Chapter 4

Kinematic Linkages

Chapter 4

Introduction

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 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, and human figure animation. This chapter is concerned with animating objects whose motion is relative to another object.

Chapter 4

Introduction

This chapter is concerned with animating objects whose motion is relative to another object, especially when there is a sequence of objects whose motion is relative to another object, especially when there is a sequence of objects where each object's motion can be easily 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, **linkages**.

Chapter 4

Introduction

The topic of this chapter is 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 the animator must specify rotation parameters at joints connected by rigid links.

Chapter 4

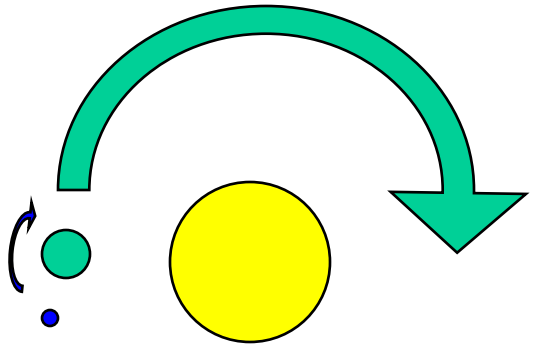
Introduction

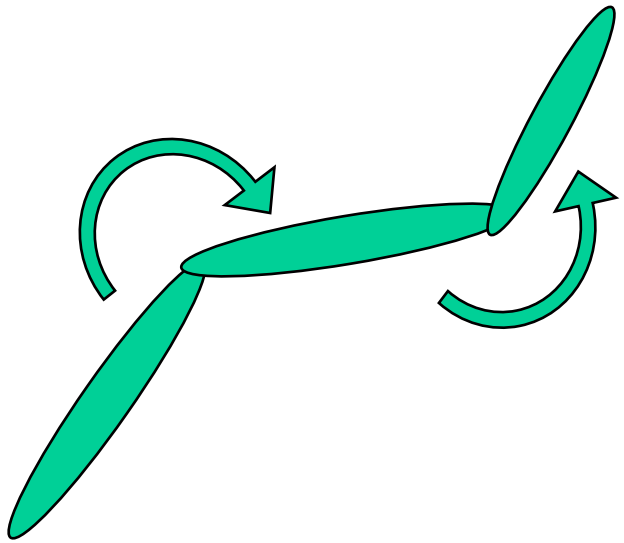
The two approaches to positioning such a hierarchy are known as forward kinematics, in which the animator must specify rotation parameters at joints, and inverse kinematics, in which the animator specifies the desired position of the hand, for example, and the system solves for the joint angles that satisfy that desire.

Chapter 4

Hierarchical Modeling

Relative motion → Parent-child relationship
Simplifies motion specification





Constrains motion → Reduces dimensionality

Chapter 4

Modeling & animating hierarchies

3 aspects

1. Linkages & Joints – the relationships
2. Data structure – how to represent such a hierarchy
3. Converting local coordinate frames into global space

Chapter 4

Some terms

Joint – allowed relative motion & parameters
Joint Limits – limit on valid joint angle values
Link – object involved in relative motion
Linkage – entire joint-link hierarchy
Armature – same as linkage
End effector – most distant link in linkage
Articulation variable – parameter of motion associated with joint
Pose – configuration of linkage using given set of joint angles
Pose vector – complete set of joint angles for linkage

Arc – of a tree data structure – corresponds to a joint
Node – of a tree data structure – corresponds to a link

Chapter 4

Use of hierarchies in animation

Forward Kinematics (FK)

animator specifies values of articulation variables

→ global transform for each linkage is computed

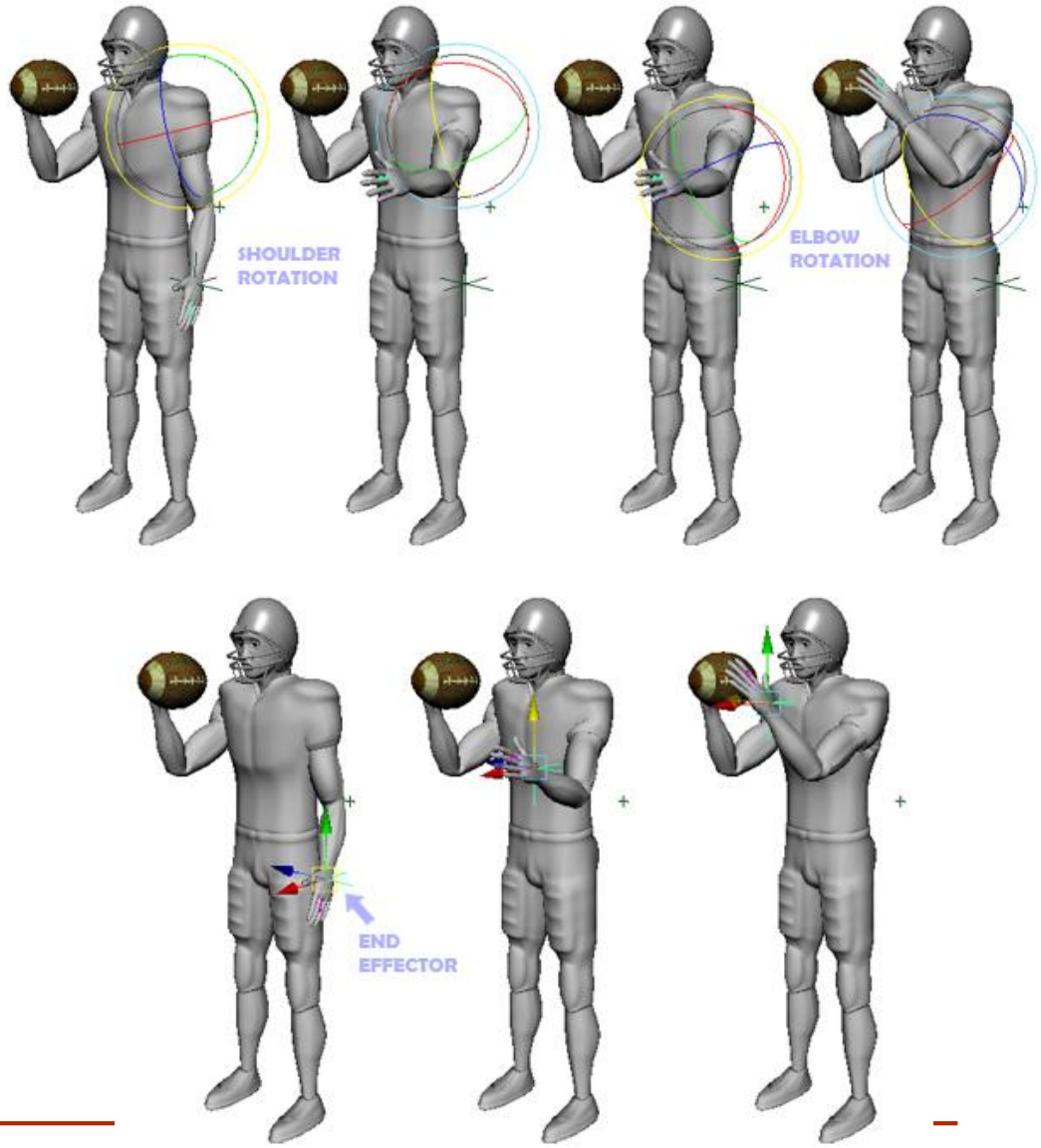
Inverse Kinematics (IK)

animator specifies final desired global transform for end effector (and possibly other linkages)

→ Values of articulation variables are computed

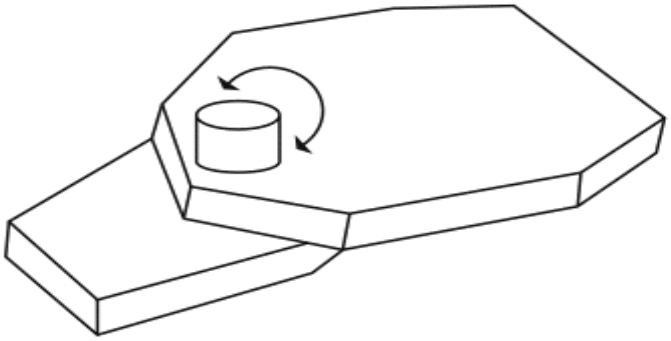
Chapter 4

Forward & Inverse Kinematics

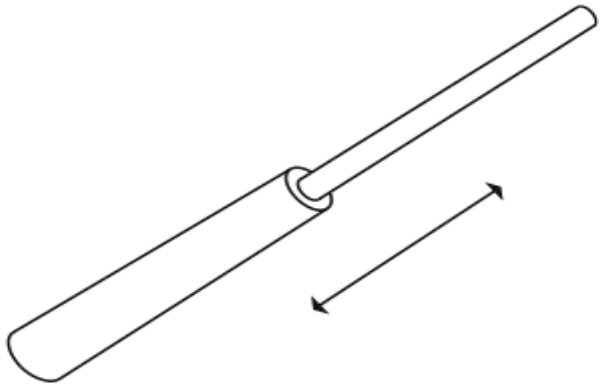


Chapter 4

Joints – relative movement



Revolute joint



Prismatic joint

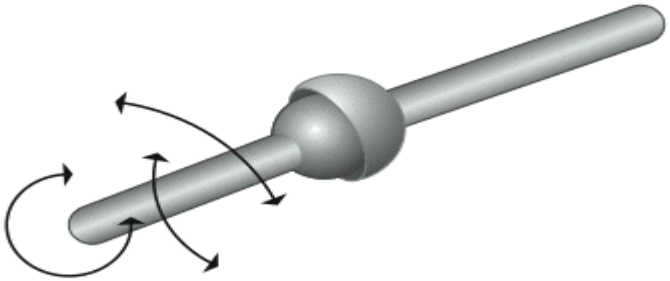
Chapter 4

Joints

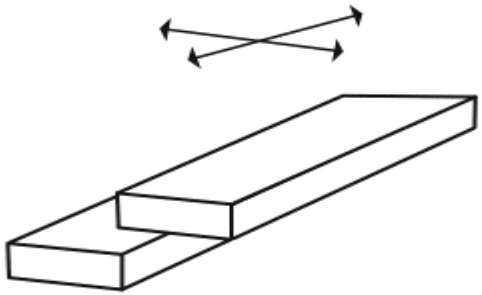
The joints from the previous slide allow motion in one direction and are said to have one **degree-of-freedom** (DOF). Structures in which more than one DOF are coincident are called complex joints. Complex joints include the planar joint and the ball-and-socket joint. Planar joints are those in which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

