Planning with Dynamics

Project

1. Discrete plan:
   - Plan a path through *configuration space*.
2. Reference trajectory:
   - Approximate it with a feasible trajectory through *state space*.
3. Feedback control:
   - Compute control actions to follow that trajectory.

- No one solution. Don’t panic! This remains an active area of research!
- We’ve provided a bag of tools, but there is a lot of space for creativity.

A Powerful Paradigm

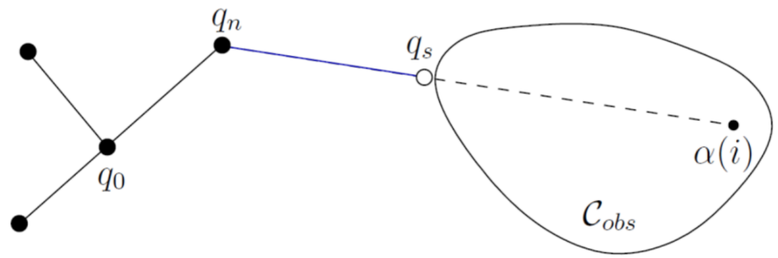
1. Discrete plan:
   - Plan a path through *configuration space*.
2. Reference trajectory:
   - Approximate it with a feasible trajectory through *state space*.
3. Feedback control:
   - Compute control actions to follow that trajectory.

Re-plan with new information.

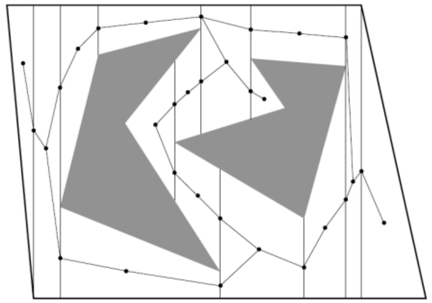
Dinesh Thakur, Giuseppe Loianno, Laura Jarin-Lipschitz, Alex Zhou, and Vijay Kumar, 2019
[“Autonomous Inspection of a Containment Vessel using a Micro Aerial Vehicle”]

Blending Path Planning and Trajectory Planning

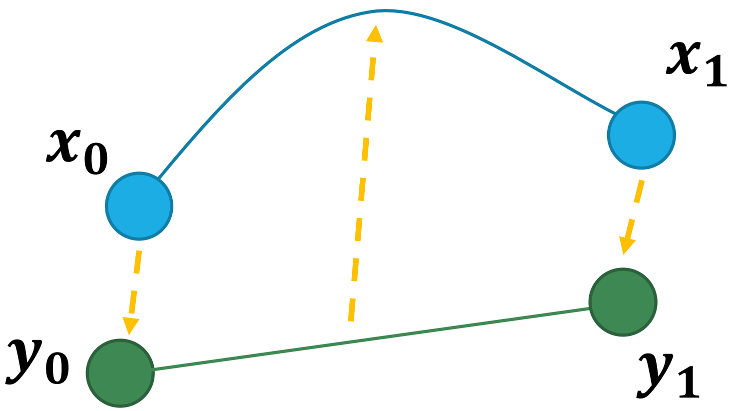
When we first talked about path planning, we assumed the existence of a *local planning method* to connect two configurations.



- Traveling from waypoint to waypoint.
- Accessing and departing from a roadmap.
- Adding new milestones to a PRM.
- Adding vertices to an RDT/RRT.



The differential flatness property makes this relatively easy. It lets us cheaply solve the boundary value problem.



Planning with Dynamics

Can we incorporate dynamics directly into the initial plan?

Why would we need to?

Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar, 2016
"Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU"

