Planning, dynamics and motion graphs

D.A. Forsyth, UIUC

Kinematic planning

- Construct path in configuration space $x(t)$

$$0 \leq t \leq 1$$

  - so that there are no collisions

$$g(x) > 0$$

  - starts at start and ends at goal

$$x(0) = a \qquad x(1) = b$$

Q: what if we wanted the shortest (etc)?

- We produced $\hat{x}(t)$
  - path that meets constraints
- Use this to initialize

$$\min_{x(t)} \int_0^1 h(x, \dot{x}, \ldots, t) dt$$

$$g(x) > 0$$

$$x(1) = b$$
$$x(0) = a$$

Polish by…

- Discretize approximate path
- Solve

$$x_i = x(i\Delta t)$$

$$\min_{\delta_i} \sum_i h(x_i + \delta_i, \ldots)\Delta t$$

$$g(x_i + \delta_i) > 0$$

You might also need to interpolate path segments here

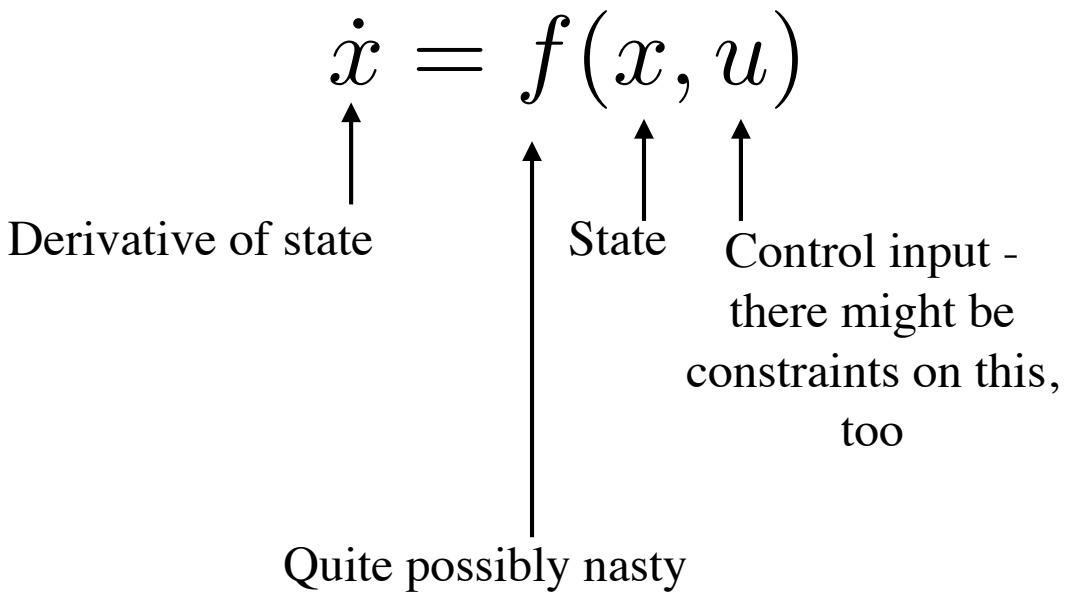
$$x(1) = b$$
$$x(0) = a$$

Q: Why not use…

- Simple splines
  - A smoothing spline
  - A B-spline
  - A subdivision curve
  - etc
- with vertices as control points?

Dynamics make planning harder

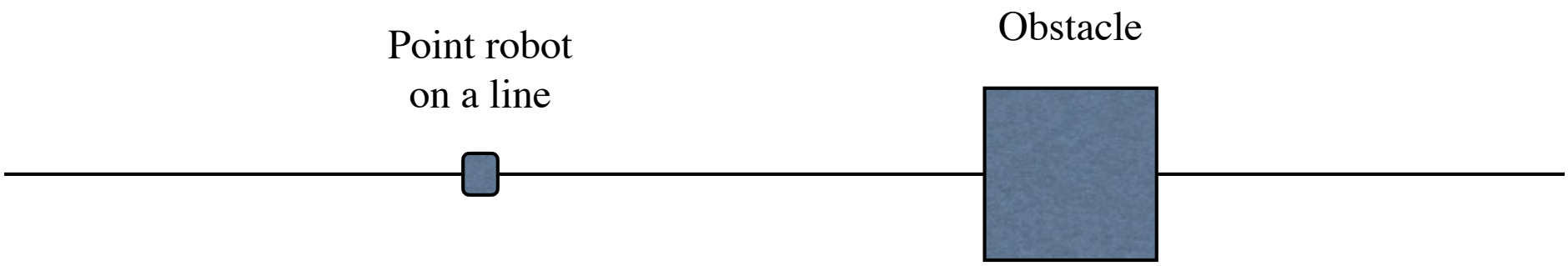
- Dynamics introduce differential constraints



Phase space

- Configuration space + all relevant derivatives

Phase space



- Configuration space: x
  - (with complications created by obstacle)
- State is:
  - (x, v)
  - so phase space is 2D
- Dynamics are:

$$\begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ u \end{pmatrix}$$

Simplest case - extend obstacles

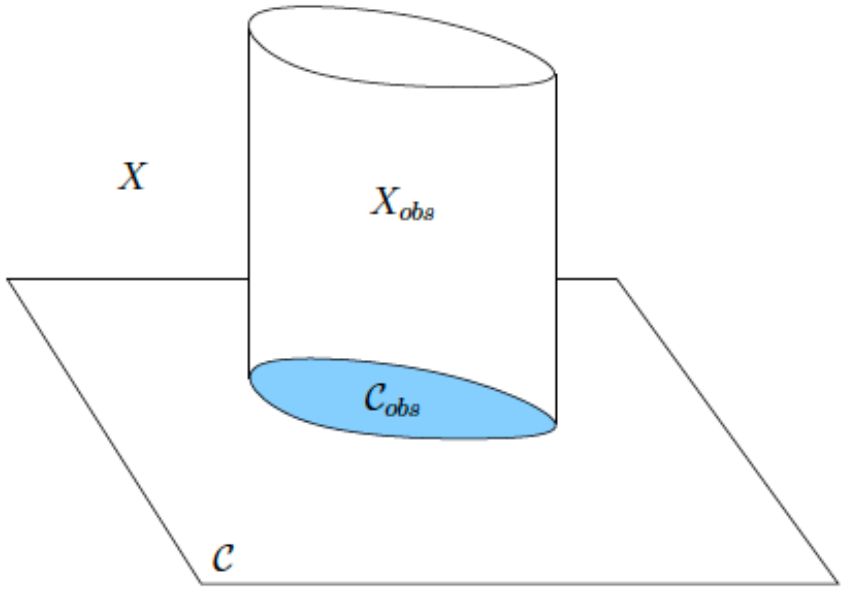


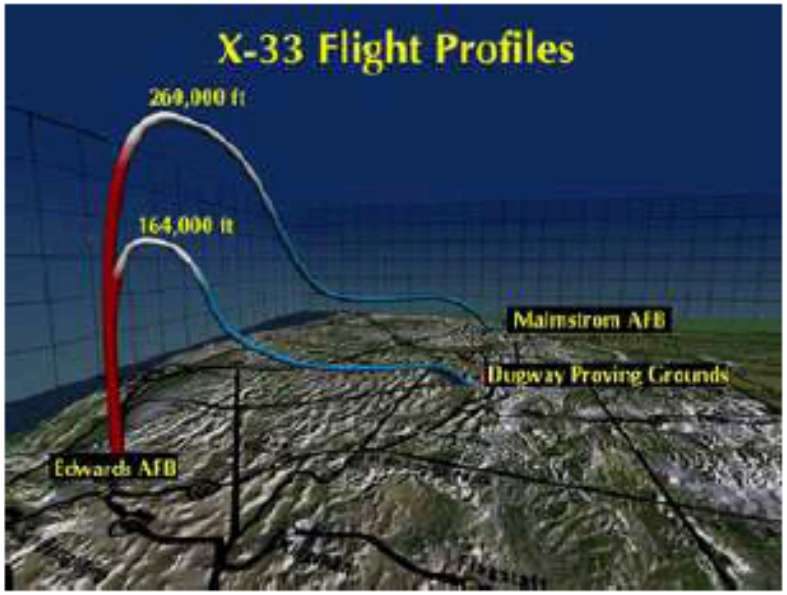
Figure 14.1: An obstacle region $\mathcal{C}_{obs} \subset \mathcal{C}$ generates a cylindrical obstacle region $X_{obs} \subset X$ with respect to the phase variables.

= derivatives

Lavalle, Ch 14

Constraints on phase variables, too

NASA/Lockheed Martin X-33

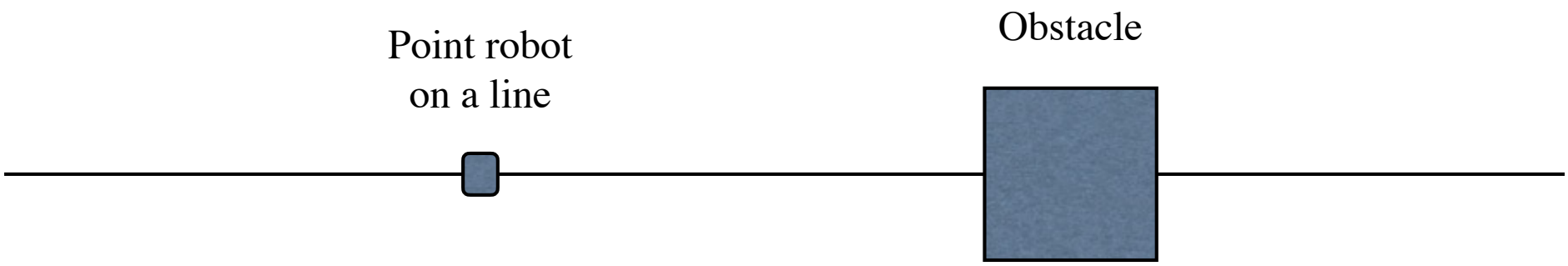


Re-entry trajectory

Figure 14.2: In the NASA/Lockheed Martin X-33 re-entry problem, there are complicated constraints on the phase variables, which avoid states that cause the craft to overheat or vibrate uncontrollably. (Courtesy of NASA)