Chapter 4

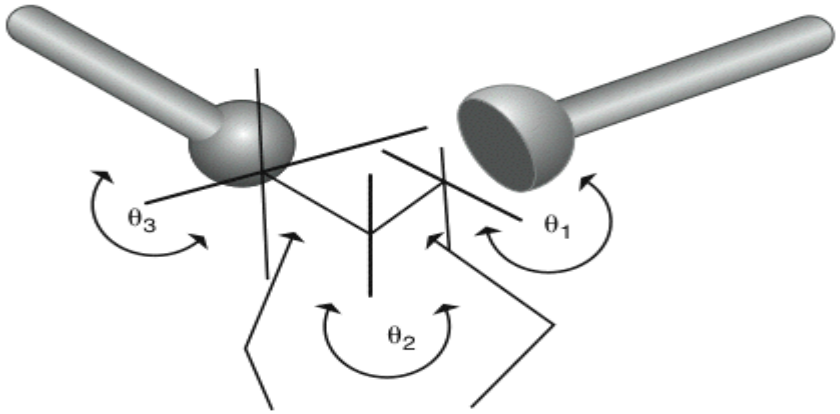
Complex Joints



Ball-and-socket joint

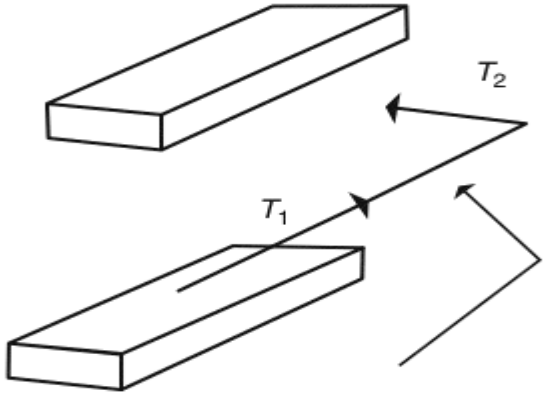


Planar joint



zero-length linkages

Ball-and-socket joint modeled as 3 one-degree joints with zero-length links



zero-length linkage

Planar joint modeled as 2 one-degree prismatic joints with zero-length links

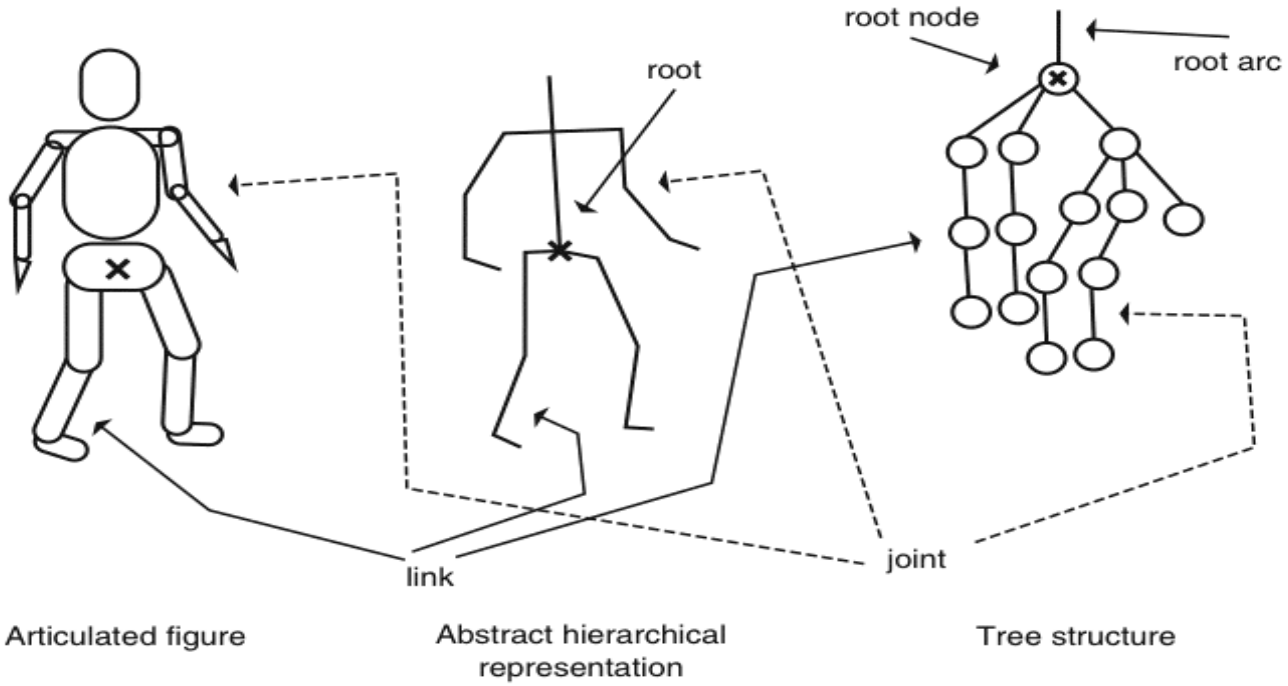
Chapter 4

Hierarchical Structure

Human figures and animals are conveniently modeled as hierarchical linkages. Such linkages can be represented by a tree structure of **nodes** connected by **arcs**. The highest node of the tree is the **root node**, 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**. 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 as the **child node**.

Chapter 4

Hierarchical structure



Chapter 4

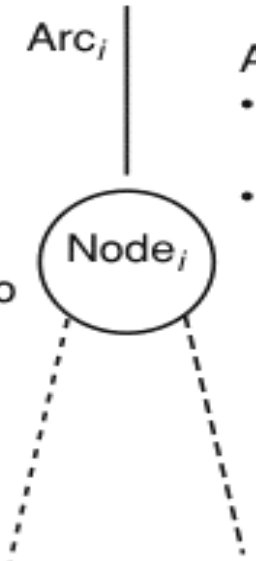
Tree structure

A node contains the information necessary to define the object part in a position ready to be articulated. It 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.

Chapter 4

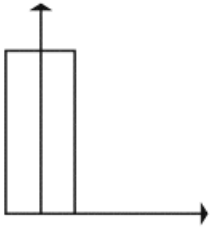
Tree structure



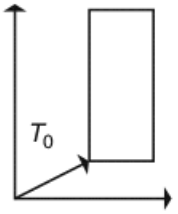
Node$_i$ contains
- a transformation to be applied to object data to position it so its point of rotation is at the origin (optional)
- object data

Arc$_i$ contains
- a constant transformation of Link$_i$ to its neutral position relative to Link$_{i-1}$.
- a variable transformation responsible for articulating Link$_i$

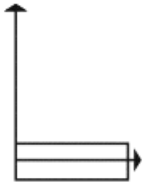
Tree structure



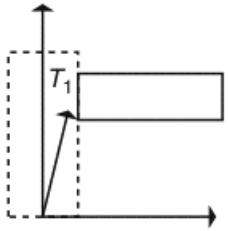
Original definition of root object ($Link_0$)



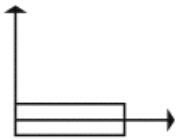
Root object ($Link_0$) transformed (translated and scaled) by T_0 to some known location in global space



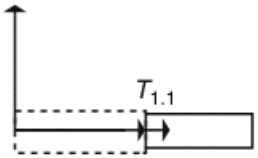
Original definition of $Link_1$



$Link_1$ transformed by T_1 to its position relative to untransformed $Link_0$



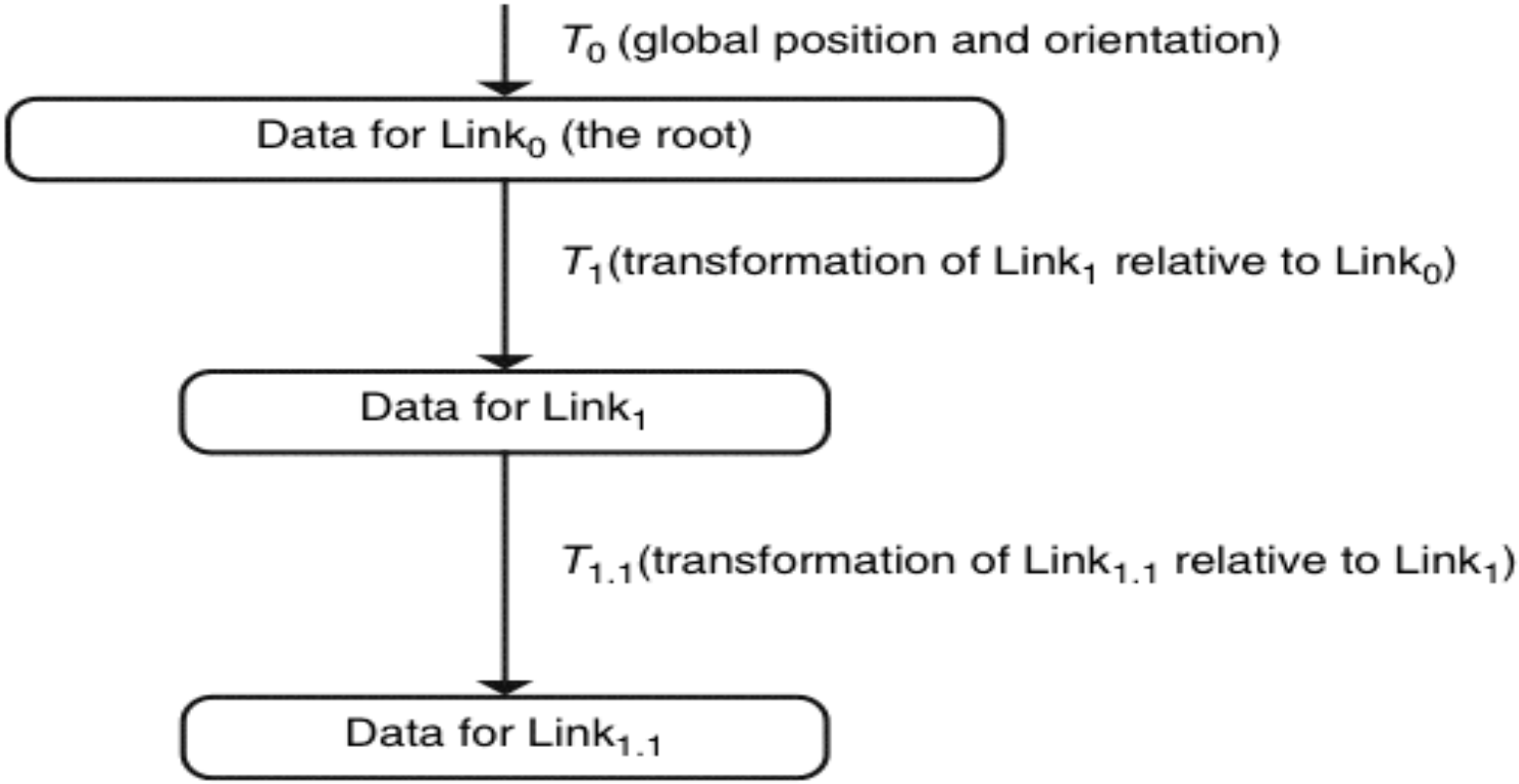
Original definition of $Link_{1.1}$



$Link_{1.1}$ transformed by $T_{1.1}$ to its position relative to untransformed $Link_1$

