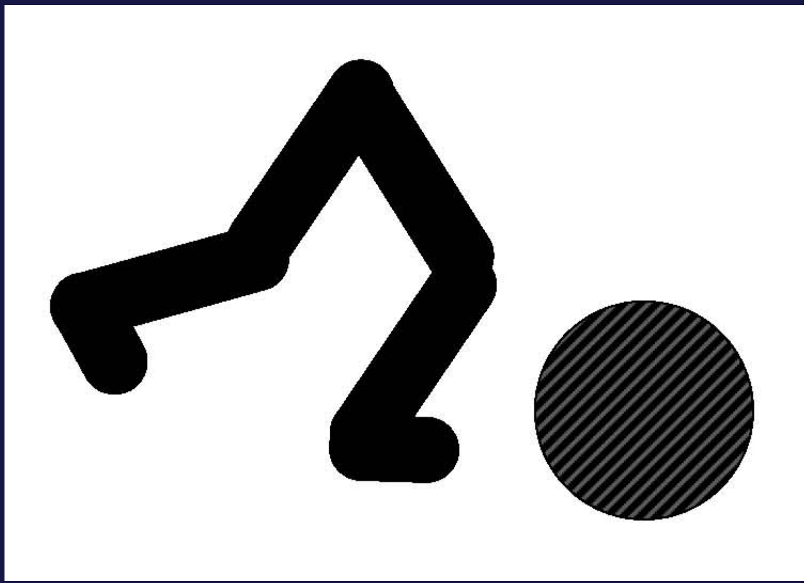
Simulation (three lectures)

How used in games?
Dynamics
Collisions—simple
Collisions—harder
  detection
  bounding boxes
  response

Controllers
Mocap + simulation
User control
What is the future?



Credits

Demos, slides, figures from

- Michiel van de Panne (UBC)
- Michael Mandel's talk at GDC (CMU alum)
- Victor Zordan (UC Riverside)
- Petros Faloutsos (UCLA)

Difficulties of Controller Design

Difficult to design complex coordination of limbs

Results can look stiff and unrealistic

More ballistic: not so many DOFs to specify directly

Control

Joint-level Control

- pose control—poses specified by artist
- continuous control—tracking mocap or programmer-specified function

Hierarchical Control (layered)

- State machine picks low level controller based on sensors or timing
- Low level controller controls joints

Combined approaches

Joint-level Control

Proportional-Derivative (PD) Controller

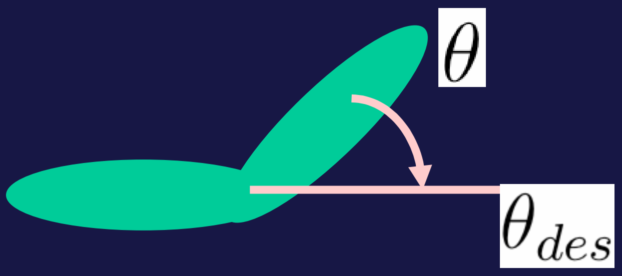
Actuate each joint towards desired target:

$$\tau = k_s(\theta_{des} - \theta) + k_d(-\dot{\theta})$$

θ_{des} is desired joint angle and θ is current angle

k_s and k_d are spring and damper gains

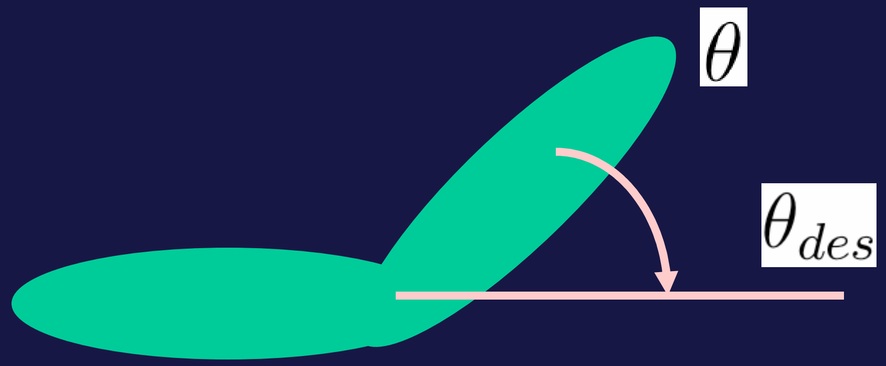
Acts like a damped spring attached to joint (rest position at desired angle)



Control

What should k's be?

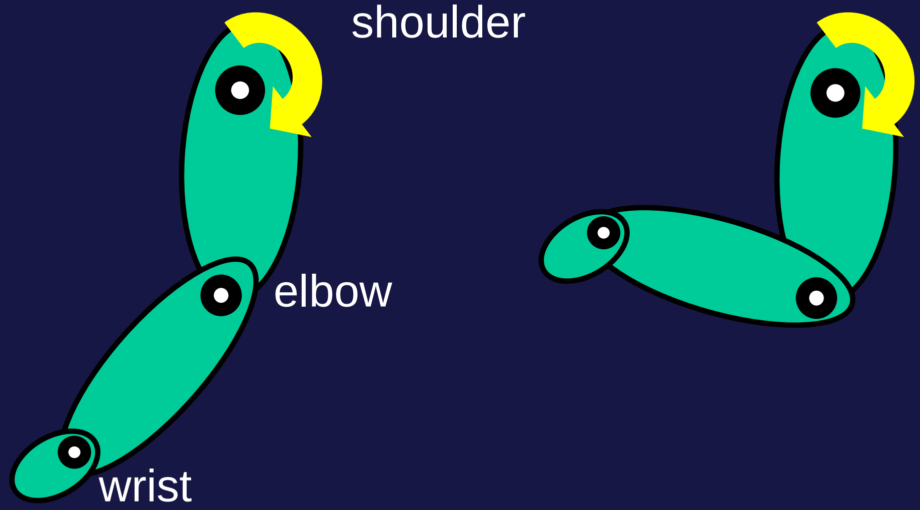
Where does θ_{des} come from?



Choosing Controller Gains

Gains are often hand tuned (tedious for 15x2 or more!)

Reduce tuned parameters to a single spring and damper: scale by effective MOI of the chain about the joint



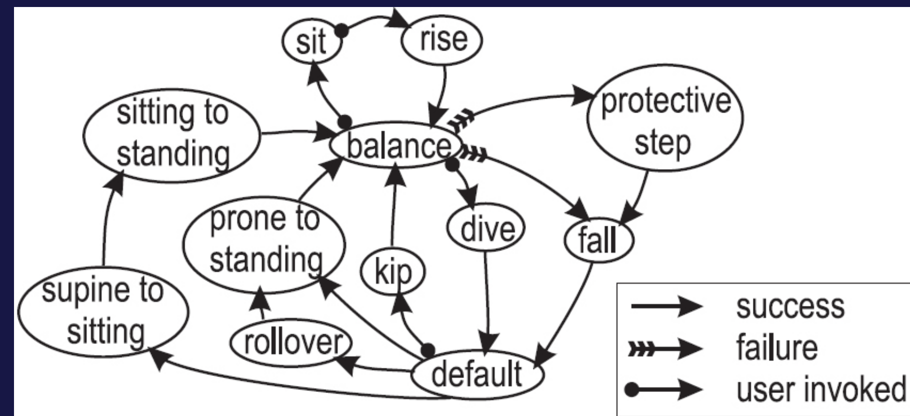
Perhaps more like natural dynamics of a behavior

(see [Zordan '02] for more...)

Pose Control

- Artist selects key poses and dynamics interpolates between them
- Very effective but requires patience and tuning
- Diving (Wooten)
- Getting up (Faloutsos)

Complex Behaviors from Simple Behaviors (Faloutsos 01)



