Robotics

by J. U. Korein
J. Ish-Shalom

This paper is a survey that is intended to give the reader an introduction to some issues and problems in the field of robotics today. The first section discusses industrial applications of robotics and the requirements they engender. A substantial section is included on robot programming, including programming languages, motion programming, and techniques. This is followed by a section on trajectory planning. Issues in both robot-level trajectory planning and task-level trajectory planning are discussed. The section on control is divided into three parts: controller objectives, the system model, and controller types. Very brief discussions of actuators, sensing, and end effectors are also included.

The term *robot* was coined by Karel Capek in his 1923 play R.U.R., depicting class struggle in a society with automated workers.[1] *Robot* is the Czech word for worker. The word was picked up by science fiction authors in the 1930s and 1940s;[2–5] Isaac Asimov first used the term *robotics*. These authors were inspirational to scientists and engineers such as Joseph Engelberger, who participated in the development of early industrial robots.[6]

In the early 1950s, R. Goertz developed teleoperator manipulators for use in handling radioactive materials.[7] George Devol, who worked with Engelberger, holds the patent on the first industrial robot (1961).[8] The first *computer-controlled* robot was developed by Ernst at M.I.T., also in 1961.[9]

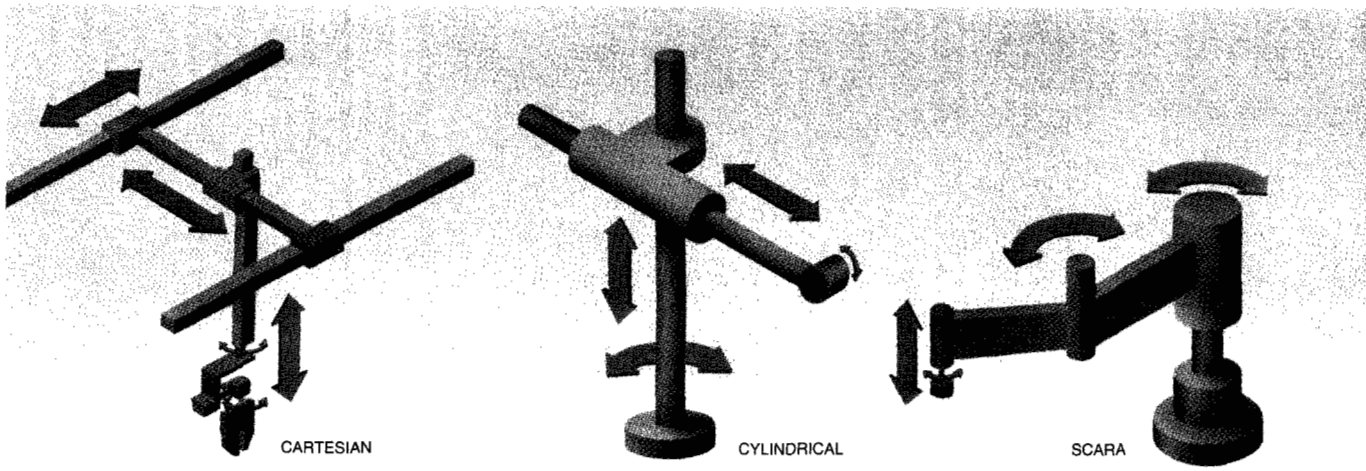
There is no universally accepted definition of the term *robot*. Typical definitions encompass notions of mobility, programmability, and the use of sensory feedback in determining subsequent behavior. Rather than pursue explicit definitions, we will proceed to discuss the general nature of the systems which are typically called robots today.

Industrial robots used for manipulation of goods typically consist of one or two arms and a controller. The term *controller* is used in at least two different ways in this paper, so we will make the distinction here. In this context, we mean the computer system used to control the robot, often called a *robot workstation controller*. The term *controller* is used to mean the embodiment of the actual *robot servo control system*, especially in the section on control. Robot arms come in a variety of different types, a few of which are shown in Figure 1. The "hand" of the robot is called its *end effector*. The types of joints (revolute, sliding), their arrangement, and the geometry of the links that connect them comprise the *kinematic structure* of the robot.

The controller may be programmed to operate the robot in a number of ways, thus distinguishing it from hard automation. The controller is also responsible for the monitoring of auxiliary sensors that detect the presence, distance, velocity, shape, weight, or other properties of objects. Robots may be equipped with vision systems, depending on the application for which they are used. Most often, industrial robots are stationary, and work is transported to them by conveyers or robot carts, which are often called *autonomous guided vehicles* (AGV).

Autonomous guided vehicles are becoming increasingly widely used in industry for materials transport. Most frequently, these vehicles use a sensor to follow

Figure 1 Types of robot arms



a wire in the factory floor.[10] Some systems employ an arm mounted on an AGV.[11]

Motivation

Forces driving the field. The largest single force driving robotics is the need to increase productivity and reduce costs in manufacturing. The circumstances under which robots can help achieve these goals depend on a number of factors, and the nature of a manufacturing task may or may not lend itself to the current capabilities of robots.

If the product volume is small, manual labor may be more cost-effective. If the volume is very large and homogeneous, greater throughput can typically be obtained by developing hard automation for the job. Robots are strongest in the middle ground.

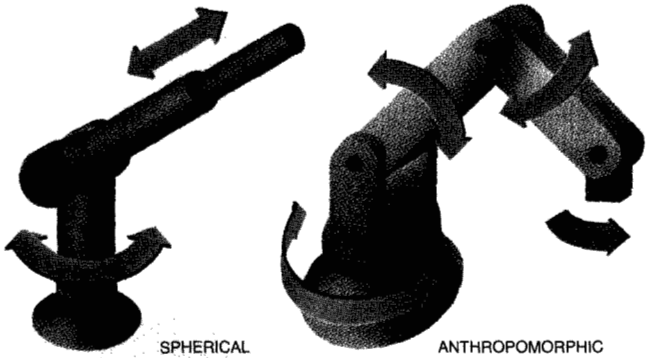
Robot programmability provides major advantages over hard automation. If there are to be many models or options on a product, programmability allows the variations to be handled easily. If product models change frequently, as in the automotive industry, it is generally far less costly to reprogram a robot than to rework hard automation. A robot workstation may be programmed to perform several tasks in succession rather than just a single step on a line. This makes it easy to accommodate fluctuations in product volume by adding or removing workstations. Also, because robots may be reprogrammed to do different tasks, it is often possible to amortize their cost over several products.

Robots can also perform many applications that are poorly suited to human abilities. These include manipulation of small and large objects like electronic parts and turbine blades, respectively. Another of these applications is work in unusual environments like clean rooms, furnaces, high-radiation areas, and space.[6,12]

Japan has led the world in the use of robots in manufacturing. The two sectors making heaviest use of robots are the automotive and electronics industries.[13] Growth of industrial robotics in the United States has been steady in recent years, but has not been as rapid as popular magazine articles of the early 1980s led the public to expect.[14,15]

Interest in legged locomotion has been stimulated by applications in traversing rough terrain and in unmanned exploration of unknown environments.[16,17]

Aside from economic motivations, there are many unanswered scientific questions about how biological organisms produce the remarkable sensorimotor behavior that we observe. Finally, the notion of simulating biological organisms has a certain instinctive reproductive appeal and offers the possibility of satisfying our curiosity as to how we have come to be as we are.



Applications. In this section, we consider several representative applications of robots, in order to understand the requirements that they engender.

Spot welding involves applying a welding tool to some object, such as a car body, at specified discrete locations. This requires the robot to move its hand (end effector) to a sequence of positions with sufficient accuracy to perform the task properly. It is desirable to move at high speed to reduce cycle time, while avoiding collisions and excessive wear or damage to the robot.

Pick and place is the name commonly given to the operation of picking up a part and placing it appropriately for subsequent operations. Pick-and-place operations have some requirements in addition to those for spot welding. The part must not be dropped. It must be held securely enough to prevent it from slipping in the gripper but gently enough to avoid damage. In addition, care must be taken to avoid disturbing the part during approach and departure.

Spot-welding and pick-and-place operations are characterized by their *point-to-point* nature; what happens at the beginning and the end of the motion is critical, but there is some latitude in choosing the intermediate trajectory.

Spray painting requires covering a surface with an even coat of paint. This is typically done by prespecifying the trajectory along which the arm will move. The trajectory specifies both position and orientation of the nozzle as a function of time.

Seam welding requires that a welding torch continuously follow a seam on a surface. Unlike spray painting, seam welding typically requires real-time correction of the path to accommodate small deviations of the actual seam from the expected path.

Spray painting and seam welding are both *continuous-path* applications; position and orientation as a function of time are important throughout the motion.

Electronic testing by robots is being used increasingly. One application is that of testing the continuity between pins, which involves primarily point-to-point operations. Another application is the detection of flaws in printed circuits by probing along metal traces on circuit boards.

Metrology is now often performed using automated coordinate measuring machines, which are essentially very slow and accurate robots. They are used to measure dimensions of mechanical parts, usually by a sequence of point-to-point motions.

Assembly is an application of increasing importance. Robotic assembly may be done in different ways. One typical method is to equip a simple robot with a special end effector for a particular task, such as inserting a component. The robot is programmed to perform a single operation as a single step in an assembly line. Each robot is fed parts of a single type from a part feeder, which presents them in the correct orientation. In this approach, the robot is used in the same way as hard automation is traditionally used. Feeder mechanisms, which are often quite ingenious, are discussed in Reference 18.

An alternative method is to feed all parts directly into a robot workstation in which the entire assembly is to be completed. Part feeders and magazines may be arranged about the workstation, as may a variety of tools and fixtures required for the assembly. Another option is that the workstation is presented with a "kit" of preoriented parts containing all components required for the assembly. To have individual robot workstations do independent assembly of complete products is extremely advantageous for flexible production capacity.

Nevins and Whitney[12] analyzed a number of product assemblies: a refrigerator compressor, an electric jig-

saw, an induction motor, a toaster oven, a bicycle brake, and an automobile alternator. They determined that these assemblies could be performed using a relatively small set of operations. These include simple peg-in-hole insertion, push-and-twist insertion, simultaneous multiple peg-in-hole insertion, screw insertion, force-fit insertion, removal of locating pins, flipping parts over, providing and removing temporary support, crimping sheet metal, and welding or soldering.

Performing these operations by precise positioning requires tight tolerances on part dimensions and part positions as well as accurate robot positioning. Requirements on these tolerances are substantially reduced if the robot end effector can comply with forces it encounters during the assembly process. For example, the tolerance required for inserting a peg into a chamfered hole is significantly less when one is guided by the forces one encounters, rather than proceeding stubbornly in some assumed direction. This is easily demonstrated by attempting to insert a key into a keyhole using the two different approaches.

Machining of mechanical parts is a growing application of robotics technology. Operations like grinding, deburring, and sanding parts require the ability to follow surfaces and to maintain the forces required to perform the specified operation.[19]

An unusual but fascinating application is sheep-shearing. Recently, Trevelyan, Kovesi, and Ong[20] constructed a robot system that would perform sheep-shearing on live sheep. The robot was sufficiently adaptable to cut the wool without harming the sheep. Sheep appreciate compliance.

Requirements. A number of stringent requirements are imposed upon robots in order for them to be competitive in the world of manufacturing.

- *Reliability* and *durability* are very important. An industrial robot must work every day, often all day, to pay for itself.
- Robots are not usually as fast as hard automation or human workers doing the same job. Currently, the *speed* of robots is constrained by computational as well as mechanical factors.
- *Accuracy* of robots is important for such applications as precise electronic test and for assembly tasks.
- The ability to comply with the environment is important in assembly and machining applications. Precisely machined parts are usually expensive. *Compliant motion* is needed to perform adequately with affordable parts.
- It is highly desirable that robots be sufficiently *configurable* to allow new sensors to be incorporated. Sensory input should be available to provide continuous servo control and to produce discrete transitions in system behavior.
- *Ease of programming* is important, so that robotic applications may be developed quickly.
- *Versatility* is needed to avoid the cost of special-purpose fixturing required for new robot workstations.
- *Cleanliness* is of increasing importance in many electronic applications.

Robot programming

Robot programming is the means by which a robot is instructed to perform its task. In this section, we examine some of the tools and techniques that have been used and proposed for robot programming.[21]

Guiding. *Guiding* is the process of moving a robot through a sequence of motions to "show it" what it must do.[22,23] One guidance method is to physically drag around the end effector of the robot, while it records joint positions at frequent intervals along the trajectory. The robot then plays back the motion just as it was recorded. An alternative is a master-slave or teleoperator configuration. Early systems of this type were first used to manipulate radioactive materials remotely.[7] Teleoperator techniques are now employed to guide the Space Shuttle manipulator.

Guiding may also be applied using a *teach pendant*, which is a box with keys that are used to command the robot. Several modes of operation are often available on the teach pendant. In *joint mode*, a pair of buttons is used to move each joint back and forth. In addition to joint mode, one or more cartesian modes may be provided. In cartesian modes, buttons are associated with cartesian axes in some three-dimensional coordinate system. Two examples are *world mode*, in which the coordinate system is aligned with the base or frame of the robot, and *hand mode*, in which the coordinate system is always aligned with the gripper, as shown in Figure 2. In the figure, the X, Y, and Z axes of the hand coordinate system are labeled X', Y', and Z', respectively.

For point-to-point applications, guiding systems usually allow the user to specify a few key positions to be recorded. The system interpolates between adja-

cent pairs of these positions, providing a path that does not play back every fumble and overshoot of the teacher.

Guiding is limited as a robot-programming technique, because it does not provide conditionality or iteration. Some systems provide so-called *extended guiding* capabilities that include teaching in a coordinate system that may be moved at run-time and conditional branching between motion sequences.[21]

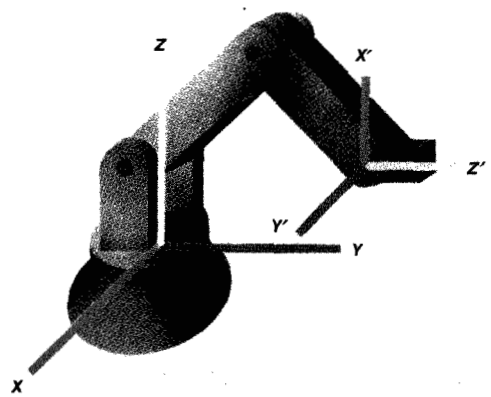
Programming languages. It is no surprise that the well-established technology of programming languages should be adopted for programming robots, and many robot-programming languages have been developed. The first was a limited language, called MHI, designed by Ernst at MIT in 1961.[9] Two languages developed at Stanford University, WAVE[24] and AL,[25,26] were particularly influential in the field. Conventional languages like C,[27] Lisp,[28] Pascal,[29] and Basic[30] have been extended with subroutine libraries for robot control. Robot languages in use in industry today include AML (IBM),[31] HELP (GE),[32] Karel (GMF),[33] LM (Scemi, Inc.),[34] MCL (McDonnell Douglas),[35] RAIL (Automatix),[36] and VAL-II (Unimation, Adept),[37] and others. Bonner and Shin[38] have published a survey of robot-programming languages.

The nature of robot programs. First and foremost, robot programs are computer programs.[31] Consequently, a large part of the body of knowledge that has been acquired about programming in the last couple of decades is applicable to robots. Robot programs deal with a richer variety of I/O devices than conventional programs, which distinguishes them in a variety of ways.

Robot programs must command robots to move; thus, the way in which motion is specified is important. Also, the programs use information obtained from sensors. One way of using sensory information is to monitor sensors until a prescribed condition occurs and then perform or terminate a specified action in response. Another use of sensory information is to use feedback from sensors to modify the robot's behavior continuously.

When objects are manipulated in the physical world, many peculiar things can happen. One occurrence of a part may differ slightly from another in ways which cause a program to fail. The tolerances on the dimensions of the object, the position of the object, and the position of the robot all vary from operation to operation. This may lead to jamming or wedging

Figure 2 World and hand coordinate frames



of pegs or screws, dropping or knocking down parts, and other difficulties. Obtaining bounds on errors has been addressed in the research literature. The problem of bounds has been approached numerically by Taylor[39] and symbolically by Brooks.[40] If an event such as a collision is not foreseen by the programmer, it is usually desirable to stop moving things around and request operator intervention. In order to make a program robust, the programmer must try to anticipate and test erroneous conditions. Consequently, robust robot programs are rather heavily weighted with error-handling and recovery apparatus. Note that the programmer's ability to make the program robust depends on the availability of sensors to detect problems.

No robot is an island. Industrial robots function in factory environments, which contain other robots, automation equipment, and computers. Effective communication among these elements is critical to Computer-Integrated Manufacturing (CIM). Communication in the factory is often structured hierarchically.[41] Some of the levels often considered are device control, workstation control, and area control. *Device control* typically includes control of a single robot or other device. *Workstation control* coordinates activities of multiple robots and other devices within a workstation. Concurrent program language constructs may be expected to play a major role at this level. *Area control* includes coordination of workstations, recording of manufacturing data, scheduling, and routing of work.

Motion commands. Robot-programming languages must contain specifications of motion for the robot. Some of the commands used to specify motion are discussed here. The *joint-level move* is the most fundamental motion command. In the simplest case, the command might be written

move (*joint, goal*)

The joint is either a sliding or revolute joint that is to be brought to some linear or angular position. Even in this simplest case, much is left unsaid. For a revolute joint without joint limits, the path has not been fully specified, because the goal could be reached in either a clockwise or a counterclockwise fashion. How fast should the motion be? How accurately must the goal be achieved? Is a little overshoot allowable? There are always trade-offs between these considerations. In the interest of simplicity and flexibility, it is desirable to have a number of optional parameters to allow these factors to be controlled, and it is desirable to use specified defaults when they are not. With large numbers of optional parameters, keyword parameters are advantageous.

For this simple move and most of the others discussed in this section, goals are given in absolute terms. It is worth noting that it is often more convenient to specify all of these in terms relative to the current position.

Joint moves may involve more than one joint, for example:

move ($\langle j_1, j_2, j_3 \rangle$, $\langle goal_1, goal_2, goal_3 \rangle$);

In this example, we have used AML notation[31] to specify an aggregate (list) of three joints, and another of three goals. It is convenient to think of such a goal as a point in *joint space.*

Specification of a goal for a group of joints means that they are to be executed in parallel. If the trajectory for each of these joints is planned independently, the motions end at different times. *Coordinated motion* (with all joints halting simultaneously) requires that the trajectory planner scale all trajectories so that their elapsed time is the same as for that of the slowest joint.

In most cases, it is far more convenient to specify *cartesian motions* than joint-level motions. As in the case of guiding, cartesian motion specifications are made with respect to some cartesian coordinate system. The position and orientation of a rigid object in space may be described with six numbers corresponding to its six degrees of freedom. Suppose, for example, the object were located at some reference position, with a reference point at the origin, as shown in Figure 3. Thus, (A) rotation ϕ produces (B), then rotation θ produces (C), and similarly rotation ψ produces (D). The object is brought into an arbitrary orientation by three rotations (ϕ, θ, ψ) about axes rigidly affixed to the object; these are often called Euler angles. An ordered triple (p_x, p_y, p_z) specifies an arbitrary translation. So the position and orientation of an object with respect to a reference frame may be specified by giving the values of $p_x, p_y, p_z, \phi, \theta, \psi$, which bring a reference object to that position and orientation.

The position and orientation of an object with respect to a reference frame may also be represented as a 4-by-4 homogeneous transformation matrix.[42] The form of the matrix is as follows:

$$\begin{matrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{matrix}$$

The reference coordinate system (X, Y, Z) and object coordinate system (X, Y, Z) are shown in Figure 4. If we ignore the last row, each column is a three-element vector expressed in terms of the reference coordinate system. In Figure 4, **n**, **o**, and **a** are unit vectors in the directions of the axes X, Y, and Z. **p** is a translation vector, giving the location of the origin of the object coordinate system. The conventions for the names of these vectors are taken from Paul.[42] Because the object is often a gripper, **o** indicates the axis along which the gripper *opens*, **a** indicates the *approach* direction in which the gripper points, and **n** indicates their common *normal* in the direction $\mathbf{o} \times \mathbf{a}$.

A similar (transposed) representation is used in computer graphics.[43] The homogeneous matrix representation is particularly convenient for manipulation (although not computationally efficient[44]). In the context of robotics, the homogeneous matrix representation has come to be called a *frame.* Subsequently, when we speak of the position of an object, we will mean both its position and orientation, as described by a frame.