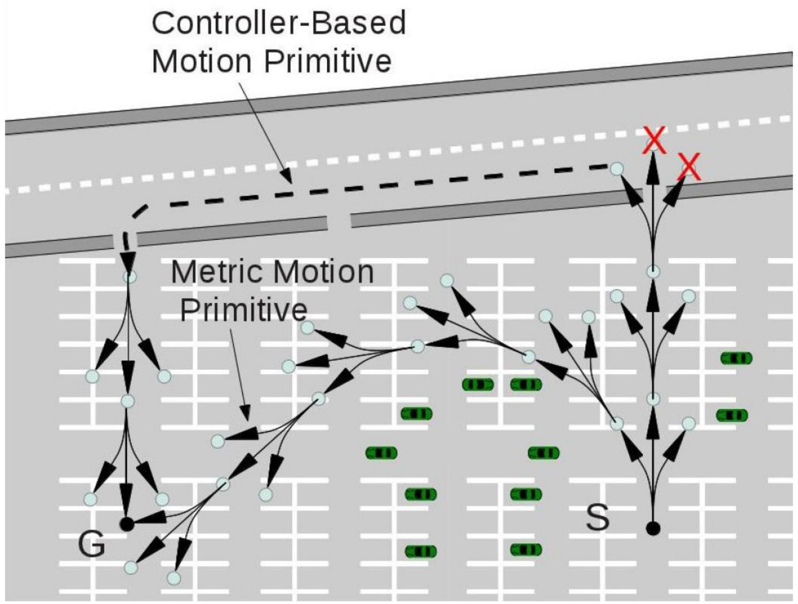
How would you specify constraints for this trajectory?

“Motion Primitives”

A small twist on our discrete search idea:

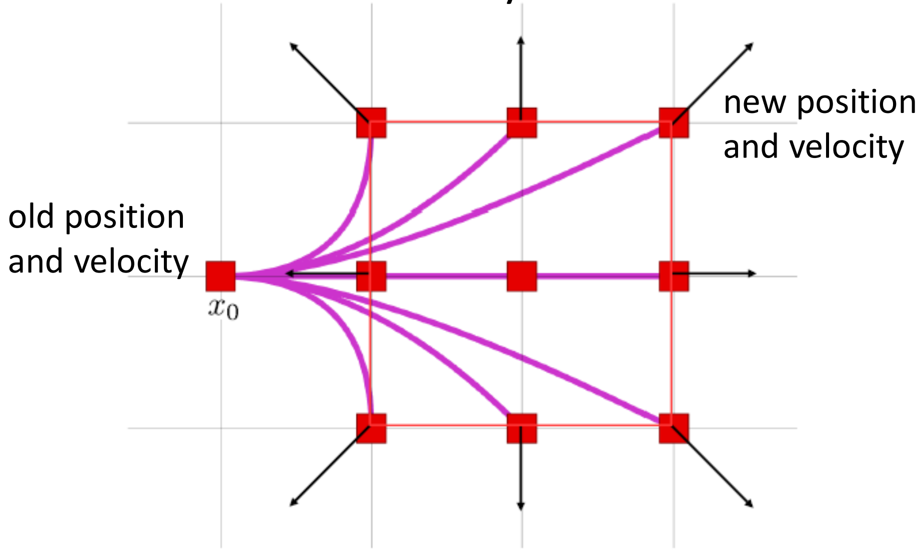
- Don’t search for a sequence of neighboring configurations,
- Instead, search for a sequence of short actions or “motion primitives.”

A motion primitive takes you from one *state* to another *state* over some time T.



Butzke, 2014

for a second order system



Liu, 2017

Search-Based Trajectory Planning

Extended Example: Sikang Liu, 2018

ie. Implicit Graph Search using Motion Primitives

- We might know "good," short trajectories in free space (e.g. from differential flatness).
- We don't actually need an *explicit* graph representation to use A* search.

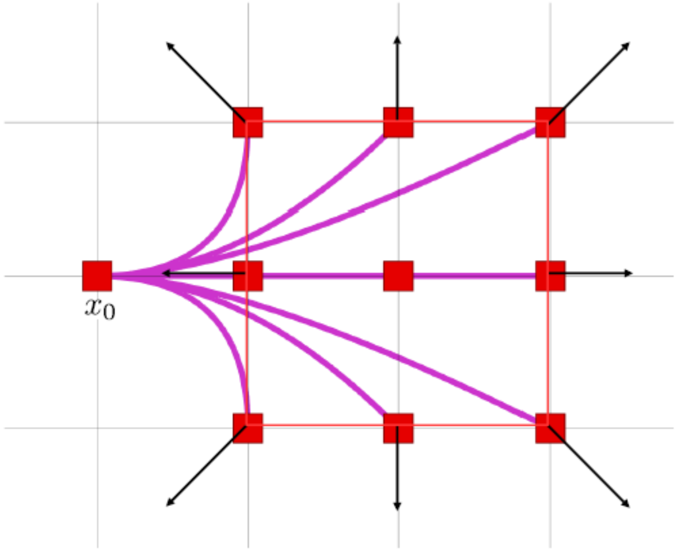
Only modification to A*:

'Expand' vertices by applying the motion primitives to yield new vertices.

