Computer Animation
6-Kinematics
SS 18

Prof. Dr. Charles A. Wüthrich,
Fakultät Medien, Medieninformatik
Bauhaus-Universität Weimar
caw AT medien.uni-weimar.de

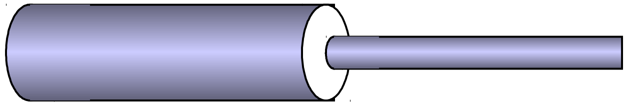
Hierarchical modeling

- Hierarchical modeling is placing constraints on objects organized in a tree like structure
- Examples can be:
  - A planet system
  - A robot arm
- The latter is quite common in graphics: it is constituted by objects connected end to end to form a multibody jointed chain
- These are called *articulated figures*

- They stem from robotics
- Robotics literature speaks with a different terminology:
  - Manipulator: the sequence of objects connected by joints
  - Links: the rigid objects making the chain
  - Effector: the free end of the chain
  - Frame: local coordinate system associated to each link

Hierarchical modeling

- In graphics, most of the links are revolute joints: here one link rotates around a fixed point of the other link
- The other interesting joint for graphics is the prismatic joint, where one link translates relative to the other
- Joints restrain the degree of freedom (DOF) of the links
- Joints with more than one degree of freedom are called *complex*
- Typically, when a joint has n>1 DOF it is modeled as a set of n one degree of freedom joints

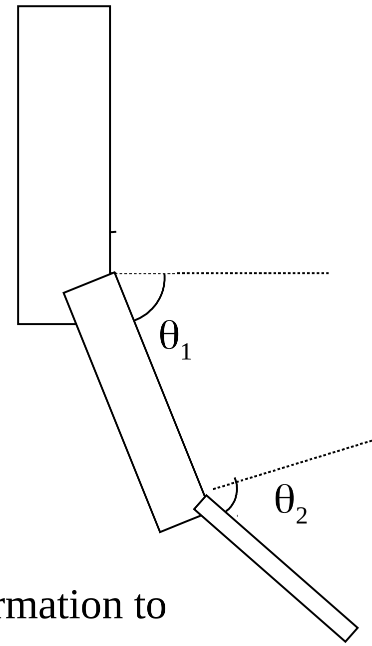


Hierarchical modeling

- Humans and animals can be modeled as hierarchical linkages
- These are represented as a tree structure of nodes connected by arcs
- The highest node of this structure is called the root node, and is the node that has position WRT the global coordinate system
- All other nodes have their position only as relative to the root node
- A node that has no child is called a leaf node
- Each node contains the info necessary to define the position of the corresponding part
- Two types of transformations are associated with an arc leading to a node:
  - Rotation and translation of the object to its position of attachment to the father link
  - Information responsible for the joint articulation

Hierarchical modeling

- How does this work?
- The idea is simple, store at each node
  - Info on the node geometry
  - The transformation (its rotation) with respect to the father node in the tree
- To obtain the position of the i-th node in the chain, one has to simply multiply the transformations to obtain the position of the current arc to be displayed
- The root node of course contains info of its absolute position and orientation in the global coord. system



T_0: transformation to rotate K_0 in WCS

T_1: transformation to rotate K_1 WRT K_0 = rotation by θ_1

T_2: transformation to rotate K_2 WRT K_1 = rotation by θ_2

- To obtain the position of K_2 in WCS, one will then have to multiply $T_0T_1T_2$

Forward kinematics

- Traversing the tree of the nodes produces the correct picture of the object
- Traversal is done depth first until a leaf is met
- Once the corresponding arc is evaluated, the tree is backtracked up until the first unexplored node is met
- This is repeated until there are no nodes left inexplored

- A stack of transforms is kept
- When tree is traversed downwards, the corresponding trans-formation is added to the stack
- Moving up pops the transformation from the stack
- Current node position is generated through multiplying the current stack transforms