In order to move the end effector of an arm to a specified position in terms of the world coordinate system, we may write a command of the following form:

move-hand(*dest*);

Figure 3 Reorienting an object with three successive rotations

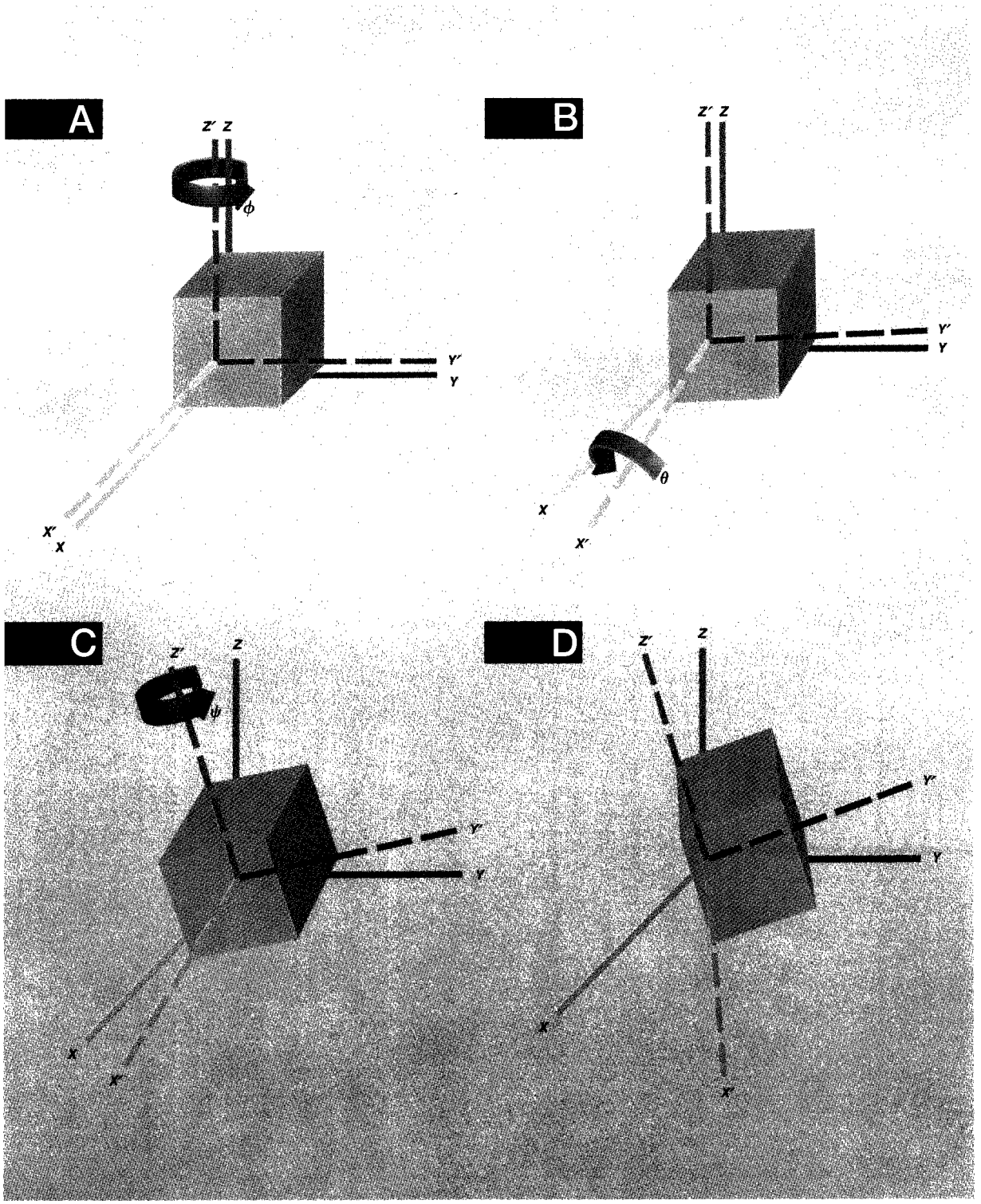
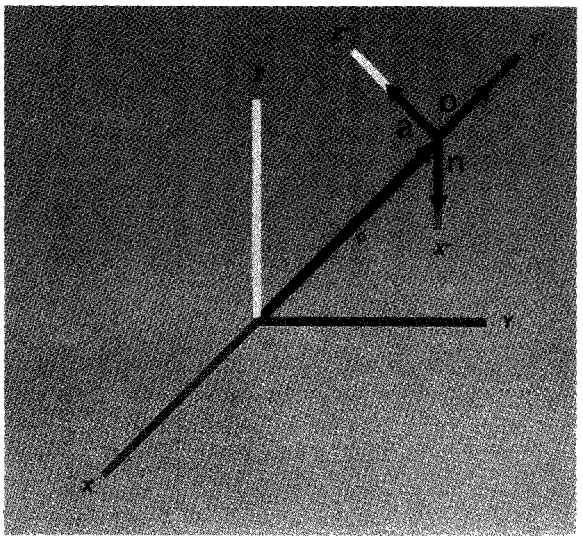


Figure 4 Vectors in homogeneous coordinate matrix



where *dest* is a variable that contains a specification of position and orientation as a frame. The specified goal may not be within the workspace of the robot. The robot workspace is determined by the geometry of the robot links, the placement and types of joints, and their limits of mobility. When orientation is considered, the dimensionality of the space within which the robot hand works is six. When the number of joints is less than six, most of the workspace is inaccessible, just as a planar linkage cannot access the space outside the plane. This situation is perfectly acceptable for wide classes of robots that perform tasks like pick and place on a horizontal surface or pancake assembly (stacking). These tasks may never require that the robot point its gripper in any direction besides down, for example. Using a frame to specify a goal is, in a sense, too general for robots of this nature. However, when a variety of robots with different kinematic structures must work in concert, the use of frames provides them with a common language.

Even when there are six joints, joint limits may make certain configurations for the end effector unreachable. Consider, for example, a cartesian robot whose "forearm" is constrained to point downward. (See the cartesian robot in Figure 1.) It is not possible to point the end effector upward, because it would have to occupy the same space as the forearm.

It is often the case that a particular position for the end effector may be achieved by several different configurations of the robot arm. For example, if the robot's elbow can flex either way from the outstretched position, most end effector positions may be achieved in either an elbow-up or elbow-down configuration. If the robot has more than six joints (often called *redundant*), there are generally an infinite number of configurations by which to achieve a desired end effector position. In many cases, the programmer is not concerned with the choice of configuration, and it is satisfactory to use some default. Note, however, that if the default is chosen to minimize time, it usually depends on the previous position. This means that a subroutine performing that command behaves differently in different contexts. The choice may be important if obstacles are present. Also, the accuracy of the move may depend on which trajectory is taken. For these reasons, an optional parameter that depends on the kinematic structure of the particular robot is usually a necessary evil.

It is extremely important in robot programming to be able to specify the position of an object with respect to one reference frame and then obtain that position with respect to another reference frame. For example, when a mechanical part is defined, it is convenient to describe its features (such as grasp positions and connectors) in terms of a coordinate system rigidly affixed to the part. When it comes time for the robot to pick up the part, the robot must know the position of the feature with respect to the world coordinate system. Homogeneous transformations make this type of conversion very simple. R. Paul was an early proponent of this technique in robotics,[42,45] which is now widely used.

Suppose that HANDLE is the name of a frame that describes the position of the handle of a mechanical part with respect to a coordinate system rigidly affixed to that part. Suppose also that *WIDGET* is the name of a frame which describes the position of the part in terms of the world coordinate system. Then the position of the handle in terms of the world coordinate system is just a new frame, as follows:

$$WIDGET \cdot HANDLE,$$

where the dot ($\cdot$) is the matrix multiplication operation. Suppose that a number of widget positions are given with respect to a tray, whose position is in turn given with respect to the world coordinate frame. The handle of the ith widget could be specified by

$$TRAY \cdot WIDGET_i \cdot HANDLE,$$

where *WIDGET* is (in this case) an array of frames.

Let us now define a frame *variable* called *HAND*, which gives the position of the robot hand in the world coordinate system. Whenever we move the robot hand to some position *GOAL* (also in world coordinates), we are essentially asking it to change the value of *HAND* so as to satisfy the following equation:

$$HAND = GOAL.$$

Suppose now that the situation is more complex. The hand is holding a screwdriver whose position is defined with respect to the hand, and we want to position it at some offset over a screwhole on the widget. Because all definitions are with respect to local coordinate frames, the equation we wish to solve is the following:

$$HAND \cdot SCREWDRIVER = WIDGET \cdot SCREWHOLE \cdot OFFSET.$$

We may rewrite this goal

$$HAND = WIDGET \cdot SCREWHOLE \cdot OFFSET \cdot SCREWDRIVER^{-1},$$

and attain it by issuing the command

move-hand($WIDGET \cdot SCREWHOLE \cdot OFFSET \cdot SCREWDRIVER^{-1}$).

This approach is satisfactory for stationary goals. Consider, however, the situation where the widget is on a moving conveyer. We might define a function *WIDGET*( ), which returns a frame describing the position of the widget at the time it is called. In this case, either the time between the evaluation of *WIDGET*( ) and the execution of *move-hand*( ) must be very short or else some extrapolation must be done to account for the intervening time. In either case, there is some advantage in having a move command, to which the function may be passed directly, instead of its being evaluated beforehand, as above. The robotics library RCCL for the C language, developed by Hayward and Paul at Purdue University, provides this capability.[27]

A concept that has proved useful in the context of robot programming is that of *affixment*. Suppose that the widget is at some position specified by the variable *WIDGET*. Now we direct the robot arm to pick up the widget and move it to a new position. The position specification *WIDGET* is now out of date; that is no longer where the widget is. The idea of affixment is to allow one to specify durations over which one frame is rigidly affixed to another. If one changes, the other will be updated accordingly, without the need for explicit updating by the programmer. Affixment is supported by the languages AL[26] and LM.[34] Affixment may be thought of as an equality constraint specified between two coordinate frames. One might imagine further reductions in programming effort being introduced by a system that could maintain a wider variety of constraints between coordinate frames. Methods of maintaining constraints within the robot control system are discussed by Geschke,[46] Mason,[47] and Ish-Shalom.[48]

Trajectory specification. Earlier, in the section on guiding, we briefly discussed *straight-line motion.* The "straight line" in straight-line motion may apply to the tip of some tool that the arm is carrying, or it may apply to the wrist. Tool-tip motion may be used to follow a path, as in the application of adhesive, or to ensure the direction of an approach, as in an insertion operation. The reason that straight-line motions are not used exclusively is that other trajectories are often faster, both for mechanical and computational reasons.[44] This is discussed later in this paper in the section on trajectory planning.

Via-point moves are the usual means by which obstacle avoidance is achieved in practice.[45] Via-points are simply intermediate positions through or near which the robot is constrained to pass on the way to its goal. Suppose that we wish the robot arm to pull away from a handle on object *A* and then approach one on object *B*. Assuming that these positions are known to sufficient precision, we might write the following order:

move-via($A \cdot HANDLE \cdot OFFSET$, $B \cdot HANDLE \cdot OFFSET$, $B \cdot HANDLE$).

That is, we specify two intermediate frames, as well as the final destination. The idea is to allow the system to compute a trajectory whose segments are fairly close to straight lines, while rounding the intermediate corners to avoid stopping.

Compliant motion is motion that conforms with forces which are encountered. As was mentioned in the section on assembly applications, compliant motion is important, because there is always some error in the specification and realization of a trajectory. Some examples are insertion, part mating, following a surface, and turning a crank. There are as yet no real conventions for specifying compliance in a program. One method that has been used in conjunction with *impedance control* (discussed later) is the *explicit feedback* approach,[28,49,50] in which a stiffness parameter is specified with a positioning command.

In the case of a single joint, one might write the following command:

move-joint-compliant(p, k).

This will cause the joint to move to p as follows, always exerting a force f:

$f = -ke$,

where e (error) is the current deviation from the desired position p. The spring constant k is transformed to a feedback signal to be used by the controller. There are limits on achievable values of k arising from the stability limits of the controller and from the maximum force the actuator can produce.

A robot program is not a mathematical function; in computer science terms, its results are just a big side effect.

More complex types of compliance may be achieved by defining a coordinate system called a *compliance frame*. In this system, all forces directed through the origin (called the *center of compliance*) generate pure translations. Similarly, torques about the axes generate pure rotations.[51] The explicit feedback approach may be used to specify compliance in an arbitrary compliance frame by generalizing the equation above to the following:

$\mathbf{f} = -K\mathbf{e}$,

where K is a six-by-six matrix of values and $\mathbf{f}$ and $\mathbf{e}$ are vectors.[21]

Another approach, called *hybrid control*, requires the user to explicitly specify the compliance coordinate frame. Position and force goals are then specified along (and about) the axes of that frame. Typically, one uses force goals in directions where resistance is met (as into a surface), and positional goals in directions where there is none.[51]

A different approach is used to describe the behavior of the system in terms of relationships that are to be maintained with respect to force, velocity, position, and other measured and controlled quantities. For example, $\mathbf{f}_d \cdot \mathbf{v}_m = 0$ indicates that the desired force is in a direction orthogonal to a measured velocity.[46,48]

Guarded motions are critically important concepts in robot programming.[52] A guarded move is one that may be terminated on the basis of sensory data acquired in real time. Some examples of guarded moves are the following:

- Bring the hand down until it hits the table top.
- Close the gripper until substantial resistance is detected.
- Move until a photosensor detects that something is between the fingers of the gripper.
- Turn a screw until a substantial resisting torque is detected.

For example, AML[31] includes guarded moves that may be terminated when thresholds are reached by built-in force sensors and by changes in boolean inputs.

A simple but useful generalization is that of associating a set of arbitrary termination conditions with an arbitrary action.[26,53-55]

Concurrency issues. The issue of concurrent execution of program statements has received widespread attention in computer science. The primary motivation has been to make effective use of multiple processors to speed up computation. In robotics, the problem is compounded by the need to control mechanical processes concurrently with one another and with computational processes. Many of the considerations are very similar, but there are important differences, too. Physical processes cannot, in general, be suspended and resumed as computations can, because of physical laws (like Newton's laws) and the presence of external forces (like gravity). Concurrency is required for speed, because it is highly undesirable to serialize a set of mechanical tasks. However, there are tasks that cannot be done serially at all, without great imagination. To experience this, the reader may try the experiment of going about his normal routine for an hour while keeping one hand in his pocket. A variety of mechanisms have been used to describe concurrent activities in programs.[56-59] Concurrent programming constructs have been provided in the programming languages Concurrent Pascal[58] and ADA[60] and in the robot programming languages AL,[26] TEACH,[61,62] MCL,[35] and OWL.[53]

Programming methodology. The testing and debugging of robot programs is characterized by a much greater degree of trial and error than many other kinds of programming. A robot program is not a mathematical function; in computer science terms, its results are just a big side effect. Rather than measuring every clearance in a workcell, it is usually simpler to start with a guess and refine it while watching the robot go through its paces. Another issue is speed. It is common to debug robot programs at low speed, and then crank them up as fast as is possible without producing intolerable losses in accuracy. There are many such guesses in a complex program, usually embodied as defined constants. The need for a fast program revision cycle is one reason why interpreters are widely used for robot programming. Relative ease of implementation is another.

Several types of calibrations are required in robotics. Many robots determine joint position by counting pulses as the joint travels, in which case the robot must be calibrated when it is powered up. Each joint must travel to its limits, usually tripping a switch, to establish its absolute position.

Now suppose there is some workpiece that is to be manipulated. One way to establish its position is by determining the position of a set of reference objects with a known relationship to the workpiece. The problem is then reduced to determining the location of the reference objects to establish an appropriate transformation. A technique that has been used effectively is to locate posts by passing an open gripper over them, breaking a beam of light. By repeating this several times in different directions, a post may be located; several posts in a known configuration establish a coordinate system.[63]

A technique that has proved extremely effective in robotic applications is called *programmed recalibration*. Suppose that a robot must repeatedly perform a sequence of insertions at specified positions. After some period of time, mechanical drift can build up sufficiently to cause the robot to miss. Suppose, however, on each insertion, the robot monitors the variation of its position from the nominal position, and adjusts its coordinate frame. This allows the robot to function indefinitely without accumulating error. This technique is extremely important for reliability in typical repetitive applications.

When a vision system is used with the robot, calibration is necessary to establish a common coordinate system between the camera and the robot.[64,65]

Higher-level programming. *Off-line programming* is a term that is usually applied to a collection of techniques for robot programming without actually using a robot. A Computer-Aided Design (CAD) system is typically used to model the robot workstation, parts, and auxiliary equipment. Then the simulated robot is programmed and its task executed in the simulated environment.[66] Collisions between objects may be checked by using a *collision-detection algorithm*. The utility of off-line programming for debugging robot programs is limited by the inability of most commercial solid modeling programs to include error tolerance information in geometrical models.[67,68] If a model does not include uncertainties in part position, part dimensions, and robot position, the simulation will succeed in situations where a real application would fail. Another difficulty with simulation is that force sensing must be modeled by collision detection, which is computationally expensive. This makes it inconvenient to model guarded moves and all but impossible to model force-guided compliant motions such as surface following.[69]

A Computer-Aided Design (CAD) system is typically used to model the robot workstation, parts, and auxiliary equipment.

A *task-level program* is a high-level specification of a task, without explicit mention of the robot or how the details of its job are to be performed. For an assembly task, the highest-level description might simply be a description of the relationships among the parts in the final assembly. A lower-level task specification might include a specification of intermediate states of the assembly. A still lower-level task specification is a sequence of high-level operations to be performed to achieve the intermediate states. The problem of going from the highest of these levels to the lowest has been one of the problems considered in planning research in the field of artificial intelligence.[70,71]

Figure 5 Example AUTOPASS program for support bracket assembly

```
1. OPERATE nutfeeder WITH car-ret-tab-nut AT fixture.nest
2. PLACE bracket IN fixture SUCH THAT bracket.bottom
       CONTACTS car-ret-tab-nut.top
       AND bracket.hole IS ALIGNED WITH fixture.nest
3. PLACE interlock ON bracket SUCH THAT
       interlock.hole IS ALIGNED WITH bracket.hole
       AND interlock.base CONTACTS bracket.top
4. DRIVE IN car-ret-intlk-stud INTO car-ret-tab-nut
       AT interlock.hole
       SUCH THAT TORQUE IS EQ 12.0 IN-LBS USING air-driver
       ATTACHING bracket AND interlock
5. NAME bracket interlock car-ret-intlk-stud car-ret-tab-nut
       ASSEMBLY support-bracket
```

All of the specifications just mentioned are called *task-level* specifications as distinguished from *robot-level* specifications, because they do not describe the details of how the robot is to perform an operation. The program does not specify paths that avoid obstacles or specify specific grasp positions. The translation of such a specification into a robust, working robot program is the central research problem in robot programming. No complete, working, task-level programming system has ever been implemented. Early work in task-level program specification was done by Feldman,[72] Paul,[45] Taylor at Stanford,[39] and Lozano-Perez at MIT.[73] The specification for a task-level language called AUTOPASS by Lieberman and Wesley at IBM[74] served as a focus for much of the subsequent research in the area. An example of a task specification in AUTOPASS is shown in Figure 5. Some of the operations required to execute task-level programs, such as obstacle avoidance, grasp planning, and fine-motion planning, are discussed in the section on trajectory planning. An excellent survey of the area of task-level programming by Lozano-Perez appears in the book *Robot Motion: Planning and Control.*[75]

Trajectory planning

In the previous section on robot programming, we discussed the types of commands that are used to program robots. We now discuss some of the issues in trajectory planning that must be considered in the implementation of robot motion commands. This will be augmented by the discussion of control in the following section.

Trajectory planning for robot-level programming. Consider the implementation of a joint-level move command. We assume here that we have simple independent position controllers for each axis. At regular intervals of time, each controller reads a position value or *setpoint p* from a memory location and generates an actuator command to drive the axis towards *p*. The length of these time intervals may vary from 0.1 to 100 ms, depending on the controller and application requirements.

Simply setting *p* to the desired final joint position causes the controller to servo to that position. However, it is usually desirable to be able to specify the trajectory of the joint as a function of time. This is done by setting *p* to a series of intermediate positions along the trajectory function to allow for coordinated motion and continuous path motion. In specifying a trajectory, the physical limits of the system must be considered. It is common to model these limits as constant maximum values for acceleration and velocity.

The trajectory goes from the initial to the final position, with initial and final velocities zero, subject to limits on speed and acceleration. A trajectory used in many industrial robots is shown in Figure 6. The motion consists of a constant acceleration phase, followed by a constant velocity phase, followed by a constant deceleration phase. If the acceleration, velocity, and deceleration in the three respective phases are all set to the constant maximum values assumed in the model, this trajectory is time-optimal under those modeling assumptions. Intuitively, this strategy is comparable to flooring the accelerator, then coasting at the speed limit, and finally slamming on the brakes. Planning this trajectory requires determining the time for each phase of the motion and determining the parameters for each trajectory segment.

In the case of a coordinated motion for multiple joints, the trajectories for the joints must be planned together, so that they all take the same length of time. For trajectories of this type, one computationally convenient method is to keep the three phases of motion the same for all joints. The time for each phase is then determined by the slowest joint for that phase.[44] This method has the advantage that the interpolated trajectory follows a straight line in *joint space.* An alternative approach is to compute the total times separately for each joint and scale all joints to that total time, phasing them separately. This is optimal in terms of motion time (given the modeling assumptions) because the total time for the motion is the same as the total time for the slowest joint. However, the separate phasing of the

joints requires additional computation, and the trajectory no longer follows a straight line in joint space.