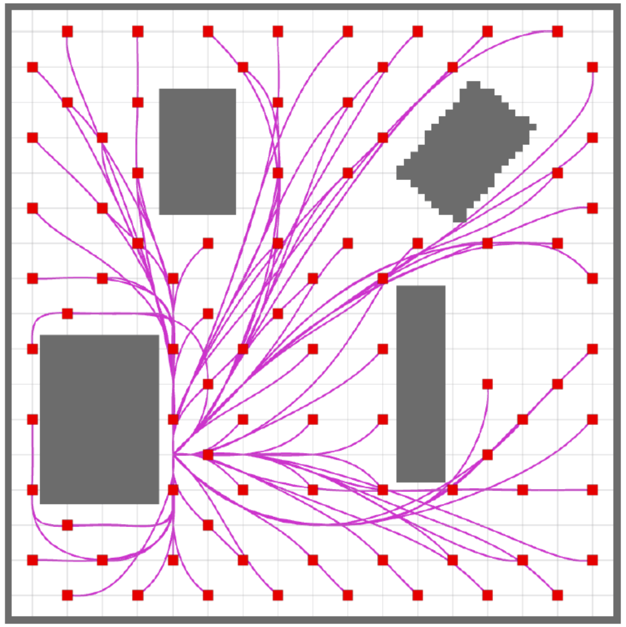
Challenges:

- Good primitives
- Good heuristic

Select finite number of motion primitives.