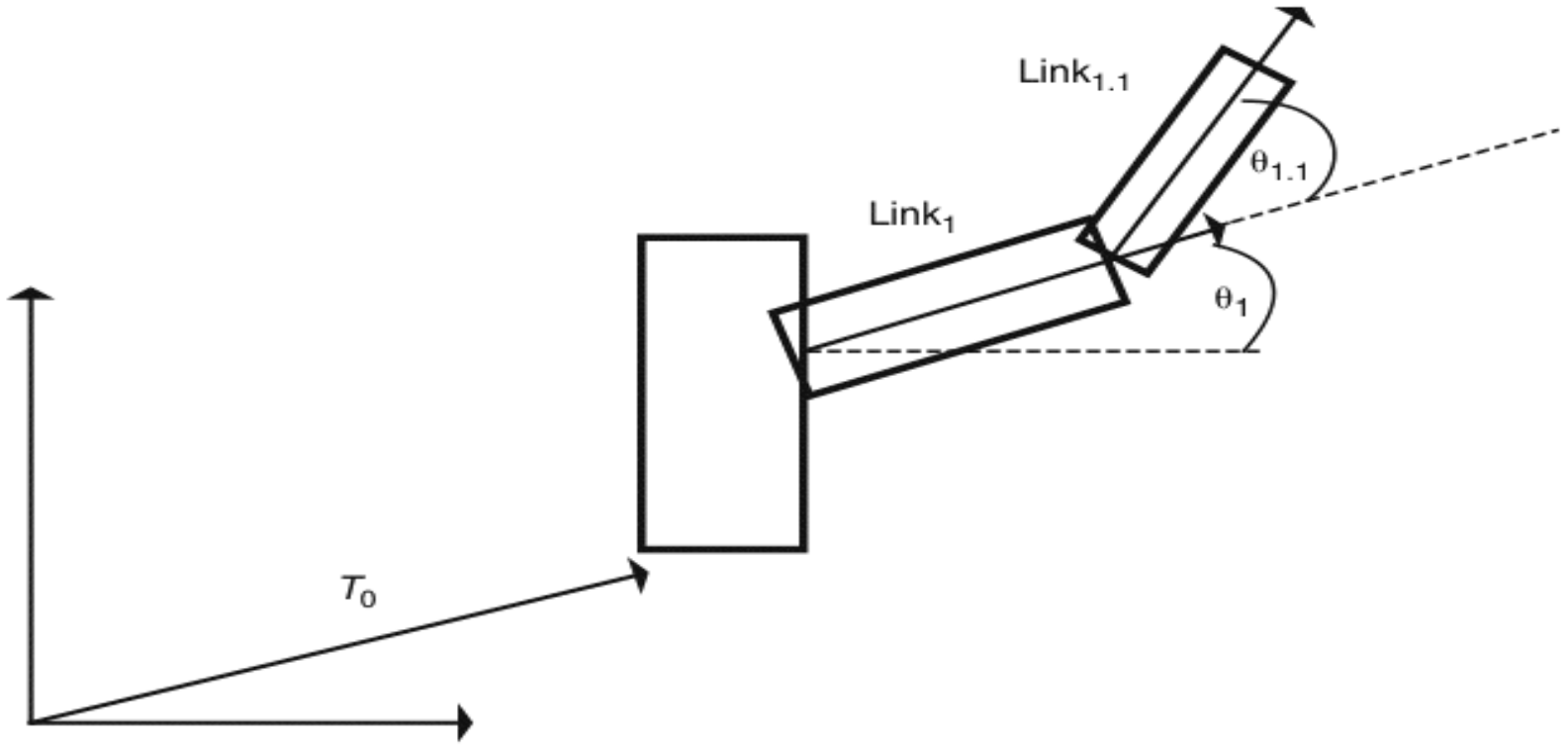
Chapter 4

Tree structure



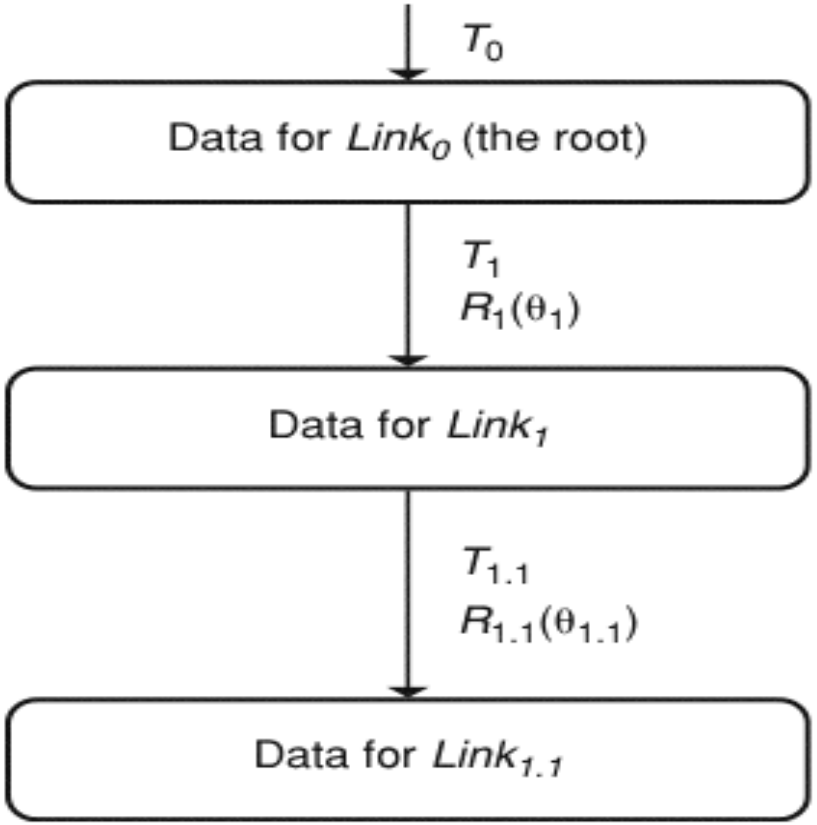
Chapter 4

Relative movement



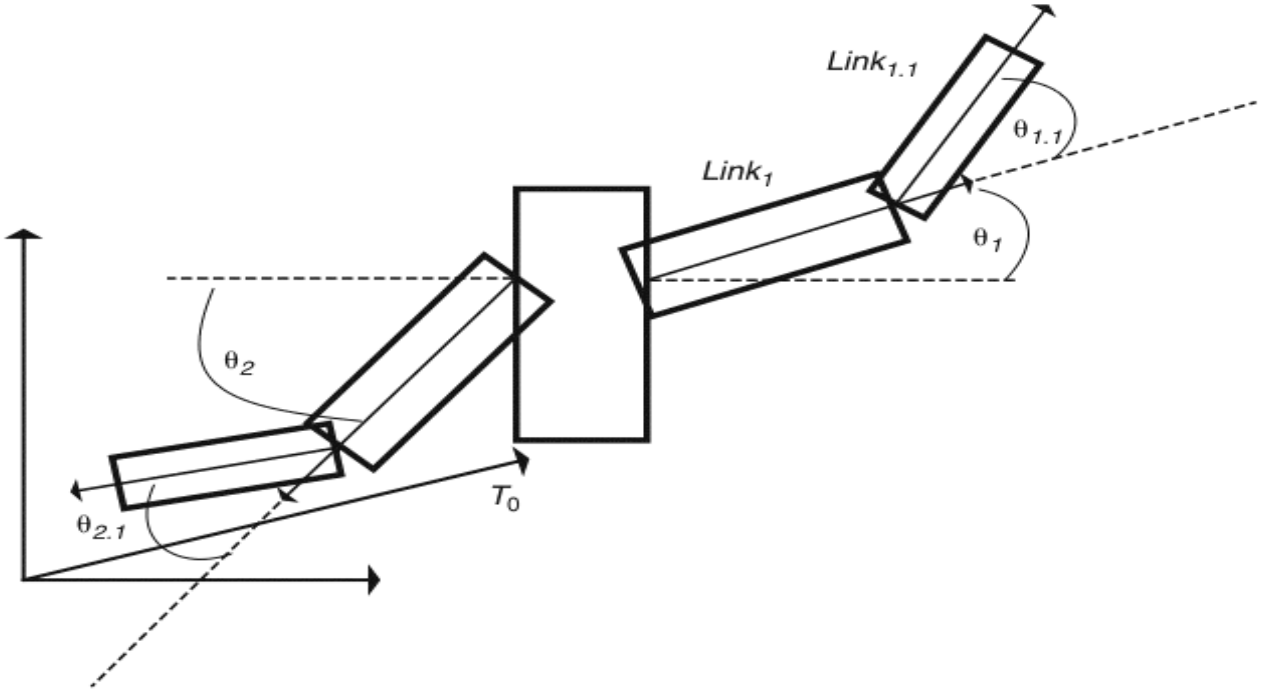
Chapter 4

Relative movement



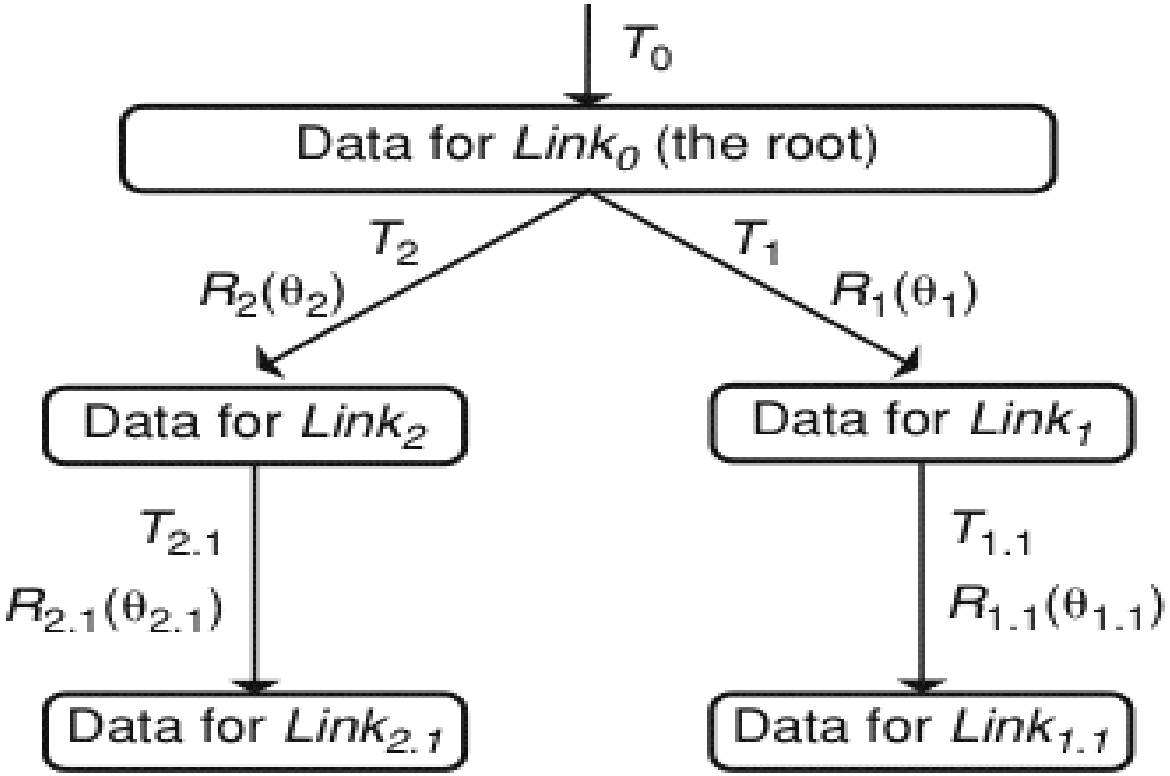
Chapter 4

Tree structure



Chapter 4

Tree structure



Chapter 4

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 in a recursive fashion. Whenever an arc is followed down the tree hierarchy, its transformations are concatenated to the transformations of its parent node. Whenever we move back up to a node due to the recursion, 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.

Chapter 4

Forward Kinematics

In C-like pseudo-code, 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
- Amatrx: 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

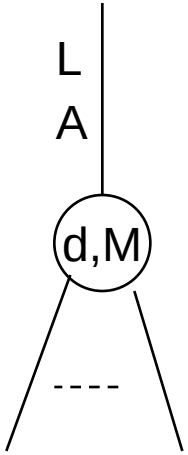
Chapter 4

Forward Kinematics

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

Tree traversal



NOTE:
Node points to first child
Each child points to sibling
Last sibling points to NULL

```
traverse (arcPtr,matrix)
{
        // concatenate arc matrices
        matrix = matrix*arcPtr->Lmatrix
        matrix = matrix*arcPtr->Amatrix;

        // get node and transform data
        nodePtr=acrPtr->nodePtr
        push (matrix)
        matrix = matrix * nodePtr->matrix
        aData = transformData(matrix,dataPtr)
        draw(aData)
        matrix = pop();

        // process children
        If (nodePtr->arc != NULL) {
                nextArcPtr = nodePtr->arc
                while (nextArcPtr != NULL) {
                        push(matrix)
                        traverse(nextArcPtr,matrix)
                        matrix = pop()
                        nextArcPtr = nextArcPtr->arc
                }
        }
}
```

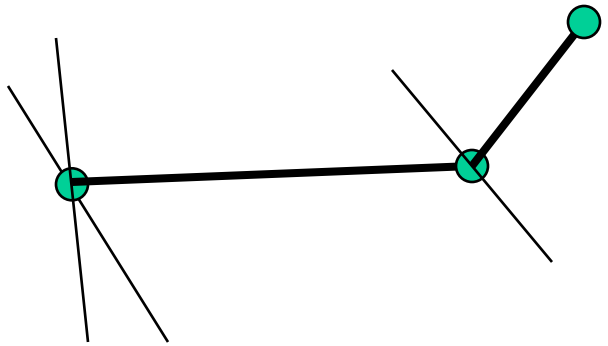
Chapter 4

OpenGL Single linkage

```
glPushMatrix();
For (i=0; i<NUMDOFS; i++) {
        glRotatef(a[i],axis[i][0], axis[i][1], axis[i][2]);
        if (linkLen[i] != 0.0) {
                draw_linkage(linkLen[i]);
                glTranslatef(0.0,linkLen[i],0.0);
        }
}
glPopMatrix();
```

OpenGL concatenates matrices

A[i] – joint angle
Axis[i] – joint axis
linkLen[i] – length of link



Chapter 4

Forward Kinematics

Example video.

Chapter 4

Inverse Kinematics

Introductory video.

Chapter 4

Inverse Kinematics

In inverse kinematics, 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, then it is **underconstraint**. The **reachable workspace** is that volume which the end effector can reach. The **dextrous workspace** is the volume that the end effector can reach in any orientation.

