on precise, repeatable digital servo loops. There was no interaction between the robot and its work. If the diecasting machine were moved, the robot could in no way adapt to the new position, any more than an NC milling machine could successfully cut a part if the stock were arbitrarily relocated during cutting. If the diecasting machine were moved, the robot could, however, be taught to do the job.

The success of the industrial robot lay in its application to jobs in which task positions were absolutely defined and in its reliability and positioning repeatability in lieu of adaptation. The industrial robot was a piece of automation like a transfer line component; however, the position sequence could be taught to it directly — there were no cams or gears to cut.

Automation in the form of an industrial robot was different from all previous forms of automation. In previous models, some machine was introduced, which performed the task faster or differently from the existing process; in the case of the industrial robot, a worker performing a job was replaced by a machine having an anthropomorphically identifiable arm, which performed the job in much the same way as the replaced operator. Both forms of automation increase productivity.

The Sensor-Controlled Robot

Simultaneously with the development of the industrial robot, an attempt was made to automate the teleoperator, an attempt made possible only by the development of digital computers. A device, which we shall refer to as a sensor-controlled robot, was proposed by Shannon and Minsky in 1958. The sensor-controlled robot was to consist of a teleoperator equipped with all forms of sensors connected to a computer. The computer was to be informed of a goal and the robot, by means of its sensors, would size up the envi-
environment and decide on the actions necessary to accomplish the required goal. Although this device was never built, Ernst [4] built a robot with touch sensors located in the hand, which could be programmed to perform tasks such as locating and picking up blocks and putting them in a box (see Fig. 2). Programming was in the form of instructions such as

"move in direction $x$ with speed $v$ until sense element $s$ indicates a "or" if sense element $s$ indicates $1$, go to the next instruction, otherwise continue the same action." A program of 600 lines of code, made up of instructions of this form resembling an assembly language program was required for the block program. The lack of any global idea of the position of objects limited this robot as much as the complete lack of task information limited the position-controlled industrial robot.

Vision

In 1963, Roberts demonstrated the feasibility of processing a digitized halftone picture of a scene to obtain a mathematical description of the blocklike objects in the scene, expressing their location and orientation by homogeneous transformation [5]. This work was important for two reasons: it demonstrated that objects could be identified and located in a digitized halftone image, and it introduced homogeneous transformations as a suitable data structure for the description of the relative position and orientation between objects. If the relative position and orientation between objects is represented by homogeneous transformations, the operation of matrix multiplication of homogeneous transformations can establish the overall relationship between any two objects [6,7].

Hand-Eye System

Touch feedback, because of its slow, groping nature, was dropped in favor of vision as an input mechanism. By 1967, a computer equipped with a television camera as an input mechanism could, in real time, identify objects and their location [8]. Scenes normally contained more than one object; the vision processing would first locate edges, then vertices, and finally identify all the objects in the scene. These objects were represented in a world model. The world model frequently contained prototypes of the possible objects that could comprise the scene, and after vision processing, the world model would contain a set of instances of these objects whose positions and orientation were described by homogeneous transformations. The manipulator, stripped of its touch sensors, would then rely on position-servoed joint axes. Homogeneous transformations, however, expressed the position and orientation of the end-effector in Cartesian coordinates, not as the angles between a series of nonorthogonal joints. Pieper was able to apply the theories of kinematics relating to closed-link chains to obtain a solution to this problem, and the manipulator could then be commanded to move to Cartesian positions in its workspace [9].

The manipulator was controlled by means of a small number of high-level language subroutines and functions similar in concept to the scientific library, which provides such familiar routines as SIN, COS, and SQRT. In the case of the manipulator, there was some form of initialization routine, and further routines to move the manipulator and to open and close its gripper. The manipulator was treated as an output device, much like a line printer, information was transmitted to it, and the required actions were performed. The routines shown in Table 1 provided the above functions.

When a routine was called, program execution was suspended until the action was completed. A manipulator program took the form of a sequence of subroutine calls, typically with constant arguments. Of the four subroutines listed, only CLOSE provided for any interaction with the environment in which the manipulator was working. If an object of a certain known size was to be picked up, the size could be provided to CLOSE in order to verify that the object was correctly acquired.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>SUBROUTINE</td>
<td>initialize the arm.</td>
</tr>
<tr>
<td>SERVO(i)</td>
<td>SUBROUTINE</td>
<td>move the manipulator until all the joints are as specified in the array $i$.</td>
</tr>
<tr>
<td>OPEN(DIST)</td>
<td>SUBROUTINE</td>
<td>open the gripper to DIST.</td>
</tr>
<tr>
<td>CLOSE(DIST)</td>
<td>FUNCTION</td>
<td>close the gripper as far as it will close and return FALSE if the opening is less than DIST.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>SUBROUTINE</td>
<td>initialize the arm.</td>
</tr>
<tr>
<td>SERVO(i)</td>
<td>SUBROUTINE</td>
<td>move the manipulator until all the joints are as specified in the array $i$.</td>
</tr>
<tr>
<td>OPEN(DIST)</td>
<td>SUBROUTINE</td>
<td>open the gripper to DIST.</td>
</tr>
<tr>
<td>CLOSE(DIST)</td>
<td>FUNCTION</td>
<td>close the gripper as far as it will close and return FALSE if the opening is less than DIST.</td>
</tr>
</tbody>
</table>

