CHAPTER 1

Introduction

1.1 BACKGROUND

The history of industrial automation is characterized by periods of rapid change in popular methods. Either as a cause or, perhaps, an effect, such periods of change in automation techniques seem closely tied to world economics. Use of the **industrial robot**, which became identifiable as a unique device in the 1960s [1], along with computer-aided design (CAD) systems and computer-aided manufacturing (CAM) systems, characterizes the latest trends in the automation of the manufacturing process. These technologies are leading industrial automation through another transition, the scope of which is still unknown [2].

In North America, there was much adoption of robotic equipment in the early 1980s, followed by a brief pull-back in the late 1980s. Since that time, the market has been growing (Fig. 1.1), although it is subject to economic swings, as are all markets.

Figure 1.2 shows the number of robots being installed per year in the major industrial regions of the world. Note that Japan reports numbers somewhat differently from the way that other regions do: they count some machines as robots that in other parts of the world are not considered robots (rather, they would be simply considered "factory machines"). Hence, the numbers reported for Japan are somewhat inflated.

A major reason for the growth in the use of industrial robots is their declining cost. Figure 1.3 indicates that, through the decade of the 1990s, robot prices dropped while human labor costs increased. Also, robots are not just getting cheaper, they are becoming more effective—faster, more accurate, more flexible. If we factor these *quality adjustments* into the numbers, the cost of using robots is dropping even faster than their price tag is. As robots become more cost effective at their jobs, and as human labor continues to become more expensive, more and more industrial jobs become candidates for robotic automation. This is the single most important trend propelling growth of the industrial robot market. A secondary trend is that, economics aside, as robots become more capable they become *able* to do more and more tasks that might be dangerous or impossible for human workers to perform.

The applications that industrial robots perform are gradually getting more sophisticated, but it is still the case that, in the year 2000, approximately 78% of the robots installed in the US were welding or material-handling robots [3].

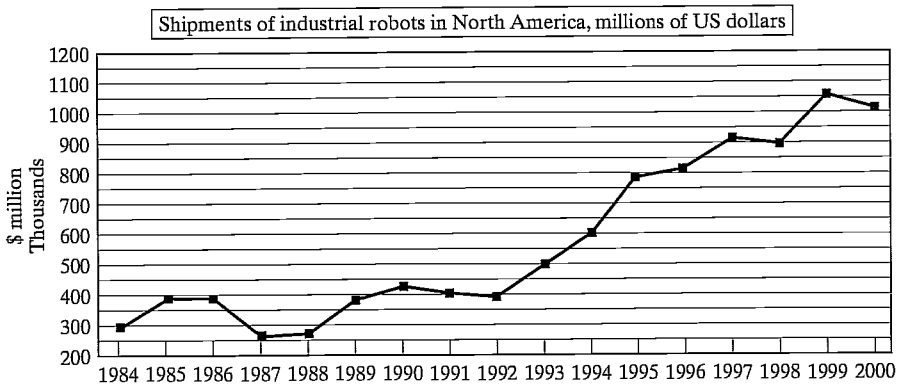

FIGURE 1.1: Shipments of industrial robots in North America in millions of US dollars [3].

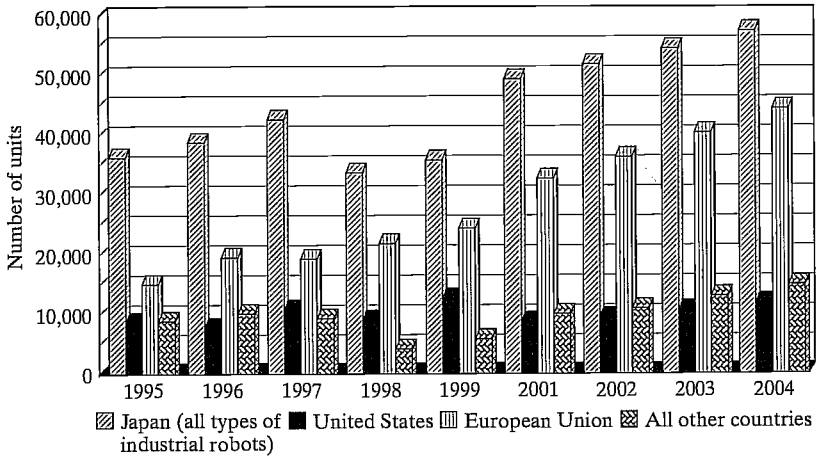

FIGURE 1.2: Yearly installations of multipurpose industrial robots for 1995–2000 and forecasts for 2001–2004 [3].

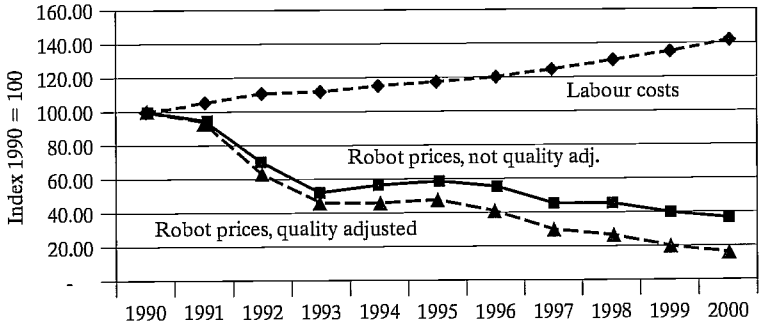

FIGURE 1.3: Robot prices compared with human labor costs in the 1990s [3].