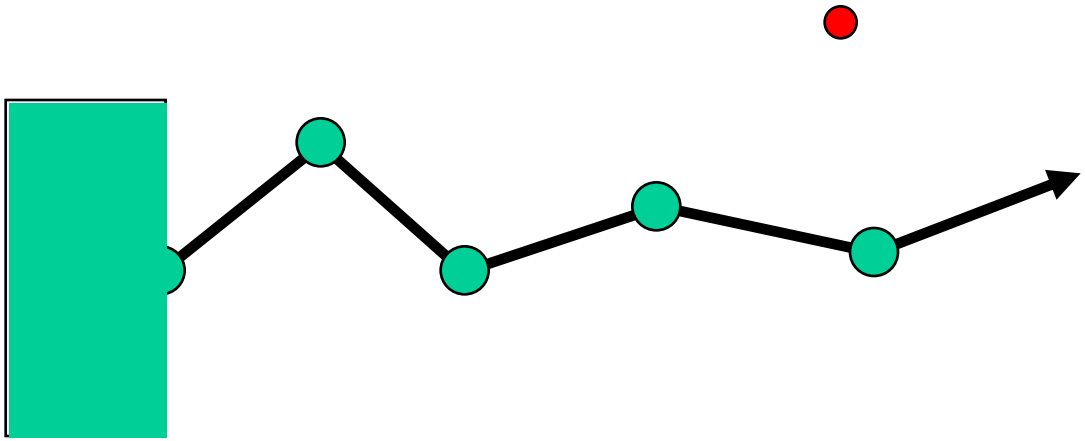
Chapter 4

Inverse kinematics

Given goal position (and orientation) for end effector

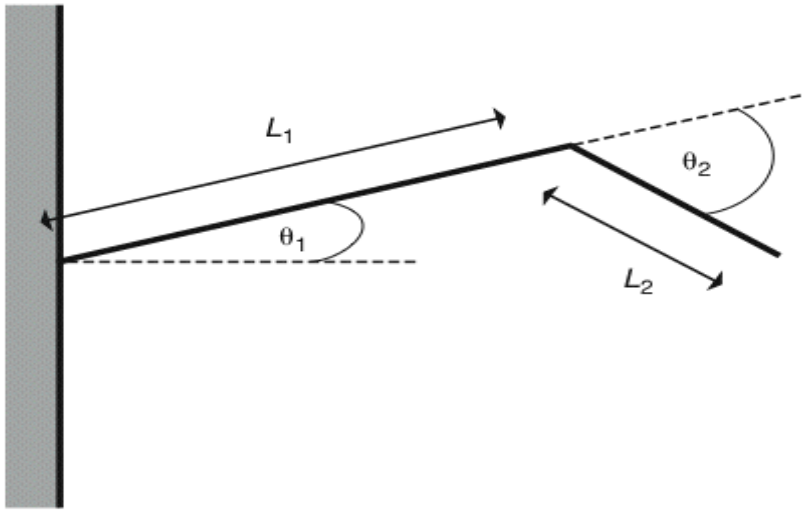
Compute internal joint angles

If simple enough => analytic solution
Else => numeric iterative solution

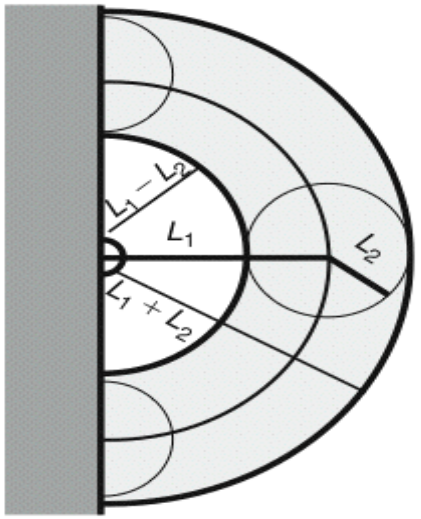


Chapter 4

Inverse kinematics - spaces



Configuration

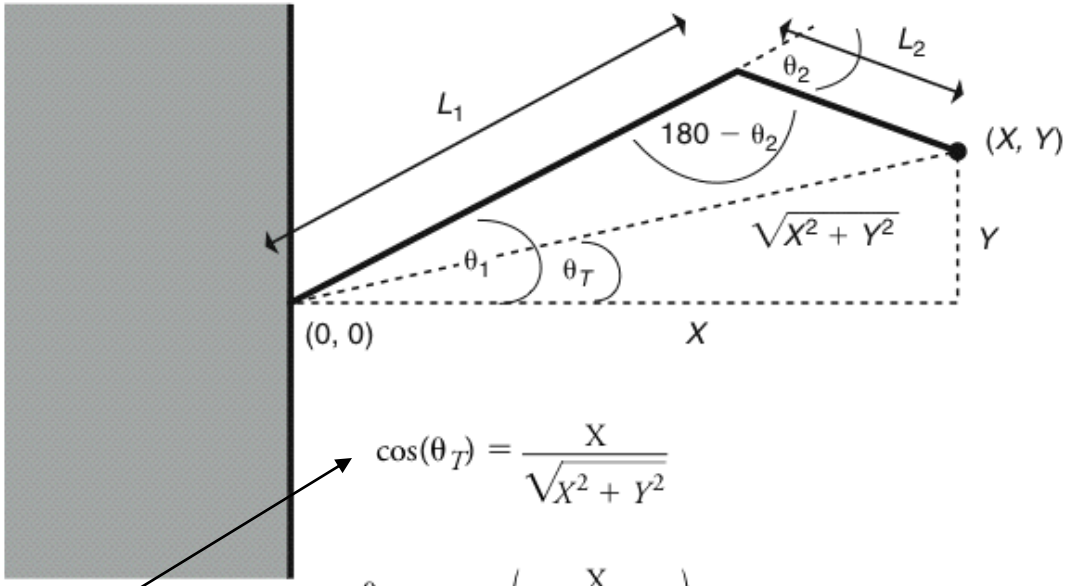


Reachable workspace

Configuration space
Reachable workspace
Dextrous workspace

Chapte

Analytic inverse kinematics



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

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

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

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

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

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

Note: typos in the book

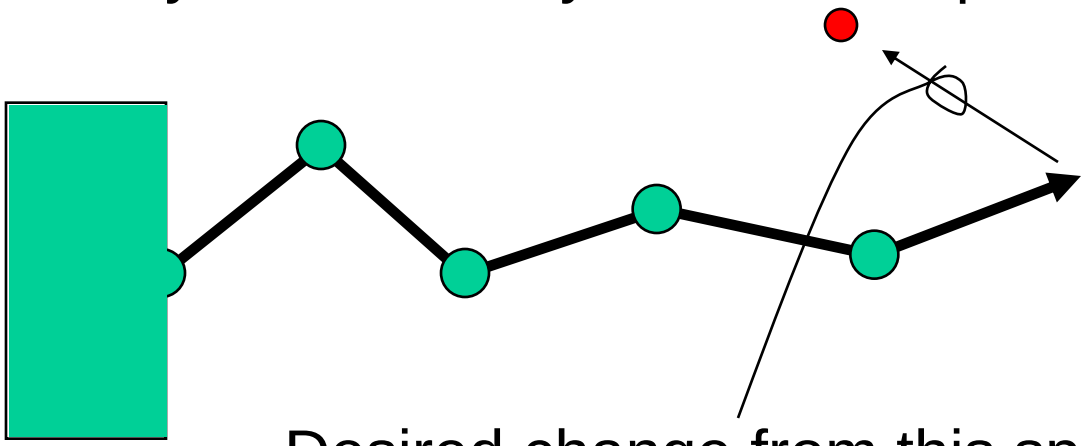
Chapter 4

IK - numeric

If linkage is too complex to solve analytically

E.g., human arm is typically modeled as 3-1-3 or 3-2-2 linkage

Solve iteratively – numerically solve for step toward goal



Desired change from this specific pose

Compute set of changes to the pose to effect that change

Chapter 4

Inverse Kinematics

For those problems that are too complex to find an analytical solution, 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**.

Chapter 4

IK math notation

$$y_1 = f_1(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_2 = f_2(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_3 = f_3(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_4 = f_4(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_5 = f_5(x_1, x_2, x_3, x_4, x_5, x_6)$$
$$y_6 = f_6(x_1, x_2, x_3, x_4, x_5, x_6)$$

$$Y = F(X)$$

Chapter 4

IK – math notation

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:

$$\partial y_i = \frac{\partial f_i}{\partial x_1}\partial x_1 + \frac{\partial f_i}{\partial x_2}\partial x_2 + \frac{\partial f_i}{\partial x_3}\partial x_3 + \frac{\partial f_i}{\partial x_4}\partial x_4 + \frac{\partial f_i}{\partial x_5}\partial x_5 + \frac{\partial f_i}{\partial x_6}\partial x_6$$

$$\partial Y = \frac{\partial F}{\partial X}\partial X$$

Chapter 4

Inverse Kinematics - Jacobian

$$\partial_Y = \frac{\partial_F}{\partial_X} \partial_X$$

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

Desired motion of end effector

Unknown change in articulation variables

The *Jacobian* is the matrix relating the two: it's a function of current avar (*articulation variables)* values

Chapter 4

Inverse Kinematics - Jacobian

$$V = J(\theta)\dot{\theta}$$

$$V = [v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T \qquad \theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]^T$$

Change in position (linear velocities)

Change in orientation (rotational velocities)

$$J = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \cdots & \frac{\partial p_x}{\partial \theta_6} \\ \frac{\partial p_y}{\partial \theta_1} & \cdots & & \\ \cdots & & & \\ \frac{\partial \alpha_z}{\partial \theta_1} & & & \frac{\partial \alpha_z}{\partial \theta_6} \end{bmatrix}$$

Chapter 4

Inverse Kinematics

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.

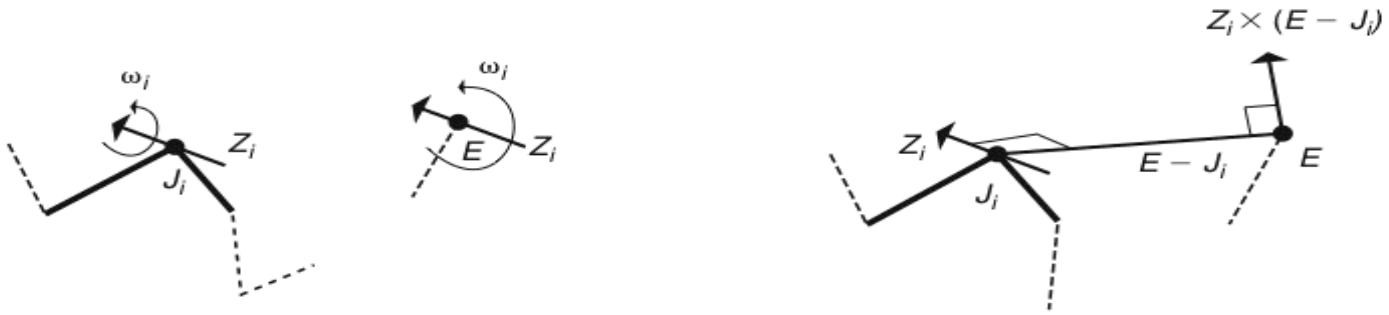
Chapter 4

Inverse Kinematics

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 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.

Chapter 4

IK – computing the Jacobian



a) Angular velocity, ω_i

b) Linear velocity, $Z_i \times (E - J_i)$

E — end effector
J_i — i th joint
Z_i — i th joint axis
ω_i — angular velocity of i th joint

Change in orientation

Change in position

Chapter 4

Inverse Kinematics

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 on the previous slide. 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.

