<< RETURN

Controlling Reactive, Motion Capture-driven Simulated Characters

Victor B. Zordan

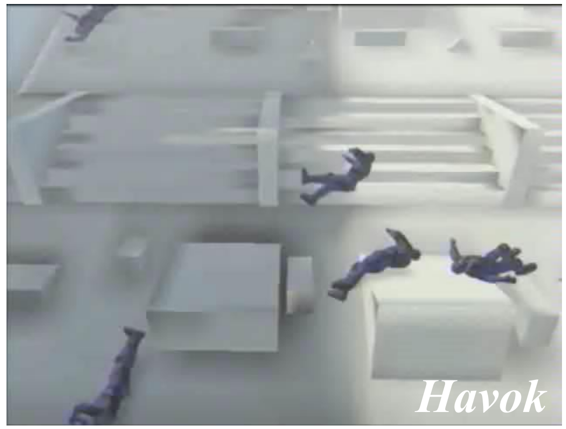
University of California at Riverside

Motion capture-driven simulations?
Motivation:

Motion capture is already the industry standard for lifelike, 3D characters

Physical 'ragdolls' and engines are gaining in use

Motion capture-driven simulations?
Motivation:



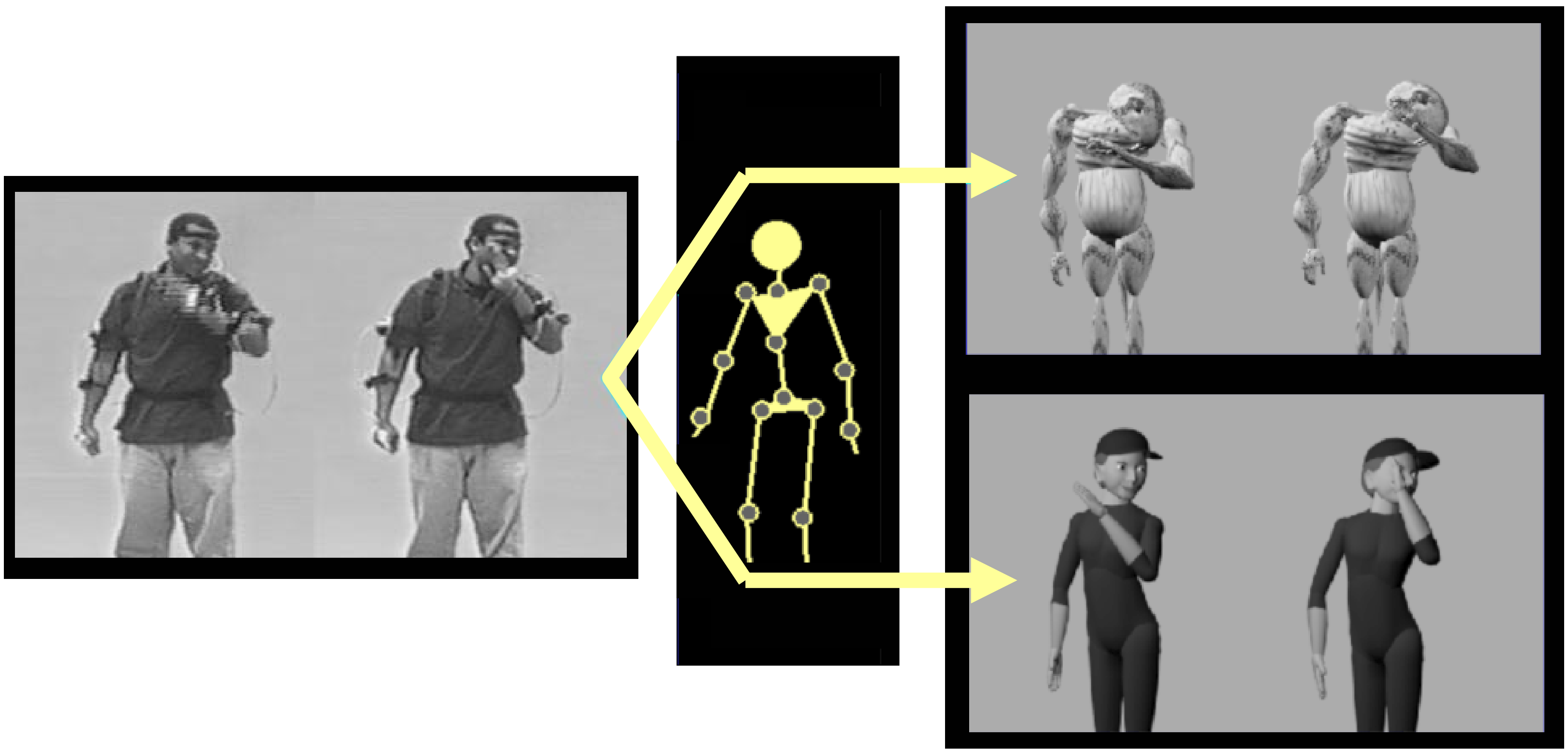
As the cost of simulation computation goes down and demand goes up, we will see a tighter coupling of the simulation and motion capture techniques

Examples of *blending* are already appearing (Havok2)

What are mocap-driven simulations?

Dynamically simulated characters that follow motion capture, *actively*

Motion capture data | **Dynamic model** | **Final animation**



Why use mocap-driven simulations?
To get the best compromise between:

Human motion capture

+rich with style & detail

- hard to adapt or to be made to 'respond' to new scenarios

Dynamic simulation

+physically realistic

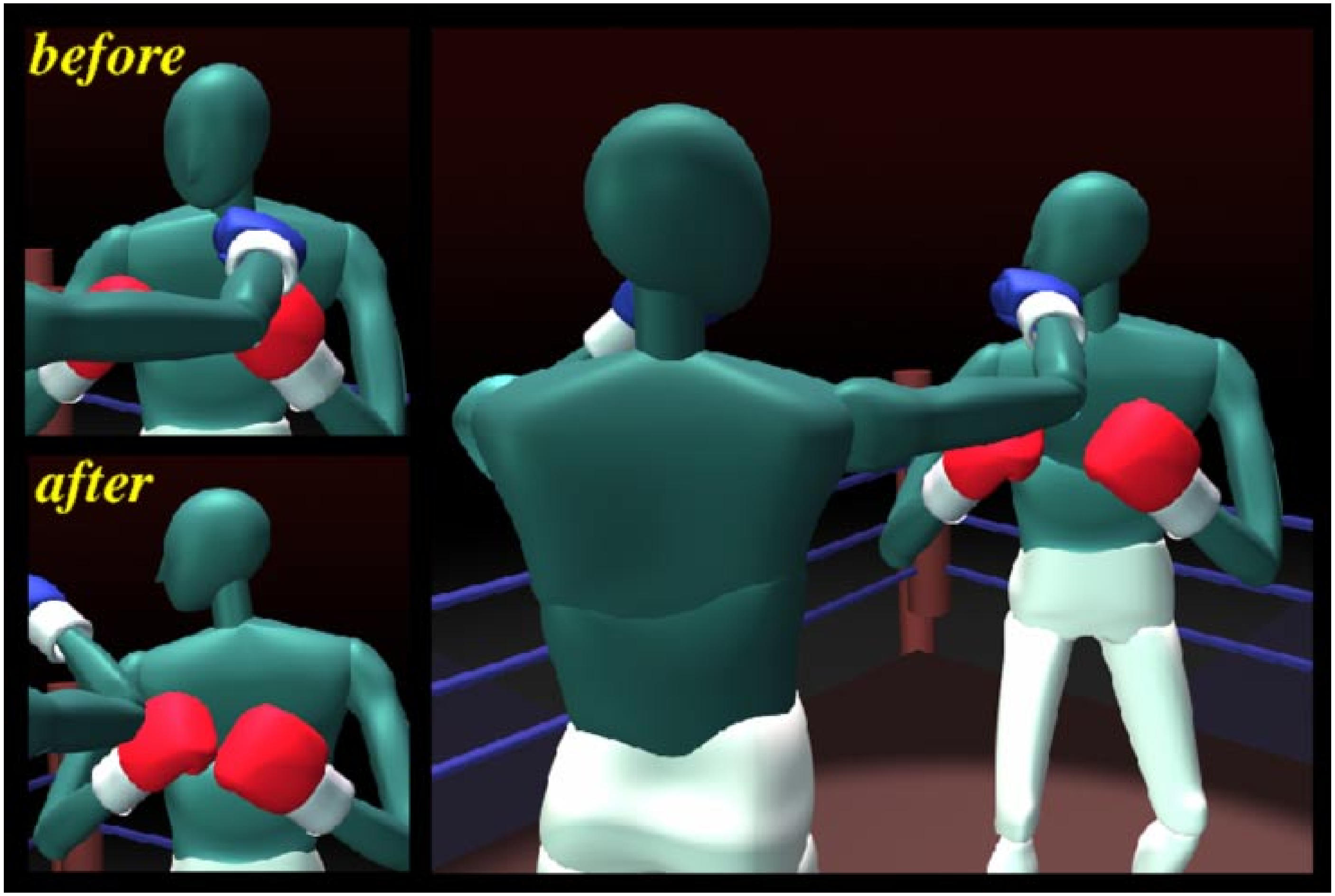
+handles a changing environment & can 'react' in believable ways

- requires a controller to actuate

Respond to new scenarios?

A changing environment?

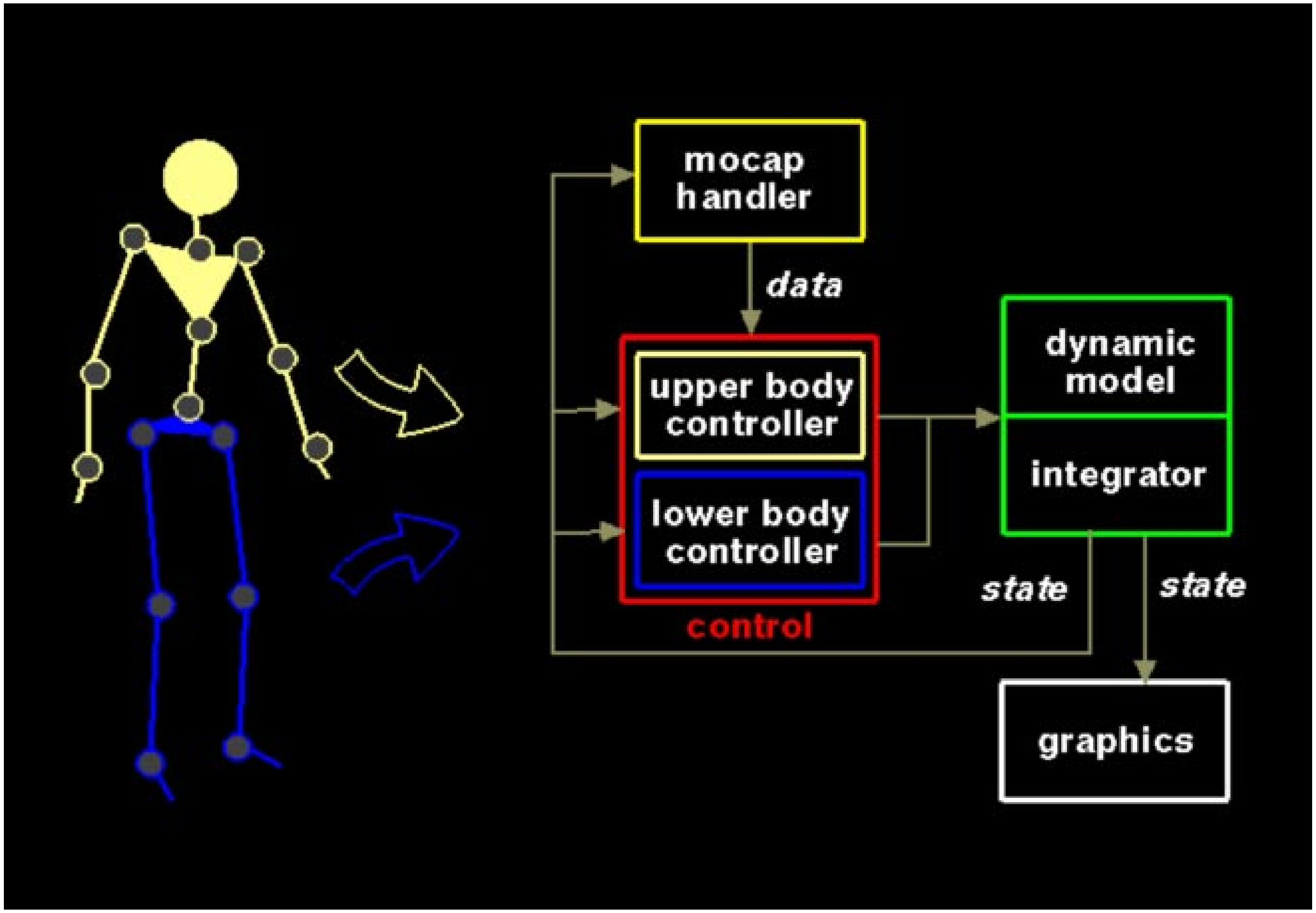
Reacting in believable ways? Huh?



Why do we want realistic reactions?