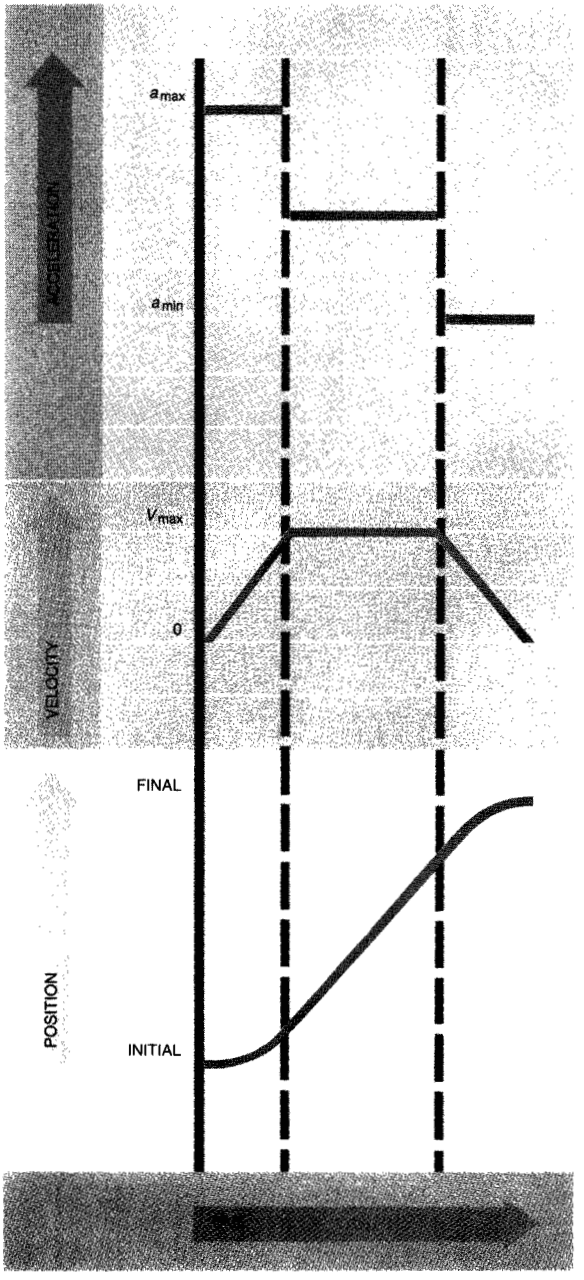
Kinematics. When a cartesian position goal is specified for the end effector of the robot, it must ultimately be converted to a control signal for each joint actuator. If the joint controllers require joint positions, it is necessary to produce these from the cartesian specification. This problem is called the *inverse kinematics problem* in the robotics literature. There is no general analytic solution to this problem for an arbitrary robot. This is in sharp contrast to the *forward kinematics problem*, in which a cartesian specification is obtained from joint angles. For open kinematic chains, which include most robots, forward kinematics may always be computed by multiplying coordinate frame matrices. The form of the inverse solution depends on the geometry of the robot joints and links. However, for a wide class of robots called *kinematically simple*, efficient analytic solutions may be obtained.[42] The criterion for kinematic simplicity is that three consecutive joint axes intersect at a point. In this sense, most industrial robots are kinematically simple.

Numerical solutions to the inverse kinematics problem are much more general. These methods typically use the Jacobian matrix **J**, which expresses the rate of change in the cartesian variables with respect to the joint variables. A single formulation, such as Newton-Raphson, may be applied to a wide variety of linkages. Analytic solutions have usually been favored for real-time computation because of their greater speed and because they produce all solutions, rather than converging to a single one.[76] Recently, new numerical techniques have been developed that exhibit substantially greater speed than conventional methods and produce multiple solutions.[77]

When the robot has the same number of joints as there are degrees of freedom in its environment, it is said to be *perfectly constrained.* There are a small finite number of solutions to the inverse kinematics problem, depending on how many revolute joints are present. Some of the solutions to the mathematical problem are not usable, because they lie outside the range of joint limits of the robot. The most obvious criterion for choosing among those that are left is some measure of closeness to the current position.

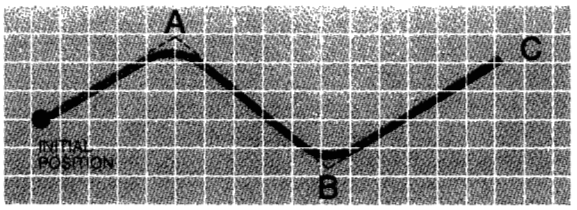
When the robot has redundant degrees of freedom, there are an infinite number of solutions to the inverse kinematic problem. Different methods are required to solve this problem in a way that uses the redundancy effectively.[78]

Figure 6 A commonly used trajectory showing acceleration, velocity, and position as functions of time



In a number of systems, the explicit solution of the inverse kinematics problem has been avoided by performing the control calculations to obtain desired

Figure 7 Via-point path



forces completely in cartesian space. The joint torques may then be obtained by multiplying the cartesian forces by the transpose of the Jacobian matrix.[79,80]

Inverse kinematic velocity and acceleration problems are not discussed here. For a discussion of these topics, see the introduction to Reference 69.

Cartesian trajectory planning. Consider the execution of a *move* command whose goal is a cartesian position that is specified as a frame. This goal may be converted to a joint-level goal by calling an inverse kinematics routine. Then the trajectory-planning techniques already discussed for joint-level planning may be used. This approach generally does not produce straight-line motion in cartesian space. There is one important exception. For the wrist point of a cartesian manipulator, joint space *is* cartesian space. Thus, a straight line for the wrist only requires linear dependence between coordinated joint trajectories. However, a kinematic conversion is still required for tool-tip, straight-line motion if the orientation of the end effector changes.

In order to achieve straight-line motion, trajectory planning must be done in cartesian space. The most common approach to achieving straight-line cartesian motion was pioneered by Paul in the mid-1970s.[81]

Choose a model that assumes constant maximum accelerations and velocities in cartesian space. Then the same trajectory function that we used in joint space can be used in cartesian space. By interpolating along this trajectory, we obtain a series of cartesian frames. Each of these frames may be converted to a joint-level setpoint vector by the application of an inverse kinematics subroutine. That is, motion along a straight line or any other trajectory in cartesian space may be achieved by repeated application of the inverse kinematics routine. Of course, if this is to be repeated for every setpoint, the kinematics routine must be extremely fast. Computation may be saved at the price of greater deviation from the trajectory by precomputing only some of the cartesian setpoints, performing inverse kinematics, and interpolating the results in joint space. Alternative approaches to this problem have been analyzed by Taylor.[44]

A variety of methods have been used to perform via-point motion, in which a number of intermediate frames are specified as well as the goal frame. If the path between these is piecewise linear, the end effector will have to stop at each position, because it cannot instantaneously change the direction of its velocity vector. One method is to blend together the line segments of the path by quadratic arcs that deviate from the via points by a specified distance.[81] (See Figure 7.) An alternative is to use cubic splines.[82]

Limitations of conventional approach. In the conventional trajectory planning schemes just described, we have assumed fixed upper bounds on acceleration and velocity in the planning space. These assumptions are often unrealistic.

A more realistic assumption is that the limit on the amount of force (or torque) a joint may generate is a given constant. For a single linear joint, we may write

$$ma = f_{\text{actuator}} + f_{\text{other}}.$$

That is, the acceleration depends on the mass of the payload, the force generated by the actuator, and other forces such as friction and gravity. This means that if there is a constant limit on actuator force, the limit on acceleration will vary with mass and other forces. Even in this simple case, the assumption of a fixed acceleration limit does not account for varying payload, gravity, or interactions with objects in the environment, as in compliant motion. The abilities to deal well with changing payload and to perform compliant motion are of critical importance in many robotics applications.

In fact, it is not even reasonable to assume a fixed torque limit for many actuators; the torque limit often depends on motor velocity.

In the case of a complete robot system including revolute joints, matters are made considerably worse by the fact that the forces acting on each joint depend on the positions, velocities, and accelerations of all the other joints.

These considerations mean that, even for joint-level trajectories, any assumptions about fixed acceleration limits must be based on the worst case. This results in motions that are usually slower than necessary, or else the actuators may be incapable of following the requested trajectory.

Additional problems are introduced by cartesian trajectory planning. Assuming fixed limits for velocity and acceleration in cartesian space compounds the problems that arise when these assumptions are made for joint space. Consider a vector function $\mathbf{x}(\mathbf{q})$ that gives cartesian positions as a function of joint positions. The cartesian velocity vector $\dot{\mathbf{x}}$ is related to the joint velocity vector $\dot{\mathbf{q}}$ by its derivative with respect to $\mathbf{q}$, the Jacobian matrix $\mathbf{J}$. The values of the elements of the Jacobian vary over the workspace of the robot. When one of the values approaches zero, it means that a very small change in a cartesian variable corresponds to a large change in a joint variable. This means that a very large angular velocity of a joint is required to achieve some seemingly reasonable cartesian velocity. In one such situation, shown in Figure 8A, the arm is initially pointed straight up. To bring the tip directly downward just slightly, it is necessary to make relatively large changes in two joint angles, as shown in Figure 8B. The relationship between cartesian and joint accelerations exhibits similar difficulties. The situation is similar to that of backing up a long truck; a small adjustment of the steering wheel may cause a large displacement of the tail end.

Dynamics. Taking dynamic limits into account in trajectory planning is an important area of research.[83–90] An important result by Hollerbach and Flash at MIT is the discovery of a time-scaling property in manipulator dynamics.[85] Suppose that a trajectory is planned without consideration of dynamic limitations of the manipulator. It may be determined, using the *inverse dynamics* computation, whether achieving each setpoint along that trajectory will require manipulator force limits to be exceeded.[91] Hollerbach develops a method that allows the speed of the manipulator along its path to be scaled to bring it within specified torque limits, without re-executing the inverse dynamics computation.

Recently, Bobrow, Dubowsky, and Gibson devised an algorithm for determining the time-optimal trajectory of a manipulator along a prespecified path, given dynamic constraints.[86] The problem is formulated in state space, with distance along the path on one axis and velocity along the other. (See Figure 9.) The dynamic constraints are represented as a forbidden region in state space. It is known (a result of Pontryagin's maximum principle) that the time-optimal trajectory is a sequence of motion segments consisting of maximum accelerations and maximum decelerations. Bobrow et al. have also developed a simple algorithm to compute the switch points between the acceleration and deceleration segments so as not to enter the forbidden region. The computation of this trajectory is too slow to be suitable for use at move execution time, but may be used for preplanning trajectories and as a basis of comparison of suboptimal trajectories.

Figure 8 The singularity problem: (A) arm straight up; (B) small downward motion requires large angular motion

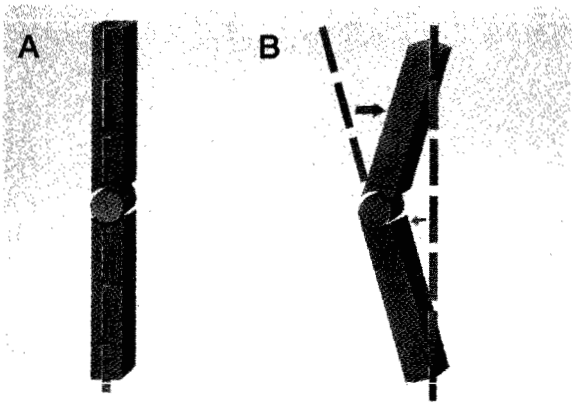


Figure 9 Optimal trajectory in state space

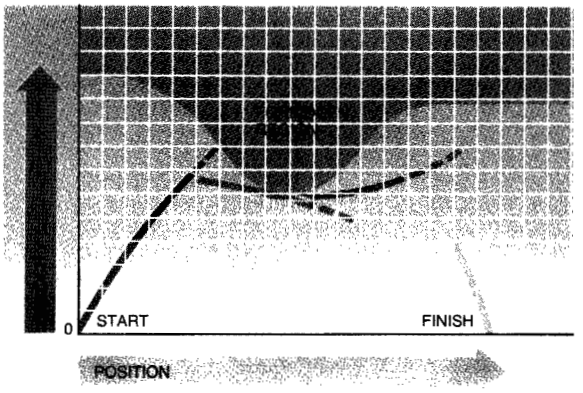
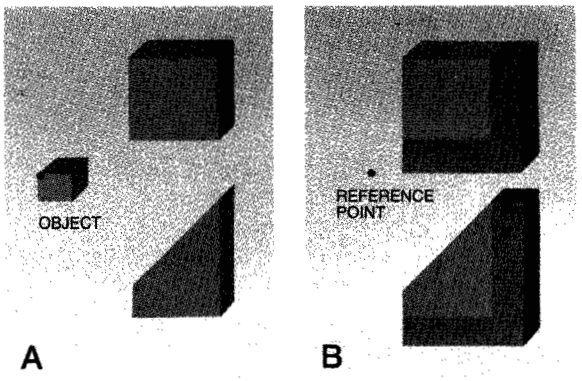


Figure 10 Obstacle avoidance and dual problem: (A) moving a polygonal object through a maze of obstacles without reorientation; and (B) moving a point through grown obstacles



These techniques for optimizing time along a known path have led to several algorithms for finding the time-optimal trajectory over any possible path. Sahar and Hollerbach[90] and Brown[89] have both used the approach of graph search through the elements of a tesselated space to find the best trajectory. These algorithms require a great deal of computation, even for simple examples. Rajan takes a different approach, representing the search space as a parameterized set of paths.[87] Rajan chooses cubic splines for his parameterization, on the assumption that the optimal path will be smooth. The Bobrow-Dubowski-Gibson algorithm is used to find the minimum time trajectory along that path. Initially, a single spline curve is used, and the path is varied by changing the free parameters of the spline. This allows the best path that can be constructed with a single spline to be found. This path is then subdivided at a knot point, and the best path using two splines throughout the knotpoint is found. The knotpoint is then perturbed to find one that produces a locally optimum path. This process may then be repeated, recursively subdividing the splines.

Trajectory planning for task-level programming. Task-level programming requires the generation of robot arm trajectories from geometric models of the robot, the objects to be manipulated, and their environment. Some of the important parts of this problem that have been studied are obstacle avoidance, grasp planning, and fine-motion planning.

Obstacle avoidance. Obstacle avoidance is a difficult problem and one that has received considerable attention.[88,92–99] One approach, developed by Lozano-Perez and Wesley, converts the problem of moving an object through a clutter of obstacles, as in Figure 10A, to an equivalent problem of moving a single reference point through an environment of *grown* obstacles, as in Figure 10B.[93,94] This important transformation greatly simplifies the problem and leads to a direct solution for distance-optimal paths for polygons in the plane. These techniques were applied to path planning for cartesian manipulators.

Lozano-Perez generalized and made heavy use of the important notion of *configuration space*, the space of all possible configurations of the entity under consideration. For example, the configuration space of a robot has one dimension for each of its joints; the configuration of an object constrained to move on a surface is three-dimensional (x, y, θ). Insight into many geometrical problems may be gained by posing them in configuration space.

Note that a robot configuration is a point in the configuration space of that robot, which often has dimensionality six. The high dimensionality of the configuration space of general robots has limited the utility of pure configuration space approaches for robot path planning.[100]

Brooks[98] has developed a special-purpose solution for an anthropomorphic manipulator, using four of its six degrees of freedom. Free space is described in two ways: as a freeway for the upper arm, and as a freeway for the forearm. The motions of the two components are analyzed separately, and then constraints are propagated between the two solutions.

Schwartz and Sharir have shown that the general obstacle-avoidance problem for a robot can be solved in polynomial time. Their algorithm is of theoretical interest only, because of the large size of the exponents.[96]

The obstacle-avoidance approaches that we have discussed use global knowledge of the geometry of the environment. A different, local approach to obstacle avoidance is discussed in the next section, on control.

Grasp planning. A number of constraints must be satisfied in grasp planning. No unexpected collisions must occur, there must be no slip while carrying the object, and the grasp position must be such that the object can be picked up and put down.

In 1972, R. Paul used a set of heuristics based on *orientation vectors* for the robot gripper. Paul required that the center of mass of the object lie between the jaws of the gripper, to prevent rotation, and that the jaws grip two parallel faces, a face and an edge, or two edges coplanar with the mass center. A set of orientation vectors is constructed for each object to be grasped. Paul uses a command called *move-instance* to move an object from position A to position B. For each orientation vector of the object, this requires the computation of the intersection of the ranges of approach angles for both positions A and B. This avoids penetration of the support plane and violation of robot joint limits.[45]

This and other early work on grasp planning make the assumption that the gripper may be positioned so that, at grasp time, simply closing the jaws produces a stable grasp on the workpiece. In the presence of uncertainty, it may be knocked over or the grasp may not be stable. More recent work by Mason[47] views a grasp as a sequence of pushes, and identifies sequences that may be guaranteed to produce a stable grasp in the presence of positional uncertainty.

Fine-motion planning. The job of a task-level planner is to produce a manipulator program. Of particular importance and difficulty is the generation of commands for fine motions involving contact, where success of the strategy depends on error bounds, on position, and on the use of compliance. An approach taken by both Taylor and Lozano-Perez in the 1970s was to maintain parameterized strategy skeletons, and to choose from among them on the basis of the values of goals and error bounds.[39,73]

Dufay and Latombe have proposed an approach in which a ground plan is initially formulated and then data are gathered about its performance during a training phase. Using the execution traces from the training phase, an inductive phase then modifies the ground plan to cope with problems encountered in training.[101]

An approach recently taken by Lozano-Perez, Mason, and Taylor[102] is the automatic synthesis of motion strategies from task geometry by backward chaining.[70] Given the desired final position of an object to be placed, its *preimage* is constructed in configuration space. The preimage is the region of all starting positions from which the object may be moved to its destination with a straight-line motion in configuration space, as shown in Figure 11. By constructing the preimage for the preimage, etc., a sequence of moves is generated. Errors in position and direction may be taken into account by shrinking the preimages. Compliant motions may also be incorporated by making the basic move operation a compliant one. When a point that is undergoing compliant motion encounters a surface, it slides along the surface if its angle of incidence is outside the range of values where friction will cause it to stick. This range is usually called the *friction cone* of the surface, which is illustrated in Figure 12.

Figure 11 Preimage in which the shaded area is the region from which a point may move to the bottom of the slot with a single straight-line motion

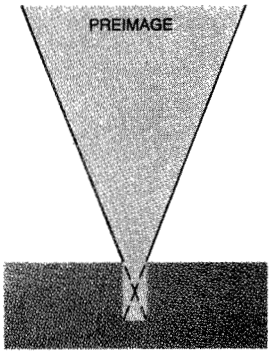
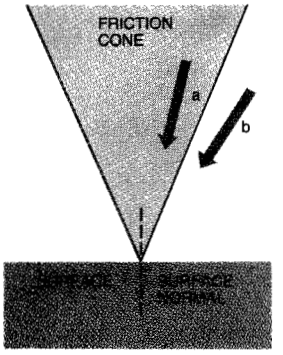


Figure 12 Friction cone showing a point undergoing compliant motion toward a surface: (a) the point sticks if its angle of incidence lies within the friction cone; and (b) otherwise the point slides



Robot control

In this section we describe the state of the art of robot control within the general framework shown in Figure 13. This framework is helpful, because a general solution to the robot control problem does not yet exist, and only several restricted subclasses of the robot control problem are solved. The general framework for robot control allows us to classify all robot controllers, to understand the context in which current controllers were designed, and to understand the performance one can expect to achieve from each design. This framework also indicates directions for future research and development to achieve higher-performance robot controllers. The following aspects of robot control are considered:

- Controller objectives
- System model
- Control methods

The most important feature that characterizes a robot controller is the *system model* that was considered in its design. The system model depends on the robot controller objectives and includes information on the robot mechanical design, actuators, sensors, and the "world" the robot is manipulating.

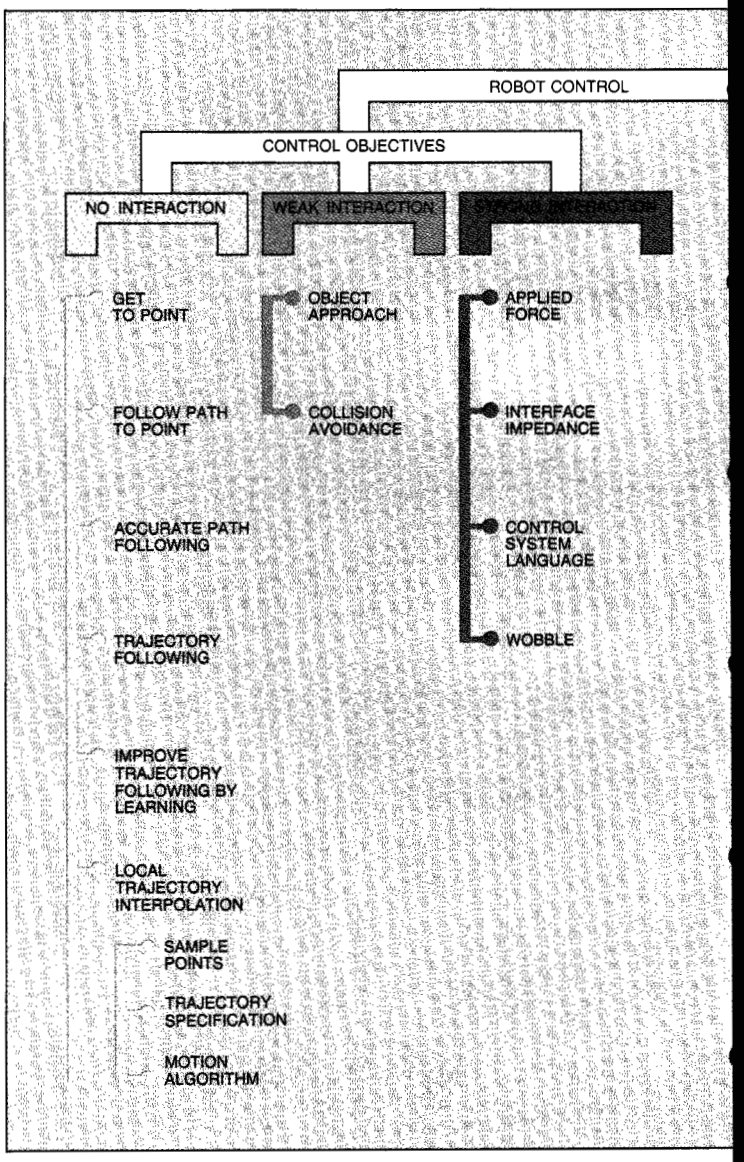
The robot *controller objectives* are the responsibilities delegated to the robot controller. The controller objectives are specified in terms of the system model considered. Examples of objectives for a robot arm end effector are "follow a given trajectory in free space" or "move to a specified position." A more complex example is "move to a specified position while avoiding obstacles." This might include obstacles which are moving or unexpected. Two examples of objectives which involve compliant motion are "insert a peg into a hole" and "command a given point on a robot to behave as if it were a spring or a damper." In this section, we discuss how controller objectives may be attained, rather than how the specifications are written or obtained.

