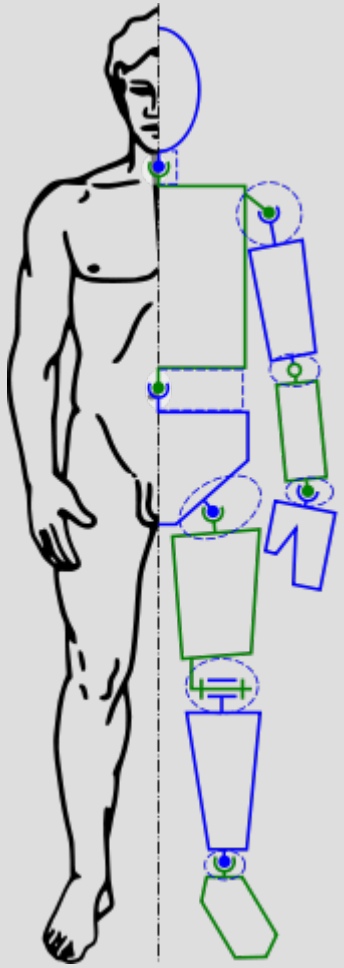
Kinematics Overview

- Forward Kinematics
- Inverse Kinematics
- IK Issues
- Solving IK
  - Analytical Solutions
  - Numerical Solutions
    - Jacobian
    - CCD
    - Forward/Backward reaching



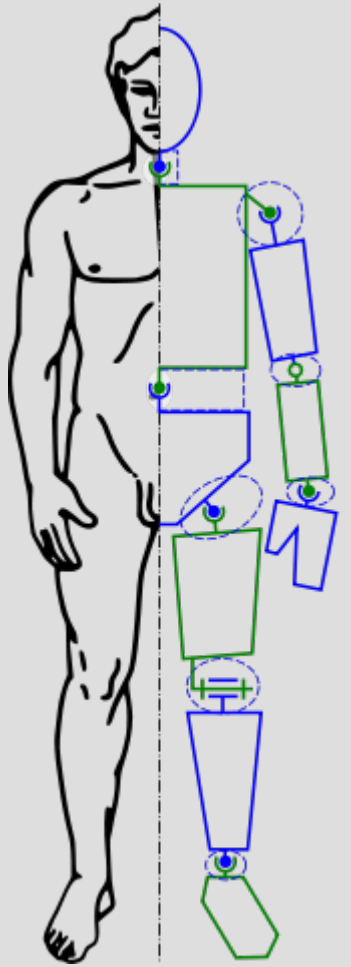
Kinematics vs. Dynamics

Kinematics

Describes the positions of the body parts as a function of the joint angles.

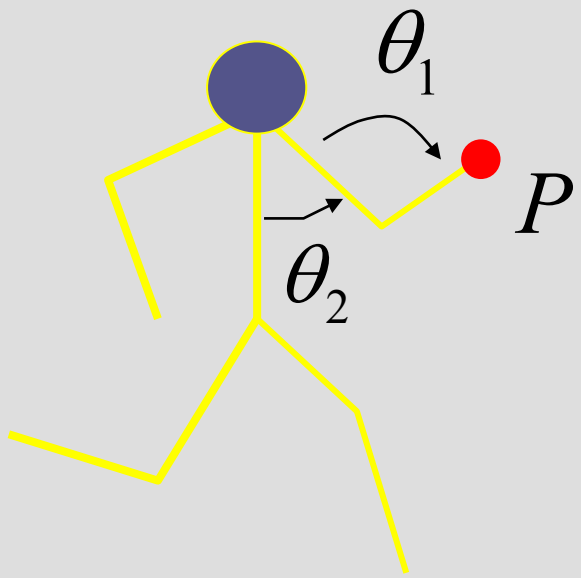
Dynamics

Describes the positions of the body parts as a function of the body mass and applied forces.



Forward and Inverse Kinematics

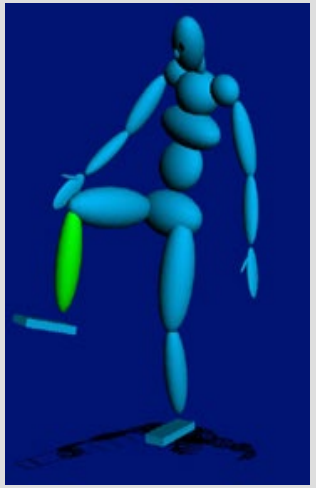
- Forward kinematics
  - joint space → cartesian space $P = f(\theta_1, \theta_2)$
- Inverse kinematics
  - cartesian space → joint space $\theta_1, \theta_2 = f^{-1}(P)$



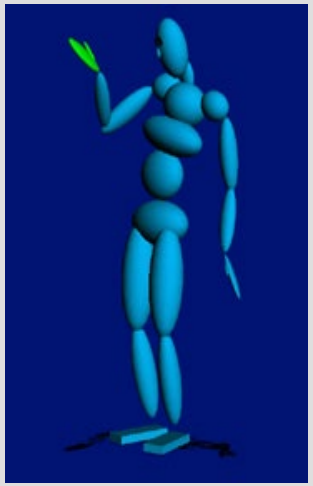
Forward Kinematics

- Describes the positions of the body parts as a function of the joint angles.

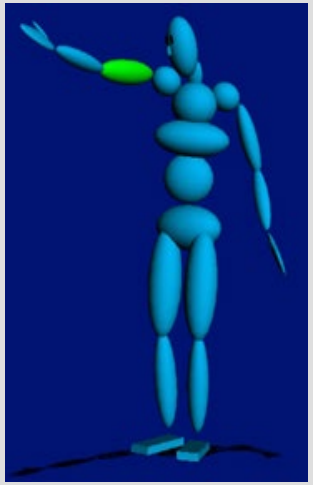
1 DOF: knee



2 DOF: wrist



3 DOF: arm



Forward Kinematics - details

- The local and world matrix construction within the skeleton is an implementation of *forward kinematics*
- Forward kinematics refers to the process of computing world space geometric descriptions (matrices…)

  based on joint DOF values (usually rotation angles and/or translations)

Forward Kinematics

- The following vector represents the array of M joint DOF values:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1 & \phi_2 & \dots & \phi_M \end{bmatrix}$$

- The following vector represents an array of N DOFs that describe the end effector in world space:

$$\mathbf{e} = \begin{bmatrix} e_1 & e_2 & \dots & e_N \end{bmatrix}$$

- The forward kinematic function f() computes the world space end effector DOFs from the joint DOFs:

$$\mathbf{e} = f\left(\mathbf{\Phi}\right)$$

Forward Kinematics

For example,

If our end effector is a full joint with orientation,

e would contain 6 DOFs:

3 translations and 3 rotations.

If we were only concerned with the end effector position, **e** would just contain the 3 translations.

Inverse Kinematics

- The goal of inverse kinematics is to compute the vector of joint DOFs that will cause the end effector to reach some desired goal state

- In other words, it is the inverse of the forward kinematics problem

$$\mathbf{\Phi} = f^{-1}(\mathbf{e})$$

Inverse Kinematics

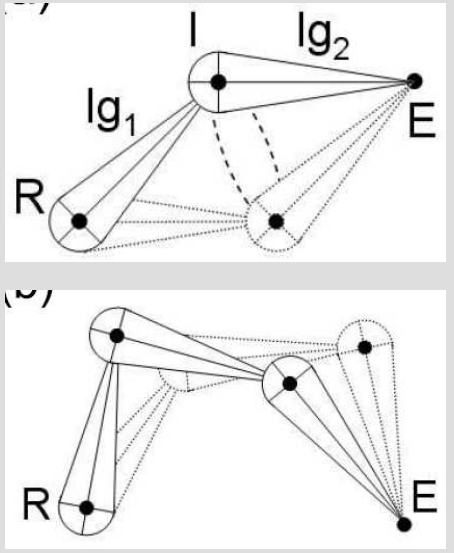
- Crucial in computer animation
- Needed every time a character lifts, touches, or manipulates a virtual object
- Interesting and difficult problem
  - Infinite solutions
- Open research question: finding a configuration that solves IK and looks natural

IK Desired Features

- Generality
  - handles different types of objects / systems
  - handles different types of constraints
  - handles different numbers of constraints
- Robustness (finds a solution if there is one)
  - singularity free
- Chooses good answers (if there are many)
  - provides a good mechanism for saying what is desirable
  - naturalness
  - joint limits
- Speed
- Repeatability / predictability

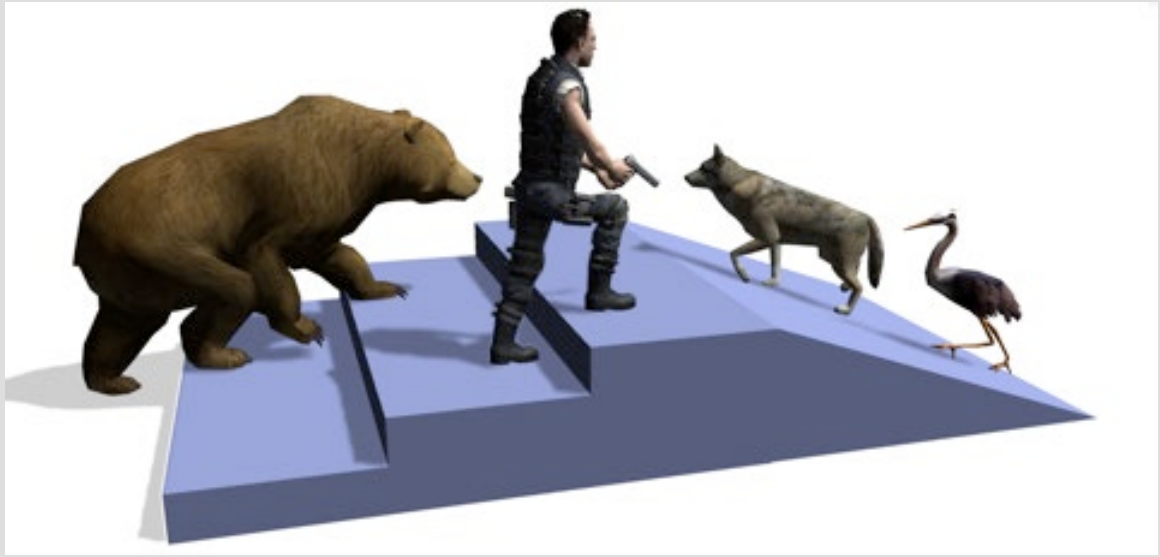
Inverse Kinematics Issues

- IK is challenging because while f() may be relatively easy to evaluate, $f^{-1}()$ usually isn't
  - e.g., f() is often highly non-linear
- For one thing, there may be several possible solutions for **Φ**, or there may be no solutions
  - Underactuated: number of DOFs > number of constraints
- Even if there is a solution, it may require complex and expensive computations to find it
- As a result, there are many different approaches to solving IK problems



Inverse Kinematics

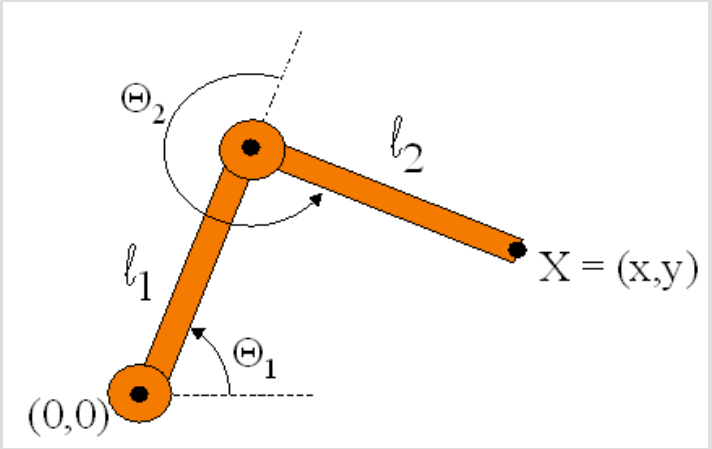
- Goals
  - Keep end of limb fixed while body moves
  - Position end of limb by direct manipulation
  - (More general: arbitrary constraints)