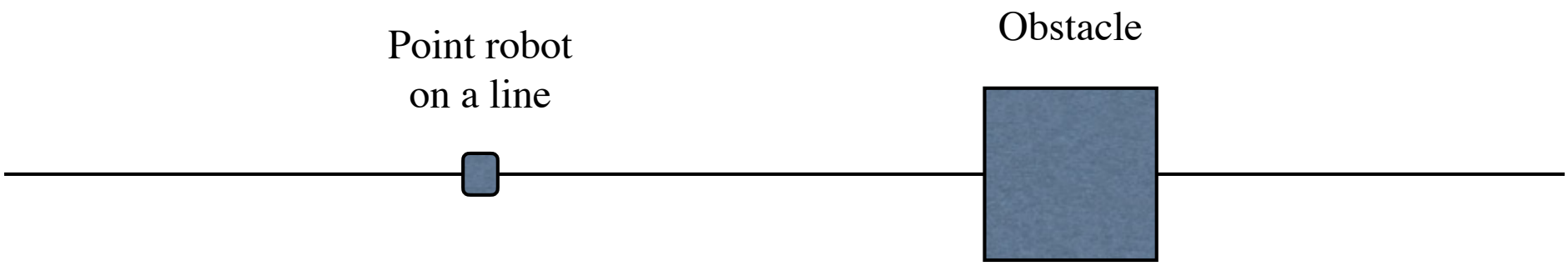
Lavalle, Ch 14

Phase space



- Configuration space: x
  - (with complications created by obstacle)
- State is:
  - (x, v)
  - so phase space is 2D
- Velocity limits are: -1<v<1
  - draw phase space with obstacles

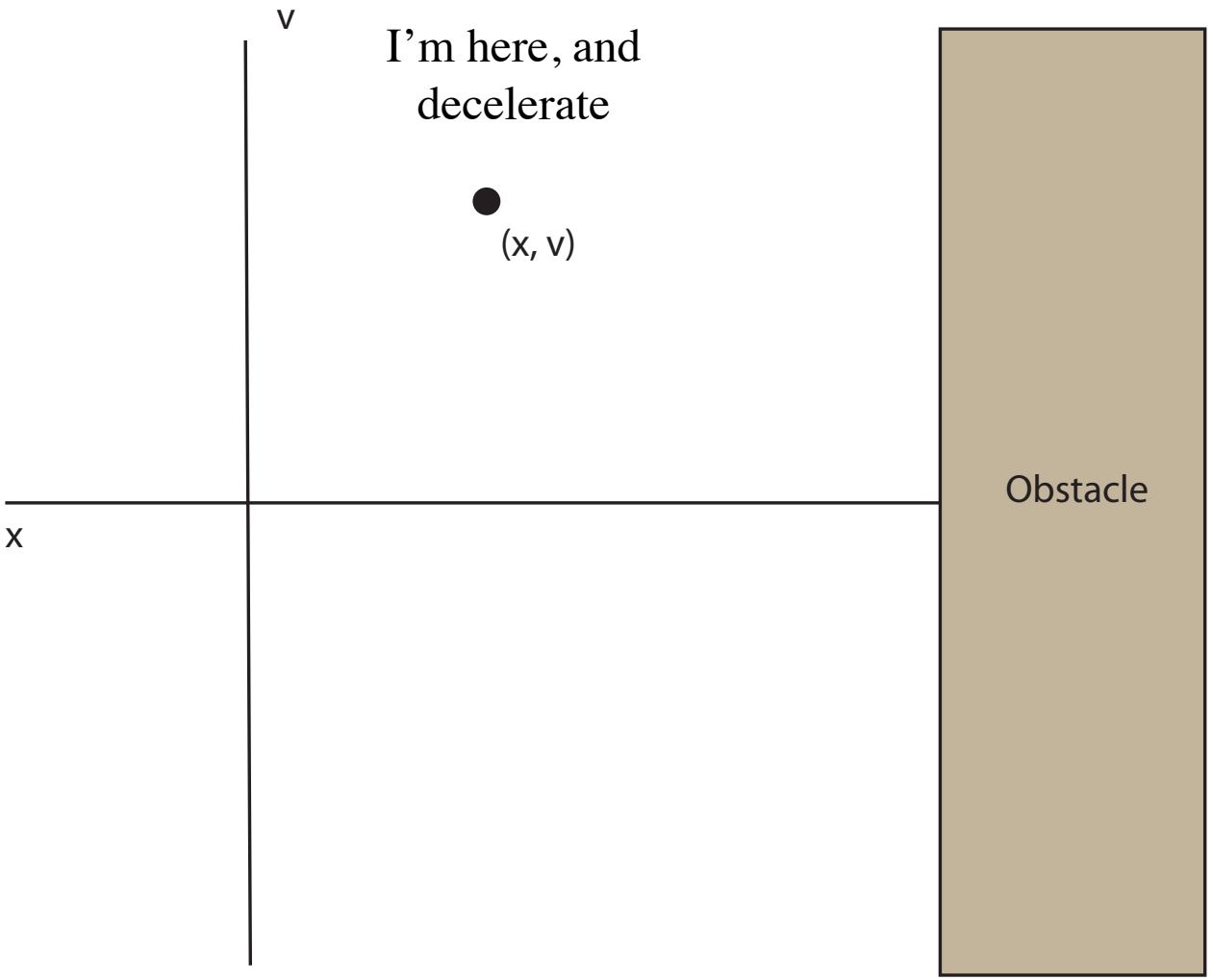
Control limits create obstacles



- State is: (x, v)
- differential constraints:

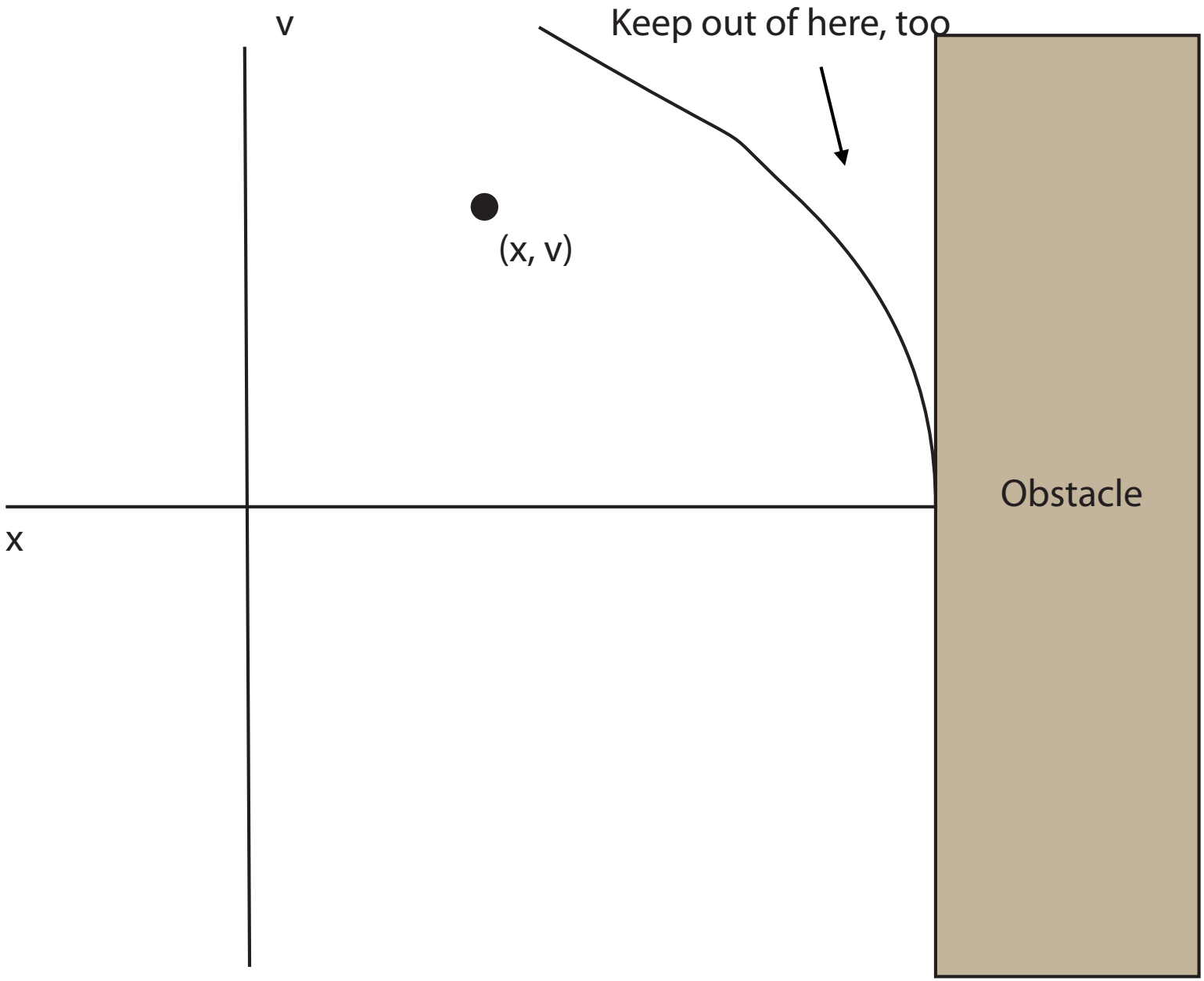
$$\begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ u \end{pmatrix}$$

Control constraint: $-1 \leq u \leq 1$



Distance travelled
until stationary
at acceleration = -1

$$\Delta x = \frac{v^2}{2}$$


v
Keep out of here, too
(x, v)
Obstacle
x

Key issue

- You have to think about control input as well
- Compare:
  - RRT in kinematic planning -
    - check is there no obstacle
  - RRT in dynamic planning
    - what feasible control gets you from qnear to qrand?
      - there might not be one
      - it might be hard to find