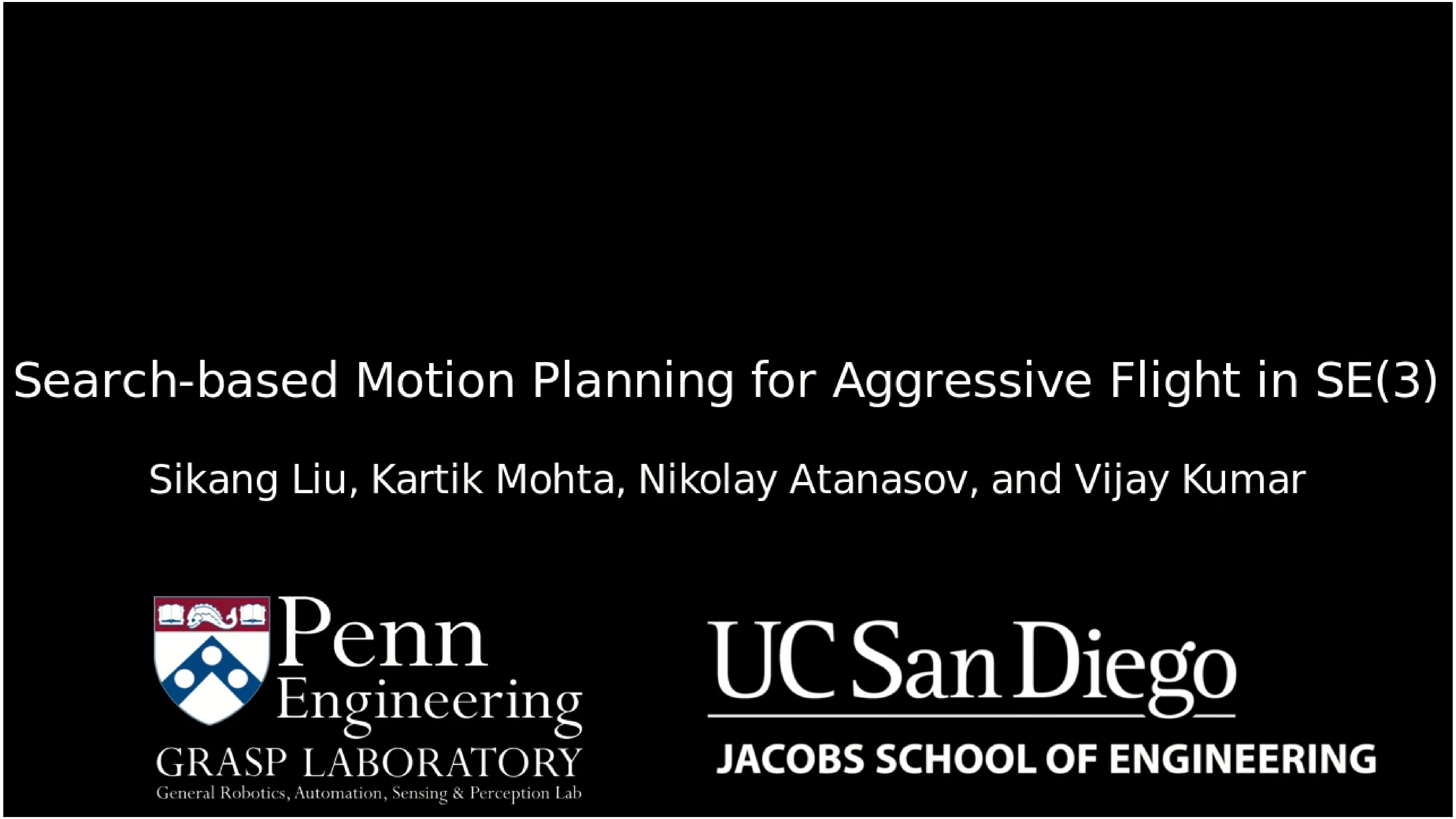


Graph represents sequences of motions primitives.



Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar, 2018

"Search-Based Motion Planning for Aggressive Flight in SE(3)"



What are good quadrotor motion primitives?

Main Idea: Design “optimal” motion primitives using differential flatness.

Consider the extended state

Connection to SE(3)?

$$\mathbf{s}(t) := [\mathbf{x}(t)^\mathsf{T}, \dot{\mathbf{x}}(t)^\mathsf{T}, \ddot{\mathbf{x}}(t)^\mathsf{T}]^\mathsf{T} = [\mathbf{p}^\mathsf{T}, \mathbf{v}^\mathsf{T}, \mathbf{a}^\mathsf{T}]^\mathsf{T}$$

For a constant jerk input, the state changes as

$$t \in [0, \tau]$$

$$\mathbf{s}(t) = F(\mathbf{u}_m, \mathbf{s}_0, t) := \begin{bmatrix} \mathbf{u}_m \frac{t^3}{6} + \mathbf{a}_0 \frac{t^2}{2} + \mathbf{v}_0 t + \mathbf{p}_0 \\ \mathbf{u}_m \frac{t^2}{2} + \mathbf{a}_0 t + \mathbf{v}_0 \\ \mathbf{u}_m t + \mathbf{a}_0 \end{bmatrix}$$

What is the cost of this segment?

$$C(\mathbf{s}_n, \mathbf{u}_m) = C(\mathbf{u}_m) = (\|\mathbf{u}_m\|^2 + \rho)\tau$$

Consider the cost function and optimization problem

$$\Phi^*(t) = \underset{\Phi(t)}{\arg\min}\, J + \rho T = \underset{\Phi(t)}{\arg\min} \int_0^T \|\mathbf{j}\|^2 dt + \rho T$$

$$\text{s.t. } \mathbf{s}_0 \leftarrow \Phi(0),\ \mathbf{s}_g \leftarrow \Phi(T)$$

It can be shown this is the lowest cost trajectory from s_0 to s_g.

(But note there was nothing special about s_g, it’s just where the constant jerk trajectory lands.)

