CHAPTER

Kinematic Linkages

5

In describing an object's motion, it is often useful to relate it to another object. Consider, for example, a coordinate system centered at our sun in which the moon's motion must be defined. It is much easier to describe the motion of the moon relative to the earth and the earth's motion relative to the sun than it is to come up with a description of the moon's motion directly in a sun-centric coordinate system. Such sequences of relative motion are found not only in astronomy but also in robotics, amusement park rides, internal combustion engines, human figure animation, and pogo sticks.

This chapter is concerned with animating objects whose motion is relative to another object, especially when there is a sequence of objects where each object's motion can easily be described relative to the previous one. Such an object sequence forms a *motion hierarchy*. Often the components of the hierarchy represent objects that are physically connected and are referred to by the term *linked appendages* or, more simply, as *linkages*. Another common aspect of relative motion is that the motion is often restricted. The moon's position relative to the earth, for example, can be specified by a single parameter (in this case, an angle) since, at least for this discussion, it rotates around the earth in a fixed plane at a fixed distance. The plane and distance are built into the hierarchical model so the animator is only concerned with specifying one rotational parameter. This is an example of a model with *reduced dimensionality* because the hierarchical structure enforces constraints and requires fewer parameters than would be needed to specify the position of the moon otherwise.

This chapter addresses how to form data structures that support such linkages and how to animate the linkages by specifying or determining position parameters over time. As such, it is concerned with *kinematics*.

Of course, a common use for kinematic linkages is for animating human (or other) figures in which limbs are defined by a hierarchy of rotational joints connected by rigid links. The two approaches to positioning such a hierarchy are known as *forward kinematics*, in which the animator must specify rotation parameters at joints, and *inverse kinematics* (IK), in which the animator specifies the desired position of the hand, for example, and the system solves for the joint angles that satisfy that desire. Figure 5.1, demonstrating forward kinematics, shows a sample sequence of rotating a limb's joints. Figure 5.2, demonstrating IK, shows a sample sequence of positioning the hand at the desired location as a procedure automatically solves the required joint angles. These techniques are the subject of this chapter after discussing the fundamentals of modeling such hierarchies.

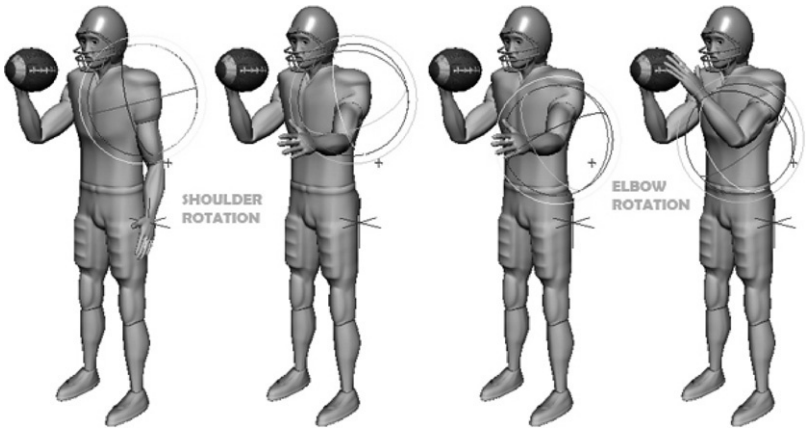


FIGURE 5.1

Sample sequence of forward kinematic specification of joint rotations.

(Image courtesy of Domin Lee.)

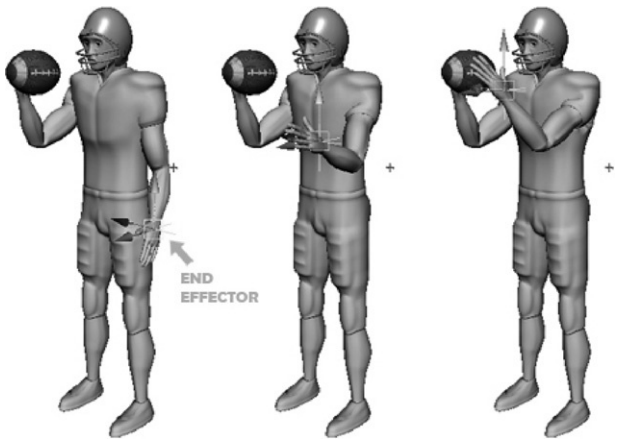


FIGURE 5.2

Sample sequence showing positioning the hand (identified as the end effector) to the desired position as a procedure solves for the required joint angles.

(Image courtesy of Domin Lee.)

5.1 Hierarchical modeling

Hierarchical modeling is the enforcement of relative location constraints among objects organized in a treelike structure. Planetary systems are one type of hierarchical model. In planetary systems, moons rotate around planets, which rotate around a sun, which moves in a galaxy. A common type of

hierarchical model used in graphics has objects that are connected end to end to form multibody jointed chains. Such hierarchies are useful for modeling animals and humans so that the joints of the limbs are manipulated to produce a figure with moving appendages. Such a figure is often referred to as *articulated* and the movement of an appendage by changing the configuration of a joint is referred to as *articulation*. Because the connectivity of the figure is built into the structure of the model, the animator does not need to make sure that the objects making up the limbs stay attached to one another.

Much of the material concerning the animation of hierarchies in computer graphics comes directly from the field of robotics (e.g., [3]). The robotics literature discusses the modeling of *manipulators*, a sequence of objects connected in a chain by *joints*. The rigid objects forming the connections between the joints are called *links*, and the free end of the chain of alternating joints and links is called the *end effector*. The local coordinate system associated with each joint is referred to as the *frame*.

Robotics is concerned with all types of joints in which two links move relative to one another. Graphics, on the other hand, is concerned primarily with *revolute* joints, in which one link rotates about a fixed point of the other link. The links are usually considered to be pinned together at this point, and the link farther down the chain rotates while the other one remains fixed—at least as far as this joint is concerned. The other type of joint is the *prismatic* joint, in which one link translates relative to another (see Figure 5.3).

The joints of Figure 5.3 allow motion in one direction and are said to have one *degree of freedom* (DOF). Structures in which more than one DOF are coincident are called *complex joints*. Complex joints include the planar joint and the ball-and-socket joint. Planar joints are those in which one link slides on the planar surface of another. Sometimes when a joint has more than one ($n > 1$) DOF, such as a ball-and-socket joint, it is modeled as a set of n one-DOF joints connected by $n - 1$ links of zero length (see Figure 5.4). Alternatively, multiple DOF joints can be modeled by using a multiple-valued parameter such as Euler angles or quaternions.

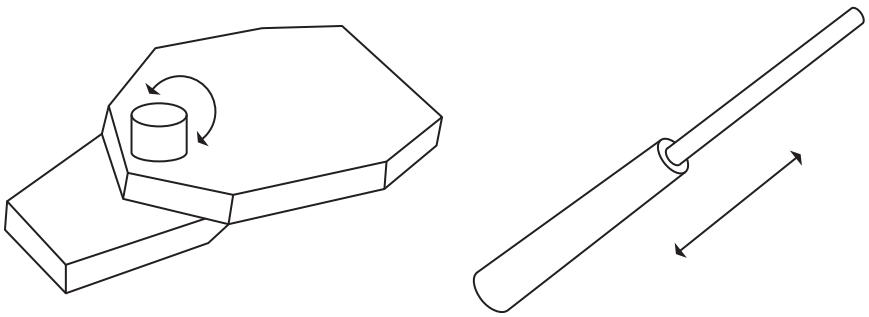


FIGURE 5.3

The two basic single DOF joints.

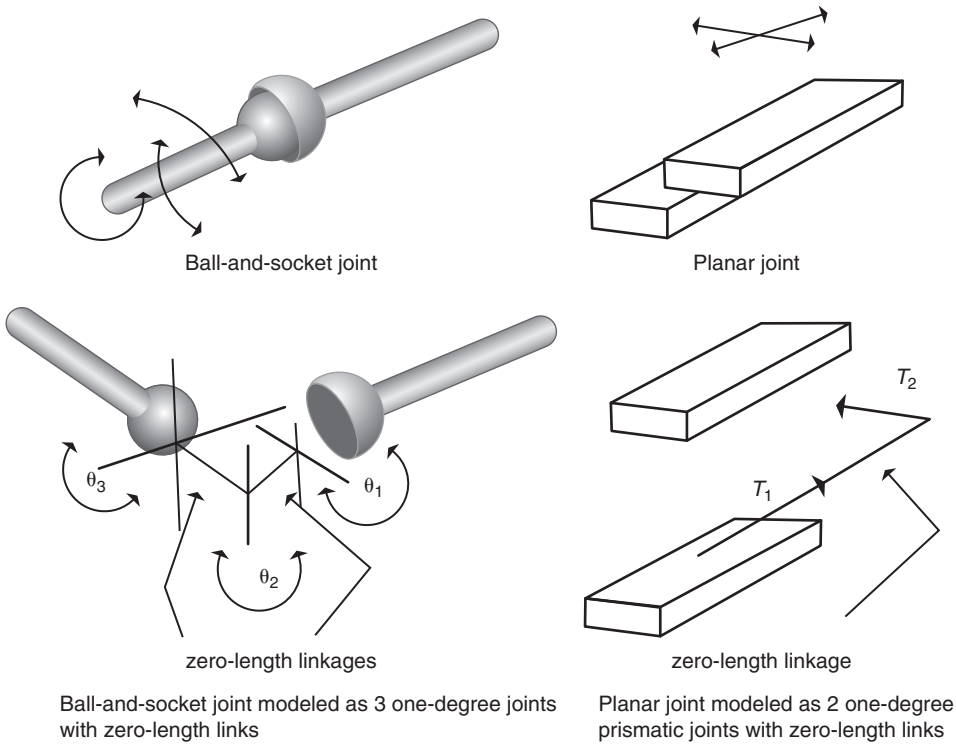


FIGURE 5.4

Modeling complex joints.