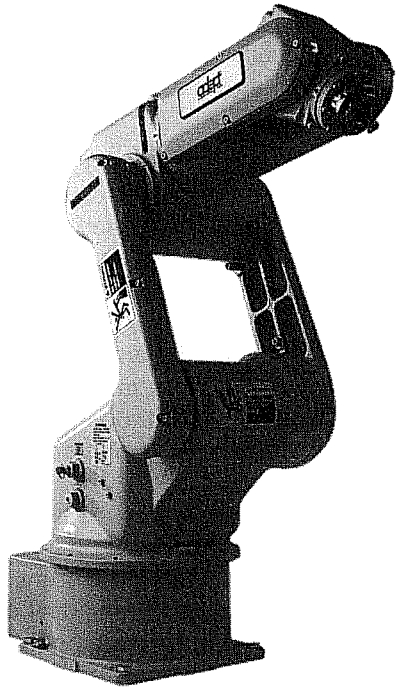

FIGURE 1.4: The Adept 6 manipulator has six rotational joints and is popular in many applications. Courtesy of Adept Technology, Inc.

A more challenging domain, **assembly** by industrial robot, accounted for 10% of installations.

This book focuses on the mechanics and control of the most important form of the industrial robot, the **mechanical manipulator**. Exactly what constitutes an industrial robot is sometimes debated. Devices such as that shown in Fig. 1.4 are always included, while numerically controlled (NC) milling machines are usually not. The distinction lies somewhere in the sophistication of the programmability of the device—if a mechanical device can be programmed to perform a wide variety of applications, it is probably an industrial robot. Machines which are for the most part limited to one class of task are considered **fixed automation**. For the purposes of this text, the distinctions need not be debated; most material is of a basic nature that applies to a wide variety of programmable machines.

By and large, the study of the mechanics and control of manipulators is not a new science, but merely a collection of topics taken from "classical" fields. Mechanical engineering contributes methodologies for the study of machines in static and dynamic situations. Mathematics supplies tools for describing spatial motions and other attributes of manipulators. Control theory provides tools for designing and evaluating algorithms to realize desired motions or force applications. Electrical-engineering techniques are brought to bear in the design of sensors and interfaces for industrial robots, and computer science contributes a basis for programming these devices to perform a desired task.

1.2 THE MECHANICS AND CONTROL OF MECHANICAL MANIPULATORS

The following sections introduce some terminology and briefly preview each of the topics that will be covered in the text.

Description of position and orientation

In the study of robotics, we are constantly concerned with the location of objects in three-dimensional space. These objects are the links of the manipulator, the parts and tools with which it deals, and other objects in the manipulator's environment. At a crude but important level, these objects are described by just two attributes: position and orientation. Naturally, one topic of immediate interest is the manner in which we represent these quantities and manipulate them mathematically.

In order to describe the position and orientation of a body in space, we will always attach a coordinate system, or **frame**, rigidly to the object. We then proceed to describe the position and orientation of this frame with respect to some reference coordinate system. (See Fig. 1.5.)

Any frame can serve as a reference system within which to express the position and orientation of a body, so we often think of *transforming* or *changing the description of* these attributes of a body from one frame to another. Chapter 2 discusses conventions and methodologies for dealing with the description of position and orientation and the mathematics of manipulating these quantities with respect to various coordinate systems.

Developing good skills concerning the description of position and rotation of rigid bodies is highly useful even in fields outside of robotics.

Forward kinematics of manipulators

Kinematics is the science of motion that treats motion without regard to the forces which cause it. Within the science of kinematics, one studies position, velocity,

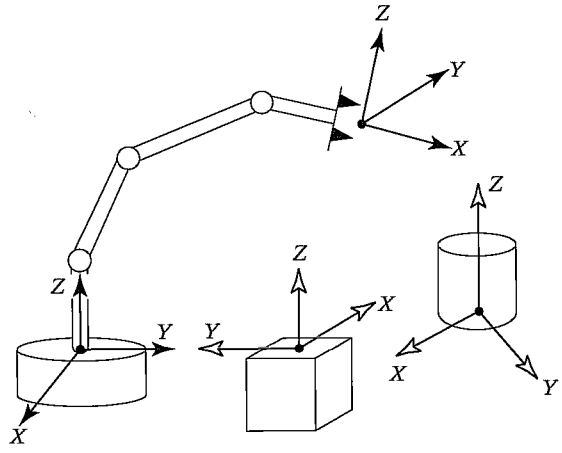

FIGURE 1.5: Coordinate systems or "frames" are attached to the manipulator and to objects in the environment.

acceleration, and all higher order derivatives of the position variables (with respect to time or any other variable(s)). Hence, the study of the kinematics of manipulators refers to all the geometrical and time-based properties of the motion.

Manipulators consist of nearly rigid **links**, which are connected by **joints** that allow relative motion of neighboring links. These joints are usually instrumented with position sensors, which allow the relative position of neighboring links to be measured. In the case of rotary or **revolute** joints, these displacements are called **joint angles**. Some manipulators contain sliding (or **prismatic**) joints, in which the relative displacement between links is a translation, sometimes called the **joint offset**.

The number of **degrees of freedom** that a manipulator possesses is the number of independent position variables that would have to be specified in order to locate all parts of the mechanism. This is a general term used for any mechanism. For example, a four-bar linkage has only one degree of freedom (even though there are three moving members). In the case of typical industrial robots, because a manipulator is usually an open kinematic chain, and because each joint position is usually defined with a single variable, the number of joints equals the number of degrees of freedom.