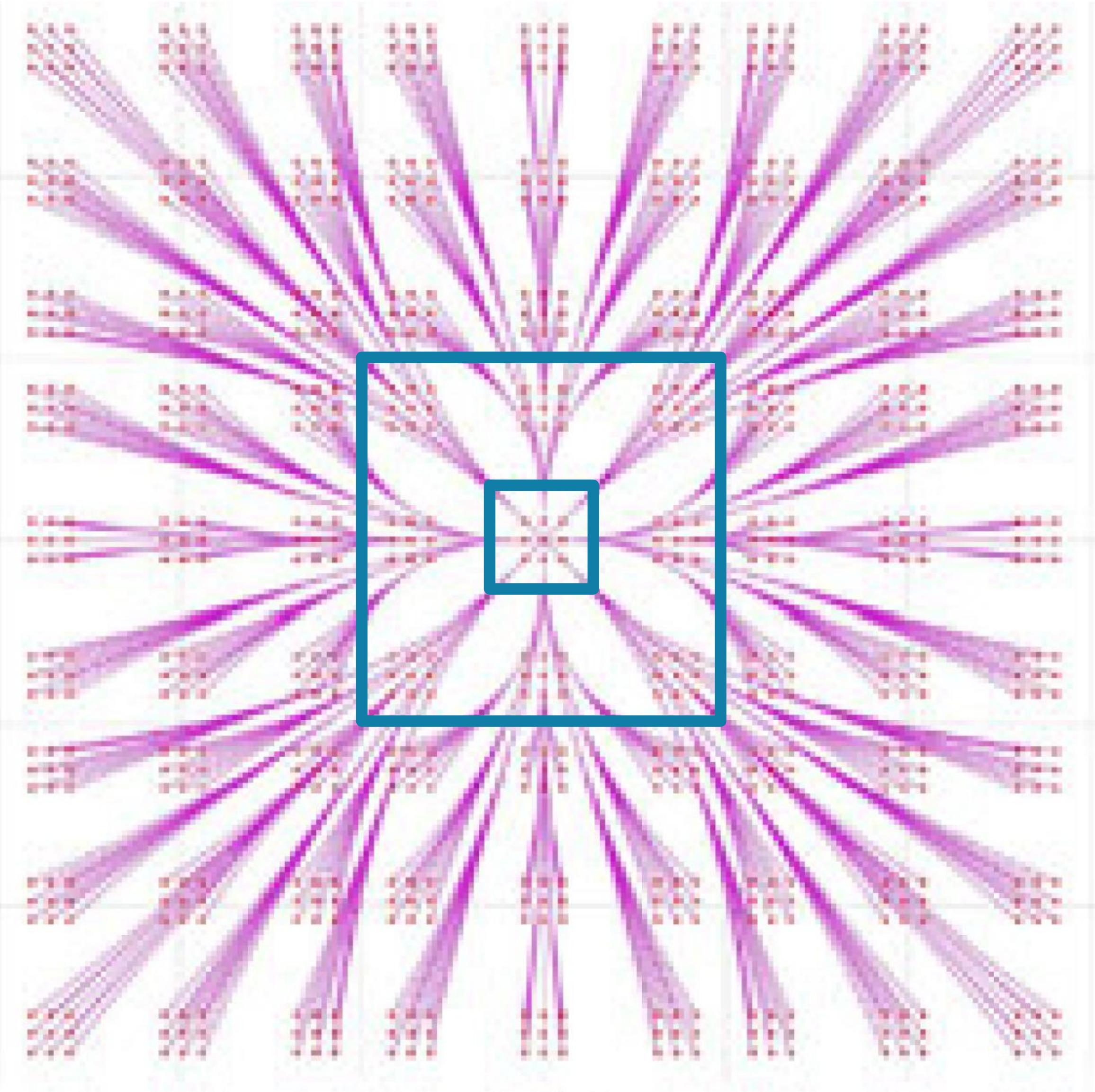
We can chain together motion primitives.

Breadth-first expansion over horizon of two motion primitives.

Select from 9 primitives.

Lines are constant jerk motion primitives.

Dots represents a state: a unique position *and a specific velocity and acceleration*.