5.1.1 Data structure for hierarchical modeling

Human figures and animals are conveniently modeled as hierarchical linkages. Such linkages can be represented by a tree structure of *nodes* connected by *arcs*.[1] The highest node of the tree is the *root note*, which corresponds to the root object of the hierarchy whose position is known in the global coordinate system. The position of all other nodes of the hierarchy will be located relative to the root node. A node from which no arcs extend downward is referred to as a *leaf node*. "Higher up in the hierarchy" refers to a node that is closer to the root node. When discussing two nodes of the tree connected by an arc, the one higher up the hierarchy is referred to as the *parent node*, and the one farther down the hierarchy is referred to as the *child node*.

The mapping between the hierarchy and tree structure relates a node of the tree to information about the object part (the link) and relates an arc of the tree (the joint) to the transformation to apply to all of

[1]The connections between nodes of a tree structure are sometimes referred to as links; however, the robotics literature refers to the objects between the joints as links. To avoid overloading the term *links, arcs* is used here to refer to the connections between nodes in a tree.

the nodes below it in the hierarchy. Relating a tree arc to a figure joint may seem counterintuitive, but it is convenient because a node of the tree can have several arcs emanating from it, just as an object part may have several joints attached to it. In a discussion of a hierarchical model presented by a specific tree structure, the terms *node, object part*, and *link* are used interchangeably since all refer to the geometry to be articulated. Similarly, the terms *joint* and *arc* are used interchangeably.

In the tree structure, there is a root arc that represents a global transformation to apply to the root node (and, therefore, indirectly to all of the nodes of the tree). Changing this transformation will rigidly reposition the entire structure in the global coordinate system (see Figure 5.5).

A node of the tree structure contains the information necessary to define the object part in a position ready to be articulated. In the case of rotational joints, this means that the point of rotation on the object part is made to coincide with the origin. The object data may be defined in such a position, or there may be a transformation matrix contained in the node that, when applied to the object data, positions it so. In either case, all of the information necessary to prepare the object data for articulation is contained at the node. The node represents the transformation of the object data into a link of the hierarchical model.

Two types of transformations are associated with an arc leading to a node. One transformation rotates and translates the object into its position of attachment relative to the link one position up in the hierarchy. This defines the link's neutral position relative to its parent. The other transformation is the variable information responsible for the actual joint articulation (see Figure 5.6).

A simple example

Consider the simple, two-dimensional three-link example of Figure 5.7. In this example, there is assumed to be no transformation at any of the nodes; the data are defined in a position ready for articulation. $Link_0$, the root object, is transformed to its orientation and position in global space by T_0.

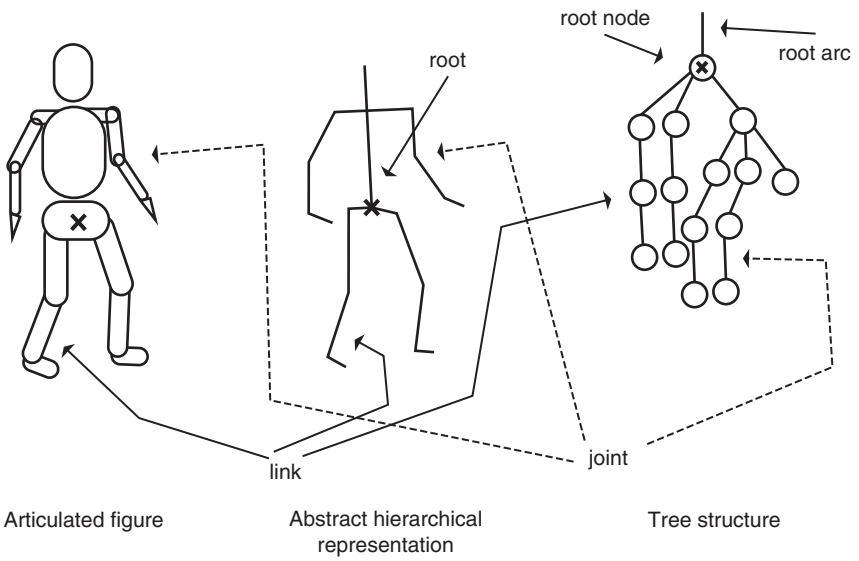


FIGURE 5.5

Example of a tree structure representing a hierarchical structure.

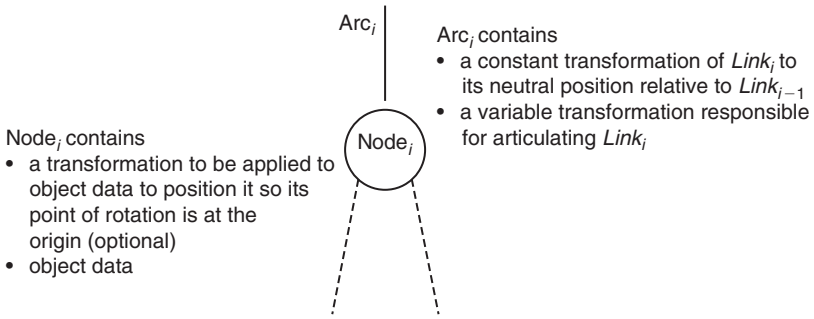


FIGURE 5.6

Arc and node definition.

Because all of the other parts of the hierarchy will be defined relative to this part, this transformation affects the entire assemblage of parts and thus will transform the position and orientation of the entire structure. This transformation can be changed over time in order to animate the position and orientation of the rigid structure. $Link_1$ is defined relative to the untransformed root object by transformation T_1. Similarly, $Link_{1.1}$ is defined relative to the untransformed $Link_1$ by transformation $T_{1.1}$. These relationships can be represented in a tree structure by associating the links with nodes and the transformations with arcs. In the example shown in Figure 5.8, the articulation transformations are not yet included in the model.

An arc in the tree representation contains a transformation that applies to the object represented by the node to which the arc immediately connects. This transformation is also applied to the rest of the linkage farther down the hierarchy. The vertices of a particular object can be transformed to their final positions by concatenating the transformations higher up the tree and applying the composite transformation matrix to the vertices. A vertex, V_0, of the root object, $Link_0$, is located in the world coordinate system, V_0' by applying the rigid transformation that affects the entire structure; see Equation 5.1.

$$V_0' = T_0 V_0 \tag{5.1}$$

A vertex of the $Link_1$ object is located in the world coordinate system by transforming it first to its location relative to $Link_0$ and then relocating it (conceptually along with $Link_0$) to world space by Equation 5.2.

$$V_1' = T_0 T_1 V_1 \tag{5.2}$$

Similarly, a vertex of the $Link_{1.1}$ object is located in world space by Equation 5.3.

$$V_{1.1}' = T_0 T_1 T_{1.1} V_{1.1} \tag{5.3}$$

Notice that as the tree is traversed farther down one of its branches, a newly encountered arc transformation is concatenated with the transformations previously encountered.

As previously discussed, when one constructs the static position of the assembly, each arc of the tree has an associated transformation that rotates and translates the link associated with the child node