Forward kinematics

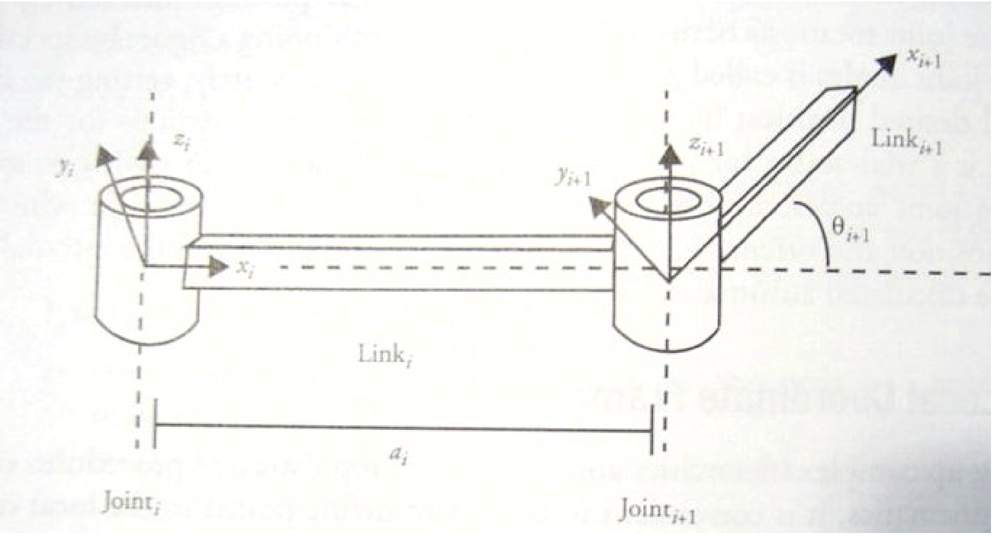
- To animate the whole, the rotation parameters are manipulated and the corresponding transforms are actualized
- A complete set of rotations on the whole arcs is called a *pose*
- A pose is obviously a vector of rotations

- Moving an object by positioning all its single arcs manually is called forward kinematics
- This is not so user-friendly
- Instead of specifying the whole links, the animator might want to specify the end position of the effector
- The computer computes then the position of the other links
- This is called *inverse kinematics*

Denavit-Hartenberg Notation

- Used in robotics
- Frames are described relative to an adiacent frame by 4 parameters describing position and orientation of a child frame WRT parent frame
- Let us take a simple configuration like in this drawing, where the link rotates only in one direction

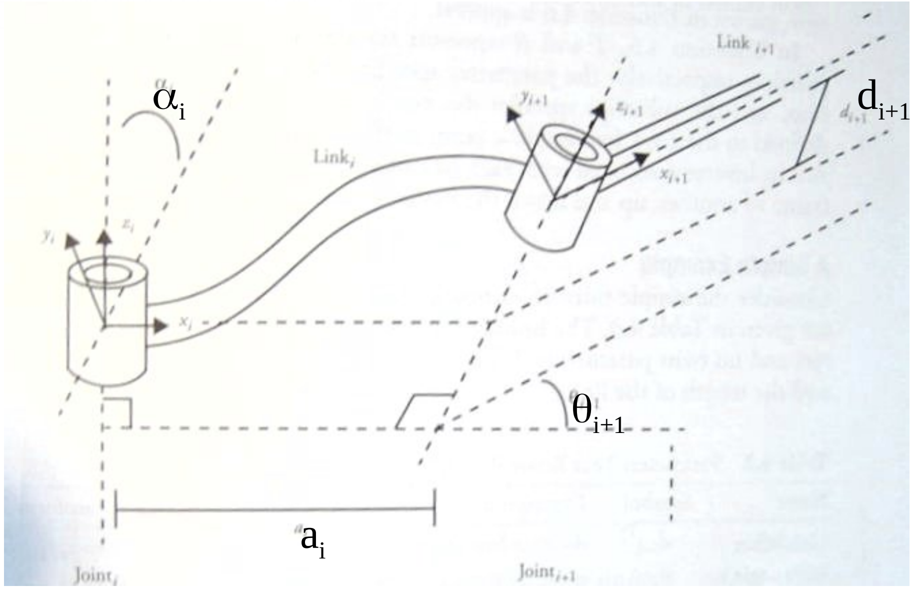
- a_i: link length
- Θ_{i+1}: joint angle, i.e. rotation around z axis with the last link direction as 0 angle



