

Computer Animation

Lesson 4 - Forward and inverse kinematics

Remi Ronfard, Nov 2019

Principle 4 Slow in & Slow out

Slow in and slow out deals with the spacing of the inbetween drawings between the extreme poses. Mathematically, the term refers to second- and third-order continuity of motion.

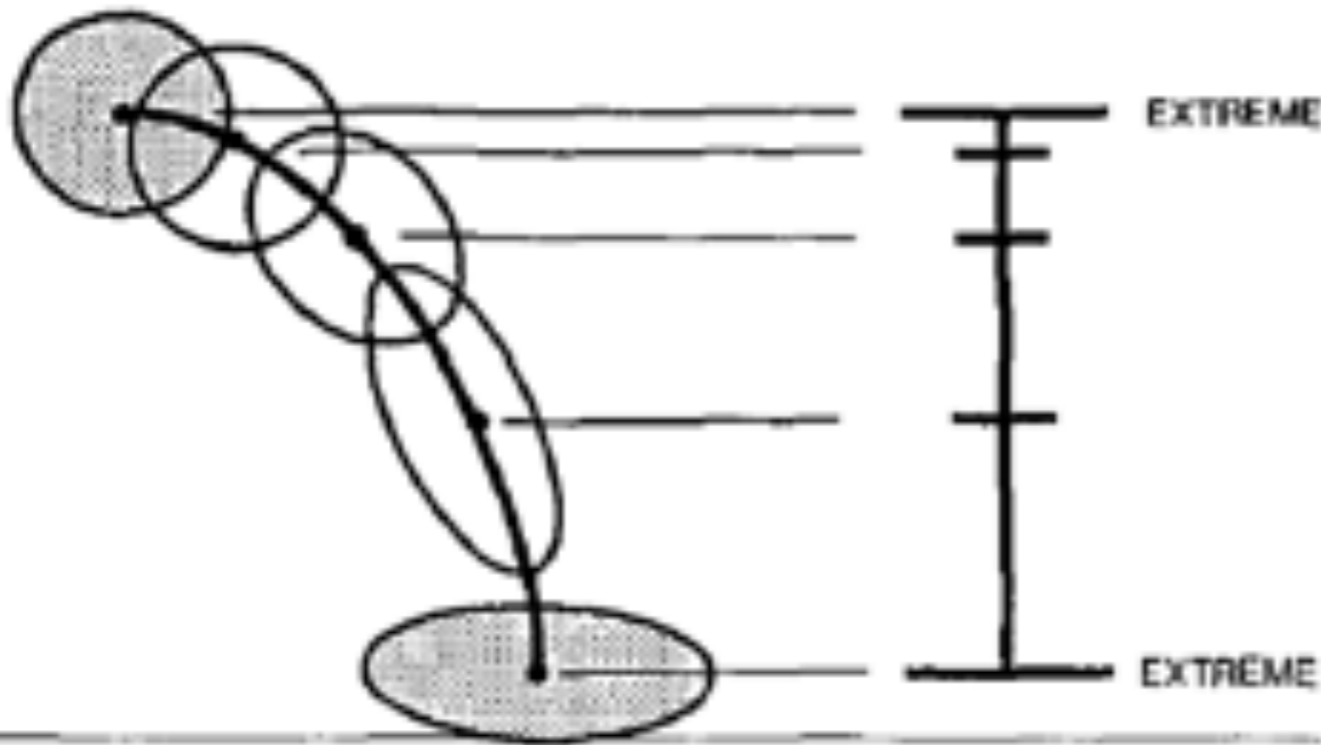


FIGURE 9. Timing chart for ball bounce.



FIGURE 10a. This spline controls the (y/y) translation of Lasso II. Dips in the spline illustrate inbetween drawings.