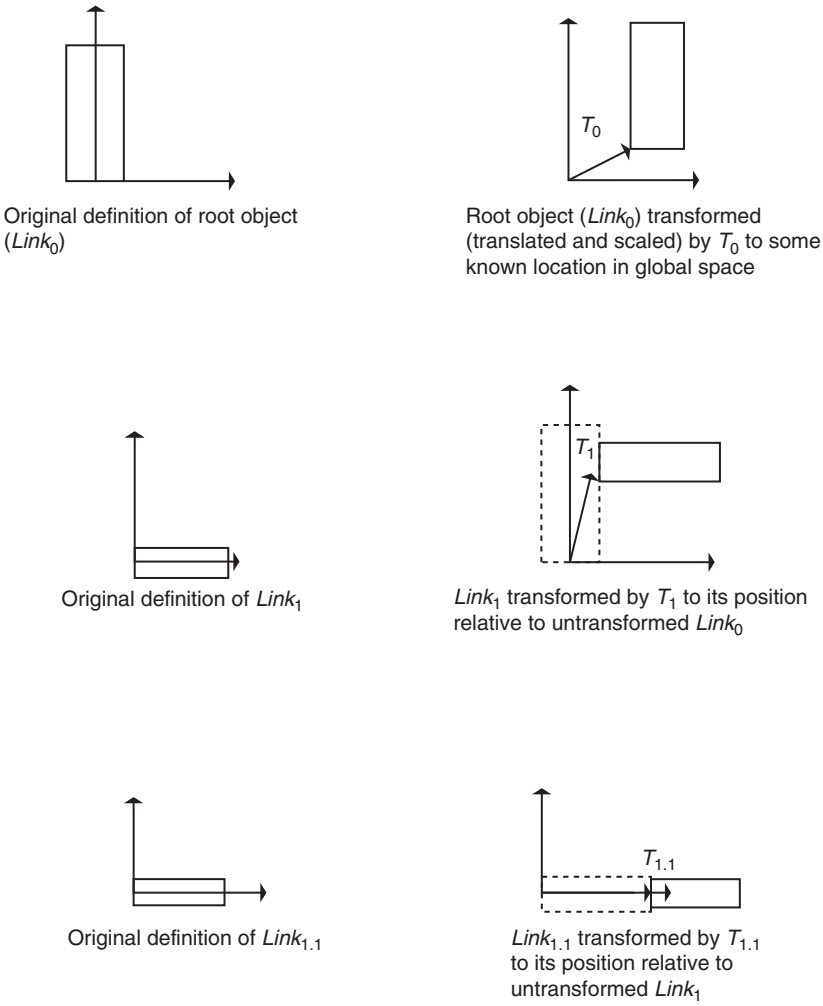


FIGURE 5.7

Example of a hierarchical model.

relative to the link associated with the parent node. To easily animate a revolute joint, also associated with the arc is a parameterized (variable) transformation, $R_i(\theta_i)$, which controls the rotation at the specified joint (see Figure 5.9). In the tree representation that implements a revolute joint, the rotation transformation is associated with the arc that precedes the node representing the link to be rotated (see Figure 5.10). The rotational transformation is applied to the link before the arc's constant transformation. If a transformation is present at the node (for preparing the data for articulation), then the rotational transformation is applied after the node transformation and before the arc's constant transformation.

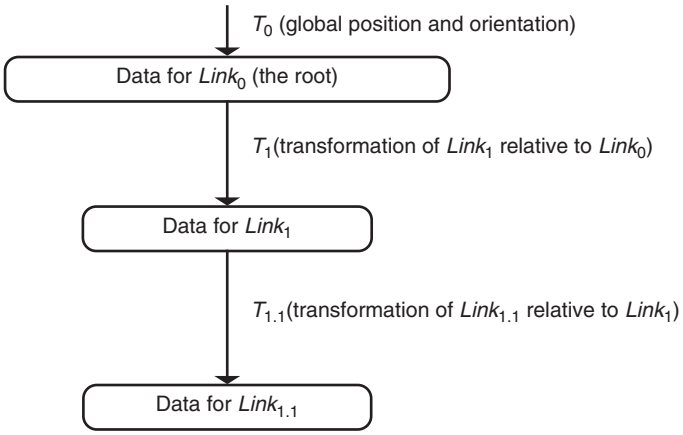


FIGURE 5.8

Example of a tree structure.

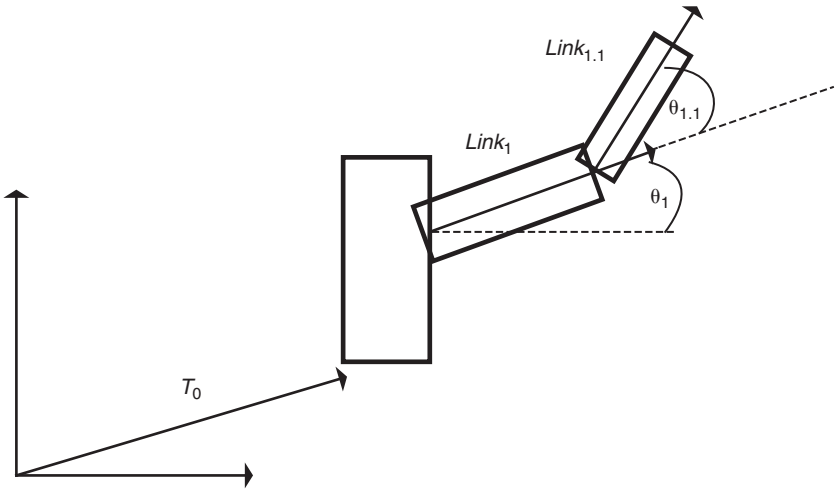


FIGURE 5.9

Variable rotations at the joints.

To locate a vertex of $Link_1$ in world space, one must first transform it via the joint rotation matrix. Once that is complete, then the rest of the transformations up the hierarchy are applied (see Eq. 5.4).

$$V_1^{'} = T_0 T_1 R_1(\theta_1) V_1 \tag{5.4}$$

A vertex of $Link_{1.1}$ is transformed similarly by compositing all of the transformations up the hierarchy to the root, as in Equation 5.5.

$$V_{1.1}^{'} = T_0 T_1 R_1(\theta_1) T_{1.1} R_{1.1}(\theta_{1.1}) V_{1.1} \tag{5.5}$$

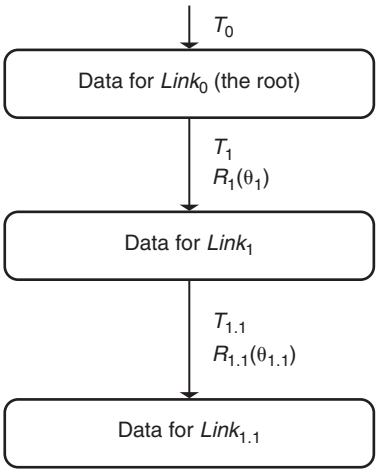


FIGURE 5.10

Hierarchy showing joint rotations.

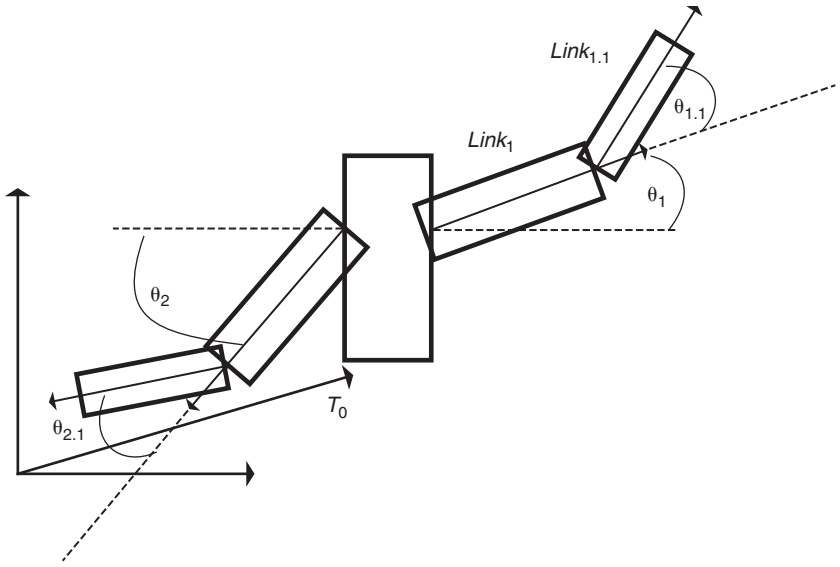


FIGURE 5.11

Hierarchy with two appendages.

In the case of a second appendage, the tree structure would reflect the bifurcations (or multiple branches if more than two). Adding another arm to our simple example results in Figure 5.11.

The corresponding tree structure would have two arcs emanating from the root node, as in Figure 5.12. Branching in the tree occurs whenever multiple appendages emanate from the same object. For example, in a simplified human figure, the root hip area (see Figure 5.5) might branch into the torso

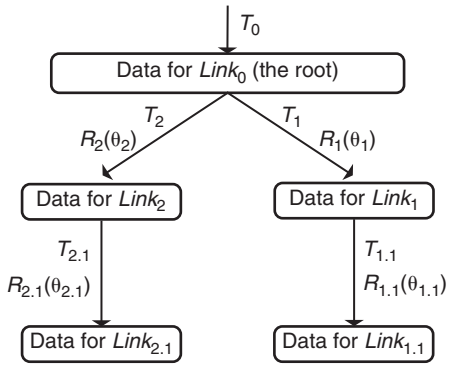


FIGURE 5.12

Tree structure corresponding to hierarchy with two appendages.

and two legs. If prismatic joints are used, the strategy is the same; the only difference is that the rotation transformation of the joint (arc) is replaced by a translation.

5.1.2 Local coordinate frames

In setting up complex hierarchies and in applying computationally involved procedures such as IK, it is convenient to be able to define points in the local coordinate system (frame) associated with a joint and to have a well-defined method for converting the coordinates of a point from one frame to another. A common use for this method is to convert points defined in the frame of a joint to the global coordinate system for display purposes. In the example above, a transformation matrix is associated with each arc to represent the transformation of a point from the local coordinate space of a child node to the local coordinate space of the parent node. Successively applying matrices farther up the hierarchy can transform a point from any position in the tree into world coordinates. The inverse of the transformation matrix can be used to transform a point from the parent's frame to the child's frame. In the three-dimensional case, 4×4 transformation matrices can be used to describe the relation of one coordinate frame to the next. Robotics has adopted a more concise and more appropriate parameterization for physical devices called the Denavit-Hartenberg notation. Although this notation is not used much in graphics anymore because it lacks flexibility appropriate for synthetic linkages, it remains an effective way to describe the physical configurations used in robotics. See Appendix B.4 for details of the Denavit-Hartenberg notation.