Given the system model and the controller objectives, many *control methods* may be used in the controller design to achieve the required task with satisfactory or optimal performance. There are two basic types of control: open-loop control and closed-loop control.

In *open-loop control* the robot actuator commands are independent of the actual, achieved robot motion. For example, consider the case of a task in which a robot arm is to move from point A to point B, and a force is pushing the arm away from point B. An open-loop control cannot modify the arm command to overcome the disturbing force. In contrast, when *closed-loop control* is exercised, a sensory measurement of the actual motion can be used to generate an appropriate correction to the actuator commands, in order to overcome the unpredictable disturbances and achieve the required task. Closed-loop control is used to overcome uncertainty in the controlled system.

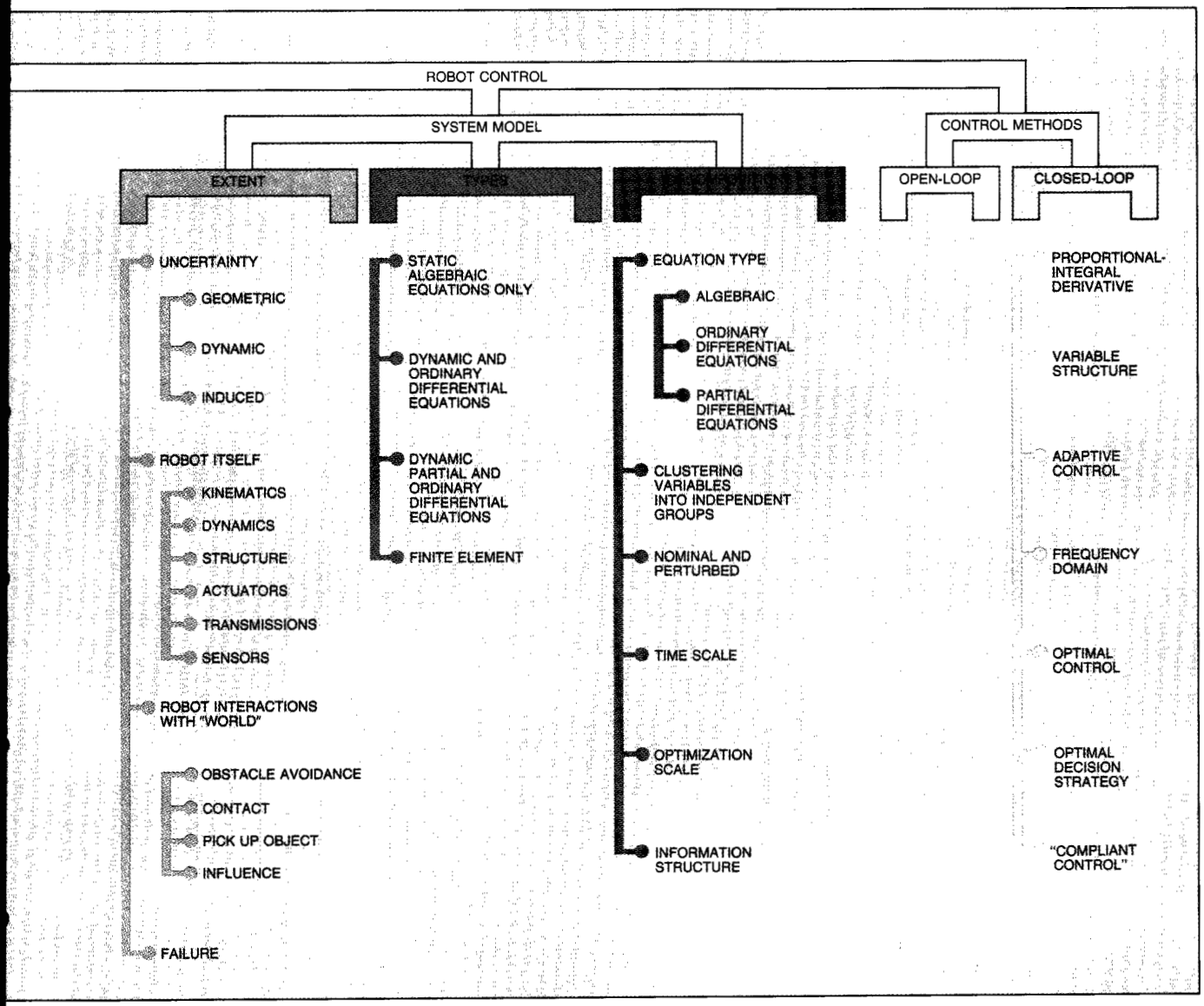
In most cases, closed-loop control is divided into a *feedforward* part and a *feedback* part. The feedforward part is a function of only the commands to the controller. The feedback part is a function of the actual measured responses as well. When the feedback part does not exist, the feedforward part constitutes an open-loop controller.

Figure 13 Framework for robot control



Because the choice of system model depends on the controller objectives, we begin with a discussion of these objectives.

Controller objectives. Figure 13 shows the robot controller objectives being divided into the following three categories, depending upon the amount of interaction required between the robot and the objects in its environment:



- No interaction, as in following a given trajectory in free space
- Weak interaction, as in approaching an object or avoiding obstacles
- Strong interaction, as in compliant motion such as grasping

The most common controller objective is for the robot end effector to follow a trajectory that is completely defined in advance by a higher-level *trajectory planner* that is responsible for handling interactions with the environment. From the point of view of the controller, there is no interaction between the robot and its environment. Currently, even designing a robot controller that can achieve this objective is difficult and is still a subject of research. Interestingly, the control-design problem involved with *weak interactions* is not necessarily more difficult than that

of following a trajectory with a given tolerance. In contrast, the problem of designing a robot controller with an objective of *strong interaction* is much more difficult. In this case, the best hope for a satisfactory solution is to delegate to the robot controller the responsibility of responding to the local interactions between the robot and its environment. This is because the controller provides the most direct path between sensors and actuators and has the required detailed information on both the local interaction model and its current state. It is in an ideal position to take quick corrective action.[48]

No interaction motions. The most common case in current robot systems is that in which no interaction with the environment is considered. The following is a list of objectives of this type:

- Get to a point.
- Follow an approximate path to a point with sufficient accuracy to avoid collisions.
- Follow a given path accurately, as in laser cutting.
- Follow a trajectory.
- Improve trajectory accuracy by learning.
- Perform local trajectory interpolation.

The simplest task of a robot is to move its hand from one point to another. The simplest case is for the final position of the hand to lie within a specified tolerance and to achieve this within some settling time. A more difficult objective is to keep the hand within a specified tolerance along its entire path. This objective is required to avoid collisions. Still more difficult is to follow a specified trajectory, which requires that the hand be at a specified point along the path at a specified time. To improve tracking performance, learning methods were suggested by Raibert[103] and Craig.[104]

We now consider the improvement of trajectory accuracy by learning. Virtually all robots at work in factories today repeat their motion in cycles. This causes part of the error encountered to repeat itself from cycle to cycle. Craig[104] designed an adaptation scheme to modify the required trajectory and cancel any repeated tracking errors sensed in previous cycles. Using this method, the repetitive error caused by the use of an inaccurate or approximate model for the trajectory planning can be learned by the robot and compensated. Therefore, the errors that remain are reduced to those that are not repetitive. Thus, a more experienced robot can do a better job, as one expects.

We now consider the problem of local trajectory interpolation by the controller. The motion planner computes a representation of the required trajectory. In most cases, the representation is a set of evenly spaced in time position sample points: the goals at each sample of time. Often the sample rate of the position goals provided by the motion planner is not sufficient for smooth operation of the controller, and interpolation is used to estimate intermediate points. This method of specifying the trajectory to the controller has a number of undesirable features. It requires that the motion planner perform excessive computation to produce a large number of sample points. This large number of sample points further burdens the communication to the controller with a large amount of data that requires a short transmission latency. Although the controller is usually required to be sufficiently intelligent to compute interpolation points, it does not receive any direct information on the motion trend (e.g., whether the robot is accelerating or slowing down) which can help the controller achieve better performance. Several alternative methods of specifying the local trajectory objective are proposed to alleviate these problems:

- Generate trajectory sample points at a lower rate, but include with each point both the desired position and several of its derivatives (usually velocity and acceleration).
- Specify a trajectory to the controller by passing parameters of the functions to be achieved (e.g., polynomials or spline functions).
- Specify motion algorithms that depend on measured data (e.g., guarded move).

Collision avoidance. Several researchers have investigated dynamic collision avoidance within the robot controller. By dynamic collision avoidance, we mean that the robot path is adjusted in real time so as to avoid collision with obstacles whose position or trajectory is not known in advance. The global problem of obstacle avoidance, discussed in an earlier section, is a very difficult one. An approach using only local information is discussed here.

This method involves constructing a *potential field* in which obstacles make a positive contribution and the goal makes a negative contribution, where both contributions depend on the distances involved. At each moment in time, the arm follows the position gradient to the minimum potential, consequently approaching the goal while at the same time being deflected from obstacles.

This obstacle-avoidance method was implemented for a two-degree-of-freedom manipulator by Hogan[105] and Andrews.[106] Kleinwaks implemented another dynamic obstacle-avoidance scheme for a three-degree-of-freedom articulated robot, using Optimal Decision Strategy.[107] Khatib[79] implemented a more general dynamic obstacle-avoidance control that included not just the robot hand but the entire arm of a six-degree-of-freedom PUMA 600 robot. Obstacle avoidance by the entire robot arm was achieved by including a deflection potential generated by the distance of each one of the robot links to the obstacles in its path. In Khatib's implementation the position and orientation of the obstacles were collected by an industrial vision system, thus allowing the robot to sense moving obstacles and respond in real time.

If the robot is to turn a crank without pulling on its pivot, its motion should comply with the geometric constraints of the crank.

One problem with these systems is that a potential is generated even by objects that cannot cause a collision. This results in extraneous avoidance motion. Krogh[108] extended the idea of a position-dependent potential field to a generalized potential field that depends on both positions and velocities of the robot hand with respect to each obstacle along its path. Krogh's extension has not been implemented, but appears promising for overcoming some of the artifacts of position-only methods.

A basic characteristic of the potential field approach is that the robot path is determined by local properties of the potential field. Therefore, there is no guarantee that the robot motion is appropriate in any global sense or that the robot can achieve its intended goal. Even using only local information, these methods have been shown to produce globally acceptable paths in many typical situations. In complex situations a more general solution might be obtained by using a global trajectory planner to provide the controller with intermediate subgoals, and then using the local techniques.[108]

Compliant motion. When the robot task involves a strong interaction between the robot and its environment, the robot motion is constrained by that interaction. Most strong interaction tasks arise when the robot is required to perform a motion that involves contact with objects in the robot environment. For these types of motion, the robot has to comply with the environment, and, therefore, it is called *compliant motion.* A common example of compliant motion in industrial robotics is the ordinary problem of inserting a peg into a hole.

Another simple example of compliant motion is the task of a robot turning a crank. This task is used here to demonstrate several solution strategies for compliant motion. The *natural constraints* on a crank that arise from geometrical constraints (the pivot) reduce the number of degrees of freedom from six (in free space) to one (turning around the pivot). If the robot is to turn a crank without pulling on its pivot, its motion should comply with the geometric constraints of the crank and only apply motion to the one free degree of freedom left in the manipulation problem.

Mason[109] introduced a convenient way to specify the motion of the unconstrained degrees of freedom in compliant motion by specifying additional *artificial constraints* that are carried out by the robot actuators. When combined with the *natural constraints* imposed by the task geometry, these constrain the manipulation problem to have a unique solution that is the required goal trajectory.[48,109]

Four approaches to compliant motion are considered in robotics:

- Force control
- Impedance control
- Task-level control
- Wobble (dithering or vibration)

We now explain each of these approaches by following the solution of the crank-turning problem.

Force control. A common approach to compliant motion in robotics is to control the force applied by the robot hand rather than the position of the hand. Force control permits compliance by allowing the user to specify the amount of force to be used to

resist the forces that naturally arise in manipulation. This is in contrast to conventional position control, where the robot exerts an arbitrarily large force to achieve the specified position. Force control may be specified in joint coordinates or any set of cartesian coordinates. Often force is specified for some degrees of freedom and position for others.

For example, a robot may turn a crank using combined force and position control by applying position control to the one revolute degree of freedom and force control to all the others to null forces in all directions other than the tangential one.[110] This strategy relies on accurate knowledge of the coordinate system of the crank, because a slight error in the direction can lead to large forces on the crank pivot. More information may be found in a recent survey of robot force control by Whitney.[111]

Impedance control. The requirement for precise knowledge about the direction of the degree of freedom in which motion is permitted may be reduced by the use of impedance control.[105] Using impedance control, one specifies not only the required position of the robot hand but also its required local behavior. This behavior is expressed as the required relation between the position error and the force to be applied by the robot to correct it. The required relation may include time derivatives of the position error as well. One impedance-control strategy to solve the crank-turning example[106] is to specify a desired crank position $\mathbf{x}_0$ that is a quarter of a turn ahead of the current position. The relationship between the force $\mathbf{f}$ and the position error $\mathbf{x} - \mathbf{x}_0$ is given as the equation for a spring:

$$\mathbf{f} = -K(\mathbf{x} - \mathbf{x}_0). \tag{1}$$

The spring constant matrix K must not be too large. The specification of Equation (1) can also be obtained by specifying a potential field that is a function of position $\mathbf{x}$. The force $\mathbf{f}$ results from the gradient of that potential with respect to position. The potential field specification leads to a nonlinear generalization of Equation (1) and was also used to include local obstacle-avoidance specifications for control.[105,112]

Equation (1) can be generalized by including a bias force $\mathbf{f}_0$, as shown in Equation (2):

$$\mathbf{f} - \mathbf{f}_0 = -K(\mathbf{x} - \mathbf{x}_0). \tag{2}$$

The system then tries to achieve a steady state in which it is pushing with force $\mathbf{f}_0$.

By varying the spring constant in Equation (2), we can vary the required control continuously from force control when $K = 0$ to position control when K approaches infinity. (For simplicity, K is assumed to be a scalar.) Thus, we see that impedance control is a generalization of both position and force control.

Task-level control. In some cases one would like to further generalize the requirements from the robot controller beyond position, force, and impedance. Such a generalization, suggested by Ish-Shalom,[48] is a Control System (CS) language to describe *artificial constraints* that uniquely specify the goal trajectory. The Control System language system structure is shown in Figure 14. The CS language description of the required *artificial constraints* includes a set of equations and a set of inequalities that describe the desired relations among objects the robot is manipulating. The equations and inequalities generally depend on sensory measurements obtained by the robot system. Geschke[46] implemented a subset of such a system that included vector equations but ignored dynamics resulting in slow motion.

Using the CS language approach, one can specify the *artificial constraints* for the crank-turning problem by using a cross-product equation,

$$\mathbf{f} \times \mathbf{v} = 0. \tag{3}$$

Equation (3) specifies that the direction of the force $\mathbf{f}$ the robot needs to apply should be parallel to the measured velocity $\mathbf{v}$ at that point.[48]

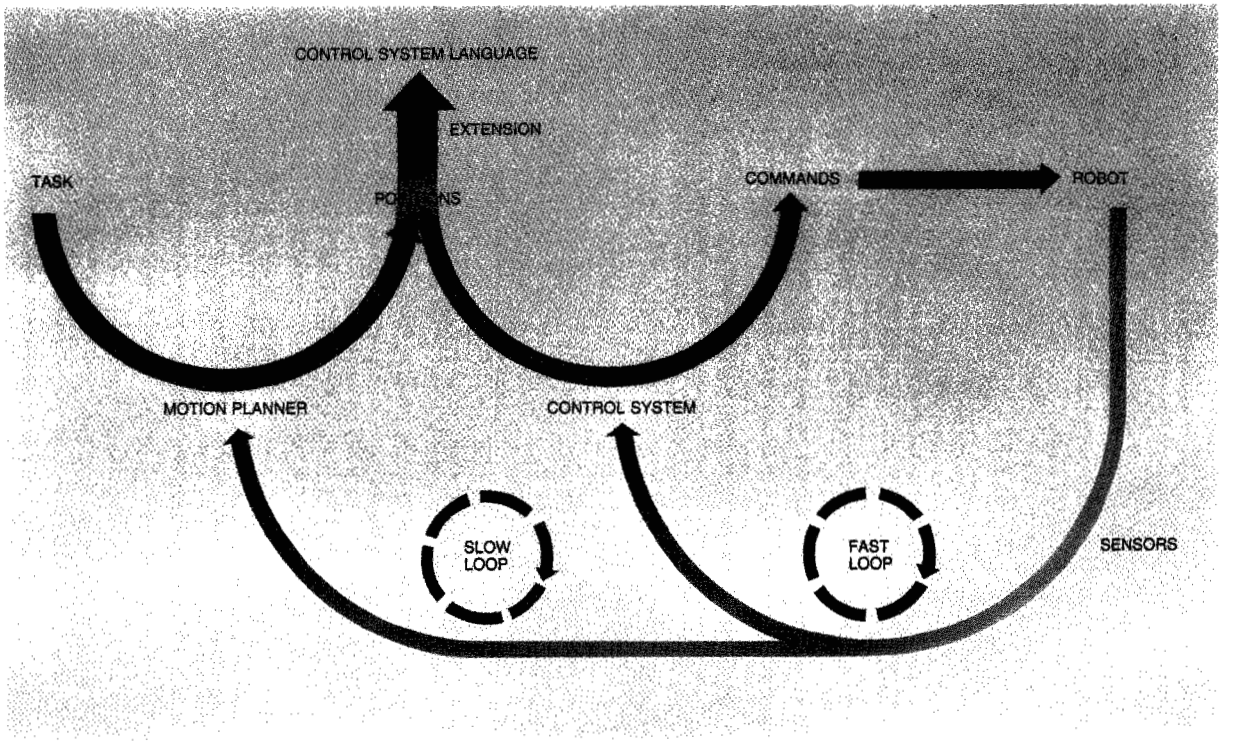
Wobble. In the Wave[113] and AL[26] robot languages one can specify a wobble (dithering or vibration) amplitude for a motion. Adding a high-frequency wobble can help in assembly tasks by breaking *stiction* (i.e., sticking friction) and by causing parts to mate by random interactions. Usually the wobble is at a frequency beyond the response of the closed-loop robot control, and it is thus best done open-loop. Therefore, a separate specification has to be added to the design of the robot controller so that such a motion can be achieved. In the case of the crank-turning example, wobble can be used to break sticking and to determine the initial direction in which the crank is free to turn.

System model

Several aspects of the *system model* are considered:

- The *extent* of the system model
- The *type of equations* describing the model
- The *model decompositions* used

Figure 14 Control system language system structure



Model extent. The following issues may be considered in the system model:

- Uncertainty
- Model of the robot itself
- Robot interaction with the world
- Failures

Currently, there is no robot controller that considers all these items. In some cases, not all of the issues need be considered, as in the case of a laser-cutting robot that has no mechanical interactions with the part it is cutting. We now consider each of these issues in more detail.

Uncertainty. There are many sources of uncertainty in a robot system. These include uncertainties about the values of geometric or dynamic parameters, such as link dimensions and inertias, respectively. They also include uncertainties that are induced by intentional approximations, such as the use design of independent joint servos or the neglect of a "world" model.

Geometric uncertainty limits the robot accuracy, but it can still allow quite good repeatability of an open-loop controlled robot in many cases. Geometric uncertainty can be largely overcome by a closed-loop feedback control, where an appropriate sensor measurement is used to observe the required relationship and modify the robot actuator commands appropriately. *Dynamic uncertainty* limits the predictability of the dynamic response of the robot. This, in turn, limits path-tracking accuracy. The dynamic uncertainty also limits the ability of feedback control to correct for uncertainty. To simplify the control system many *approximations* are used in the system model. Often these approximations actually result in better robot performance because the simpler approximate control computations allow for a higher repetition rate of the control calculation. Using feedback control, the uncertainty can be reduced to a limit determined primarily by the accuracy and resolution of the available sensors.

Robot model. Usually the system model includes only a model of the robot itself. This is appropriate

in the case where the robot has no interaction with the workpiece and there is no concern for obstacles.

To achieve high reliability of manufacturing lines, the mean time to failure for the robot should be of the order of months.