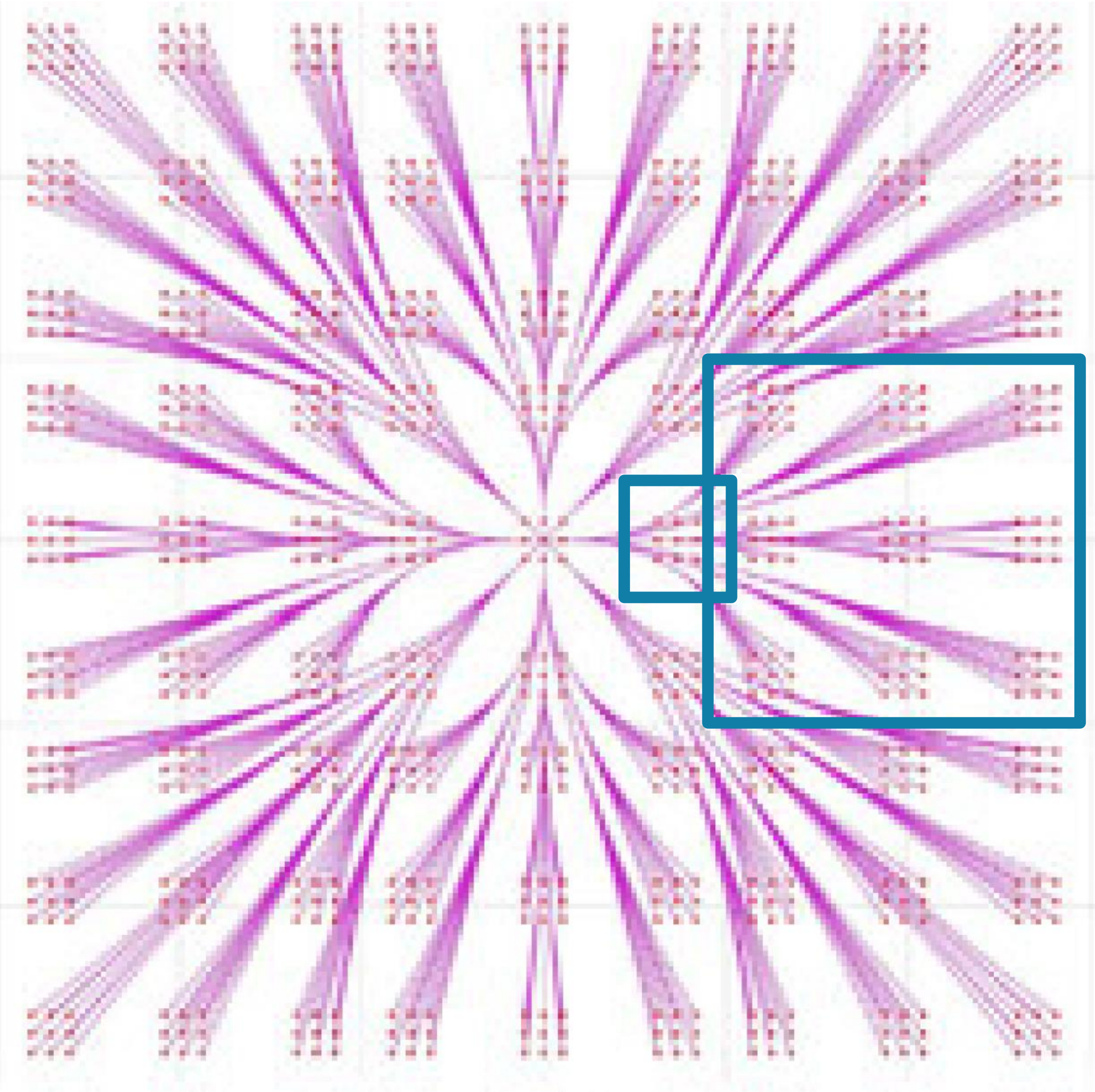
We can chain together motion primitives.

Breadth-first expansion over horizon of two motion primitives.

Select from 9 primitives.

Lines are constant jerk motion primitives.

Dots represents a state: a unique position *and a specific velocity and acceleration*.



What is the cost of a trajectory of motion primitives?

Simply the sum of the motion primitive costs.

$$\min_{N,\mathbf{u}_{0:N-1}} \left(\sum_{n=0}^{N-1} \|\mathbf{u}_n\|^2 + \rho N \right) \tau$$

sum of segment costs

$$\text{s.t. } F_n(t) := F(\mathbf{u}_n, \mathbf{s}_n, t), \ \mathbf{u}_n \in \mathcal{U}_M$$

each segment is a constant input trajectory

$$\mathbf{s}_{n+1} = F_n(\tau) = F_{n+1}(0), \ \mathbf{s}_N \in \mathcal{X}^{goal}$$

the end of one segment is the beginning of the other

$$F_n(t) \subset \mathcal{X}^{free}$$

the final segment reaches the goal

the segments don't collide

$$\Phi^*(t) \leftarrow [\mathbf{s}_0 \xrightarrow{\mathbf{u}_0^*} \mathbf{s}_1 \ldots \xrightarrow{\mathbf{u}_{N-1}^*} \mathbf{s}_N]$$

the optimal trajectory is the sequence of steps with the smallest cost

Optimization as an Implicit Graph Search!