When implementing a kinematic linkage, especially when the linkage contains joints with multiple DOF, there are options to consider when choosing the method to implement rotational parameterization of a local coordinate frame. Part of the issue involves the user interface. What kind of information is required from the user to specify the rotational values at the joint? Users are often already comfortable with specifications such as *x*-*y*-*z* Euler angles. The other part of the issue involves the method used to implement the rotations in order to produce the required transformations. If the user specifies the rotation(s) interactively, then the operations are mapped directly to this internal representation. Quaternions are often used to actually implement the transformations in order to avoid the gimbal lock problem with Euler and fixed-angle representations.

5.2 Forward kinematics

Evaluation of a hierarchy by traversing the corresponding tree produces the figure in a position that reflects the setting of the joint parameters. Traversal follows a depth-first pattern from root to leaf node. The traversal then backtracks up the tree until an unexplored downward arc is encountered. The downward arc is then traversed, followed by backtracking up to find the next unexplored arc. This traversal continues until all nodes and arcs have been visited. Whenever an arc is followed down the tree hierarchy, its transformations are concatenated to the transformations of its parent node. Whenever an arc is traversed back up the tree to a node, the transformation of that node must be restored before traversal continues downward.

A stack of transformations is a conceptually simple way to implement the saving and restoring of transformations as arcs are followed down and then back up the tree. Immediately before an arc is traversed downward, the current composite matrix is pushed onto the stack. The arc transformation is then concatenated with the current transformation by premultiplying it with the composite matrix. Whenever an arc is traversed upward, the top of the stack is popped off of the stack and becomes the current composite transformation. (If node transformations, which prepare the data for transformation, are present, they must not be included in a matrix that gets pushed onto the stack.)

In the C-like pseudocode in Figure 5.13, each arc has associated with it the following:

- nodePtr: A pointer to a node that holds the data to be articulated by the arc.
- Lmatrix: A matrix that locates the following (child) node relative to the previous (parent) node.

```
traverse(arcPtr,matrix)
{
   ; get transformations of arc and concatenate to current matrix
   matrix = matrix*arcPtr->Lmatrix      ; concatenate location
   matrix = matrix*arcPtr->Amatrix      ; concatenate articulation
   ; process data at node
   nodePtr = arcPtr->nodePtr            ; get the node of the arc
   push(matrix)                         ; save the matrix
   matrix = matrix * nodePtr->matrix    ; ready for articulation
   articulatedData = transformData(matrix,dataPtr) ; articulate the data
   draw(articulatedData);               ; and draw it
   matrix = pop()                       ; restore matrix for node's children
   ; process children of node
   if (nodePtr->arcPtr!= NULL) {        ; if not a terminal node
      nextArcPtr = nodePtr->arcPtr      ; get first arc emanating from node
      while (nextArcPtr != NULL) {      ; while there's an arc to process
         push(matrix)                   ; save matrix at node
         traverse(nextArcPtr,matrix)    ; traverse arc
         matrix = pop()                 ; restore matrix at node
         nextArcPtr = nextArcPtr->arcPtr  ; set next child of node
      }
   }
}
```

FIGURE 5.13

Forward kinematics pseudocode.

- Amatrix: A matrix that articulates the node data; this is the matrix that is changed in order to animate (articulate) the linkage.
- arcPtr: A pointer to a sibling arc (another child of this arc's parent node); this is NULL if there are no more siblings.

Each node has associated with it the following:

- dataPtr: Data (possibly shared by other nodes) that represent the geometry of this segment of the figure.
- Tmatrix: A matrix to transform the node data into position to be articulated (e.g., put the point of rotation at the origin).
- ArcPtr: A pointer to a single child arc.

The code in Figure 5.13 uses `Push()` and `Pop()` calls that operate on a matrix stack in order to save and restore the transformation matrix.

There is a root arc that holds the global transformation for the figure and points to the root node of the figure. The hierarchy is traversed by passing a pointer to the root arc and a matrix initialized as the identity matrix to the traversal routine:

```
traverse(rootArcPtr,I)  ;  'I' is identity matrix
```

To animate the linkage, the parameters at the joints (rotation angles in our case) are manipulated. These parameters are used to construct the changeable transformation matrix associated with the tree arc.

A completely specified set of linkage parameters, which results in positioning the hierarchical figure, are called a pose. A pose is specified by a vector (the *pose vector*) consisting of one parameter for each DOF.

In a simple animation, a user may determine a key position interactively or by specifying numeric values and then interpolate the joint values between key positions. Specifying all of the joint parameter values for key positions is called forward kinematics and is an easy way to animate the figure. Unfortunately, getting an end effector to a specific desired position by specifying joint values can be tedious for the user. Often, it is a trial-and-error process. To avoid the difficulties in having to specify all of the joint values, IK is sometimes used, in which the desired position and orientation of the end effector are given and the internal joint values are calculated automatically.

5.3 Inverse kinematics

In IK, the desired position and possibly orientation of the end effector are given by the user along with the initial pose vector. From this, the joint values required to attain that configuration are calculated, giving the final pose vector. The problem can have zero, one, or more solutions. If there are so many constraints on the configuration that no solution exists, the system is called *overconstrained*. If there are relatively few constraints on the system and there are many solutions to the problem posed, then it is *underconstrained*. The *reachable workspace* is that volume which the end effector can reach. The *dexterous workspace* is the volume that the end effector can reach in any orientation.

Once the joint values are calculated, the figure can be animated by interpolating from the initial pose vector values to the final pose vector values calculated by IK. However, for large differences between initial and final pose vectors, this does not provide precise control over the path that the end effector follows. Alternatively, a series of intermediate end effector positions (and possibly

orientations) can first be calculated and each one of these then used as input to an IK problem. In this way, the path the end effector takes is prescribed by the animator.

If the mechanism is simple enough, then the joint values (the pose vector) required to produce the final desired configuration can be calculated analytically. Given an initial pose vector and the final pose vector, intermediate configurations can be formed by interpolation of the values in the pose vectors, thus animating the mechanism from its initial configuration to the final one. However, if the mechanism is too complicated for analytic solutions, then an incremental approach can be used that employs a matrix of values (the *Jacobian*) that relates changes in the values of the joint parameters to changes in the end effector position and orientation. The end effector is iteratively nudged until the final configuration is attained within a given tolerance. In addition to the Jacobian, there are other incremental formulations that can be used to effect inverse kinematic solutions.

5.3.1 Solving a simple system by analysis

For sufficiently simple mechanisms, the joint values of a final desired position can be determined analytically by inspecting the geometry of the linkage. Consider a simple two-link arm in two-dimensional space with two rotational DOFs. Link lengths are L_1 and L_2 for the first and second link, respectively. If a position is fixed for the base of the arm at the first joint, any position beyond $|L_1 - L_2|$ units from the base of the link and within $L_1 + L_2$ of the base can be reached (see Figure 5.14).

Assume for now (without loss of generality) that the base is at the origin. In a simple IK problem, the user gives the (X, Y) coordinate of the desired position for the end effector. The joint angles, θ_1 and θ_2, can be determined by computing the distance from the base to the goal and using the law of cosines to compute the interior angles. Once the interior angles are computed, the rotation angles for the two links can be computed (see Figure 5.15). Of course, the first step is to make sure that the position of the goal is within the reach of the end effector; that is, $|L_1 - L_2| \leq \sqrt{X^2 + Y^2} \leq L_1 + L_2$.