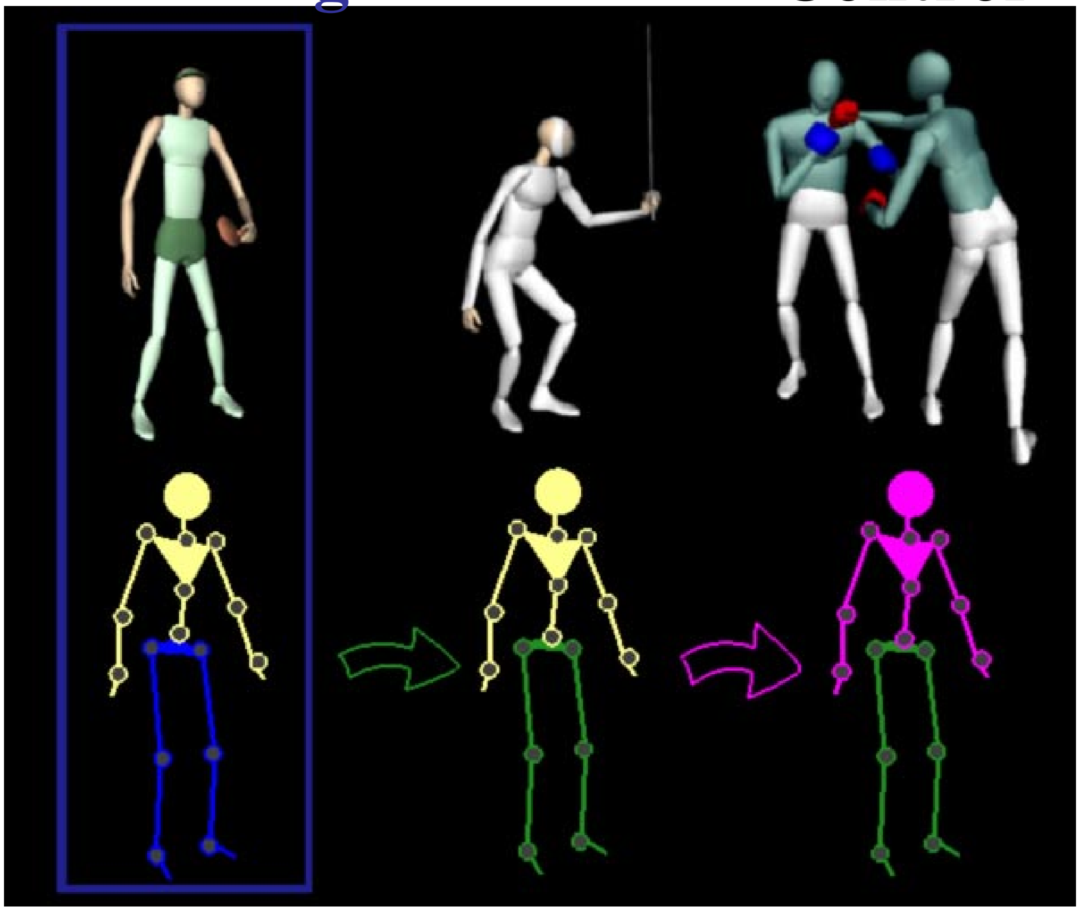
Beyond 'ragdolls' that 'play dead', want characters that *take a lickin' and keep on tickin'*

Overview:
System Layout

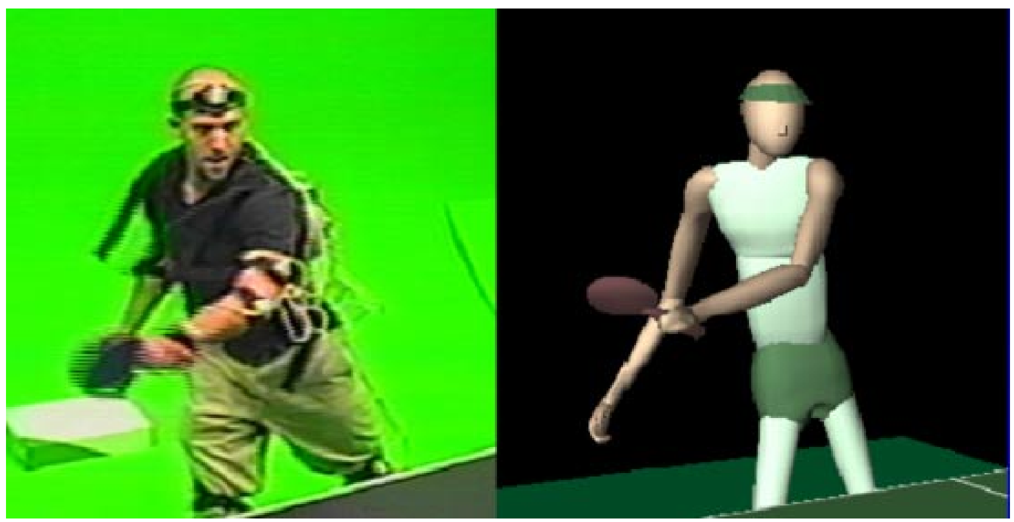


Overview:
Building a reactive character

Tracking Balance Control



Tracking Control

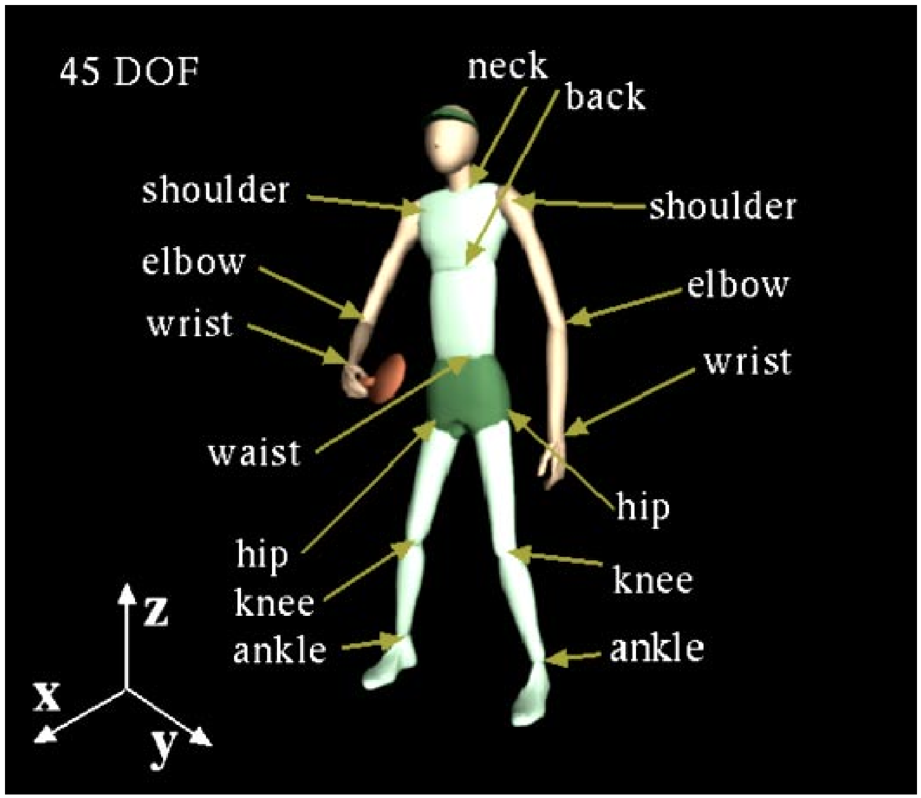


Tracking Control

Converted Mocap Data → Tracking Control → Dynamic Model

Riverside Graphics Lab

Equations of motion - computed by automatically (SD-Fast)

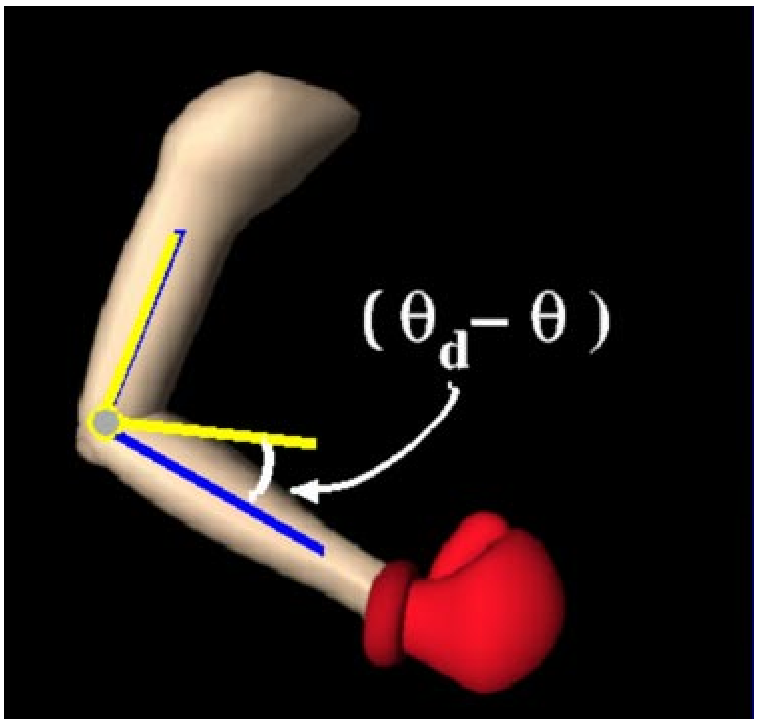


**Boxing sim
no wrists
(39 dof)**

Tracking Control

Converted Mocap Data → **Tracking Control** → Dynamic Model

PD-servo controller computes torques



$$\tau = k(\theta_d - \theta) - b(\dot{\theta})$$

θ_d from motion data

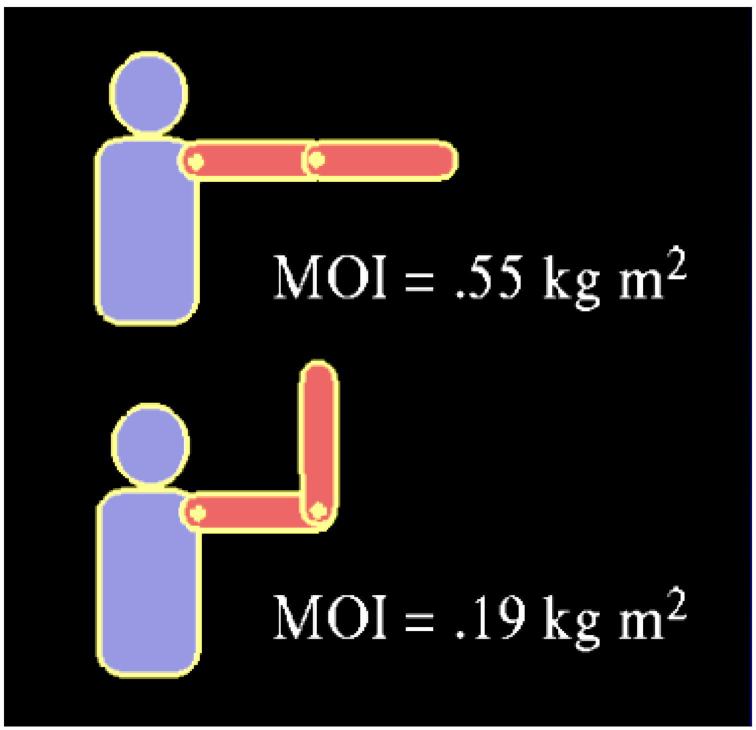
k and b are uniform stiffness and damping

Note: No joint limits, instead influenced by data

Tracking Control

Converted Mocap Data → **Tracking Control** → Dynamic Model

Inertia scaling for stiffness and damping



k and b are scaled by moment of inertia:

$$k = k' * MOI_{effect}$$
$$b = b' * MOI_{effect}$$

tune for uniform k and b

Then:

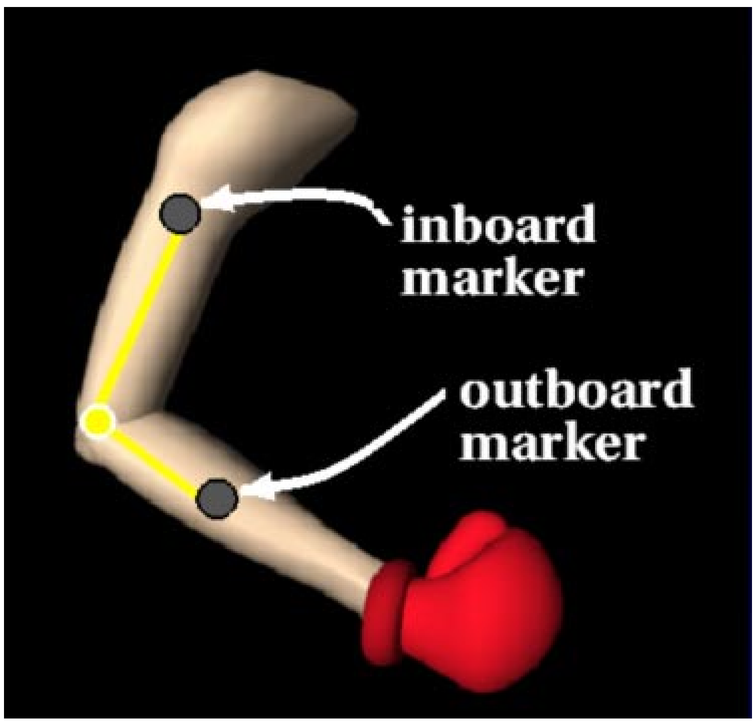
high stiffness + moderate damping = good tracking