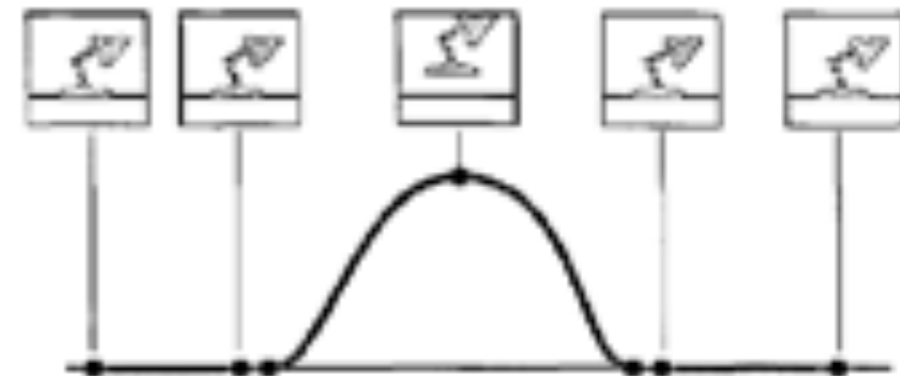


FIGURE 10b. Two extra inbetween drawings added to the spline which smooths the dips and prevents it from appearing too flat.

6. Slow In & Slow Out

Fast numerical methods for inverse kinematics by Bill Baxter

Overview

- The IK problem and formulation
 - ◆ Forward vs Inverse Kinematics
 - ◆ Joint types
 - ◆ Rigid body math
- Jacobian inverse method (J^{-1})
- Jacobian transpose method (J^T)
- Method of Cyclic Coordinate Descent (CCD)

The IK Problem

- We will only consider manipulators that are open kinematic chains composed of rigid links
- (i.e. A jointed robot arm with no loops)



Fast numerical methods for inverse kinematics by Bill Baxter

Joint Types

- 1 DOF Joint types:

- Revolute



- Prismatic



More Joint types

- Many higher order joint types can be represented by combinations of 1-DOF joints

- Just make axes intersect

- 2DoF Revolute

RR



- Planar

PP



- Cylindrical

RP



Fast numerical methods for inverse kinematics by Bill Baxter

Forward Kinematics

- Let \mathbf{q} be the configuration of manipulator
(coords in state space, i.e. A vector of all the θ_i, d_i)
- Let \mathbf{x} be the tip of the manipulator
(coords in cartesian space + maybe orientation, a 3 or 6-vector)



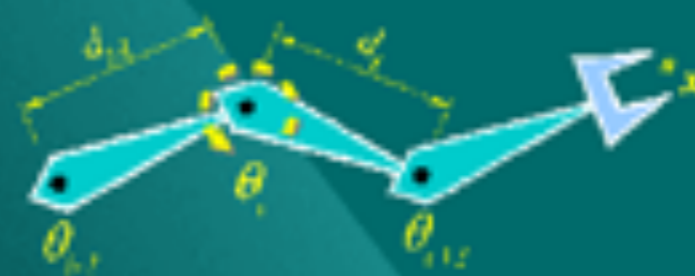
- For a given manipulator we can write down a closed-form formula for f such that

$$\mathbf{x} = f(\mathbf{q})$$

That's FORWARD KINEMATICS

So what is f ?

- Revolute joints



Single link

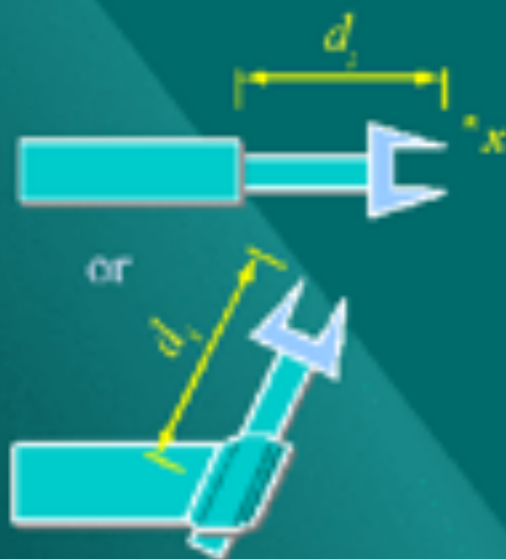
$${}^i T_{i+1} = \begin{bmatrix} {}^i R_{i-1}(\theta_i) & d_i \\ \mathbf{0} & 1 \end{bmatrix}$$

$${}^i \mathbf{x}_{i+1} = {}^i T_{i+1} {}^i \mathbf{x}_i$$

Fast numerical methods for inverse kinematics by Bill Baxter

So what is f ?