In this simple scenario, there are only two solutions that will give the correct answer; the configurations are symmetric with respect to the line from (0, 0) to (X, Y). This is reflected in the equation in

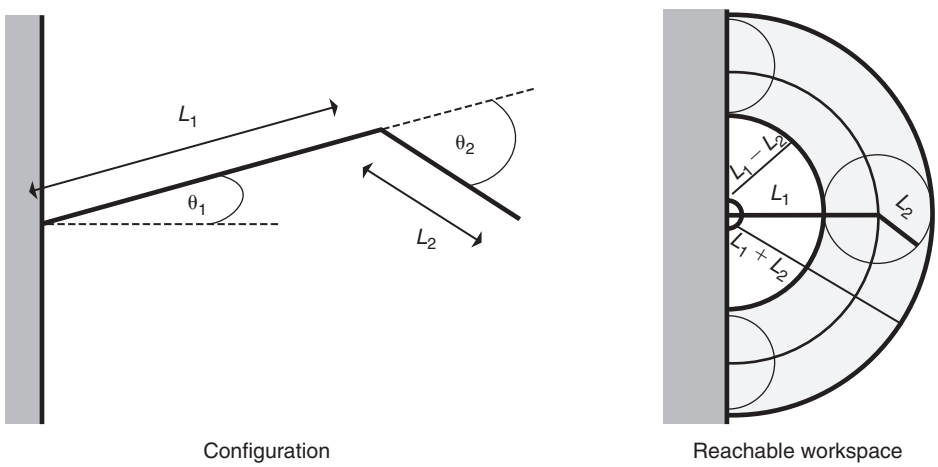
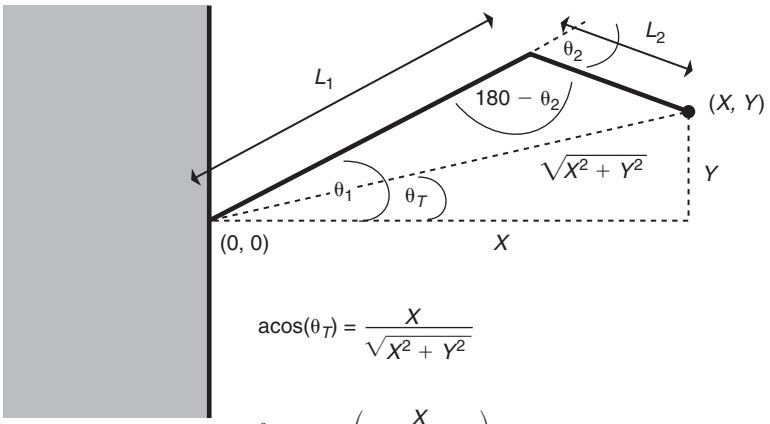


FIGURE 5.14

Simple linkage.



$$\text{acos}(\theta_T) = \frac{X}{\sqrt{X^2 + Y^2}}$$

$$\theta_T = \text{acos}\left(\frac{X}{\sqrt{X^2 + Y^2}}\right)$$

$$\cos(\theta_1 - \theta_T) = \frac{L_1^2 + X^2 + Y^2 - L_2^2}{2L_1\sqrt{X^2 + Y^2}} \qquad \text{(cosine rule)}$$

$$\theta_1 = \text{acos}\left(\frac{L_1^2 + X^2 + Y^2 - L_2^2}{2L_1\sqrt{X^2 + Y^2}}\right) + \theta_T$$

$$\cos(180 - \theta_2) = -\cos(\theta_2) = \frac{L_1^2 + L_2^2 - (X^2 + Y^2)}{2L_1L_2} \qquad \text{(cosine rule)}$$

$$\theta_2 = \text{acos}\left(\frac{L_1^2 + L_2^2 - X^2 + Y^2}{2L_1L_2}\right)$$

FIGURE 5.15

Analytic solution to a simple IK problem.

Figure 5.15; the inverse cosine is two-valued in both plus and minus theta ($\pm\theta$). However, for more complicated cases, there may be infinitely many solutions that will give the desired end effector location.

The joint values for relatively simple linkages can be solved by algebraic manipulation of the equations that describe the relationship of the end effector to the base frame. Most linkages used in robotic applications are designed to be simple enough for this analysis. However, for many cases that arise in computer animation, analytic solutions are not tractable. In such cases, iterative numeric solutions must be relied on.

5.3.2 The Jacobian

Many mechanisms of interest to computer animation are too complex to allow an analytic solution. For these, the motion can be incrementally constructed. At each time step, a computation is performed that determines a way to change each joint angle in order to direct the current position and orientation of the end effector toward the desired configuration. There are several methods used to compute the change in joint angle, but most involve forming the matrix of partial derivatives called the *Jacobian*.

To explain the Jacobian from a strictly mathematical point of view, consider the six arbitrary functions of Equation 5.6, each of which is a function of six independent variables. Given specific values for the input variables, x_i, each of the output variables, y_i, can be computed by its respective function.

$$\begin{aligned} y_1 &= f_1(x_1, x_2, x_3, x_4, x_5, x_6) \\ y_2 &= f_2(x_1, x_2, x_3, x_4, x_5, x_6) \\ y_3 &= f_3(x_1, x_2, x_3, x_4, x_5, x_6) \\ y_4 &= f_4(x_1, x_2, x_3, x_4, x_5, x_6) \\ y_5 &= f_5(x_1, x_2, x_3, x_4, x_5, x_6) \\ y_6 &= f_6(x_1, x_2, x_3, x_4, x_5, x_6) \end{aligned} \tag{5.6}$$

These equations can also be used to describe the change in the output variables relative to the change in the input variables. The differentials of y_i can be written in terms of the differentials of x_i using the chain rule. This generates Equation 5.7.

$$dy_i = \frac{\partial f_i}{\partial x_1} dx_1 + \frac{\partial f_i}{\partial x_2} dx_2 + \frac{\partial f_i}{\partial x_3} dx_3 + \frac{\partial f_i}{\partial x_4} dx_4 + \frac{\partial f_i}{\partial x_5} dx_5 + \frac{\partial f_i}{\partial x_6} dx_6 \tag{5.7}$$

Equations 5.6 and 5.7 can be put in vector notation, producing Equations 5.8 and 5.9, respectively.

$$dY = \frac{\partial F}{\partial X} dX \tag{5.8}$$

$$dY = \frac{\partial F}{\partial X} dX \tag{5.9}$$

A matrix of partial derivatives, $\frac{\partial F}{\partial X}$, is called the *Jacobian* and is a function of the current values of x_i. The Jacobian can be thought of as mapping the velocities of X to the velocities of Y (Eq. 5.10).

$$\dot{Y} = J(X)\dot{X} \tag{5.10}$$

At any point in time, the Jacobian is a function of the x_i. At the next instant of time, X has changed and so has the transformation represented by the Jacobian.

When one applies the Jacobian to a linked appendage, the input variables, x_i, become the joint values and the output variables, y_i, become the end effector position and orientation (in some suitable representation such as x-y-z fixed angles).

$$Y = \begin{bmatrix} p_x & p_y & p_z & \alpha_x & \alpha_y & \alpha_z \end{bmatrix}^T \tag{5.11}$$

In this case, the Jacobian relates the velocities of the joint angles, $\dot{\theta}$, to the velocities of the end effector position and orientation, $\dot{Y}$ (Eq. 5.12).

$$V = \dot{Y} = J(\theta)\dot{\theta} \tag{5.12}$$

V is the vector of linear and rotational velocities and represents the desired change in the end effector. The desired change will be based on the difference between its current position/orientation to that specified by the goal configuration. These velocities are vectors in three-space, so each has an x, y, and z component (Eq. 5.13).

$$V = \begin{bmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T \tag{5.13}$$

$\dot{\theta}$ is a vector of joint value velocities, or the changes to the joint parameters, which are the unknowns of the equation (Eq. 5.14).

$$\dot{\theta} = \begin{bmatrix} \dot{\theta}_1 & \dot{\theta}_2 & \dot{\theta}_3 & \dots & \dot{\theta}_n \end{bmatrix}^T \tag{5.14}$$

J, the Jacobian, is a matrix that relates the two and is a function of the current pose (Eq. 5.15).

$$J = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \dots & \frac{\partial p_x}{\partial \theta_n} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} & \dots & \frac{\partial p_y}{\partial \theta_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial \alpha_z}{\partial \theta_1} & \frac{\partial \alpha_z}{\partial \theta_2} & \dots & \frac{\partial \alpha_z}{\partial \theta_n} \end{bmatrix} \tag{5.15}$$

Each term of the Jacobian relates the change of a specific joint to a specific change in the end effector. For a revolute joint, the rotational change in the end effector, ω, is merely the velocity of the joint angle about the axis of revolution at the joint under consideration. For a prismatic joint, the end effector orientation is unaffected by the joint articulation. For a rotational joint, the linear change in the end effector is the cross-product of the axis of revolution and a vector from the joint to the end effector. The rotation at a rotational joint induces an instantaneous linear direction of travel at the end effector. For a prismatic joint, the linear change is identical to the change at the joint (see Figure 5.16).

The desired angular and linear velocities are computed by finding the difference between the current configuration of the end effector and the desired configuration. The angular and linear velocities of the end effector induced by the rotation at a specific joint are determined by the computations shown in Figure 5.16. The problem is to determine the best linear combination of velocities induced by the various joints that would result in the desired velocities of the end effector. The Jacobian is formed by posing the problem in matrix form.

In assembling the Jacobian, it is important to make sure that all of the coordinate values are in the same coordinate system. It is often the case that joint-specific information is given in the coordinate

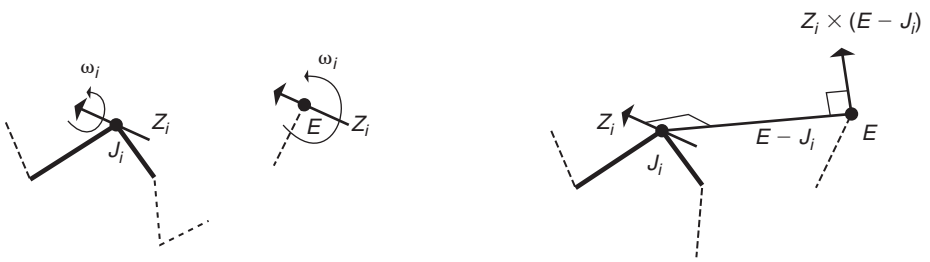


FIGURE 5.16

Angular and linear velocities induced by joint axis rotation.

system local to that joint. In forming the Jacobian matrix, this information must be converted into some common coordinate system such as the global inertial (world) coordinate system or the end effector coordinate system. Various methods have been developed for computing the Jacobian based on attaining maximum computational efficiency given the required information in local coordinate systems, but all methods produce the derivative matrix in a common coordinate system.