Riverside Graphics Lab

Tracking Control

Converted Mocap Data → Tracking Control → Dynamic Model

Convert raw motion capture data to joint angles

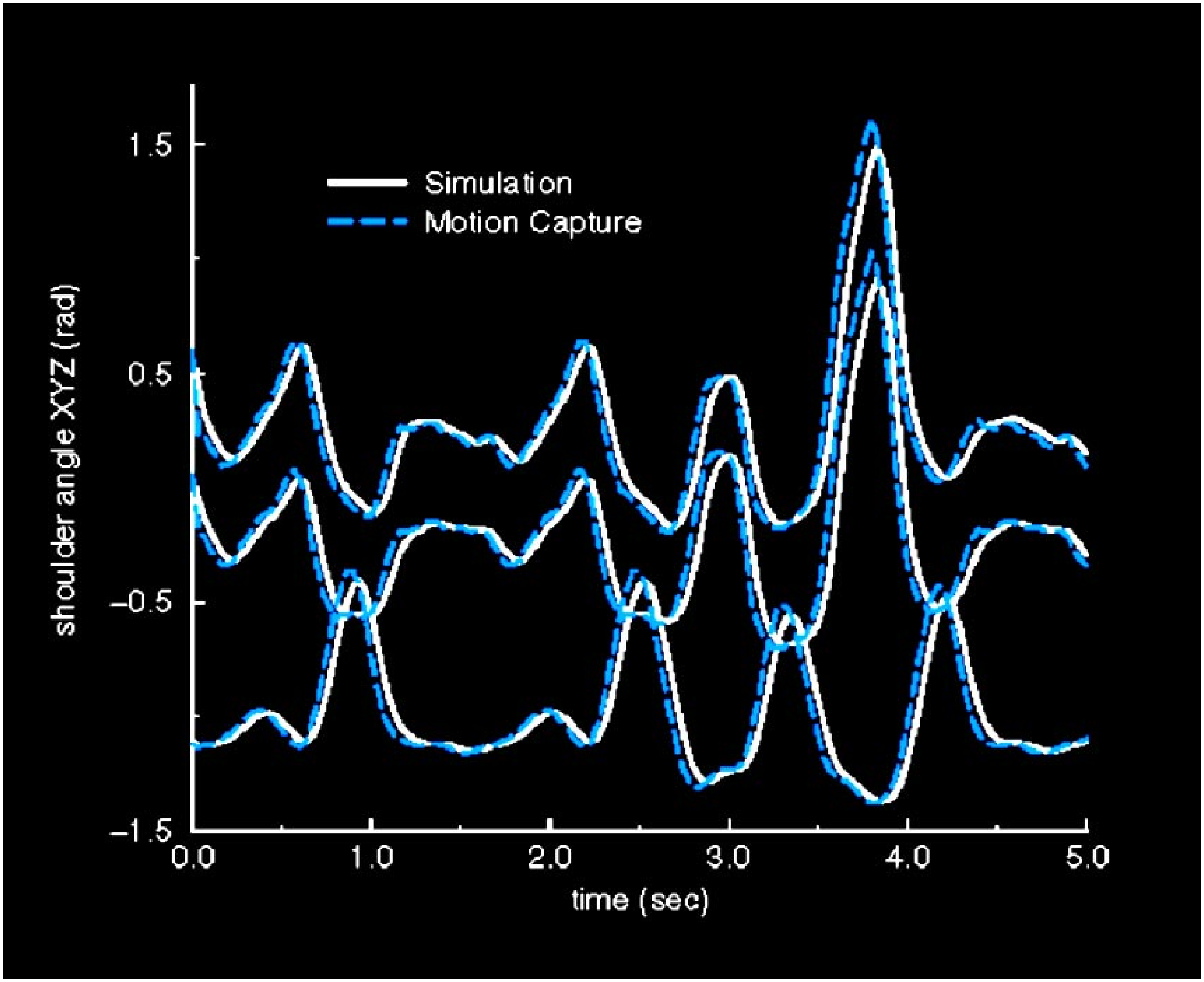


Optical: map/fit to skeleton

Electromagnetic: preprocess using marker orientation data for joint angles as

$$\Theta_{desired} = \Theta_{in}^{T}\,\Theta_{out}$$

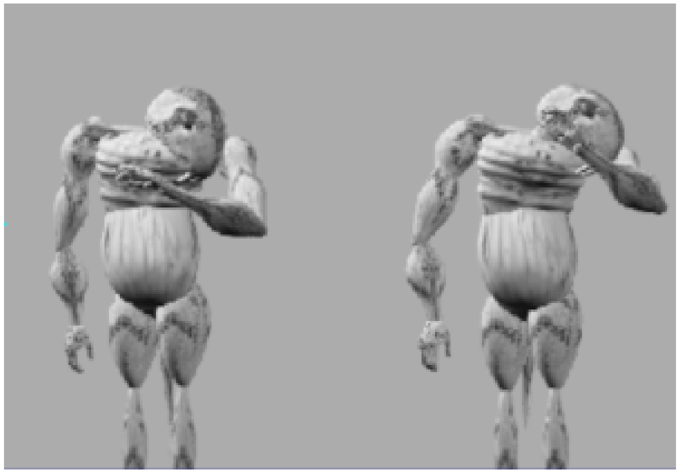
Then for both, fit spline thru samples (sim '*prefers*' such smoothed inputs)



Simulation
Motion Capture
shoulder angle XYZ (rad)
1.5
0.5
−0.5
−1.5
0.0
1.0
2.0
3.0
4.0
5.0
time (sec)

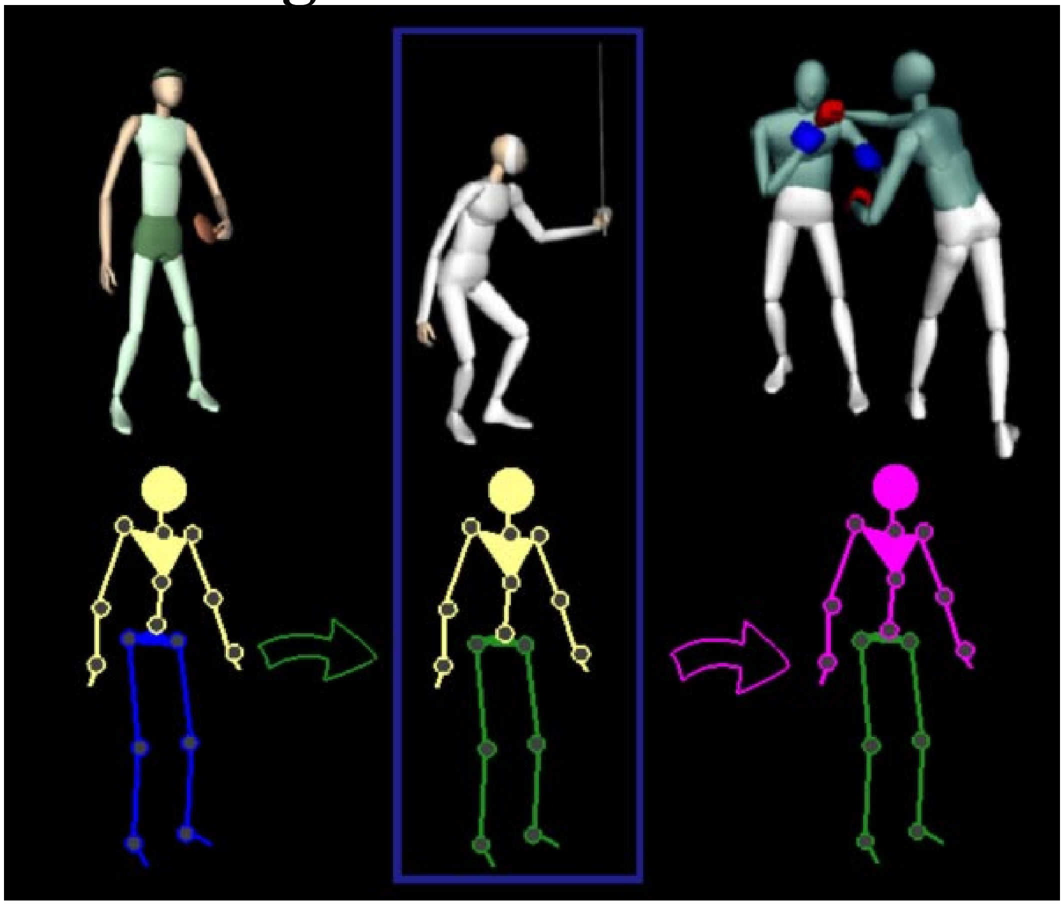
Tracking control is flexible enough to follow a large variety of motions...

...from the waist up

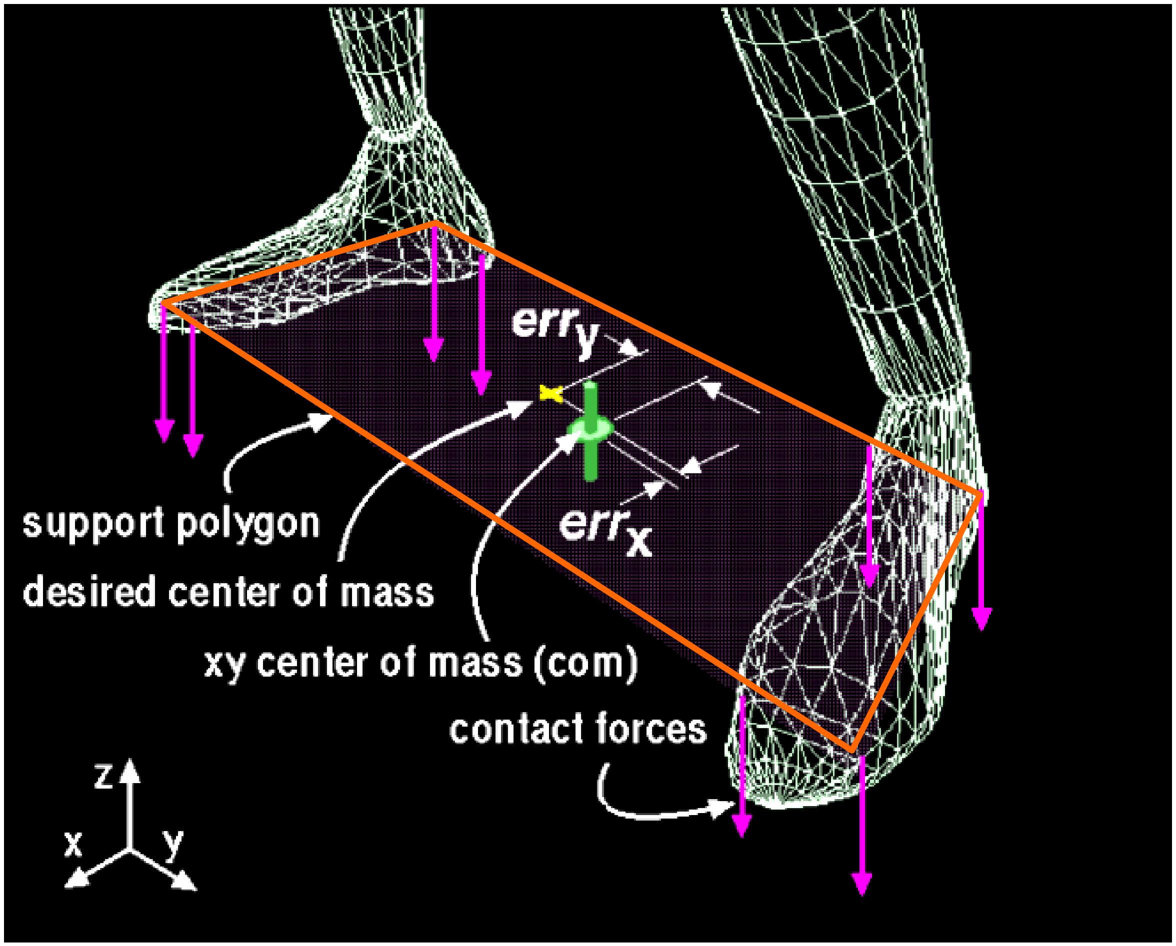


How about the rest of the body?
Need lower-body control

Tracking **Balance** **Control**



Lower-body Control
Balanced standing



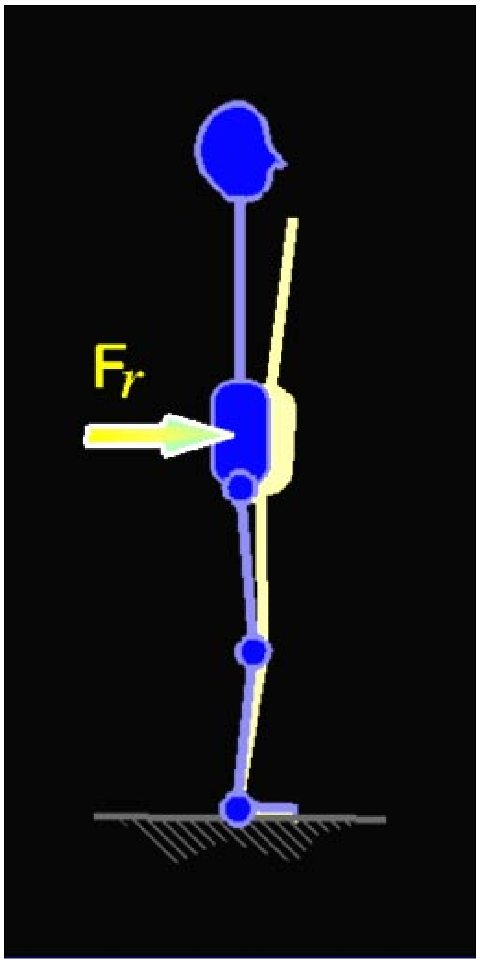
Controller's goal:

Keep the simulation's center of mass (com) safely inside the support polygon made by the feet

To accomplish the goal:

Pick a desired com and minimize errors by making corrections in the leg actuation

Lower-body Control
External balance force

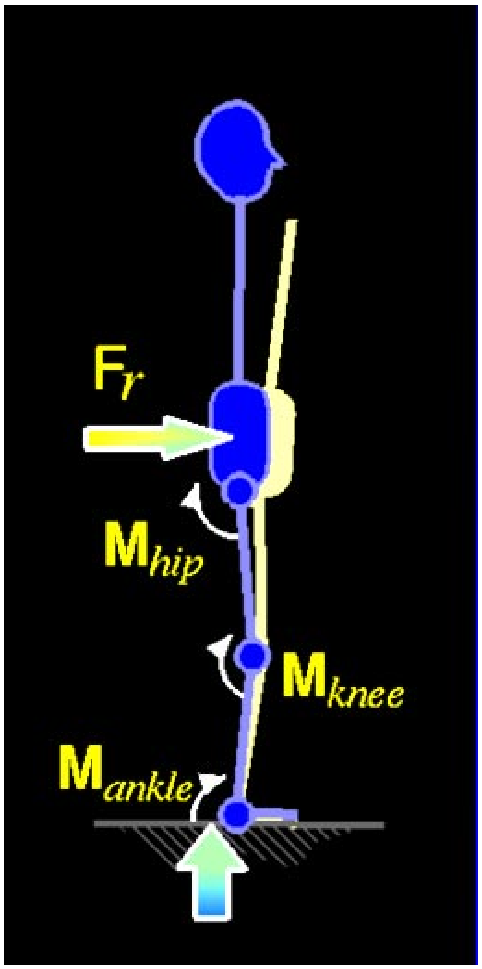


First compute the required pelvis force that would result in balance, but don't apply it directly...