Denavit-Hartenberg Notation

- If the joint is non planar, then one adds additional paramenters
- For general case, the x axis of the i-th joint is defined as the ⊥ segment to the z-axes of the *i*-th and *(i+1)*-th frames
- The link twist parameter α_i is the rotation of the i+1th frame‘s z axis around the ⊥ relative to the z axis of the *i*-th frame
- The link offset d_{i+1} specifies the distance along the z axis (rotated by α_i) if the *(i+1)*-th frame from the *i*-th x axis

Name	Symbol	
Link offset	d_i	Distance $x_{i-1}\,x_i$ along z_i
Joint angle	θ_i	Angle $x_{i-1}\,x_i$ about z_i
Link length	a_i	Distance $z_i\,z_{i+1}$ along x_i
Link twist	α_i	Angle $z_i\,z_{i+1}$ about x_i



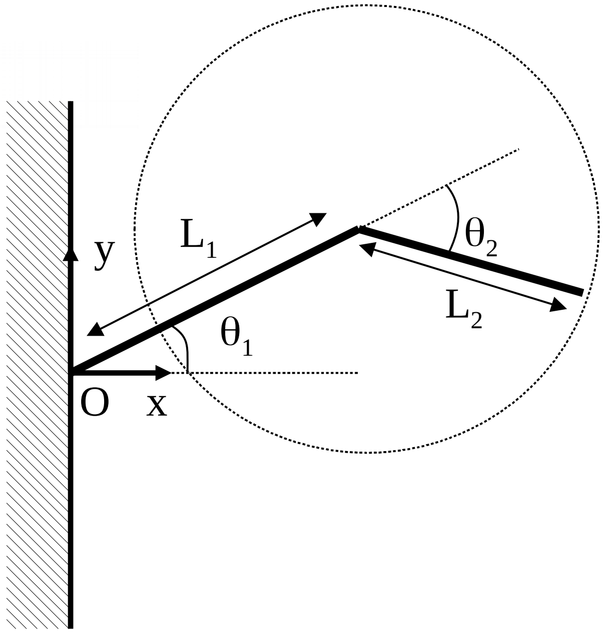
Inverse kinematics

- The user gives the position of the end effector and the computer computes the joint angles
- One can have zero, one or multiple solutions
  - No solution: overconstrained problem
  - Multiple solutions: underconstrained problem
  - Reachable workspace: volume that end effector can reach
  - Dextrous workspace: volume that end effector can reach in any orientation
- Computing the solution to the problem can at times be tricky
- If the mechanism is simple enough, then the solution can be computed analytically
- Given an initial and a final pose vector, the solution can be computed by interpolating the values of the pose vector
- If the solution cannot be computed analytically, then there is a method based on the jacobian to compute incrementally a solution

Inverse kinematics

- Consider the figure: the 2nd arm rotates aroond the end of the 1st arm.
- It is clear that all positions between $|L_1-L_2|$ and $|L_1+L_2|$ can be reached by the arm.
- Set the origin like in the drawing
- In inverse kinematics, the user gives the (X,Y) position of the end effector