At the free end of the chain of links that make up the manipulator is the **end-effector**. Depending on the intended application of the robot, the end-effector could be a gripper, a welding torch, an electromagnet, or another device. We generally describe the position of the manipulator by giving a description of the **tool frame**, which is attached to the end-effector, relative to the **base frame**, which is attached to the nonmoving base of the manipulator. (See Fig. 1.6.)

A very basic problem in the study of mechanical manipulation is called **forward kinematics**. This is the static geometrical problem of computing the position and orientation of the end-effector of the manipulator. Specifically, given a set of joint

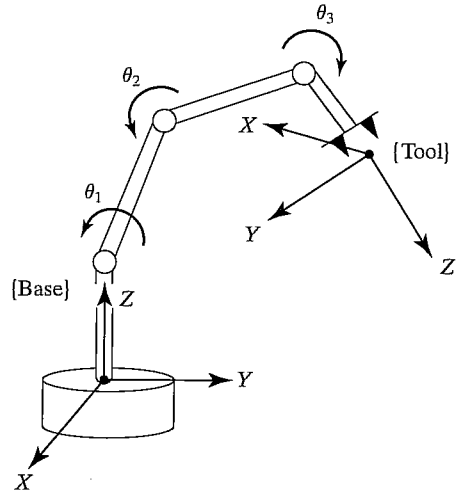

FIGURE 1.6: Kinematic equations describe the tool frame relative to the base frame as a function of the joint variables.

angles, the forward kinematic problem is to compute the position and orientation of the tool frame relative to the base frame. Sometimes, we think of this as changing the representation of manipulator position from a **joint space** description into a **Cartesian space** description.[1] This problem will be explored in Chapter 3.

Inverse kinematics of manipulators

In Chapter 4, we will consider the problem of **inverse kinematics.** This problem is posed as follows: Given the position and orientation of the end-effector of the manipulator, calculate all possible sets of joint angles that could be used to attain this given position and orientation. (See Fig. 1.7.) This is a fundamental problem in the practical use of manipulators.

This is a rather complicated geometrical problem that is routinely solved thousands of times daily in human and other biological systems. In the case of an artificial system like a robot, we will need to create an algorithm in the control computer that can make this calculation. In some ways, solution of this problem is the most important element in a manipulator system.

We can think of this problem as a *mapping* of "locations" in 3-D Cartesian space to "locations" in the robot's internal joint space. This need naturally arises anytime a goal is specified in external 3-D space coordinates. Some early robots lacked this algorithm—they were simply moved (sometimes by hand) to desired locations, which were then recorded as a set of joint values (i.e., as a location in joint space) for later playback. Obviously, if the robot is used purely in the mode of recording and playback of joint locations and motions, no algorithm relating

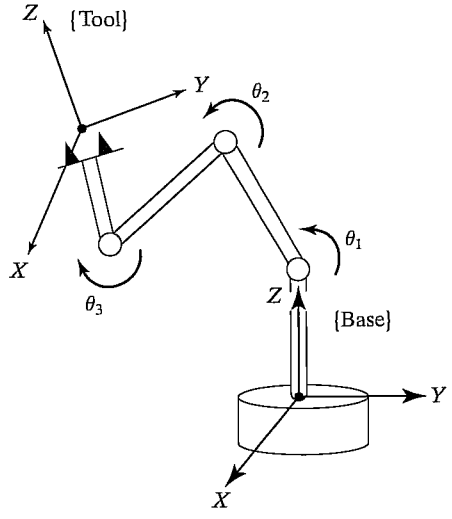

FIGURE 1.7: For a given position and orientation of the tool frame, values for the joint variables can be calculated via the inverse kinematics.

[1] By *Cartesian space*, we mean the space in which the position of a point is given with three numbers, and in which the orientation of a body is given with three numbers. It is sometimes called *task space* or *operational space*.

joint space to Cartesian space is needed. These days, however, it is rare to find an industrial robot that lacks this basic inverse kinematic algorithm.

The inverse kinematics problem is not as simple as the forward kinematics one. Because the kinematic equations are nonlinear, their solution is not always easy (or even possible) in a closed form. Also, questions about the existence of a solution and about multiple solutions arise.

Study of these issues gives one an appreciation for what the human mind and nervous system are accomplishing when we, seemingly without conscious thought, move and manipulate objects with our arms and hands.

The existence or nonexistence of a kinematic solution defines the **workspace** of a given manipulator. The lack of a solution means that the manipulator cannot attain the desired position and orientation because it lies outside of the manipulator's workspace.

Velocities, static forces, singularities

In addition to dealing with static positioning problems, we may wish to analyze manipulators in motion. Often, in performing velocity analysis of a mechanism, it is convenient to define a matrix quantity called the **Jacobian** of the manipulator. The Jacobian specifies a **mapping** from velocities in joint space to velocities in Cartesian space. (See Fig. 1.8.) The nature of this mapping changes as the configuration of the manipulator varies. At certain points, called **singularities**, this mapping is not invertible. An understanding of the phenomenon is important to designers and users of manipulators.

Consider the rear gunner in a World War I–vintage biplane fighter plane (illustrated in Fig. 1.9). While the pilot flies the plane from the front cockpit, the rear gunner's job is to shoot at enemy aircraft. To perform this task, his gun is mounted in a mechanism that rotates about two axes, the motions being called azimuth and elevation. Using these two motions (two degrees of freedom), the gunner can direct his stream of bullets in any direction he desires in the upper hemisphere.