Taking actions into account

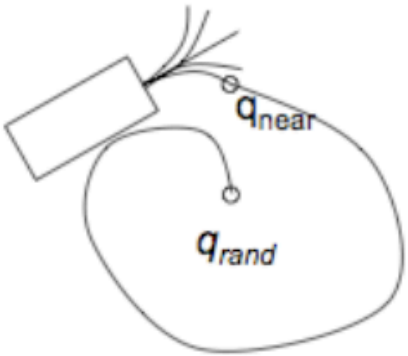
q_{near}

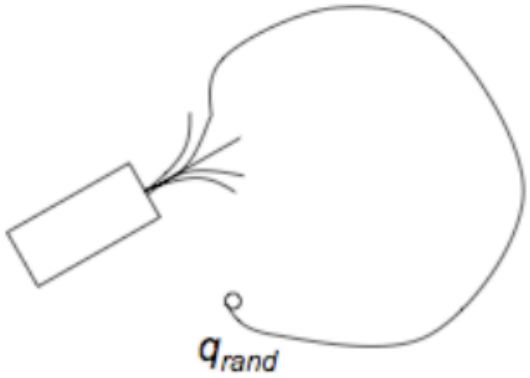
$q' = f(q,u)$ --- use action u from q to arrive at q'

Notice this f isn't the f in slide 6 —
it maps initial state to final state given control u

chose $u_* = \arg\min(d(q_{rand}, q'))$

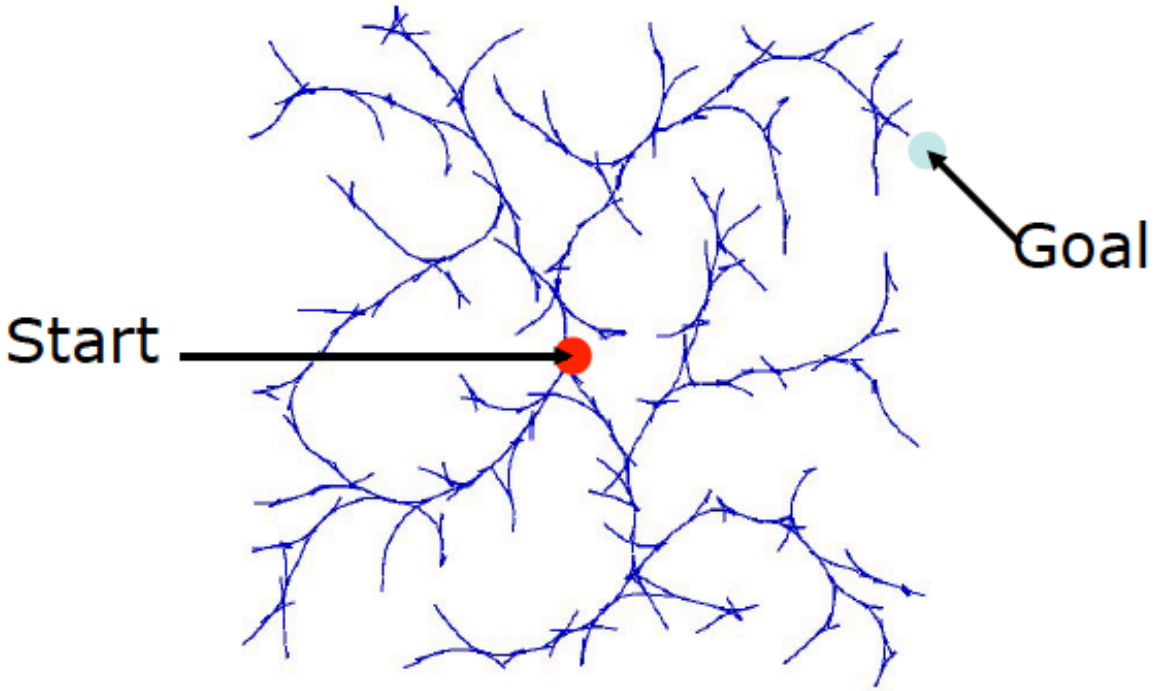
Is this the best?





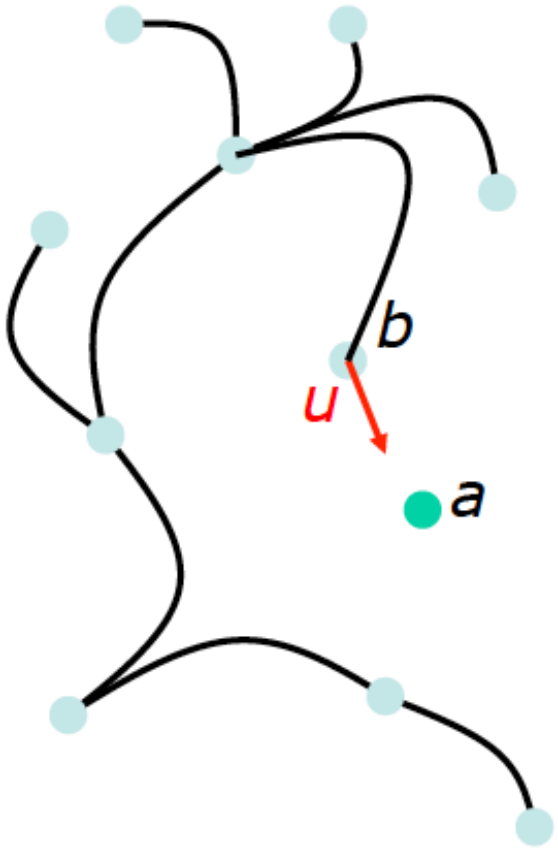
How it Works

- Build a rapidly-exploring random tree in state space (X), starting at s_{start}
- Stop when tree gets sufficiently close to s_{goal}



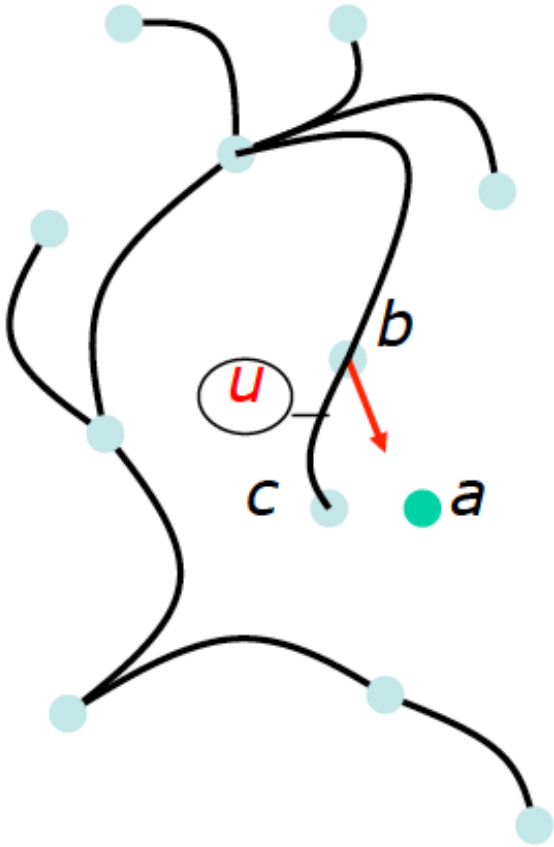
Building an RRT

- To extend an RRT:
  - Pick a random point *a* in *X*
  - Find *b*, the node of the tree closest to *a*
  - Find control inputs *u* to steer the robot from *b* to *a*



Building an RRT

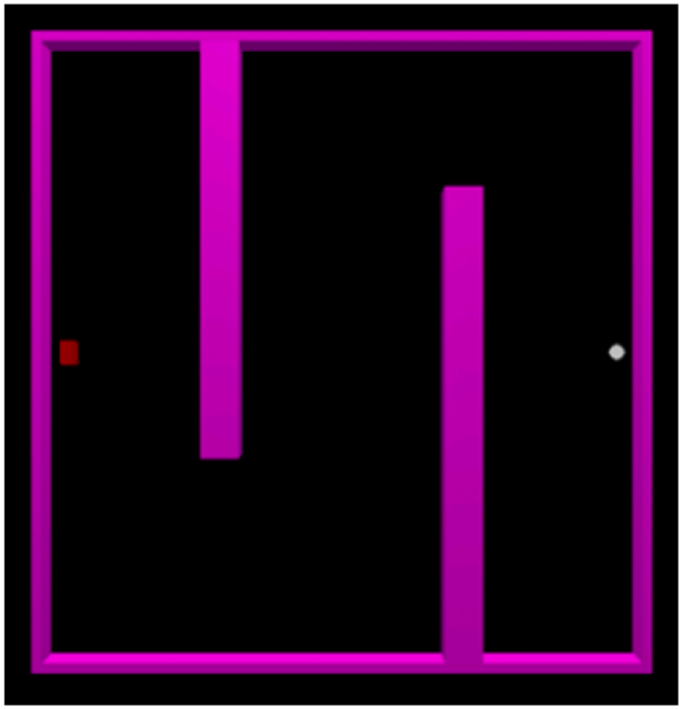
- To extend an RRT (cont.)
  - Apply control inputs *u* for time δ, so robot reaches *c*
  - If no collisions occur in getting from *a* to *c*, add *c* to RRT and record *u* with new edge

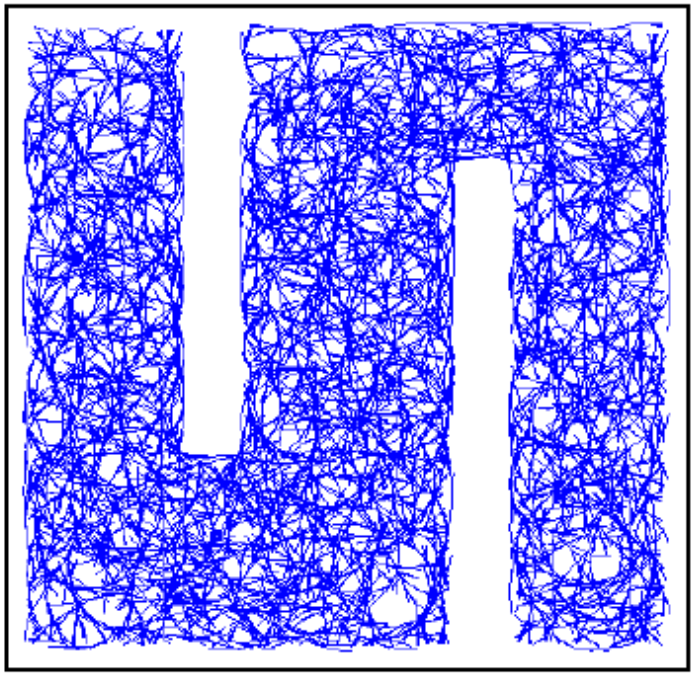


Executing the Path

- Once the RRT reaches s_{goal}
  - Backtrack along tree to identify edges that lead from s_{start} to s_{goal}
  - Drive robot using control inputs stored along edges in the tree