Chapter 4

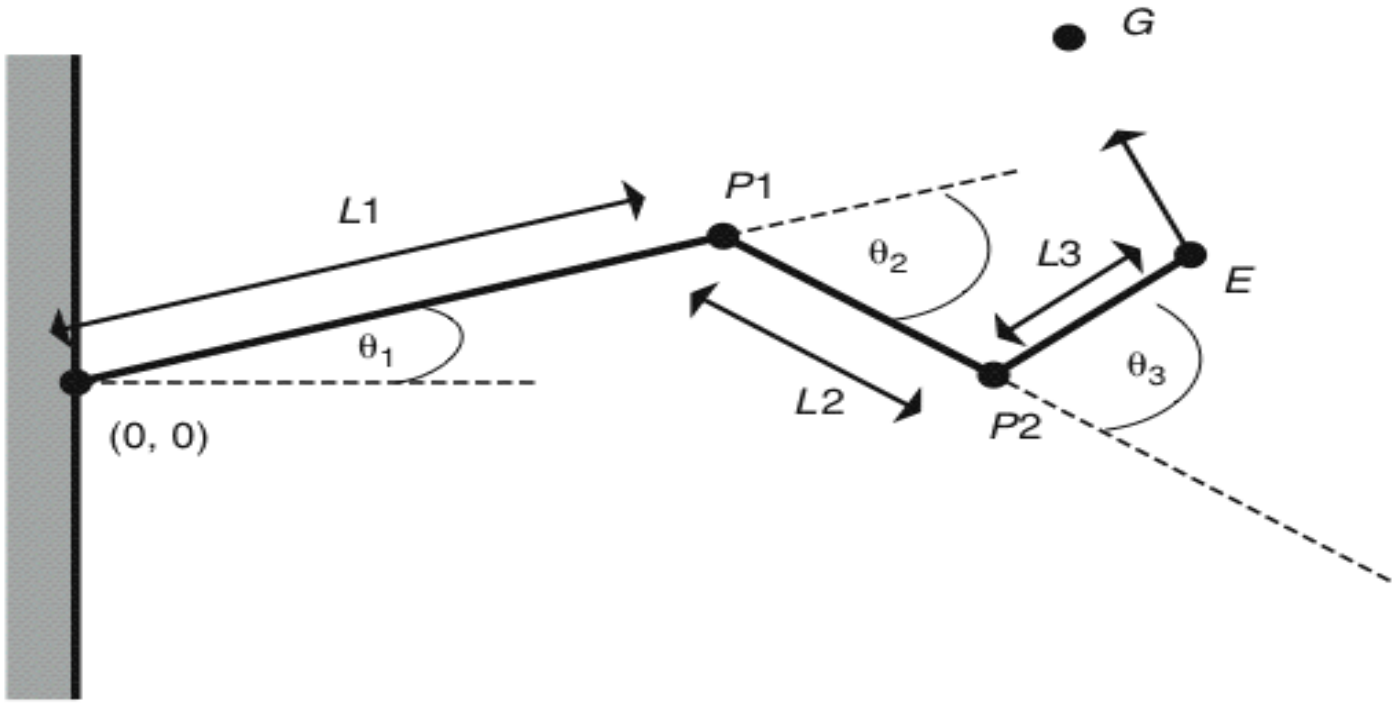
Inverse Kinematics

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

Chapter 4

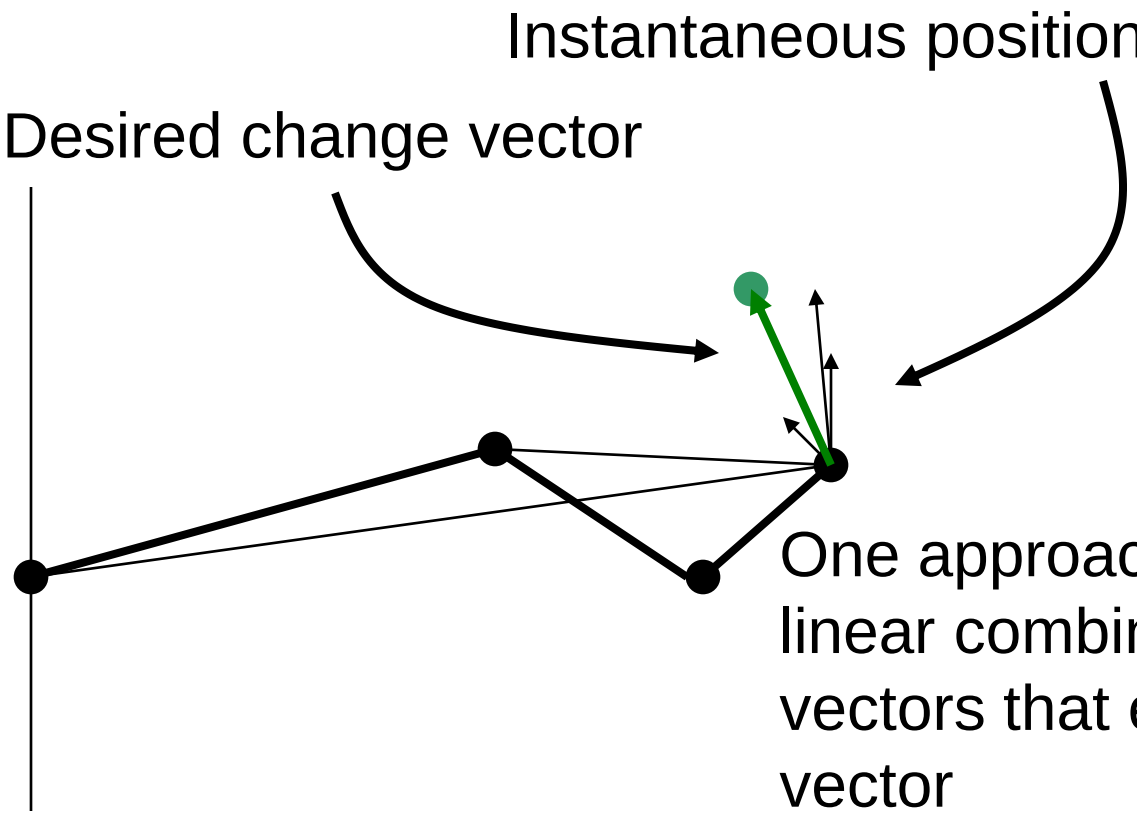
IK - configuration

Example: move end effector E to the goal position G.



Chapter 4

IK – compute positional change vectors induced by changes in joint angles



One approach to IK computes linear combination of change vectors that equal desired vector

Chapter 4

Inverse Kinematics

The desired change to the end effector is the difference between the current position of the end effector and the goal position:

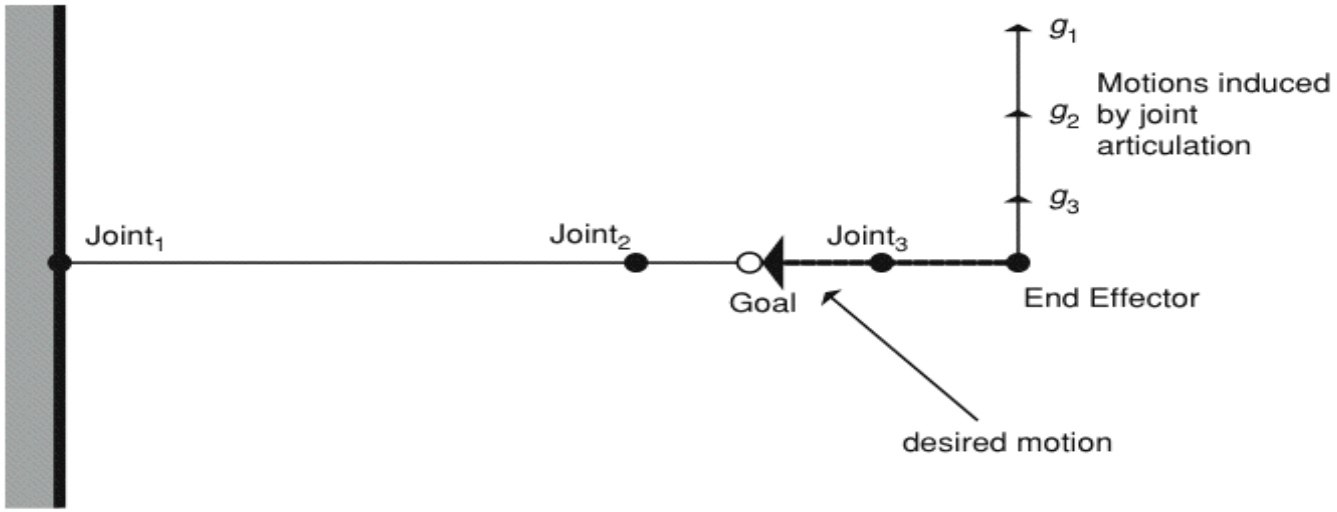
$$V = \begin{bmatrix} (G - E)_x \\ (G - E)_y \\ (G - E)_z \end{bmatrix}$$

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

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

Chapter 4

IK - singularity



Some singular configurations are not so easily recognizable

Near singular configurations are also problematic – why?

Chapter 4

Inverse Kinematics – Near Singular Configurations

A configuration that is only close to being a singularity can still present major problems. If the joints of the linkage in the previous slide 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 errors can result in unpredictable motion.

Chapter 4

Inverse Kinematics - Numeric

Given
- Current configuration
- Goal position/orientation

Determine
- Goal vector
- Positions & local coordinate systems of interior joints (in global coordinates)
- Jacobian

$V = J \dot{\theta}$ Is in same form as more recognizable : $Ax = b$

Solve & take small step – or clamp acceleration or clamp velocity

Repeat until:
- Within epsilon of goal
- Stuck in some configuration
- Taking too long

Chapter 4

Solving

If J square, compute inverse, J^{-1}

If J not square, usually under-constrained: more DoFs than constraints

Requires use of pseudo-inverse of Jacobian

Avoid direct computation of inverse by solving Ax=B form

$$V = J\theta$$

$$J^T V = J^T J\theta$$

$$(J^T J)^{-1} J^T V = (J^T J)^{-1} J^T J\theta$$

$$J^+ V = \theta$$

$$J^+ V = \theta$$

$$J^T \boxed{(JJ^T)^{-1} V} = \theta$$

$$\beta = (JJ^T)^{-1} V$$

$$(JJ^T)\beta = V$$

$$J^T \beta = \theta$$

Chapter 4

Solving

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 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 Jocobian ceases to accurately describe the relation ship between the changes in the joint angles and changes in end effector position and orientation.

Chapter 4

Solving

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.

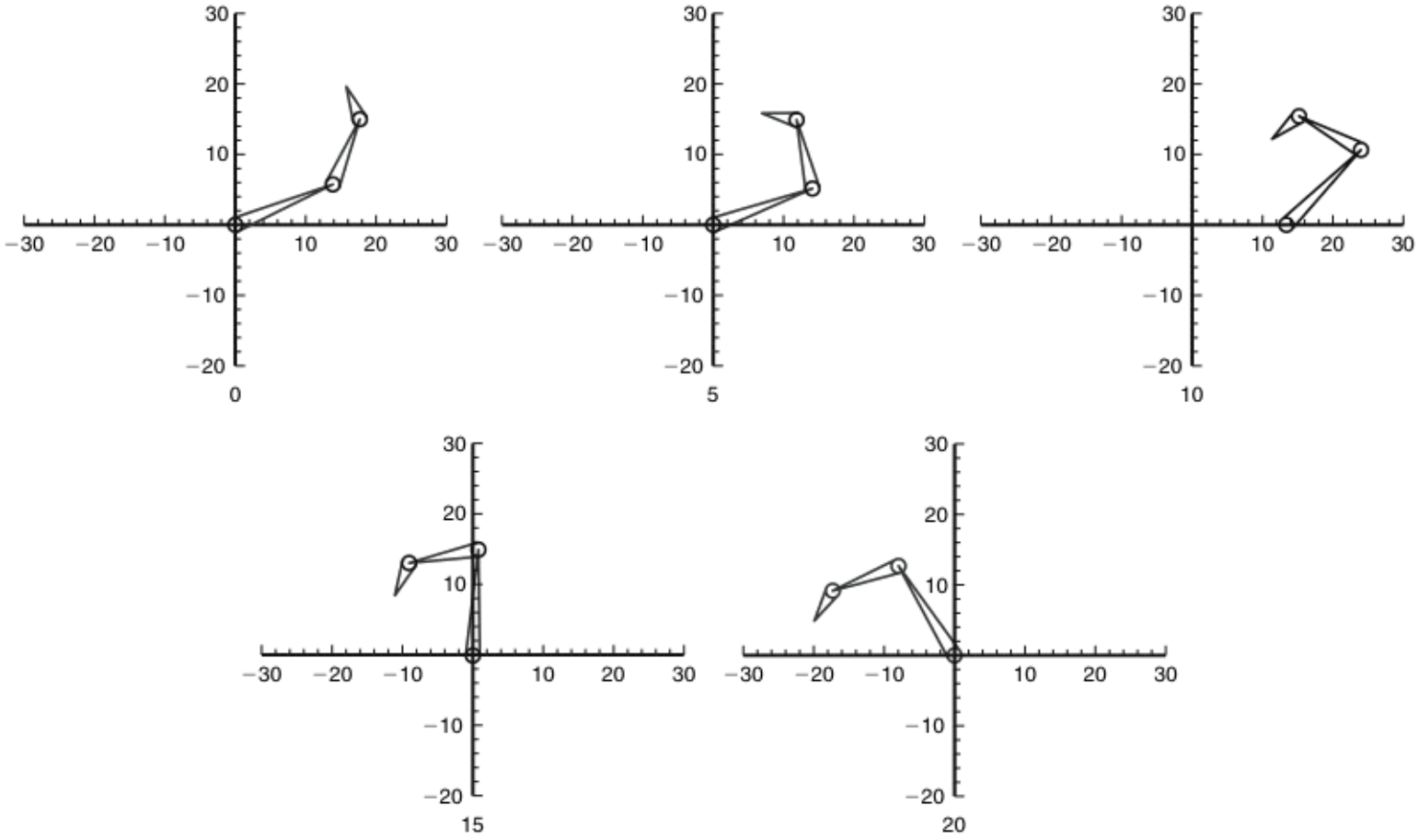
Chapter 4

Solving: Example

Consider a two-dimensional 3-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 for linearly interpolated intermediate goal positions for the end effector. The next slides shows frames of 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.