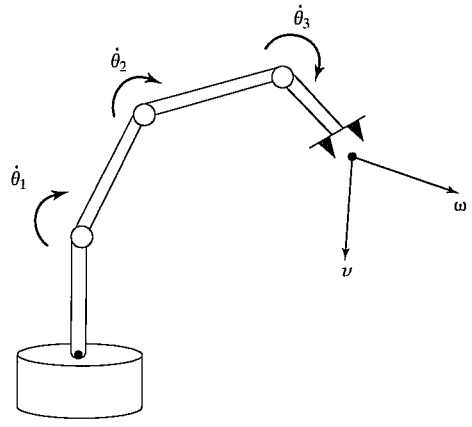

FIGURE 1.8: The geometrical relationship between joint rates and velocity of the end-effector can be described in a matrix called the Jacobian.

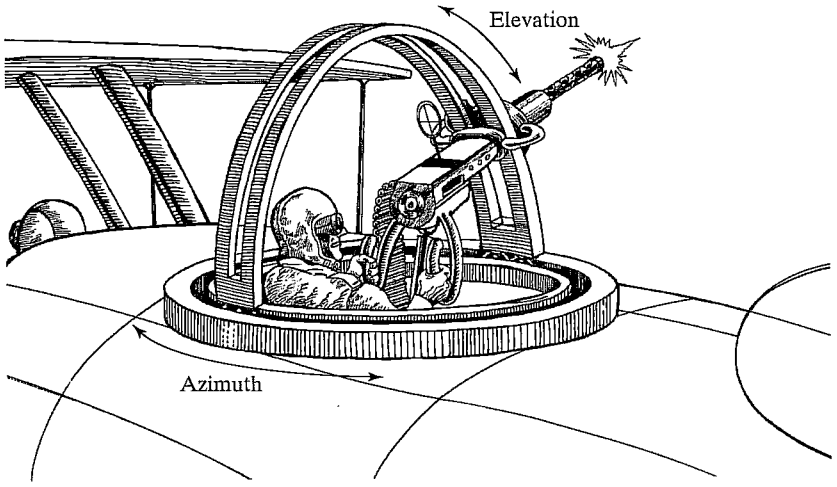

FIGURE 1.9: A World War I biplane with a pilot and a rear gunner. The rear-gunner mechanism is subject to the problem of singular positions.

An enemy plane is spotted at azimuth one o'clock and elevation 25 degrees! The gunner trains his stream of bullets on the enemy plane and tracks its motion so as to hit it with a continuous stream of bullets for as long as possible. He succeeds and thereby downs the enemy aircraft.

A second enemy plane is seen at azimuth one o'clock and elevation 70 degrees! The gunner orients his gun and begins firing. The enemy plane is moving so as to obtain a higher and higher elevation relative to the gunner's plane. Soon the enemy plane is passing nearly overhead. What's this? The gunner is no longer able to keep his stream of bullets trained on the enemy plane! He found that, as the enemy plane flew overhead, he was required to change his azimuth at a very high rate. He was not able to swing his gun in azimuth quickly enough, and the enemy plane escaped!

In the latter scenario, the lucky enemy pilot was saved by a *singularity*! The gun's orienting mechanism, while working well over most of its operating range, becomes less than ideal when the gun is directed straight upwards or nearly so. To track targets that pass through the position directly overhead, a very fast motion around the azimuth axis is required. The closer the target passes to the point directly overhead, the faster the gunner must turn the azimuth axis to track the target. If the target flies directly over the gunner's head, he would have to spin the gun on its azimuth axis at infinite speed!

Should the gunner complain to the mechanism designer about this problem? Could a better mechanism be designed to avoid this problem? It turns out that you really can't avoid the problem very easily. In fact, any two-degree-of-freedom orienting mechanism that has exactly two rotational joints cannot avoid having this problem. In the case of this mechanism, with the stream of bullets directed

straight up, their direction aligns with the axis of rotation of the azimuth rotation. This means that, at exactly this point, the azimuth rotation does not cause a change in the direction of the stream of bullets. We know we need two degrees of freedom to orient the stream of bullets, but, at this point, we have lost the effective use of one of the joints. Our mechanism has become **locally degenerate** at this location and behaves as if it only has one degree of freedom (the elevation direction).

This kind of phenomenon is caused by what is called a **singularity of the mechanism**. All mechanisms are prone to these difficulties, including robots. Just as with the rear gunner's mechanism, these singularity conditions do not prevent a robot arm from positioning anywhere within its workspace. However, they can cause problems with *motions* of the arm in their neighborhood.

Manipulators do not always move through space; sometimes they are also required to touch a workpiece or work surface and apply a static force. In this case the problem arises: Given a desired contact force and moment, what set of **joint torques** is required to generate them? Once again, the Jacobian matrix of the manipulator arises quite naturally in the solution of this problem.

Dynamics

Dynamics is a huge field of study devoted to studying the forces required to cause motion. In order to accelerate a manipulator from rest, glide at a constant end-effector velocity, and finally decelerate to a stop, a complex set of torque functions must be applied by the joint actuators.[2] The exact form of the required functions of actuator torque depend on the spatial and temporal attributes of the path taken by the end-effector and on the mass properties of the links and payload, friction in the joints, and so on. One method of controlling a manipulator to follow a desired path involves calculating these actuator torque functions by using the dynamic equations of motion of the manipulator.

Many of us have experienced lifting an object that is actually much lighter than we expected (e.g., getting a container of milk from the refrigerator which we thought was full, but was nearly empty). Such a misjudgment of payload can cause an unusual lifting motion. This kind of observation indicates that the human control system is more sophisticated than a purely kinematic scheme. Rather, our manipulation control system makes use of knowledge of mass and other dynamic effects. Likewise, algorithms that we construct to control the motions of a robot manipulator should take dynamics into account.