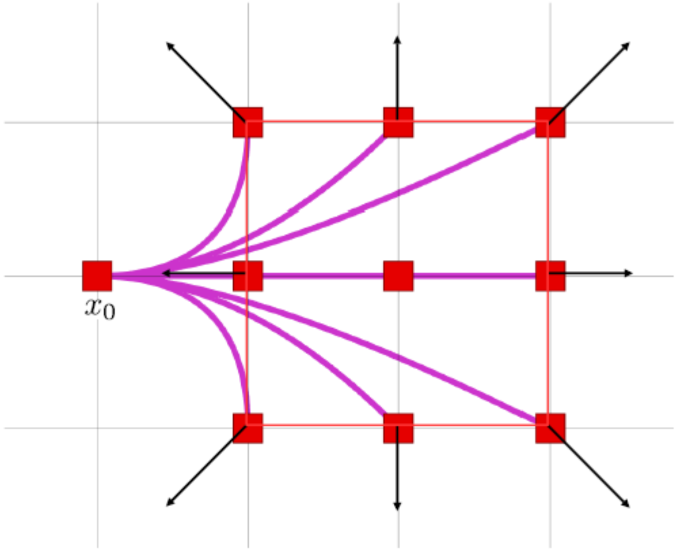
We don't actually need an *explicit* graph representation to use A* search.

(And the search space could be infinite, but that's ok.)

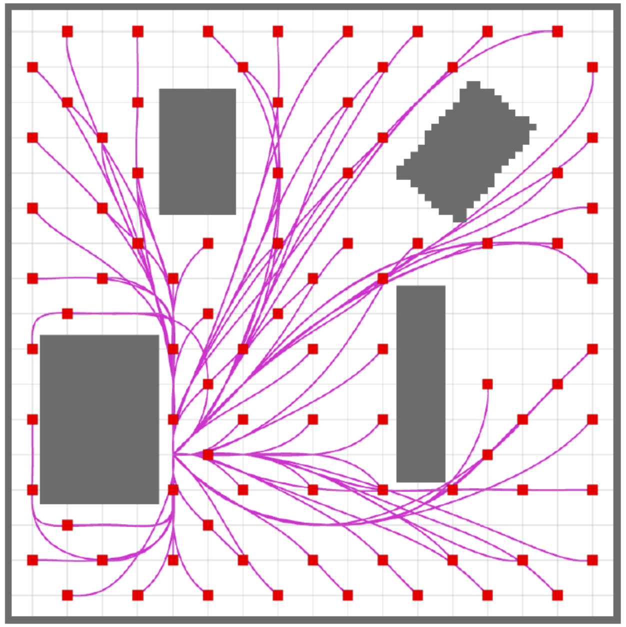
Only modification to A*:

'Expand' vertices by applying the motion primitives to yield new vertices.

Select finite number of motion primitives.



Graph represents sequences of motions primitives.



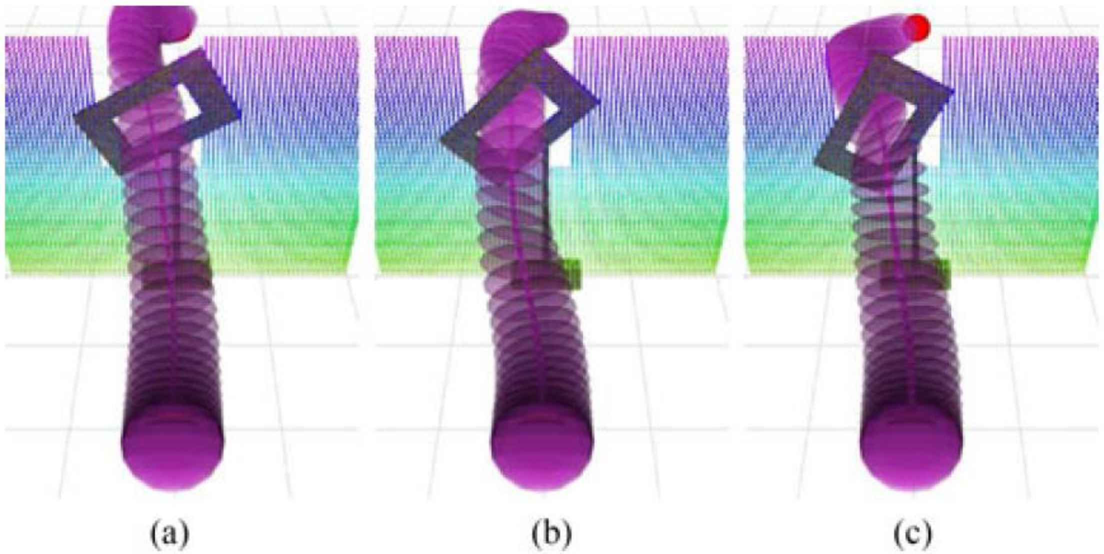
What are the feasible "neighbors?"

All states I can reach with a motion primitive.