In general, the following elements bear consideration in formulating the model:

- Kinematics: Relations between motion of robot links and cartesian positions
- Dynamics: Relations between applied forces and torques and robot link positions as a function of time
- Robot structure (rigid or nonrigid links)
- Actuators
- Mechanical transmission elements between actuators and robot links
- Sensors

Robot world interaction model. In order to adequately control interactions between the robot and its environment, a model of these interactions is needed in the controller. Some of these interactions include

- Obstacle avoidance
- Contact with objects
- Picking up objects, which results in changes of inertia
- Performing a task that requires the robot to achieve a given relationship among objects in the robot world, as in grinding a weld seam to smooth the surface

Robot failure model. A robot is a mechanical device, and as such it can have mechanical and electrical failures. The problem is further compounded in that a single robot is usually just one part of a manufacturing line. In order to achieve high reliability of manufacturing lines, the mean time to failure for the robot should be of the order of months. Also, robots are expected to work hard, often several shifts. The life expectancy of a robot should be at least ten years, which means that it is expected to perform for over a hundred thousand hours. One can compare this to the average life of a car, which is only several thousand active hours.

In order to achieve high reliability, low maintenance, and long life expectancy, a robot failure model is desirable. Reliable performance might be achieved if the robot could continue to function with minor (or maybe even major) failures. A robot can continue to function under failure conditions if it has sufficient redundancy built into it and its control is sufficiently flexible to adapt to the failure met. A failure model is also helpful in the diagnosis of robot failure, which may reduce downtime. Currently, robot failure models are practically nonexistent or extremely rudimentary.

Model types. There are several types of mathematical models commonly used to describe robot systems:

- Static model, with only algebraic equations
- Dynamic model, with only ordinary differential equations (ODEs)
- Dynamic model, with both ordinary and partial differential equations (PDEs)
- Finite-element model

Static model. By a static model we mean a model that uses only algebraic equations to describe relationships among variables. Many robots are designed with this kind of model in mind. That is, the robot model involves only kinematic relationships and ignores any dynamic effects. The relationships are just the kinematic transformation between the required position in the task space coordinate system (usually world cartesian coordinates) and the required joint angles. Such a model is sensible when the robot joint actuators are controlled by a position command and have a very high inherent stiffness. An actuator has high positional stiffness if the actuator position does not deviate much from the commanded position even when a disturbing force is applied.

Most robots that use step-motors for their actuators fall under this category, because step-motors are commanded to move a given number of steps and they are very stiff. In this situation, the control system is relatively simple, because an open-loop controller can be used and no joint position or

velocity sensors are required. Therefore, the cost of the system is very low; currently such a robot can be purchased for just several thousands of dollars. Nevertheless, such a robot can have very good repeatability because of the high stiffness of the step-motor actuators. Unfortunately, in this case, there is a trade-off between robot speed and the positional resolution it can achieve. Newer control techniques[114–116] may be used to improve this aspect of step-motor performance. However, this requires that a dynamic model of the robot be considered, as we discuss next.

To model static and dynamic effects of flexibility, it is necessary to consider partial differential equations.

The problem of controlling a system described by algebraic equations is not a control problem in the classical sense. The problems involved are those of evaluating a function and finding its inverse. These problems are often studied in the field of numerical analysis.

Dynamic model with ODEs only. In most cases, the command to an actuator corresponds to the steady-state force or velocity to be produced by the actuator. In this case, and in general, a dynamic model is required when the command to the actuator is not the same as the actuator output. A dynamic model is also required when one wants fast motion and at the same time wants to specify the robot motion as a function of time, rather than merely final position.

The main requirement for a dynamic model arises from the use of feedback control. It is almost never possible to design a stable feedback controller without a dynamic model. The simplest dynamic model involves only ordinary differential equations (ODEs). The simplest form of a dynamic model for a robot is a second-order differential equation that results from Newton's second law. The simplest dynamic model is for a single linear axis of a rigid-link cartesian robot,

$$m\ddot{x} = u(t), \tag{4}$$

where m is the total mass the linear actuator has to move and $\ddot{x}$ is the acceleration (second derivative of position) resulting from the controlled input force $u(t)$ produced by the linear actuator. Each linear axis of a rigid-link cartesian robot has an independent equation of the form of Equation (4). Friction and actuator dynamics are omitted from Equation (4) for simplicity.

For a serial-link revolute rigid robot, the equivalent of Equation (4) is much more complex. Following Bejczy,[117] it is as follows:

$$M(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{u}(t), \tag{5}$$

where $\mathbf{q}(t)$ is a column vector of generalized coordinates representing joint positions, $M(\mathbf{q})$ is an $n \times n$ matrix of acceleration-related terms, $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})$ is a column vector function representing a combination of gravitational, centripetal, and Coriolis forces, and $\mathbf{u}(t)$ is a column vector that represents the input torque at each joint.

Note that for a cartesian robot without a wrist, the dynamic equations of the type of Equation (4) are linear and decoupled. That is, the equation for each axis is independent of those for the other axes. In contrast, the equations of the type of Equation (5) for an anthropomorphic robot, like a human being, are nonlinear and coupled. The control of a system with linear dynamics is well understood, whereas for nonlinear equations there is at this time no good theory for controller design. For this reason, a large variety of heuristics have been developed for robot control.

Dynamic model with PDEs. A real robot is not made of infinitely rigid links. Lack of consideration of the flexibility of the robot structure (links) results in static and dynamic model errors. In order to model static and dynamic effects of flexibility, it is necessary to use a more complex model involving partial differential equations (PDEs). Even calculating the static position of the robot under its own weight becomes complex, and finite-element methods may be required.

One wonders how it is possible that the flexibility of the robot structure is considered by very few controllers. The reason is that the sensors used for feedback control of each actuator are always located

exactly at that actuator position. When the sensors are collocated with the actuator, stable control is easy to achieve.[118] Collocated control implies that each joint must be controlled as an independent system. Consequently, coordination of the motion of the joints is limited to feedforward only. This method, which is currently in common practice, results in relatively poor control of coordination between axes.

A current trend in robotics is toward lighter-weight robots to reduce mass and improve accelerations. Often, however, lighter weight is concomitant with greater flexibility. Another trend is toward the use of *endpoint sensing*, in which the sensor is placed so as to directly measure its relationship to the workpiece and thus increase the robot precision. Both of these trends imply the increasing importance of accurately modeling and controlling flexible structures. Cannon and other authors have studied this problem recently.[118–123]

Model decompositions. As seen from the discussion in the previous sections, the model for a robotic system is usually very complex. It involves nonlinear algebraic equations, nonlinear ordinary differential equations, and even partial differential equations. In order to simplify the analysis and control design using these equations, the model is decomposed into as many independent parts as possible. Furthermore, an attempt is made to obtain a hierarchical decomposition. One method is to take advantage of direct mathematical decompositions of the model equations. Another is to separate the controller objectives upon which the model depends. An example is that of separating the specification of a nominal trajectory from the specification of local behavior along that trajectory. Some of the methods used in system decomposition are the following:

- Separation of equations of different types (algebraic only, or including ODEs, or including ODEs and PDEs)
- Clustering of variables into independent groups
- Separation of nominal control from perturbation control
- Separation of activities whose time scales are different
- Separation into a simplified global problem and more detailed local problems
- Clustering of related information and processing

In most cases, even after these decompositions, the subsystems are still too complex, and further approximations are used. Usually one can afford to use only a relatively simple model for the control design process. A more complex model can be used (and usually is) for design verification and system simulation.

Control methods

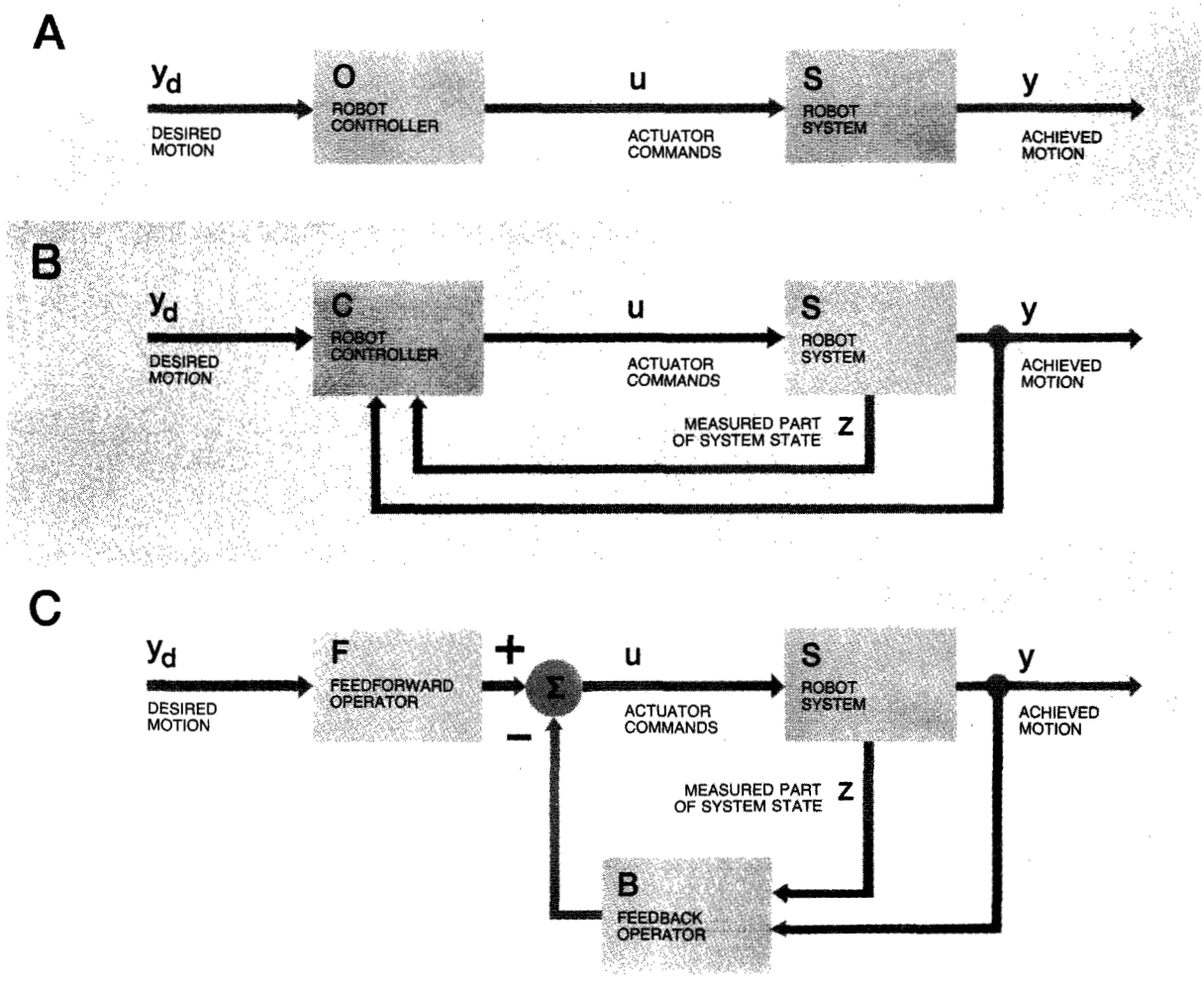
Figure 15 shows three standard forms of controller structures. Part A shows an open-loop controller, and Parts B and C depict closed-loop controllers. Part B shows a generic closed-loop controller, and Part C depicts a commonly used decomposition into *feedforward* and *feedback* parts. In standard control terminology the desired motions $\mathbf{y}_d(t)$ are the inputs to the controller, $\mathbf{u}(t)$ are the actuator commands to the robot, $\mathbf{y}(t)$ are the motions achieved by the robot, and $\mathbf{z}(t)$ are other measured variables related to the robot motion, where t is time. The desired motion $\mathbf{y}_d(t)$ can represent desired position or force in either task space or joint space.

However, this standard controller model does not encompass some types of input requirements. One example is the specification of the desired response to external perturbations (stiffness) required in impedance control. Therefore, in looking at the controllers in Figure 15 one must remember that there are other input requirements besides $\mathbf{y}_d(t)$. Specifications of this type are required for compliant motion.

Most of the work in robotics has concentrated on open-loop control or the feedforward part of the controller. The feedforward controllers are usually designed to cancel in one way or another the nonlinear dynamics of the robot [e.g., Equation (5) for an anthropomorphic robot]. The method used to cancel the nonlinear dynamics is critical to performance when implemented on a real system with limits on computational speed and accuracy. In our discussion we concentrate on closed-loop control methods and the design methods for the feedback control part, as shown in Figure 15C, which are crucial to overcoming the inherent uncertainty in the system.

As discussed earlier, the system can be decomposed into several parts, which usually implies that the controller can be decomposed in a similar way. The most common decomposition is a decentralized controller structure that has independent joint controllers. This feedback controller considerably simplifies both the dependencies between the controller parts and the required real-time control computations, but it has many limitations.

Figure 15 Open-loop and closed-loop, feedforward and feedback control: (A) open-loop robot controller; (B) closed-loop robot controller; (C) closed-loop robot controller partitioned into feedforward and feedback operators



There are several mechanical problems that make robot control difficult. These include static friction and backlash, which make it hard to achieve force control or very high translational resolution. For this reason, a number of direct-drive robots that have low friction and backlash have recently been constructed.[114–116,124–126] Stable control for direct-drive robots is much more demanding, because there is no friction to decouple the joints and absorb some of the energy of structural vibration.

Recently a survey of current industrial robot controls was done by Luh.[29] Discussions and selected papers appear in References 75 and 127. Several books that discuss robot control are available.[42,128–131] In this section we just point out the basics of several control methods, their relations, and their relative merits, including the following:

- Proportional integral derivative (PID) control
- Variable structure (sliding mode control)
- Adaptive control
- Frequency-domain design methods
- Optimal control
- Optimal decision strategy (ODS)
- Compliant control

Proportional integral derivative (PID) control. The fundamental problem in controlling anthropomorphic robots is that the actuators produce motion in joint coordinates, whereas the motion is specified and measured in world coordinates. The rigid body dynamics of a robot are basically those of a moving mass. Thus they can be described by a second-order differential equation, as exemplified by Equation (5). A simple method for control is to cancel the term $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})$, which represents a combination of nonlinear Coriolis, centripetal, and gravity forces. This is done, first, by dividing the control signal vector, $\mathbf{u}$, into feedforward and feedback components ($\mathbf{u} = \mathbf{u}_{ff} + \mathbf{u}_{fb}$). Then choose $\mathbf{u}_{ff}$ to be equal to $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})$ to obtain a cancellation, and then multiply both sides by the inverse of $M(\mathbf{q})$. Note that $M(\mathbf{q})$, the inertia matrix, is symmetric positive definite and can always be inverted. This results in a linear second-order ordinary differential equation[132] with the new control input $\tilde{\mathbf{u}}_{fb}(t)$,

$$\ddot{\mathbf{q}}(t) = M^{-1}(\mathbf{q})\mathbf{u}_{fb}(t) = \tilde{\mathbf{u}}_{fb}(t). \quad (6)$$

To control such a second-order plant, one can use a PID controller of the following form, in which the scalar case is shown for simplicity:

$$\tilde{u}_{fb}(t) = -k_e e(t) - k_I \int_0^t e(t)dt - k_v \dot{e}(t), \quad (7)$$

where the joint position error is given by

$$e(t) = q(t) - q_d(t). \quad (8)$$

Here, $q(t)$ is the joint position, $q_d(t)$ is the desired joint position, and $\dot{e}(t)$ is the time derivative of the joint position error. The PID controller has feedback control signal $\tilde{\mathbf{u}}_{fb}$ proportional to a weighted sum of three forms of joint error. The P (proportional) term corresponds to position error, the I (integral) term corresponds to the accumulated position error, and the D (derivative) term corresponds to velocity error. The P term is required in order to achieve zero error; the I term is required in order to try to achieve a zero steady-state error; and the D term is required to achieve the motion without oscillations.

Note that the feedforward term $\mathbf{u}_{ff}$ requires an inverse dynamics computation. Methods of this type are often called *computed torque control.*[117] In a similar method, the errors e are given in cartesian space and an appropriate coordinate transformation is used. In robotics, this is called *resolved motion force control.*[133]

The main problem with these methods is the inaccuracy in the cancellation of $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})$ and in the knowledge of the inertia matrix $M(\mathbf{q})$. Both the variable structure control and adaptive control methods offer a remedy for some of these problems while still using a similar controller structure. Another approach to solving this control problem is to use a robot that has independent linear second-order joint dynamics as in Equation (4). Cartesian robots are one case; a mechanically decoupled revolute direct-drive arm developed by Asada[124] is another. Thus, a complex control problem was solved by a careful mechanical design.

The basic idea in all adaptive control schemes is that the controller's job is to measure the current system and control it.

Variable structure control. Explicit accounting for parameter uncertainty can be achieved by using sliding mode control, sometimes known as suction control.[134] In this method, a set of equations are used to define a virtual surface in phase space that includes the goal. The control system first brings the system to this surface and then causes it to slide along to the goal. The system is forced to stay close to the surface by using a control that always points the trajectory toward the surface. Each time the system crosses the surface, the control switches the trajectory back toward the surface. This switching technique overcomes inaccuracies in the model, allowing a simplified model to be used. Control on the sliding surface is based on a simple differential equation.

One difficulty is that switching control can impose undesirable high-frequency oscillations in the robot. This problem was resolved[135] by adding a boundary layer around the switching surface to smooth the control. A simulation comparison by Slotine[135] shows that, even for relatively small uncertainty in the robot model, the sliding mode control already can achieve better performance than the computed torque method. Furthermore, the large-motion joints and small-motion joints (wrist) can be con-

trolled independently, treating their dynamic interaction as model uncertainty. This results in a much simpler robot controller while still achieving good performance. The main problem with this method is that actuator torque limitations cannot be directly incorporated into the design. A suggested solution was to augment this method with the *optimal decision strategy* (ODS), discussed later in this paper.

Adaptive control. Three methods of adaptive control are discussed here. Each method has a different way to deal with robot system model uncertainty and complexity. The basic idea in all adaptive control schemes is that the system model is not completely known and therefore the controller's job is to first measure the current system and then control it.

The first method uses *model reference adaptive control.*[136] In this case, the controller tries to identify the system locally and cancel any undesired terms, so that the system looks like a set of well-behaved, independent, linear second-order systems for each joint. This method is somewhat similar to the computed torque method, but here the robot-coupled dynamics are measured on the fly rather than precomputed from a model. In comparison with the variable structure method, this adaptive control method eliminates the model uncertainty by continuously measuring and updating the model. When using adaptive control good performance can be expected, even when the robot is carrying a payload that is unknown in advance.

A second method[137] uses a *self-tuning-type adaptive control method.* In this method the parameters of a set of second-order difference equation models—one for each joint—are estimated in real time. The model parameters are used for real-time calculation of the controller gains that are required locally. Another way to describe this method is that at each local point the controller has three jobs: (1) measure what system model it needs to control; (2) calculate the required control gains; and (3) use the calculated control gains to control the system.

The third method[138] is similar to the second method. In this method, however, the known robot model dynamics are first canceled in the feedforward loop (as in the computed torque method), and only then is a self-tuning-type adaptive control method applied to the perturbed system. One can expect this method to give the best performance among the three methods mentioned when the perturbations from the known model are small. The disadvantage of this method compared to the others is that much more computation is required to obtain the feedforward term. Currently, the adaptive control methods just described have only been demonstrated in simulation. A major problem with all adaptive control methods is that of guaranteeing global stability. Currently there is just one theorem about positive, real functions that gives conditions for global stability of an adaptive control method, and its application leads to poor performance.

The advantage of optimal control is that it is a systematic method that can handle systems with strong coupling between the system states.

Frequency domain design methods. These methods deal primarily with linear systems. By using *describing functions*, the method is extended to some nonlinear systems as well. Beyond proportional integral derivative (PID) control, frequency-domain methods are rarely mentioned in current robot control literature, because they are usually difficult to apply to multi-input, multi-output systems like robots. Recently, Cannon used these methods[118] for endpoint control of a flexible-link robot with a redundant "micro" actuator. The flexibility implies an infinite-dimensional model for the system, which makes it virtually impossible to prove stability for any closed-loop controller. Nevertheless, in this case a simple controller was constructed using frequency-domain techniques, and its stability could be proved because of the special structure of the infinite-dimensional-system eigenvalues (pole and zero locations).

Optimal control. In general, optimal control, using the maximum principle, can be applied to any nonlinear system. However, the resulting control is usually of the open-loop type and very complex to calculate. It is used with a simplified model for motion planning in robotics, but it is not usually used in the robot controller itself because open-loop controllers cannot deal effectively with model uncertainty.

The advantage of optimal control is that it is a systematic method that can handle systems with strong coupling between the system states. In the special case when the system dynamics are linear and the optimization criterion is quadratic in both the system state and the controls (an LQ optimal control problem), a simple closed-loop controller can be obtained using state feedback. In this case, the control is just the product of the gain matrix with the system state vector. Using LQ optimal control, multi-input, multi-output stochastic systems can be designed with an explicit trade-off between desired performance and control effort.