FK vs. IK

Forwards Kinematics

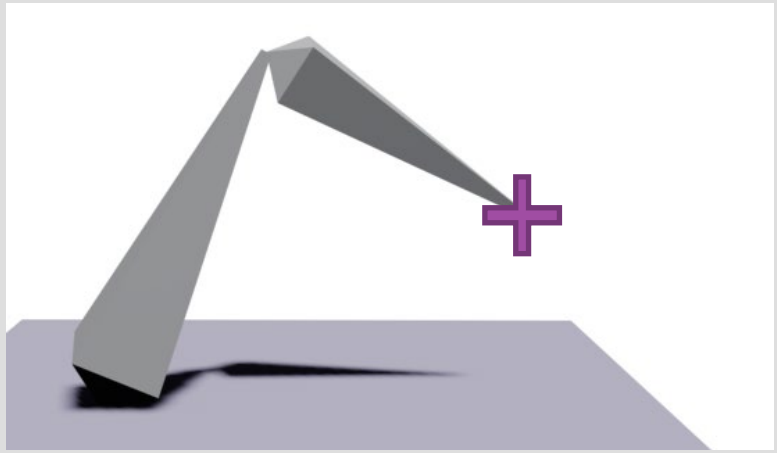
- Animator specifies joint angles
- Computer finds position of end-effector: X



$$X = \begin{pmatrix} l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \end{pmatrix}$$

Inverse Kinematics

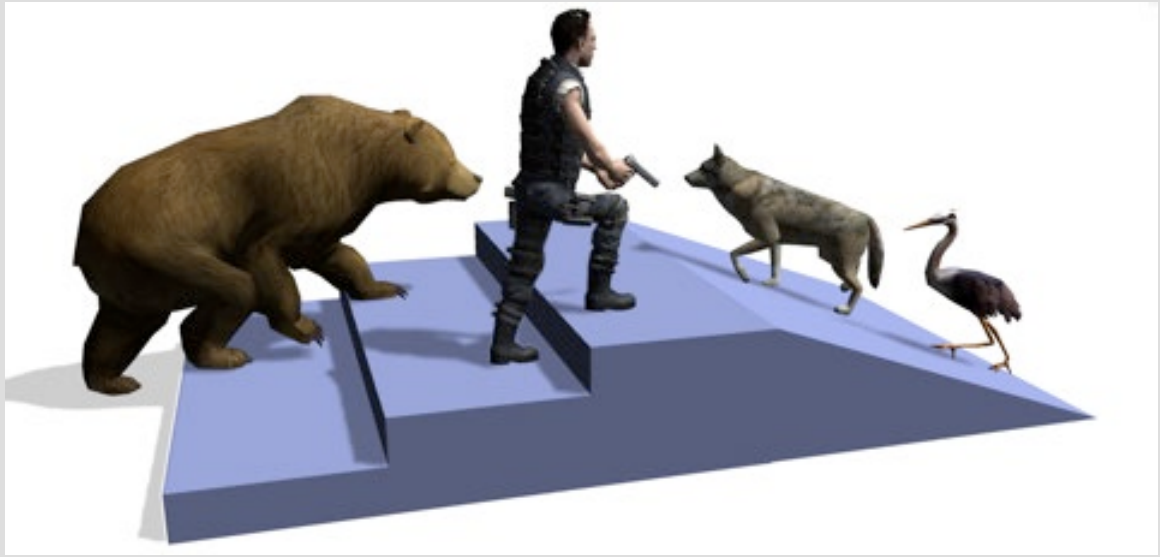
- Animator specifies end-effector positions
- Computer finds joint angles $\Theta=(\Theta_1,\Theta_2,\ldots,\Theta_n)$



$$\theta_1, \theta_2 = f(X)$$

Inverse Kinematics

- Solutions
  - Analytical
  - Numerical
  - Newer: *Example-Based*

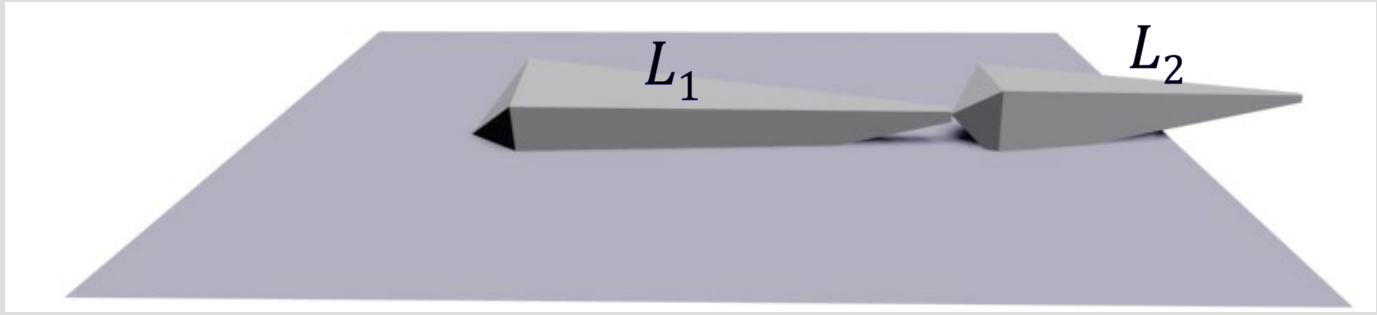


Analytical vs. Numerical Solutions

- One major way to classify IK solutions is into analytical and numerical methods
  - Analytical methods: attempt to mathematically solve an **exact solution** by directly inverting the forward kinematics equations. This is only possible on relatively simple chains.
  - Numerical methods: use **approximation** and **iteration** to converge on a solution. They tend to be more expensive, but far more general purpose.
    - Jacobian methods
    - Method of Cyclic Coordinate Descent (CCD)

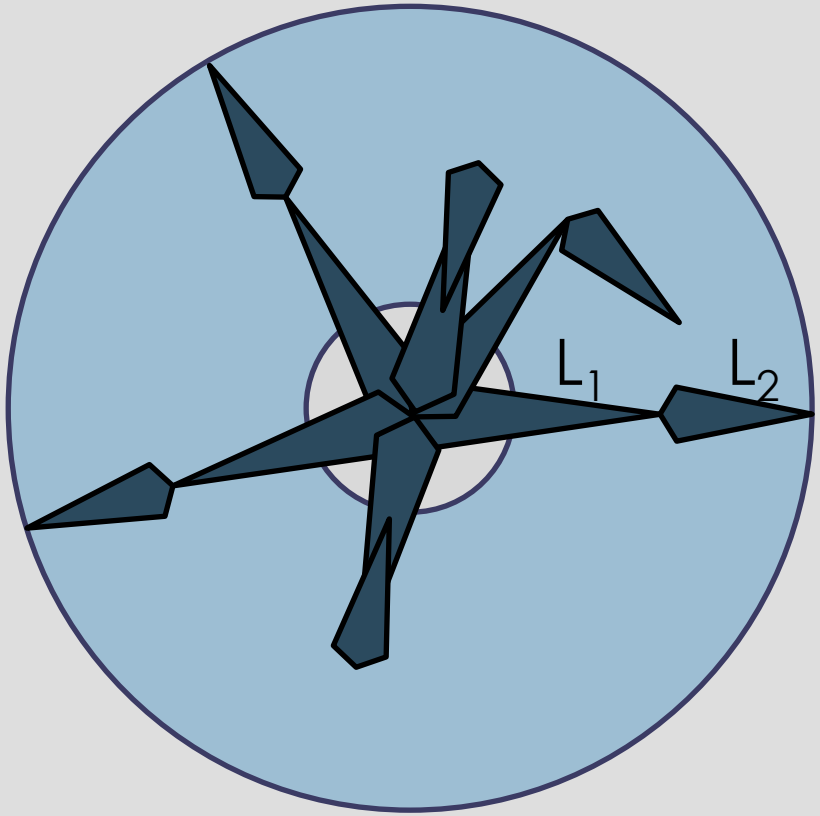
Analytical solution

- For simple mechanisms:
  - joint values can be determined analytically by inspecting the geometry of the linkage
- Consider a simple 2-link arm in 2-D space with 2 rotational DOF
  - Reachable workspace: If a position is fixed for base, any position beyond |L1 – L2| and within |L1 + L2| can be reached



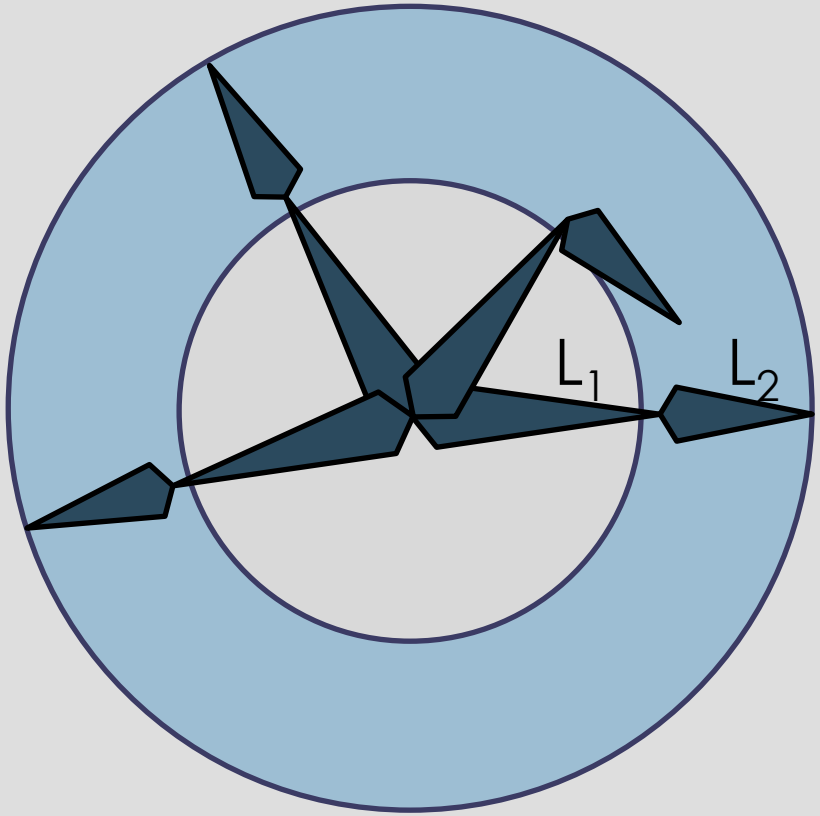
Analytical solution

- Reachable workspace: If a position is fixed for base, any position beyond |L1 – L2| and within |L1 + L2| can be reached