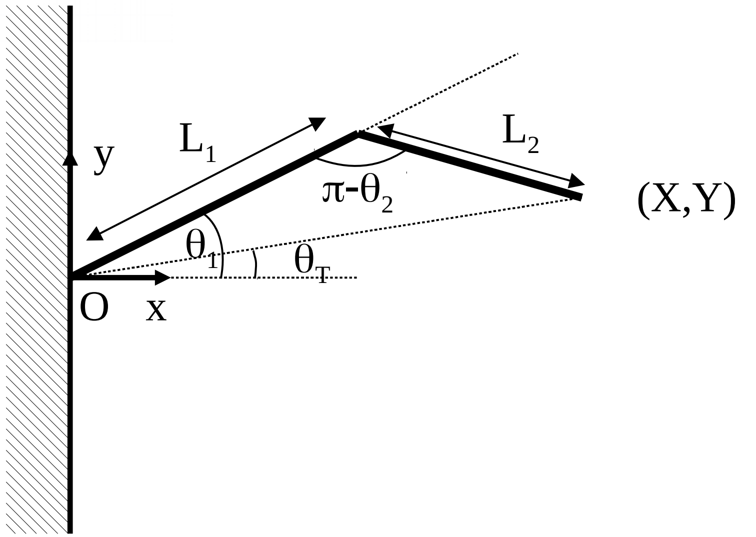
- Obviously there are only solutions if $|L_1-L_2| \leq \sqrt{X^2+Y^2} \leq |L_1+L_2|$



Inverse kinematics

- $\cos\theta_T = X/(X^2+Y^2)^{½}$
  $\Rightarrow \theta_T = \mathrm{acos}(X/(X^2+Y^2)^{½})$
- Because of the cosine rule we have also that
  $\cos(\theta_1-\theta_T) =$
  $(L_1^2+X^2+Y^2-L_2^2)/2L_1\sqrt{X^2+Y^2}$
  and
  $\cos(\pi-\theta_2) =$
  $(L_1^2+L_2^2-(X^2+Y^2)^{½})/2L_1L_2$
  from which we have
  $\theta_1 = \mathrm{acos}((L_1^2+X^2+Y^2-L_2^2)$
  $/2L_1(X^2+Y^2)^{½} + \theta_T$
  and
  $\theta_2 = \mathrm{acos}((L_1^2+L_2^2-(X^2+Y^2))/2L_1L_2)$

- Note that two solutions are possible, simmetric with respect to the line joining the origin and (X,Y)



Inverse kinematics

- In general, for the quite simple armatures used in robotics it is possible to implement such analytic solutions
- Unfortunately this works only for simple cases
- For more complicated armatures, the number of possible solutions there may be infinite solutions for a given effector location, and computations become so difficult to do that iterative numeric solution must be used

Using the Jacobian

- When the solution is not analytically computable, incremental methods converging to the solution are used
- To do this, the matrix of the partial derivatives has to be computed
- This is called the *Jacobian*

- Suppose you have six independent variables and you have a six unknowns that are functions of these variables

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

  or, in vector notation,

  $\underline{Y}=\underline{F}(\underline{X})$

Using the Jacobian

- What happens when the input variables change?

The equations can be written in differential form:

$dy_i = \partial f_i/\partial x_1 \, dx_1 + \partial f_i/\partial x_2 \, dx_2$
$+ \partial f_i/\partial x_3 \, dx_3 + \partial f_i/\partial x_4 \, dx_4$
$+ \partial f_i/\partial x_5 \, dx_5 + \partial f_i/\partial x_6 \, dx_6$

or, in vector form

$d\underline{Y} = \partial \underline{F}/\partial \underline{X} \, d\underline{X}$

- Given n equations in n variables, the matrix

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

is called the Jacobian matrix of the system

- The Jacobian can be seen as a mapping of the velocities of $\underline{X}$ to velocities of $\underline{Y}$

Using the Jacobian

- The Jacobian matrix is a linear function of the x_i variables
- When time moves on to the next instant, X has changed and so has the Jacobian

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

- When the jacobian is applied to a linked appendage, the x_i variables are the angles of the joints and the y_i variables are end effector positions

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

where V is the vector of linear and rotational changes and represents the desired change in the end effector