The way to take advantage of the special LQ case in robotics is to use a two-level control synthesis of nominal and perturbed systems.[128,139,140] The nominal solution is done in the open-loop manner. Assuming that the system trajectory is close to the nominal one, a linear time-varying model for the perturbed system is acceptable. One can also use LQ optimal control to design a closed-loop controller for the perturbed system. This optimal controller can be designed to handle the resulting coupling in the perturbed model, the model uncertainty, and a trade-off between them and actuator limitations. No other known method solves these three problems simultaneously and results in a controller that is guaranteed to be stable under a variety of uncertainties. However, there are some drawbacks. The resulting control needs measurements of all the states of the system, which is usually impractical. An "observer" may be used to overcome this problem, but only at the cost of severely sacrificing performance or stability in the face of uncertainty. Moreover, the resulting control usually has strong coupling between joints. This coupling can cause instability because the sensors and actuators are usually not collocated, and the structural flexibility cannot be modeled perfectly. Finally, the performance criterion is of an integral type and does not accurately represent real actuators that saturate when some maximum torque is reached.

Optimal decision strategy (ODS). In many cases during the robot motion the saturation limits of the actuators are reached. Spong, Thorpe, and Kleinwaks[107,141] applied optimal decision strategy (ODS) to the problem of robot trajectory-following to overcome the joint torque limits with guaranteed asymptotic stability in the face of bounded model uncertainty. The method uses a pointwise optimal control law, which, at each sample instant, minimizes a weighted norm of the error between the vector of actual joint accelerations and a "desired acceleration vector," subject to hard limit constraints on the torque at each joint. The resulting control can be computed in real time and has guaranteed tracking properties that are quantifiable, within given limits on model uncertainty and actuator torque. The controller has two terms: one to account for the model uncertainty and the other to account for the actuator saturation limits. In the real-time control loop, a calculation of a quadratic programming optimization is performed at each sample time. The resulting control was implemented for a three-link revolute robot on a Motorola 68000 microprocessor.[107] The accounting for model uncertainty is similar to that obtained by Slotine,[135] but it is based on different principles.

Compliant control. As mentioned before, compliant motion requires a control system that will not only follow a given input, but will also impose a required relationship between manipulation variables such as position and force in impedance control. Therefore, current control design methods are not particularly suitable for such a design. In some cases, model reference control may be used where the reference model is constructed to represent the desired robot behavior. Ish-Shalom[48] has suggested a systematic method to translate general compliant control objectives in the form of equations and inequalities (CS language objectives description) into a general optimal control formulation. Furthermore, for some class of equation objectives with linear dynamics, the general optimal control problem is reduced to an LQ optimal control problem that can be practically solved, and the resulting control can be implemented. Unfortunately this method still has many unresolved questions and has not yet been implemented. One of the problems is that of dealing with nonlinear robot equations of the form of Equation (5). This problem is addressed by Koditschek[142] with a technique called natural motion control. This technique uses a PD-type control for which the controller objectives are encoded in the feedback control. The control objectives are formulated as a set of equations that the system should try to satisfy, as in the CS language. Thus the natural motion of the closed-loop system produces the desired robot motion.

Taking an example from the human body, Hogan[143] suggested a method of impedance control using coactivation of an agonist/antagonist pair of actuators. The idea can be demonstrated by comparing the response to force exhibited by a tense arm or a relaxed one. Impedance control can also be achieved

by specifying the P gain in a PID control when the I gain equals 0, which is the most common method used.[112]

In the case where compliant control is associated with force control, all the standard control methods discussed before can be used by making the controlled variable the output actuator force, rather than position, velocity, or acceleration. Such methods have been implemented and discussed by many authors.[20,45,79,105,111–113,133,143–147]

The speed, precision, and utility of robots depend on the actuators and the means by which they will transmit power to their joints.

Actuators and drives

The speed, precision, and utility of robots depend on the actuators and the means by which they transmit power to their joints. Some important criteria for the evaluation of actuators are their dynamic range, the precision with which they may be controlled, the force or torque that they can generate, their size, mass, and cleanliness. The most common type of actuators for robots are electric, hydraulic, and pneumatic.[6]

Pneumatic actuators provide a great deal of torque for the size of the actuator. However, they are hard to control precisely because of the compressibility of air. They are often used in simple robots that merely move back and forth between hard stops, but they are not widely used in more programmable robots. A notable exception is the Utah/MIT hand, which uses an agonist/antagonist pair of pneumatic actuators.[148]

Most advanced robots are either hydraulically or electrically driven. A number of factors tend to favor hydraulic actuators for robots that must carry heavy payloads. One reason is that the torque-to-mass ratio is currently better for hydraulic actuators. Environmental requirements are important. Hydraulic actuators tend to drip hydraulic fluid or at least produce particles, which makes them unsuitable for the clean environments that are often required for electronics manufacturing. On the other hand, some electric motors may be explosion hazards in volatile applications such as paint spraying.

Electric motors have become increasingly popular for powering small-to-medium-sized robots. The design of new types of motors has become an important topic in robotics research. Conventional motors spin fast and generate low torque, thus requiring a transmission for speed reduction. This may be done with gears or with a widely used device called a *harmonic drive*[149] that provides speed reductions of the order of a hundred to one in a very small package. However, transmissions introduce a number of factors, including static friction, binding, wear, backlash, and cogging, that make it difficult to model the motion produced.[150]

A recent trend has been in the direction of *direct drive* arms in which no speed reduction is necessary. At IBM[115,151,152] and elsewhere,[153–155] new types of motors have been invented that produce high torque at low speed. In conjunction with suitable mechanical robot designs, these motors show promise for reducing some of the unpredictable aspects of actuator behavior that have made robot control something less than a science.

Sensing

A variety of sensors and sensing techniques are used in robotics. In this short section we mention a few important ones.

Binary sensors. The most widely used sensors in robotics are binary sensors. The breaking of a beam of light or the depressing of a switch are used to detect the presence of parts in almost every application. It is not unusual to have to monitor hundreds of such sensors in a robot workstation.

Force sensing. Strain gauges are commonly used to measure force. Absolute accuracy, dynamic range, linearity, and hysteresis are some parameters by which the utility of these devices may be judged. Force sensors in the gripper may be used to sense collisions, the presence of an object, tightness of grasp, and the weight of objects.

Force feedback for servocontrol purposes may be obtained in several different ways. In order to register

force along three orthogonal axes and moments about those axes, force-sensing wrists have been devised and are now commercially available.[12,110,156] An alternative is to sense force or torque directly at the joints. For armature-controlled DC servomotors, armature current may be used as a feedback signal.[29] Another alternative is *endpoint sensing*, which is done at the tip. Endpoint sensing is discussed in the following section.

Multiple force sensors may be use to detect slip. Arrays of sensors produce a "force image" that may be used to detect parts and determine their positions from their "footprints."[157]

In order to maintain contact with a surface, as in performing compliant motion, it is desirable to have continuous force feedback from a surface. This suggests the advantage of sensors mounted on a nonplanar, nonrigid surface, like that of a fingertip.[158-161] In attempting to understand the sensory requirements of compliant motion, researchers are being led to re-examine the incredible tactile mechanisms of the human skin.[162-164]

Endpoint sensing. Direct measurement of the relationship between the manipulator end effector and the workpiece is called *endpoint sensing*, as was mentioned in the previous section. Wrist-mounted force sensors, sensors on the end effector, and structured light projected from the end effector are all endpoint sensing techniques. It is highly desirable to use endpoint sensing because it provides a direct measurement of the error to be corrected. However, as was noted earlier, effective control of endpoint sensing is complicated by the flexibility of robot structures.

Proximity sensing. A variety of types of proximity sensors are in use in robotics. Critical parameters are the range of distances over which the sensor is useful, accuracy, linearity, and sensitivity to environmental conditions. Ultrasonic sensors have been widely used in mobile robots. These sensors have, in the past, been somewhat limited by their inability to sense objects at close range accurately. However, a technique used by Miller allows an ultrasonic sensor located in the base of a gripper to accurately sense objects as close as one inch.[165] Another method was developed by Ish-Shalom for a mobility instrument for the blind.[166] Ultrasonic sensors tend to be somewhat sensitive to temperature variations, atmospheric disturbances (humidity, turbulence), and extraneous reflections.

Laser sensors are another means of implementing proximity sensors. A new interferometer-based technique by Williams and Wickramisinghe has been developed to perform micrometer (μm) resolution measurements over distances of a meter. Ranging results with an accuracy of 2 μm have been demonstrated at a distance of 20 cm.[167] Proximity sensors can also be used to acquire *range maps* by scanning over a scene.[168,169]

In the field of robotics, computer vision is often used for the identification and location of parts.

Vision. In the field of robotics, computer vision is often used for the identification and location of parts. The problem of binary two-dimensional vision in environments with controlled lighting is sufficiently well understood to be widely used in industry. A number of companies sell products that may be used to identify parts with learned features with an overhead camera. This kind of system is useful for pick and place of parts under the following three conditions: (1) the part types are known in advance; (2) their orientation may be determined from their profile; and (3) there are no parts touching or occluding one another. Dealing with more difficult problems, like picking a part out of a bin, requires three-dimensional vision. Experimental systems of this type have been developed, but they are as yet too slow and unreliable for commercial use. However, robot vision is promising and continues to be an active area of research in computer vision.[170-172]

Another application of vision is visual servoing, in which the image is used to determine and correct deviation from the desired path. Using special-purpose hardware to compute image moments sixty times a second, Andersson has succeeded in servoing a robot to catch a ping-pong ball.[173]

Structured lighting is a technique used to greatly simplify the image by shining stripes of light on an

object. The image is thresholded, leaving only a deformed light stripe in the image, with its shape revealing surface orientation and irregularities. Structured light has been used effectively to correct paths in applications such as seam welding.[174–176]

The strongest industrial economic incentives for computer vision come from the area of inspection and measurement, rather than robot control. Two-dimensional vision is used widely in industry for the inspection of mechanical and electronic parts. Inspection systems may also employ robots for positioning of parts and camera.

End effectors

Special-purpose end effectors. Special-purpose end effectors are used in most industrial applications. Conventional grippers consist of two parallel fingers with pads for friction, but many variations may be found to handle objects of different shapes, sizes and materials. (See Engelberger[6] for an interesting survey.) Other types of end effectors such as torches, paint guns, screwdrivers, ladles, drills, routers, electromagnets, and suction devices may be used for different applications.

An extremely important and widely used device for applications requiring compliance is the *remote center compliance* device.[12,177,178] The RCC mechanically complies with the forces and torques that arise in insertion operations, providing a substantial increase in performance for assembly applications.

One approach to achieving high-precision positioning with robots is to introduce a small, high-resolution positioning device at the end effector. This strategy, called *coarse-fine* or *macro-micro* positioning, avoids the requirement for a single device that can be positioned accurately in a large workspace.[179–181] A planar positioning device of this type, developed by Hollis, has a motion resolution of 0.5 μm.[182]

Quick-change grippers. The use of a single special-purpose end effector is impractical when an arm is to be used for a number of different types of operations, as in an assembly application. This has led to the use of switchable end effectors. One variety consists of a special "gripper" to which a variety of tools with a standard interface may be attached. This interface must include suitable electrical and pneumatic connections to operate the tools and carry sensory information. A rack of end effectors may be located in the workspace of the robot, and it may be programmed to pick them up and put them down between the steps of its task.

Frequent tool changing may impact the rate of production. An alternative method which has been adopted by some manufacturers is to outfit the end effector with a turret which carries multiple tools or grippers at once (usually two to six). Typically, one tool at a time is made operational by rotating that tool to the arm interface within the turret. This saves the time of picking up and putting down tools, at the cost of carrying around the bulk and weight of the turret with multiple tools.

Sensors in hand. The IBM 7565 robot[63] uses an LED and binary light sensor on opposite fingers of the gripper to determine whether there is an object between the fingers. It also has strain gauges in the finger pads, which are used primarily in guarded moves, often searching for an object, checking for the presence of an object, or attaining a specified grip force.

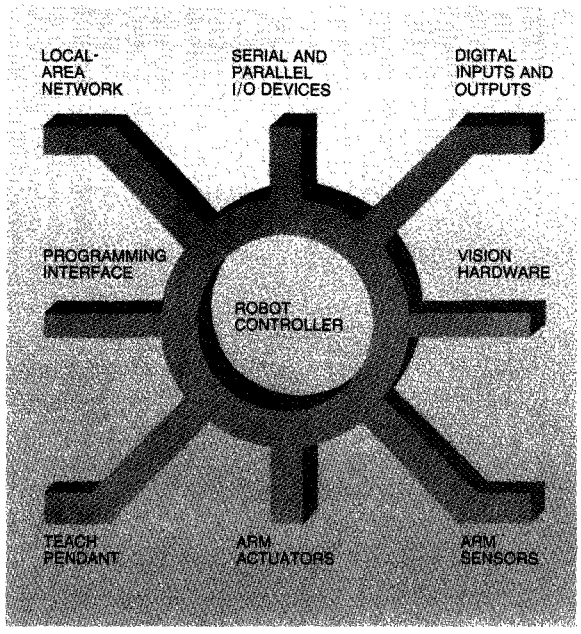
Force-sensing wrists, discussed earlier, obtain measurements that may be converted to cartesian forces and torques.

An experimental hand developed at AT&T Bell Laboratories includes a proximity sensor in its base and arrays of touch sensors on its finger pads.[165,183] This configuration allows the proximity sensor to sense the distance to an object to be grasped, and the touch sensors to detect slip and goodness of grip.

General-purpose hands. The development of articulated robot hands has been fostered by the desire to achieve higher levels of dexterity. By increasing the dexterity of a single tool, the need for special-purpose tools and tool changing will be reduced. However, the dexterity and versatility of a human hand are not attained without complexity.

After a study of the capabilities of a variety of kinematic structures, Salisbury designed the Stanford/JPL hand. This hand has three fingers with three degrees of freedom in each finger.[184] Three generations of development by Jacobsen and others have led to the current version of the Utah/MIT hand.[148] This hand has four fingers, each with four degrees of freedom. Each degree of freedom requires two actuators for flexion and extension, so the fingers require 32 actuators, not including the wrist. Special tapes have been designed for use as "tendons." These tapes have great durability and low friction, which allows

Figure 16 Robot controller interface



them to slide over one another. The actuators powering the hand are pneumatic. The hand may be teleoperated in a master-slave configuration by manipulating a model; it has been designed to be extremely strong and fast and is roughly the size of the human hand. Another interesting anthropomorphic hand is a three-fingered hand developed by Hitachi, Ltd. The fingers of the hand are moved by the expansion and contraction of thin wire made of shape memory alloy, in response to controlled temperature changes.[185]

For more information on robot hands, see the recent book by Mason and Salisbury.[186]

Robot workstation controllers

A robot workstation controller is a computer system used to control a robot or robotic workstation. This is not to be confused with the term *robot controller*, which was used earlier in this paper to describe the portion of the system that embodies the control system. In this section, we will use "controller" to mean robot workstation controller.

Robot controllers must support a wide variety of interfaces, as suggested by Figure 16. First, a controller must provide a suitable interface for the actuators and sensors of one or more robot arms. If the robot is to be sighted, an interface to vision hardware must also be provided. Digital inputs are required to handle the large numbers of simple binary sensors used in the workplace. Digital outputs are required to signal feeders, conveyors, and other devices. A variety of "intelligent devices" that communicate through serial or parallel ports are commercially available; standard interfaces allow a controller to make use of these devices. Local-area network support is required for communication with central computers. This permits centralized record-keeping, global coordination of activities in the factory, and communications with solid modeling systems. The controller must provide interfaces for line attendants, maintenance personnel, and the developers of robotic applications. The requirements for these interfaces may vary from a teach pendant to a graphics workstation.

In order to provide a good control system for the robot, substantial computational ability is required. Controllers often employ a collection of microprocessors to support these computational needs.[55,187] Special-purpose processors have also been proposed for the types of computations that are required in robot control. These include signal processors and systolic arrays to perform the matrix multiplications often required in control,[188–190] special processors for kinematics and dynamics, including fast trigonometric operations,[191,192] and special processors for vision.[193,194]

The controller must carry out real-time activities such as servo control and repeated polling of sensors to monitor conditions. Both servo control and quick response to conditions require guaranteed latencies rather than good overall throughput. This puts demands on the real-time hardware and the operating system.

One problem with current systems is the difficulty of integrating new sensors and devices into the controller for use in compliant or guarded moves. This problem is currently being addressed in the design of new robot controllers.[54,55]

Concluding remarks

In conclusion, we consider the state of the art in robotics in light of the requirements stated early in the paper.

Reliability is a common problem for complex electromechanical systems. Simplification of mechanical

systems and reduction of friction resulting from the use of direct-drive motors may ultimately improve mechanical reliability. The amount of research being done in this area is disproportionately small in comparison with its importance.

Robot *speed* and *accuracy* are being addressed on several fronts. Electromechanical improvements include the use of new light composite structural materials, higher-performance motors, and direct drive. These improvements lead to stringent requirements for the design of robot control systems. With these advances, control-system performance is increasingly limited by speed of computation. Limitations on computational speed are being addressed by the steady improvements in microprocessor technology, by multiprocessor architectures, and by special-purpose hardware for dynamics, kinematics, and control computation. Robots are still slower than people for many tasks, but are already better than people at tasks requiring high positional accuracy.

The ability to perform *compliant motion* is being addressed by the development of new primitives for describing motion and the design of control systems capable of executing them directly. This task is being simplified by the development of direct-drive robots with simpler dynamics. The implementation of compliant motion control is frequently made difficult by current robot workcell controller designs, which make the incorporation of additional sensors difficult. Robots are just starting to be able to comply with external forces. Many compliant tasks which are routine for people still cannot be done by robots.

Robot workcell *configurability* is being addressed by the design of new controller software and hardware that permit sensors and new motion primitives to be easily integrated into the system. The increased use of sensing is critical to many new application areas. Unfortunately, commercially available robot controllers are not highly configurable.

Conventional high-level languages for robot programming include data types, commands, and error-handling facilities, which are convenient for coding some robotic tasks. However, substantial effort is required to program a typical robotic application at this level. Offline programming has simplified the construction of simple, position-oriented robot programs. Work in fine-motion planning, obstacle avoidance, and the analysis of tolerances has contributed to our understanding of task-level programming. However, it is not currently possible to program a robot by giving only a task-level specification.

Cleanliness problems are just beginning to be addressed. Direct-drive robots, which have less friction than conventional robots, may be helpful in this respect.

The field of robotics is in its infancy. Robots have become economically feasible for a number of applications. However, the research challenges are as easy to see as the difference between a current-day industrial robot and a human being.

Acknowledgments

The authors would like to thank a number of people for discussions and comments that have contributed to the substance of this paper. These include Ralph Hollis, Russ Taylor, Larry Lieberman, Jim Colson, V. T. Rajan, and Roger Tsai. We would also like to thank the community of researchers in the many fields that constitute robotics for giving us something to write about.

Cited references and notes

1. K. Capek, *R.U.R.*, Doubleday, Page and Co., New York (1923).
2. E. Binder, "I, robot," *Amazing Stories* **13,** 8–18 (January 1939), Ziff Davis Publishing Co., Chicago, IL.
3. E. Binder, "Adam Link, champion athlete," *Amazing Stories* **14,** 28–47 (July 1940), Ziff Davis Publishing Co., Chicago, IL.
4. I. Asimov, *I, Robot*, Del-Ray Books, Ballantine Books, New York (1950).
5. I. Asimov, *Robots and Empire*, Doubleday and Co., Garden City, NY (1985).
6. J. Engelberger, *Robotics in Practice*, AMACOM, A Division of American Management Associates, New York (1980).
7. R. C. Goertz, "Fundamentals of general purpose remote manipulators," *Nucleonics* **10,** 36–42 (November 1952).
8. *Programmed Article Transfer*, U.S. Patent No. 2,988,237 (1961).
9. H. A. Ernst, *A Computer-Controlled Mechanical Hand*, Sc.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA (1961).
10. D. Downie, "Automatic guided vehicles move into the assembly line," *Modern Materials Handling* **1,** 78–82 (January 1985).
11. J. Nava, "Mobile robots in clean room manufacturing," *Robotics Age* **7,** No. 12, 24–26 (December 1985).
12. J. L. Nevins and D. E. Whitney, "Assembly research," *Automatica* **16,** 595–613 (1980).
13. T. Noguchi, "Recent assembly-inspection robots and their introduction," *Denshi Zairyo* (*Electronic Parts and Materials*; translated from Japanese by the Ralph McElroy Co., Austin, TX), pp. 22–27 (July 1984).
14. O. Friedrich, "The robot revolution," *Time Magazine*, p. 72 (December 8, 1980).
15. B. Levin and A. Doi, "Here come the robots," *Newsweek*, p. 58 (August 9, 1982).
16. M. Raibert and I. Sutherland, "Machines that walk," *Scientific American* **248,** 44–53 (January 1983).