Build basic pose controllers

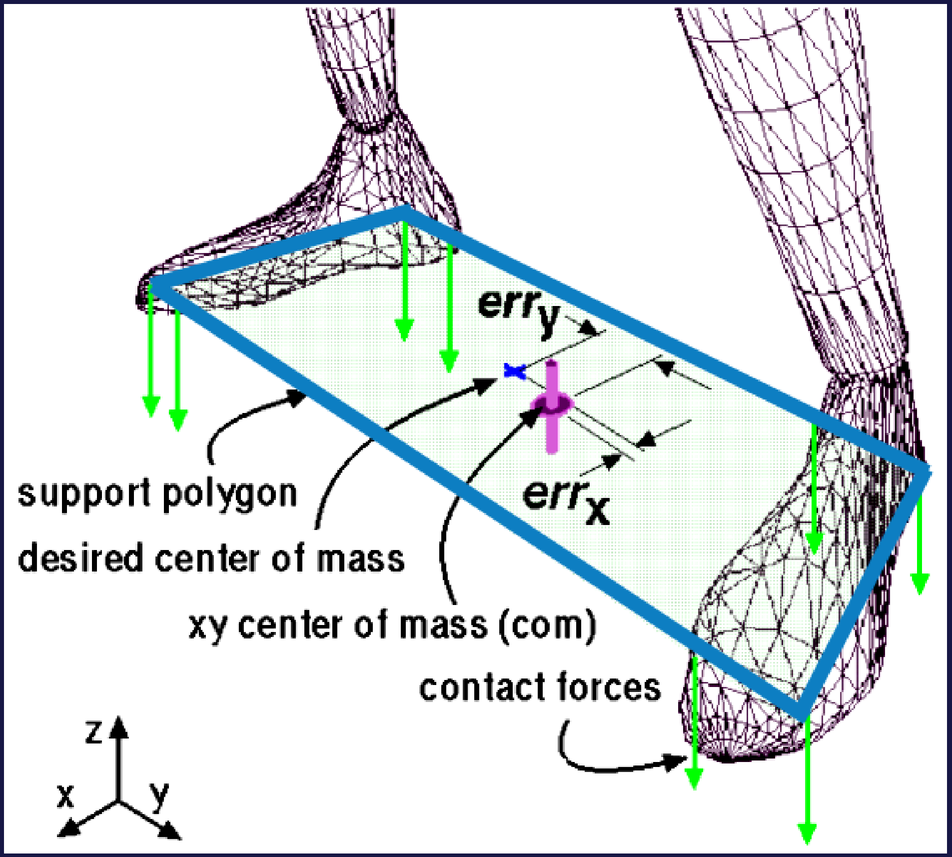
Classify transitions between behaviors

Supervisory controller swaps between them when conditions met

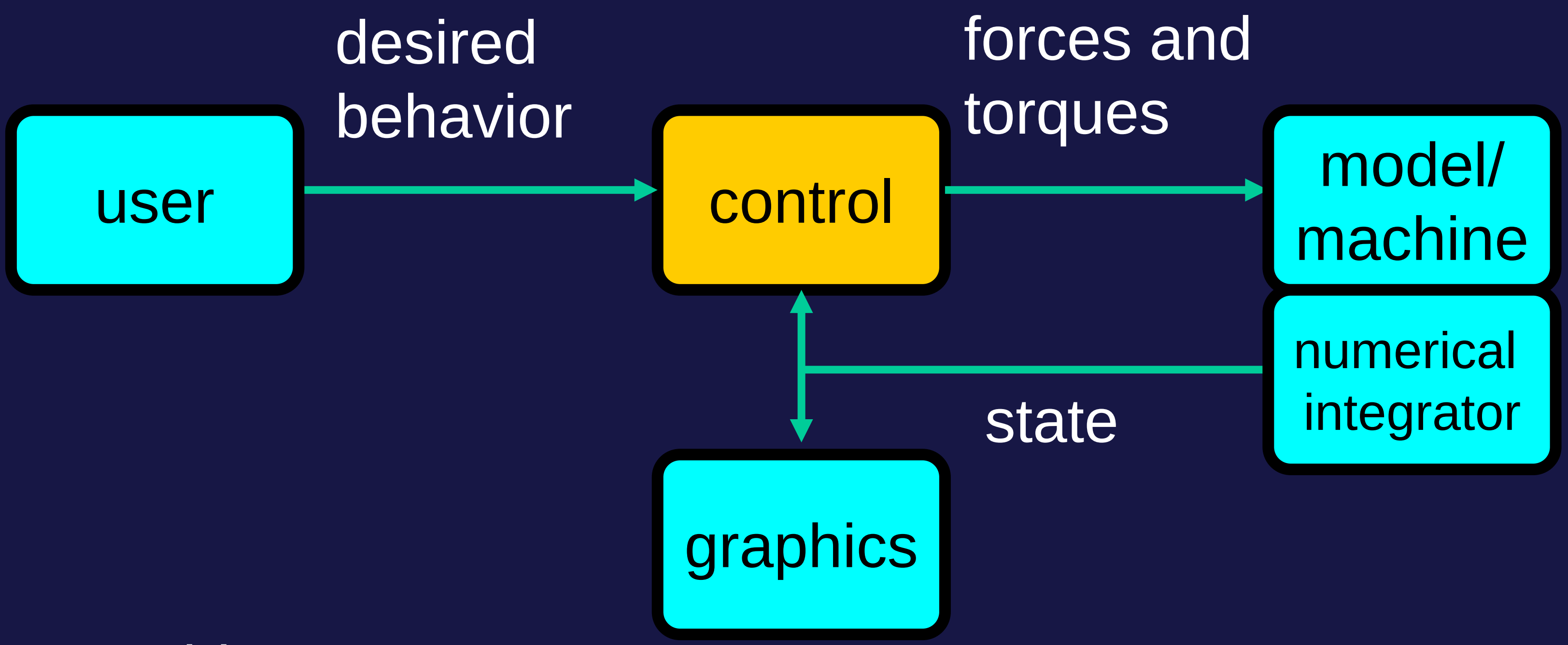
Balance: Programmer specified function

Goal: Keep the center of mass (COM) inside the support polygon

Pick a desired COM and minimize errors by making corrections in the desired angles for ankles and hips



Hierarchical Control



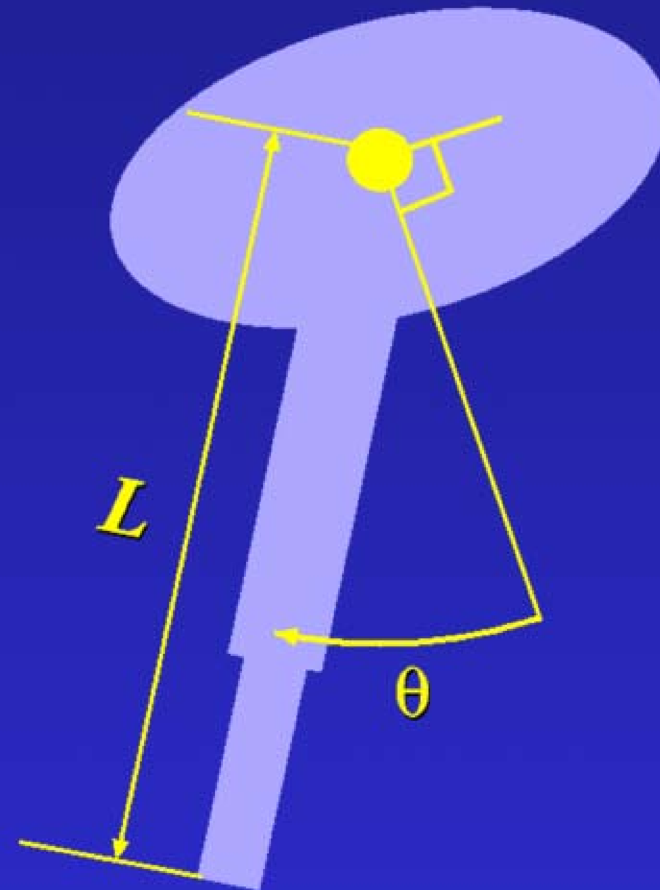
State machine
Control actions
Low-level PD servos