- Prismatic joint



Single link

$${}^i T_{i+1} = \begin{bmatrix} {}^i R_{i-1} & -d_i \\ \mathbf{0} & 1 \end{bmatrix}$$

$${}^i x_{i+1} = {}^i T_{i+1} {}^{i+1} x_{i+1}$$

All together

$$x = {}^0 x_n = {}^0 T_1 {}^1 T_2 \cdots {}^{n-1} T_n x$$

Other Representations

There are several ways to mathematically represent structural relationships in kinematic manipulators:

- Separate Rotation + translation: $T = A(x) - d$
 - Rotation as 3x3 matrix
 - Rotation as quaternion
 - Rotation as euler angles
- Homogeneous matrices: $T = H(R,d)$

Fast numerical methods for inverse kinematics by Bill Baxter

Kinematics

- Forward: joint params \rightarrow EE position
- Inverse: EE position \rightarrow joint params

Forward kinematics

$$\begin{aligned}x &= f(\mathbf{q}) \\x &= ({}^0T_1 {}^1T_2 \cdots {}^{n-1}T_n x)(\mathbf{q}) \\x &= {}^0T_1(q_1) {}^1T_2(q_2) \cdots {}^{n-1}T_n(q_n) x\end{aligned}$$

Inverse kinematics

$$f^{-1}(x) = \mathbf{q} \quad ??$$

So what is f^{-1} ?

- We have no idea

Forward kinematics

$$\begin{aligned}x &= f(\mathbf{q}) \\x &= ({}^0T_1 {}^1T_2 \cdots {}^{n-1}T_n x)(\mathbf{q}) \\x &= {}^0T_1(q_1) {}^1T_2(q_2) \cdots {}^{n-1}T_n(q_n) x\end{aligned}$$

Inverse kinematics

$$f^{-1}(x) = \mathbf{q} \quad ??$$

- Find answer numerically

Fast numerical methods for inverse kinematics by Bill Baxter

An idea

Forward kinematics

$$x = f(q)$$

$$x = ({}^0T_1(q_1) \cdots {}^{n-1}T_n(q_n))x$$

$$x = ({}^0T_1(q_1) \cdots {}^{n-1}T_n(q_n))^n x$$

Inverse kinematics

$$f^{-1}(x) = q$$

- f is nonlinear because of all those **sin**'s and **cos**'s in the rotations.
- Idea: find linear approximation to f^{-1}

The Jacobian

Forward kinematics

$$x = f(q) = (f_1(q), f_2(q), \dots, f_n(q))$$

- The Jacobian gives the linear approximation to f

$$J_f(q) = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial q_1} & \cdots & \frac{\partial f_n}{\partial q_n} \end{bmatrix}$$