17. D. E. Orin, "Supervisory control of a multilegged robot," *International Journal of Robotics Research* **1,** No. 1, 79–91 (Spring 1982).
18. G. Boothroyd, C. Poli, and L. Murch, *Automatic Assembly*, Marcel Dekker, New York (1982).
19. D. E. Whitney, "State of the art and research needs in robot contact sensing," *Workshop on Intelligent Robots: Achievements and Issues*, SRI International, Menlo Park, CA (November 1984), pp. 137–142.
20. J. P. Trevelyan, P. D. Kovesi, and M. C. H. Ong, "Motion control for a sheep shearing robot," *International Symposium on Robotics Research* **1,** 175–190 (September 1983).
21. T. Lozano-Perez, *Robot Programming*, MIT AI Memo 698a, Massachusetts Institute of Technology, Cambridge, MA (December 1982).
22. D. Grossman, *Programming a Computer Controlled Manipulator by Guiding Through the Motions*, Research Report RC-6393, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (1977).
23. P. D. Summers and D. D. Grossman, "Xprobe: An experimental system for programming robots by example," *International Journal of Robotics Research* **3,** No. 1, 25–39 (Spring 1984).
24. R. P. Paul, "WAVE, A model based language for manipulator control," *The Industrial Robot* **4,** No. 1, 10–17 (March 1977).
25. R. Finkel, R. Taylor, R. Bolles, R. Paul, and J. Feldman, *AL, A Programming System for Automation*, Stanford AI Memo 177, Stanford University, Stanford, CA 94305 (November 1974).
26. S. Mujtaba and R. Goldman, *AL User's Manual*, Stanford AI Memo 323, Stanford University, Stanford, CA 94305 (January 1979).
27. V. Hayward, *Introduction to RCCL: A Robot Control C Library*, Purdue Technical Report TR-EE 83-43, Purdue University, Lafayette, IN 47907 (October 1983).
28. D. Silver, *The Little Robot System*, MIT AI Memo 273, Massachusetts Institute of Technology, Cambridge, MA (January 1973).
29. J. S. Luh, "An anatomy of robots and their controls," *IEEE Transactions on Automatic Control* **AC-28,** 133–153 (February 1983).
30. A. Gilbert, G. Pelton, R. Wang, and S. Motiwalla, "AR-BASIC,® an advanced and user-friendly programming system for robots," *SME Robots* **8,** 20.47–20.64 (June 1984).
31. R. Taylor, P. Summers, and J. Meyer, "AML: A Manufacturing Language," *Robotics Research* **1,** No. 3, 19–41 (Fall 1982).
32. *GE Allegro Documentation*, General Electric Corporation, Schenectady, NY (1982).
33. M. Ward and K. Stoddard, "Karel: A programming language for the factory floor," *Robotics Age* **7,** No. 2, 10–14 (September 1985).
34. J. C. Latombe and E. Mazer, "LM: A high-level language for controlling assembly robots," *Eleventh International Symposium on Industrial Robotics*, Tokyo, Japan (October 1981), pp. 683–690; published by the Japan Industrial Robot Association, 3-5-8 Shibu Kuen Minato-ku, Tokyo, Japan.
35. *Robotic System for Aerospace Batch Manufacturing*, McDonnell Douglas, Inc., St. Louis, MO (February 1980).
36. J. Franklin and G. Vandenburg, "Programming Vision and Robotics Systems with RAIL," *SME Robots* **6,** 392–406 (March 1982).
37. B. Shimano, C. Geschke, C. Spalding III, and P. Smith, "A robot programming system incorporating real-time and supervisory control," *SME Robots* **8,** 20.103–20.119 (June 1984).
38. S. Bonner and K. G. Shin, "A comparative study of robot languages," *IEEE Computer* **15,** No. 12, 82–96 (December 1982).
39. R. Taylor, *The Synthesis of Manipulator Control Programs from Task Level Specifications*, Ph.D. Thesis, Stanford University, Stanford, CA 94305 (1976); available through University Microfilms, 300 N. Zeeb Road, Ann Arbor, MI 48106.
40. R. Brooks, "Symbolic error analysis and robot planning," *Robotics Research* **1,** No. 4, 29–68 (1983).
41. J. Albus, A. Barbera, and R. Nagel, "Theory and practice of hierarchical control," *Proceedings, 23rd IEEE Computer Society International Conference*, Washington, DC (1981); published by IEEE, 345 East 47th St., New York, NY 10017.
42. R. Paul, *Robot Manipulators*, MIT Press, Cambridge, MA (1981).
43. J. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Co., Reading, MA (1982).
44. R. H. Taylor, "Planning and execution of straight line manipulator trajectories," *IBM Journal of Research and Development* **23,** No. 4, 424–436 (July 1979).
45. R. Paul, *Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm*, Ph.D. Thesis, Stanford University, Stanford, CA 94305 (1972); available through University Microfilms, 300 N. Zeeb Road, Ann Arbor, MI 48106.
46. C. C. Geschke, "A system for programming and controlling sensor-based robot manipulators," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-5,** No. 1, 1–7 (January 1983).
47. M. Mason, *Manipulator Grasping and Pushing Operations*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA (1982).
48. J. Ish-Shalom, "The CS language: A new approach to robot motion design," *International Journal of Robotics Research* **4,** No. 1, 42–58 (Spring 1985).
49. J. Nevins and D. Whitney, *The Force Vector Assembler Concept*, Charles Stark Draper Laboratory Report No. E-2754, Cambridge, MA (March 1973).
50. N. Hogan, "Control of mechanical impedance of prosthetic joints," *Proceedings of the 1980 Joint Automatic Control Conference*, San Francisco, CA (August 1980); copyright American Automatic Control Council; distributed by IEEE Service Center, Piscataway NJ 08854.
51. M. Mason, "Compliance," *Robot Motion: Planning and Control*, M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Perez, and M. Mason, Editors, MIT Press, Cambridge, MA (1982).
52. P. Will and D. Grossman, "An experimental system for computer controlled mechanical assembly," *IEEE Transactions on Computers* **C-24,** No. 9, 879–888 (1975).
53. M. D. Donner, "The design of OWL: A language for walking," *ACM SIGPLAN Notices* **18,** No. 6, 158–165 (1983).
54. R. Taylor, J. Korein, G. Maier, and L. Durfee, "Architecture for a general purpose automation controller," *Third International Symposium on Robotics Research*, MIT Press, Cambridge, MA (1986).
55. J. Korein, G. Maier, R. Taylor, and L. Durfee, "A configurable environment for motion programming and control," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA (April 1986); may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
56. E. W. Dijkstra, "Cooperating sequential processes," *Programming Languages*, F. Genuys, Editor, Academic Press, Inc., New York (1968), pp. 43–112.
57. C. A. R. Hoare, "Towards a theory of parallel programming,"

Operating Systems Techniques, Academic Press, Inc., New York (1972), pp. 61–71.
58. P. Brinch Hansen, "The programming language concurrent Pascal," *IEEE Transactions on Software Engineering* **SE-1,** No. 2, 199–207 (June 1975).
59. C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM* **21,** No. 8, 666–677 (August 1978).
60. *Reference Manual for the Ada Programming Language*, J. D. Ichbiah, Editor, U.S. Department of Defense, Advance Research Projects Agency, Washington, DC (1980).
61. C. F. Ruoff, "TEACH—A concurrent robot control language," *Proceedings IEEE COMPSAC*, Chicago, IL (November 1979), pp. 442–445.
62. C. F. Ruoff, "An advanced multitasking robot system," *Industrial Robot* **7,** No. 2, 87–98 (June 1980).
63. *A Manufacturing Language: Concepts and User's Guide*, IBM 7565 Manufacturing System Software Library, Order No. 08007, IBM Corporation; available through IBM branch offices.
64. R. Y. Tsai, *A Versatile Camera Calibration Technique for High Accuracy 3D Machine Vision Metrology Using Off-the-Shelf Cameras and Lenses*, Research Report RC-11413, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (September 1985).
65. L. Foulloy and R. Kelley, "Improving the Precision of a Robot," *IEEE International Conference on Robotics*, Atlanta, GA (March 1984), pp. 62–67; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
66. J. Meyer, "An emulation system for programmable sensory robots," *IBM Journal of Research and Development* **25,** No. 6, 955–962 (November 1981).
67. A. Requicha, "Towards a theory of geometric tolerancing," *International Journal of Robotics Research* **2,** No. 4, 45–59 (Winter 1983).
68. V. Srinivasan and R. Jayaraman, "Issues in Conditional Tolerances for CAD Systems," *IEEE International Conference on Robotics and Automation*, St. Louis, MO (March 1985), pp. 373–375; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
69. J. Korein, "Solid modelling requirements for robotics applications," *NSF Workshop on Intelligent Robots: Achievements and Issues*, SRI International, Menlo Park, CA (November 1984), pp. 299–314.
70. N. J. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Co., Palo Alto, CA (1980).
71. *A Structure for Plans and Behavior*, North-Holland–Elsevier, New York (1977).
72. J. Feldman, "The Stanford Hand-Eye Project," *First International Joint Conference on Artificial Intelligence*, London, England (1971), pp. 350–358.
73. T. Lozano-Perez, *The Design of a Mechanical Assembly System*, MIT AI Memo 397, Massachusetts Institute of Technology, Cambridge, MA (1976).
74. L. Lieberman and M. Wesley, "AUTOPASS: An automatic programming system for computer controlled mechanical assembly," *IBM Journal of Research and Development* **21,** No. 4, 321–333 (1977).
75. *Robot Motion: Planning and Control*, M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Perez, and M. Mason, Editors, MIT Press, Cambridge, MA (1982).
76. D. L. Pieper, *The Kinematics of Manipulators Under Computer Control*, Ph.D. Thesis, Stanford University, Stanford, CA 94305 (1969); may be obtained through University Microfilms, 300 N. Zeeb Road, Ann Arbor, MI 48106.
77. M. Takano, "A new effective solution for inverse kinematics problem (synthesis) of a robot with any type of configuration," *Journal of the Faculty of Engineering, University of Tokyo* **38,** No. 2, 107–135 (1985).
78. J. U. Korein, *A Geometric Investigation of Reach*, MIT Press, Cambridge, MA (1985).
79. O. Khatib, "Dynamic control of manipulators in operational space," *Sixth IFToMM Congress on Theory of Machines and Mechanisms*, New Delhi, India (December 1983).
80. W. A. Wolovich and H. Elliott, "A computational technique for inverse kinematics," *23rd IEEE Conference on Decision and Control*, Las Vegas, NV (December 1984), pp. 1359–1363; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
81. R. Paul, "Manipulator Cartesian path control," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-9,** 702–711 (1979).
82. R. A. Finkel, *Constructing and Debugging Manipulator Programs*, Ph.D. Thesis, Stanford University, Stanford, CA 94305 (1976); may be obtained through University Microfilms, 300 N. Zeeb Road, Ann Arbor, MI 48106.
83. M. E. Kahn and B. Roth, "The near minimum time control of open loop articulated chains," *Journal of Dynamic Systems, Measurement and Control* **93,** 164–172 (1971).
84. P. M. Lynch, "Minimum time sequential axis operation of a cylindrical two axis manipulator," *Proceedings of Joint Automatic Control Conference*, Charlottesville, VA (1981).
85. J. Hollerbach, "Dynamic scaling of manipulator trajectories," *Journal of Dynamic Systems, Measurement and Control* **106,** 102–106 (1984).
86. J. Bobrow, S. Dubowsky, and J. Gibson, "Time optimal control of robotic manipulators along specified paths," *International Journal of Robotics Research* **4,** No. 3, 3–17 (1985).
87. V. T. Rajan, "Minimum time trajectory planning," *IEEE International Conference on Robotics and Automation*, St. Louis, MO (March 1985), pp. 759–764; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
88. E. G. Gilbert and D. W. Johnson, "The application of distance functions to the optimization of robot motion in the presence of obstacles," *IEEE Conference on Decision and Control*, Las Vegas, NV (December 1984), pp. 1338–1344; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
89. M. L. Brown, *Optimal Robot Planning via State Space Networks*, M.S. Thesis, Princeton University, Princeton, NJ (1984); may be obtained through University Microfilms, 300 N. Zeeb Road, Ann Arbor, MI 48106.
90. G. Sahar and J. Hollerbach, "Planning of minimum time trajectories for robot arms," *IEEE International Conference on Robotics and Automation*, St. Louis, MO (March 1985), pp. 751–758; may be obtained through IEEE, 345 East 47th Street, New York, NY 10016.
91. J. M. Hollerbach, "A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamic formulation complexity," *IEEE Transactions on Systems, Man and Cybernetics* **SMC-10,** No. 11, 730–736 (November 1980).
92. S. Udupa, "Collision detection and avoidance in computer controlled manipulators," *Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA (1977).
93. T. Lozano-Perez, "An algorithm for planning collision free paths among polyhedral obstacles," *Communications of the ACM* **22,** No. 10, 560–570 (October 1979).
94. T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-11,** No. 10, 681–698 (October 1981).

95. J. Schwartz and M. Sharir, *On the Piano Mover's Problem I: The Case of a Two Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers*, CS Report 39, NYU Courant Institute (October 1981); may be obtained through New York University, Courant Institute, Washington Square, New York, NY 10003.
96. J. Schwartz and M. Sharir, *On the Piano Mover's Problem II: General Properties for Computing Topological Properties of Real Algebraic Manifolds*, CS Report 41, NYU Courant Institute (February 1982); may be obtained through New York University, Courant Institute, Washington Square, New York, NY 10003.
97. R. Brooks, *A Subdivision Algorithm in Configuration Space for Findpath with Rotation*, MIT AI Memo 684, Massachusetts Institute of Technology, Cambridge, MA (February 1983).
98. R. Brooks, "Planning collision free motions for pick and place operations," *First International Symposium on Robotics Research*, M. Brady and R. Paul, Editors, MIT Press, Cambridge, MA (1984), pp. 5–38.
99. H. Kuntze and W. Schill, "Methods for collision avoidance in computer controlled industrial robots," *Twelfth International Symposium on Industrial Robots*, Paris, France (June 1982), pp. 519–530; published by IFS Ltd., 35–39 High Street, Kempston, Bedford, England.
100. T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers* **C-32,** No. 2, 108–120 (1983).
101. B. Dufay and J. C. Latombe, "An approach to automatic robot programming based on inductive learning," *First International Symposium on Robotics Research*, M. Brady and R. Paul, Editors, MIT Press, Cambridge, MA (1984), pp. 97–115.
102. T. Lozano-Perez, M. Mason, and R. Taylor, "Automatic synthesis of fine motion strategies for robots," *First International Symposium on Robotics Research*, M. Brady and R. Paul, Editors, MIT Press, Cambridge, MA (1984), pp. 65–96.
103. M. Raibert, *A State Space Model for Sensorimotor Control and Learning*, AI Lab Memo AIM-351, Massachusetts Institute of Technology, Cambridge, MA (January 1976).
104. J. J. Craig, "Adaptive control of manipulators through repeated trials," *Proceedings of the American Control Conference* (June 1984), pp. 1566–1573; American Automatic Control Council; distributed by IEEE Service Center, Piscataway, NJ 08854.
105. N. Hogan, "Programmable impedance control of industrial manipulators," *MIT Conference on CAD/CAM Technology in Mechanical Engineering*, Cambridge, MA (March 1982).
106. J. R. Andrews, *Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator*, M.Sc. Thesis, Massachusetts Institute of Technology, Cambridge, MA (February 1983).
107. J. M. Kleinwaks, *Trajectory Control and Obstacle Avoidance for Robot Manipulators with Bounded Inputs*, Ph.D. Thesis, Cornell University, Ithaca, NY (June 1985); may be obtained through University Microfilms, 300 N. Zeeb Road, Ann Arbor, MI 48106.
108. B. H. Krogh, "A generalized potential field approach to obstacle avoidance control," *Robotics International, SME Conference on Robotics Research*, Bethlehem, PA, Paper MS84-484 (August 1984); published by Robotics International of SME, Dearborn, MI.
109. M. T. Mason, "Compliance and force control for computer controlled manipulators," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-11,** No. 6, 418–431 (June 1981).
110. R. Paul and B. Shimano, "Compliance and Control," *Proceedings of the Joint Automatic Control Conference*, San Francisco, CA (1976), pp. 694–699; may be obtained through the American Society of Mechanical Engineers, P.O. Box 930, One SME Drive, Dearborn, MI 48121.
111. D. E. Whitney, "Historical perspective and state of the art in robot force control," *IEEE Conference on Robotics and Automation*, St. Louis, MO (March 1985), pp. 262–268; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
112. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *IEEE Conference on Robotics and Automation*, St. Louis, MO (March 1985), pp. 500–505; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
113. R. Paul, *Wave: A Model-Based Language for Manipulator Control*, SME Technical Paper MR 76-615 (1976); may be obtained through the American Society of Mechanical Engineers, P.O. Box 930, One SME Drive, Dearborn, MI 48121.
114. R. Welburn, "Ultra high torque motor system for direct drive robotics," *Robots 8 Conference*, Detroit, MI (June 1984), pp. 19.63–19.71.
115. J. Ish-Shalom and D. M. Manzer, "Commutation and control of step motors," *Proceedings of the 14th Annual Symposium on Incremental Motion Control Systems and Devices*, Champaign, IL (June 1985), pp. 283–292.
116. R. Curran and G. Mayer, "The architecture of the AdeptOne® direct-drive robot," *Proceedings, American Control Conference* (June 1985), pp. 716–721; distributed by IEEE Service Center, Piscataway, NJ 08854.
117. A. C. Bejczy, *Robot Arm Dynamics and Control*, JPL Technical Memorandum 33-669 (February 1974); may be obtained through the California Institute of Technology, Pasadena, CA 95109.
118. R. H. Cannon, Jr., and E. Schmitz, "Initial experiments on the end-point control of a flexible one-link robot," *International Journal of Robotics Research* **3,** No. 4, 62–75 (Fall 1984).
119. R. H. Cannon, Jr., T. O. Binford, J. D. Meindl, and R. Brooks, *First Annual Report of the Center for Automation and Manufacturing Science*, Stanford University, Stanford, CA 94305; Contract No. F49620-82-C-0092 (November 1983).
120. W. J. Book and M. Majett, "Controller design for flexible, distributed parameter mechanical arms via combined state space and frequency domain techniques," *Robotics Research and Advanced Applications*, ASME Dynamics, Systems and Controls Division, Dearborn, MI (November 1982), pp. 101–120.
121. D. E. Whitney, W. J. Book, and A. Maizza-Neto, "Feedback control of two beam, two joint system with distributed flexibility," *ASME Journal of Dynamic Systems, Measurement and Control* **97,** No. 2, 424–431 (December 1975).
122. J. J. Mendelson and J. R. Rinderle, "Design of a Compliant Robotic Manipulator," *Robotics International, SME Conference on Robotics Research*, Bethlehem, PA, Paper MS84-495 (August 1984); published by Robotics International of SME, Dearborn, MI.
123. D. E. Hardt and A. D. Zalucky, "Active control of robot structure deflections," *Robotics Research and Advanced Applications*, ASME Dynamics, Systems and Controls Division, Dearborn, MI (November 1982), pp. 83–100.
124. H. Asada and K. Youcef-Toumi, "Analysis and design of a direct-drive arm with a five-bar-link parallel drive mechanism," *Proceedings of the American Control Conference*, San Diego, CA (June 1984), pp. 1224–1230; distributed by IEEE