A second use of the dynamic equations of motion is in **simulation**. By reformulating the dynamic equations so that acceleration is computed as a function of actuator torque, it is possible to simulate how a manipulator would move under application of a set of actuator torques. (See Fig. 1.10.) As computing power becomes more and more cost effective, the use of simulations is growing in use and importance in many fields.

In Chapter 6, we develop dynamic equations of motion, which may be used to control or simulate the motion of manipulators.

[2]We use *joint actuators* as the generic term for devices that power a manipulator—for example, electric motors, hydraulic and pneumatic actuators, and muscles.

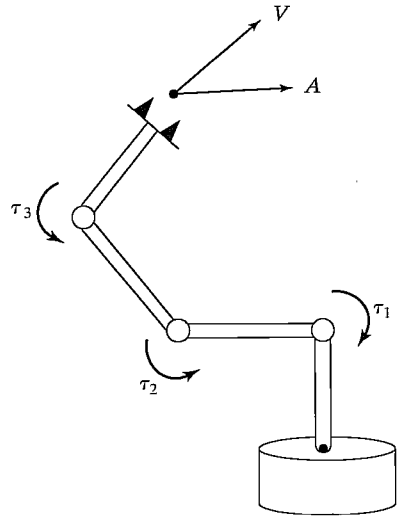

FIGURE 1.10: The relationship between the torques applied by the actuators and the resulting motion of the manipulator is embodied in the dynamic equations of motion.

Trajectory generation

A common way of causing a manipulator to move from here to there in a smooth, controlled fashion is to cause each joint to move as specified by a smooth function of time. Commonly, each joint starts and ends its motion at the same time, so that the manipulator motion appears coordinated. Exactly how to compute these motion functions is the problem of **trajectory generation**. (See Fig. 1.11.)

Often, a path is described not only by a desired destination but also by some intermediate locations, or **via points**, through which the manipulator must pass en route to the destination. In such instances the term **spline** is sometimes used to refer to a smooth function that passes through a set of via points.

In order to force the end-effector to follow a straight line (or other geometric shape) through space, the desired motion must be converted to an equivalent set of joint motions. This **Cartesian trajectory generation** will also be considered in Chapter 7.

Manipulator design and sensors

Although manipulators are, in theory, universal devices applicable to many situations, economics generally dictates that the intended task domain influence the mechanical design of the manipulator. Along with issues such as size, speed, and load capability, the designer must also consider the number of joints and their geometric arrangement. These considerations affect the manipulator's workspace size and quality, the stiffness of the manipulator structure, and other attributes.

The more joints a robot arm contains, the more dextrous and capable it will be. Of course, it will also be harder to build and more expensive. In order to build

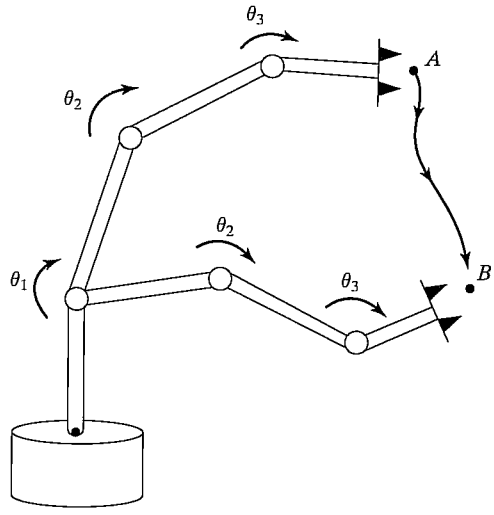

FIGURE 1.11: In order to move the end-effector through space from point A to point B, we must compute a trajectory for each joint to follow.

a useful robot, that can take two approaches: build a **specialized robot** for a specific task, or build a **universal robot** that would able to perform a wide variety of tasks. In the case of a specialized robot, some careful thinking will yield a solution for how many joints are needed. For example, a specialized robot designed solely to place electronic components on a flat circuit board does not need to have more than four joints. Three joints allow the position of the hand to attain any position in three-dimensional space, with a fourth joint added to allow the hand to rotate the grasped component about a vertical axis. In the case of a universal robot, it is interesting that fundamental properties of the physical world we live in dictate the "correct" minimum number of joints—that minimum number is six.

Integral to the design of the manipulator are issues involving the choice and location of actuators, transmission systems, and internal-position (and sometimes force) sensors. (See Fig. 1.12.) These and other design issues will be discussed in Chapter 8.

Linear position control

Some manipulators are equipped with stepper motors or other actuators that can execute a desired trajectory directly. However, the vast majority of manipulators are driven by actuators that supply a force or a torque to cause motion of the links. In this case, an algorithm is needed to compute torques that will cause the desired motion. The problem of dynamics is central to the design of such algorithms, but does not in itself constitute a solution. A primary concern of a **position control system** is to compensate automatically for errors in knowledge of the parameters of a system and to suppress disturbances that tend to perturb the system from the desired trajectory. To accomplish this, position and velocity **sensors** are monitored by the **control algorithm**, which computes torque commands for the actuators. (See

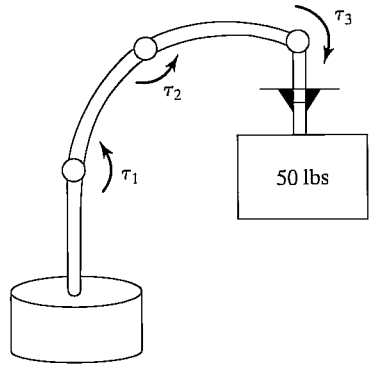

FIGURE 1.12: The design of a mechanical manipulator must address issues of actuator choice, location, transmission system, structural stiffness, sensor location, and more.

