Inverse Kinematics

Animating Characters

Many editing techniques rely on either:

- Interactive posing
- Putting constraints on bodyparts' positions and orientations (includes mapping sensor positions to body motion)
- Optimizing over poses or sequences of poses

All three tasks require inverse kinematics

Goal

Several different approaches to IK, varying in capability, complexity, and robustness

We want to be able to choose the right kind for any particular motion editing task/tool

IK Problem Definition

1) Create a handle on body
   - position or orientation

2) Pull on the handle

3) IK figures out how joint angles should change

More Formally

Let:

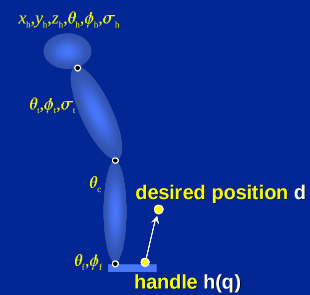
- **q** — **actor *state vector* (joint bundle)**
- **C(q)** — **constraint functions that pull handles**

Then:

solve for q such that C(q) = 0

What's a Constraint?

q=[$x_h, y_h, z_h, \theta_h, \phi_h, \sigma_h, \theta_t, \phi_t, \sigma_t, \theta_c, \theta_f, \phi_f$ **]**



Can be rich, complicated

But most common is very simple:

Position constraint just sets difference of two vectors to zero:

C(q) = h(q) - d = 0

The *Real* problem & Approaches

The IK problem is usually very underspecified

- many solutions
- most bad (unnatural)
- how do we find a good one?

Two main approaches:

- Geometric algorithms
- Optimization/Differential based algorithms

Geometric

Use geometric relationships, trig, heuristics

Pros:

- fast, reproducible results

Cons:

- proprietary; no established methodology
- hard to generalize to multiple, interacting constraints
- cannot be integrated into dynamics systems

Optimization Algorithms

Main Idea: use a numerical metric to specify which solutions are good

metric - a function of state q (and/or state velocity) that measures a quantity we'd like to minimize

Example

Some commonly used metrics:

- joint stiffnesses
- minimal power consumption
- minimal deviation from "rest" pose

Problem statement:

Minimize metric $G(q)$
subject to satisfying $C(q) = 0$

An Approach to Optimization

If $G(q)$ is quadratic, can use Sequential Quadratic Programming (SQP)

- original problem highly non-linear, thus difficult
- SQP breaks it into sequence of quadratic subproblems
- iteratively improve an initial guess at solution
- How?

Search and Step

Use constraints and metric to find direction Δq that moves joints closer to constraints

Then $q_{new} = q + a\,\Delta q$ where

$$\min_{a} C(q + a\,\Delta q)$$

Iterate whole process until $C(q)$ is minimized

Breaking it Down

Performing the integration $q_{new} = q + a\,\Delta q$ is easy (Brent's alg. to find a)

Finding a good Δq is much trickier

Enter Derivatives.

What Derivatives Give Us

We want:

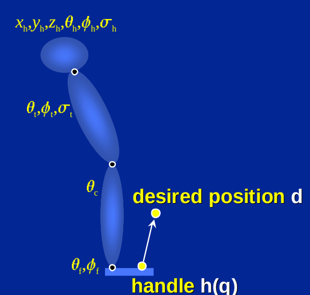
- a direction in which to move joints so that constraint handles move towards goals

Constraint Derivatives tell us:

- in which direction constraint handles move if joints move

Constraint derivatives

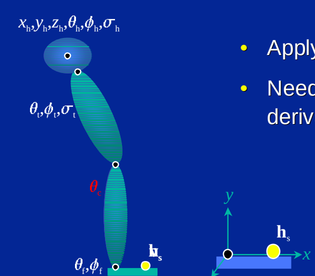
q=[$x_h, y_h, z_h, \theta_h, \phi_h, \sigma_h, \theta_t, \phi_t, \sigma_t, \theta_c, \theta_f, \phi_f$]



C(q) = h(q) - d = 0

$$\frac{\partial C(\mathbf{q})}{\partial \mathbf{q}} = \frac{\partial h(\mathbf{q})}{\partial \mathbf{q}}$$