Balancing force to control center of mass:

$$F_{r_{(x,y)}} = k_r\,(err) - b_r\,(\dot{err})$$

Lower-body Control
Virtual actuator method



Inspired by
Pratt *(1995)*

Convert force to torques for virtual actuator:

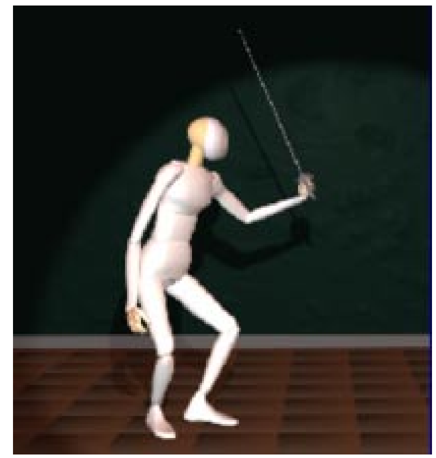
$$M_{(h->a)} = F_r \times X_{(h->a)}$$

$$\tau_{balance} = {}^{J}T_{0}\ {}^{0}M_{(h->a)}$$

$$\tau' = \tau_{track} + \tau_{balance}$$

Lower-body Control
Using the motion capture data

Add in info about the action taking place by extracting data from the mocap:



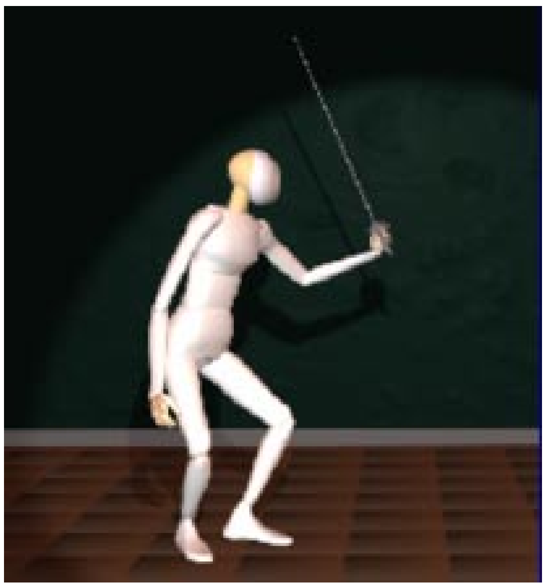
Desired as estimate com:

$$com_{mocap} = \sum \frac{m_i (x_{marker\ i})}{m_{total}}$$

Also, track the data in hips, knees, ankles

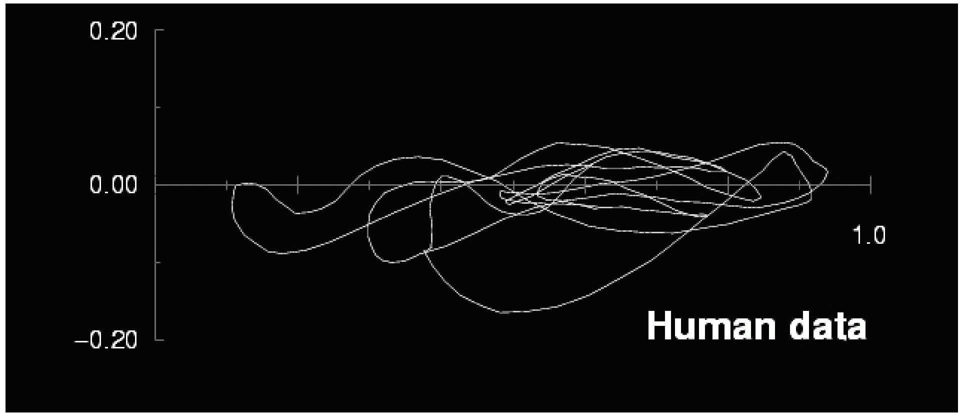
Full-body mocap-driven simulations

Riverside Graphics Lab

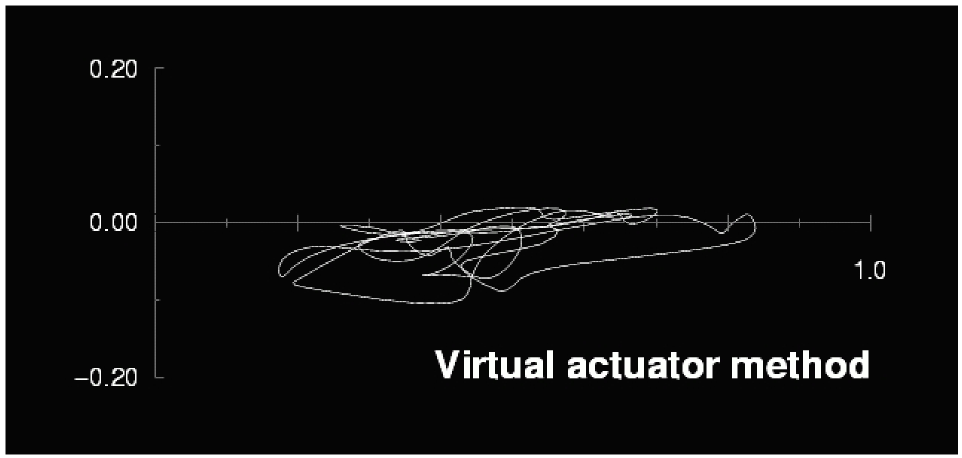



Full-body mocap-driven simulations

com estimated

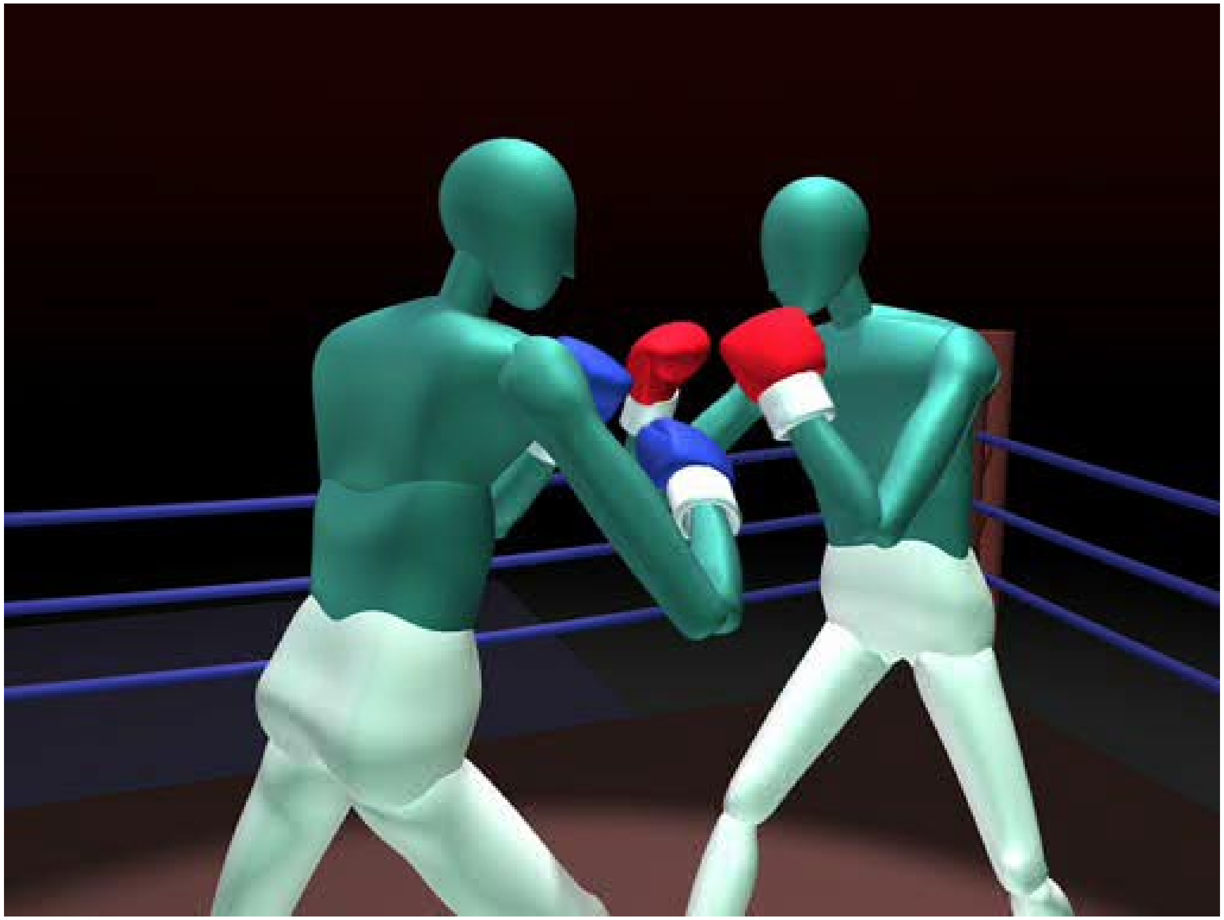


simulation com



Comparison for dancing motion (sim in blue from previous slide) normalized from one foot to the other on the horizontal

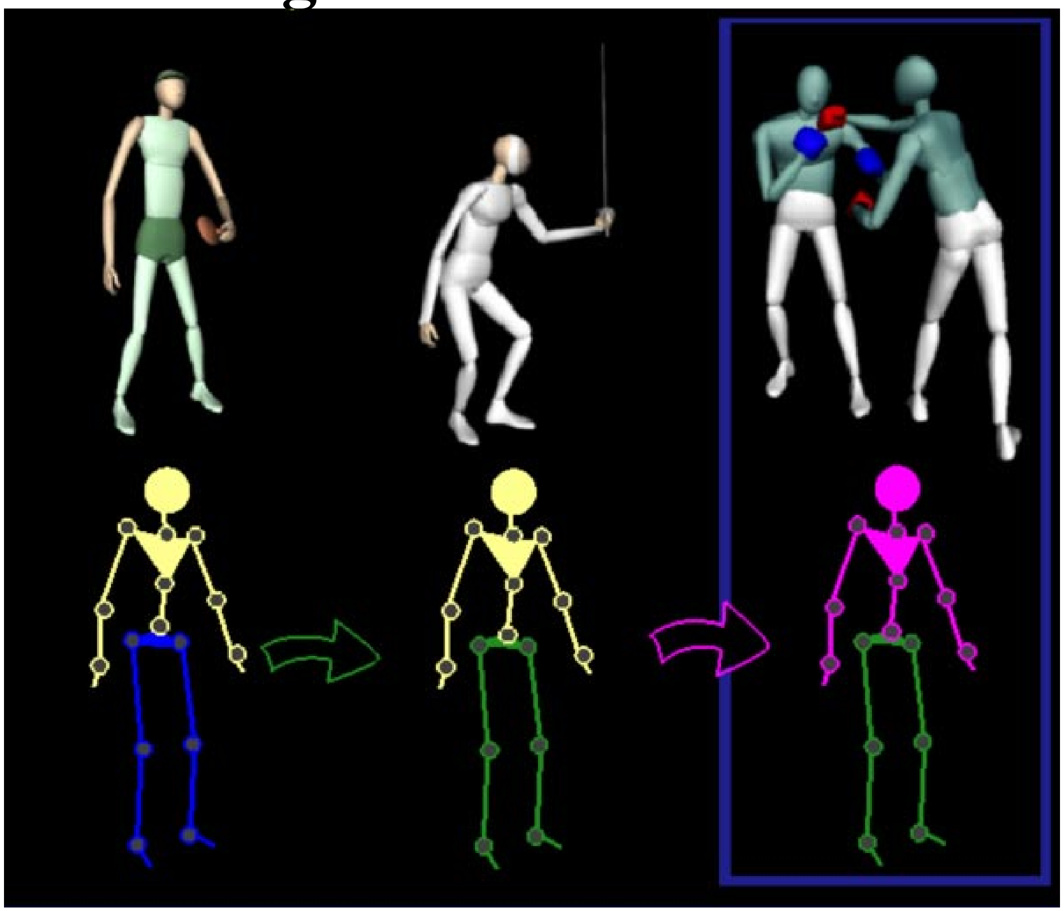
Full-body mocap-driven simulations



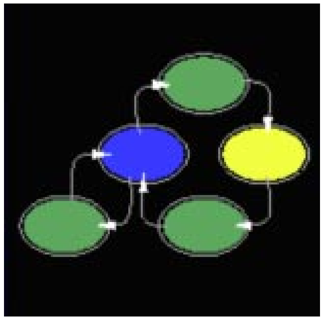
Footwork is nice, but lets see some contact!