Hopper: Dynamic Model

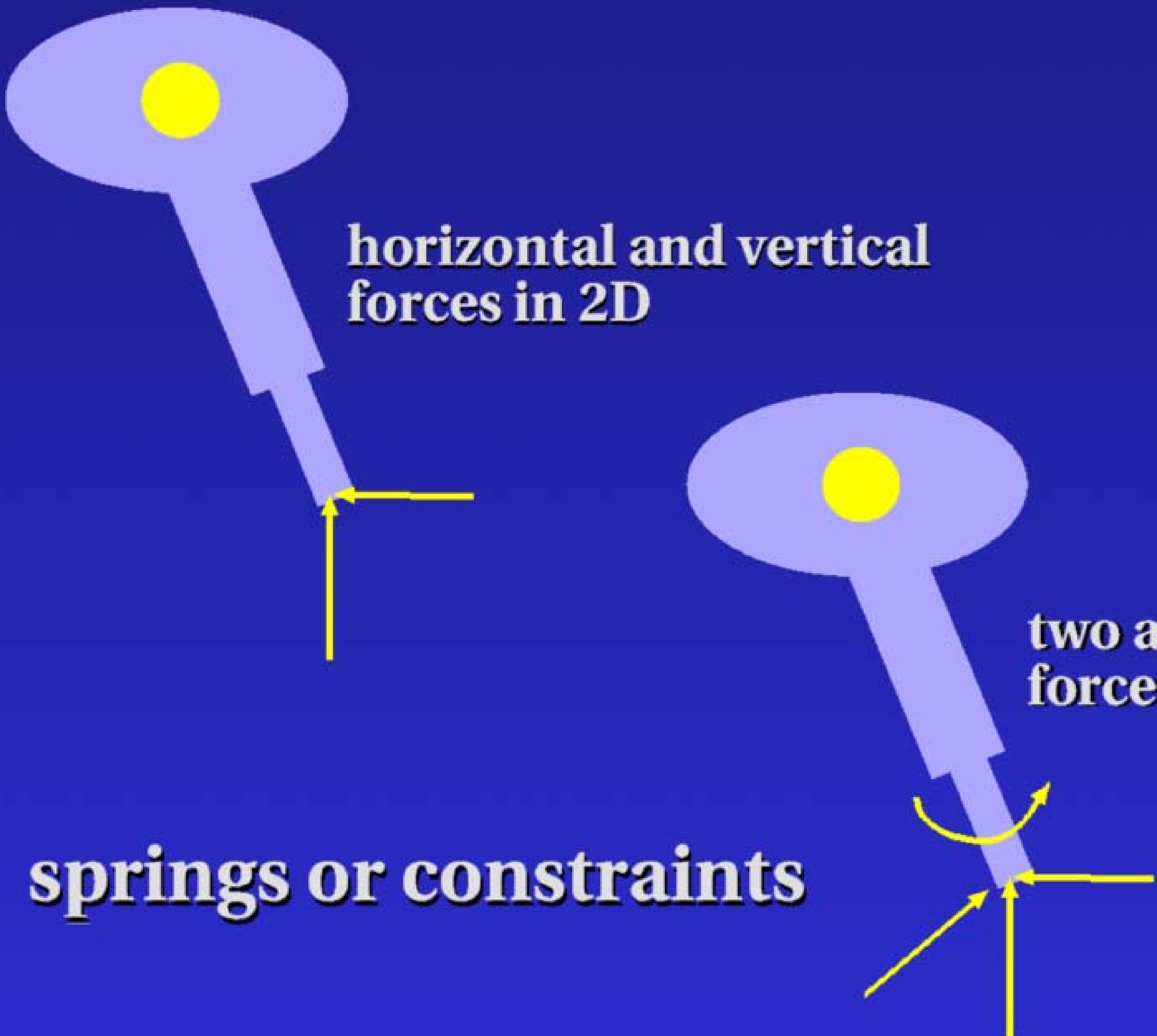
3 rigid bodies

2 controlled degrees of freedom for 2D

4 controlled degrees of freedom for 3D



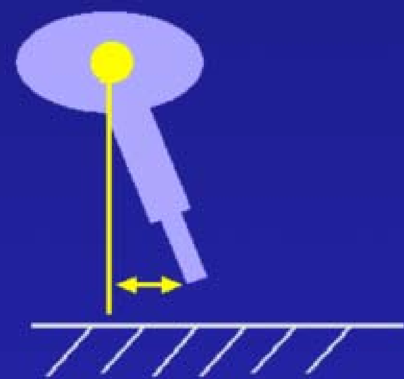
Ground Contact Model



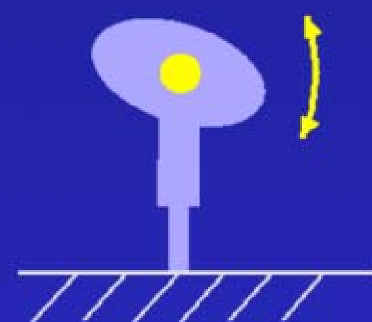
springs or constraints

Control of Hopping

Velocity



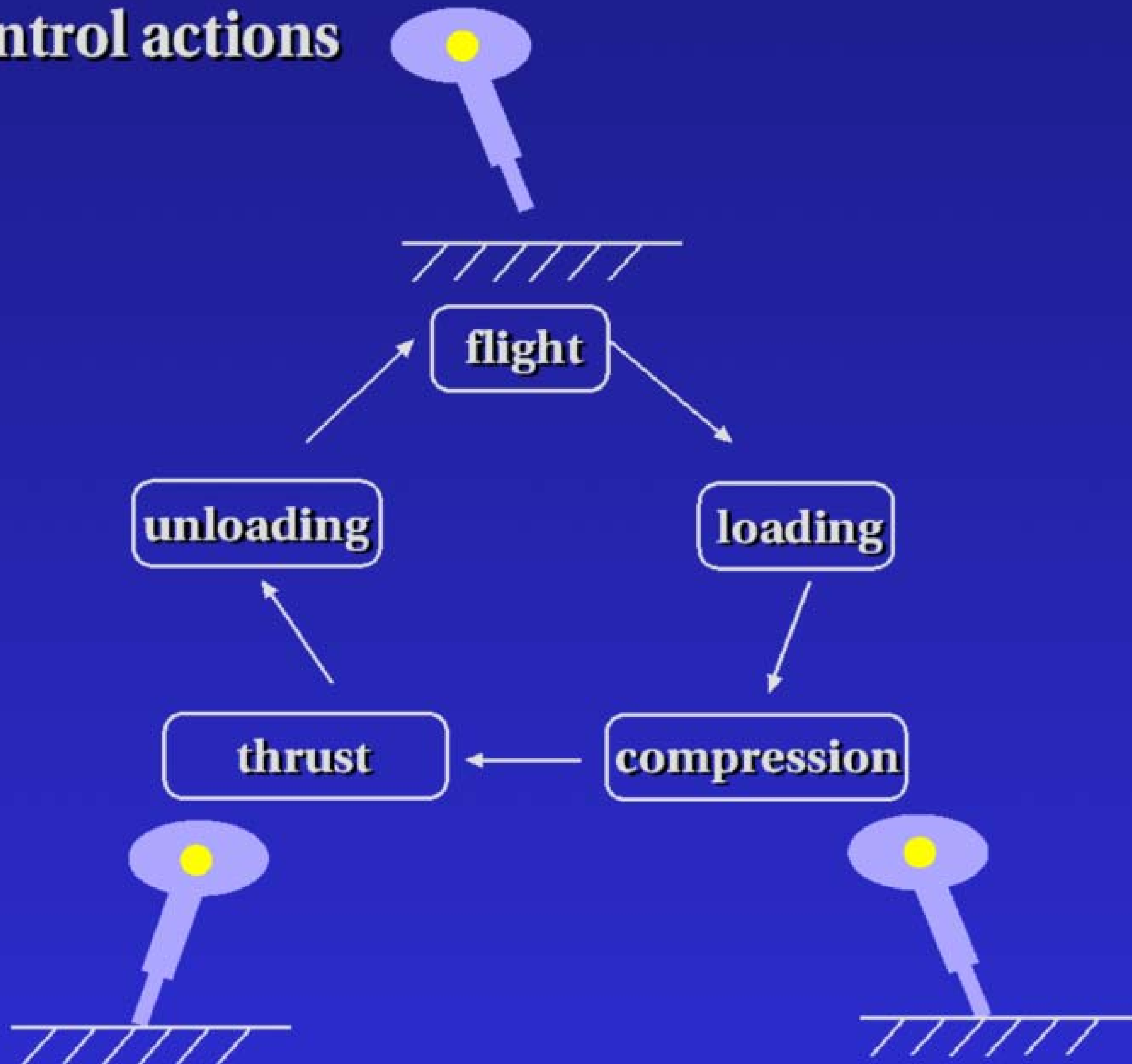
Body attitude



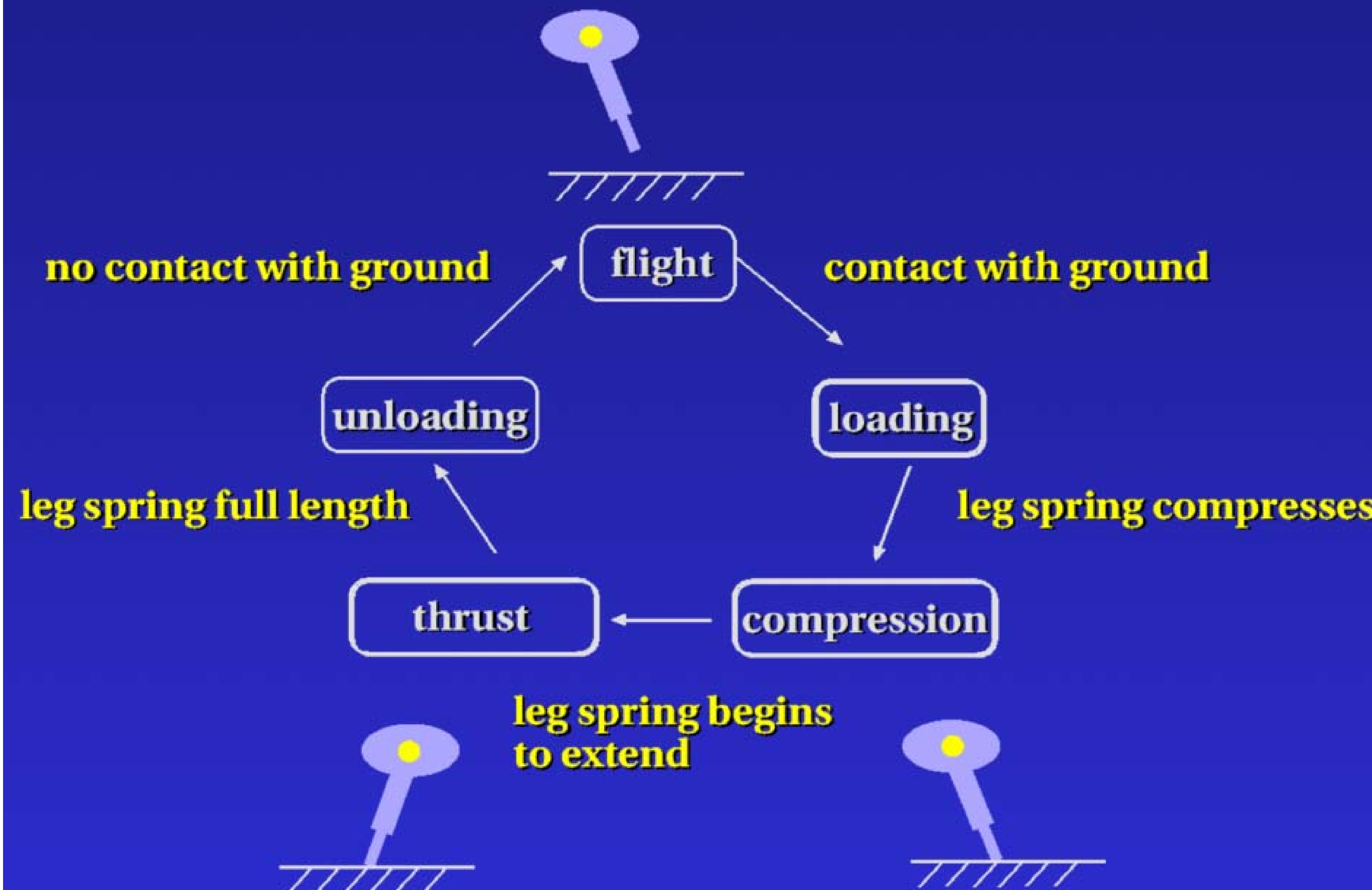
Hopping height

State Machine

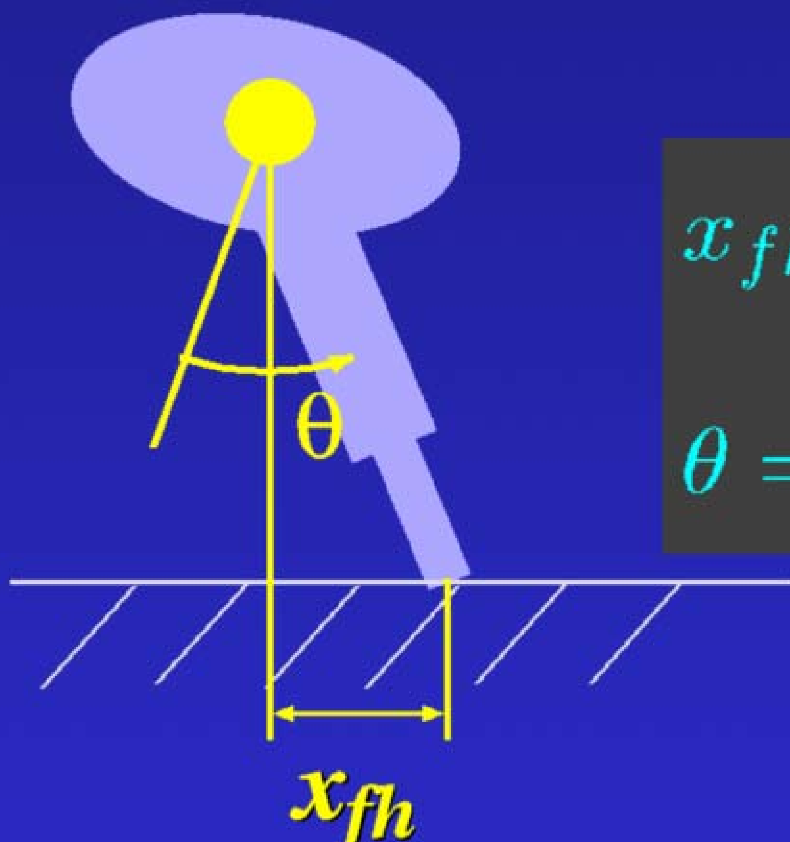
structure control actions