Computing Derivatives

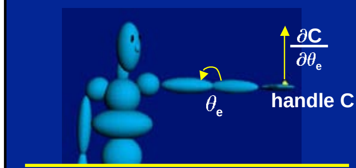


- Apply the chain rule
- Need to know how to compute derivatives for each transformation

$$h_w = \mathbf{T}(x_h, y_h, z_h)\mathbf{R}(\theta_h, \phi_h, \sigma_h)\,\mathbf{TR}(\theta_t, \phi_t, \sigma_t)\;\mathbf{TR}(\theta_c)\mathbf{TR}(\theta_f, \phi_f)\,h_s$$

$$\frac{\partial h_w}{\partial \theta_c} = \mathbf{T}(x_h, y_h, z_h)\mathbf{R}(\theta_h, \phi_h, \sigma_h)\,\mathbf{TR}(\theta_t, \phi_t, \sigma_t)\;\mathbf{T}\frac{\partial \mathbf{R}(\theta_c)}{\partial \theta_c}\mathbf{TR}(\theta_f, \phi_f)\,h_s$$

Jacobian Matrix



$\frac{\partial C}{\partial q}$

	...	θ_e	...
x	.	0	.
y	.	1	.
z	.	0	.

Can compute Jacobian for each constraint / handle

Value of Jacobian depends on current state

Jacobian linearly relates joint angle velocity to constraint velocity

Jacobian Matrix

Efficient techniques for computing Jacobians use a recursive traversal to compute all partial derivatives.

Unconstrained Optimization

Main Idea: treat each constraint as a separate metric, then just minimize combined sum of all individual metrics, plus the original

- Many names: penalty method, soft constraints, Jacobian Transpose
- physical analogy: placing damped springs on all constraints
  - *each spring pulls on constraint with force proportional to violation*

Unconstrained Optimization

Minimize $G'(q) = G(q) + \sum_i w_i C_i(q)^2$

Move in the direction of the objective function gradient:

$$\frac{\partial G'}{\partial q} = \frac{\partial G}{\partial q} + 2\sum_i w_i C_i \frac{\partial C_i}{\partial q}$$

$$q = q_o + \alpha \frac{\partial G'}{\partial q}$$

We need to efficiently compute derivatives of the objective G and constraints C.

Unconstrained Performance

Pros:

- Simple, no linear system to solve, each iteration is fast
- near-singular configurations less of problem

Cons:

- Constraints fight against each other and original metric
- sloppy interactive dragging (can't maintain constraints)
- linear convergence

Constrained Optimization

- Many formulations (*e.g.* Lagrangian, Lagrange Multipliers)
- All involve solving a linear system comprised of Jacobians, the quadratic metric, and other quantities

$$\underset{\mathbf{q}}{\text{minimize}} \quad G(\mathbf{q})$$
$$\text{subject to} \quad \mathbf{C}(\mathbf{q})$$

Result: constraints satisfied (if possible), metric minimized subject to constraints

Lagrangian formulation

Given

$$\underset{\mathbf{q}}{\text{minimize}} \quad G(\mathbf{q})$$
$$\text{subject to} \quad \mathbf{C}(\mathbf{q})$$

We define a Lagrangian $L(\mathbf{q},\boldsymbol{\lambda}) = G(\mathbf{q}) - \boldsymbol{\lambda} \cdot \mathbf{C}$

$$\underset{\mathbf{q},\boldsymbol{\lambda}}{\text{minimize}} \quad G(\mathbf{q}) - \boldsymbol{\lambda} \cdot \mathbf{C}$$

Lagrangian formulation

At the solution of

$$\underset{\mathbf{q},\boldsymbol{\lambda}}{\text{minimize}} \quad G(\mathbf{q}) - \boldsymbol{\lambda} \cdot \mathbf{C}$$

We have

$$\frac{\partial G(\mathbf{q}) - \boldsymbol{\lambda} \cdot \mathbf{C}}{\partial \{\mathbf{q},\boldsymbol{\lambda}\}} = \mathbf{0}$$

Solving the Lagrangian

To solve $\frac{\partial G(\mathbf{q}) - \boldsymbol{\lambda} \cdot \mathbf{C}}{\partial \{\mathbf{q}, \boldsymbol{\lambda}\}} = \mathbf{0}$ iteratively