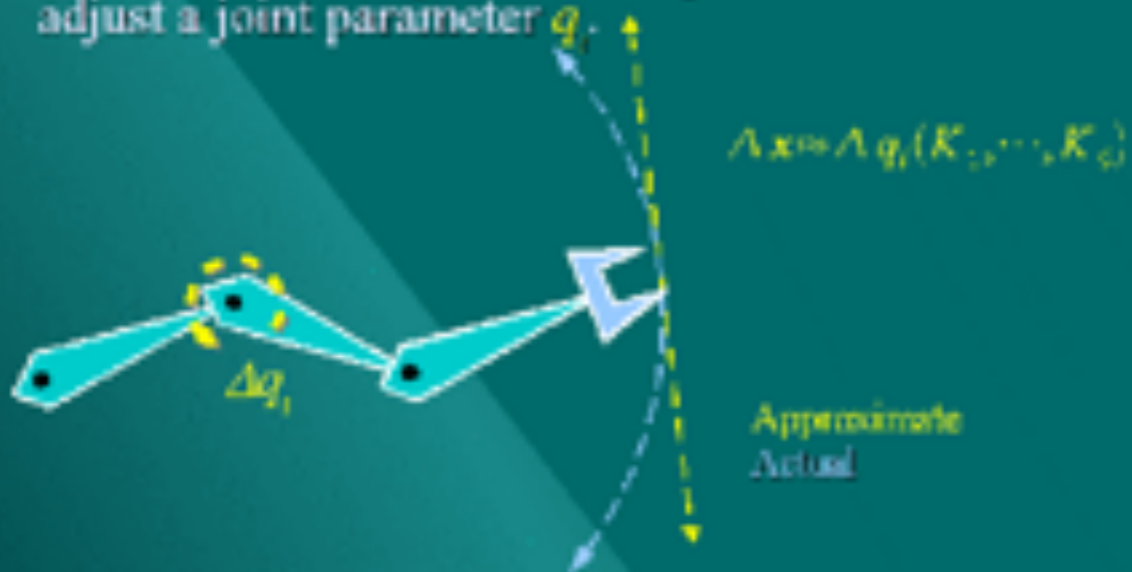
$J_f(q)$ is a $6 \times n$ matrix.

- The partials in J_f must be evaluated at some \hat{q} to yield the *local* linear approximation to f
- $f(\hat{q}) + [J_f(\hat{q})](q - \hat{q})$ is the 1st order Taylor approx. of f at \hat{q} .

Fast numerical methods for inverse kinematics by Bill Baxter

The Jacobian

- Put another way, J tells us approximately how much x will change in world space when we adjust a joint parameter q_i .



The Jacobian

- Collectively, J tells us how x changes locally in world space with respect to *all* the parameters q_i .



Fast numerical methods for inverse kinematics by Bill Baxter

Computing the Jacobian

- To get the Jacobian we have to know how to compute derivatives of our forward kinematic equation.
- The forward kinematics is generally some matrices and/or quaternions.

Matrix Derivatives

- Interested in affect of time dependant matrix, $A(t)$, on a transformed point $X = A(t)x$.

$$\begin{aligned}\dot{X}(t_0) &= [\dot{A}(t_0)]x \\ &= [\dot{A}(t_0)][A^{-1}(t_0)A(t_0)]x \\ &= [\dot{A}(t_0)A^{-1}(t_0)][A(t_0)]x \\ &= [\dot{A}(t_0)A^{-1}(t_0)]X\end{aligned}$$

- $[\dot{A}A^{-1}]$ is called the *tangent operator*

Fast numerical methods for inverse kinematics by Bill Baxter

Rotation Matrix Derivatives

- What's the tangent operator for a rotation?

$$\begin{aligned} RR^T &= I && \text{(by orthonormality)} \\ \dot{R}R^T + R\dot{R}^T &= 0 && \text{(from differentiation)} \\ \dot{R}R^T + (\dot{R}R^T)^T &= 0 \\ RR^T &= -(\dot{R}R^T)^T && \text{(skew symmetric)} \\ \dot{R}R^{-1} &= \dot{R}R^T = \Omega && \text{(so tangent operator is skew symmetric)} \end{aligned}$$

- Ω is the *angular velocity* matrix of $R(t)$

The Angular Velocity Matrix

- Ω is skew symmetric so it can be written

$$\Omega = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_x & 0 & -\omega_z \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

- And its effect on on a vector r is equivalent to the cross product with $\omega = (\omega_x, \omega_y, \omega_z)$

$$\Omega r \equiv \omega \times r$$

- ω is the axis of the rotation, ω is the amount.