Transitions for State Machine



Velocity

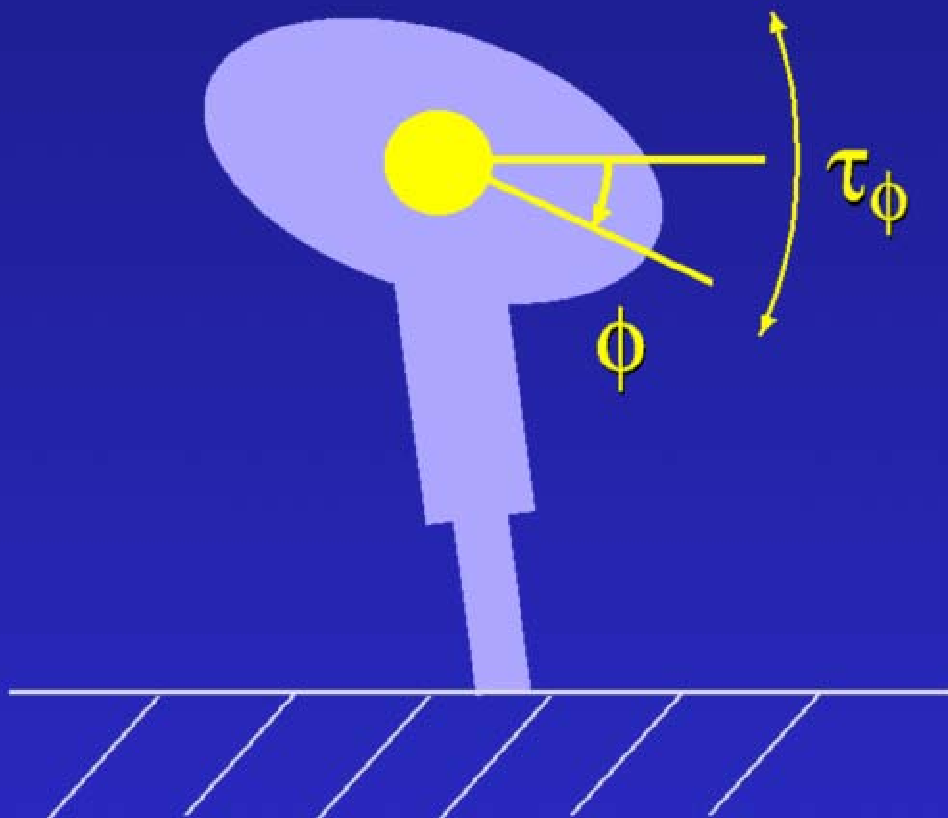


$$x_{fh} = \frac{1}{2} t_s \dot{x} - k_{\dot{x}}(\dot{x}_d - \dot{x})$$

$$\theta = f(x_{fh})$$

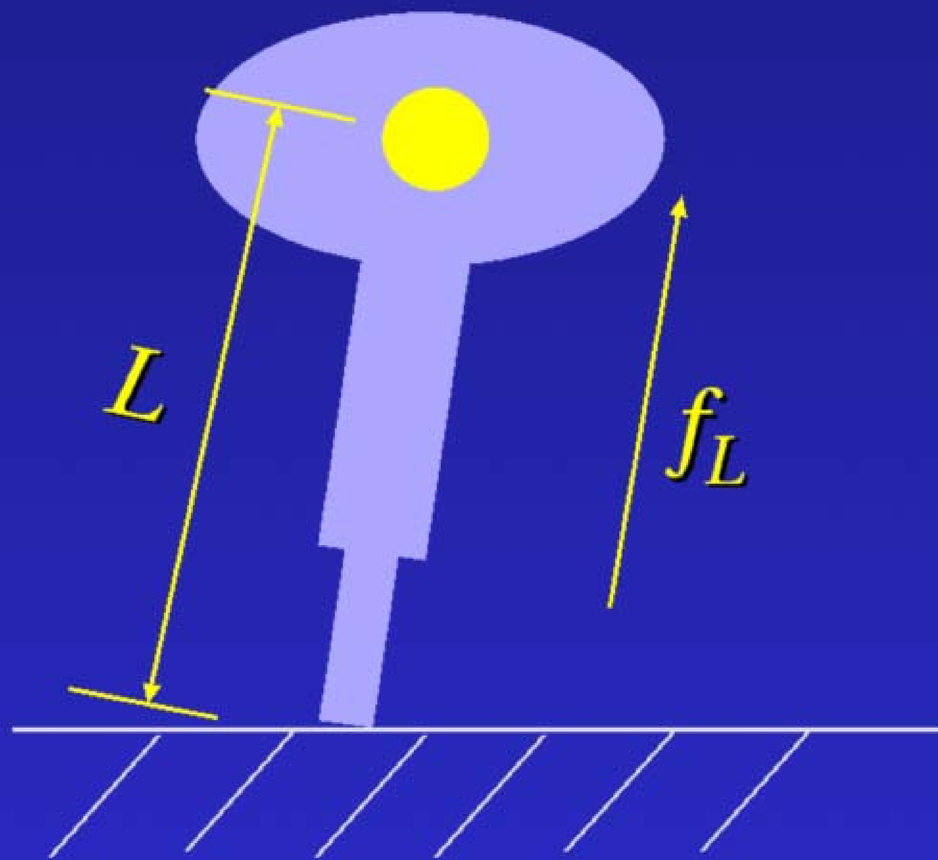
inverted pendulum model
leg positioned wrt to world coordinates, not body

Body attitude



$$\tau_\phi = k_\phi(\phi - \phi_d) + b_\phi(\dot{\phi} - \dot{\phi}_d)$$

Hopping height



$$f_L = k_L(L - L_d) + b_L(\dot{L} - \dot{L}_d)$$

Generalizing from 2D -> 3D

velocity:

$$x_{fh} = \frac{1}{2} t_s \dot{x} - k_{\dot{x}} (\dot{x}_d - \dot{x})$$

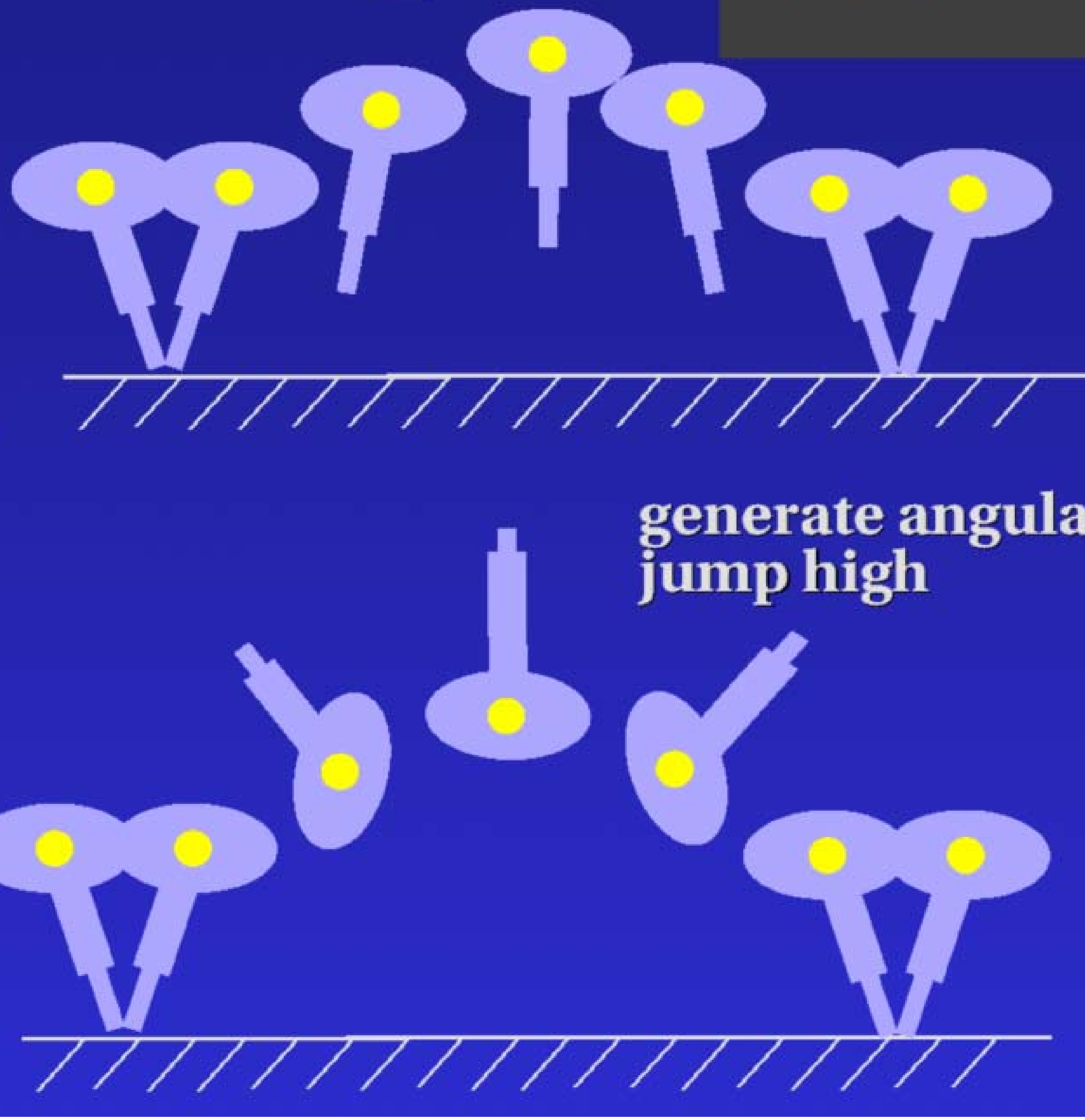
$$y_{fh} = \frac{1}{2} t_s \dot{y} - k_{\dot{y}} (\dot{y}_d - \dot{y})$$

pitch, roll, yaw:

$$\tau_\phi = k_\phi (\phi - \phi_d) + b_\phi (\dot{\phi} - \dot{\phi}_d)$$

Gymnastic Flips

$$2n\pi = \dot{\phi} T_f$$

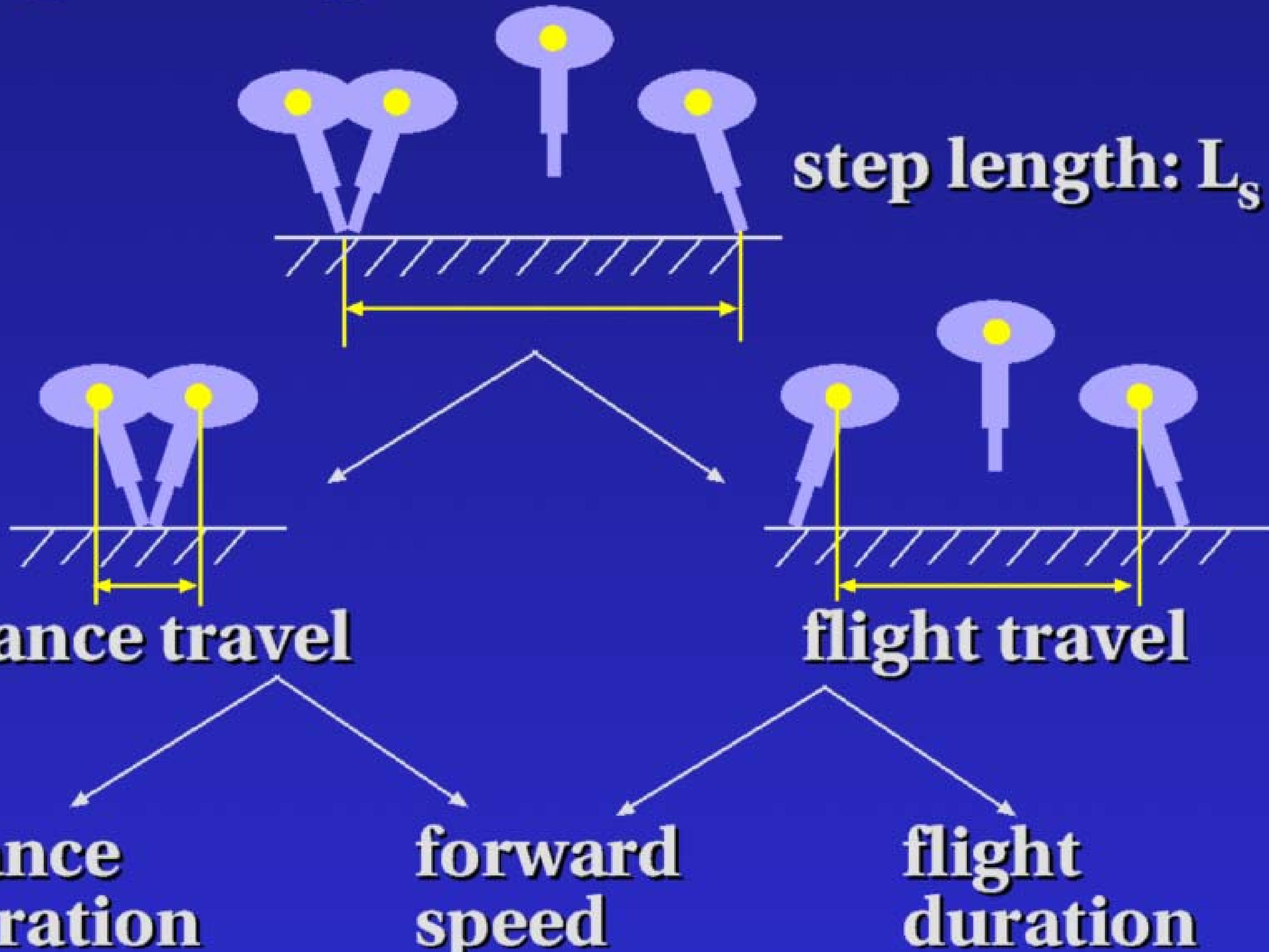


running

generate angular velocity
jump high

flipping

Step Length Control



Robotics

CMU and MIT 1987, with Marc Raibert

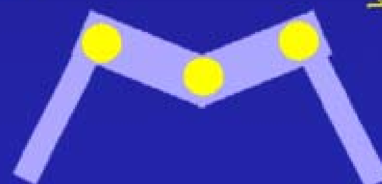
Useful for video games?

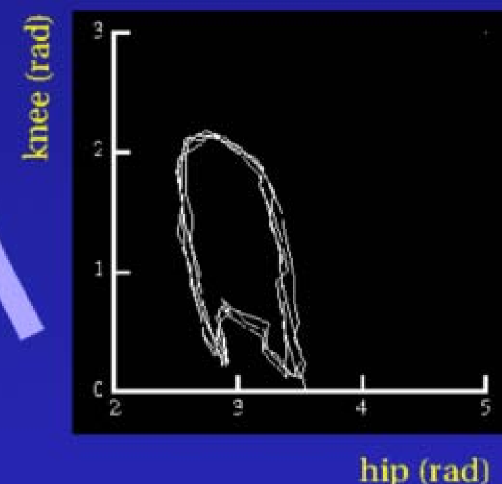
- Working on a physical robot is impressive – but is that good enough for a video game?
- Needs to work every time for every input… Or have graceful failure modes.
- And how are we going to do more interesting things than just hopping???

Where do control laws come from?