A simple example

Consider the simple three-revolute-joint, planar manipulator of Figure 5.17. In this example, the objective is to move the end effector, E, to the goal position, G. The orientation of the end effector is of no concern in this example. The axis of rotation of each joint is perpendicular to the figure, coming out of the paper. The effect of an incremental rotation, g_i, of each joint can be determined by the cross-product of the joint axis and the vector from the joint to the end effector, V_i (Figure 5.18), and form the columns of the Jacobian. Notice that the magnitude of each g_i is a function of the distance between the locations of the joint and the end effector.

The desired change to the end effector is the difference between the current position of the end effector and the goal position (Eq. 5.16).

$$V = \begin{bmatrix} (G-E)_x \\ (G-E)_y \\ (G-E)_z \end{bmatrix} \tag{5.16}$$

A vector of the desired change in values is set equal to the Jacobian matrix (Eq. 5.17) multiplied by a vector of the unknown values, which are the changes to the joint angles.

$$J = \begin{bmatrix} ((0,0,1)\times E)_x & ((0,0,1)\times (E-P_1)_x & ((0,0,1)\times (E-P_2)_x \\ ((0,0,1)\times E)_y & ((0,0,1)\times (E-P_1)_y & ((0,0,1)\times (E-P_2)_y \\ ((0,0,1)\times E)_z & ((0,0,1)\times (E-P_1)_z & ((0,0,1)\times (E-P_2)_z \end{bmatrix} \tag{5.17}$$

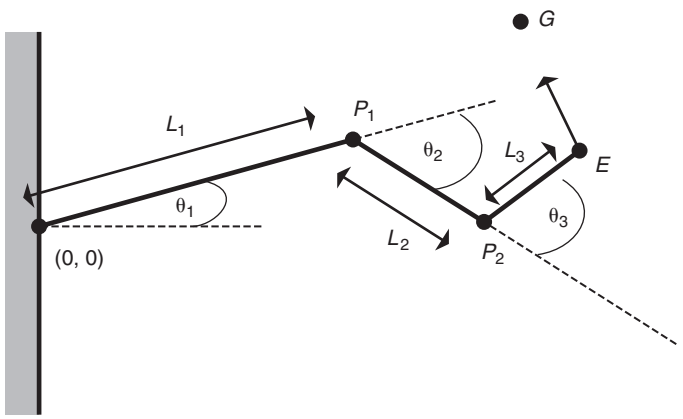


FIGURE 5.17

Planar three-joint manipulator.

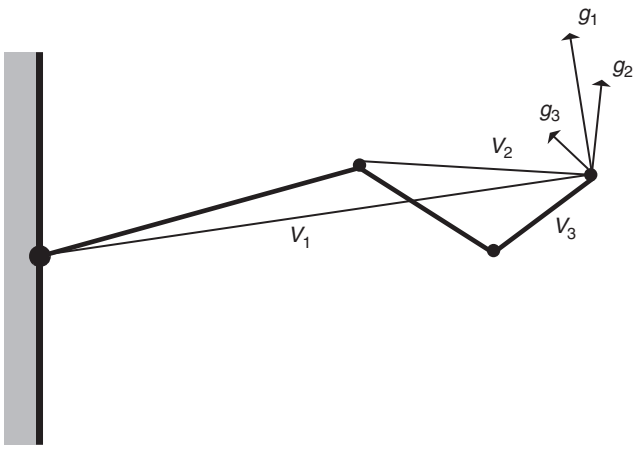


FIGURE 5.18

Instantaneous changes in position induced by joint angle rotations.

5.3.3 Numeric solutions to IK

Solution using the inverse Jacobian

Once the Jacobian has been computed, an equation in the form of Equation 5.18 is to be solved. In the case that J is a square matrix, the inverse of the Jacobian, J^{-1}, is used to compute the joint angle velocities given the end effector velocities (Eq. 5.19).

$$V = J\dot{\theta} \tag{5.18}$$

$$J^{-1}V = \dot{\theta} \tag{5.19}$$

If the inverse of the Jacobian (J^{-1}) does not exist, then the system is said to be singular for the given joint angles. A singularity occurs when a linear combination of the joint angle velocities cannot be formed to produce the desired end effector velocities. As a simple example of such a situation, consider a fully extended planar arm with a goal position somewhere on the forearm (see Figure 5.19). In such a case, a change in each joint angle would produce a vector perpendicular to the desired direction. Obviously, no linear combination of these vectors could produce the desired motion vector. Unfortunately, all of the singularities of a system cannot be determined simply by visually inspecting the possible geometric configurations of the linkage.

Additionally, a configuration that is only close to being a singularity can still present major problems. If the joints of the linkage in Figure 5.19 are slightly perturbed, then the configuration is not singular. However, in order to form a linear combination of the resulting instantaneous change vectors, very large values must be used. This results in large impulses near areas of singularities. These must be clamped to more reasonable values. Even then, numerical error can result in unpredictable motion.

Problems with singularities can be reduced if the manipulator is redundant—when there are more DOF than there are constraints to be satisfied. In this case, the Jacobian is not a square matrix and, potentially, there are an infinite number of solutions to the IK problem. Because the Jacobian is not square, a conventional inverse does not exist. However, if the columns of J are linearly independent

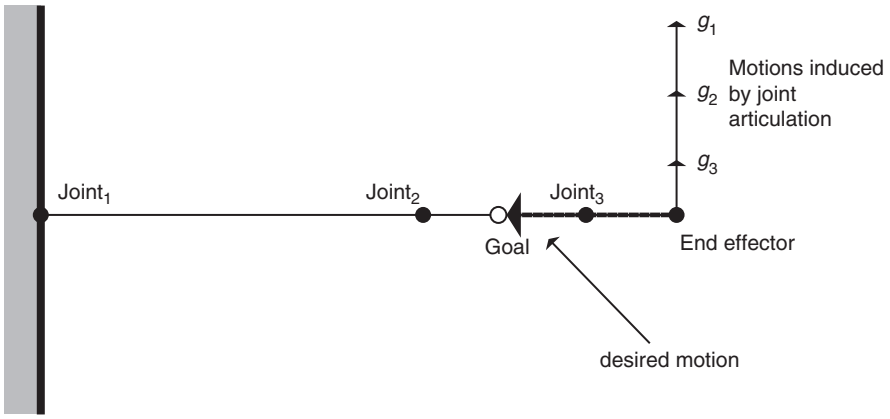


FIGURE 5.19

Simple example of a singular configuration.

(i.e., J has full column rank), then $(J^T\mathrm{J})^{-1}$ exists and instead, the *pseudoinverse*, J^+, can be used as in Equation 5.20. This approach is viable because a matrix multiplied by its own transpose will be a square matrix whose inverse may exist.

$$\begin{aligned} V &= J\dot{\theta} \\ J^T V &= J^T J\dot{\theta} \\ (J^T J)^{-1} J^T V &= (J^T J)^{-1} J^T \dot{\theta} \\ J^+ V &= \dot{\theta} \end{aligned} \tag{5.20}$$

If the rows of J are linearly independent, then JJ_T is invertible and the equation for the pseudoinverse is $J^+ = J^T(JJ^T)^{-1}$. The pseudoinverse maps the desired velocities of the end effector to the required changes of the joint angles. After making the substitutions shown in Equation 5.21, LU decomposition can be used to solve Equation 5.22 for β. This can then be substituted into Equation 5.23 to solve for $\dot{\theta}$.

$$\begin{aligned} J^+ V &= \dot{\theta} \\ J^T (JJ^T)^{-1} V &= \dot{\theta} \end{aligned} \tag{5.21}$$

$$\begin{aligned} \beta &= (JJ^T)^{-1} V \\ (JJ^T)\beta &= V \end{aligned} \tag{5.22}$$

$$J^T \beta = \dot{\theta} \tag{5.23}$$

Simple Euler integration can be used at this point to update the joint angles. The Jacobian has changed at the next time step, so the computation must be performed again and another step taken. This process repeats until the end effector reaches the goal configuration within some acceptable (i.e., user-defined) tolerance.

It is important to remember that the Jacobian is only valid for the instantaneous configuration for which it is formed. That is, as soon as the configuration of the linkage changes, the Jacobian ceases to accurately describe the relationship between changes in joint angles and changes in end effector position and

orientation. This means that if too big a step is taken in joint angle space, the end effector may not appear to travel in the direction of the goal. If this appears to happen during an animation sequence, then taking smaller steps in joint angle space and thus recalculating the Jacobian more often may be in order.