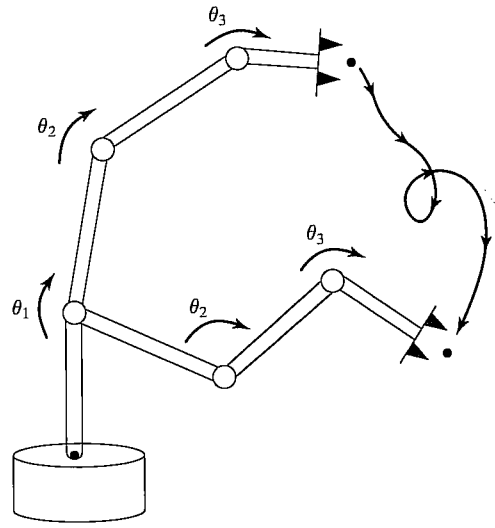

FIGURE 1.13: In order to cause the manipulator to follow the desired trajectory, a position-control system must be implemented. Such a system uses feedback from joint sensors to keep the manipulator on course.

Fig. 1.13.) In Chapter 9, we will consider control algorithms whose synthesis is based on linear approximations to the dynamics of a manipulator. These linear methods are prevalent in current industrial practice.

Nonlinear position control

Although control systems based on approximate linear models are popular in current industrial robots, it is important to consider the complete nonlinear dynamics of the manipulator when synthesizing control algorithms. Some industrial robots are now being introduced which make use of **nonlinear control** algorithms in their

controllers. These nonlinear techniques of controlling a manipulator promise better performance than do simpler linear schemes. Chapter 10 will introduce nonlinear control systems for mechanical manipulators.

Force control

The ability of a manipulator to control forces of contact when it touches parts, tools, or work surfaces seems to be of great importance in applying manipulators to many real-world tasks. **Force control** is complementary to position control, in that we usually think of only one or the other as applicable in a certain situation. When a manipulator is moving in free space, only position control makes sense, because there is no surface to react against. When a manipulator is touching a rigid surface, however, position-control schemes can cause excessive forces to build up at the contact or cause contact to be lost with the surface when it was desired for some application. Manipulators are rarely constrained by reaction surfaces in all directions simultaneously, so a mixed or **hybrid** control is required, with some directions controlled by a **position-control law** and remaining directions controlled by a **force-control law**. (See Fig. 1.14.) Chapter 11 introduces a methodology for implementing such a force-control scheme.

A robot should be instructed to wash a window by maintaining a certain force in the direction perpendicular to the plane of the glass, while following a motion trajectory in directions tangent to the plane. Such split or **hybrid** control specifications are natural for such tasks.

Programming robots

A robot programming language serves as the interface between the human user and the industrial robot. Central questions arise: How are motions through space described easily by the programmer? How are multiple manipulators programmed

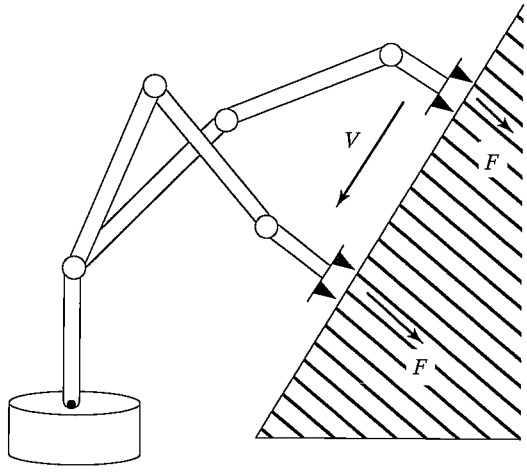

FIGURE 1.14: In order for a manipulator to slide across a surface while applying a constant force, a hybrid position–force control system must be used.

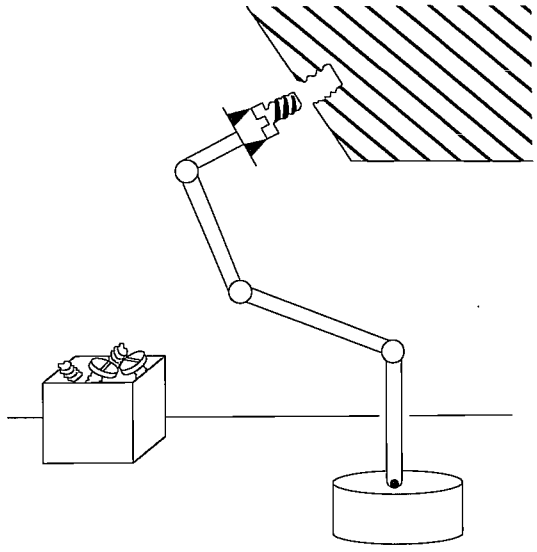

FIGURE 1.15: Desired motions of the manipulator and end-effector, desired contact forces, and complex manipulation strategies can be described in a *robot programming language.*

so that they can work in parallel? How are sensor-based actions described in a language?

Robot manipulators differentiate themselves from **fixed automation** by being "flexible," which means programmable. Not only are the movements of manipulators programmable, but, through the use of sensors and communications with other factory automation, manipulators can *adapt* to variations as the task proceeds. (See Fig. 1.15.)

In typical robot systems, there is a shorthand way for a human user to instruct the robot which path it is to follow. First of all, a special point on the hand (or perhaps on a grasped tool) is specified by the user as the **operational point**, sometimes also called the **TCP** (for Tool Center Point). Motions of the robot will be described by the user in terms of desired locations of the operational point relative to a user-specified coordinate system. Generally, the user will define this reference coordinate system relative to the robot's base coordinate system in some task-relevant location.