- The desired change will be based on the difference between the current position/orientation to the desired goal configuration

Using the Jacobian

- Such velocities are vectors in 3 space, so each has x,y,z components
- $\dot{\vartheta}$ is a vector of joint angle velocities which is the unkowns
- The Jacobian matrix J relates the two and is a function of the current pose
- Each term of the Jacobian relates the change of a specific joint to a specific change in the end effector
- The rotational change in the end effector is the velocity of the joint angle around its axis of revolution at the joint currently considered

- $V=[v_x,v_y,v_z,\omega_x,\omega_y,\omega_z]^T$

$$\dot{\vartheta} = \left[\dot{\vartheta}_1, \dot{\vartheta}_2, \ldots, \dot{\vartheta}_n\right]$$

$$J = \begin{bmatrix} \frac{\partial v_x}{\partial \vartheta_1} & \frac{\partial v_x}{\partial \vartheta_2} & \cdots & \frac{\partial v_x}{\partial \vartheta_n} \\ \frac{\partial v_y}{\partial \vartheta_1} & \frac{\partial v_y}{\partial \vartheta_2} & \cdots & \frac{\partial v_y}{\partial \vartheta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \omega_z}{\partial \vartheta_1} & \frac{\partial \omega_z}{\partial \vartheta_2} & \cdots & \frac{\partial \omega_z}{\partial \vartheta_n} \end{bmatrix}$$

Using the Jacobian

- How are the angular and linear velocities computed?
- One finds the difference between the end effector's current position and desired position
- The problem is to find out the best linear combination of velocities induced by the various joints that would achieve the desired velocities of the end effector
- The Jacobian is formed (by posing the problem in angle form)
- Once the Jacobian is formed, it has to be inverted in order to solve the problem
- If the Jacobian is square, then
  - From $V = J\dot{\vartheta}$ we have $J^{-1}V = \dot{\vartheta}$
  - If J^{-1} does not exist, the system is called singular

Using the Jacobian

- If the Jacobian is non square then if the manipulator is redundant it is still possible to find solutions to the problem
- This is done by using the *pseudoinverse* matrix
  $J^{+}=(J^{T}J)^{-1}J^{T}=J^{T}(JJ^{T})^{-1}$
- The pseudoinverse maps desired velocities of the end effector to the required velocities at the joint angle
- after making the following substitutions
  $J^{+}V=\theta$
  $J^{T}(JJ^{T})^{-1}V=\theta$
  $\beta=(JJ^{T})^{-1}V$
  $(JJ^{T})\beta=V$
  $J^{T}\beta=\theta^{\circ}$
  (*)
- And LU decomposition can be used to solve this eq. for β
- Remember that the Jacobian varies at every instant
- This means that if a too big step is taken in angle space, the end effector might travel to the wrong place

---

(*) due to the clumsiness of the program I am using here, I have decided to indicate derivative vectors as θ° instead than with a dot on top, which allows me to avoid an eq. editor

Using the Jacobian

- The pseudoinverse minimizes joint angle rates, but this might at times result in „innatural“ movements
- To better control the kinematic model, a control expression can be added to the pseudo inverse Jacobian solution
- The control expression is used to solve for certain control angle rates having certain attributes, and adds nothing to the desired end effector

- $\dot{\theta} = (J^{+}J - I)z$
  $V = J\dot{\theta}$
  $V = J(J^{+}J - I)z$
  $V = (JJ^{+}J - J)z$
  $V = (J - J)z$
  $V = 0z$
  $V = 0 \qquad (*)$
- To bias the angle towards a specific solution, desired angle gains α are added to the equations, and the equation is solved like before.
- In fact, for $\alpha=0$ one has the same pseudoinverse solution