Service Center, Piscataway, NJ 08854.
125. H. Kwakern, Y. Ono, and M. Nikaido, "Development of high torque/precision servo system for direct-drive manipulators," *Proceedings of the Third International Symposium on Robotics Research* (September 1985), pp. 166–175; published by MIT Press, Cambridge, MA.
126. H. Kuwahara, Y. Ono, M. Nikaido, and S. T. Matsumoto, "A Precision Direct-Drive Robot Arm," *Proceedings of the American Control Conference*, San Diego, CA (June 1985), pp. 722–727; distributed by IEEE Service Center, Piscataway, NJ 08854.
127. C. S. G. Lee, R. C. Gonzalez, and K. S. Fu, *Tutorial on Robotics*, IEEE Computer Society Press, ISBN 0-8186-4515-4 (1983); may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
128. M. Vukobratovic and D. Stokic, *Scientific Fundamentals of Robotics 2, Control of Manipulation Robots: Theory and Applications*, Springer-Verlag, New York (1982).
129. Y. Koren, *Robotics for Engineers*, McGraw-Hill Book Co., Inc., New York (1985).
130. H. Asada and J. J. Slotine, *Robot Analysis and Control*, John Wiley & Sons, Inc., New York (1986).
131. J. J. Craig, *Introduction to Robotics: Mechanics and Control*, Addison-Wesley Publishing Co., Reading, MA (1986).
132. J. Y. S. Luh, M. W. Walker, and R. P. Paul, "Resolved-acceleration control of mechanical manipulators," *IEEE Transactions on Automatic Control* **AC-25,** No. 3, 468–474 (June 1980).
133. C.-H. Wu and R. P. Paul, "Resolved motion force control of robot manipulator," *IEEE Transactions on Systems, Man and Cybernetics* **SMC-12,** No. 3, 266–275 (June 1982).
134. K.-K. D. Young, "Controller design for a manipulator using theory of variable structure systems," *IEEE Transactions on Systems, Man and Cybernetics* **SMC-8,** No. 2, 210–218 (February 1978).
135. J.-J. E. Slotine, "The robust control of robot manipulators," *International Journal of Robotics Research* **4,** No. 2, 49–64 (Summer 1985).
136. S. Dubowsky and D. T. DesForges, "The application of model-reference adaptive control to robotic manipulators," *SME Journal of Dynamic Systems, Measurement and Control* **101,** 193–200 (September 1979).
137. A. J. Koivo and T.-H. Guo, "Adaptive linear controller for robotic manipulators," *IEEE Transactions on Automatic Control* **AC-28,** No. 2, 162–171 (February 1983).
138. C. S. G. Lee and M. J. Chung, "An adaptive control strategy for mechanical manipulators," *IEEE Transactions on Automatic Control* **AC-29,** No. 9, 837–840 (1984).
139. S. Desa and B. Roth, "Synthesis of control systems for manipulators using multivariable robust servomechanism theory," *International Journal of Robotics Research* **4,** No. 3, 18–34 (Fall 1985).
140. J. Ish-Shalom, "Optimal Motion for Flight Simulator," Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA (December 1982).
141. M. W. Spong, J. S. Thorpe, and J. M. Kleinwaks, "The control of robot manipulators with bounded input," *IEEE Transactions on Automatic Control* **AC-31,** No. 6, 483–490 (June 1986).
142. D. Koditschek, *Natural Motion of Robot Arms*, Yale Center for Systems Science Technical Report 8409 (February 1985); may be obtained through Yale University, New Haven, CT.
143. N. Hogan, "Adaptive control of mechanical impedance by coactivation of antagonist muscles," *IEEE Transactions on Automatic Control* **AC-29,** No. 8, 681–690 (August 1984).
144. N. Hogan, "Mechanical impedance control in assistive devices and manipulators," *Joint American Control Conference* (August 1980); distributed by IEEE Service Center, Piscataway, NJ 08854.
145. M. H. Raibert and J. J. Craig, "Hybrid position/force control of manipulators," *ASME Journal of Dynamic Systems, Measurement and Control* **102,** 126–133 (1981).
146. J.-J. E. Slotine, "Robustness issues in robot control," *IEEE Conference on Robotics and Automation*, St. Louis, MO (March 1985), pp. 656–661; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
147. H. West and H. Asada, "A method for the design of hybrid position/force controllers for manipulators constrained by contact with the environment," *IEEE International Conference on Robotics and Automation*, St. Louis, MO (March 1985), pp. 251–259; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
148. S. Jacobsen, J. Wood, D. Knutti, and K. Biggers, "The Utah/MIT hand: Work in progress," *First International Symposium on Robotics Research*, M. Brady and R. Paul, Editors, MIT Press, Cambridge, MA (1984), pp. 601–654.
149. *Harmonic Drive Designers Manual*, Harmonic Drive Development, Emhart Machinery Group, Wakefield, MA.
150. R. L. Hollis, *Advances in Robotic Manipulation: Building a Better Mousetrap*, Research Report RC-11234 (No. 50547), IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (1985).
151. J. Pawlettko and H. Chai, "Linear step motors," *Theory and Application of Step Motors*, B. Kuo, Editor, West Publishing Co., St. Paul, MN (1974), pp. 316–326.
152. J. Pawlettko and H. Chai, "Linear stepping motor with uncoupled phases," *Proceedings of the 13th Annual Symposium on Incremental Motion Control Systems and Devices*, Champaign, IL (1984), pp. 245–250; sponsored by the Incremental Motion Control System Society.
153. H. Asada and K. Youcef-Toumi, "Development of a direct drive arm using high torque brushless motors," *First International Symposium on Robotics Research*, M. Brady and R. Paul, Editors, MIT Press (1984), pp. 583–600.
154. G. Horner, R. Lacey, and P. Lawrenson, "High performance brushless PM motors for robotic and actuator applications," *Proceedings of the First European Conference on Electrical Drives, Motors and Controls*, Leeds, England (1982).
155. R. Welburn, "Ultra high torque motor system for direct drive robots," *Proceedings of Robots 8*, Detroit, MI (June 1984), pp. 19.63–19.71; sponsored and published by Robotics International of the Society of Manufacturing Engineers.
156. *Robotic End of Arm Products*, Barry Wright Corp., Watertown, MA (1985).
157. L. D. Harmon, "Automated touch sensing: A brief perspective and several new approaches," *IEEE International Conference on Robotics*, Atlanta, GA (March 1984), pp. 326–331; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
158. R. Bajcsy, "What can we learn from one finger experiments," *First International Symposium on Robotics Research*, M. Brady and R. Paul, Editors, MIT Press, Cambridge, MA (1984), pp. 509–528.
159. K. Lau, J. Bollinger, and N. Duffie, "Automatic contour measurement for two-dimensional geometry," *SME Manufacturing Engineering Transactions*, 10th NAMRC, Toronto, Canada (May 1982), pp. 432–435; may be obtained through Society of Manufacturing Engineers, P.O. Box 930, One SME Drive, Dearborn, MI 48121.
160. K. Lau, N. Duffie, and J. Bollinger, "Automatic contour measurement for three-dimensional geometry," *SME Manufacturing Engineering Transactions*, 13th NAMRC, Berke-

ley, CA (May 1985), pp. 535–540; may be obtained through Society of Manufacturing Engineers, P.O. Box 930, One SME Drive, Dearborn, MI 48121.
161. J. K. Salisbury, Jr., "Interpretation of contact geometries from force measurement," *IEEE International Conference on Robotics*, Atlanta, GA (March 1984), pp. 240–247; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
162. J. M. Hollerbach, "Tactile sensors and interpretation of contact features," *NSF Workshop on Intelligent Robots: Achievements and Issues*, SRI International, Menlo Park, CA (November 1984), pp. 143–152.
163. P. Dario, D. de Rossi, C. Domenici, and R. Francesconi, "Ferroelectric polymer tactile sensors with anthropomorphic features," *IEEE International Conference on Robotics*, Atlanta, GA (March 1984), pp. 332–340; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
164. R. S. Fearing and J. M. Hollerbach, "Basic solid mechanics for tactile sensing," *International Journal of Robotics Research* **4,** No. 3, 40–54 (Fall 1985).
165. G. Miller, R. Boie, and M. Sibilia, "Active damping of ultrasonic transducers for robotic applications," *IEEE International Conference on Robotics*, Atlanta, GA (March 1984), pp. 379–383; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
166. J. Ish-Shalom, *Light Weight Short Range Ultrasonic Range and Velocity Detector—An Aid to the Blind*, Technion Junior Technical College, Haifa, Israel (March 1970).
167. C. Williams and H. K. Wickramisinghe, "Optical ranging by wavelength multiplexed interferometry," *Journal of Applied Physics* **60,** No. 6, 1900–1903 (September 15, 1986).
168. Y. Shirai, "Recognition of polyhedrons with a range finder," *Pattern Recognition* **4,** 243–250 (1972).
169. D. Nitzen, A. Brain, and R. Duda, "The measurement and use of registered reflectance and range data in scene analysis," *Proceedings of the IEEE* **65,** No. 2, 206–220 (February 1977).
170. J. Boissonnat, "A new approach to the problem of acquiring randomly oriented workpieces out of a bin," *Proceedings of the 7th International Joint Conference on Artificial Intelligence* **2,** 796–802 (1981).
171. J. Dessimoz, J. R. Birk, R. B. Kelley, H. A. S. Martins, and C. L. I, "Matched filters for bin picking," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6,** No. 6, 686–697 (1984).
172. R. B. Kelley, H. A. S. Martins, J. R. Birk, and J. D. Dessimoz, "Three vision algorithms for acquiring workpieces from bins," *Proceedings of the IEEE* **71,** No. 7, 803–820 (July 1983).
173. R. L. Andersson, "Real-time gray-scale video processing using a moment-generating chip," *IEEE Journal of Robotics and Automation* **RA-1,** No. 2, 79–85 (1985).
174. T. Bamba, H. Maruyama, E. Ohno, and Y. Shiga, "A visual sensor for arc-welding robots," *Proceedings of the 11th ISIR*, Tokyo, Japan (1981), pp. 151–158; published by the Japan Industrial Robot Association, 3-5-8 Shiku Kuen, Minato-ku, Tokyo, Japan.
175. V. Nicolo, "Industrial robots with sensory feedback application to continuous arc welding," *Proceedings of the 10th ISIR*, Milano, Italy (1980), pp. 15–21; published by the Japan Industrial Robot Association, 3-5-8 Shiku Kuen, Minato-ku, Tokyo, Japan.
176. C. Morgan, "Visual guidance techniques of robot arc-welding," *Proceedings of the 3rd International Conference on Robot Vision and Sensory Controls*, Cambridge, MA (1983).
177. S. Drake, *Using Compliance in Lieu of Sensory Feedback for Automatic Assembly*, Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA (1977).
178. P. C. Watson, "A multidimensional system analysis of the assembly process as performed by a manipulator," *1st North America Robot Conference*, Chicago, IL (1976).
179. A. Sharon and D. E. Hardt, "Enhancement of robot accuracy using endpoint feedback and a macro-micro manipulator system," *Proceedings of the American Control Conference* (June 1984), pp. 1836–1841.
180. R. L. Hollis, R. H. Taylor, M. Johnson, A. Levas, and A. Brennemann, "Robotic circuit board testing using a fine positioner with fiber-optic sensing," *Proceedings of the International Conference on Industrial Robots*, Tokyo, Japan (September 1985), pp. 315–322.
181. A. Sharon, *Enhancement of Robot Accuracy Using a Macro Micro Manipulator System*, M.Sc. Thesis, Massachusetts Institute of Technology, Cambridge, MA (September 1983).
182. R. L. Hollis, "A planar XY robotic fine positioning device," *Proceedings of the IEEE Conference on Robotics and Automation*, St. Louis, MO (1985), pp. 329–336; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
183. R. A. Boie, "Capacitive impedance readout tactile image sensor," *IEEE International Conference on Robotics*, Atlanta, GA (March 1984), pp. 370–378; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
184. J. K. Salisbury and J. K. Craig, "Articulated hands: Force control and kinematic issues," *International Journal of Robotics* **1,** No. 1, 4–17 (Spring 1982).
185. Y. Nakano, M. Fujie, and Y. Hosada, "Hitachi's robot hand," *Robotics Age* **6,** No. 7, 18–20 (July 1984).
186. M. T. Mason and J. K. Salisbury, *Robot Hands and the Mechanics of Manipulation*, MIT Press, Cambridge, MA (1985).
187. R. Nigam and C. G. S. Lee, "A multiprocessor based controller for the control of mechanical manipulators," *Proceedings of the IEEE Conference on Robotics and Automation*, St. Louis, MO (1985), pp. 815–821; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
188. T. Kanade, P. Khosia, and N. Tanaka, "Real-time control of CMU direct-drive arm II using customized inverse dynamics," *23rd IEEE Conference on Decision and Control*, Las Vegas, NV (December 1984), pp. 1345–1352; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
189. H. Kaufman and R. Travassos, *Parallel Computation for Developing Nonlinear Control Procedures*, Report AFWAL TR-81-3016, Wright Patterson Air Force Base, Dayton, OH (1981).
190. G. Shichman, "Personal Instrument (PI)—A PC-based signal processing system," *IBM Journal of Research and Development* **29,** No. 2, 158–169 (March 1985).
191. S. Ahmed and C. S. Besant, "Motion control of industrial robots with closed loop trajectories," *Proceedings of the IEEE International Conference on Robotics*, St. Louis, MO (March 1984), pp. 305–309; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
192. D. Orin, H. Chao, K. Olsen, and W. Schrader, "Pipeline/parallel algorithms for the Jacobian and inverse dynamics computations," *Proceedings of the IEEE Conference on Robotics and Automation*, St. Louis, MO (1985), pp. 785–789; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.
193. M. J. Kimmel, R. S. Jaffe, J. R. Mandeville, and M. A. Lavin, "MITE: Morphic Image Transform Engine—An architecture for reconfigurable pipelines of neighborhood processors," *Proceedings of the IEEE Workshop on Computer*

Architecture for Pattern Analysis and Image Database Management, Miami Beach, FL (November 1985); may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.

194. T. Gross, H. Kung, M. Lam, and J. Webb, "Warp as a machine for low-level vision," *Proceedings of the IEEE Conference on Robotics and Automation*, St. Louis, MO (1985), pp. 790–800; may be obtained through IEEE, 345 East 47th Street, New York, NY 10017.

James U. Korein *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Korein is manager of the Robot Systems Group in the Manufacturing Research Department. He has worked on robot workcell controller design, computational kinematics, and geometric modeling. Dr. Korein is author of the book *A Geometric Investigation of Reach*, published by the MIT Press. This book, based on his Ph.D. thesis, was chosen as a distinguished dissertation by the ACM and MIT Press. Dr. Korein received his B.S. degree from Washington University, St. Louis, in 1974, his M.S. degree from Columbia University, New York, in 1979, and his Ph.D. degree from the University of Pennsylvania, Philadelphia, in 1984, all in computer science. Prior to his Ph.D. work, Dr. Korein was a member of the technical staff at AT&T Bell Laboratories; he joined the IBM Research Division after completing his Ph.D.

Jahuda Ish-Shalom *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Ish-Shalom is a Research Staff Member whose current research is in the areas of sensor-based robot control, active sensing, automatic synthesis of control systems from abstract performance specifications, symbolic solution of matrix Riccati equations, and step-motor design and control. In the past, he has been involved in research on flight simulator motion, eye movement measurement, VLSI, and mobility aids for the blind. In 1985, Dr. Ish-Shalom received the IBM Research Division Outstanding Innovation Award for the invention of a novel method for commutation and control of step-motors. He invented an ultrasonic mobility aid for the blind that won first prize in the 1971 contest "Models and Essays in Natural Science and Mathematics" at the Weitzmann Institute of Science, Rehovot, Israel. Dr. Ish-Shalom received a Ph.D. in biomedical engineering from the Massachusetts Institute of Technology in 1982. He received both his B.Sc. and M.Sc. degrees in electrical engineering from Technion Israel Institute of Technology in 1974 and 1978, respectively.

Reprint Order No. G321-5287.

Database technology

by P. G. Selinger

Computers were originally invented and used to ease and automate the task of computation. As the word "computer" implies, these early machines were used for calculations, such as tabulating census data. As a side effect, the technology needed for storing data was also invented to provide the computational engine with input data and allow it to output results. The means of permanently storing data included punched cards, tape, and disks. Throughout the 1950s and most of the 1960s, the management of stored data was done as required; file systems stored data according to user-defined formats and kept a table of contents. Users shared data by equally ad hoc means, generally by taking turns accessing the same device. Over the years, database technology has evolved through at least three generations to a diverse and sophisticated set of data management tools, as discussed in this paper. This paper has three major sections. Presented first is an introduction to database technology. Presented next is a description of the evolution of database technology from early computing to the sophisticated systems of today. The third section presents a view of both the driving forces that will influence the database technology of the future and also the resulting new directions for the future.

A database management system (DBMS) is a system for managing stored information and providing protocols and a language interface to define, access, and change that information.[1-3] Database management systems can be distinguished from file systems by the level of function they provide, as well as the degree of semantics attributed to the data. A file system stores data as uninterpreted byte strings called *records*, and may or may not provide a directory of files on a per-user or per-system basis. Also, a file system may or may not provide access protection to the data records on a per-file basis. Access to data stored in a file system is through a specific access path defined when the file was created. Various file systems (called *access methods*) offer access to data sequentially or by index or hash key. In addition, a file system may provide concurrent access to files and may protect users from actions of other users by file or record-level locking.

In contrast, a database management system provides higher levels of function on data, often by invoking operations on one or more file systems. A DBMS contains more structured data (records), retrieves data based on content (*field values* within a record), and—unlike a file system—provides a greater degree of independence from the physical layout and logical format of the data and supports the concept of recovery and integrity based on a scope of work known as a *transaction*. In the remainder of this introduction, these various facilities are defined and discussed.

Users interact with the DBMS through language subsets. A data definition language (DDL) is used for defining and changing data objects, whereas a data manipulation language (DML) is used for reading or changing the instances of the data object (the data records). These languages can be used statically in programs (known as host-language embedding) or used dynamically via interactive connections to the DBMS (known as a query interface). DML interactions are typically referred to as *queries*, although the word query may also only refer to reading, not changing the data. Not all DBMSs offer all of the operations of each of these languages in both the host-language and query-interface environments.

A DBMS usually provides directory facilities at a higher, more user-oriented level than file systems do, and separates users from the intricacies of physical storage (e.g., tracks and extents). The directory of a DBMS contains a logical description of the objects it stores, including the external name of the object, its characteristics, the authorization users have on it, and its relationships with or dependencies on other objects.

A DBMS provides a grouping of operations into an atomic unit of work called a *transaction*.

A DBMS stores records as a sequence of fields that take values of a given data type, such as integer, character string of fixed length x, or character string of varying length up to a maximum of y. The DBMS may enforce data types (e.g., does not accept an alphabetic character for an integer field), perform data-type conversions (e.g., integer to floating point), support default values (e.g., zero for integers) or null values (a special value meaning "no information," that is, not among any of the values possible for the data type), and even perform operations on the data such as arithmetic.

In addition, a DBMS provides a grouping of operations into an atomic unit of work called a *transaction*. A user specifies the boundaries of a transaction. A successful termination of a set of operations that change the database is known as a COMMIT, and an unsuccessful termination is an ABORT. The result is that either all or none of the operations within a transaction are executed. This is achieved by a *recovery facility*. The recovery facility serves to preserve the integrity of the database whenever anything goes wrong. These events fall into three general categories: (1) transaction-specific events, (2) events causing loss of the DBMS volatile memory, and (3) events causing loss of the DBMS data on nonvolatile memory (e.g., disk failures).

A specific transaction may abort due to user actions (e.g., the user has halted execution of the application) or due to system actions (e.g., the transaction was chosen to be terminated because it is involved in a deadlock detected by the DBMS). When a transaction aborts, its actions are undone by the recovery facility, leaving the database in the same state that it was in at the beginning of the transaction.

The recovery facility is also invoked when the entire system (either the DBMS or the computer) crashes. When this occurs, the DBMS is restarted (automatically or by an operator). The DBMS then ensures that all transactions which were not completed at the time of the crash are undone, whereas all those which were completed have their effects reapplied to the data if necessary. For transactions that did work at multiple DBMSs, there is a third possibility: The transaction could have been negotiating its completion with other sites, but the decision may not have reached some sites before a crash occurs. This process of negotiation is called "two-phase commit." For these transactions, recovery after a crash is more complex, possibly involving extra communication among sites to determine the transaction status. During this negotiation, the recovery facility sequesters all the data changed by the transaction in order to keep its options open to either commit or abort later.

Finally, the physical media on which the DBMS data are stored, such as disks, can be damaged. When this occurs, the recovery facility can use its historical records to reconstruct the current database contents starting from a (possibly empty) prior version of the database.

These recovery functions can be implemented in a variety of ways—logs, differential files, time stamps, etc. The most popular technique is *logging*, that is, recording on nonvolatile storage (disks) the changes made to the database together with the name of the transaction that made them. Logging techniques may record changes at the physical or logical level, may write these changes before or after the data are changed, may write to two separate media, may record both "before" and "after" versions of changed data, and may store the log in a storage hierarchy (keeping only recent information online). Not all recovery techniques or DBMSs provide all the recovery functions just listed.

Because transactions are units of work that can be undone at any point until commit processing is done, it is not a good idea to permit transactions to read or change the same data that a transaction has already read or changed. Consequently, the DBMS may

provide transactions with isolation from other transactions. One way to supply this isolation is to enforce serial use of the DBMS, not permitting transactions to run concurrently. In general, this leads to unacceptable throughput. Thus, DBMSs provide varying levels of isolation and granularities of isolation through concurrency control techniques that use time stamps, locking, or predeclaration of resources. Predeclaration of resources means analyzing queries before they are executed and scheduling their execution, so that they are guaranteed not to conflict. If locking is done without predeclaring the data to be used, deadlock can occur when transaction A holds locks on some data while waiting for data that transaction B has locked, and B will not release its lock until it acquires a lock on some data that A has locked. Such deadlocks can be detected and resolved by aborting one of the participating transactions.

The first generation of data management took place during the 1950s and most of the 1960s.

History of database technology

We now discuss the evolution of data management systems into database management systems. The categorizations are for my expository purposes; other authors may label these events differently. The purpose here is to demonstrate continuously improving ease of use along with increased function.

First generation: Data management. The first generation of data management took place during the 1950s and most of the 1960s and consisted of user applications doing sequential processing of *master files*, commonly called old-master/new-master processing. This processing was modeled after the physical characteristics of magnetic tapes. The old master file was processed sequentially, record by record, against a file of changes (such as a day's orders) that were sorted into the same order (such as customer account number) to produce a new master file containing the changes. When a new application using the same master-file data was needed, another program to do that processing was written. These applications were written and maintained by users or by their data processing specialists. The data were usually stored on tape, and applications were run as batch programs. Typical applications were orders, inventory management, accounts payable, payroll, and other batch-oriented processing, representing the automation of the "back office" of a business establishment.

Second generation: Database management systems. As much of the data in the "back office" became computerized, the need emerged in the late 1960s and early 1970s for general-purpose data management systems. These systems began evolving to database management systems by centralizing the data previously stored on various magnetic tapes, storing the data under supervision of the enterprise's data processing professionals, and providing a uniform interface to the data. These systems featured disk storage, rather than the usual tape or punched-card storage, and on-line, random access to data. Examples of these systems are access methods such as ISAM[4] and VSAM.[5] These access method systems eliminated one of the problems of the first generation of data management systems, that of multiple and inconsistent copies of the same data.

As this evolution of data management progressed, systems were designed to provide a significant increase in the availability, security, integrity, and consistency of the data they stored. These systems, which included early versions of IMS,[6] can really be called second-generation database management systems. The capabilities of these general-purpose database management systems were far more advanced than those of the typical user-written first-generation data processing applications. They provided direct access to data records by keys and the ability to package many database actions into a single unit of work (called a *transaction*) which, when executed, committed to the database either all of the changes or none of them. Many data protection features were also provided, including protection against lost updates, unauthorized reading or changing, lost data due to media failure, and inconsistent data due to machine failures.

A user who needed data wrote an application program that would call the database management system once for each record required. The call specified the logical data location (called a *segment*), the *fields* requested, and the *access path* to the data (either