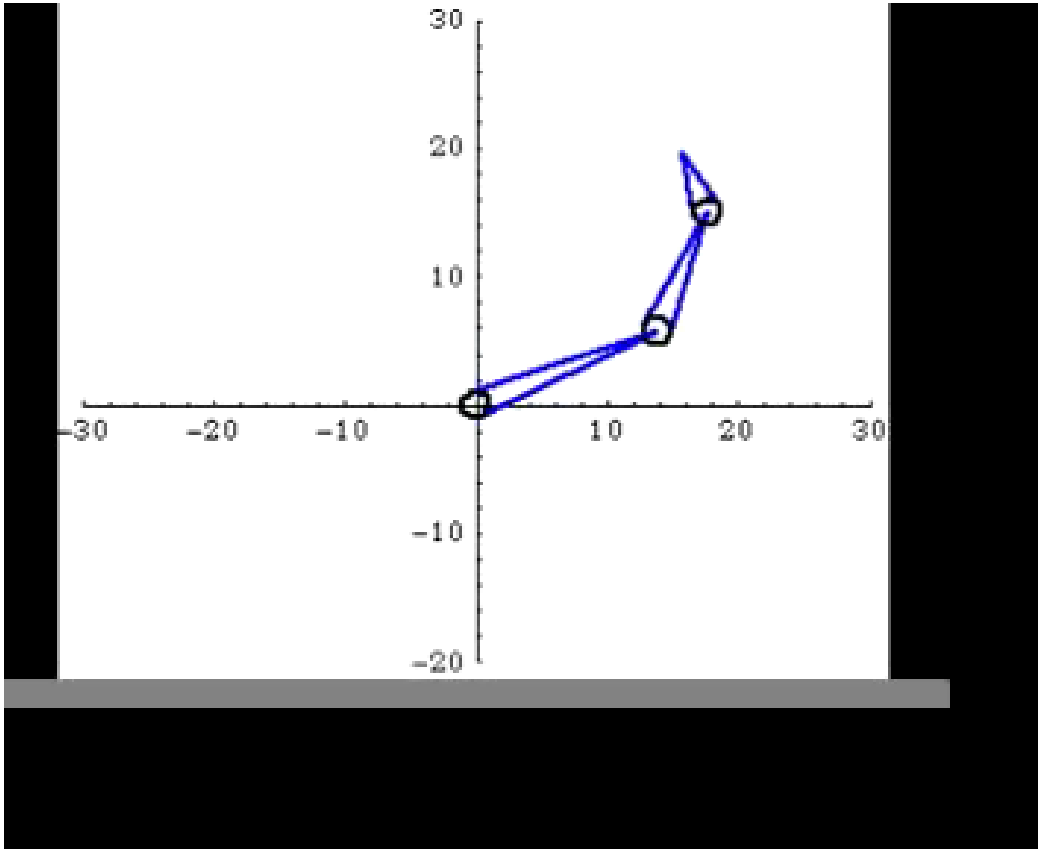
Chapter 4

IK – Jacobian solution



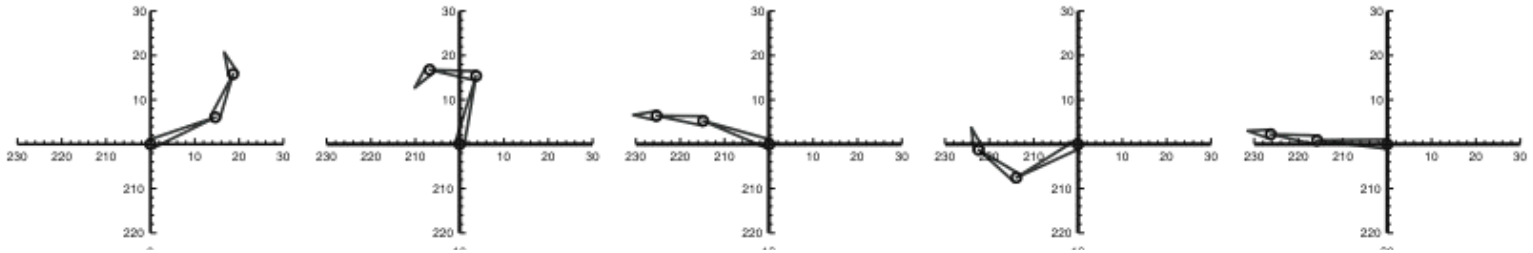
Chapter 4

IK – Jacobian solution



Chapter 4

IK – Jacobian solution - problem



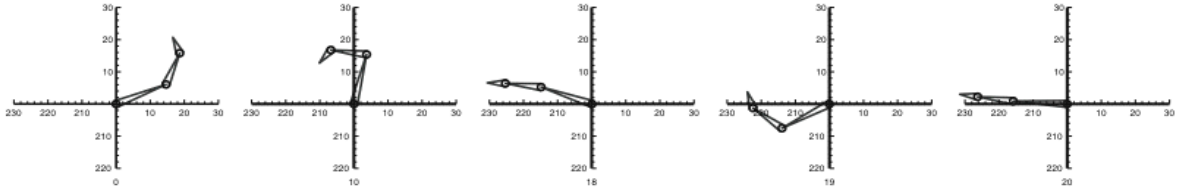
When goal is out of reach
Bizarre undulations can occur
As armature tries to reach the unreachable

→ Add a damping factor

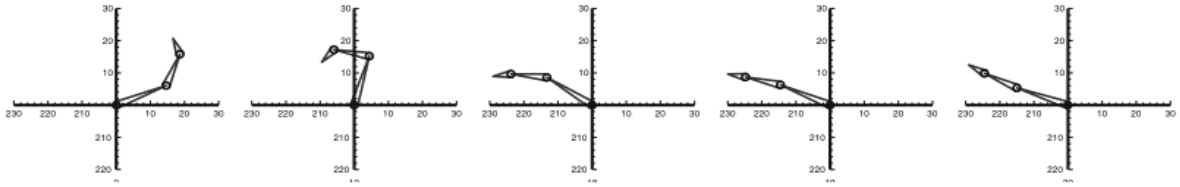
Chapter 4

IK – Jacobian w/ damped least squares

Undamped form: $\theta = J^T \left(J J^T \right)^{-1} V$



a) The pseudoinverse of Jacobian solution without damped least squares



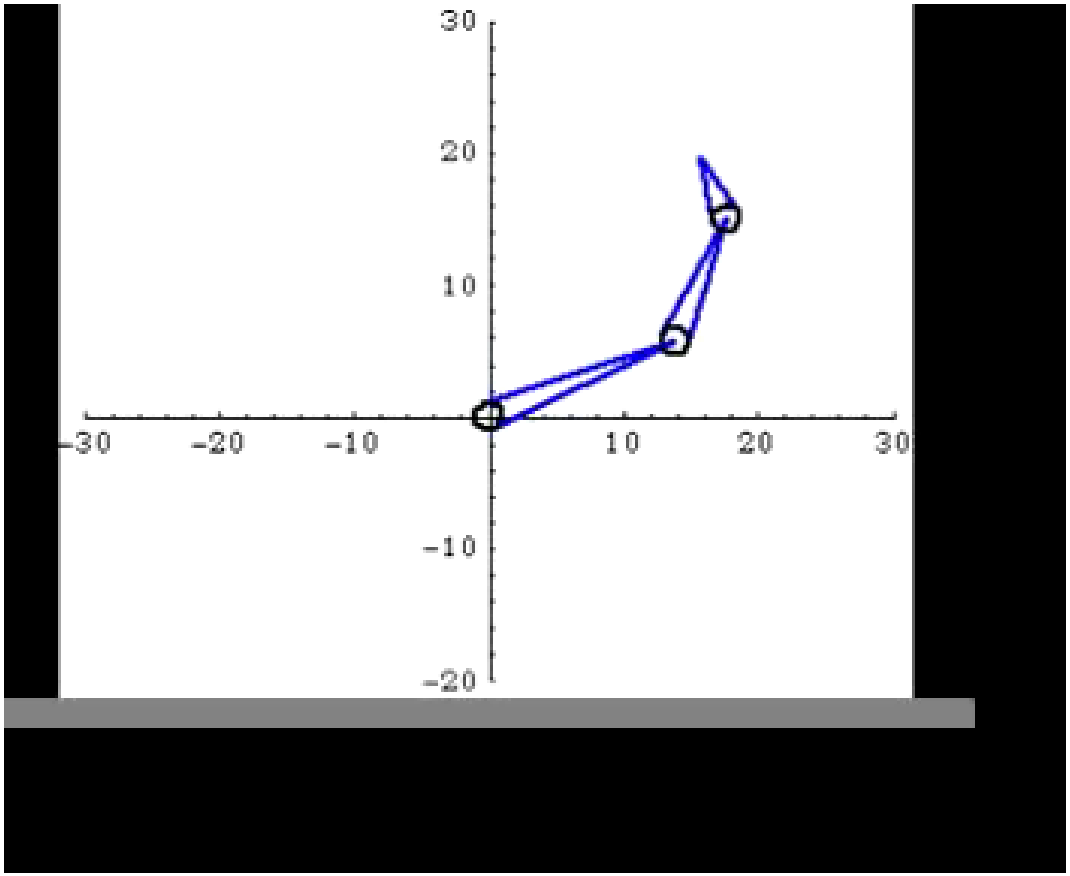
b) The pseudoinverse of Jacobian solution with damped least squares

Damped form with user parameter:

$$\theta = J^T \left(J J^T + \lambda^2 I \right)^{-1} V$$

Chapter 4

IK – Jacobian w/ damped least squares



Chapter 4

IK – Jacobian w/ control term

The pseudoinverse computes one of many possible solutions, it minimizes joint angle rates. The configuration 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.

Chapter 4

IK – Jacobian w/ control term

Physical systems (i.e. robotics) and synthetic character simulation (e.g., human figure) have limits on joint values

IK allows joint angle to have any value

Difficult (computationally expensive) to incorporate hard constraints on joint values

Take advantage of redundant manipulators - Allow user to set parameter that urges DOF to a certain value

Does not enforce joint limit constraints, but can be used to keep joint angles at mid-range values

Chapter 4

IK – Jacobian w/ control term

control expression

$$\theta = J^{+}V + (J^{+}J - I)^{-1}z$$

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

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

Change to the pose parameter in the form of the control term adds nothing to the velocity

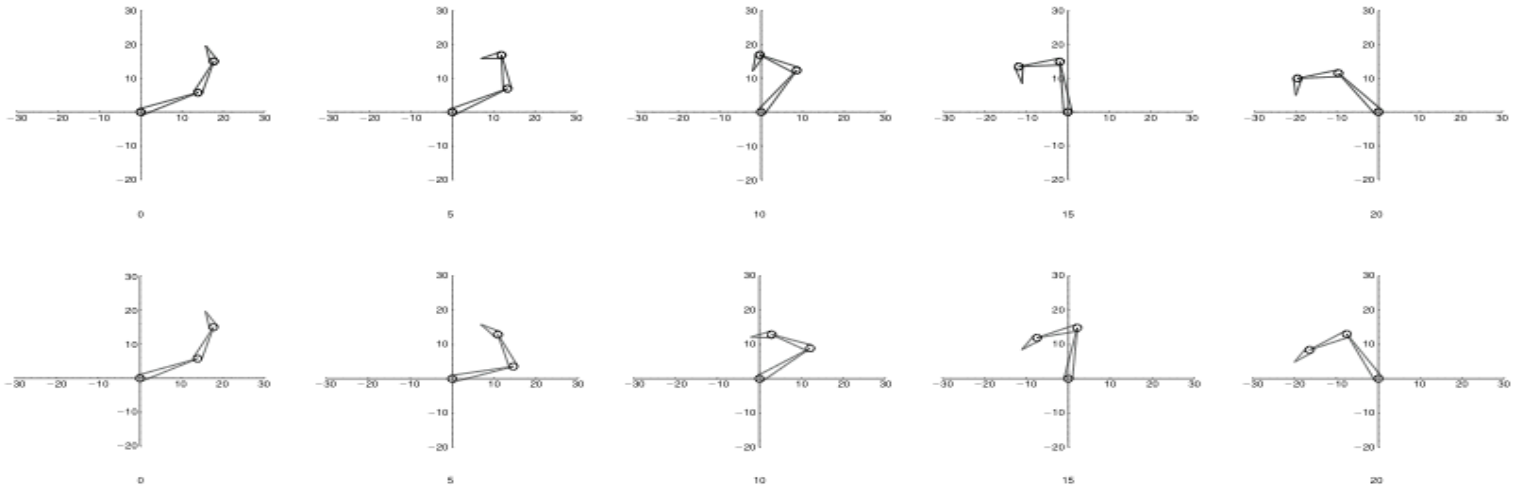
Chapter 4

IK – Jacobian w/ control term

To bias the solution toward the specific joint angles, such as the middle angle between joints, z is defined as $z=\alpha(\theta_i-\theta_{ci})^2$, where θ_i are the current joint angles, θ_{ci} are the desired joint angles, and θ_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.