observation

biomechanical literature



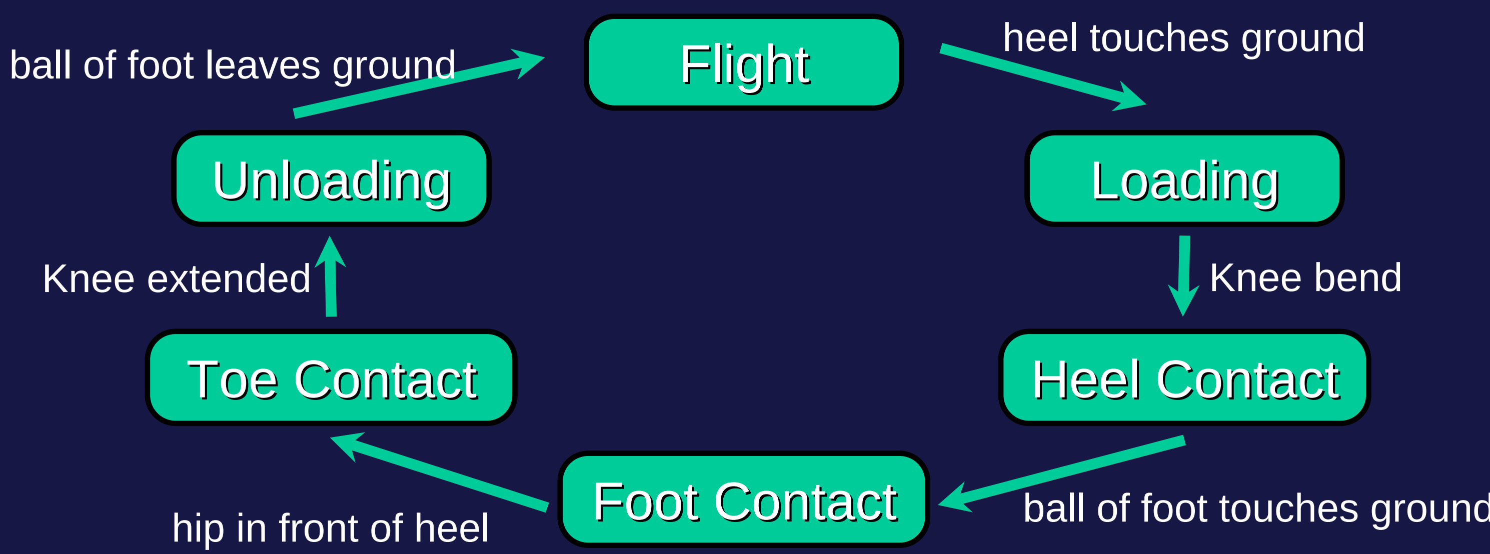


optimization

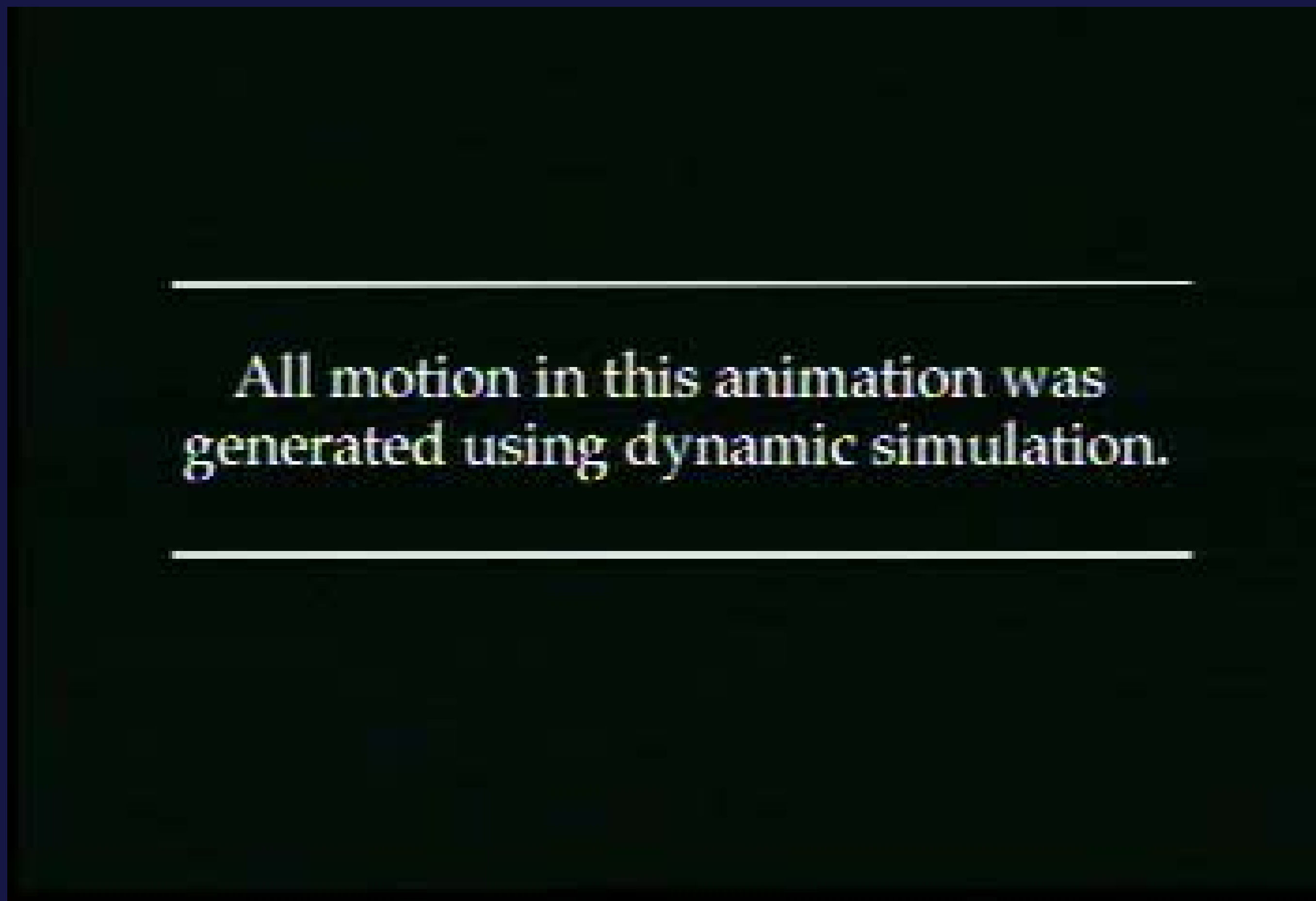
physical intuition

Control Systems for Humans

Running [Hodgins '95]