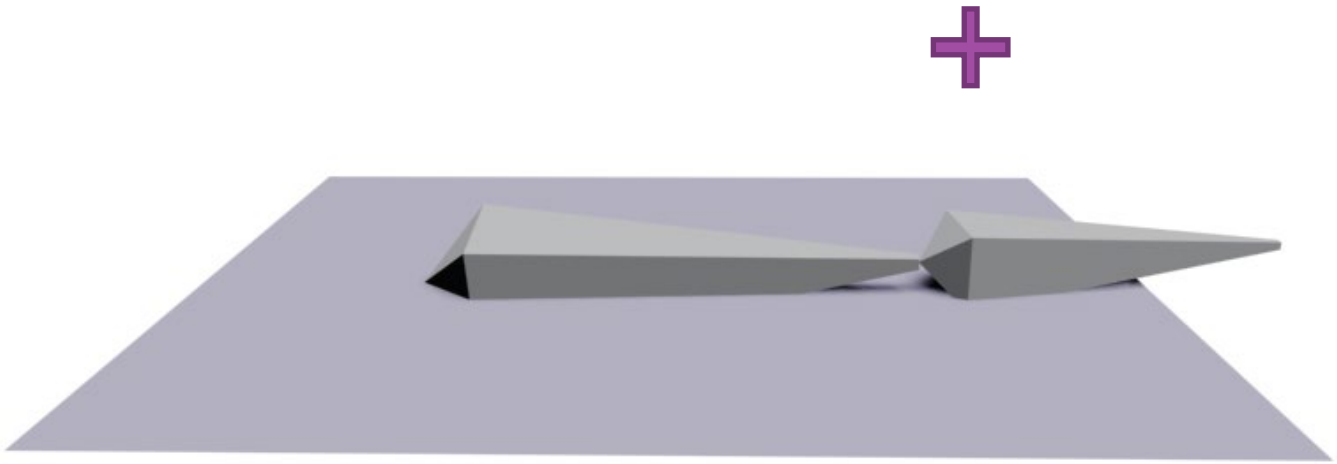


Analytical solution

- Reachable workspace

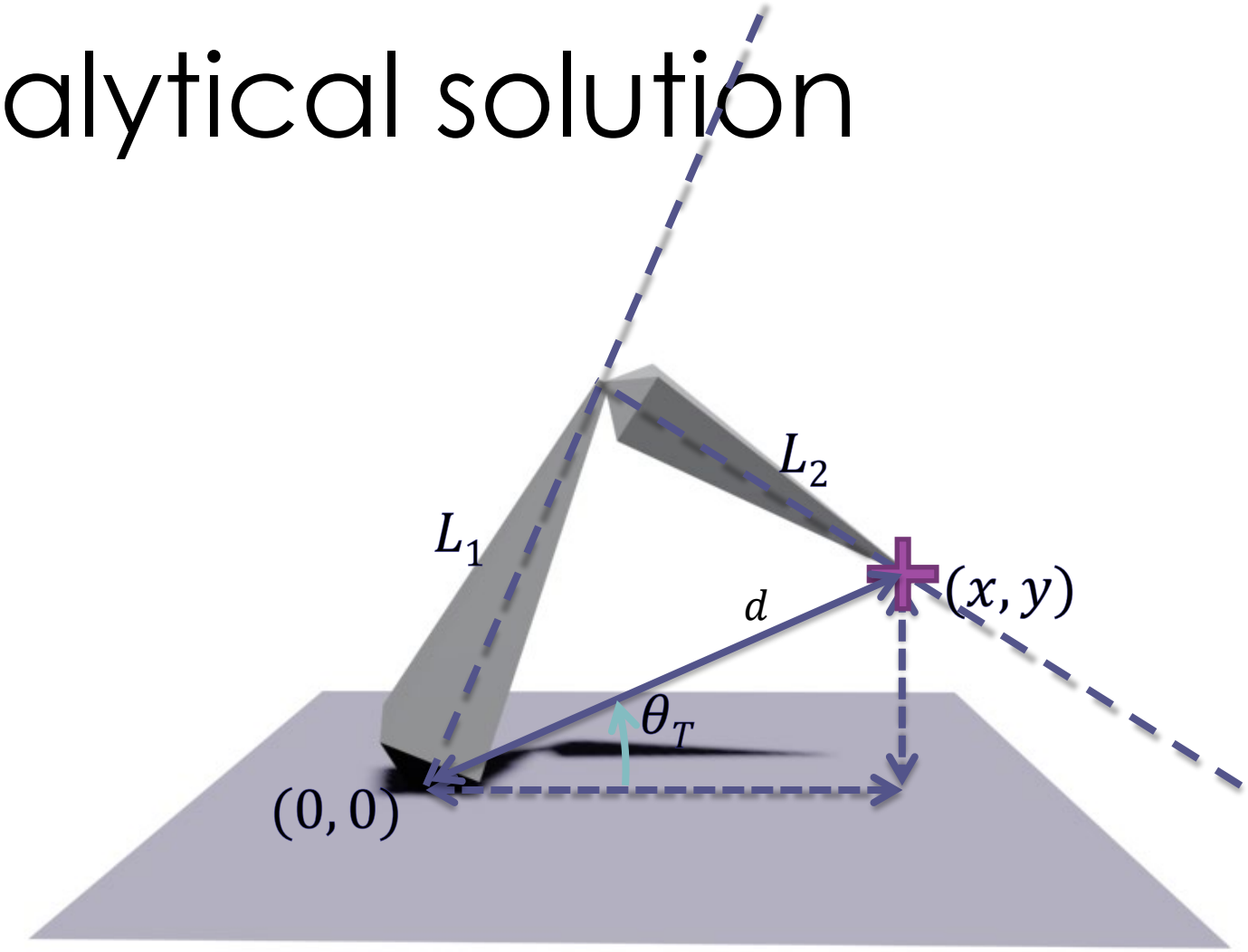


2D Analytical solution

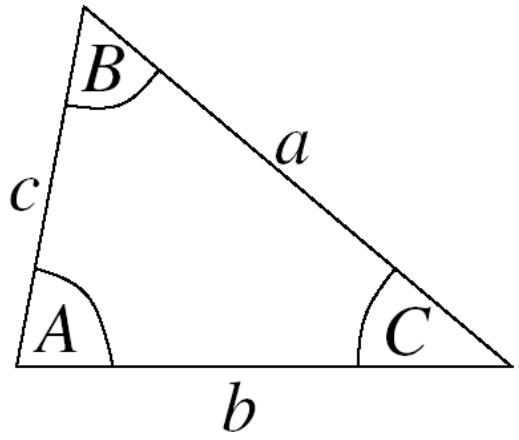


2D Analytical solution

$$d = \sqrt{x^2 + y^2}$$



Cosine and Sine rules



Sine Rule

$$\frac{a}{\sin(A)} = \frac{b}{\sin(B)} = \frac{c}{\sin(C)} \quad \text{or} \quad \frac{\sin(A)}{a} = \frac{\sin(B)}{b} = \frac{\sin(C)}{c}$$

(for finding sides) (for finding angles)

Cosine Rule

$$a^2 = b^2 + c^2 - 2bc\cos(A) \quad \text{or} \quad \cos(A) = \frac{b^2 + c^2 - a^2}{2bc}$$

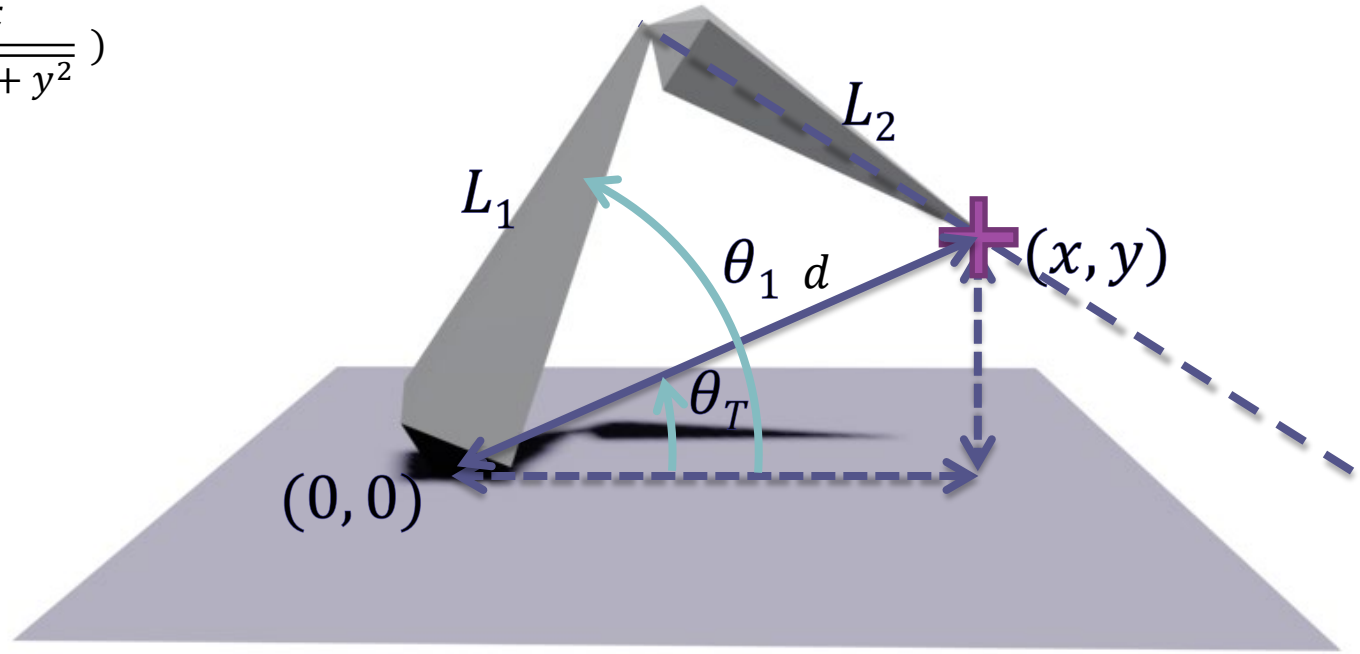
(for finding sides) (for finding angles)

2D Analytical solution

$$d = \sqrt{x^2 + y^2}$$

$$cos(\theta_T) = \frac{x}{\sqrt{x^2+y^2}}$$

$$\theta_T = cos^{-1}\left(\frac{x}{\sqrt{x^2 + y^2}}\right)$$



2D Analytical solution

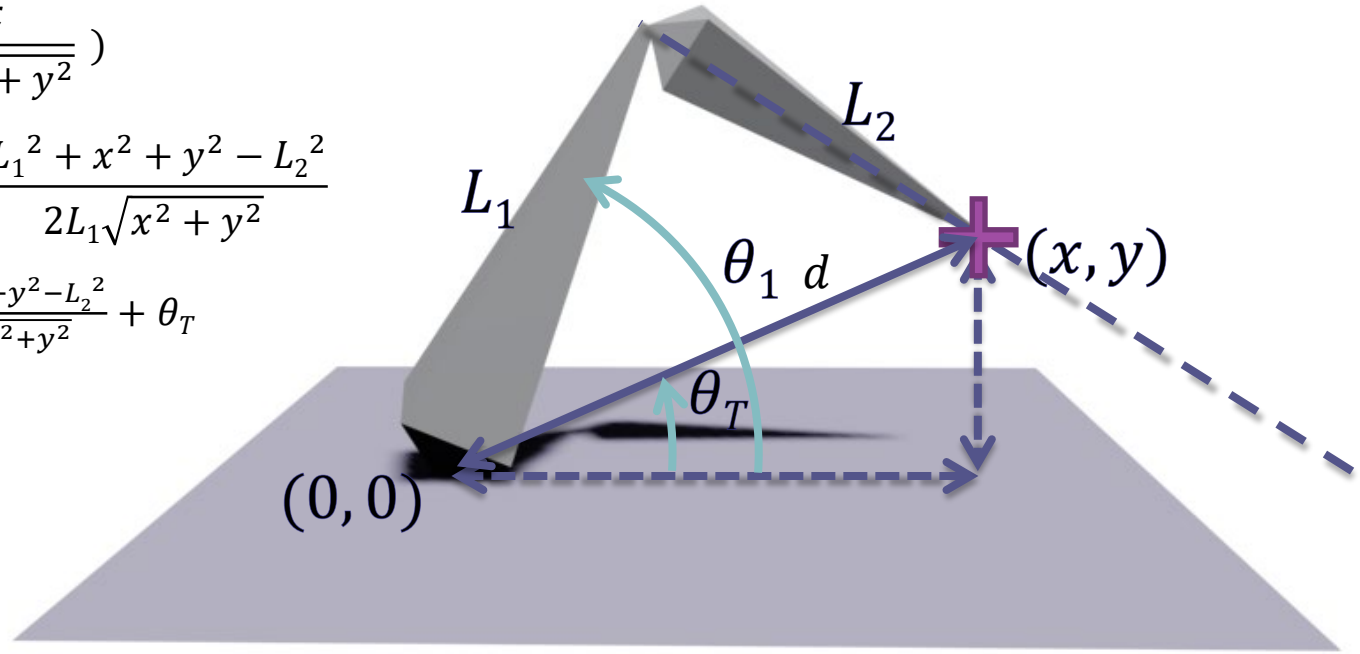
$$d = \sqrt{x^2 + y^2}$$

$$cos(\theta_T) = \frac{x}{\sqrt{x^2+y^2}}$$

$$\theta_T = cos^{-1}\left(\frac{x}{\sqrt{x^2 + y^2}}\right)$$

$$cos(\theta_1 - \theta_T) = \frac{{L_1}^2 + x^2 + y^2 - {L_2}^2}{2L_1\sqrt{x^2 + y^2}}$$

$$\theta_1 = \cos^{-1}\frac{{L_1}^2+x^2+y^2-{L_2}^2}{2L_1\sqrt{x^2+y^2}} + \theta_T$$