Chapter 4

IK – Jacobian w/ control term



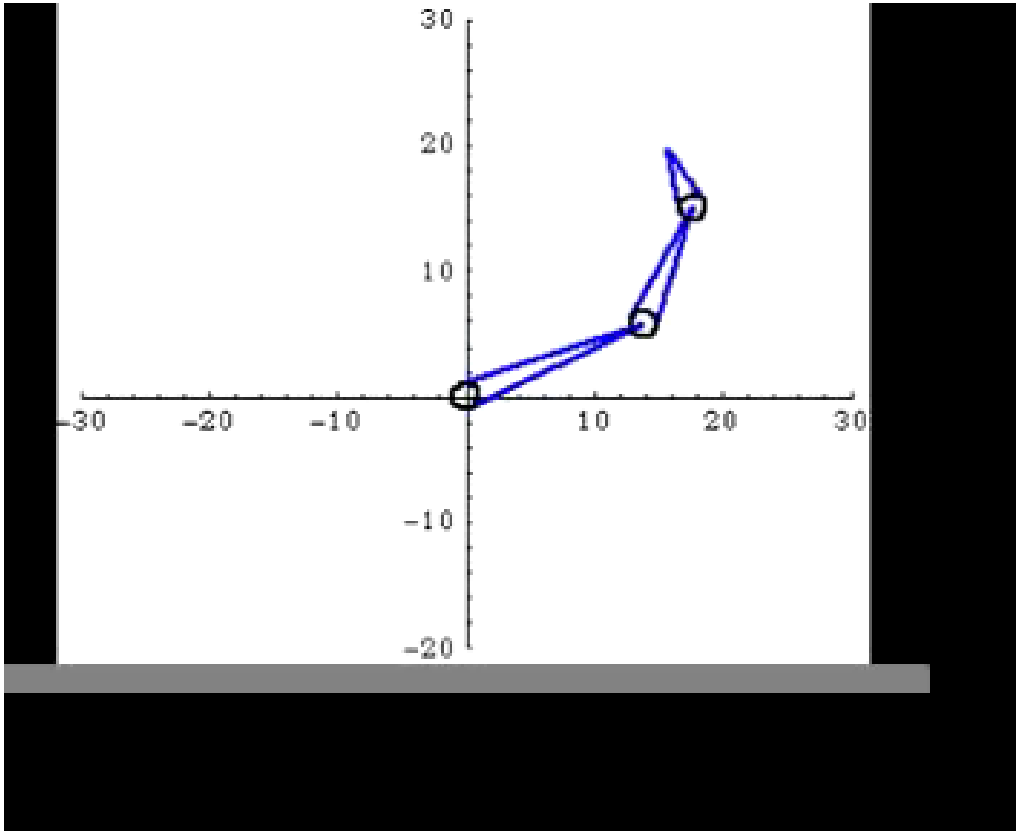
All bias to 0
Top gains = {0.1, 0.5, 0.1}
Bottom gains = {0.1, 0.1, 0.5}

$$\theta = J^{+}V + (J^{+}J - I)^{-1} z$$

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

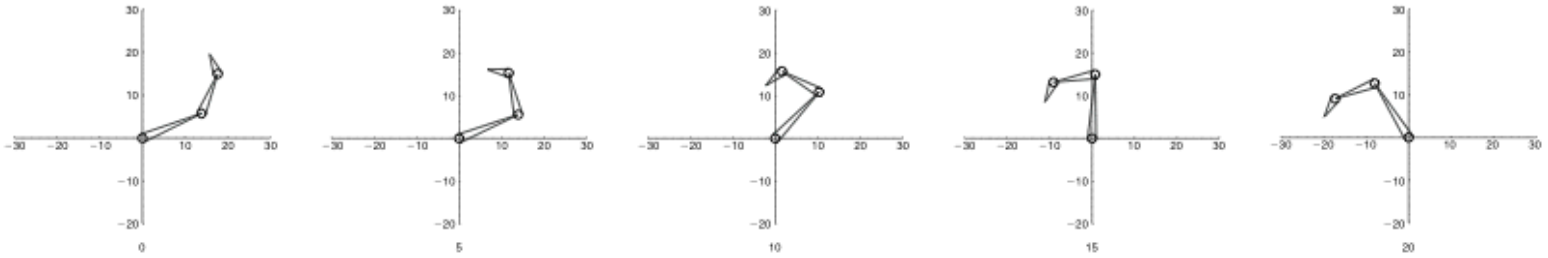
Chapter 4

IK – Jacobian w/ control term



Chapter 4

IK – alternate Jacobian

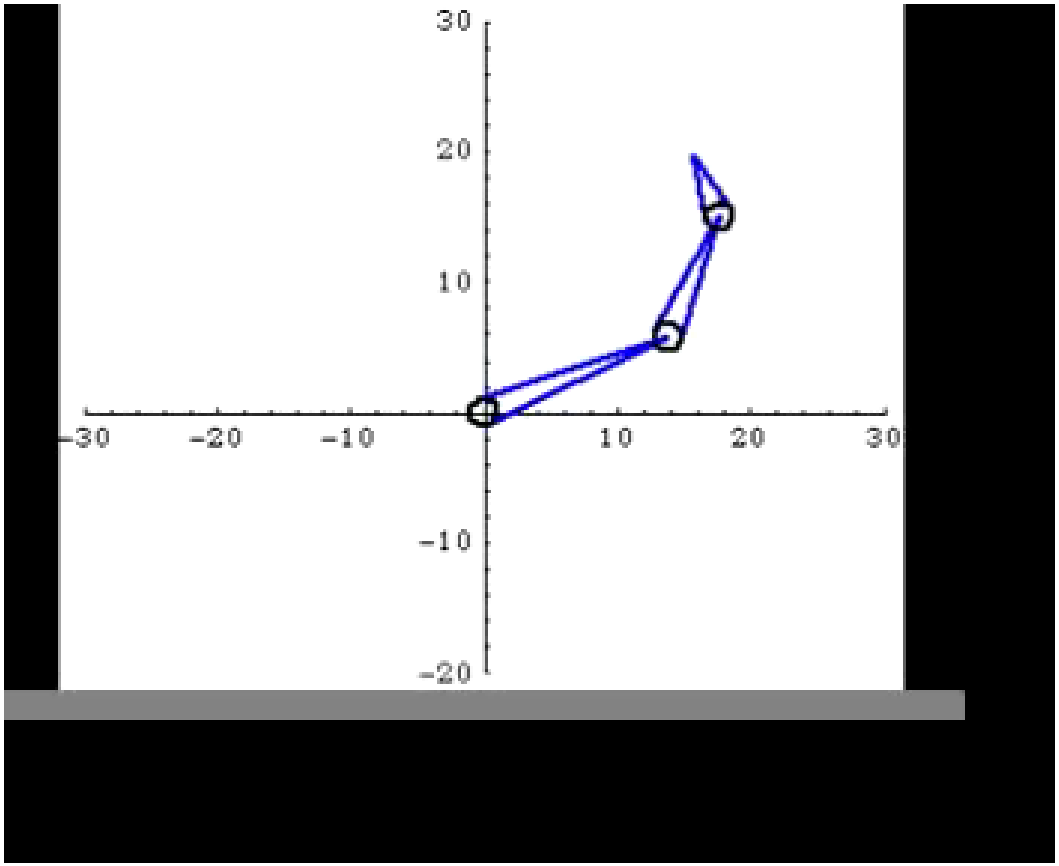


Jacobian formulated to pull the goal toward the end effector

Use same method to form Jacobian but use goal coordinates instead of end-effector coordinates

Chapter 4

IK – alternate Jacobian



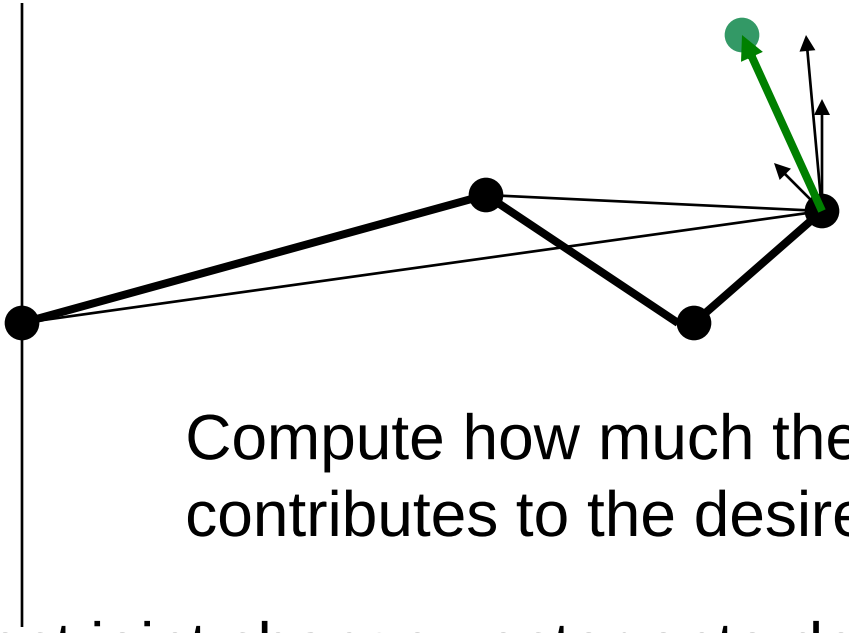
Chapter 4

IK – 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. This entails forming the dot product between the instantaneous change vector and the velocity vector.

Chapter 4

IK – Transpose of the Jacobian



Compute how much the change vector contributes to the desired change vector:

Project joint change vector onto desired change vector

Dot product of joint change vector and desired change vector → Transpose of the Jacobian

Chapter 4

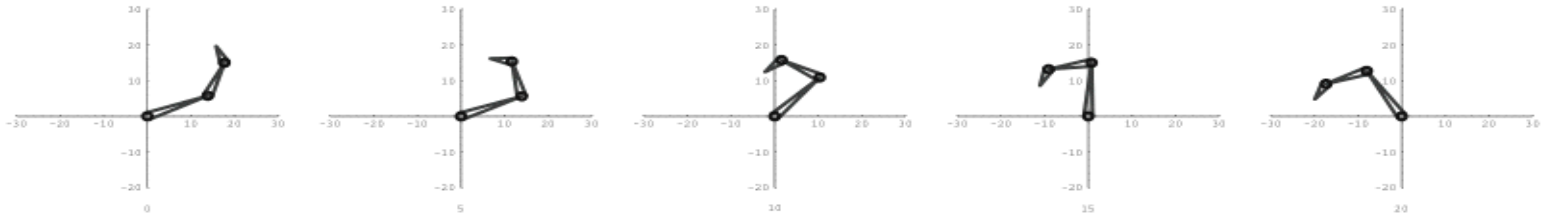
IK – Transpose of the Jacobian

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 the scaled version of this as the joint parameter change vector:

Chapter 4

IK – Transpose of the Jacobian

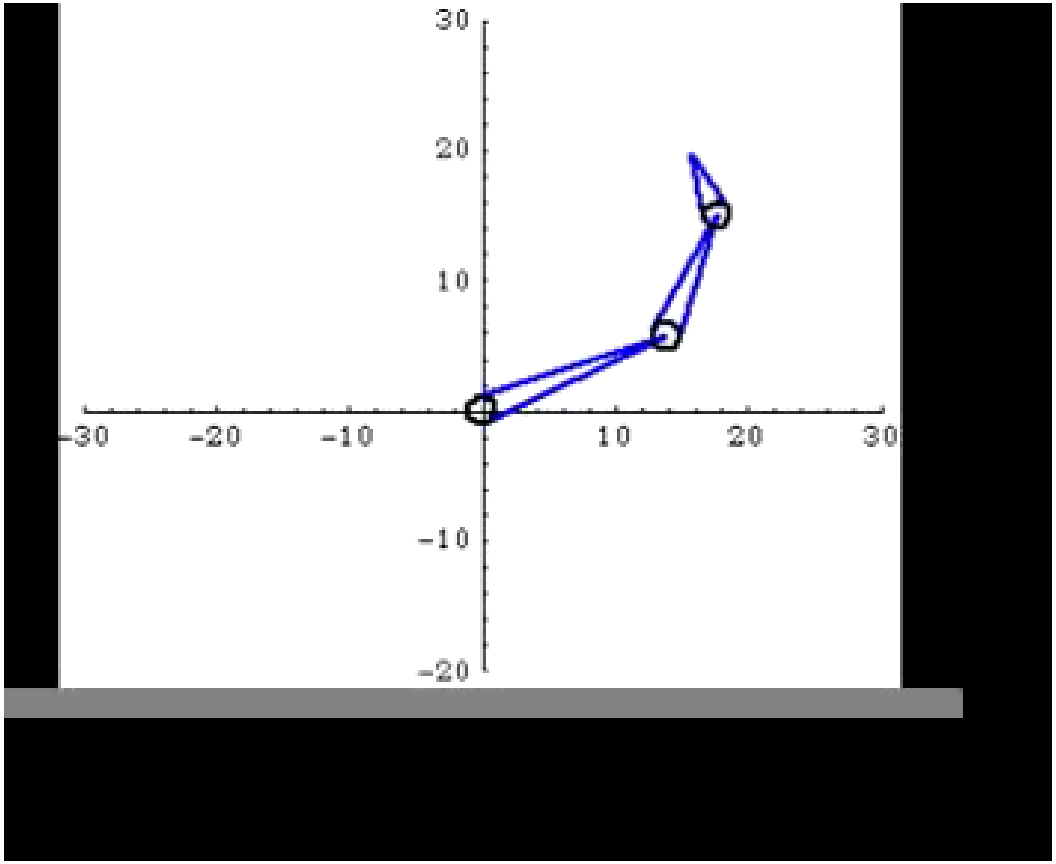
$$J^T V = \theta$$



$$J^T = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_1} & \cdots & \frac{\partial \alpha_z}{\partial \theta_1} \\ \frac{\partial p_x}{\partial \theta_2} & \cdots & & \\ \cdots & & & \\ \frac{\partial p_x}{\partial \theta_6} & & & \frac{\partial \alpha_z}{\partial \theta_6} \end{bmatrix} \qquad V = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

Chapter 4

IK – Transpose of the Jacobian



Chapter 4

IK – Transpose of the Jacobian

This avoids the expense of computing the inverse, or pseudoinverse, of the Jacobian. In certain configurations a zero vector may result. The main drawback is that even though a given instantaneous change vector might contribute to the velocity vector, it may also take the end effector well away from the desired direction.