Fast numerical methods for inverse kinematics by Bill Baxter

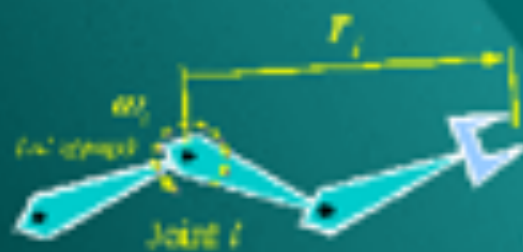
In general: **Simple Answer**

$$J = [J_1 \ J_2 \ \dots \ J_n]$$

$$J_i = \begin{bmatrix} \partial P_i / \partial q \\ \partial P_i / \partial q \\ \partial P_i / \partial q \\ \partial O_i / \partial q \\ \partial O_i / \partial q \\ \partial O_i / \partial q \end{bmatrix}$$

For revolute DOF:

$$J_i = \begin{bmatrix} \omega_i \times r_i \\ |\omega_i|^r \end{bmatrix}$$



For prismatic DOF:

$$J_i = \begin{bmatrix} \text{axis}_i^T \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



The Jacobian Inverse



Find Δq by:

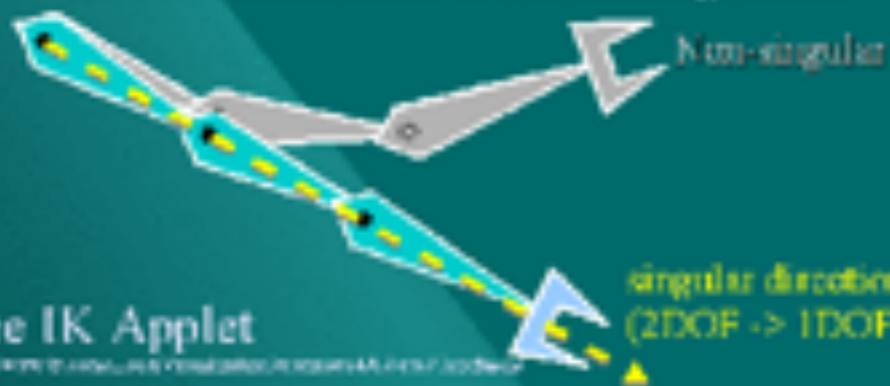
$$J^{-1} \Delta x = \Delta q$$

- J is a $6 \times n$ matrix, so how do we invert it?
- Use a *generalized inverse*.
- Generalized inverse is not unique. X is a generalized inverse of J if it has some of the following properties:
 1. $JXJ = J$
 2. $XJX = X$
 3. $(XJ)^T = XJ$
 4. $(JX)^T = JX$
- And it is a *pseudoinverse* if it has all of them.
- Write $X = J^*$. The pseudoinverse is unique.

Fast numerical methods for inverse kinematics by Bill Baxter

What's Wrong with J^{-1} ?

- Fairly slow to compute
 - Dreville's method: $O(m^3)$ ~57 mults per DOF with $m=6$
 - via $J(J^T J)^{-1}$ ($J^T J$ is regular)
- Instability around singularities
 - The Jacobian loses rank in certain configurations



- See IK Applet

Jacobian Transpose

- Jacobian transpose method uses the transpose of the Jacobian matrix rather than the p-inverse

Find Δq by:

$$\Delta q = J^T \Delta x$$

rather than:

$$\Delta q = J^{-1} \Delta x$$

- Avoids expensive inversion
- Avoids singularity problems

But why is this a reasonable thing to do?