Problem of Simple RRT Planner

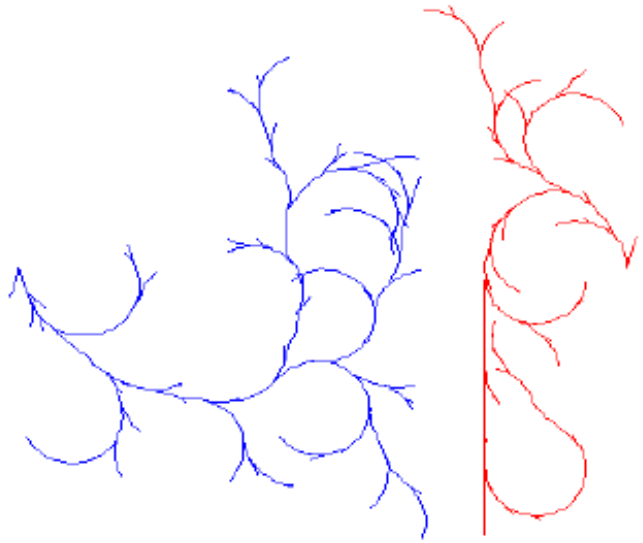




- Problem: ordinary RRT explores *X* uniformly
  → slow convergence
- Solution: bias distribution towards the goal – once in a while choose goal as new random configuration (5-10%)
- If goal is choose 100% time then it is randomized potential planner

Bidirectional Planners

- Build two RRTs, from start and goal state



- Complication: need to connect two RRTs
  - local planner will not work (dynamic constraints)
  - **bias** the distribution, so that the trees meet

RRT's

- Link
- http://msl.cs.uiuc.edu/rrt/gallery.html

- Issues/problems
- Metric sensitivity
- Nearest neighbour efficiency
- Optimal sampling strategy
- Balance between greedy search and exploration

- Applications in mobile robotics, manipulation, humanoids, biology, drug design, areo-space, animation
- Extensions – real-time RRT's, anytime RRT's dynamic domains RRT'sm deterministic RRTs, hybrid RRT's

Building an RRT

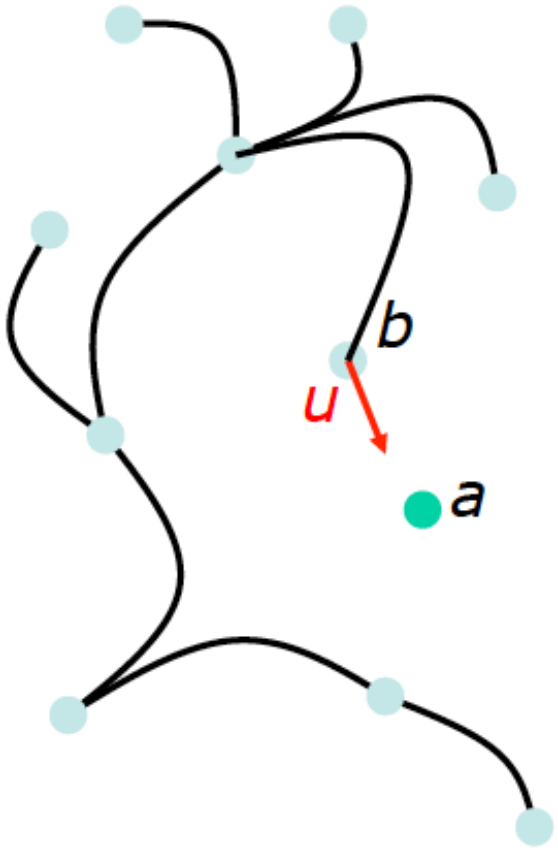
- To extend an RRT:
  - Pick a random point *a* in *X*
  - Find *b*, the node of the tree closest to *a*
  - Find control inputs *u* to steer the robot from *b* to *a*

HOW?

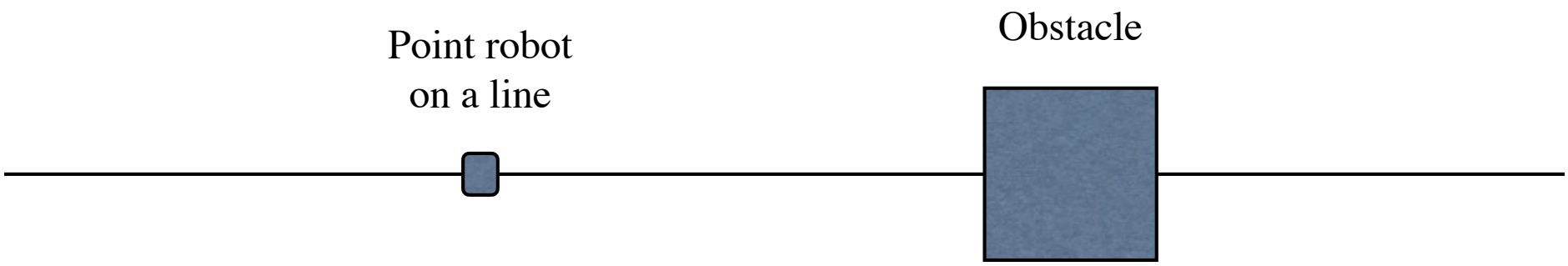
Strategies:

Optimization with simulator

Discretize



Boundary value problems



$$\begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ u \end{pmatrix}$$

- Our simple system
  - start at x=0, v=0
  - obstacle starts at x=1
  - plan to go to x=0.9, v=-2

Control constraint: $-1 \leq u \leq 1$

Boundary value problems - II

- Our simple system
  - start at x=0, v=0
  - obstacle starts at x=1
  - plan to go to x=0.9, v=-2
- Write this system

$$\frac{d^2x}{dt^2} = u$$

  - so general solution is:

$$c_0 + c_1 t + \frac{u}{2} t^2$$

$$\begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} c_0 + c_1 t + \frac{u}{2} t^2 \\ c_1 + ut \end{pmatrix}$$

What does this look like in (x, v) space?

Control constraint: $-1 \leq u \leq 1$

Boundary value problems - III

- Imagine we're at $\begin{pmatrix} a \\ b \end{pmatrix}$
- and want to go to $\begin{pmatrix} c \\ d \end{pmatrix}$
- What u do we use?
  - and can we do it?

$$\begin{pmatrix} x \\ v \end{pmatrix} = \begin{pmatrix} c_0 + c_1 t + \frac{u}{2} t^2 \\ c_1 + ut \end{pmatrix}$$

- Notice we have four constraints, four unknowns
  - didn't specify arrival time!

Boundary value problems - IV

- Start at t=0 $\begin{pmatrix} x(0) \\ v(0) \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$
- arrive at time t=s $\begin{pmatrix} x(s) \\ v(s) \end{pmatrix} = \begin{pmatrix} a + bs + \frac{u}{2}s^2 \\ b + us \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix}$

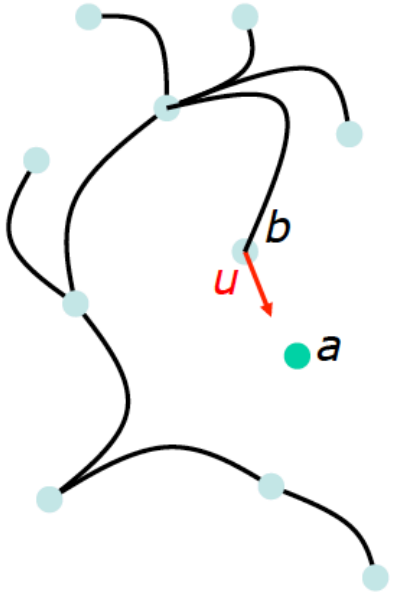
$$s = \frac{d-b}{u} \qquad (a-c)u = -b(d-b) - \frac{1}{2}(d-b)^2$$

$$w = \frac{-b(d-b) - \frac{1}{2}(d-b)^2}{(a-c)}$$

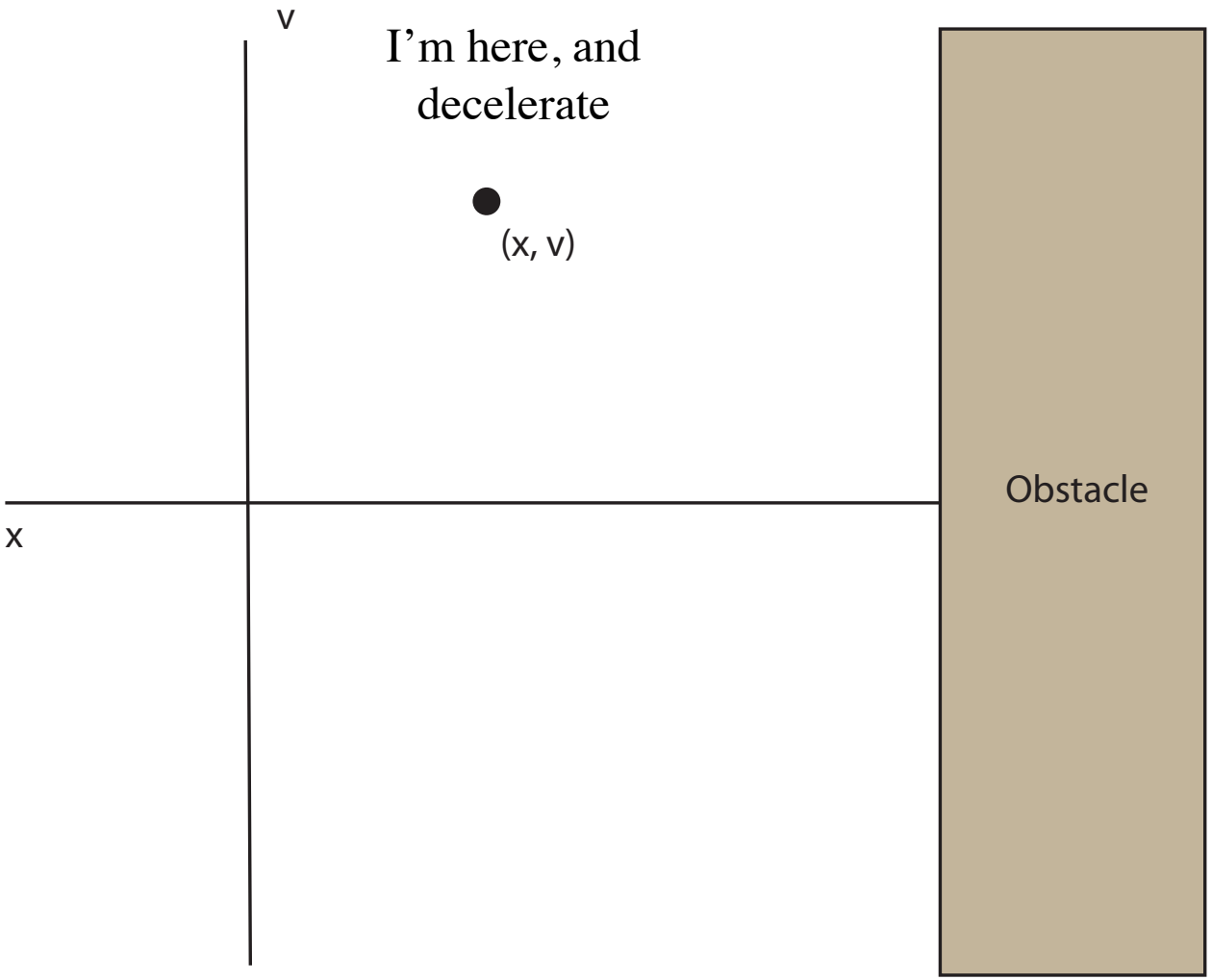
$$u = \begin{cases} 1 & \text{if } w > 1 \\ w & \text{if } -1 \leq w \leq 1 \\ -1 & \text{otherwise} \end{cases}$$