2D Analytical solution

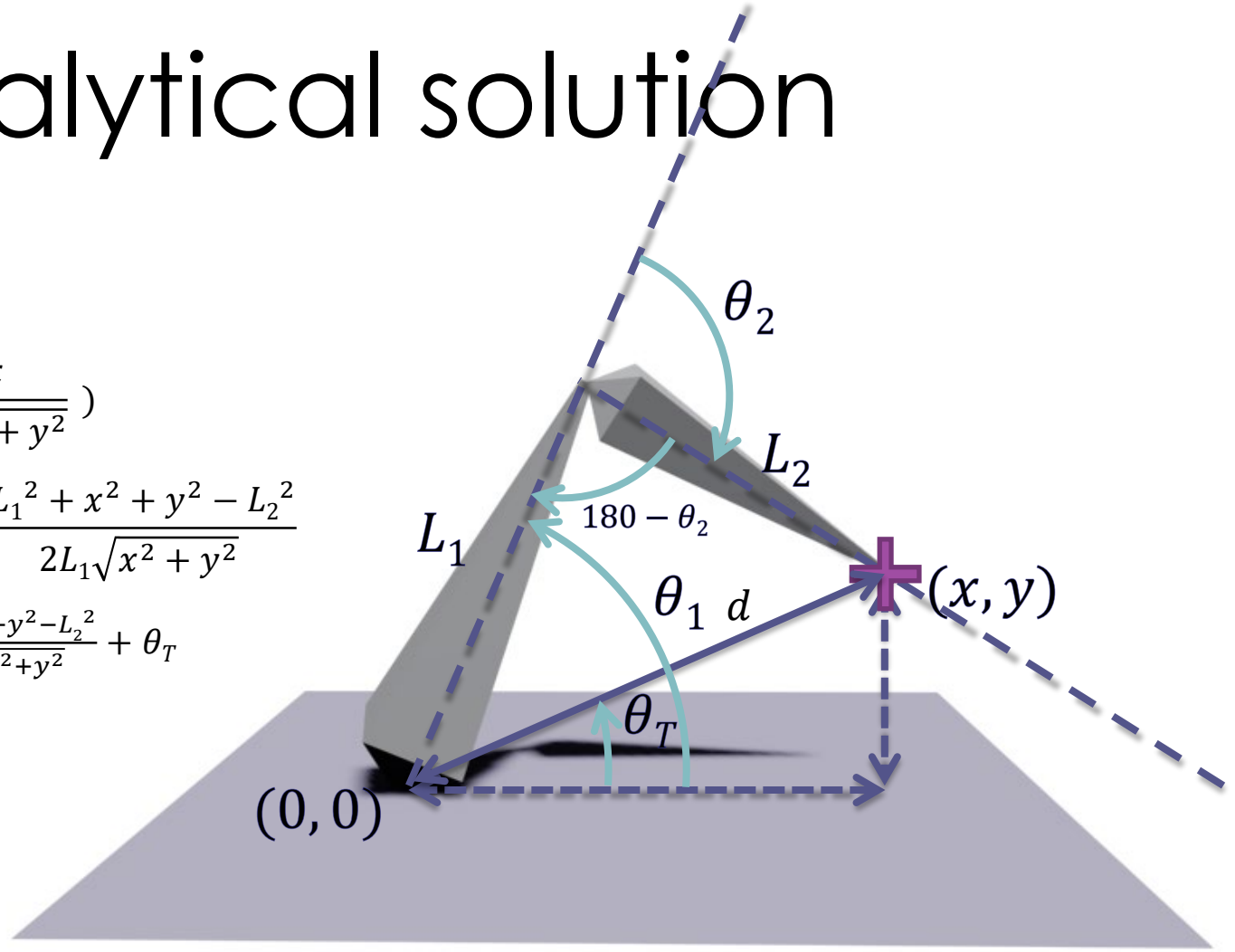
$$d = \sqrt{x^2 + y^2}$$

$$cos(\theta_T) = \frac{x}{\sqrt{x^2+y^2}}$$

$$\theta_T = cos^{-1}\left(\frac{x}{\sqrt{x^2 + y^2}}\right)$$

$$cos(\theta_1 - \theta_T) = \frac{{L_1}^2 + x^2 + y^2 - {L_2}^2}{2L_1\sqrt{x^2 + y^2}}$$

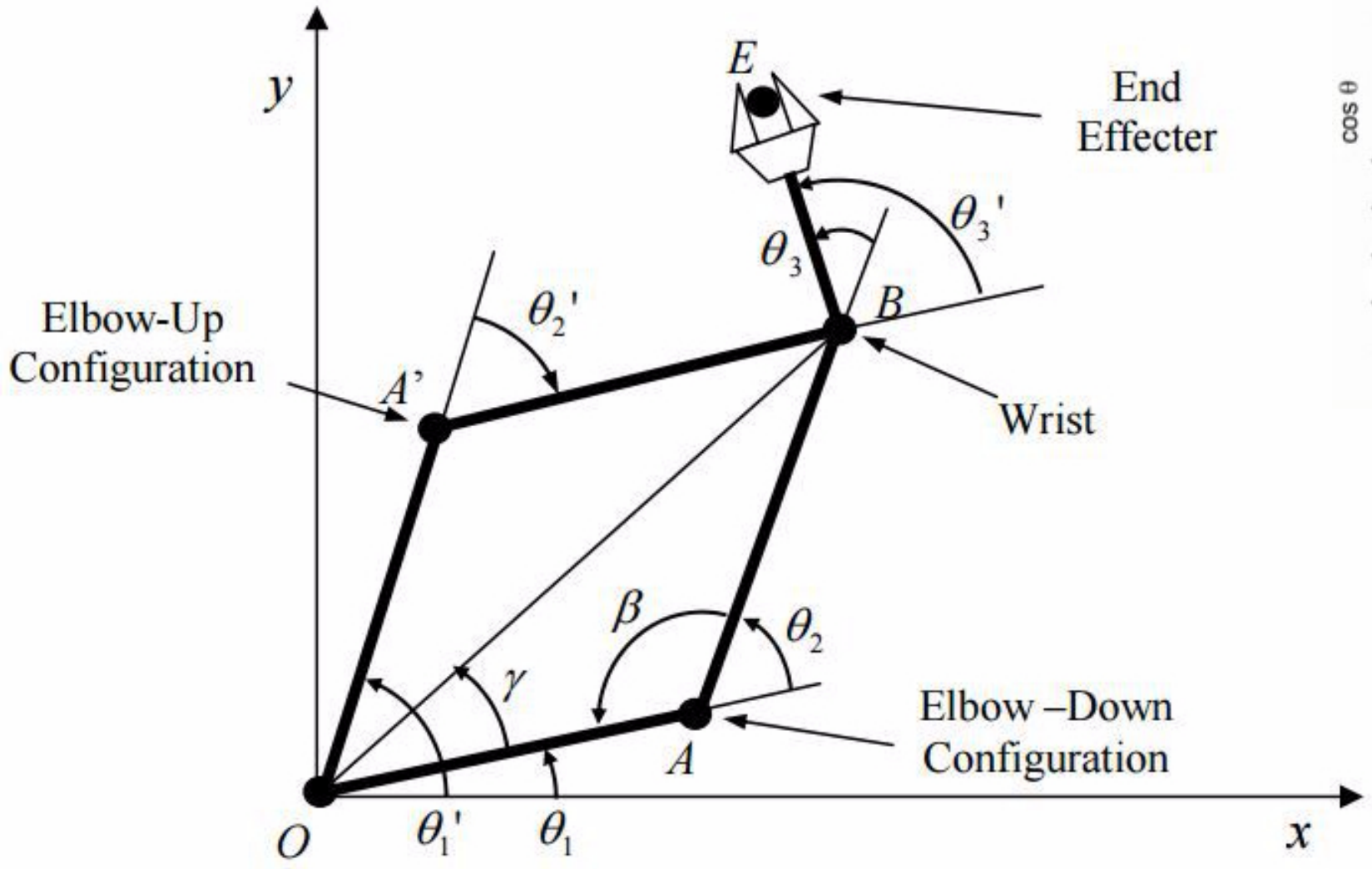
$$\theta_1 = \cos^{-1}\frac{{L_1}^2+x^2+y^2-{L_2}^2}{2L_1\sqrt{x^2+y^2}} + \theta_T$$

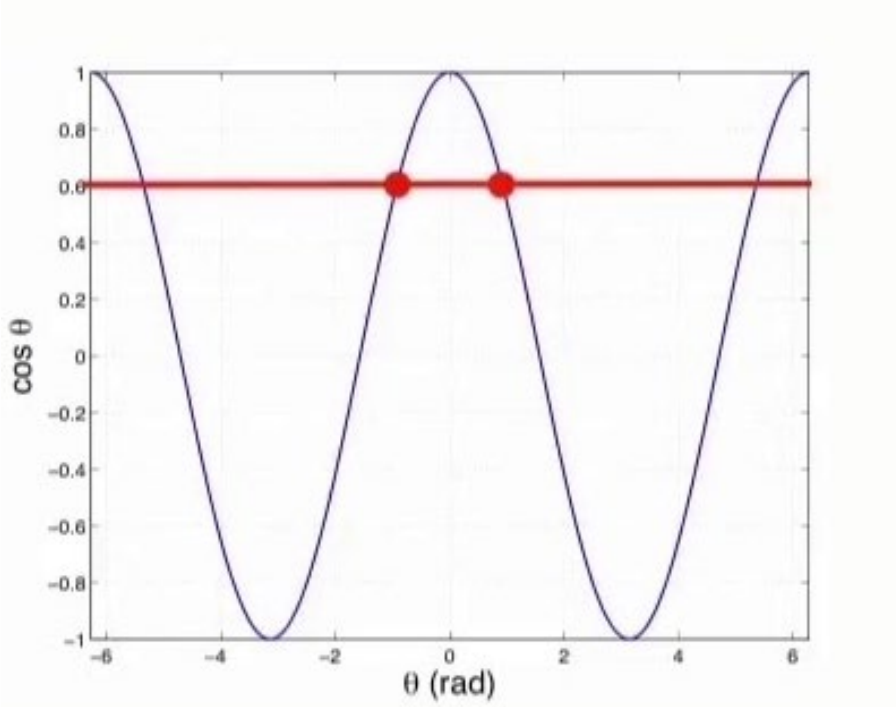


$$\cos(180 - \theta_2) = -\cos\theta_2 = \frac{{L_1}^2 + {L_2}^2 - (x^2 + y^2)}{2L_1L_2}$$

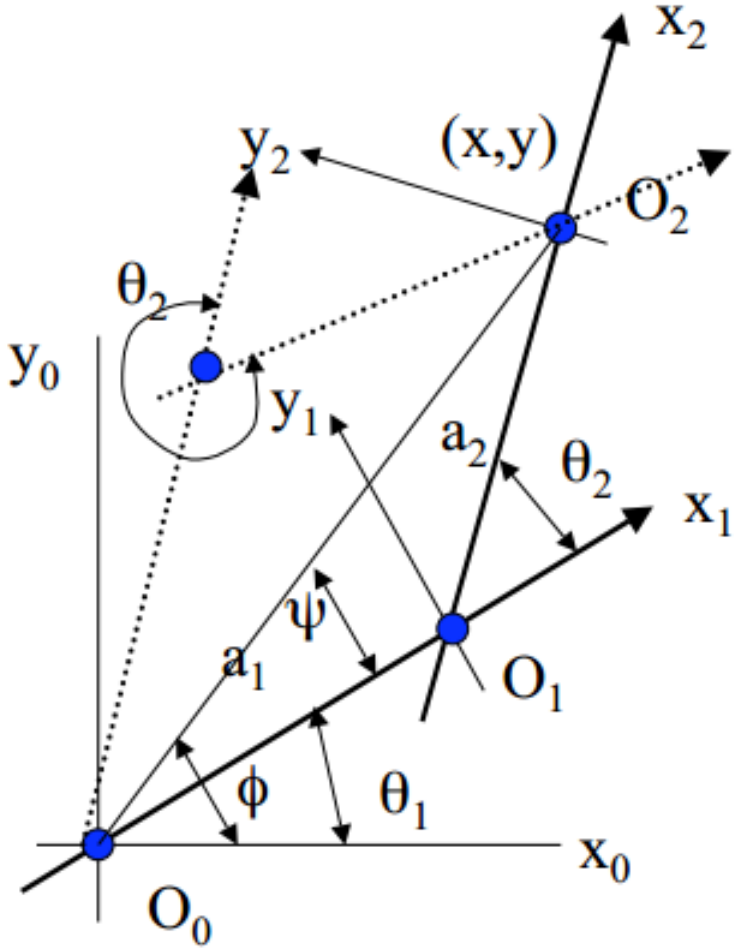
$$\theta_2 = \cos^{-1}\frac{{L_1}^2 + {L_2}^2 - x^2 + y^2}{2L_1L_2}$$