Most often, paths are constructed by specifying a sequence of **via points**. Via points are specified relative to the reference coordinate system and denote locations along the path through which the TCP should pass. Along with specifying the via points, the user may also indicate that certain speeds of the TCP be used over various portions of the path. Sometimes, other modifiers can also be specified to affect the motion of the robot (e.g., different smoothness criteria, etc.). From these inputs, the trajectory-generation algorithm must plan all the details of the motion: velocity profiles for the joints, time duration of the move, and so on. Hence, input

to the trajectory-generation problem is generally given by constructs in the robot programming language.

The sophistication of the user interface is becoming extremely important as manipulators and other programmable automation are applied to more and more demanding industrial applications. The problem of programming manipulators encompasses all the issues of "traditional" computer programming and so is an extensive subject in itself. Additionally, some particular attributes of the manipulator-programming problem cause additional issues to arise. Some of these topics will be discussed in Chapter 12.

Off-line programming and simulation

An **off-line programming system** is a robot programming environment that has been sufficiently extended, generally by means of computer graphics, that the development of robot programs can take place without access to the robot itself. A common argument raised in their favor is that an off-line programming system will not cause production equipment (i.e., the robot) to be tied up when it needs to be reprogrammed; hence, automated factories can stay in production mode a greater percentage of the time. (See Fig. 1.16.)

They also serve as a natural vehicle to tie computer-aided design (CAD) data bases used in the design phase of a product to the actual manufacturing of the product. In some cases, this direct use of CAD data can dramatically reduce the programming time required for the manufacturing process. Chapter 13 discusses the elements of industrial robot off-line programming systems.

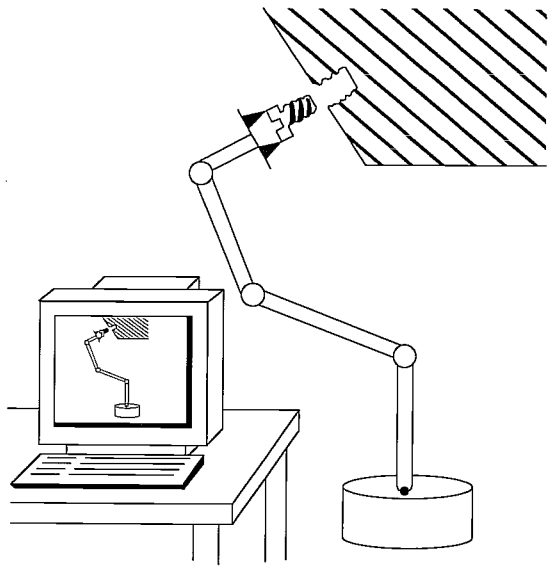

FIGURE 1.16: Off-line programming systems, generally providing a computer graphics interface, allow robots to be programmed without access to the robot itself during programming.

1.3 NOTATION

Notation is always an issue in science and engineering. In this book, we use the following conventions:

1. Usually, variables written in uppercase represent vectors or matrices. Lowercase variables are scalars.
2. Leading subscripts and superscripts identify which coordinate system a quantity is written in. For example, ${}^{A}P$ represents a position vector written in coordinate system $\{A\}$, and ${}^{A}_{B}R$ is a rotation matrix[3] that specifies the relationship between coordinate systems $\{A\}$ and $\{B\}$.
3. Trailing superscripts are used (as widely accepted) for indicating the inverse or transpose of a matrix (e.g., R^{-1}, R^{T}).
4. Trailing subscripts are not subject to any strict convention but may indicate a vector component (e.g., x, y, or z) or may be used as a description—as in P_{bolt}, the position of a bolt.
5. We will use many trigonometric functions. Our notation for the cosine of an angle θ_1 may take any of the following forms: $\cos\theta_1 = c\theta_1 = c_1$.

Vectors are taken to be column vectors; hence, row vectors will have the transpose indicated explicitly.

A note on vector notation in general: Many mechanics texts treat vector quantities at a very abstract level and routinely use vectors defined relative to different coordinate systems in expressions. The clearest example is that of addition of vectors which are given or known relative to differing reference systems. This is often very convenient and leads to compact and somewhat elegant formulas. For example, consider the angular velocity, ${}^{0}\omega_4$, of the last body in a series connection of four rigid bodies (as in the links of a manipulator) relative to the fixed base of the chain. Because angular velocities sum vectorially, we may write a very simple vector equation for the angular velocity of the final link:

$$ {}^{0}\omega_4 = {}^{0}\omega_1 + {}^{1}\omega_2 + {}^{2}\omega_3 + {}^{3}\omega_4. \tag{1.1} $$

However, unless these quantities are expressed with respect to a common coordinate system, they cannot be summed, and so, though elegant, equation (1.1) has hidden much of the "work" of the computation. For the particular case of the study of mechanical manipulators, statements like that of (1.1) hide the chore of bookkeeping of coordinate systems, which is often the very idea that we need to deal with in practice.

Therefore, in this book, we carry frame-of-reference information in the notation for vectors, and we do not sum vectors unless they are in the same coordinate system. In this way, we derive expressions that solve the "bookkeeping" problem and can be applied directly to actual numerical computation.

BIBLIOGRAPHY

[1] B. Roth, "Principles of Automation," Future Directions in Manufacturing Technology, Based on the Unilever Research and Engineering Division Symposium held at Port Sunlight, April 1983, Published by Unilever Research, UK.