Chapter 4

IK – 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. At each joint, an angle is chosen that best gets the end effector to the goal position.

Chapter 4

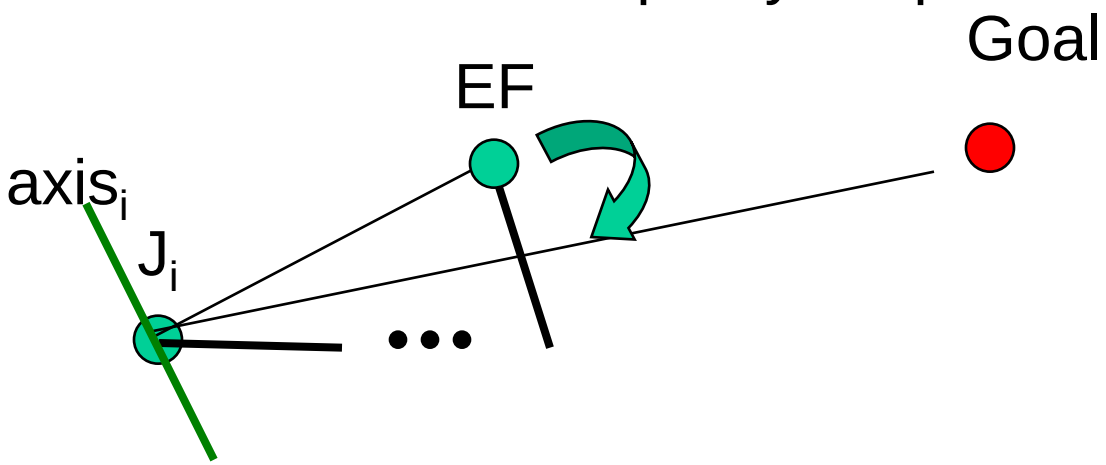
IK – cyclic coordinate descent

Heuristic solution

Consider one joint at a time, from outside in

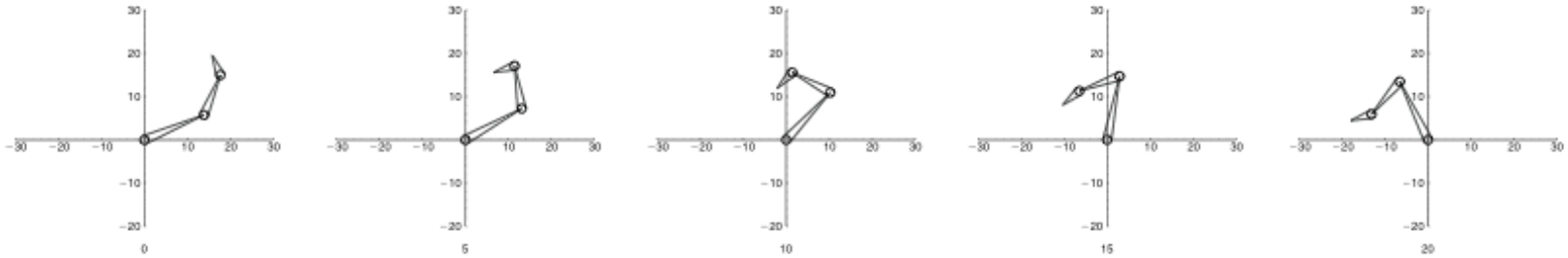
At each joint, choose update that best gets end effector to goal position

In 2D – pretty simple



Chapter 4

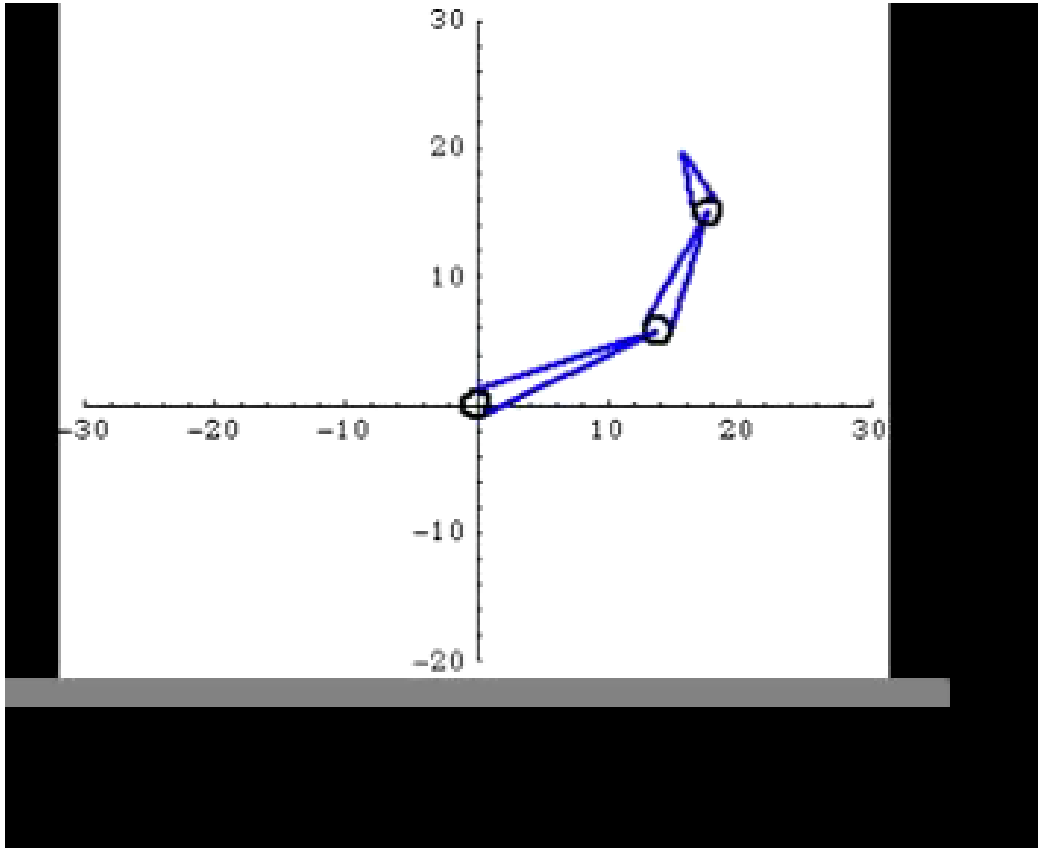
IK – cyclic coordinate descent



In 3D, a bit more computation is needed

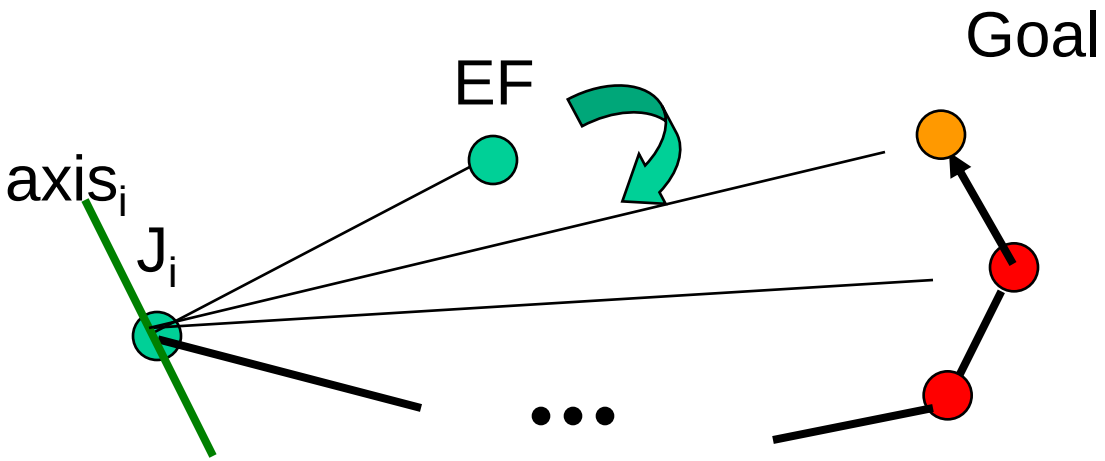
Chapter 4

IK – cyclic coordinate descent



Chapter 4

IK – cyclic coordinate descent – 3D



First – goal has to be projected onto plane defined by axis and EF

Chapter 4

IK – cyclic coordinate descent – 3D

Other orderings of processing joints are possible

Because of its procedural nature
- Lends itself to enforcing joint limits
- Easy to clamp angular velocity

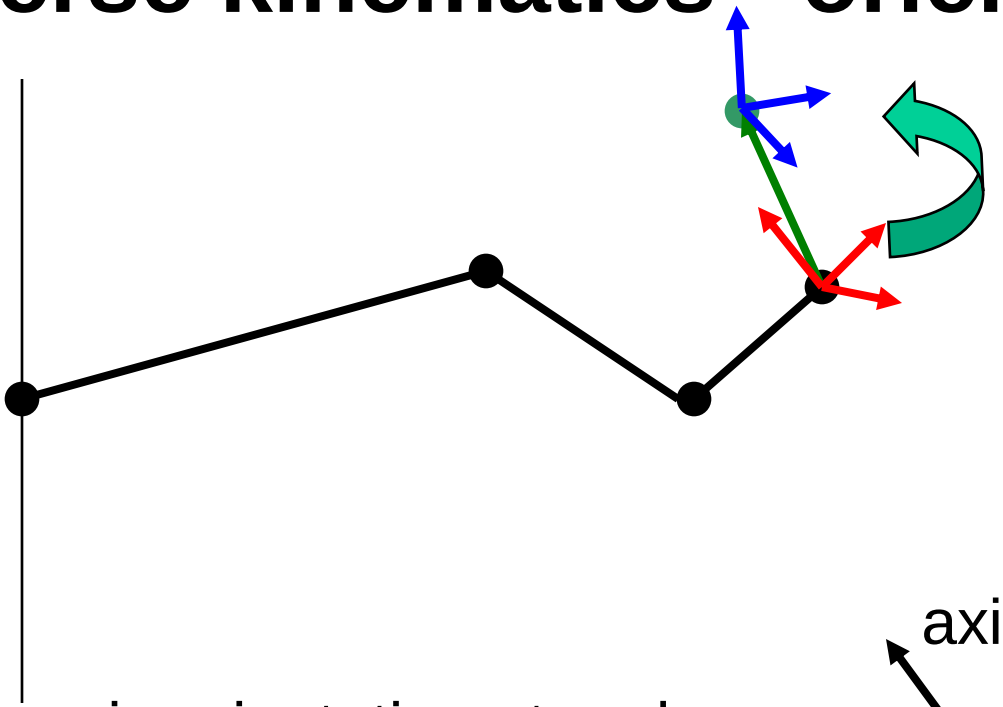
Chapter 4

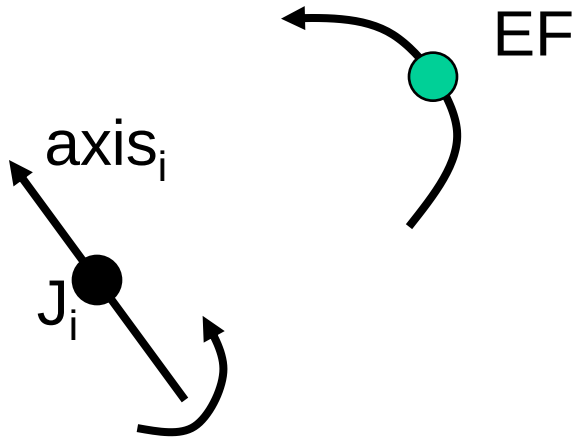
Inverse kinematics - review

- Analytic method
- Forming the Jacobian
- Numeric solutions
  - Pseudo-inverse of the Jacobian
  - J^+ with damping
  - J^+ with control term
  - Alternative Jacobian
  - Transpose of the Jacobian
  - Cyclic Coordinate Descent (CCD)

Chapter 4

Inverse kinematics - orientation





Change in orientation at end-effector is same as change at joint

Chapter 4

Inverse kinematics - orientation

How to represent orientation (at goal, at end-effector)?
How to compute difference between orientations?
How to represent desired change in orientation in V vector?
How to incorporate into IK solution?

Matrix representation: M_g, M_{ef}

Difference $M_d = M_{ef}^{-1} M_g$

Use scaled axis of rotation: $B(a_x\ a_y\ a_z)$:

- Extract quaternion from M_d
- Extract (scaled) axis from quaternion

E.g., use Jacobian Transpose method:
Use projection of scaled joint axis onto extracted axis