Fig. 1. The Unimate robot at work diecasting.

Fig. 2. The first sensor-controlled robot.
The arguments to SERVO were the six joint coordinates of the desired position of the arm. These were typically integer variables specifying the nonorthogonal coordinates of the manipulator joints. The form of motion whether joint by joint, all starting off together, coordinated, etc., was not specified. This servo routine, by itself, was not very useful, but when coupled with two solution subprograms, it provided the necessary support of the eye system. HANDPOS (T(4,4))—a subroutine—returned a 4 by 4 homogeneous transformation expressing the Cartesian coordinate position and orientation of the gripper. SOLVE (J(6), T(4,4))—a function—returned the appropriate joint coordinates J to position the gripper in the Cartesian coordinate position and orientation specified by a transform T. The routine returned FALSE if it was not possible to position the manipulator at T.

The basic concept of the system was that the manipulator was to be used as an output device for the vision system. The vision system would analyze a scene and locate objects. The necessary manipulator position would be calculated in order to grasp each object and SOLVE called to obtain the joint coordinates. The manipulator would then be moved by calling SERVO and the object grasped by calling CLOSE. Typical scheduling of the computation is shown in Fig. 3. There were three levels of control: at the top level, calls were executed; at the next level, SOLVE converted positions into joint coordinates; and at the lowest level, the manipulator was servoed.

In order to move an object, a series of manipulator positions is required: a safe approach position, the grasping position, the lift-off position, the approach position for the object, etc. In the hand-eye system, the manipulator moved and came to rest at each of these positions while the necessary joint angles for the next position were computed. While this was of little consequence in the hand-eye system, it is extremely wasteful of time and energy in general.

The hand-eye system differed from a conventional industrial robot in two ways. The motion and gripper primitives were embedded in a high-level computer language, and the manipulator was programmed symbolically in Cartesian coordinates. As the manipulator was merely an output device, the embedding of the manipulator primitives in a high-level language did not produce much benefit over conventional systems; there were no tests and, thus, no use for conditional statements. The use of Cartesian coordinates was, however, an important step and was to form the basis of a world model that would allow for decision making, planning, and task verification.

This form of manipulator programming is awkward; as basic data types representing positions and orientations do not exist in the high-level language, the program is executed step by step, requiring that the manipulator be stopped at each program step. The overhead of a compiler is not warranted, as manipulator programs are executed at a very low statement rate and an interpreter would provide more flexibility and better debugging interaction.

### Blocks World System

By 1970, a camera- and arm-equipped computer could play real-world games and the "instant insanity" puzzle was successfully solved at Stanford University [10]. In this puzzle, four cubes with different-colored faces must be stacked so that no two similar colors appear on any side. At MIT, a block structure could be observed and copied. In Japan, research led to a hand-eye system that could assemble block structures when presented with an assembly drawing [11]. In this system, a world model of fixed objects and of objects located by the vision system was represented as instances of prototype objects whose orientation was described by homogeneous transformation [12]. The manipulator system was told what object to move and the position to which it should be moved. The manipulator system then determined stable grasping position. It also determined whether the object needed to be set down and regrasped in order to be moved to the specified destination [13]. A collision avoider (never implemented) was to determine a safe path for the arm through the world model, describing the path as a sequence of homogeneous transformations. In order to avoid stopping at every point making up such a path or trajectory, a continuous curve was fitted through the sequence of joint angles corresponding to the sequence of transformations making up the path.

A manipulator trajectory was specified as six sequences of joint angles through which the six joints were to pass in a time-coordinated manner. At the first and last points of each of the six trajectories, zero velocity and acceleration constraints were imposed. At all the intermediate points, continuity of velocity and acceleration was required. A sequence of polynomial splines was calculated to meet these requirements. Unfortunately, the spline fit was a lengthy procedure and could not be performed as the manipulator moved. However, in the blocks world system, as all the positions were known, the lengthy spline fit could be computed before the manipulator moved. This introduced the concept of a planning phase and a run time phase; see Fig. 4.