We setup the linear system

$$\begin{bmatrix} \frac{\partial^2 \mathbf{G}}{\partial^2 \mathbf{q}} & \frac{\partial \mathbf{C}}{\partial \mathbf{q}}^T \\ \frac{\partial \mathbf{C}}{\partial \mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} d\mathbf{q} \\ d\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\frac{\partial \mathbf{G}}{\partial \mathbf{q}} - \frac{\partial \mathbf{C}}{\partial \mathbf{q}}^T \boldsymbol{\lambda} \\ -\mathbf{C} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{q}_{new} \\ \boldsymbol{\lambda}_{new} \end{bmatrix} = \begin{bmatrix} \mathbf{q} \\ \boldsymbol{\lambda} \end{bmatrix} + \alpha \begin{bmatrix} d\mathbf{q} \\ d\boldsymbol{\lambda} \end{bmatrix}$$

Lagrangian Performance

Pros:

- Enforces constraints exactly
- Has a good “feel” in interactive dragging
- Quadratic convergence

Cons:

- Large system of equations
- A Dark Art to master
- near-singular configurations cause instability

Why Does Convergence Matter?

Trying to drive C(q) to zero:

# Iterations	1	2	3	4	5
quadratic **C(q)**	.25	.0625	.015	.004	.0009
linear **C(q)**	.5	.25	.125	.0625	.0313
linear/quadratic	2	4	8	16	32

IK == Constrained Particle system?

We can view the inverse kinematics problem as a constrained particle system

Two types of constraints:

- **Implicit constraints**: keep points on the same body part together
- **Explicit constraints**: allow us to control the position of an arbitrary body point

Kinematic energy derivation

$$T = \int_i m_i \dot{x}_i^T \dot{x}_i \quad \text{where} \quad x = R(q)\,p$$

$$T = \int_i m_i \left[\dot{R}p_i\right]^T \left[\dot{R}p_i\right]$$

$$= \int_i m_i \left[\frac{\partial R}{\partial q}\dot{q}p_i\right]^T \left[\frac{\partial R}{\partial q}\dot{q}p_i\right]$$

$$= \int_i m_i \dot{q}^T \left[\frac{\partial R}{\partial q}\right]^T p_i \otimes p_i \left[\frac{\partial R}{\partial q}\right]\dot{q}$$

$$= \sum_j \dot{q}^T \left[\frac{\partial R}{\partial q}\right]^T \int_{j_i} \left(m_{j_i} p_{j_i} \otimes p_{j_i}\right) \left[\frac{\partial R}{\partial q}\right]\dot{q}$$

Euler Lagrange Equations

Without potential energy the Lagrangian is:

$$L = T = \sum_j \dot{q}^T \left[\frac{\partial R_j}{\partial q}\right]^T I_j \left[\frac{\partial R_j}{\partial q}\right]\dot{q}$$

So equations of motion are computed as

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) = 0$$

$$\frac{d}{dt}\left(\sum \left[\frac{\partial R_j}{\partial q}\right]^T I_j \left[\frac{\partial R_j}{\partial q}\right]\dot{q}\right) = 0$$

$$\left[\sum \left[\frac{\partial R_j}{\partial q}\right]^T I_j \left[\frac{\partial R_j}{\partial q}\right]\right]\ddot{q} + [\cdots]\dot{q} = 0$$

Mass matrix

The "*F=ma*" equation is given by

$$\left[\sum \left[\frac{\partial R_j}{\partial q}\right]^T I_j \left[\frac{\partial R_j}{\partial q}\right]\right]\ddot{q} + [\cdots]\dot{q} = 0$$

So the mass analog is given by the **mass matrix**:

$$M = \sum \left[\frac{\partial R_j}{\partial q}\right]^T I_j \left[\frac{\partial R_j}{\partial q}\right]$$

