### 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

### 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

### 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

### Inverse Kinematics
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?

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 $\Delta q$ that moves joints closer to constraints

Then $q_{\text{new}} = q + a \Delta q$ 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 $\Delta q$ 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**

$$q = \left[ x, y, z, \phi, \theta_x, \theta_y, \theta_z, \phi, \theta_x, \phi, \theta_y, \phi, \theta_z \right]$$

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

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

**Jacobian Matrix**

Can compute Jacobian for each constraint / handle

Value of Jacobian depends on current state

Jacobian **linearly** relates joint angle velocity to constraint velocity
Computing Derivatives

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

\[ \mathbf{v}_w = T(x_h, y_h, z_h) R(\theta_h, \phi_h, \sigma_h) \left. \frac{\partial}{\partial \phi} \right|_{\phi=\phi_0} T(\theta_0, \phi_0, \sigma_0) \left. \frac{\partial}{\partial \theta} \right|_{\theta=\theta_0} R(\theta, \phi, \sigma) \mathbf{v}_s \]
\[ \mathbf{v}_\theta = T(x_h, y_h, z_h) R(\theta_h, \phi_h, \sigma_h) \left. \frac{\partial}{\partial \phi} \right|_{\phi=\phi_0} T(\theta_0, \phi_0, \sigma_0) \left. \frac{\partial}{\partial \theta} \right|_{\theta=\theta_0} R(\theta, \phi, \sigma) \mathbf{v}_s \]

Jacobian Matrix

Have efficient techniques for computing Jacobians

But how do we use it to compute \( \Delta \mathbf{q} \)?
- Constrained optimization
- Unconstrained optimization

Constrained Optimization

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

\[
\begin{align*}
\text{minimize} & \quad G(\mathbf{q}) \\
\text{subject to} & \quad C(\mathbf{q})
\end{align*}
\]

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

Constrained Performance

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

Cons:
- A Dark Art to master
- near-singular configurations cause instability
**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
    \[ G(q) = G(q) + C(q)^2 \]

**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

**Why Does Convergence Matter?**

Trying to drive \( C(q) \) to zero:

<table>
<thead>
<tr>
<th># Iterations</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>quadratic C(q)</td>
<td>.25</td>
<td>.0625</td>
<td>.015</td>
<td>.004</td>
<td>.0009</td>
</tr>
<tr>
<td>linear C(q)</td>
<td>.5</td>
<td>.25</td>
<td>.125</td>
<td>.0625</td>
<td>.0313</td>
</tr>
<tr>
<td>linear/quadratic</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
</tr>
</tbody>
</table>

**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
<table>
<thead>
<tr>
<th>Recap and Conclusions</th>
<th>Recap and Conclusions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constrained optimization</strong></td>
<td><strong>Unconstrained optimization</strong></td>
</tr>
<tr>
<td>• achieves true constrained minimum of metric</td>
<td>• near-singular configurations manageable</td>
</tr>
<tr>
<td>• solid feel and fast convergence</td>
<td>• spongy feel</td>
</tr>
<tr>
<td>• involves arcane math</td>
<td>• poor convergence</td>
</tr>
<tr>
<td>• near-singular configurations must be tamed</td>
<td>• easy to get penalty method up and running</td>
</tr>
</tbody>
</table>