Simulating Behaviors



SIGGRAPH 1995

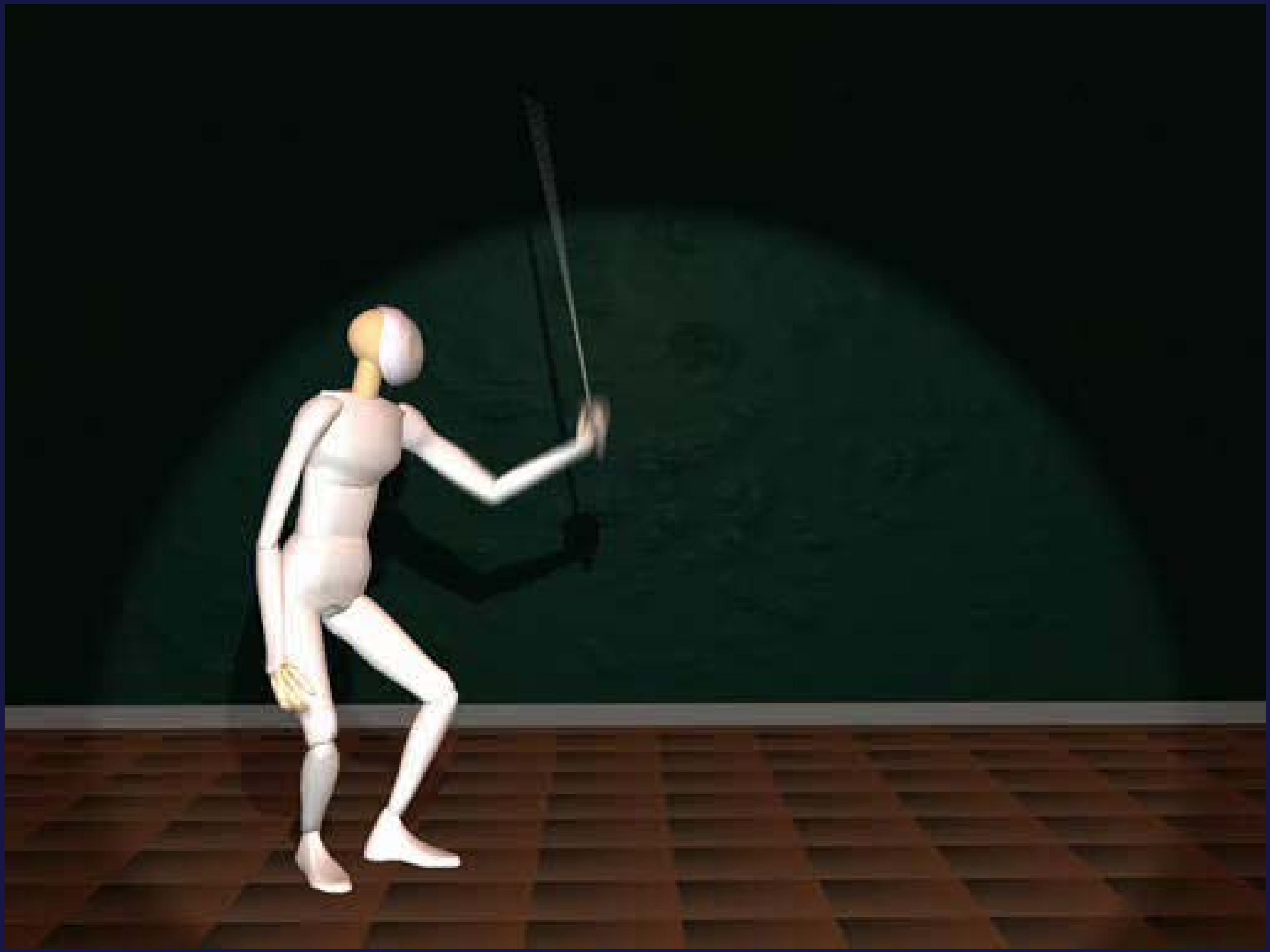
Combining Simulation and Mocap

Mocap for trajectory tracking

Mocap for control system design

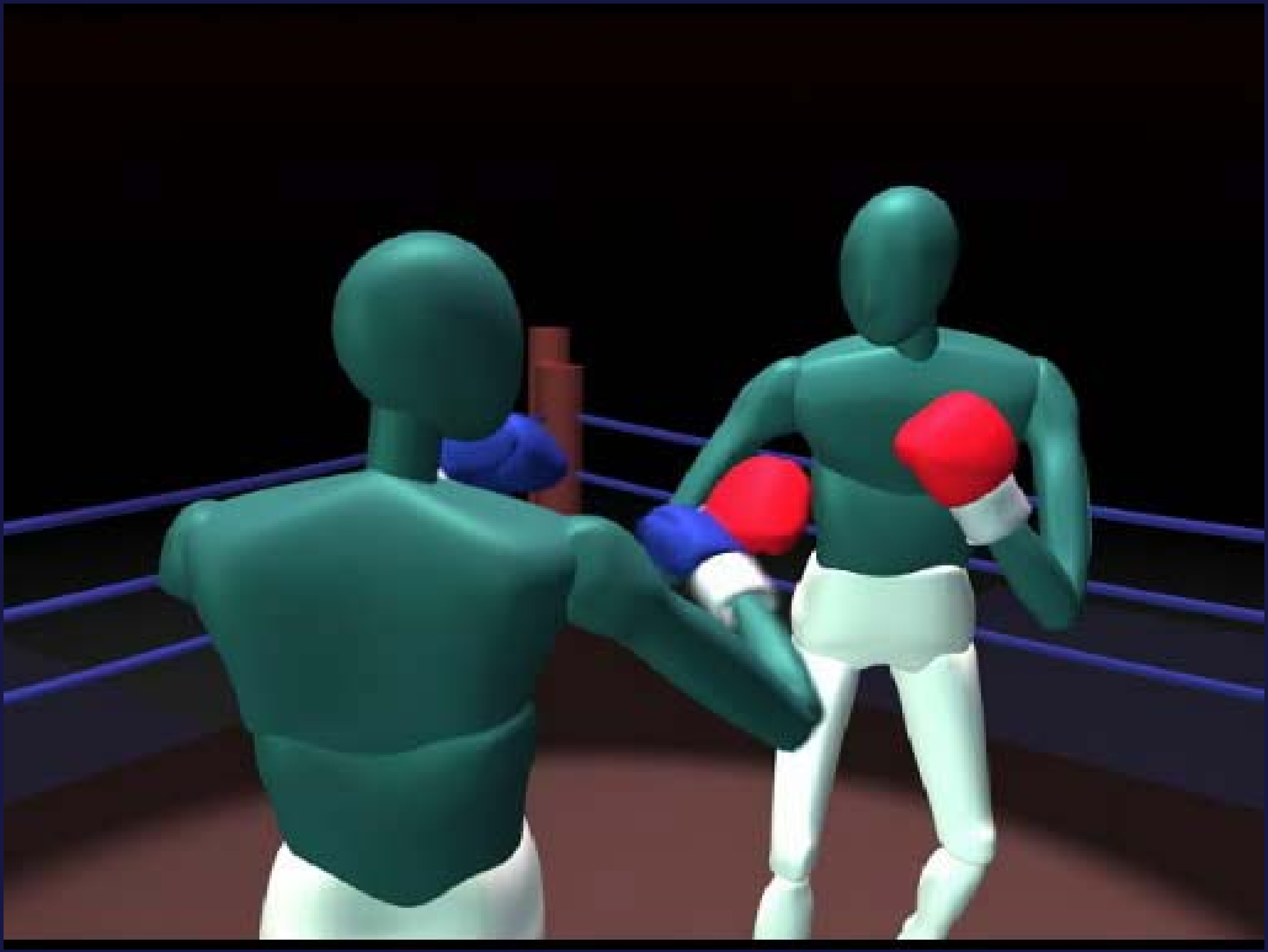
Mocap -> sim -> mocap

Combining Approaches



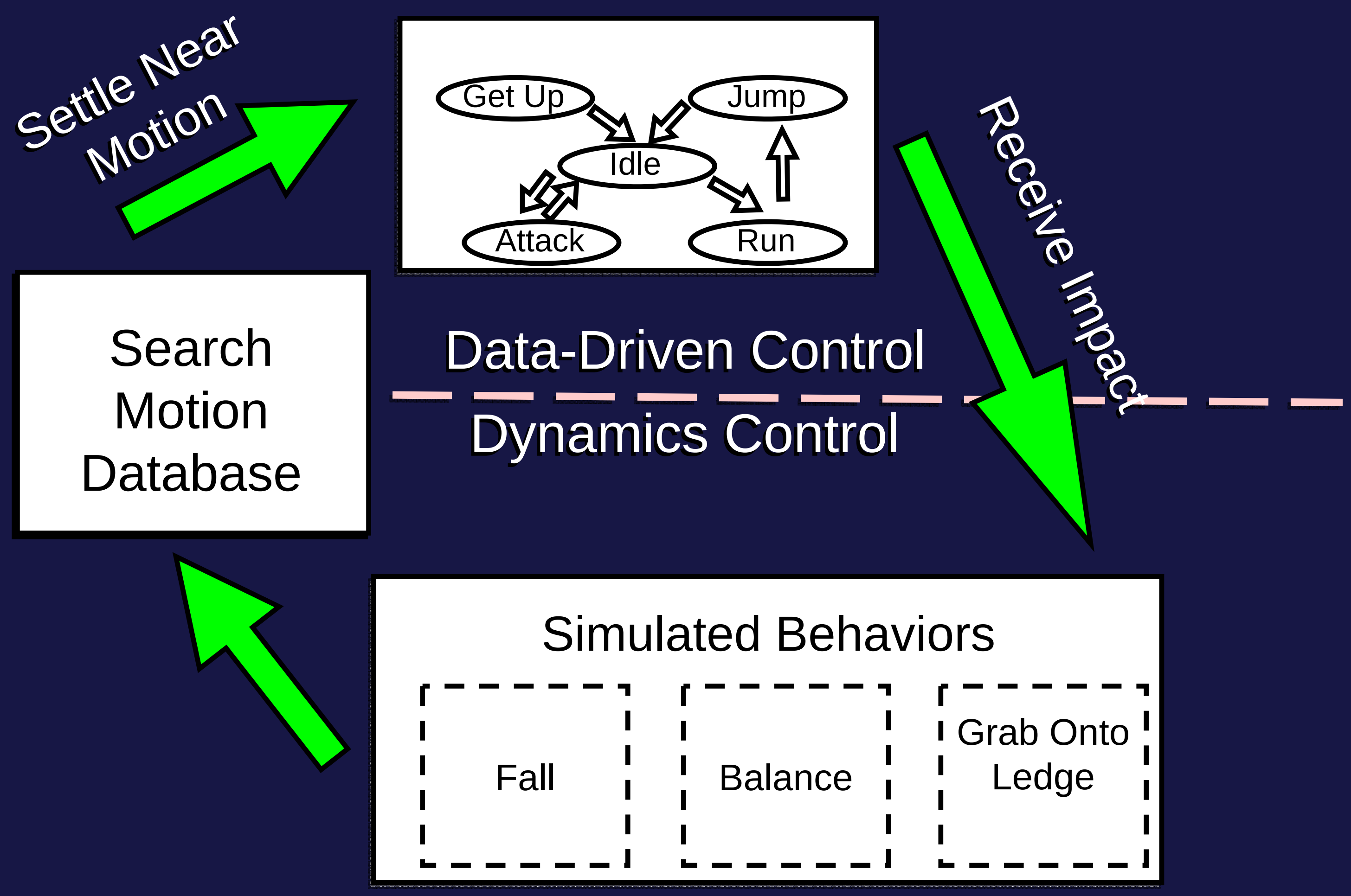
Average between balance controller and data
Victor Zordan, PhD thesis

Boxing (with opponent)



Boxing (comparison)

Mocap -> Sim -> Mocap



Executing Transitions

State space of data-driven technique:

- Any pose in the motion database

State space of dynamics-based technique:

- Set of poses allowable by joint limit constraints
- MUCH larger because it:
  - can produce motion difficult to animate or capture
  - includes unnatural poses

Clearly, some correspondence must be made to allow smooth transitions between the two

Transitions between techniques

Motion Data → Simulation

Easy. Just initialize simulation with pose and velocities extracted from motion data.

Simulation → Motion Data

Much harder. How to get near stored data?

Problem: Find nearest matches in the motion database to the current simulated motion.

1. Data Reduction/Representation

Search only some of the keyframes

Data Representation: Joint positions

2. Process into Spatial Data Structure

kd-tree works well

3. Search Structure at Runtime

Query pose comes from simulation

Nearest neighbor search problem

Choose motion most relevant to in-game situation

What's missing?

Data-Driven	Dynamics	Data-Driven	
Walk →	Simulated Fall	→ Get Up →	Idle

What's missing?

1. The fall lacks life
2. Transition has blending artifacts

Data-Driven | Dynamics | Data-Driven

Walk → Simulated Fall → Get Up → Idle

Fixing the Transition

At the time of the transition the simulation is NOT likely to be in a posture in the motion database

(It IS likely, however, to be interacting closely with the environment)

How can we get the simulation to settle near the best matching motion data?

Can we maintain physical constraints between the body and the environment?

Fixing the Transition

Solution: Settle Controller

Actuate joints using a special PD controller to settle the simulation near selected motion data

- Pose controller uses search result as target joint angles
- A physically grounded alternative to blending
  - Avoids object interpenetrations and foot sliding...
- Complex situations might be handled by more specialized controllers
- Can always finish it off with blending if necessary...

Adding Life to the falling motion

One Possibility: A Simple Pose Controller

- Look at initial conditions of an impact and choose initial desired reaction from a database of example poses
- May update desired pose as simulation evolves - still totally data-driven (and artist directed)

This can work well, but might not be as dynamic as we'd like.