Elbow up & Elbow Down





Elbow up & Elbow Down



$$x^2 + y^2 = a_1^2 + a_2^2 - 2a_1a_2\cos(\pi - \theta_2)$$

$$\cos\theta_2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2}$$

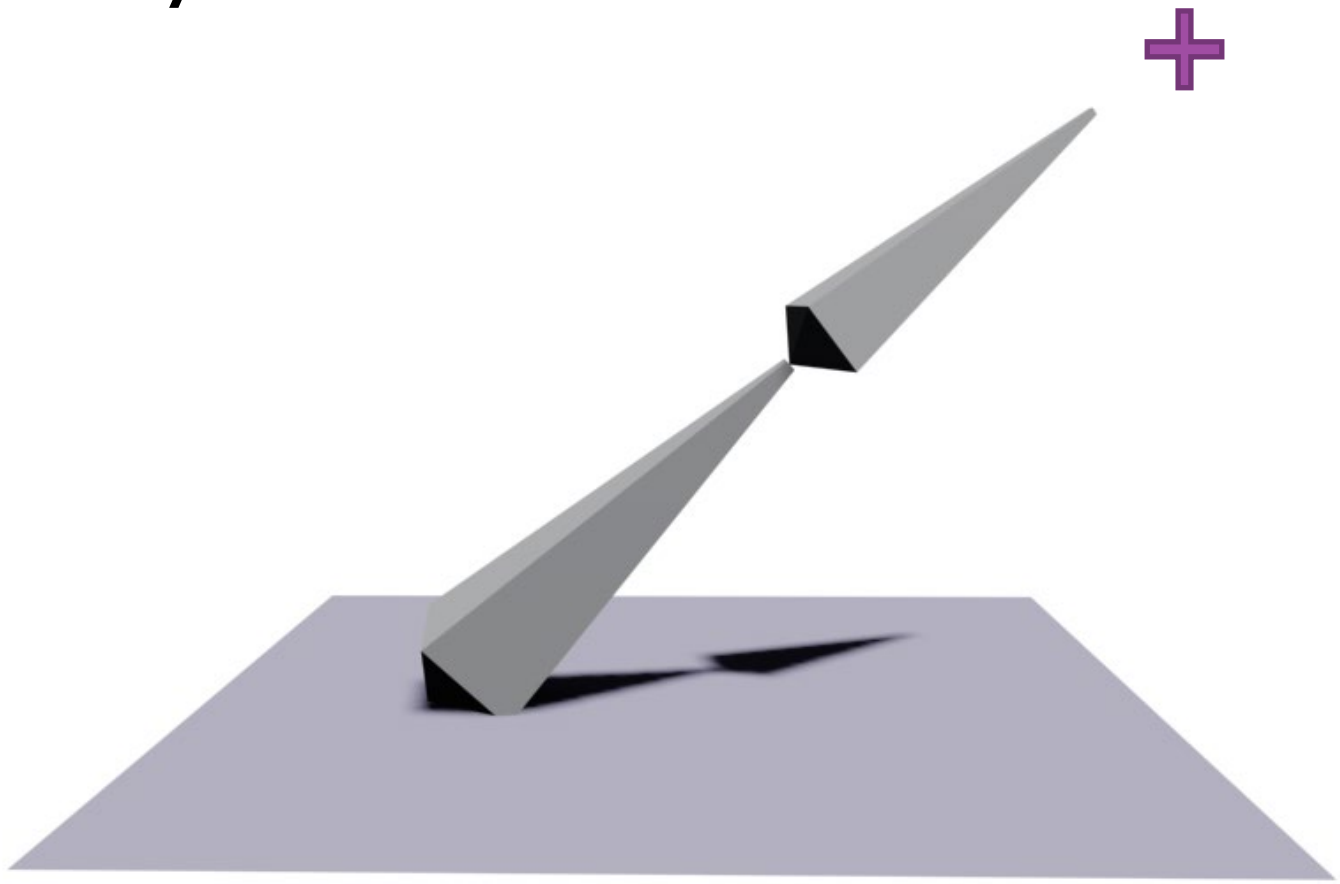
for greater accuracy

$$\tan^2\frac{\theta_2}{2} = \frac{1-\cos\theta_2}{1+\cos\theta_2} = \frac{2a_1a_2 - x^2 - y^2 + a_1^2 + a_2^2}{2a_1a_2 + x^2 + y^2 - a_1^2 - a_2^2}$$

$$= \frac{(a_1 + a_2)^2 - (x^2 + y^2)}{(x^2 + y^2) - (a_1 - a_2)^2}$$

$$\theta_2 = \pm 2\tan^{-1}\sqrt{\frac{(a_1 + a_2)^2 - (x^2 + y^2)}{(x^2 + y^2) - (a_1 - a_2)^2}}$$

Singularity

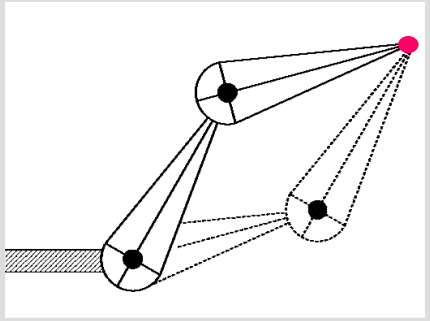


What makes IK hard?

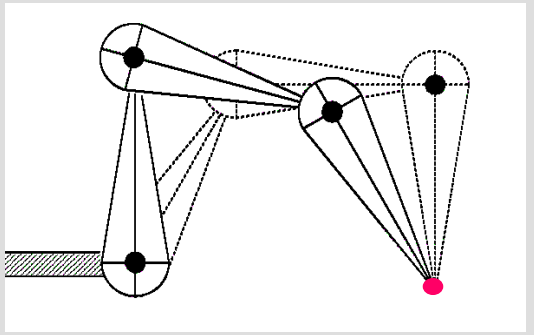
- Not always a unique solution
- Not always well-behaved
- Nonlinear problem
- Joint limits

Not always a unique solution

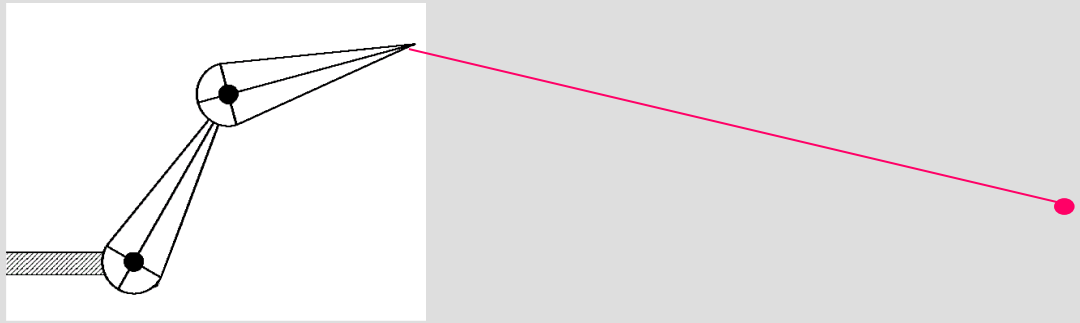
- Disjoint solutions:



- Continuum of solutions:

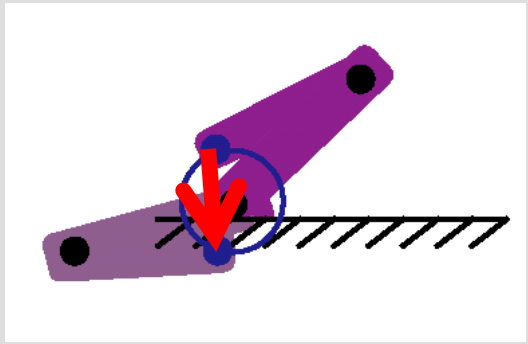


- No solution:

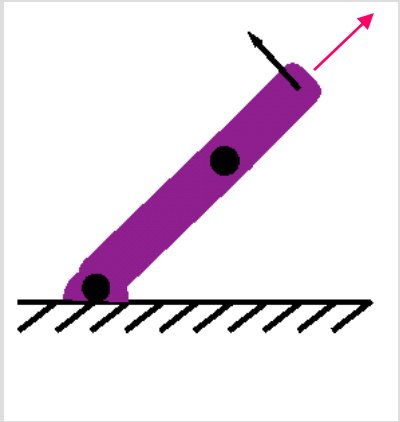


Not always well-behaved

- Small change in $\boldsymbol{X}$ can cause big change in $\boldsymbol{\theta}$

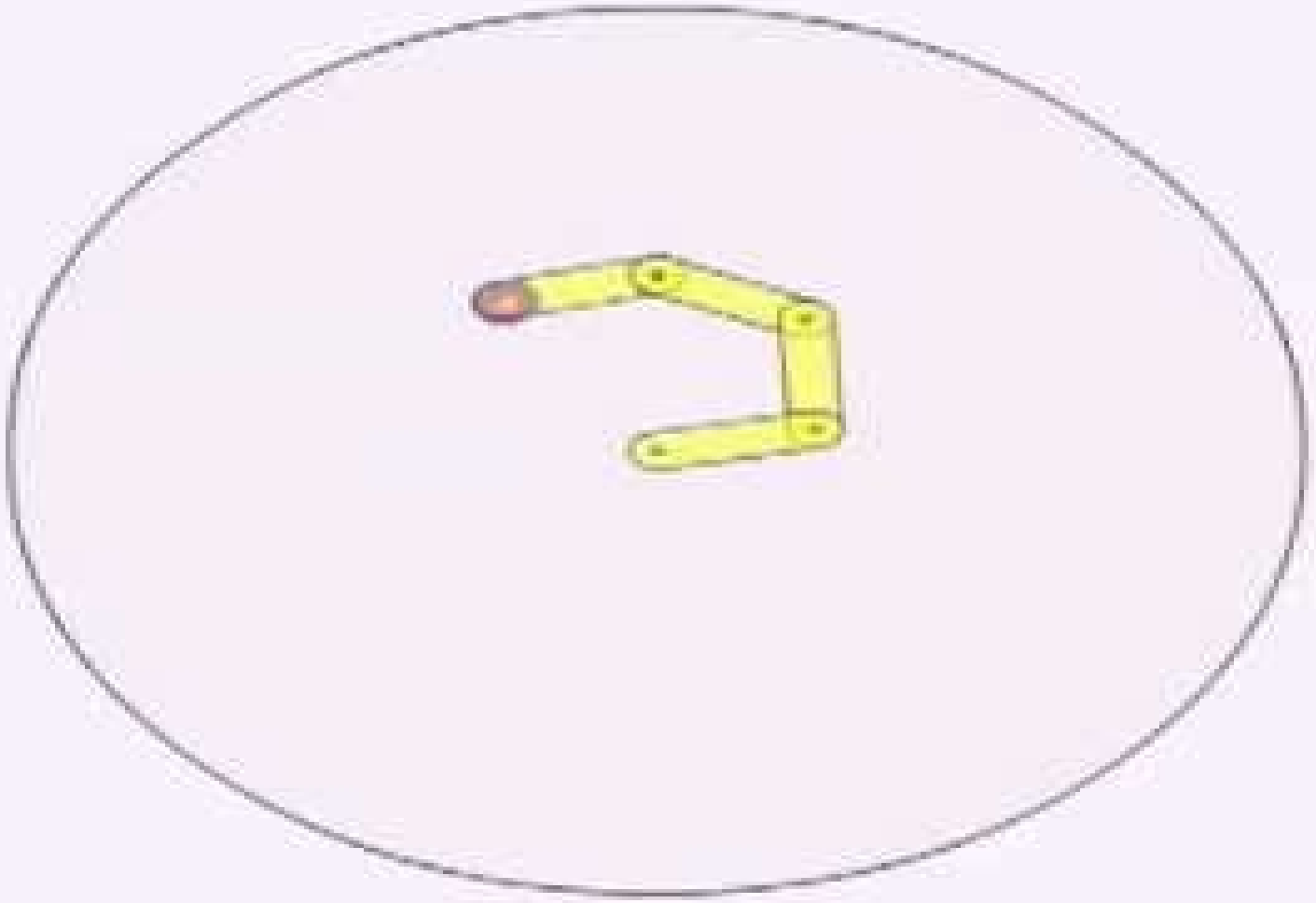


- Changing $\boldsymbol{\theta}$ might not move end towards $\boldsymbol{X}$



Reading

- Chapter 5 in *Computer Animation Algorithms and Techniques*, Rick Parent
- *See "Further Reading" on Blackboard*