Overview:
Control for hitting and reacting

Tracking Balance Control

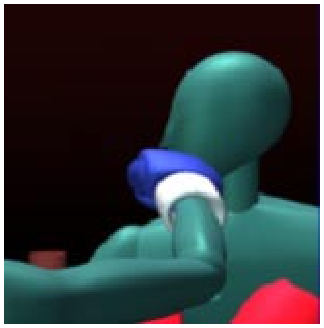


Control for acting and reacting



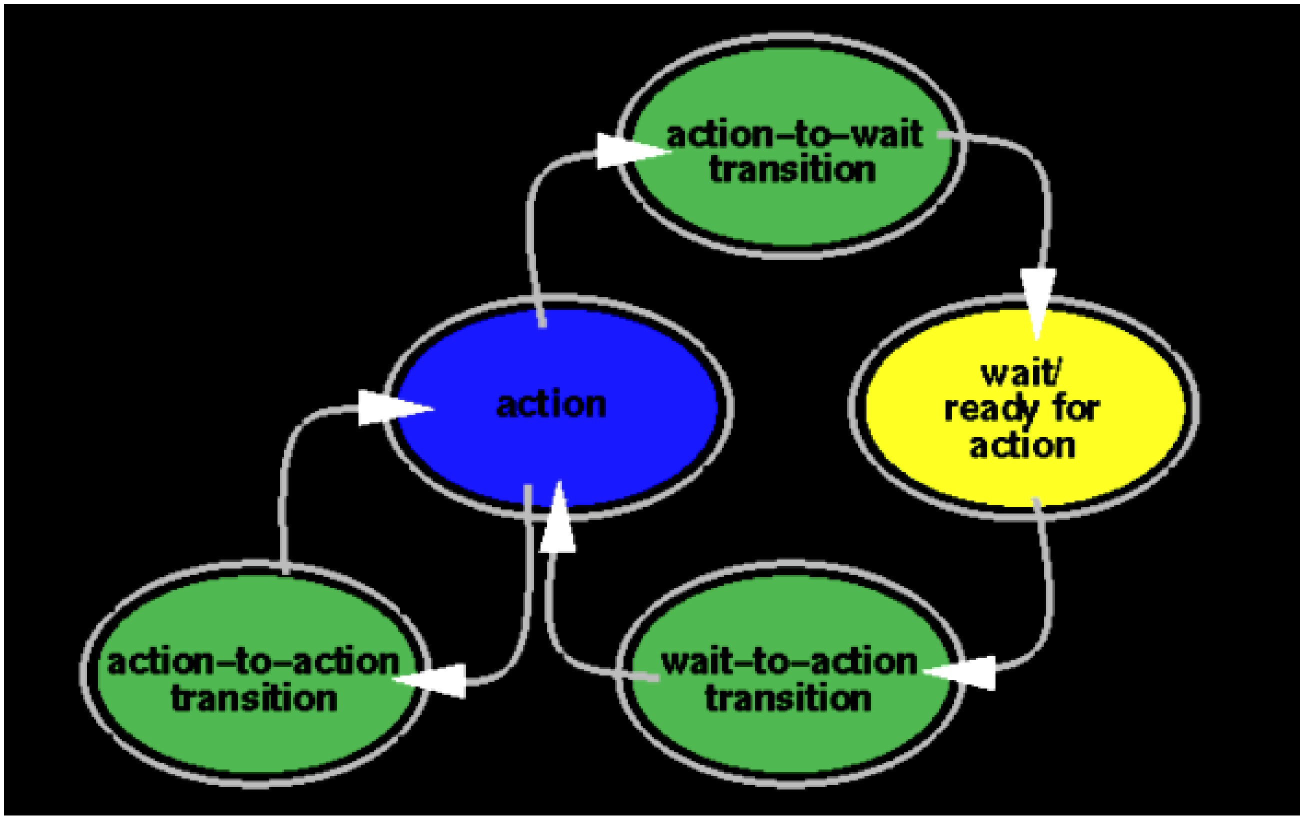
Continuous play
  state machines

Control over actions



Reacting to contact
  collision forces
  gain scheduler

Control for continuous play
Interpolation finite state machines



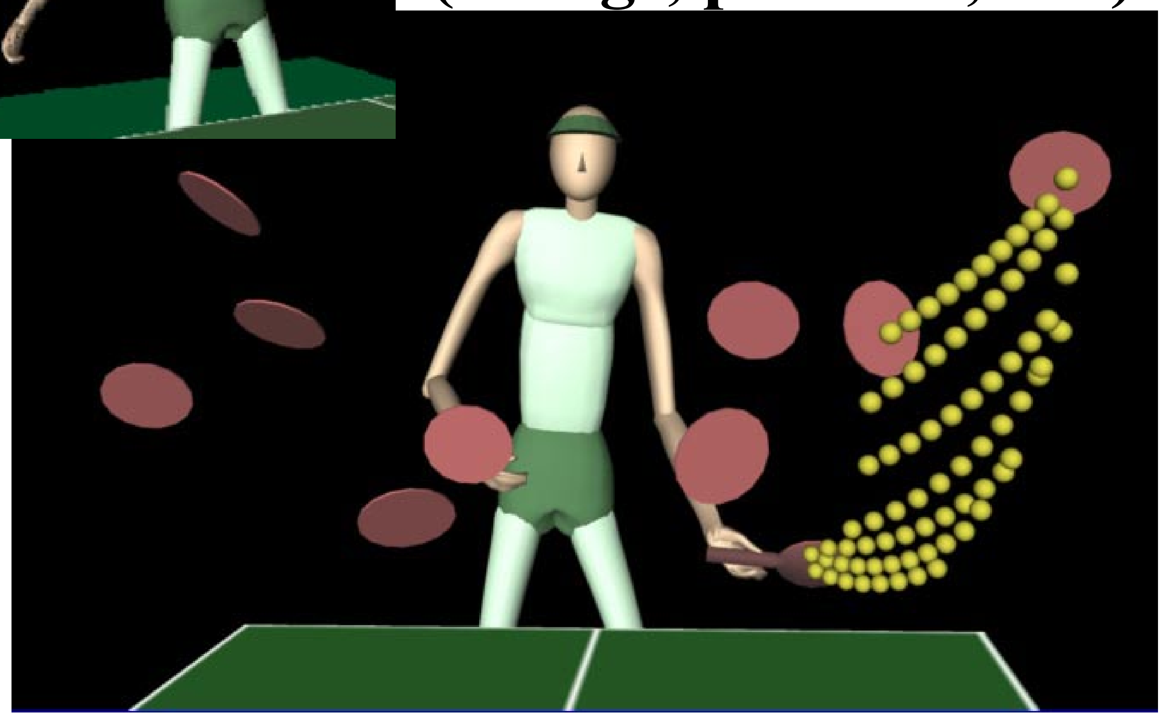
Transitions interpolate (*slerp*) from one mocap clip to the next

Control for (upper body) actions

Editing motion capture, *as usual*

Use motion capture library of examples (swings, punches, etc.)

Interpolation, IK, and warping, etc. for parametric control



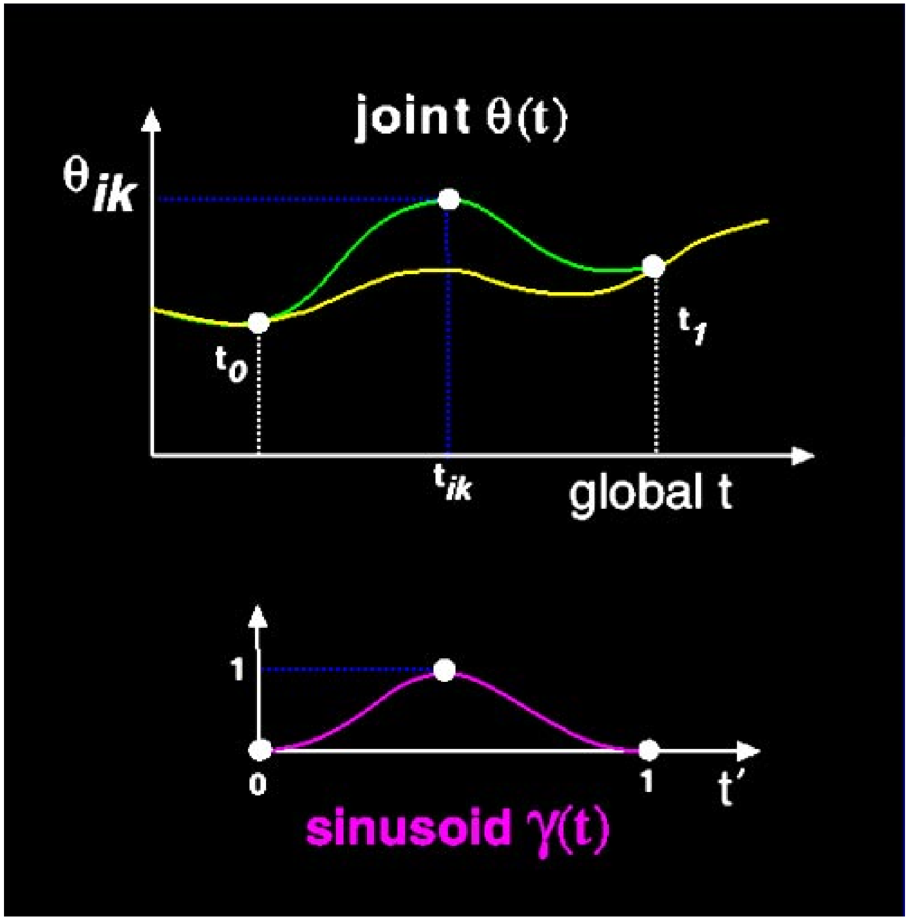
Control for actions
Edit clips for position and orientation

Use IK to *hit* target

Apply IK offsets:

$$\Delta_{offset} = \theta_{ik} - \theta_a(t_{ik})$$

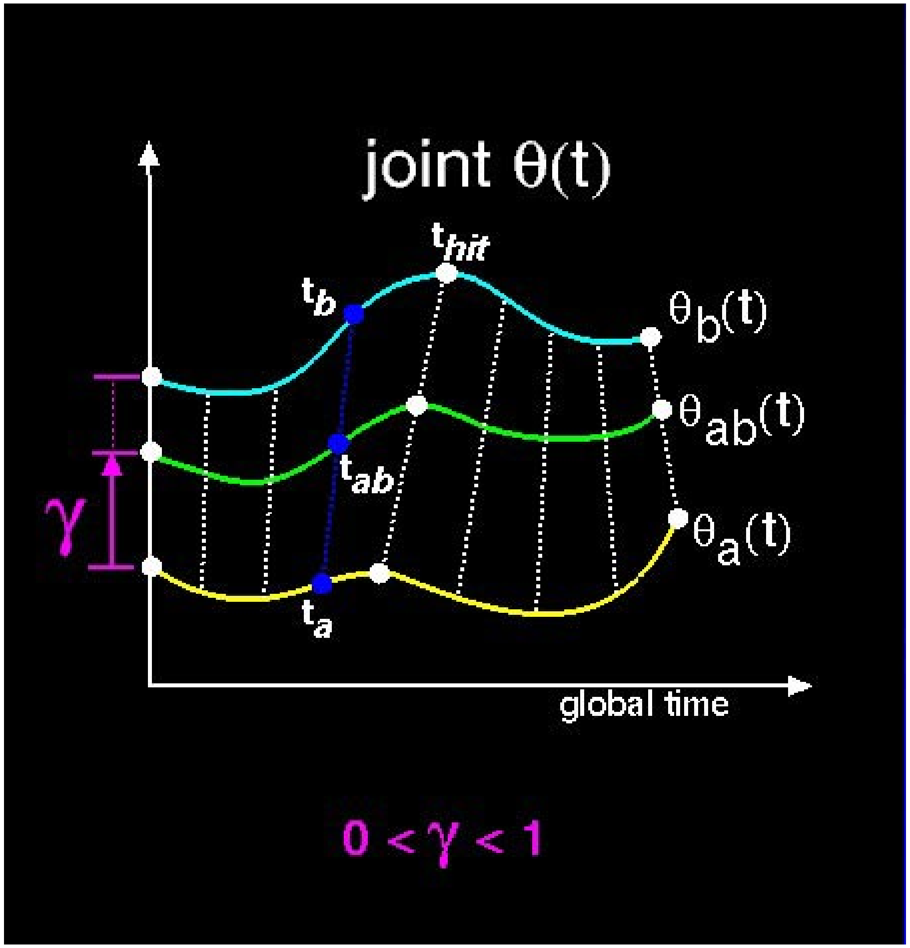
Offsets smoothed further by dynamics



Control for actions
Build new examples *'on the fly'*

Interpolate with any constant value γ to get an in-between action

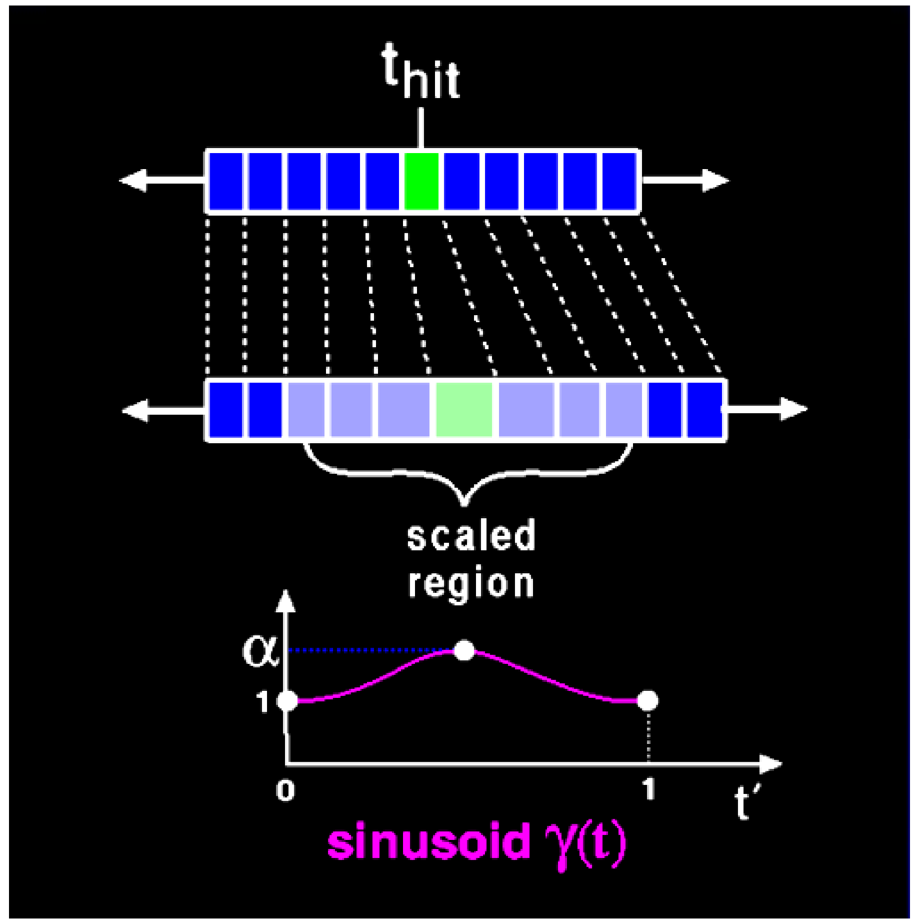
Time-warp to align important features in time: like start, target pt (hit point furthest extent, etc), and end



Control for actions
Speed-up or slow-down only

Speed of end-effector relies on angular velocity:

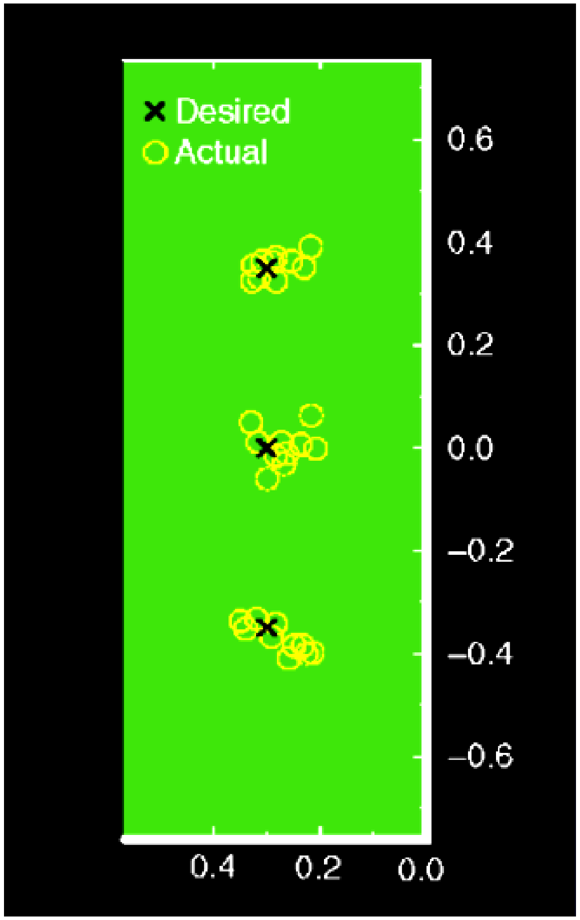
$$v(t) = \sum_{i=0}^{n\ joints} r_i \times \omega_i(t)$$