As an example, consider a two-dimensional three-joint linkage with link lengths of 15, 10, and 5. Using an initial pose vector of $\{\pi/8, \pi/4, \pi/4\}$ and a goal position of $\{-20, 5\}$, a 21-frame sequence is calculated[2] for linearly interpolated intermediate goal positions for the end effector. Figure 5.20 shows frames 0, 5, 10, 15, and 20 of the sequence. Notice the path of the end effector (the end point of the third link) travels in approximately a straight line to the goal position.

Even the underconstrained case still has problems with singularities. A proposed solution to such bad behavior is the damped least-squares approach [1] [2]. Referring to Equation 5.24, a user-supplied parameter is used to add in a term that reduces the sensitivity of the pseudoinverse.

$$\dot{\theta} = J^T(JJ^T + \lambda^2 I)^{-1}V \tag{5.24}$$

It is argued that this form behaves better in the neighborhood of singularities at the expense of rate of convergence to a solution. Figure 5.21 shows solutions using the pseudoinverse of the Jacobian

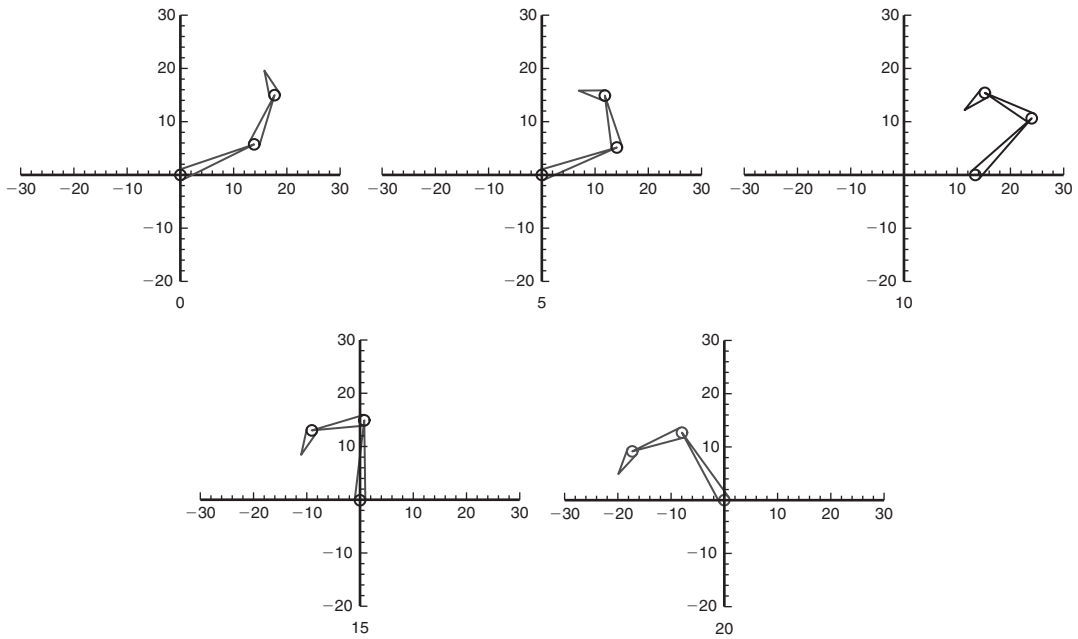


FIGURE 5.20

IK solution for a two-dimensional three-link armature of lengths 15, 10, and 5. The initial pose is $\{\pi/8, \pi/4, \pi/4\}$ and goal is $\{-20, 5\}$. Panels show frames 0, 5, 10, 15, and 20 of a 21-frame sequence in which the end effector tracks a linearly interpolated path to goal. (a) The pseudoinverse of Jacobian solution without damped least-squares. (b) The pseudoinverse of Jacobian solution with damped least-squares.

[2]The examples showing solutions of numeric IK were generated using the *LinearSolve* function of Mathematica.

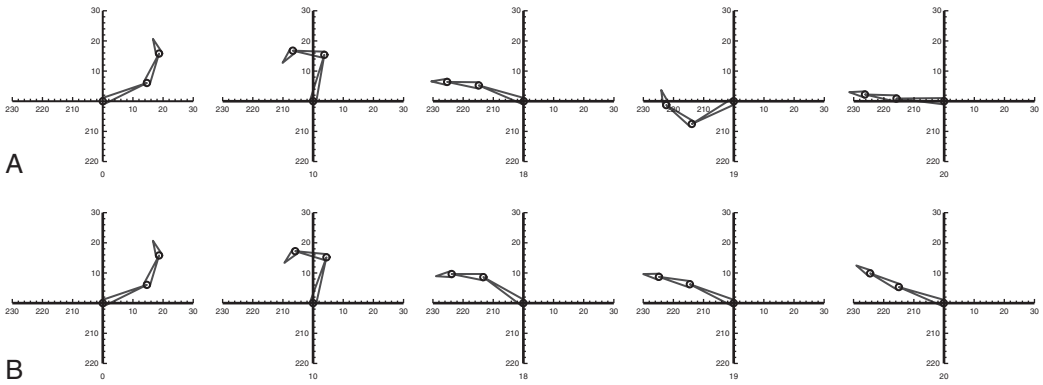


FIGURE 5.21

Example comparing the pseudoinverse of the Jacobian solution to inverse kinematics (a) without damped least-squares and (b) with damped least squares for a two-dimensional three-link armature of lengths 15, 10, and 5. The initial pose is $\{\pi/8, \pi/4, \pi/4\}$ and goal is $\{-35, 5\}$. Panels show frames 0, 10, 18, 19, and 20 of a 21-frame sequence in which the end effector tracks a linearly interpolated path to goal.

without, and then with, damping for the linkage used in the previous example. However, in this case, the goal is at $\{-35, 5\}$—out of reach of the end effector. This example demonstrates better behavior when damped as the linkage approaches the limits of its reach.

Adding more control

The pseudoinverse computes one of many possible solutions. It minimizes joint angle rates. The configurations produced, however, do not necessarily correspond to what might be considered natural poses. To better control the kinematic model such as encouraging joint angle constraints, a control expression can be added to the pseudoinverse Jacobian solution. The control expression is used to solve for control angle rates with certain attributes. The added control expression, because of its form, contributes nothing to the desired end effector velocities.[3] The form for the control expression is shown in Equation 5.25.

$$\dot{\theta} = (J^{+}J - I)z \tag{5.25}$$

In Equation 5.26 it is shown that a change to pose parameters in the form of Equation 5.25 does not add anything to the velocities. As a consequence, the control expression can be added to the pseudoinverse Jacobian solution without changing the given velocities to be satisfied [4].

$$\begin{aligned} V &= J\dot{\theta} \\ V &= J(J^{+}J - I)z \\ V &= (JJ^{+}J - I)z \\ V &= 0z \\ V &= 0 \end{aligned} \tag{5.26}$$

[3]The columns of the control term matrix are in the null space of the Jacobian and, therefore, do not affect the end effector position [1].

To bias the solution toward specific joint angles, such as the middle angle between joint limits, z is defined as in Equation 5.27, where θ_i are the current joint angles, θ_{ci} are the desired joint angles, and a_i are the joint gains. This does not enforce joint limits as hard constraints, but the solution can be biased toward the middle values so that violating the joint limits is less probable.

$$z = \alpha_i(\theta_i - \theta_{ci})^2 \tag{5.27}$$

The desired angles and gains are input parameters. The gain indicates the relative importance of the associated desired angle; the higher the gain, the stiffer the joint.[4] If the gain for a particular joint is high, then the solution will be such that the joint angle quickly approaches the desired joint angle. The control expression is added to the conventional pseudoinverse of the Jacobian (Eq. 5.28). If all gains are zero, then the solution will reduce to the conventional pseudoinverse of the Jacobian. Equation 5.28 can be solved by rearranging terms as shown in Equation 5.29.

$$\dot{\theta} = J^+V + (J^+J - I)z \tag{5.28}$$

$$\begin{aligned}
\dot{\theta} &= J^+V + (J^+J - I)z \\
\dot{\theta} &= J^+V + J^+Jz - Iz \\
\dot{\theta} &= J^+(V + Jz) - z \\
\dot{\theta} &= J^T(JJ^T)^{-1}(V + Jz) - z \\
\dot{\theta} &= J^T\left[(JJ^T)^{-1}(V + Jz)\right] - z
\end{aligned} \tag{5.29}$$

To solve Equation 5.29, set $\beta = (JJ^T)^{-1}(V + Jz)$ so that Equation 5.30 results. Use LU decomposition to solve for β in Equation 5.31. Substitute the solution for β in Equation 5.30 to solve for $\dot{\theta}$.

$$\dot{\theta} = J^T\beta - z \tag{5.30}$$

$$V + Jz = (JJ^T)\beta \tag{5.31}$$

In Figure 5.22, the difference between using a larger gain for the second joint versus using a larger gain for the third joint is shown. Notice how the joints with increased gain are kept straighter in the corresponding sequence of frames.

Alternative Jacobian

Instead of trying to push the end effector toward the goal position, a formulation has been proposed that pulls the goal to the end effector [1]. This is implemented by simply using the goal position in place of the end effector in the pseudoinverse of the Jacobian method. Comparing Figure 5.19 to Figure 5.23 for the simple example introduced earlier, the results of this approach are similar to the standard method of using the end effector position in the calculations.