By dividing the task in this manner, the planning could be performed off-line, with no time constraints. Additional calculations relating to dynamics, servo gains, and offsets could also be calculated. At run time, no delays were incurred while the solutions for the next positions were obtained, and the resulting motion was smooth and continuous.

The use of trajectories had an important safety-side effect. In order to fit the spline through the trajectory points, the manipulator joint accelerations were specified as continuous, bounded, functions of time. During execution, actual joint accelerations were compared to the specified values by the controller and the motion aborted if the difference became excessive. This simple check resulted in years of accident-free experimental operation.

The new manipulator system consisted of just one high-level procedure, MOVE_INSTANCE, with two parameters, the name of the instance of a prototype to be moved, and the desired final transformation of the instance. Based on this information, trajectories and gripper commands were assembled into a file, which, when executed by the arm, resulted in the desired action. Interaction with the actual environment was limited to the calls on the gripper CLOSE command, which verified the correct object width at each grasp. By the addition of a planning phase, the step-by-step execution of a program could be turned into a continuous motion during execution. Programs still had to be written in a high-level language that lacked appropriate data types and operations.

While this system was excellent for generating graphics displays of an ideal arm moving ideal objects, problems occurred in a real environment. No interaction was specified between the arm and the environment; the arm simply moved through the space, opening and closing its gripper. At every interaction with the environment, the forces and torques generated were ignored. Consider, for example, the task of placing an object on
a surface. The arm is commanded to move the object to zero height, and the gripper opened. The position tolerance of this move is zero, for if the arm stops above the surface and opens its gripper, then the object is dropped, not placed. If the arm tries to move below the surface, it is stopped by the object while infinite forces are exerted on the object. The placement task should be specified to move the object toward the surface until an appropriate contact force is detected and the gripper, then, opened. Similarly, in grasping an object, the hand should be centered over the object using touch feedback; the fingers should not simply be closed, possibly displacing the object.

Force and touch feedback were added to the arm, in a rather crude manner, to perform the above functions. With the addition of this feedback came a great deal of sensory information from the environment. While the position of objects could be modeled using homogeneous transformations, and problem solvers could function with such a model, there was no model for the interactive forces, torques, and touches. This information was not used by the problem solver, other than to call vision to locate an unknown object. The manipulator programming became ad hoc and experimental. In order to meet the needs of this type of programming, the WAVE system was developed.

The Wave System

In the WAVE system [14], the procedure calls previously embedded in a high-level language could be typed in directly by name with parameters or read from a file. A macro facility was added to build simple sequences of manipulator primitives to make up higher level commands such as MOVE_INSTANCE had become. An on-line macro editor provided a quick interactive way of developing these macros. Motion force and touch commands became the primitives of the language. The WAVE system functioned in two modes: a planning mode in which instructions read in were assembled into a file for later execution and a direct mode in which each instruction was executed as it was typed in.

Provision was made for defining transformations by placing the gripper in the appropriate location and typing HERE, at which point the arm position was read and converted to a homogeneous transformation. These positions could then be raodified and verified by executing a direct move to the named transformation.

Initially, the sensor information was simply used to verify task execution, as CLOSE had been used. By adding tests and jump instructions, however, a first level of error recovery was built into the execution program, adding to the robustness and reliability of the programs.

The force and touch primitives in WAVE were modifiers of motion statements; for example, the WAVE instruction to insert a pin along the z axis of station coordinates is as follows:

```
FREE 2 X,Y; COMPLY WITH FORCES IN X AND Y
SPIN 2 X,Y; COMPLY WITH TORQUES IN X AND Y
STOP Z 100; STOP WHEN THE Z FORCE EXCEEDS 100
MOVE IN; WHILE MOVING TO POSITION 'IN'
SKIP N 23; IF THE STOPPING CONDITION WAS MET
;SKIP THE NEXT INSTRUCTION
JUMP EROC; JUMP TO ERROR RECOVERY ROUTINE
```

At the time the motion is terminated by the STOP instruction, indicating that the pin is fully inserted, the position and orientation of the arm are not specified by the transform IN as the motion has been allowed to translate in X, Y, and Z and to rotate about the X and Y axis. The position and orientation of the arm, however, accurately represent the pin-in-the-hole location. Provision was made to save this information in the form of a homogeneous transformation in case the arm needed to return to this position or to a position nearby. In assembling objects, quite complicated hierarchical saving algorithms were developed resulting in remarkably adaptive behavior of the manipulator in performing assemblies of parts of low tolerance.