Building an RRT

- To extend an RRT:
  - Pick a random point *a* in *X*
  - Find *b*, the node of the tree closest to *a*
  - Find control inputs *u* to steer the robot from *b* to *a*



- Can now compute u, s
  - that takes us from closest point to sample
    - or nearby (recall constraints on u)
  - does that path intersect physical obstacle?
    - yes - either drop, or reduce s
    - no - add sample, recording u, s as well
- We now can build a tree!



Distance travelled
until stationary
at acceleration = -1

$$\Delta x = \frac{v^2}{2}$$

Boundary value problems - V

- This should strike you as being hard to generalize
  - most odes don't have easy closed form solutions
  - why use constant acceleration?
    - I got an easy answer
- Strategies
  - use a simulator
  - search a discrete set of control inputs
- Particularly important idea
  - Do many path segments in advance; cache the results (motion primitives)
  - Search that set for something that gets "close" to the endpoint
- Q:
  - what primitives? search how?

Discretization and motion graphs

- Discretization
  - build a set of motion primitives
    - start state, control input -> path, end state
  - procedures for composing them
    - what primitives can be applied in what state?
      - there is translation, rotation covariance
- Search this set of primitives for appropriate control inputs

The motion graph

- Old idea in human animation
  - Essentially, build a roadmap of what people can do by
    - joining up animation sequences
  - Control by
    - searching these sequences
- Analogy with car
  - drive around “at random”
  - build motion graph
  - search this for primitive sequences

The motion signal

- There is no reliable method for generating novel motions
  - some special cases work OK
  - Keys for special cases
    - data driven methods work well for temporal composition
    - Some motions can be blended successfully
    - Contacts create special problems
    - There are complex, cross-body correlations
- There must be some set of motion primitives

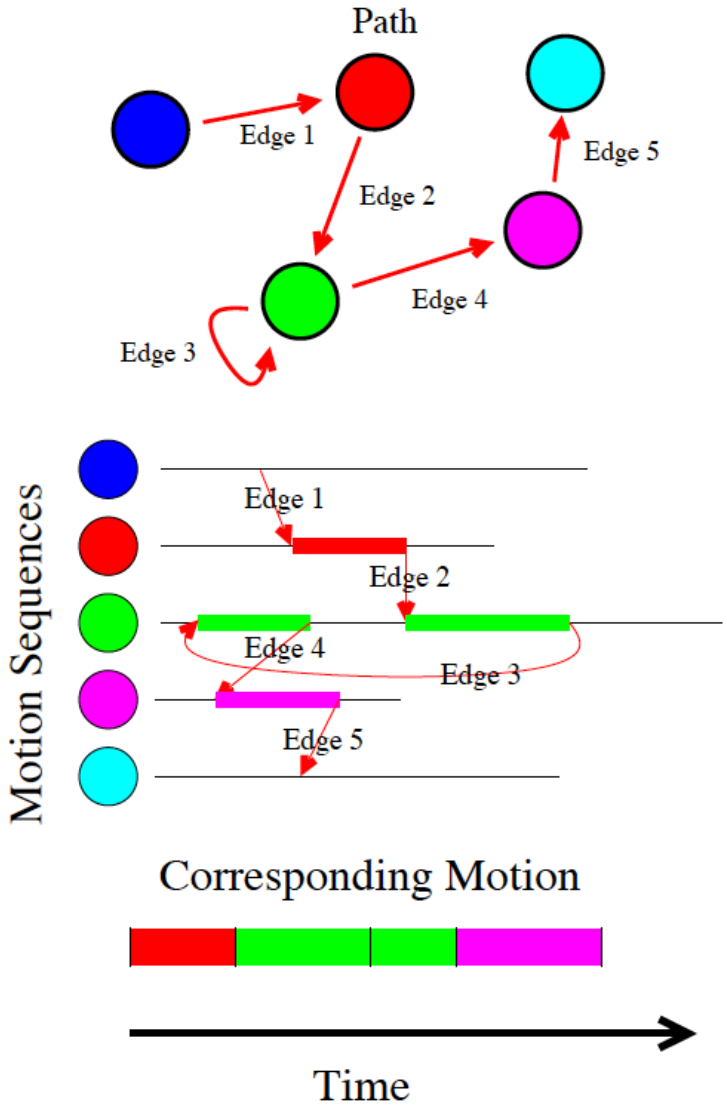


Figure 1: We wish to synthesize human motions by splicing together pieces of existing motion capture data. This can be done by representing the collection of motion sequences by a directed graph (**top**). Each sequence becomes a node; there is an edge between nodes for every frame in one sequence that can be spliced to a frame in another sequence or itself. A valid path in this graph represents a collection of splices between sequences, as the **middle** shows. We now synthesize constrained motion sequences by searching appropriate paths in this graph using a randomized search method.

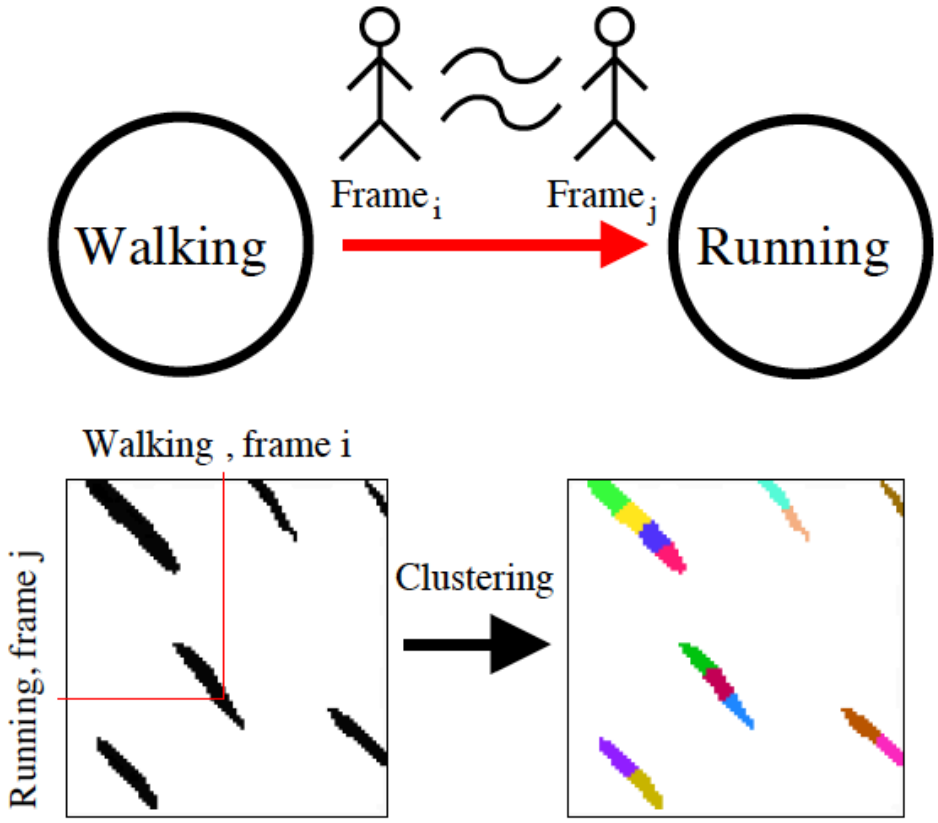


Figure 2: Every edge between two nodes representing different motion clips can be represented as a matrix where the entries correspond to edges. Typically, if there is one edge between two nodes in our graph, there will be several, because if it is legal to cut from one frame in the first sequence to another in the second, it will usually also be legal to cut between neighbors of these frames. This means that, for each pair of nodes in the graph, there is a matrix representing the weights of edges between the nodes. The i, j'th entry in this matrix represents the weight for a cut from the i'th frame in the first sequence to the j'th frame in the second sequence. The weight matrix for the whole graph is composed as a collection of blocks of this form. Summarizing the graph involves compressing these blocks using clustering.