Display Arm

Increase Iteration Per Cycl

Please hit enter after manua

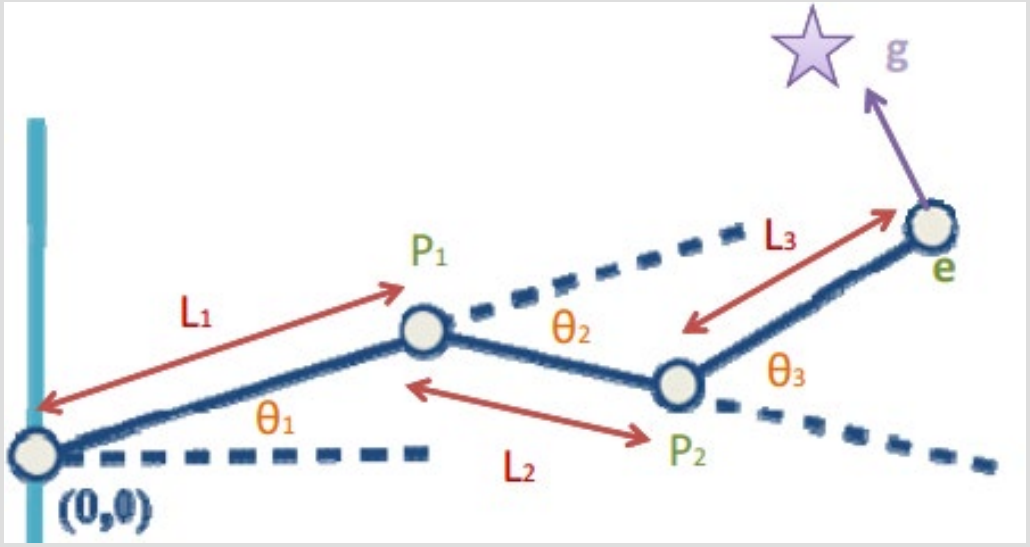
Internal Variables

Analytical vs. Numerical Solutions

- One major way to classify IK solutions is into analytical and numerical methods
  - **Analytical methods:** attempt to mathematically solve an exact solution by directly inverting the forward kinematics equations. This is only possible on relatively simple chains.
  - **Numerical methods:** use approximation and iteration to converge on a solution. They tend to be more expensive, but far more general purpose.
    - Jacobian methods
    - Method of Cyclic Coordinate Descent (CCD)
    - Forward backward reaching

Numerical Solutions

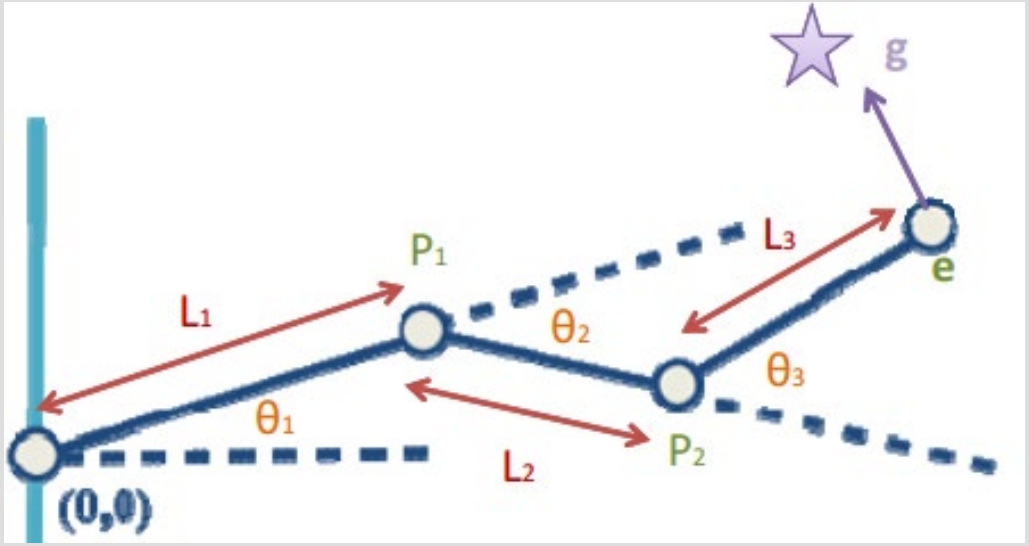
- When linkage is too complex for analytic methods
- At each time step, determine changes to joint angles that take the end effector towards goal configuration
- Need to re-compute at each time step



The Jacobian

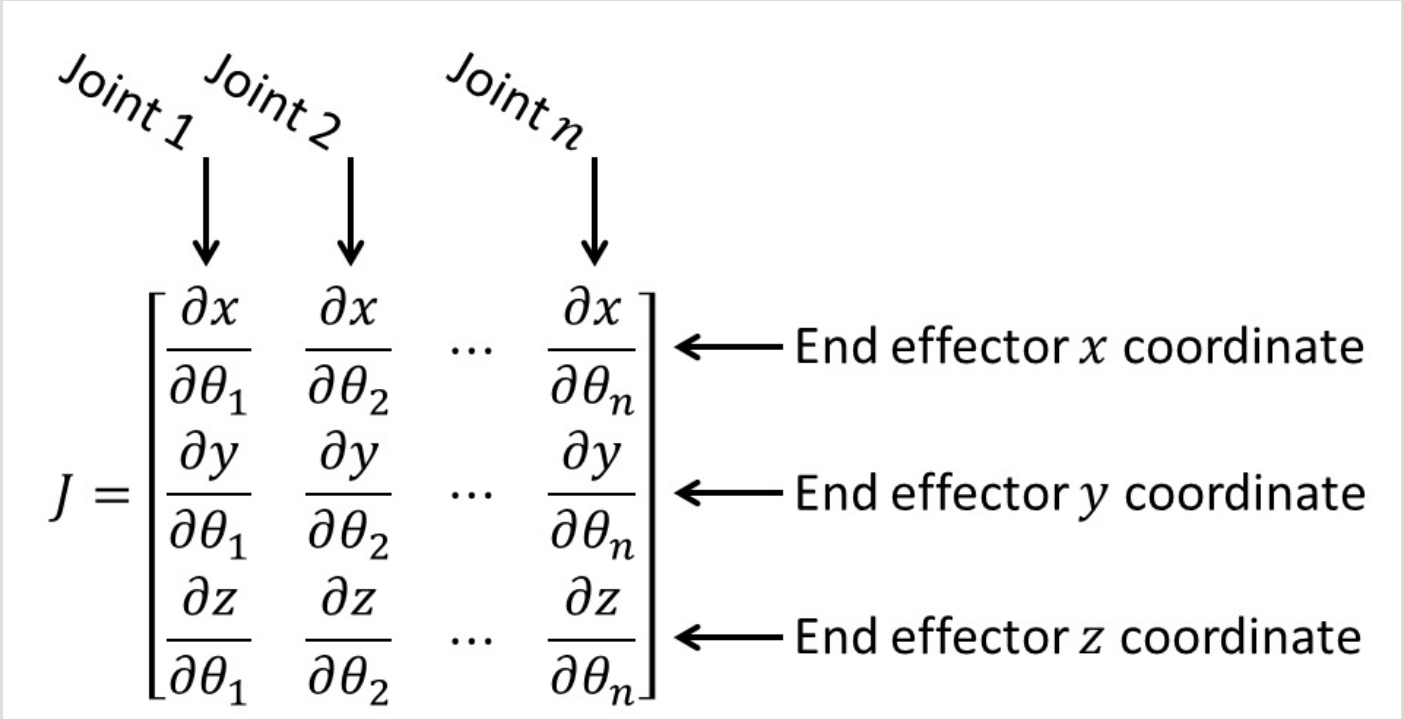
- The Jacobian poses the problem in matrix format
- For animation, we need a Jacobian for every frame

$$J = \begin{pmatrix} \frac{\partial f_x}{\partial \theta_1} & \frac{\partial f_x}{\partial \theta_2} & \frac{\partial f_x}{\partial \theta_3} \\ \frac{\partial f_y}{\partial \theta_1} & \frac{\partial f_y}{\partial \theta_2} & \frac{\partial f_y}{\partial \theta_3} \\ \frac{\partial f_z}{\theta_1} & \frac{\partial f_z}{\partial \theta_2} & \frac{\partial f_z}{\partial \theta_3} \end{pmatrix}$$



The Jacobian

- More generally:



The Jacobian

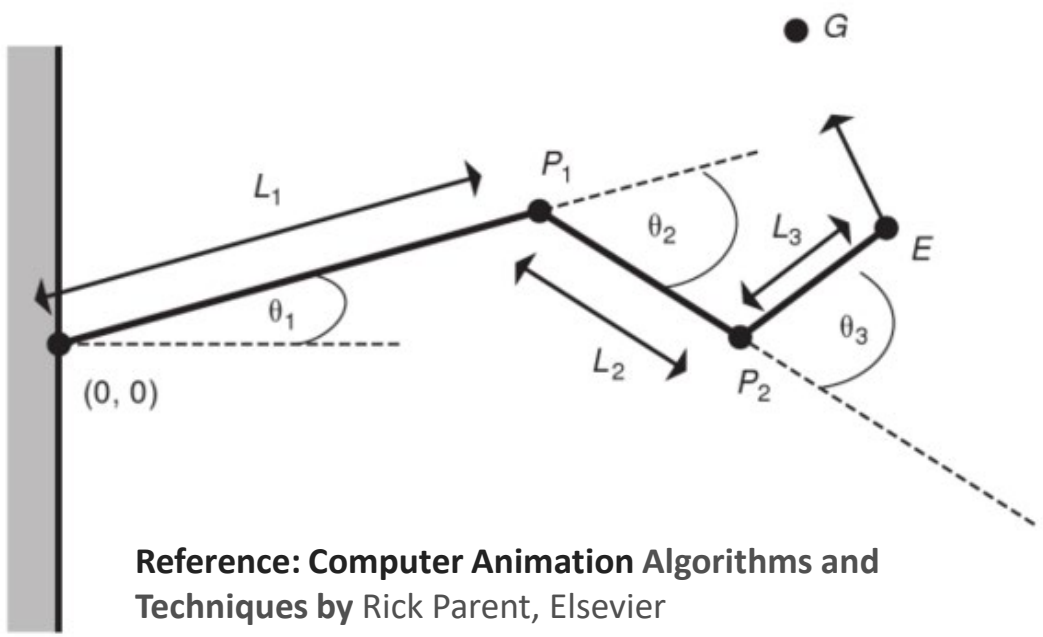
- Used to find the correct joint angle increments that will lead us to the final end effector configuration
- Jacobian matrix is a matrix of partial derivatives
  - Each entry shows how much the change in an input parameter effects an output parameter
- Determine the best linear combination of velocities induced by the joints that would result in the desired velocities of the end effector

$$J = \begin{pmatrix} \frac{\partial f_x}{\partial \theta_1} & \frac{\partial f_x}{\partial \theta_2} & \frac{\partial f_x}{\partial \theta_3} \\ \frac{\partial f_y}{\partial \theta_1} & \frac{\partial f_y}{\partial \theta_2} & \frac{\partial f_y}{\partial \theta_3} \\ \frac{\partial f_z}{\partial \theta_1} & \frac{\partial f_z}{\partial \theta_2} & \frac{\partial f_z}{\partial \theta_3} \end{pmatrix}$$

How to Compute J?