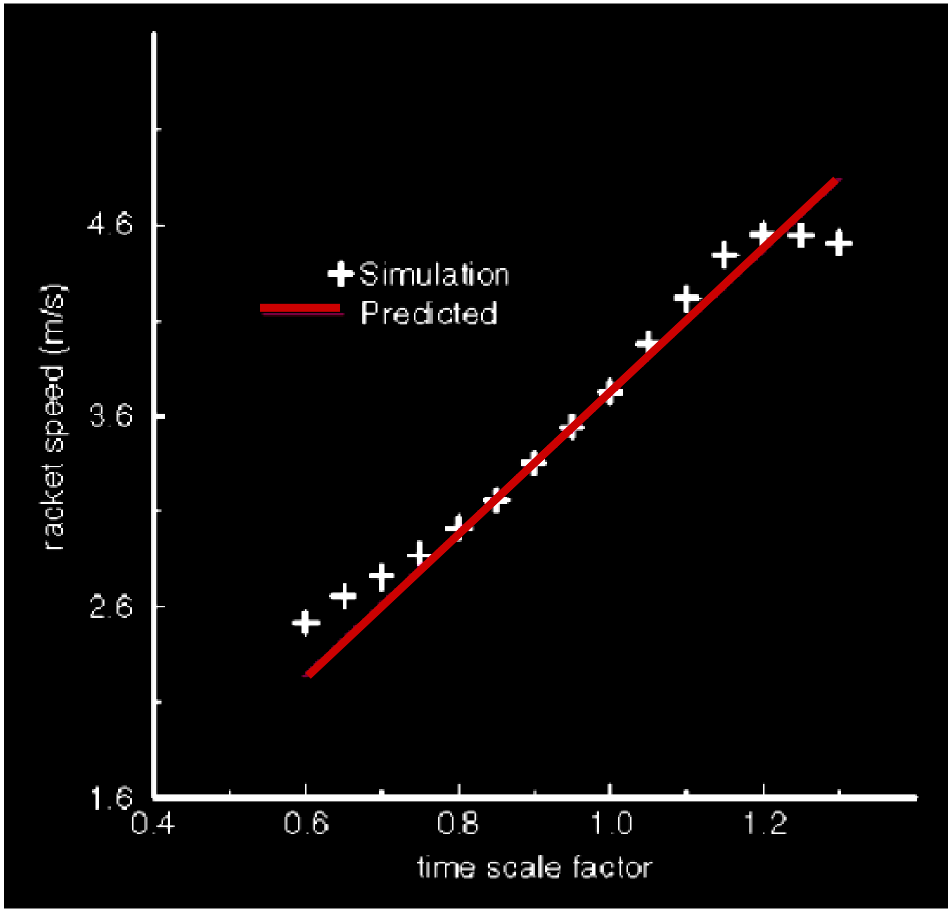


Preprocess to find unmodified speed

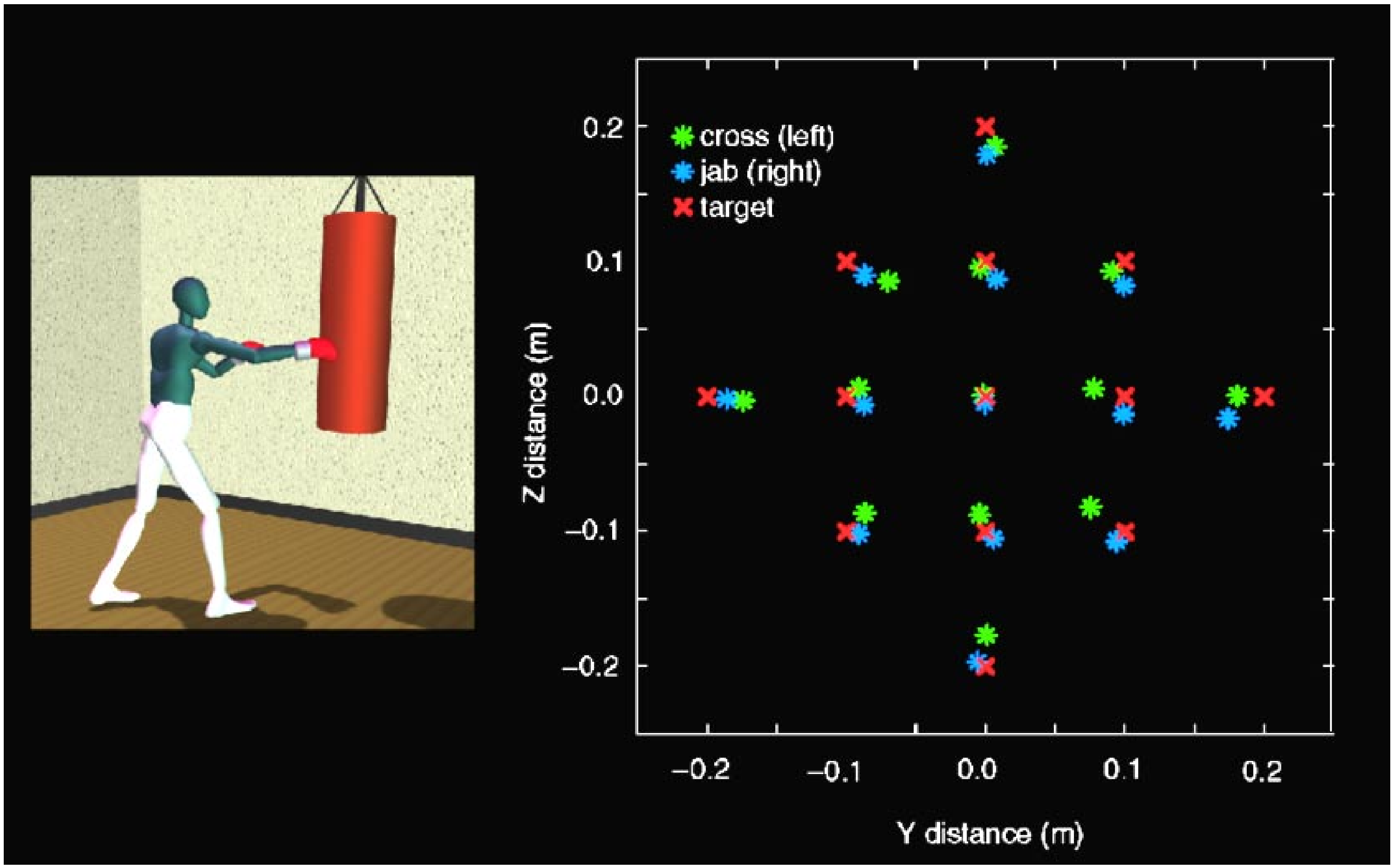
Thenm time-scale by α^{-1} at hit time

Control for table tennis simulation

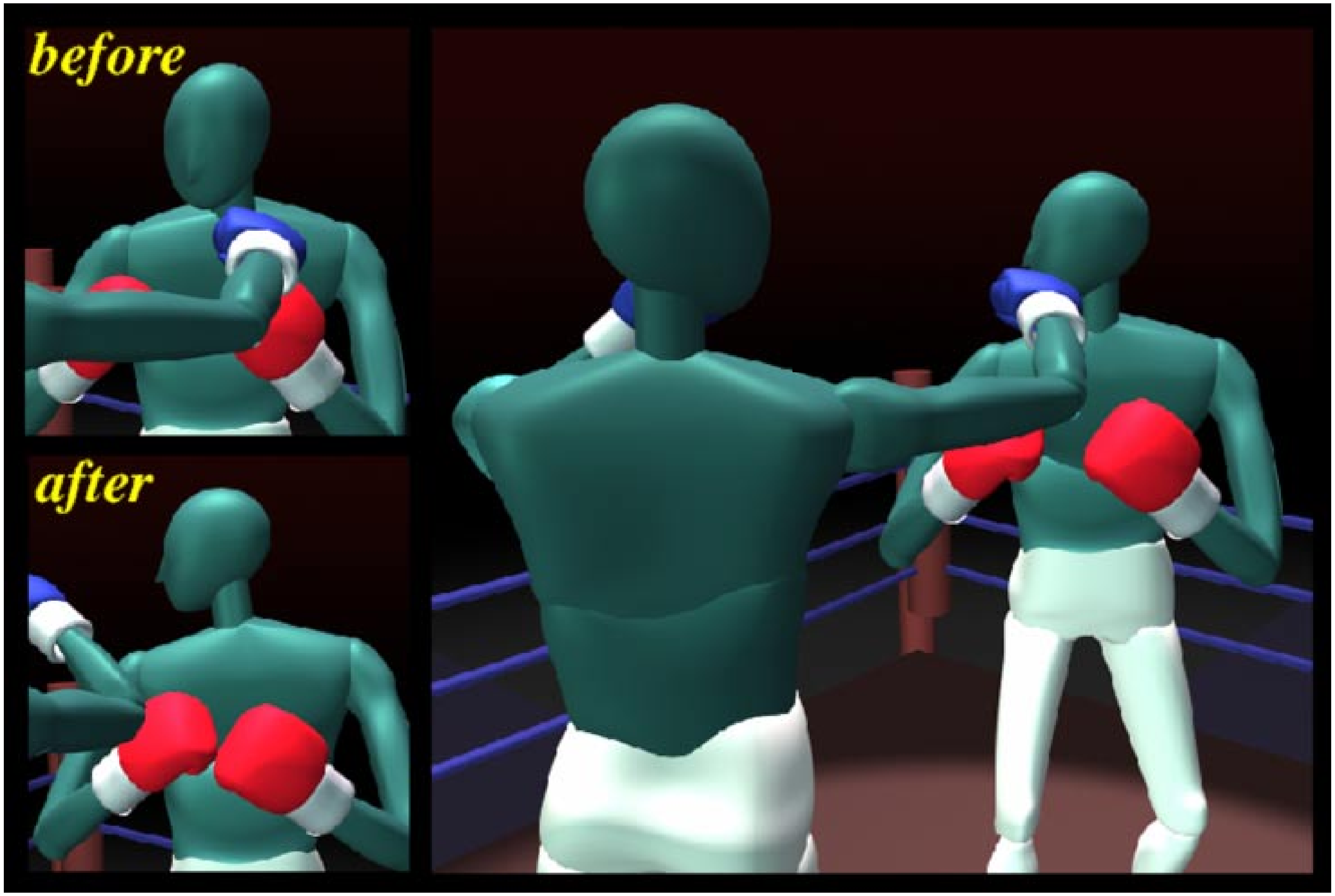




Control for boxing simulation



Control for reacting to contact

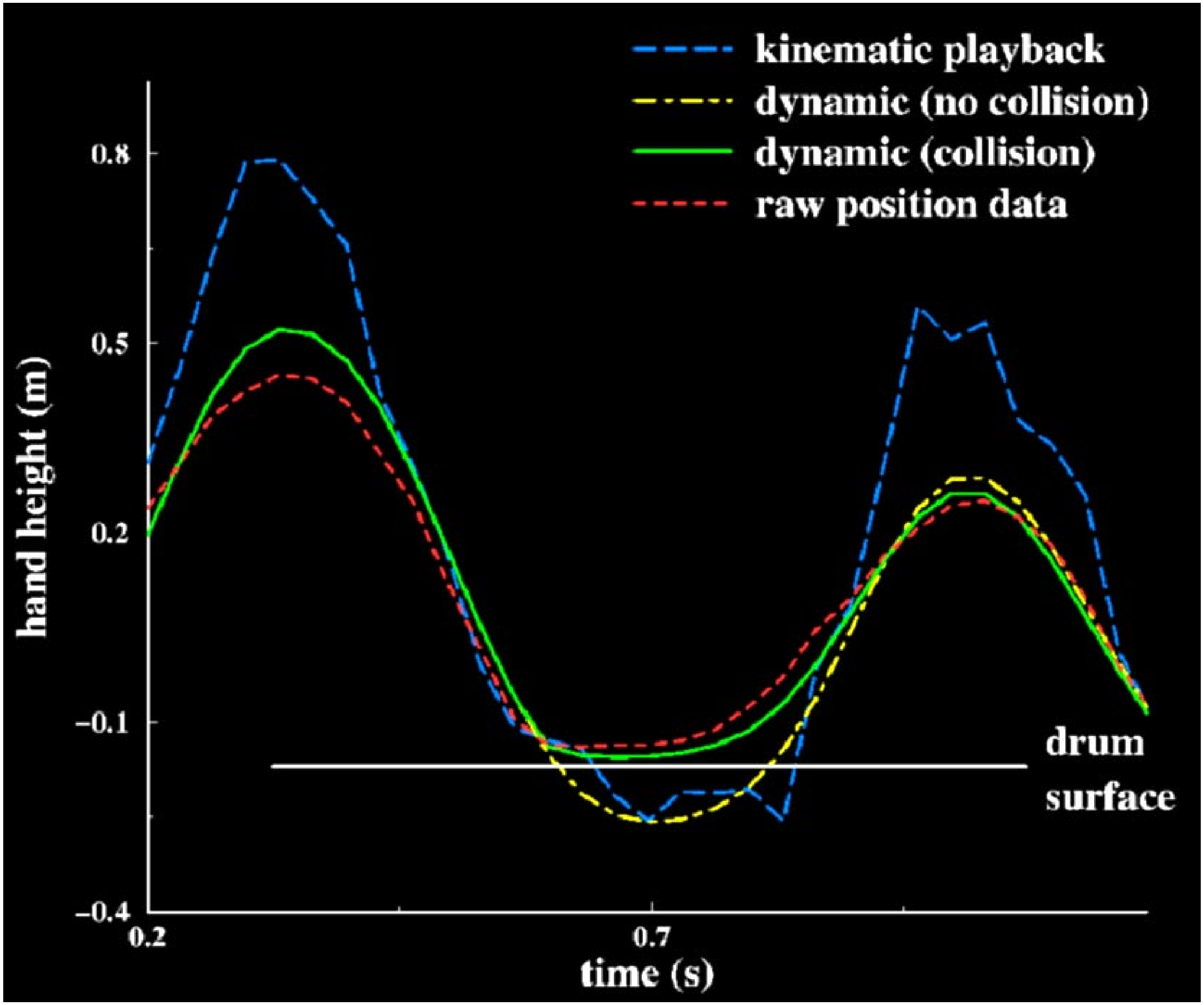


Control for reacting to contact

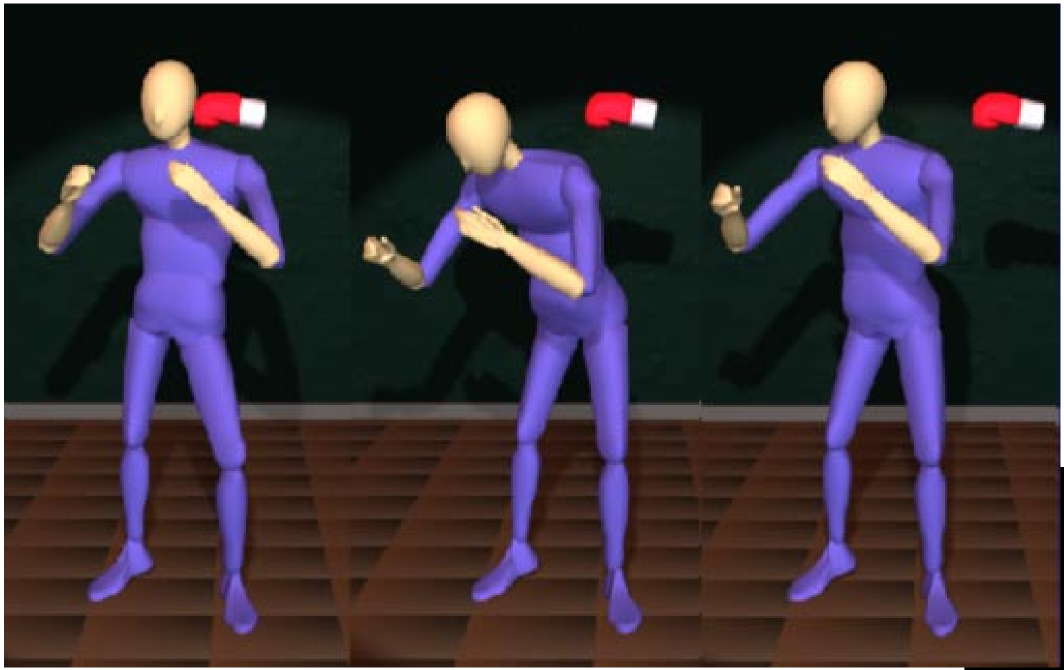
Dynamic impact adds external forces to the simulation

Collision handler detects and computes penalty force reaction

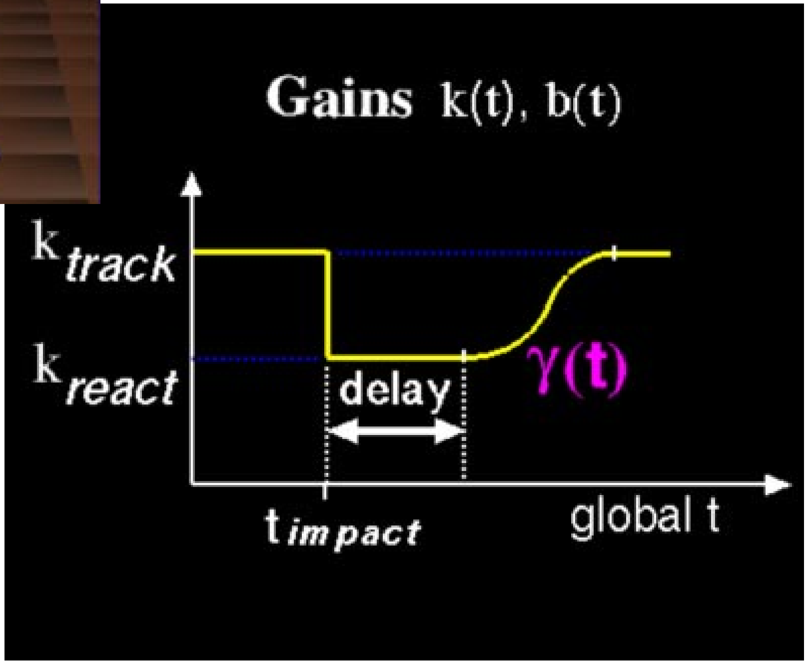
Apply reaction forces


kinematic playback
dynamic (no collision)
dynamic (collision)
raw position data
hand height (m)
0.8
0.5
0.2
−0.1
−0.4
0.2
0.7
time (s)
drum
surface

Control for reacting to contact?

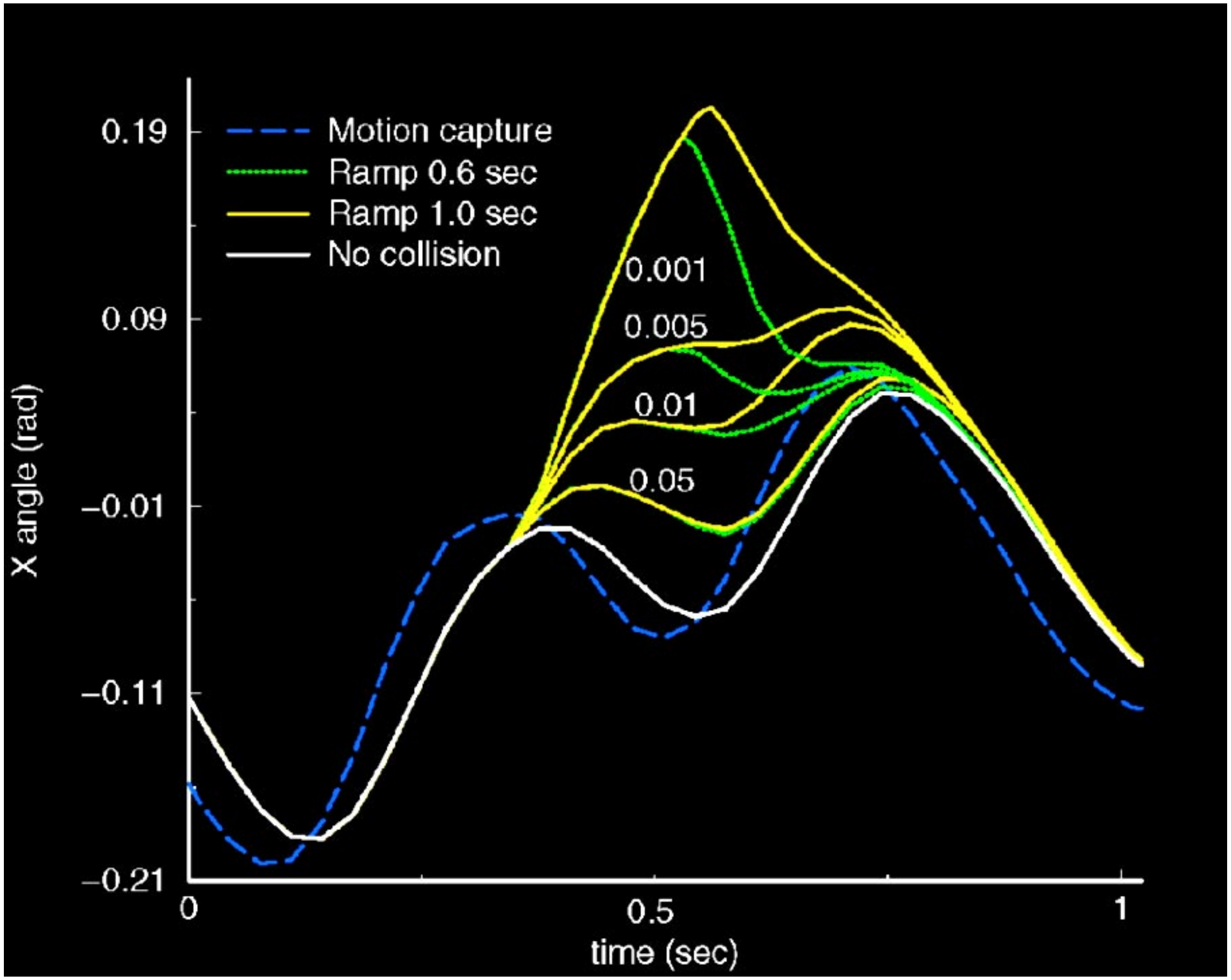


React to forces
Recover smoothly



Lower gain to avoid stiff contact, allows for bigger timestep (overall speed-up)

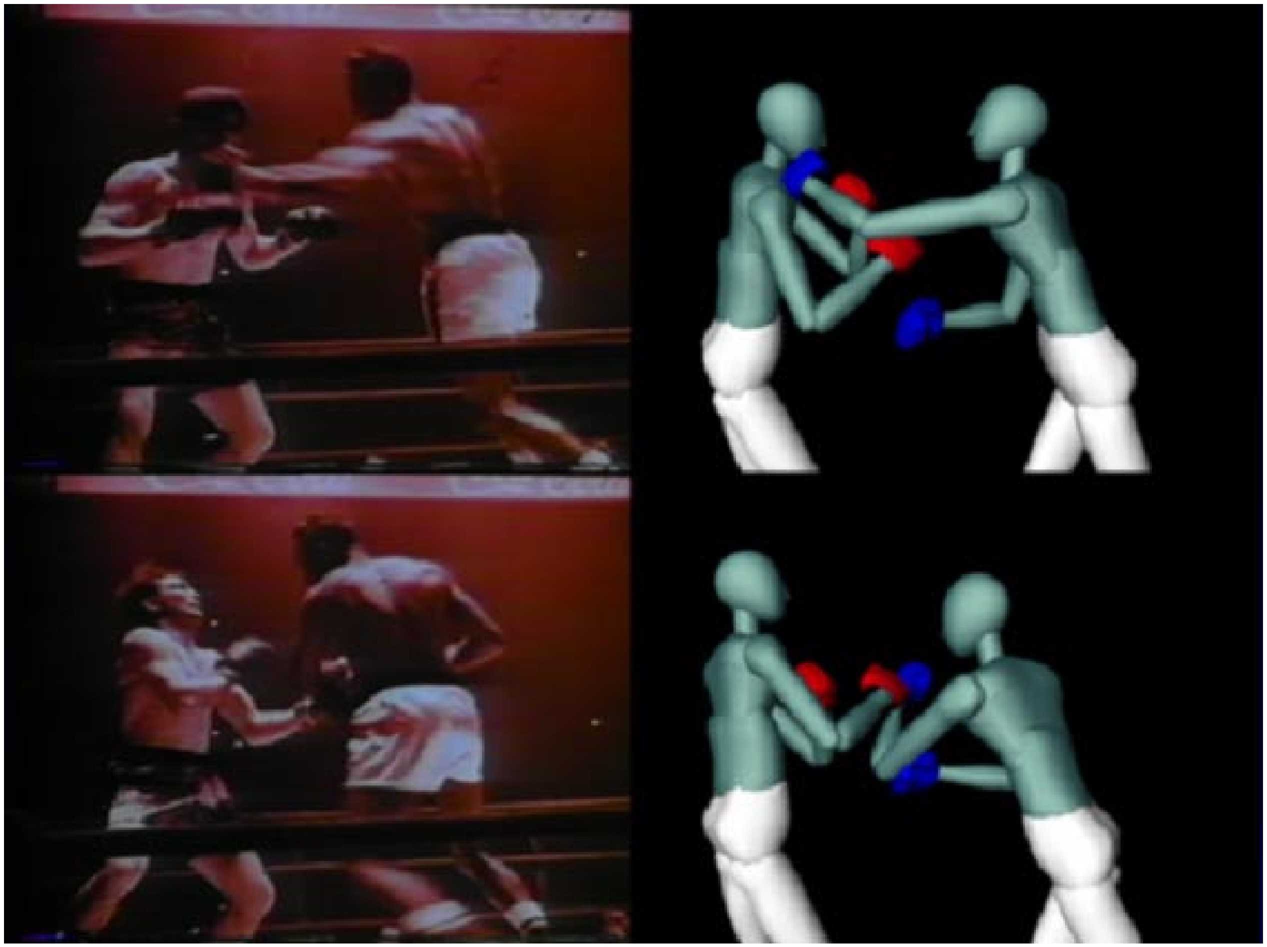
Control for reacting to contact



Creates a nice smooth space (as shown) to give good handle for desired affect

Stiff or loose-looking character can both result, based on tuning

Evaluation: real vs. simulation



the end, right?

No wait, there's more:
TRICKS and CHEATING

Riverside Graphics Lab

Okay, so sims are great, but...

How do we make them easier to contol?

Give up some (small amount) of the realism!

How do we make them fast(er)?

Give up some (more) of the realism!

Do we really need to simulate a full body? Always?

Only have to simulate what is to move based on dynamic effects, the rest can just come along for the ride (kinematically.) Likewise, only need to simulate *when* these affects are actually needed

Speed-ups:

Simulation speed relies on several factors-
But they boil down to two:

Timestep & Compute-time/per cycle

Factors that can affect these:

Integration method -> implicit solvers can take bigger steps in general (but may look over-damped... the tradeoff!)