1. Start with a set of n valid random "seed" paths in the graph G'
2. Score each path and score all possible mutations
3. Where possible mutations are:
    (a) Delete some portion of the path and replace it with 0 or 1 hops.
    (b) Delete some edges of the path and replace them with their children
4. Accept the mutations that are better than the original paths
5. Include a few new valid random "seed" paths
6. Repeat until no better path can be generated through mutations

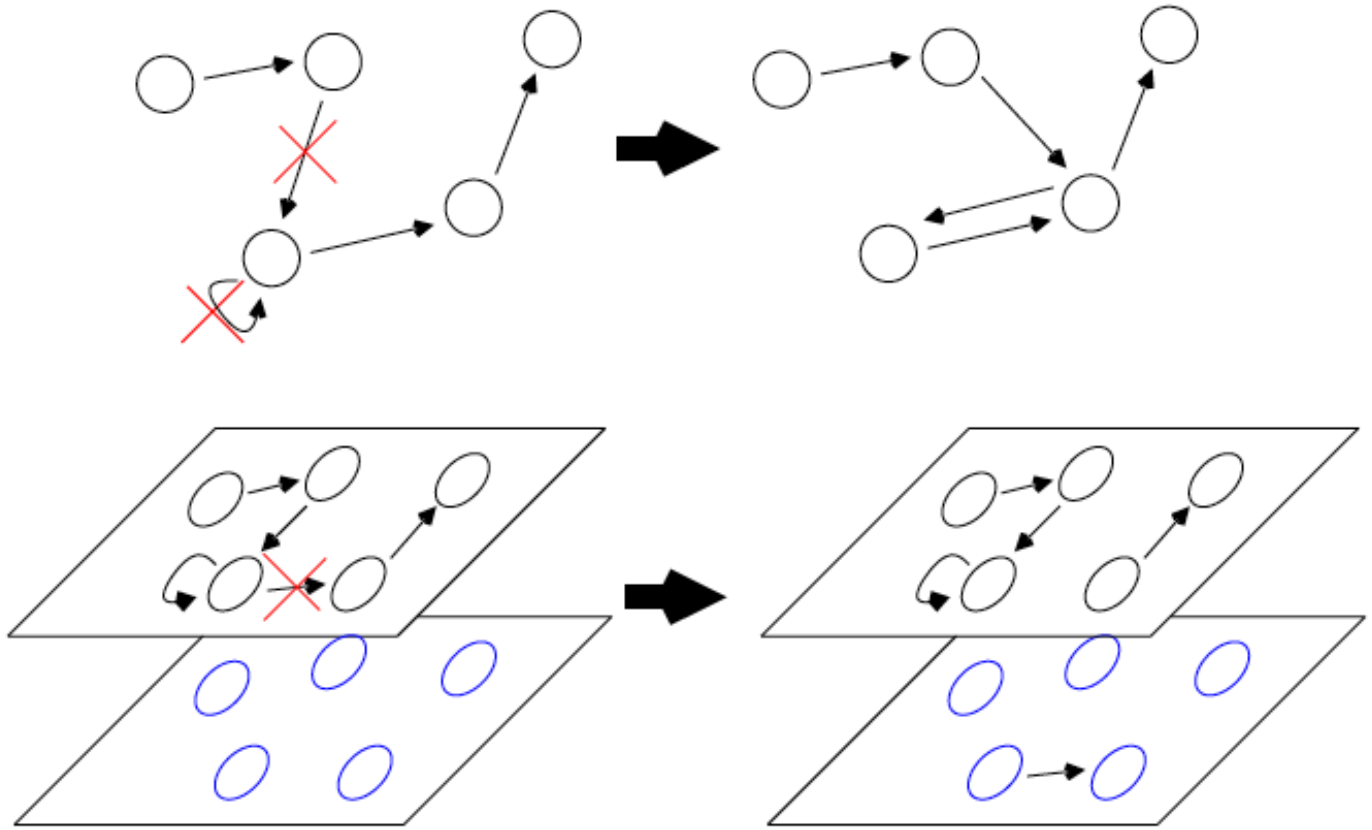


Figure 3: The two mutations are: deleting some portion of the path (top-left, crossed out in red) and replacing that part with another set of edges (top-right), and deleting some edges in the path (bottom-left) and replacing deleted edges with their children in our hierarchy (bottom-right)