Fast numerical methods for inverse kinematics by Bill Baxter

Principal of Virtual Work

- "Virtual" because amount is infinitesimal
- Work = force x distance. Work = torque x angle



$$F \cdot \Delta x = \tau \cdot \Delta q \quad (\text{energy equal in any coordinates})$$
$$F^T \Delta x = \tau^T \Delta q$$

$$\Delta x = J \Delta q \quad (\text{forward kinematics})$$

$$F^T J \Delta q = \tau^T \Delta q \quad (\text{substitution})$$

$$F^T J = \tau^T \quad (\text{transpose both sides})$$

$$\tau = J^T F$$

Jacobian Transpose

Virtual work eqn:

$$\tau = J^T F$$

Compare with:

$$\Delta q = J^{-1} \Delta x$$

- So we're taking the distance to our goal to be a force that pulls the end effector.
- With J inverse, solution was exact answer to linearized problem. This is no longer so.

Fast numerical methods for inverse kinematics by Bill Baxter

Jacobian Transpose

$$\Delta \mathbf{q} = \mathbf{J}^T \Delta \mathbf{x}$$

- This relation is not exact, but has the right trend
- So throw in a scaling factor h and iterate

$$\Delta \mathbf{q}^{(k+1)} = h \mathbf{J}^T \Delta \mathbf{x}^{(k)}$$

- The value h can be thought of as a timestep

$$\frac{\Delta \mathbf{q}}{\Delta t} = \mathbf{J}^T \dot{\mathbf{x}}$$

- So we're really just solving the diffy Q

$$\dot{\mathbf{q}} = \mathbf{J}^T \dot{\mathbf{x}}$$

- Remember that \mathbf{x} is a function of joint parameters \mathbf{q}

$$\dot{\mathbf{q}} = \mathbf{J}^T \mathbf{F}(\mathbf{q})$$

Jacobian Transpose

$$\dot{\mathbf{q}} = \mathbf{J}^T \mathbf{F}(\mathbf{q})$$

- In effect the Jacobian transpose method solves the IK problem by setting up a dynamical system that obeys the Aristotilean laws of physics.

$$\mathbf{F} = m \mathbf{v}$$

$$\boldsymbol{\tau} = I \boldsymbol{\omega}$$

- The Jacobian pseudoinverse method is equivalent to solving by Newton's method.
- Jacobian transpose is also related to solution by the method of steepest descent.

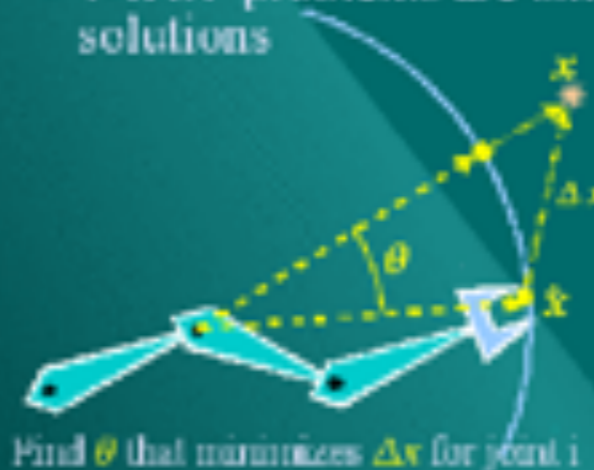
Fast numerical methods for inverse kinematics by Bill Baxter

Good and Bad of J^T

- Cheaper evaluation step than pseudoinverse
 - + No singularities
 - Scaling problems
 - J^T has nice property that solution has minimal norm at every step.
 - J^T doesn't have this property. Joints far from end effector experience larger torques, hence take disproportionately large steps.
 - Can throw in a constant diagonal scaling matrix to counteract some scaling probs
- $$\dot{q} = K J^T J^{-1}(\dot{x})$$
- where each K_i set appropriately
- Slower to converge than J^+
 - (2x slower according to Das, Slotine & Sheridan)