- Analytically

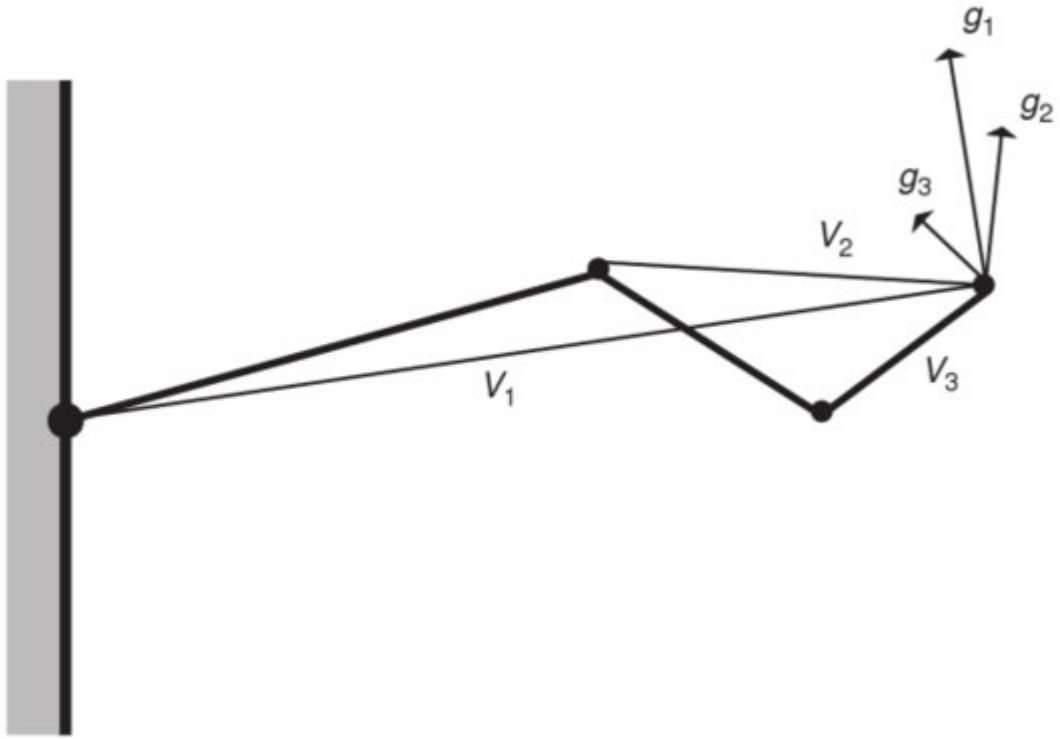
$$J = \begin{bmatrix} ((0,0,1) \times E)_x & ((0,0,1) \times (E - P_1)_x & ((0,0,1) \times (E - P_2)_x \\ ((0,0,1) \times E)_y & ((0,0,1) \times (E - P_1)_y & ((0,0,1) \times (E - P_2)_y \\ ((0,0,1) \times E)_z & ((0,0,1) \times (E - P_1)_z & ((0,0,1) \times (E - P_2)_z \end{bmatrix}$$



Reference: Computer Animation Algorithms and Techniques by Rick Parent, Elsevier

- A simple 3 revolute joint planar manipulator
- Objective is to move E to G
- Axis of rotation of each joint is perpendicular to us, coming out of the page
- Effect of incremental rotation of each joint can be determined by cross-product of joint axis and vector from joint to the end effector and form columns of J

How to Compute J?



- Notice how the magnitude of each g is a function of the distance between the locations of the joint and end effector
- The desired change to the end effector (V) is the difference between the current position of E and G
- It is set equal to J times a vector of unknowns, which are the joint angle changes

The Jacobian

- Once the Jacobian has been computed, we need to solve the equation:

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

- J relates change of joints to change an end effector
- $\dot{\theta}$ contains the joint angle velocities
- V is the vector of linear and rotational velocities
  - Represents the desired change in the end effector

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

- Inverse Jacobian $J^{-1}V = \dot{\theta}$
  - Not usually square (i.e., nxn), so the inverse may not exist
- Pseudoinverse $J^{+} = J^{T}(JJ^{T})^{-1}$
  - Matrix multiplied by its transpose will be a square matrix whose inverse might exist – acts like the inverse
- Alternatively – just use transpose of Jacobian
  - Jacobian is already an approximation to f()
  - Much faster, but lower quality that using pseudo inverse

Solving IK

- The IK function f is nonlinear
  - This implies that the Jacobian
    can only be used as an approximation that is valid near the current configuration
- So we must repeat
  the process of computing a Jacobian
  and then taking a small step towards the goal until we get close enough

Algorithm of Jacobian Method

while ($\mathbf{e}$ is too far from $\mathbf{g}$){

compute the Jacobian matrix J

compute the pseudoinverse of the Jacobian matrix— $\mathbf{J}^+$

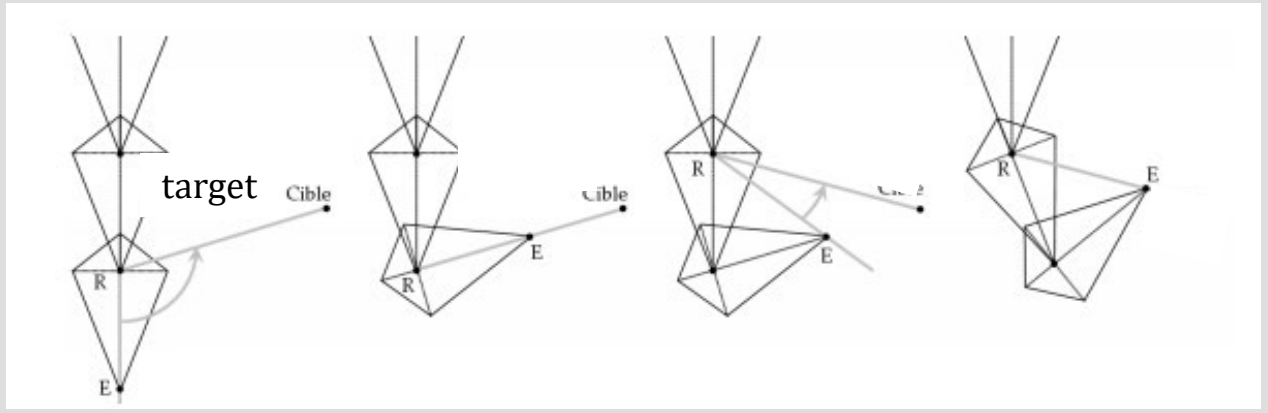
compute change in joint DOFs: $\Delta\boldsymbol{\theta} = \mathbf{J}^+ \cdot \Delta\mathbf{e}$

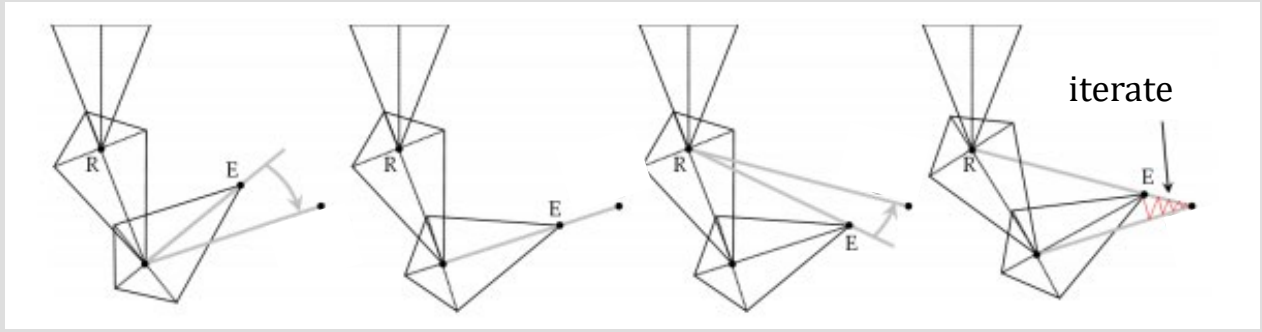
apply the change to DOFs, move a small step of $\boldsymbol{\alpha}\Delta\boldsymbol{\theta}$: $\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{\alpha}\Delta\boldsymbol{\theta}$

}

Cyclic Coordinate Descent

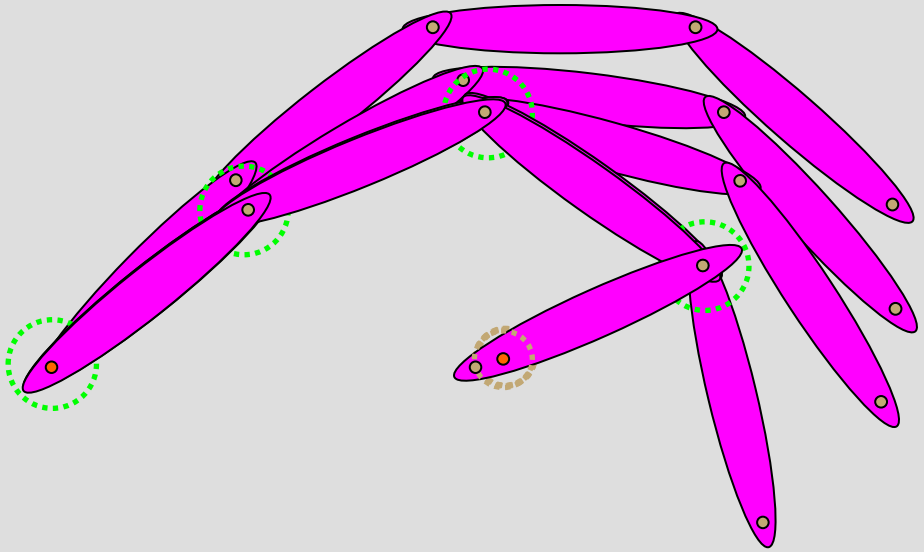
- Iterative approach based on relative orientations
- Minimize distance between goal point and end effector of the chain





Wang and Chen (A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators. IEEE Tr. On Robotics and Automation, 7:489-498, 1991).

CCD (Cyclic Coordinate Descent) Method



L.T. Wang and C.C. Chen. A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators. IEEE Tr. On Robotics and Automation, 7:489-498, 1991.

II. CYCLIC COORDINATE DESCENT

We present below an outline of the CCD algorithm for an n-link chain as shown in Fig. 1, with the following notations:

(x_T, y_T) : Position of the target

(x_E, y_E) : Position of the end-effector

(x_i , y_i) : Pivot point of the i^{th} link, i=1, 2, …, n

$\boldsymbol{t}_i$: Target vector for the i^{th} link = $(x_T - x_i , y_T - y_i)$

$\boldsymbol{e}_i$: End-effector vector for the i^{th} link = $(x_E - x_i , y_E - y_i)$

α_i : Angle between vectors $\boldsymbol{t}_i$ and $\boldsymbol{v}_i$.

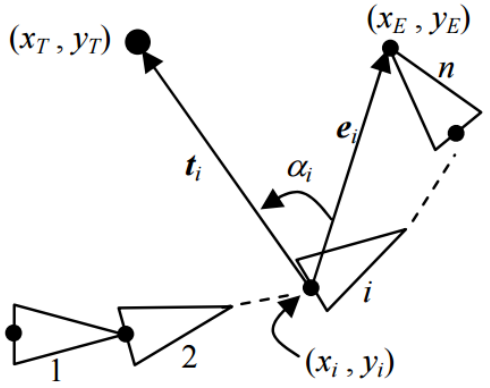


Fig. 1. An n-link joint chain.

The CCD algorithm can be concisely given as follows:

1: Set $i = n$.

2: Compute α_i

3: Rotate i^{th} link by angle α_i so that the end-effector meets the target vector $\boldsymbol{t}_i$

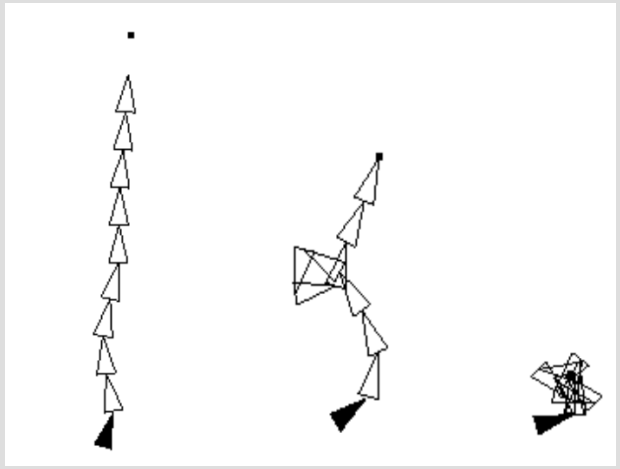
4: Decrement i, and go to step 2 if $i > 0$.