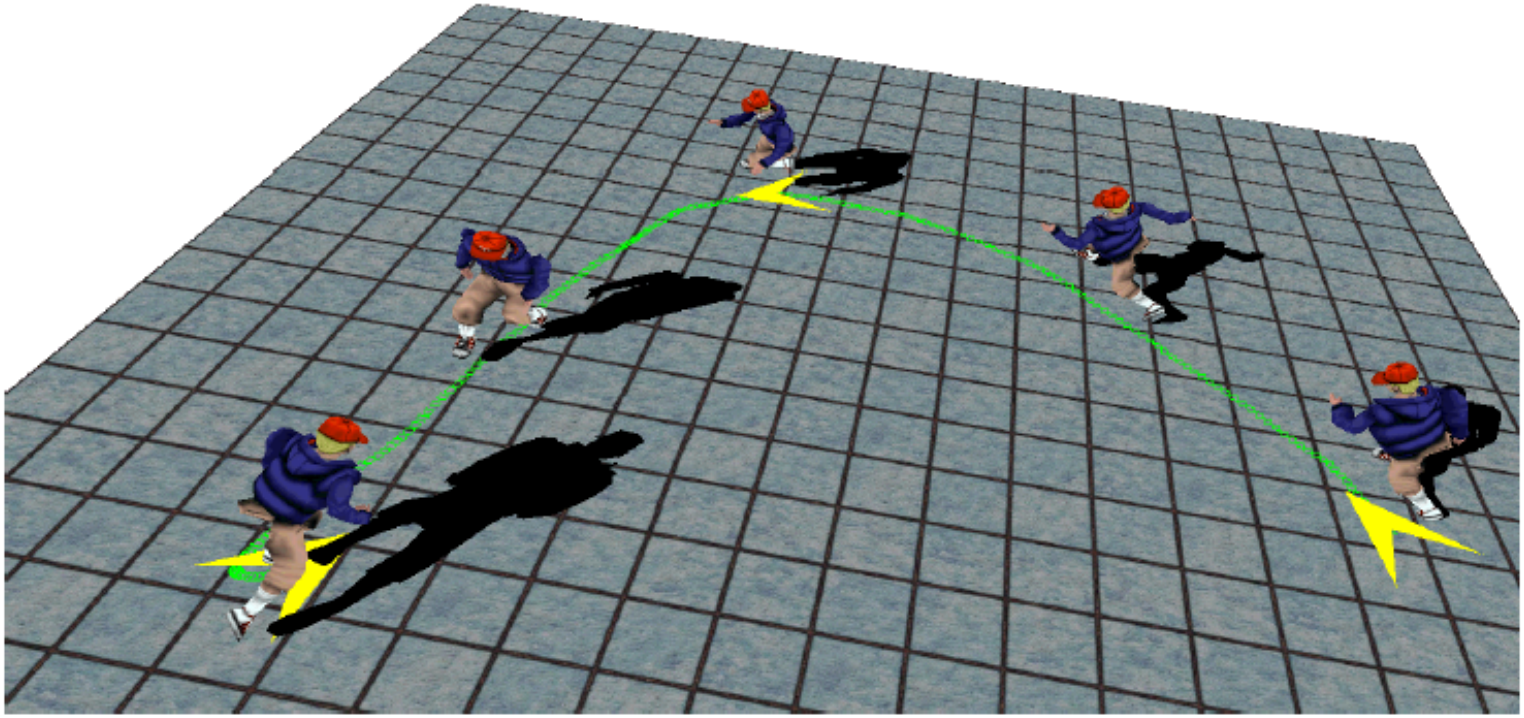
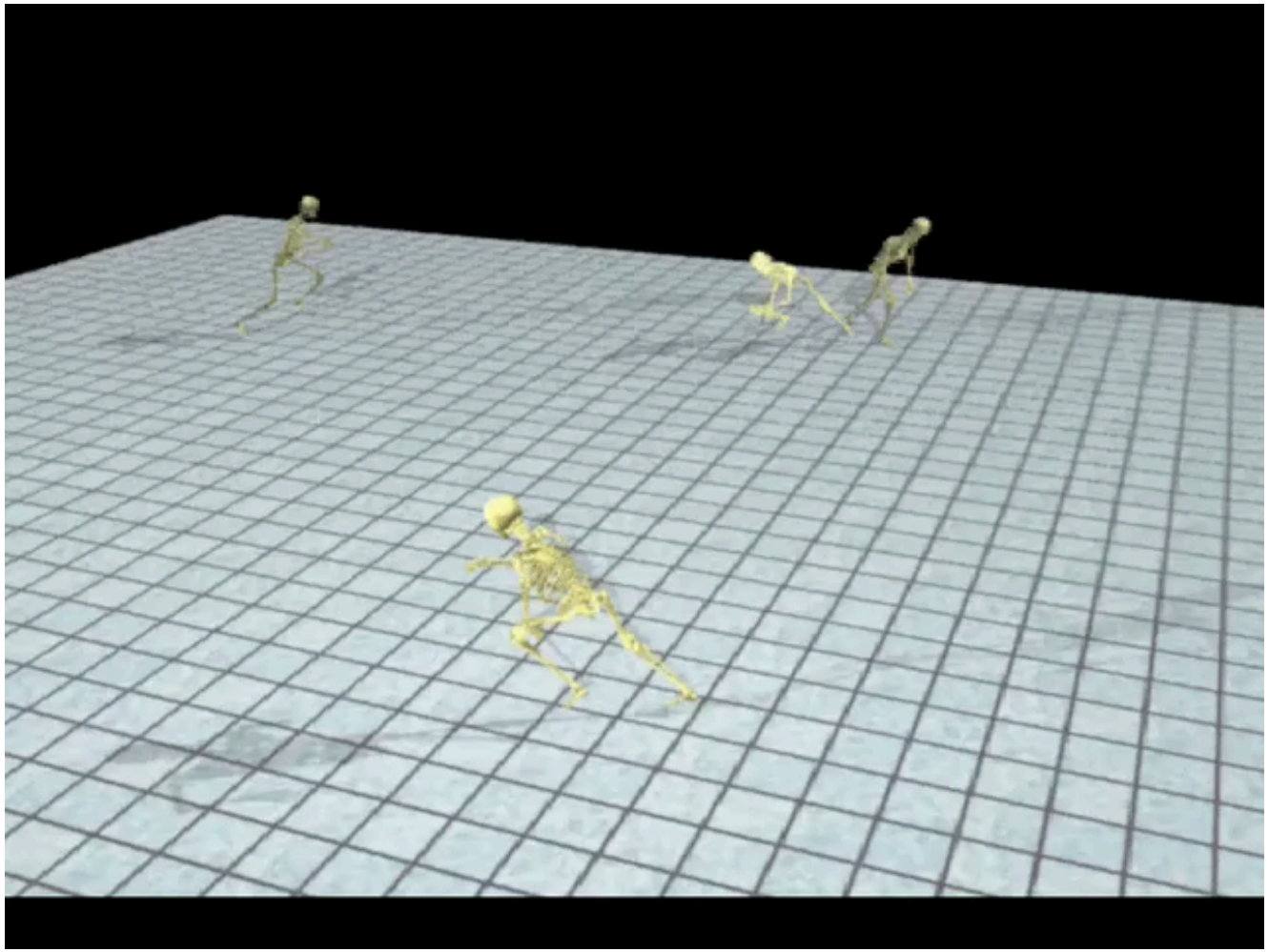
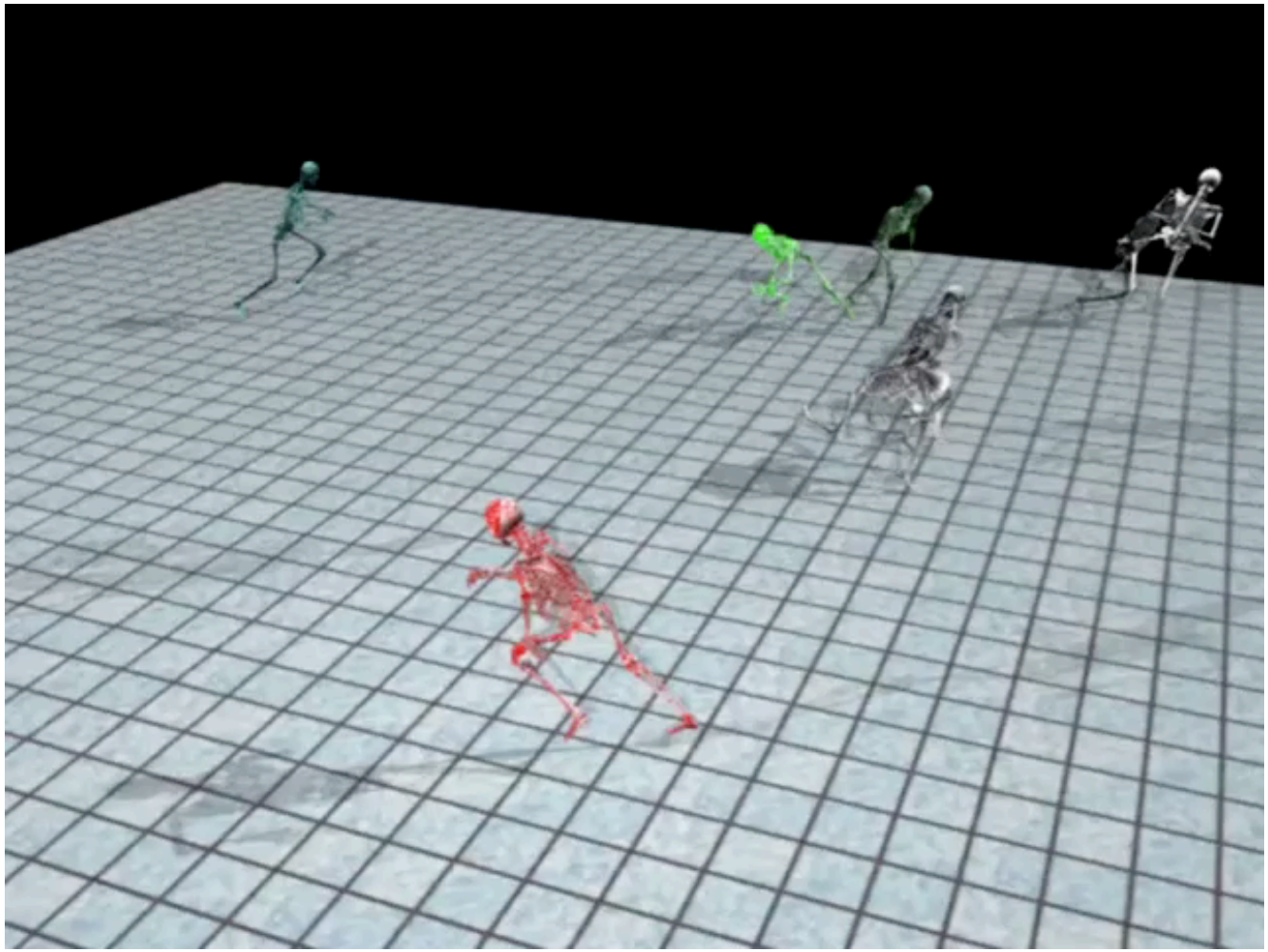
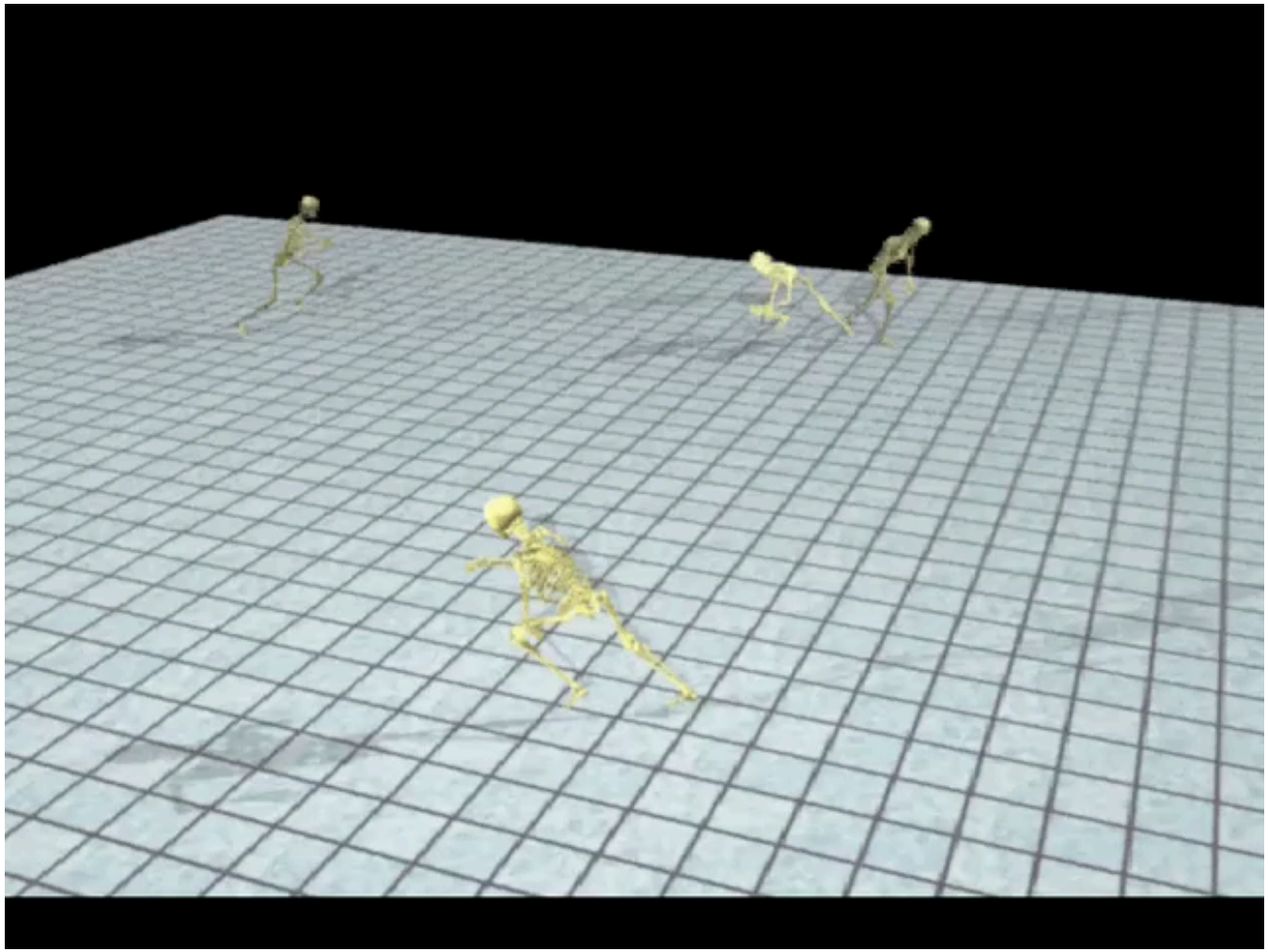
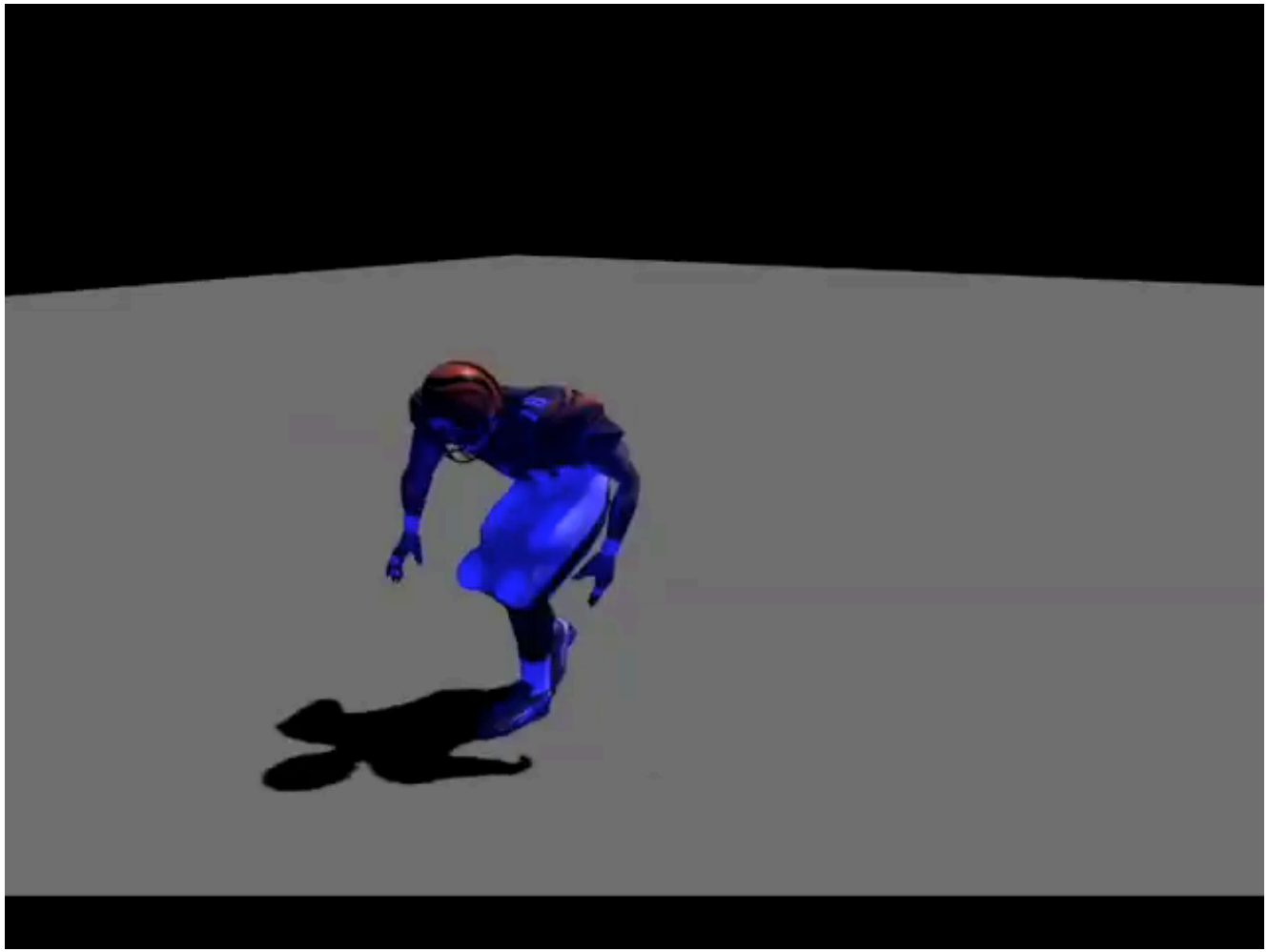