[3] This term will be introduced in Chapter 2.

[2] R. Brooks, "Flesh and Machines," Pantheon Books, New York, 2002.

[3] The International Federation of Robotics, and the United Nations, "World Robotics 2001," Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investment, United Nations Publication, New York and Geneva, 2001.

General-reference books

[4] R. Paul, *Robot Manipulators*, MIT Press, Cambridge, MA, 1981.

[5] M. Brady et al., *Robot Motion*, MIT Press, Cambridge, MA, 1983.

[6] W. Synder, *Industrial Robots: Computer Interfacing and Control*, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[7] Y. Koren, *Robotics for Engineers*, McGraw-Hill, New York, 1985.

[8] H. Asada and J.J. Slotine, *Robot Analysis and Control*, Wiley, New York, 1986.

[9] K. Fu, R. Gonzalez, and C.S.G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, New York, 1987.

[10] E. Riven, *Mechanical Design of Robots*, McGraw-Hill, New York, 1988.

[11] J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[12] M. Spong, *Robot Control: Dynamics, Motion Planning, and Analysis*, IEEE Press, New York, 1992.

[13] S.Y. Nof, *Handbook of Industrial Robotics*, 2nd Edition, Wiley, New York, 1999.

[14] L.W. Tsai, *Robot Analysis: The Mechanics of Serial and Parallel Manipulators*, Wiley, New York, 1999.

[15] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*, 2nd Edition, Springer-Verlag, London, 2000.

[16] G. Schmierer and R. Schraft, *Service Robots*, A.K. Peters, Natick, MA, 2000.

General-reference journals and magazines

[17] *Robotics World.*

[18] *IEEE Transactions on Robotics and Automation.*

[19] *International Journal of Robotics Research (MIT Press).*

[20] *ASME Journal of Dynamic Systems, Measurement, and Control.*

[21] *International Journal of Robotics & Automation (IASTED).*

EXERCISES

1.1 [20] Make a chronology of major events in the development of industrial robots over the past 40 years. See Bibliography and general references.

1.2 [20] Make a chart showing the major applications of industrial robots (e.g., spot welding, assembly, etc.) and the percentage of installed robots in use in each application area. Base your chart on the most recent data you can find. See Bibliography and general references.

1.3 [40] Figure 1.3 shows how the cost of industrial robots has declined over the years. Find data on the cost of human labor in various specific industries (e.g., labor in the auto industry, labor in the electronics assembly industry, labor in agriculture, etc.) and create a graph showing how these costs compare to the use of robotics. You should see that the robot cost curve "crosses" various the human cost curves

of different industries at different times. From this, derive approximate dates when robotics first became cost effective for use in various industries.

1.4 [10] In a sentence or two, define kinematics, workspace, and trajectory.

1.5 [10] In a sentence or two, define frame, degree of freedom, and position control.

1.6 [10] In a sentence or two, define force control, and robot programming language.

1.7 [10] In a sentence or two, define nonlinear control, and off-line programming.

1.8 [20] Make a chart indicating how labor costs have risen over the past 20 years.

1.9 [20] Make a chart indicating how the computer performance–price ratio has increased over the past 20 years.

1.10 [20] Make a chart showing the major users of industrial robots (e.g., aerospace, automotive, etc.) and the percentage of installed robots in use in each industry. Base your chart on the most recent data you can find. (See reference section.)

PROGRAMMING EXERCISE (PART 1)

Familiarize yourself with the computer you will use to do the programming exercises at the end of each chapter. Make sure you can create and edit files and can compile and execute programs.

MATLAB EXERCISE 1

At the end of most chapters in this textbook, a MATLAB exercise is given. Generally, these exercises ask the student to program the pertinent robotics mathematics in MATLAB and then check the results of the MATLAB Robotics Toolbox. The textbook assumes familiarity with MATLAB and linear algebra (matrix theory). Also, the student must become familiar with the MATLAB Robotics Toolbox. For MATLAB Exercise 1,

a) Familiarize yourself with the MATLAB programming environment if necessary. At the MATLAB software prompt, try typing *demo* and *help*. Using the color-coded MATLAB editor, learn how to create, edit, save, run, and debug m-files (ASCII files with series of MATLAB statements). Learn how to create arrays (matrices and vectors), and explore the built-in MATLAB linear-algebra functions for matrix and vector multiplication, dot and cross products, transposes, determinants, and inverses, and for the solution of linear equations. MATLAB is based on the language C, but is generally much easier to use. Learn how to program logical constructs and loops in MATLAB. Learn how to use subprograms and functions. Learn how to use comments (%) for explaining your programs and tabs for easy readability. Check out www.mathworks.com for more information and tutorials. Advanced MATLAB users should become familiar with Simulink, the graphical interface of MATLAB, and with the MATLAB Symbolic Toolbox.

b) Familiarize yourself with the MATLAB Robotics Toolbox, a third-party toolbox developed by Peter I. Corke of CSIRO, Pinjarra Hills, Australia. This product can be downloaded for free from www.cat.csiro.au/cmst/staff/pic/robot. The source code is readable and changeable, and there is an international community of users, at robot-toolbox@lists.msa.cmst.csiro.au. Download the MATLAB Robotics Toolbox, and install it on your computer by using the *.zip* file and following the instructions. Read the *README* file, and familiarize yourself with the various functions available to the user. Find the *robot.pdf* file—this is the user manual giving background information and detailed usage of all of the Toolbox functions. Don't worry if you can't understand the purpose of these functions yet; they deal with robotics mathematics concepts covered in Chapters 2 through 7 of this book.