Cyclic Coordinate Descent

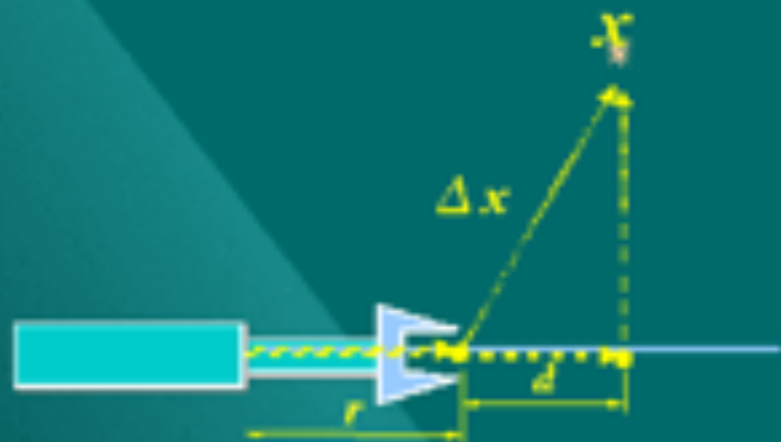
- Actually a much simpler idea
 - Just solve 1 DOF IK problems repeatedly up chain
- 1-DOF problems are simple and have analytical solutions



Fast numerical methods for inverse kinematics by Bill Baxter

CCD Math - Prismatic

Δx minimized when $d = (\Delta x \text{ -axis})$ axis



CCD Math - Revolute

Δx minimized when $r(\theta) \cdot g$ maximized

Let S be the skew symm axis matrix

$$r(\theta) = R_{\text{axis}}(\theta)r$$

$$= [I + \sin\theta |S| + (1 - \cos\theta) |S|^2] r$$



Solve single var. optimization prob:

$$e - g \cdot r(\theta)$$

$$de - 0 = d \left(g \cdot [I + \sin\theta |S| + (1 - \cos\theta) |S|^2] r \right)$$

$$= g \cdot [\cos\theta |S| + \sin\theta |S|^2] r$$

$$\theta = \tan^{-1} \left(\frac{g \cdot S r}{g \cdot S^2 r} \right)$$

Fast numerical methods for inverse kinematics by Bill Baxter

CCD Math - Revolute



Of course you may wish to optimize orientation too, in which case you need another expression for orientation error, and you minimize the combination of the two.

This still can be done closed form, but it is a bit messier.

You can derive expressions to optimize for other goals too. We just did it for point goals, but you could define your goal to be a line or a plane for instance.

Good and Bad of CCD

- | Simple to implement
 - | Often effective
 - + Stable around singular configuration
 - + Computationally cheap
 - + Can combine with other more accurate optimization method like BFS when close enough
- BUT**
- Can lead to odd solutions if per step deltas not limited, making method slower
 - Doesn't necessarily lead to smooth motion