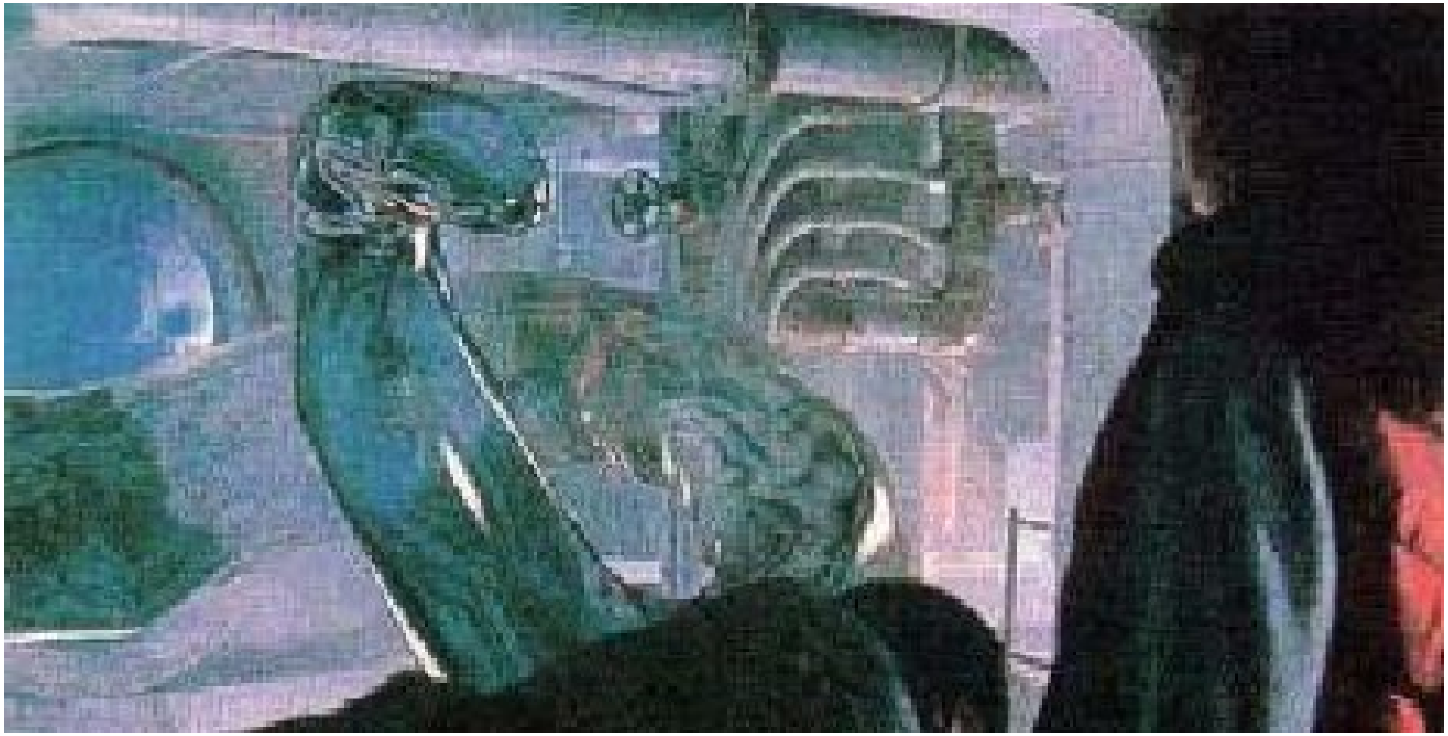
Using the Jacobian

- Simple Euler integration can be used at this point to update the joint angles
- At the next step, since the Jacobian has changed, the computations have to be redone and a new step is taken
- This is repeated until the end effector desired position is reached

Summary: articulated bodies

- Very useful for enforcing certain relationships among elements of an animation
- Allows animator to concentrate on effector forgetting the rest of the body
- Damn hard to do, to date not real in real time
- Adding control expressions can be tricky
- No physics considered. Only kinematics

End



+++ Ende - The end - Finis - Fin - Fine +++ Ende - The end - Finis - Fin - Fine +++