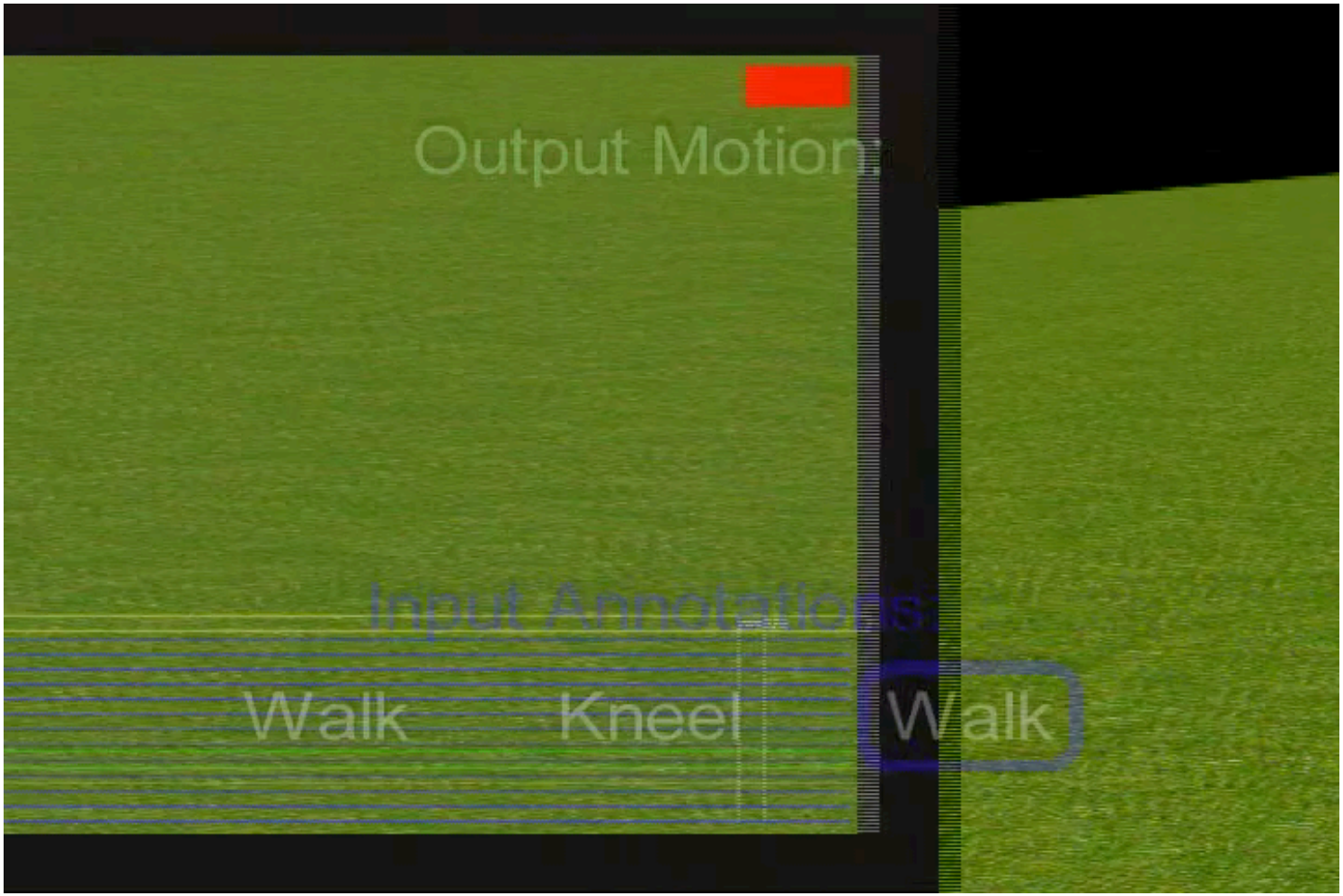
Figure 6: We can use multiple "checkpoints" in a motion. In this figure, the motion is required to pass through the arrow (body constraint) in the middle on the way from the right arrow to the left.

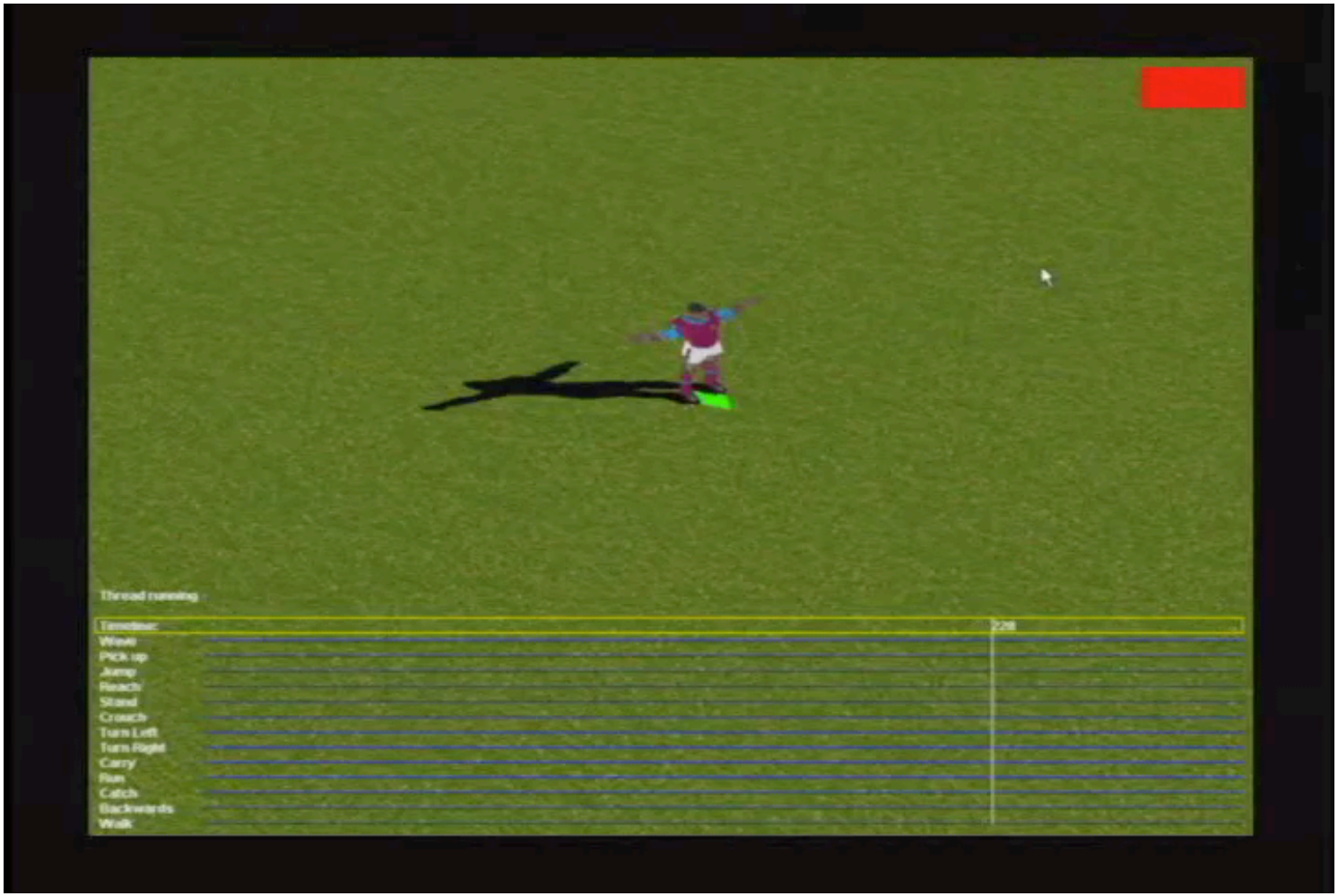









Output Motion:
Input Annotations:
Walk
Kneel
Walk


Thread running
Timeline:
228
Wave
Pick up
Jump
Reach
Stand
Crouch
Turn Left
Turn Right
Carry
Run
Catch
Backwards
Walk