The WAVE system still made use of a planning phase but eliminated the need to write programs in an inappropriate high-level language. The form of programming was similar to assembly language programming of computers and lacked structure.

The WAVE system was finally extended to run two manipulators, but in a very simple manner, chiefly through synchronization, with very limited interaction. The use of two manipulators, however, greatly simplified the task of programming assemblies as one hand could function as a jig, or fixture, for the other as needed.

Cartesian Motion

The WAVE system specified positions and orientations in Cartesian coordinates but moved the manipulator in joint coordinates. In order to provide a system capable of working on moving objects of known position and velocity, one must also move the manipulator in Cartesian coordinates. The position and orientation of the object, described by a homogeneous transformation, are simply postmultiplied by a second transformation representing the desired position and orientation of the gripper with respect to the object. The product yields a transformation representing the required base coordinate position and orientation of the manipulator. From this transformation, the joint coordinates can be obtained. These transformations must be performed at a rate sufficiently high to provide for continuous tracking motion of the manipulator. A system designed to function in this manner was developed at SRI [15] and performed these transformations at a 20 Hz rate in order to control an UNIMATE manipulator. The preplanned spline fit trajectories of WAVE were discarded in favor of a simple on-line method, which calculated trajectories segment by segment. The resultant Cartesian motion, although elegant, was simply a result of the necessity of combining Cartesian positions of objects when one object was in motion.

The programming style was also changed in that a move through a series of positions was programmed as a move to each individual position. During execution, the moves, through a series of positions, were turned into a continuous motion by the run-time trajectory calculator. During each move segment, a series of functions could be performed such as opening and closing the gripper. This form of programming was very similar to the original Unimate style of programming.

The main differences were the following:

1. Positions were specified in terms of homogeneous transformation products, one of which could represent a moving coordinate system, such as a conveyor.
2. Motions were made in a coordinated well-controlled manner.
3. Motions did not stop at each intermediate point but transitioned smoothly through intermediate points. This system eliminated the need for a planning phase.
Integrating Robot Control into High-Level Languages

With the elimination of the need for a planning phase, robot manipulator control could be represented as a sequence of program statements to move the manipulator from one position to another. A manipulator control process, much like an input/output device driver, would actually move the manipulator. The manipulator control process would also provide for smooth path motion transitions and for bringing the manipulator to rest when no further move statements were pending. The move statement would provide for synchronization between program execution and manipulator motion.

With the development of new high-level programming languages such as Algol 60, c, and Pascal, it became possible to represent the data structures necessary for manipulation directly into the high-level language. The integration of sensors with a manipulator could then be achieved by embedding the manipulator control directly into one of the above languages. Sensors would be treated as input devices, and all the well-understood control and data types of the language would be available to the robot programmer. References describe the embedding of manipulator control into PASCAL [16, 17], but any of the other languages would do as well.

Conclusions

We have traced the development of robotics through the evolution of robot manipulator programming languages. Initially, manipulator control was by means of procedures embedded in a high-level language. This evolved into geometric-based planning and execution systems where planning was time-independent and execution was time-efficient. With the addition of sensor feedback and the need to develop the ad hoc procedures involved in assembly, an interpretative system evolved. Finally, we have come full circle with manipulator control once again embedded in a high-level language. This has become possible because of the development of high-level languages, the simplification of manipulator-control eliminating the need to plan trajectories and to pre-calculate dynamics, and the increasing computer power available to provide for the economic real-time control of the manipulator.

The sensor input, computer-controlled robot is an extremely rich and interesting research area, which is beginning to develop as an independent discipline. In the past, it was seen as an interesting application of various research results, and this has resulted in a fragmented and chaotic development of robotics in general. While work in the area of sensors, actuators, kinematics, dynamics, structures, and end-effectors is of great importance, it is the area of language that distinguishes robots from all other devices. It is a tremendous undertaking to integrate the description of tasks, models, data reduction from sensors, and control of actuators into a unified system. Perhaps the greatest area of importance in language development is the identification of language primitives, that small set of instructions that will allow us to describe the infinite number of tasks a robot can perform. A closely related area is that of modeling; at present, we rely almost entirely on the geometric models of computer graphics. What kinds of models are necessary to relate force, touch, and vision? How would we model the randomly distributed parts of a clock, such that the robot could deduce the correct assembly sequence? How is a bucket of paint modeled, such that the robot could correctly stir the paint in order to mix it? How is the language to include these capabilities, yet be simple enough to be processed by a small computer? Robotics will need to develop as a strong independent area in order to answer these questions, to define needed developments in other areas, and to incorporate new developments in an intelligent and consistent manner.

References