F=mv world

Since we are only concerned with the geometric interpretation of positions we can simplify the equations by moving into the first-order world:

$$Q = M\dot{q}$$

or

$$\dot{q} = WQ$$

Constraints in the F=mv world

$$\dot{q} = W\left(Q + Q_c\right)$$

$$\dot{C} = \frac{\partial C}{\partial q}\dot{q} + \frac{\partial C}{\partial t} = 0$$

$$\frac{\partial C}{\partial q}W\left(Q + Q_c\right) + \frac{\partial C}{\partial t} = 0 \qquad Q_c = \lambda \frac{\partial C}{\partial q}$$

$$\frac{\partial C}{\partial q}W\left[\frac{\partial C}{\partial q}\right]^T \lambda = \frac{\partial C}{\partial q}WQ + \frac{\partial C}{\partial t}$$

Finally, how does this help us solve IK

Compute W

$$W = \left(\sum \left[\frac{\partial R_j}{\partial q}\right]^T I_j \left[\frac{\partial R_j}{\partial q}\right]\right)^{-1}$$

Compute λ

$$\frac{\partial C}{\partial q}W\left[\frac{\partial C}{\partial q}\right]^T \lambda = \frac{\partial C}{\partial q}WQ + \frac{\partial C}{\partial t}$$

Compute forces

$$Q_c = \lambda \frac{\partial C}{\partial q}$$

Find the change in state

$$\dot{q} = W\left(Q + Q_c\right)$$

How to specify constraints without losing your mind

Suppose we wanted these constraints:

- Distance between 2 points is d
- Direction between 2 points is orthogonal to v

We don't want to plow through equations and their derivatives every time we come up with a new constraint.

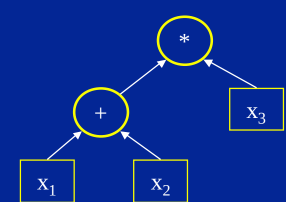
Solution: Automatic Differentiation

Automatic differentiation

The basic idea:

1. Define derivatives for a few atomic operations
2. Use the expression parse tree and the chain rule to compute derivatives of arbitrary expressions

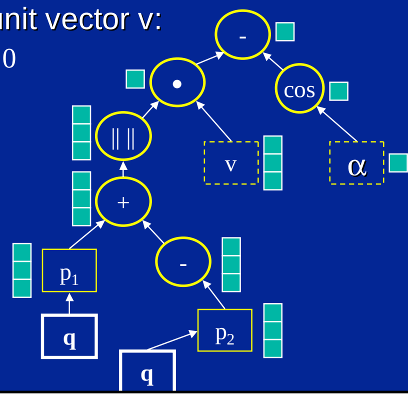
$$E = \left(x_1 + x_2\right)x_3$$



Multi-dimensional Auto Diff

Constraint: direction defined by two points must be at angle α wrt unit vector v:

$$\|p_1 - p_2\| \cdot v - \cos(\alpha) = 0$$



Recap and Conclusions

Inverse Kinematics

- Geometric algorithms
  - *fast, predictable for special purpose needs*
  - *don't generalize to multiple constraints or physics*
- Optimization-based algorithms
  - *Constrained vs. unconstrained methods*

Constrained optimization

Achieves true constrained minimum of metric

- solid feel and fast convergence
- involves arcane math
- near-singular configurations must be tamed
- Two formulations:
  - *Full Hessian (standard constrained minimization approach)*
  - *Reduced Hessian (Euler-Lagrange equations)*

Unconstrained optimization

Near-singular configurations manageable

- Constraints and the objective fight against each other
- spongy feel
- poor convergence
- easy to get penalty method up and running

Projected constraints speedup

Compute W

$$\cancel{W = \left(\sum \left[\frac{\partial R_j}{\partial q} \right]^T I_j \left[\frac{\partial R_j}{\partial q} \right] \right)^{-1}}$$

Compute only diagonal elements of M, co computing W is trivial

Compute λ

$$\frac{\partial C}{\partial q} W \left[\frac{\partial C}{\partial q} \right]^T \lambda = \frac{\partial C}{\partial q} W Q + \frac{\partial C}{\partial t}$$

Compute forces

$$Q_c = \lambda \frac{\partial C}{\partial q}$$

Find the change in state

$$\dot{q} = W \left(Q + Q_c \right)$$

Intermittent Constraints

During animation constraints may appear or disappear

This leads to abrupt changes in characters motion.

How can we alleviate this problem?