[4] *Stiffness* refers to how much something reacts to being perturbed. A stiff spring is a strong spring. A stiff joint, as used here, is a joint that has a higher resistance to being pulled away from its desired value.

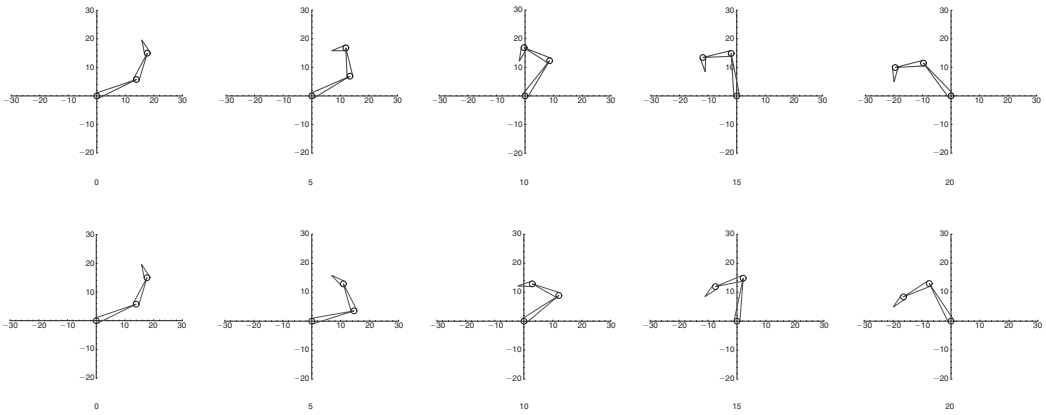


FIGURE 5.22

Inverse of the Jacobian with control term solution for a two-dimensional three-link armature of lengths 15, 10, and 5. The initial pose is {π/8, π/4, π/4} and goal is {−20, 5}. Panels show frames 0, 5, 10, 15, and 20 of a 21-frame sequence in which the end effector tracks a linearly interpolated path to goal. All joints are biased to 0; the top row uses gains of {0.1, 0.5, 0.1} and the bottom row uses gains of {0.1, 0.1, 0.5}.

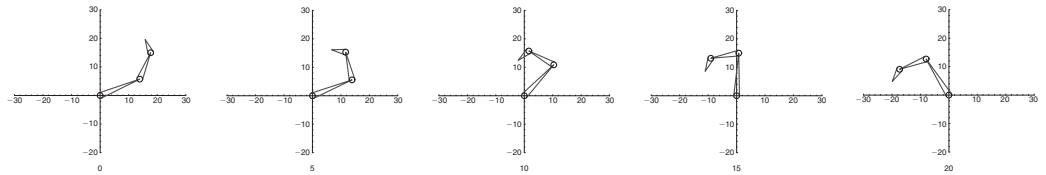


FIGURE 5.23

Inverse of the Jacobian solution formulated to pull the goal toward the end effector for a two-dimensional three-link armature of lengths 15, 10, and 5. The initial pose is {π/8, π/4, π/4} and goal is {−20, 5}. Panels show frames 0, 5, 10, 15, and 20 of a 21-frame sequence in which the end effector tracks a linearly interpolated path to goal.

Avoiding the inverse: using the transpose of the Jacobian

Solving the linear equations using the pseudoinverse of the Jacobian is essentially determining the weights needed to form the desired velocity vector from the instantaneous change vectors. An alternative way of determining the contribution of each instantaneous change vector is to form its projection onto the end effector velocity vector [5]. This entails forming the dot product between the instantaneous change vector and the velocity vector. Putting this into matrix form, the vector of joint parameter changes is formed by multiplying the transpose of the Jacobian times the velocity vector and using a scaled version of this as the joint parameter change vector.

$$\dot{\theta} = \alpha J^T V$$

Figure 5.24 shows frames from a sequence produced using the transpose of the Jacobian. Notice how the end effector tracks to the goal position while producing reasonable interior joint angles.

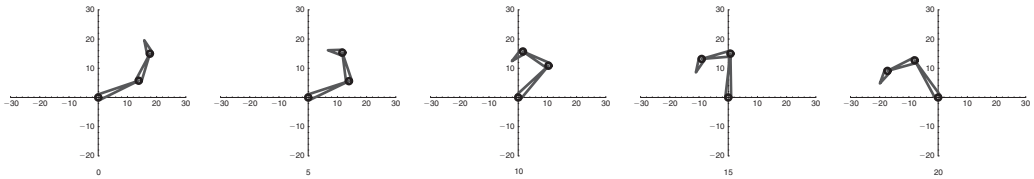


FIGURE 5.24

Transpose of the Jacobian solution for a two-dimensional three-link armature of lengths 15, 10, and 5. The initial pose is {π/8, π/4, π/4} and goal is {−20, 5}. Panels show frames 0, 5, 10, 15, and 20 of a 21-frame sequence in which the end effector tracks a linearly interpolated path to goal. The scale term used in this example is 0.1.

Using the transpose of the Jacobian avoids the expense of computing the inverse, or pseudoinverse, of the Jacobian. The main drawback is that even though a given instantaneous change vector might contribute a small component to the velocity vector, the perpendicular component may take the end effector well away from the desired direction. Also, there is an additional parameter that must be supplied by the user.

Procedurally determining the angles: cyclic coordinate descent

Instead of relying on numerical machinery to produce desired joint velocities, a more flexible, procedural approach can be taken. Cyclic coordinate descent considers each joint one at a time, sequentially from the outermost inward [5]. At each joint, an angle is chosen that best gets the end effector to the goal position.

Figure 5.25 shows frames of a sequence using cyclic coordinate descent. While the interior joint angles are different from the previous examples, they still look reasonable and the end effector tracks the solution path well.

In a variation of this approach, each instantaneous change vector is compared to the desired change vector and the joints are rank ordered in terms of their usefulness. The highest ranked joint is then changed and the process repeats. The rank ordering can be based on how recently that joint was last modified, joint limits, how close the joint gets the end effector to the goal, or any number of other measures.

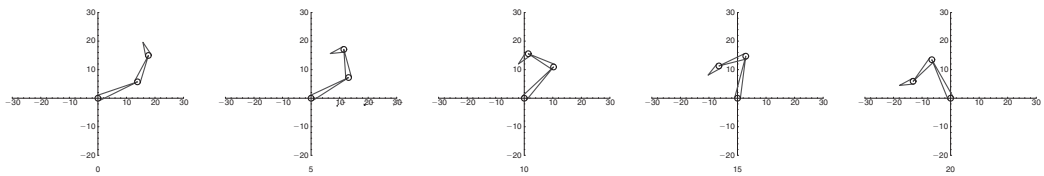


FIGURE 5.25

Cyclic coordinate descent solution for a two-dimensional three-link armature of lengths 15, 10, and 5. The initial pose is {π/8, π/4, π/4} and goal is {−20, 5}. Panels show frames 0, 5, 10, 15, and 20 of a 21-frame sequence in which the end effector tracks a linearly interpolated path to goal.

5.3.4 Summary

Various strategies for IK have been proposed in the literature. Much of this comes from robotics literature that avoids the expense of computing an inverse [1] [5]. Several strategies are presented here. The most direct solution is by analysis, but many linkages are too complex for such analysis. A numeric approach is often needed. Trade-offs among the numeric approaches include ease of implementation, possible real-time behavior, and robustness in the face of singularities. An additional approach is targeted at linkages that are specifically human-like in their DOF and heuristically solves reaching problems by trying to model human-like reaching; this is discussed in Chapter 9.

IK, on the surface, promises to solve many animation control problems. However, there is no single, obvious, and universally attractive solution to IK. There are options, however, and the various approaches do provide effective tools if used judiciously.

5.4 Chapter summary

Hierarchical models are extremely useful for enforcing certain relationships among the elements so that the animator can concentrate on just the DOF remaining. Forward kinematics gives the animator explicit control over each DOF but can become cumbersome when the animation is trying to attain a specific position or orientation of an element at the end of a hierarchical chain. IK, using the inverse or pseudoinverse of the Jacobian, allows the animator to concentrate only on the conditions at the end of such a chain but might produce undesirable configurations. Additional control expressions can be added to the pseudoinverse Jacobian solution to express a preference for solutions of a certain character. However, these are all kinematic techniques. Often, more realistic motion is desired and physically based simulations are needed.

References

[1] Buss S. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Method, Unpublished survey, http://math.ucsd.edu/∼sbuss/ResearchWeb/ikmethods/**; July 2006.**

[2] Buss S. Selectively Damped Least Squares for Inverse Kinematics. Journal of Graphics Tools 2005;10 (3):37–49.

[3] Craig J. Introduction to Robotics Mechanics and Control. New York: Addison-Wesley; 1989.

[4] Ribble E. Synthesis of Human Skeletal Motion and the Design of a Special-Purpose Processor for Real-Time Animation of Human and Animal Figure Motion. Master's thesis. Columbus, Ohio: Ohio State University; June 1982.

[5] Welman C. Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation. M.S. Thesis. Simon Frasier University; 1993.