Adding Life to the falling motion

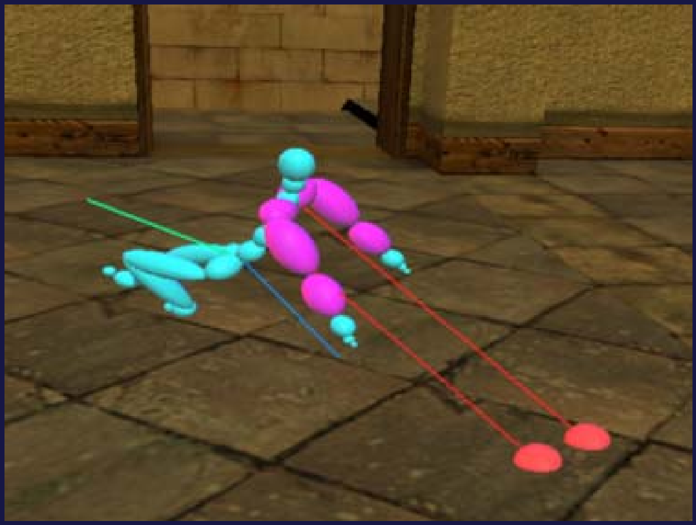
Reasonably approximate what humans do during a *full* loss of balance

highly effective motor control strategies → hard to model

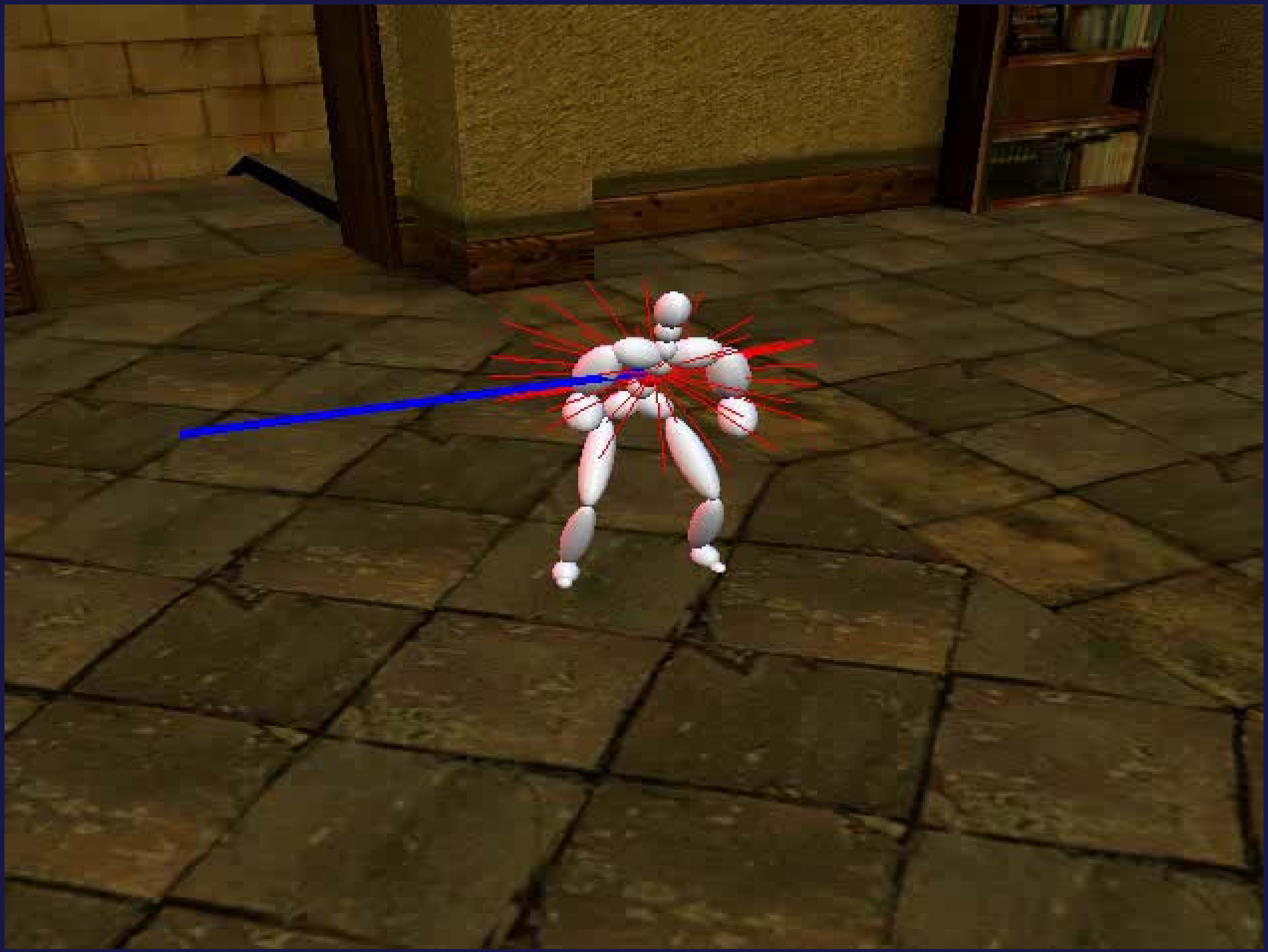
Possible Approach:

Track predicted shoulder landing locations with arms

Direction the body falls determines which arms track



Results



Data-Driven | Dynamics | Data-Driven

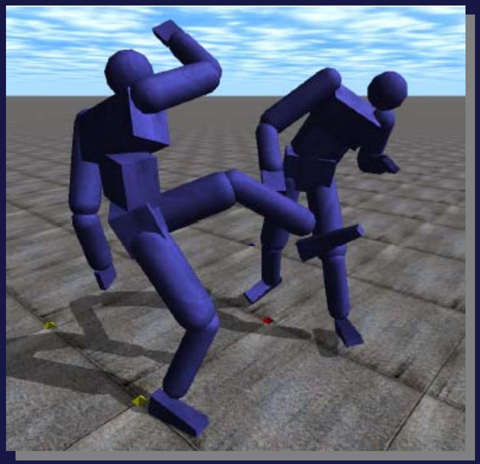
Idle → Simulated Fall → Get Up → Idle

Results: fall and roll

Data-Driven	Dynamics	Data-Driven
Idle →	Simulated Fall and Roll	→ Get Up

Physically Based Transitions Following Impacts, With Motion Capture

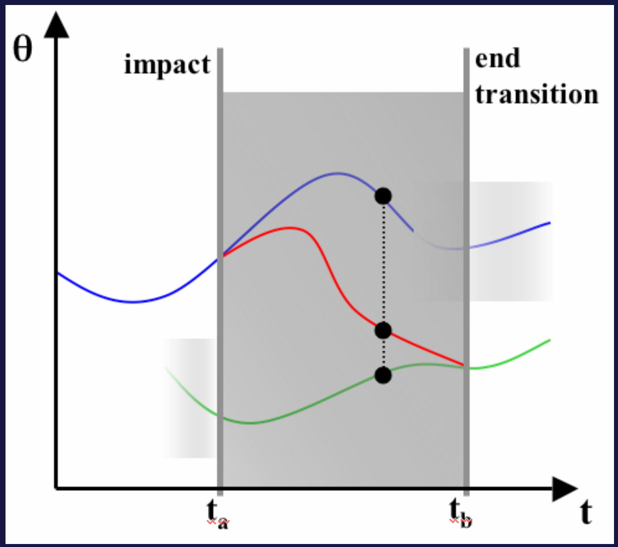
[Zordan et al. '04]



Apply impact forces to sim

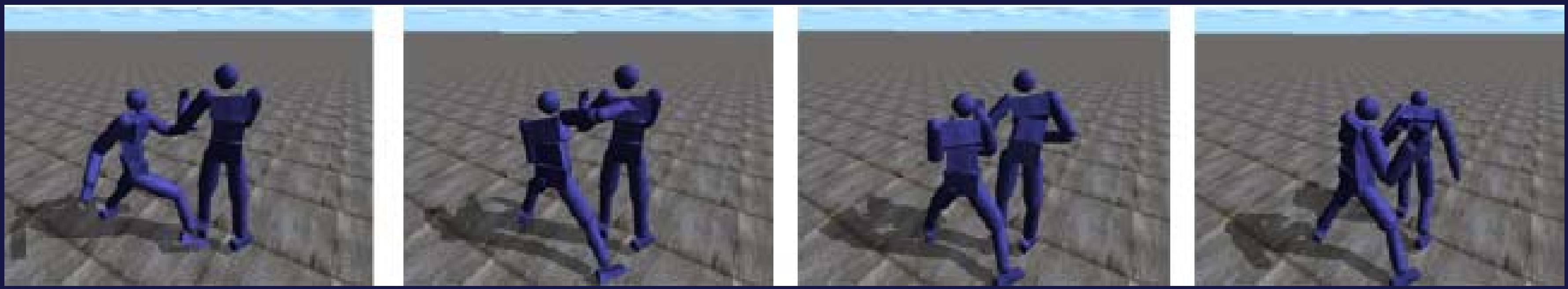
Search to find clip for after interaction

Actively track the motion clip as it transitions, to get the posture in place with joint torques



Add global positions using forces to position character

Physically Based Transitions



Internal torques mimic human reaction

External forces minimize error while not breaking the physical engine

This method uses mocap while the interaction forces are still active

Doesn’t guarantee a perfect match at the end, but hopefully we can cover this up with blending!

Dynamic Response for Motion Capture Animation

Making it Practical…

Games need to guarantee robustness

- Games can sacrifice physical realism for robustness/speed—know when using simulation is appropriate!
- Start simple—pose controllers with artist predefined reactions
- Specify only the DOFs necessary
  - Let the natural dynamics of the system guide the behavior
- Fake things (like balance control)
  - Make the ground "stickier"
  - External balancing forces to keep the body upright
  - Consider simulating only some of the body

User Control

- High-level control of characters
  - Velocity -> joystick—treat the character as a cylinder and assume that there is code to make it follow instructions (run, walk, turn, climbing stairs)
  - Button pushes -> discrete actions (kick, punch)
- A few exceptions
  - Olympic Decathlon on the Apple 2
  - Motionplayground (demo coming)
- And some failures
  - Trespasser

Novel user interaction

- http://www.cs.ubc.ca/~van/sssjava/javademo.html

User Control

Are game controls the ultimate 3d interface?

Maybe for gamers…

Stan Melax: http://www.gamasutra.com/features/20010324/melax_pfv.htm

What is the future?

Why don’t we already have fully simulated characters?

Will we ever?

What about a world that is “totally” live?