Fast numerical methods for inverse kinematics by Bill Baxter

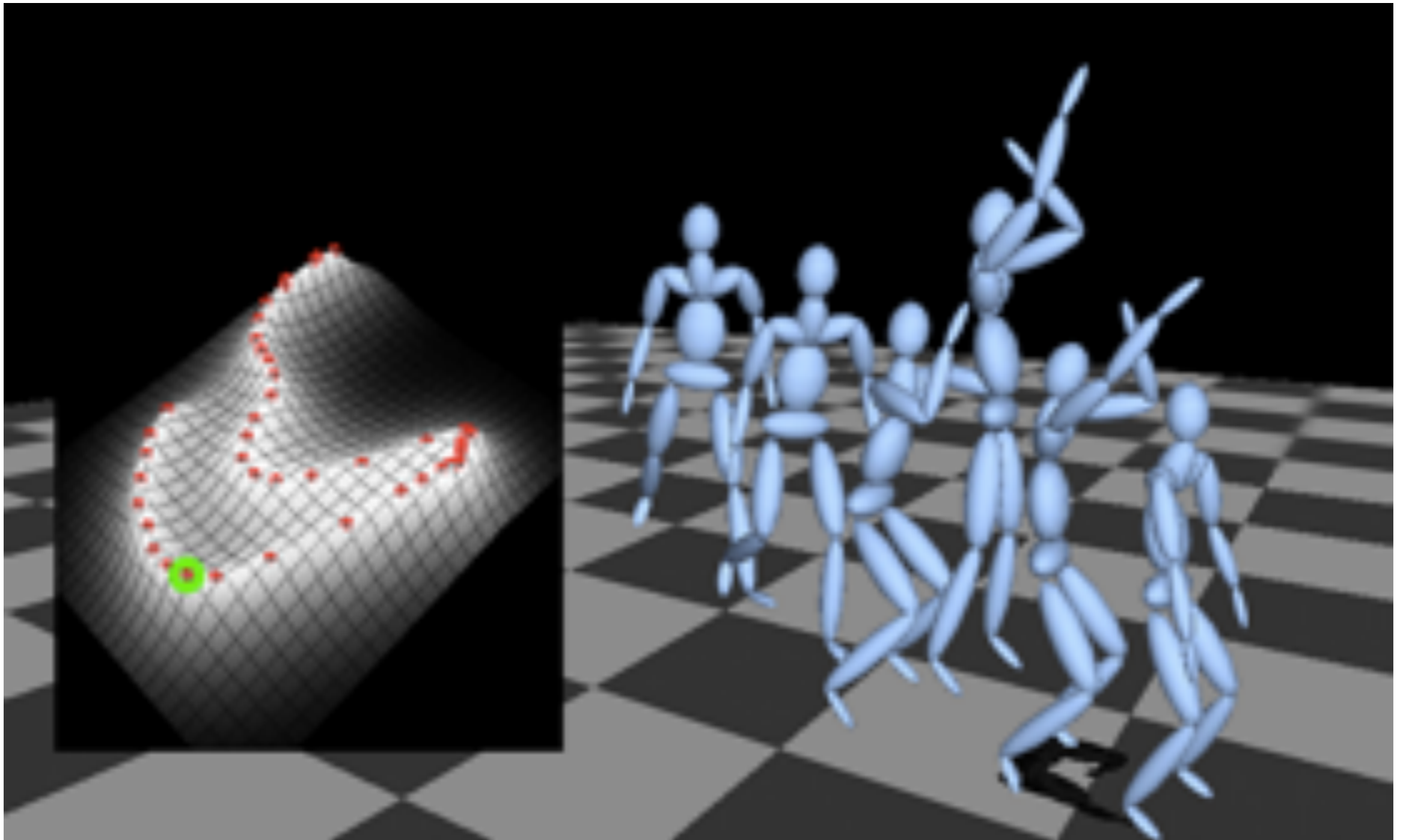
Wrap up

- Representation is important, but your call.
 - Must consider trade-off in
 - space requirements
 - computational requirements
 - system interfacing costs
 - stability and robustness
- J^T and CCD methods both quick and effective (depending on your requirements, of course)
- I ignored constraints. They are very important.

Bibliography

- Marion, T. and Odell, P. *Generalized Inverse Matrices*. New York : Wiley-Interscience, 1971.
- Das, Sriniv, and Sheridan, "Inverse Kinematic Algorithms for Redundant Systems." *Proceedings of IEEE Int'l Conference on Robotics and Automation*, pp 43-48, 1988.
- J. M. McCarthy. "An Introduction to Theoretical Kinematics." Cambridge : MIT Press, 1990
- L. Siciliano and B. Siciliano. "A Dynamic Solution to the Inverse Kinematic Problem for Redundant Manipulators." *IEEE Int'l Conference on Robotics and Automation*, pp 1081-1086, March 1987.
- Wang, and Chen, "A Combined Optimization Method for Solving the IK Problem of Mechanical Manipulators." *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 4, Aug 1991, pp 489-499.
- Wichart, Chris, "Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation". Master's Thesis, Simon Fraser University, 1994.
- A. Wikin, D. Baraff. *Physically Based Modeling*, SIGGRAPH course notes.

Paper 4 - Style-based Inverse Kinematics



Paper 4 - Style-based Inverse Kinematics



Paper 4 - Style-based Inverse Kinematics