Methods for solving constraints, especially for resolving contact -> avoid rigid constraints to avoid the need for tiny timesteps

Number of body parts -> the fewer, the faster

Ultimate speed-up: Only simulate what you need, when you need it!

Turn off the sim (change to kinematics) and back as needed, can result in amazing speed-ups, but need to make good switches between representations

Shapiro and Faloutsos ('03) offer some answers

Use *level-of-detail* to simulate only needed motion and complexity (and cull when off camera)

Carlson and Hodgins ('97) discuss this topic

Simulate only the arm or leg (or whatever) in contact and use the kinematics and mocap for the rest (*hybrid model*)

(Already seeing this in some games!)

How do we make control easier?
CHEAT (on the physics that is)

Riverside Graphics Lab

Once the academics wash up and go home, developers are left to fill in the details

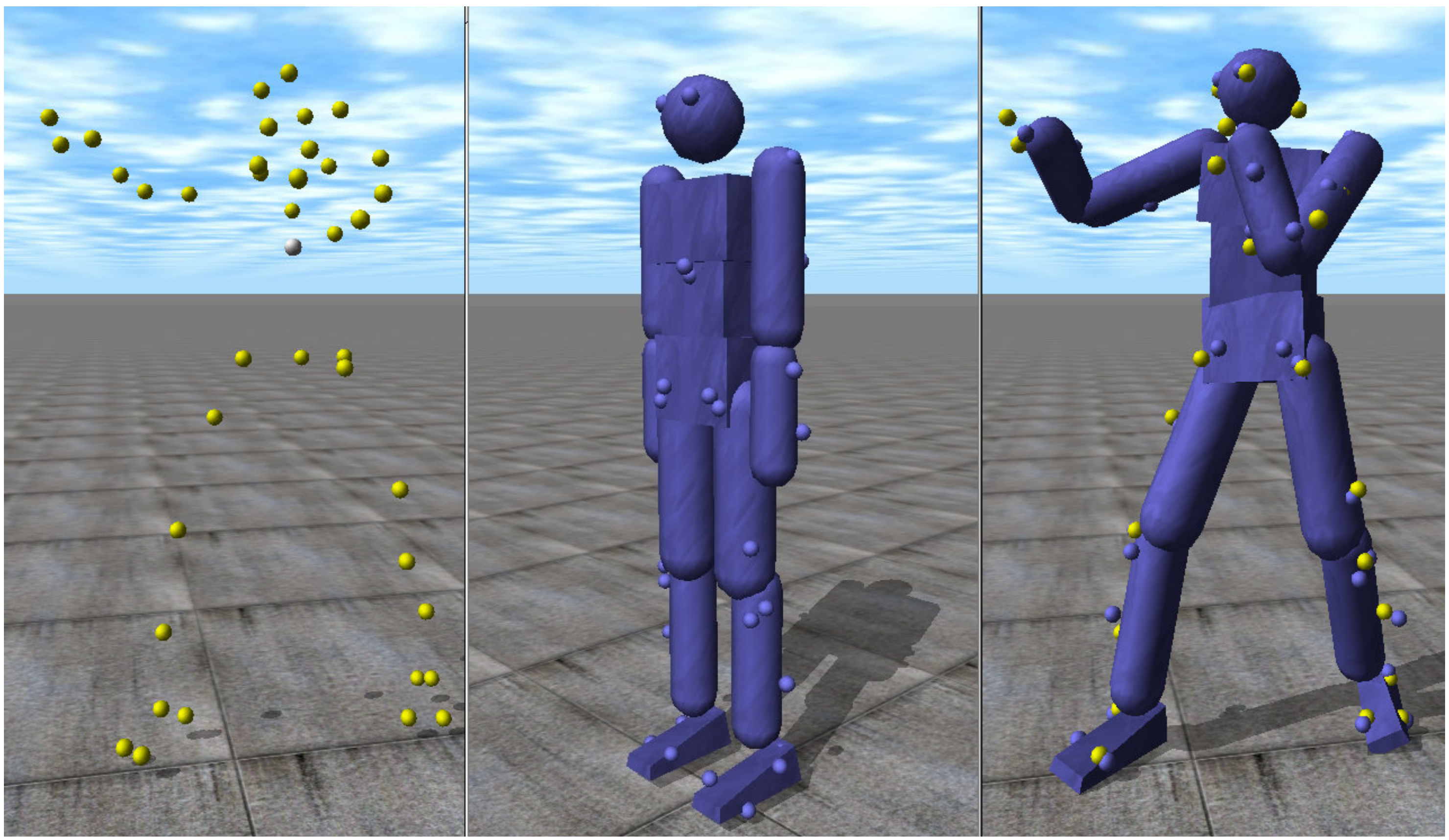
Physics in games only needs to be used when it adds to the look or gameplay. And nobody requires developers to 'play by the rules' so...

How about for starters, lets avoid torques (So unintuitive!) & apply forces, any force will do (legal or not)

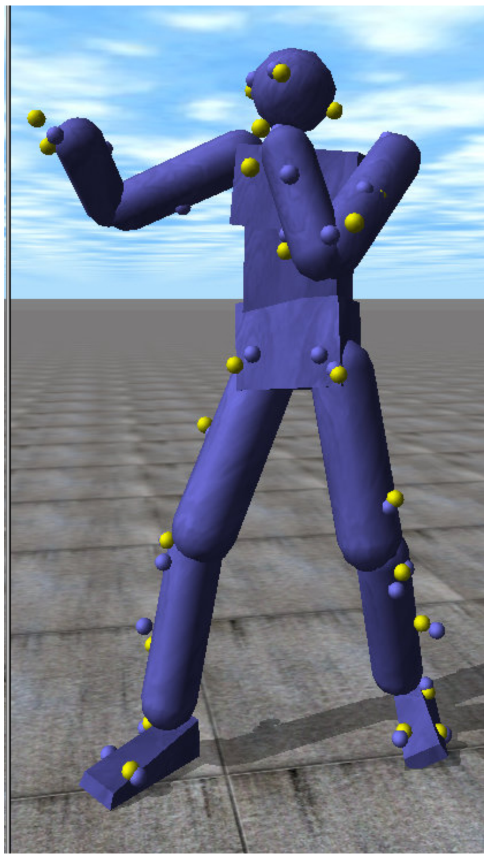
And, why do real balance control (Hard!) when there are perfectly good fake balancers that are easier to control and can result in 'pretty real'-looking motion?

Shameless plug: **We've worked on using a sim to map data to new characters while adding in ground forces (Zordan & Horst 03)**

Optical data + Simulation Posture



Use this same technique for:
Force-based control

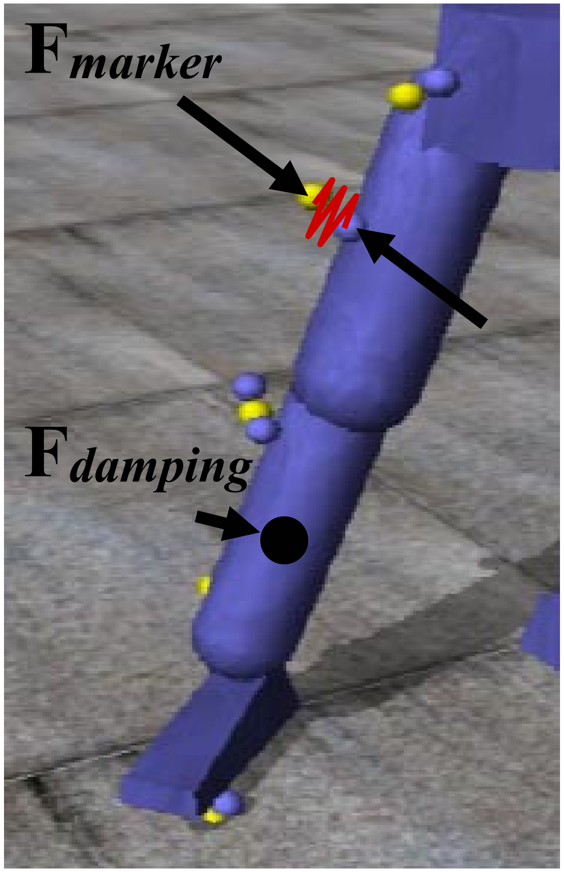


The technique controls the sim to move 'like' the actor based on the mocap, by attaching the mocap markers to the landmarks on the sim using springs and dampers

This method makes controlling easy but doesn't guarantee good reactions... must manage separately

Force-based control

Matching virtual 'landmarks' guide the simulated bodies to follow the markers using *intuitive* forces



Springs pull the simulation to the marker data

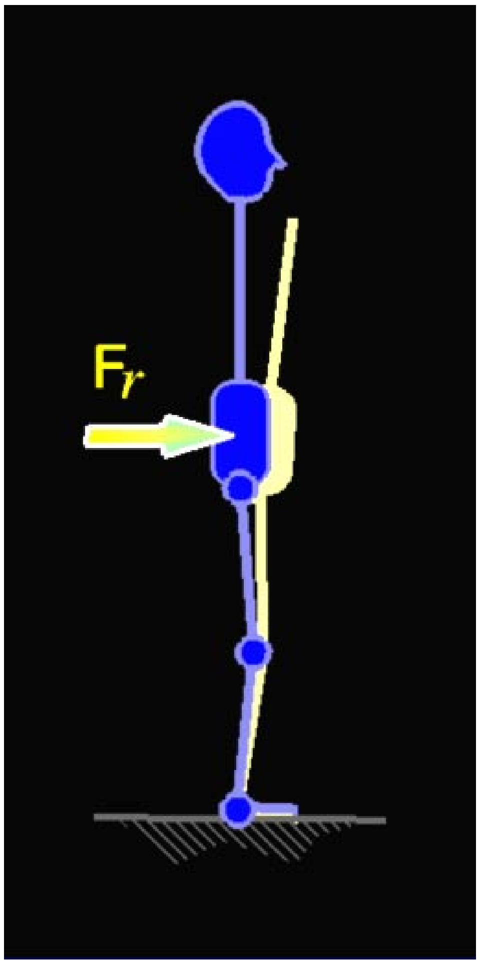
$$F_{marker} = -k_f X_{error}$$

Body forces damp motion

$$F_{damping} = -b_f V_{body}$$

CHEATING in lower-body control:

Use an external balancing force ("Hand of God" van de Panne 95)



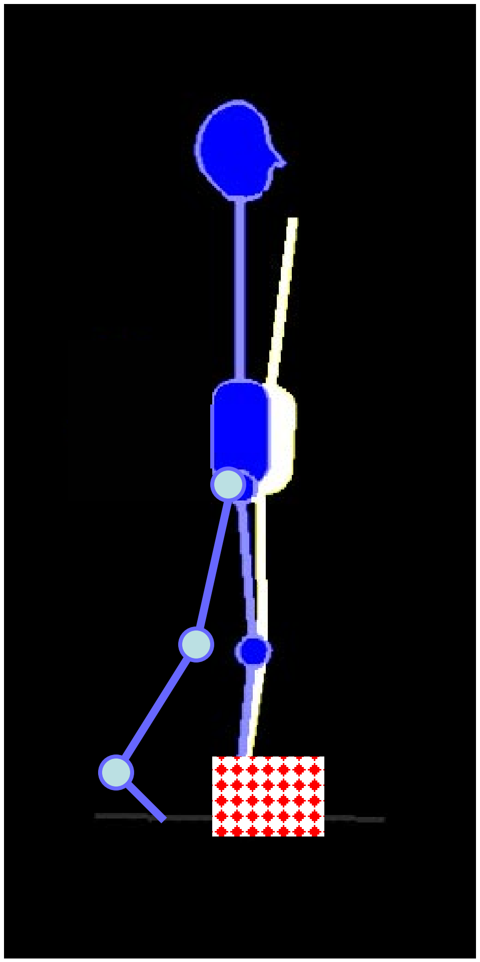
If the force only gets applied horizontally the sim will be standing on its own but just won't be **"balancing"** on its own

Cut the force when it gets too large and the sim will fall, ramp it down, cap it, plenty of options here to get 'the right look'

$$F_{r\,(x,y)} = k_r\,(\,err\,) - b_r\,(\,\dot{err}\,)$$

Riverside Graphics Lab

CHEATING in lower-body control
Or glue one foot (or both) to the ground



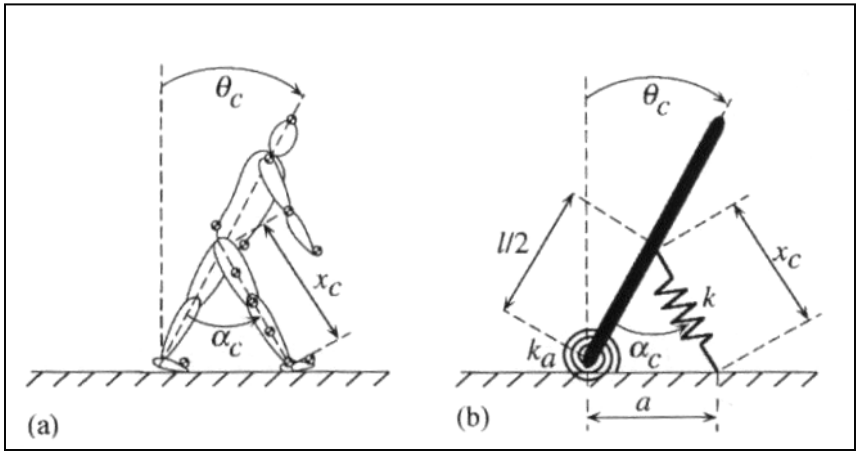
If one foot is fixed to the ground, the whole body will move but it won't fall. Gravity can still act & look right as long as the other foot can contact the ground

Let the 'glued' foot pivot on the ground for further freedom, or add a spring to mimic ankle activation

Again turn the glue off when things are 'out of balance' and let the sim fall over

Incidentally, this kind of CHEATING doesn't mean it won't be realistic...

Biomechanists study balance/falls this exact way:



(Hsai, 99)

with a spring between the ankle and the ground!

Can use simple active control to 'catch' or prevent falling
Also could use the upper body for balance, too
waving arms, etc.

Conclusions

Motion capture and dynamics are a powerful combination but does not solve the whole control problem

Hybrid dynamics/kinematics approaches will likely beat out pure dynamics alone because they provide robust control and *'unreal'* results