5: Repeat steps 1 to 4 until target position is reached. We count each repetition of the above steps as one iteration.

Cyclic Coordinate Descent

- Computationally fast
- Algorithmically simple
- Straight-forward technique
- No singularity problem

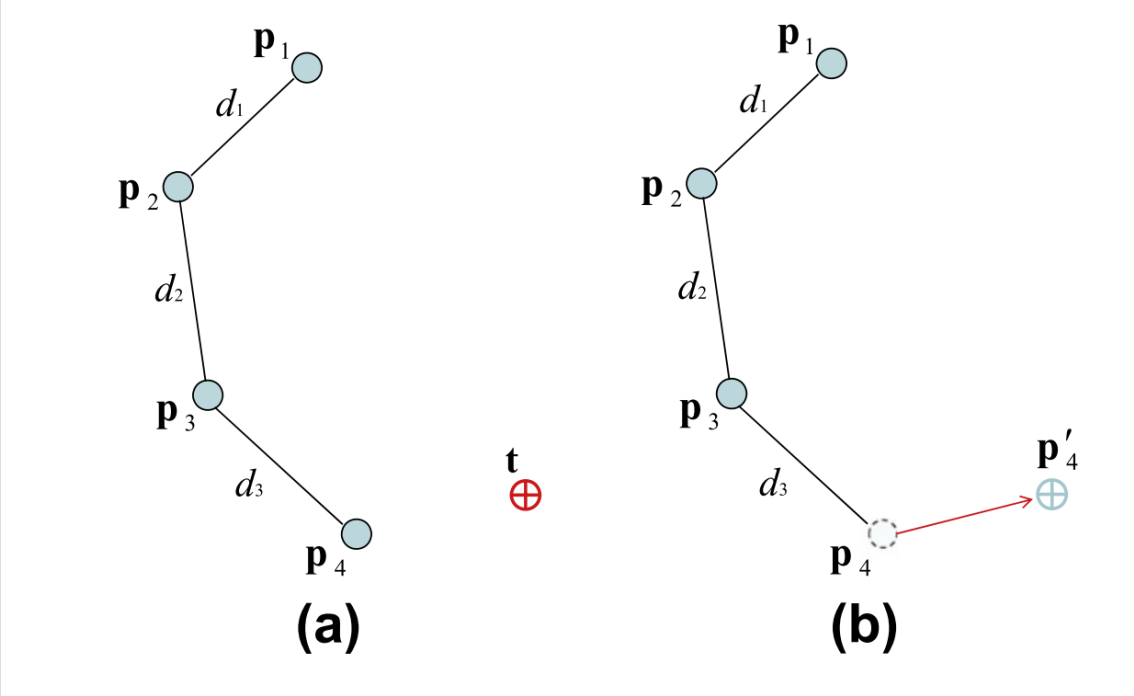
- Unrealistic results



http://thewanderingtech.50webs.com/Flash/IK%20Comparison%20Application/IKCompare.html

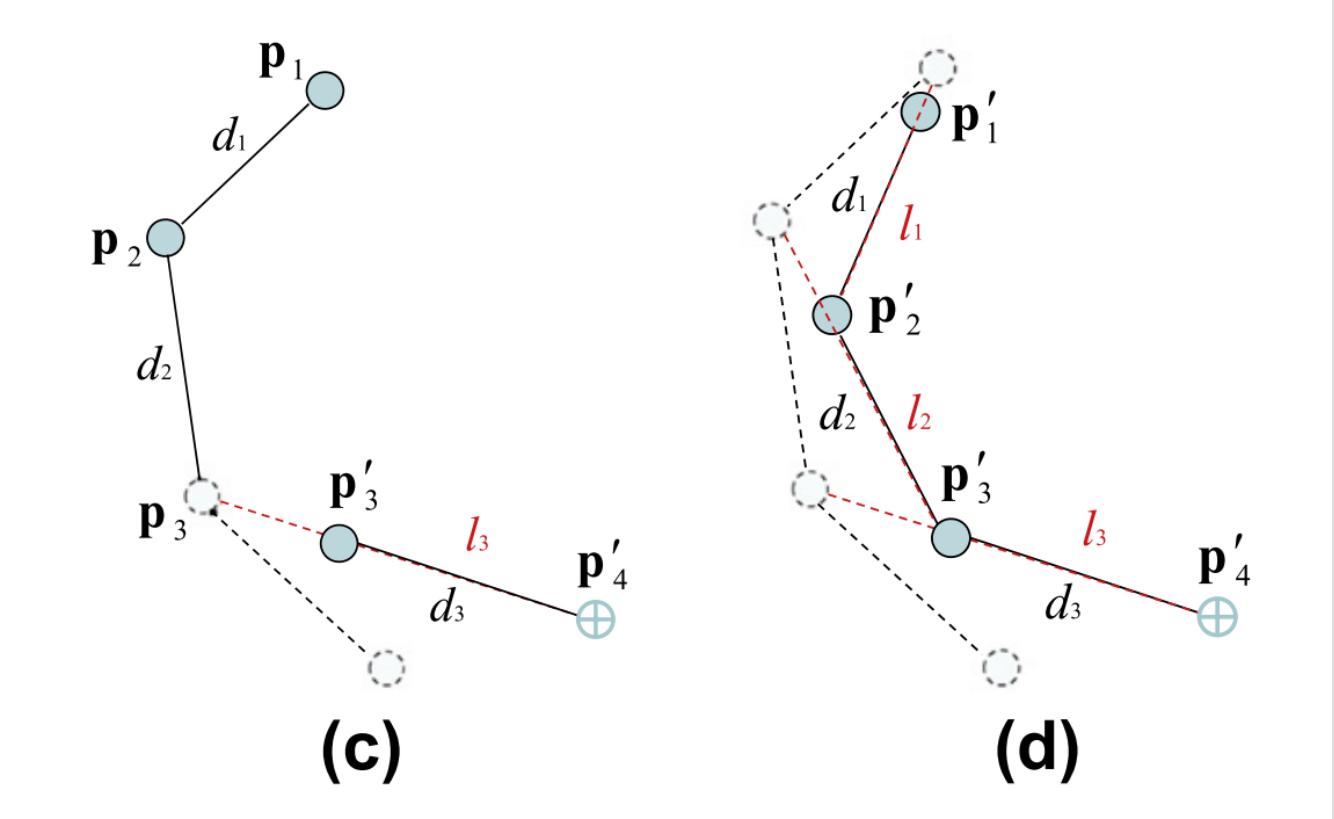
Forward and Backward Reaching Inverse Kinematics (FABRIK)

- Novel Heuristic Iterative Method
- Efficient and simple
- Supports rotational and orientational constraints



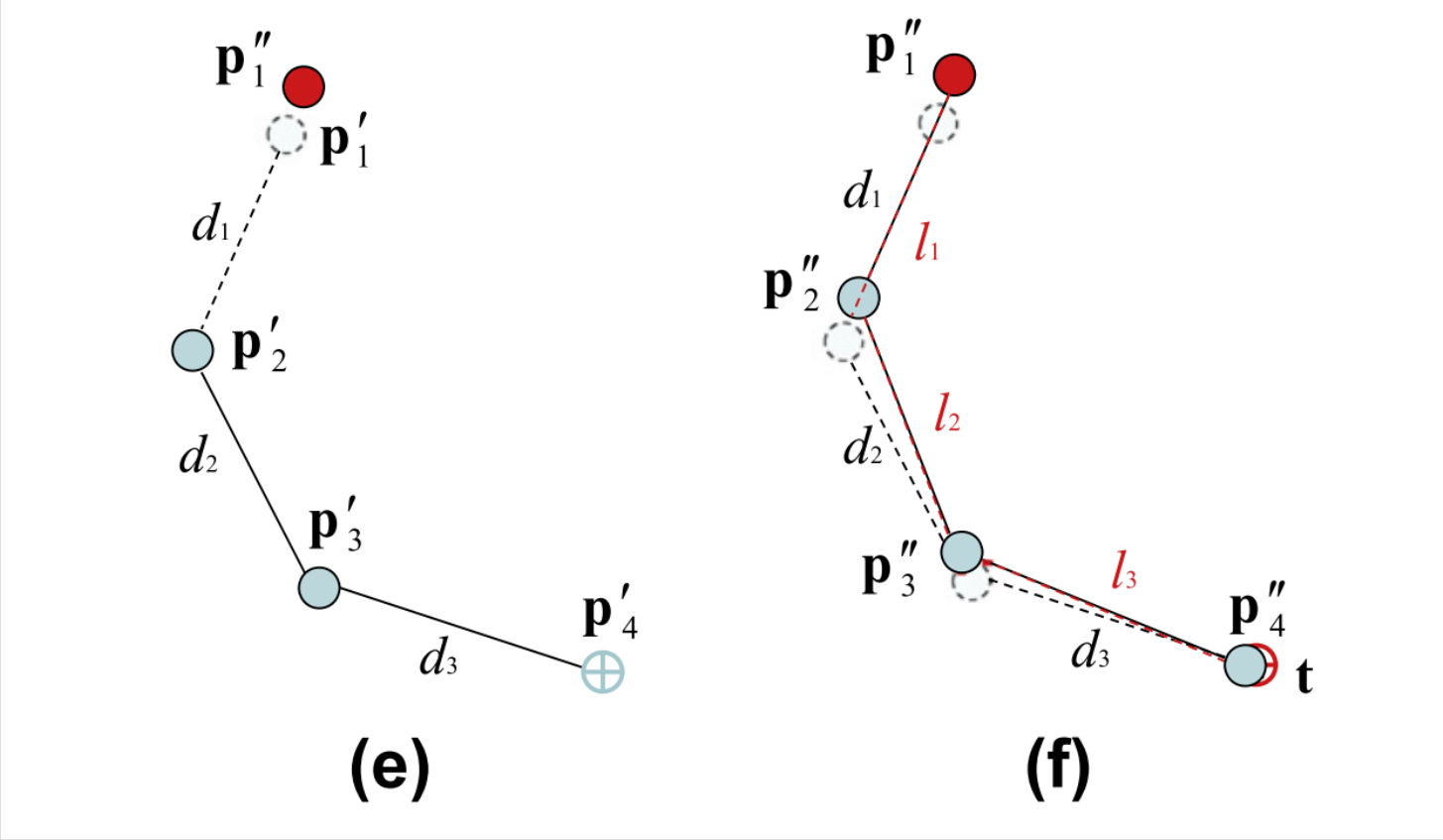
http://www.andreasaristidou.com/FABRIK.html

Forward and Backward Reaching Inverse Kinematics



http://www.andreasaristidou.com/FABRIK.html

Forward and Backward Reaching Inverse Kinematics



Kinematics Summary

- Forward Kinematics
  - Animator has complete freedom over the movements of any part of the structure
  - Amount of work is function of the complexity of the structure, much more expensive when dealing with complex structures
- Inverse Kinematics
  - Does not leave much scope for the animator to inject ‘character’ into the movements
  - Model does not possess the interpretation of a human being
  - Overall movement of structure depends on the formula -> same footprints, same movements

Resources

- http://graphics.ucsd.edu/courses/cse169_w04/welman.pdf
- Lander: Oh My God, I Inverted Kine – download from here: *graphics.cs.cmu.edu/nsp/course/15.../j**lander**_gamedev_sept98.pdf*
- *Lander: Making Kine More Flexible: graphics.cs.cmu.edu/nsp/.../j**lander**_gamedev_nov98.pdf*
- *Recent Summary: http://perso.telecom-paristech.fr/~pelachau/site/allpapers/MIG12_Huang.pdf*
- http://www.andreasaristidou.com/publications/papers/Extending_FABRIK_with_Model_C%CE%BFnstraints.pdf
- *See "Further Reading" on Blackboard*