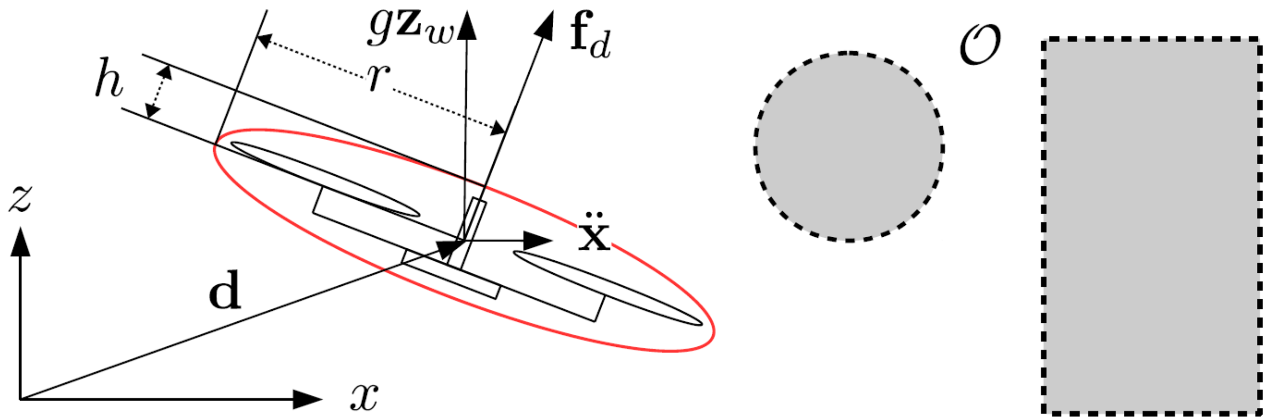
$$s_{n+1} = F(u_n, s_n, \tau), \qquad u_n \in U_M$$

Which obey dynamic constraints.

$$|\dot{\mathbf{x}}(t)| \preceq \mathbf{v}_{\max}, \ |\ddot{\mathbf{x}}(t)| \preceq \mathbf{a}_{\max}, \ |\dddot{\mathbf{x}}(t)| \preceq \mathbf{j}_{\max}$$



And yields no collisions.

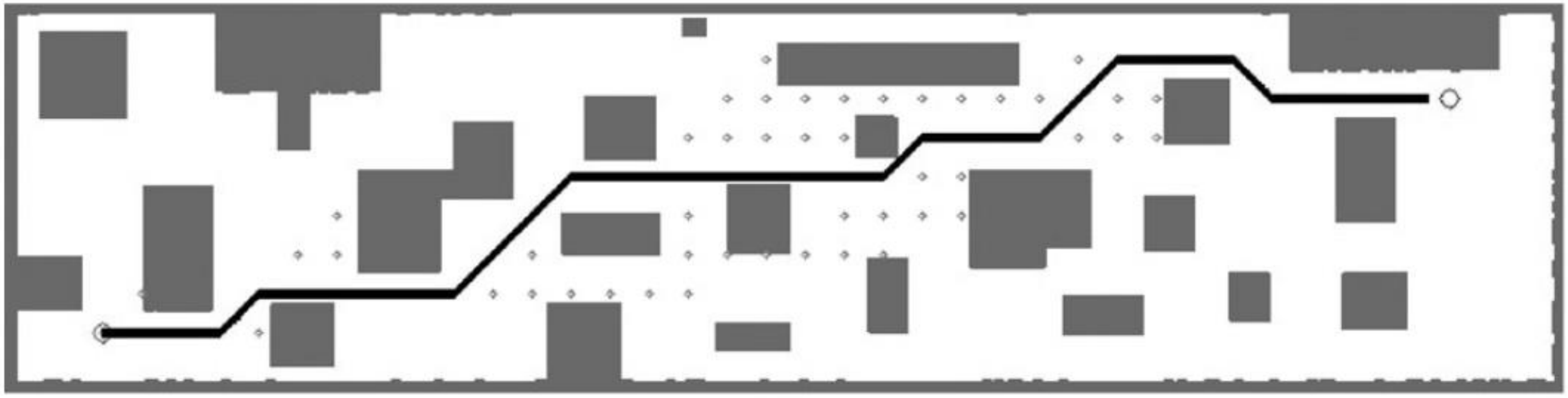


Simple Example: Constant velocity inputs in 2D

For each motion primitive, apply a constant velocity input for a short time.

Choose between 8 different directions for the velocity.

What do the paths look like?

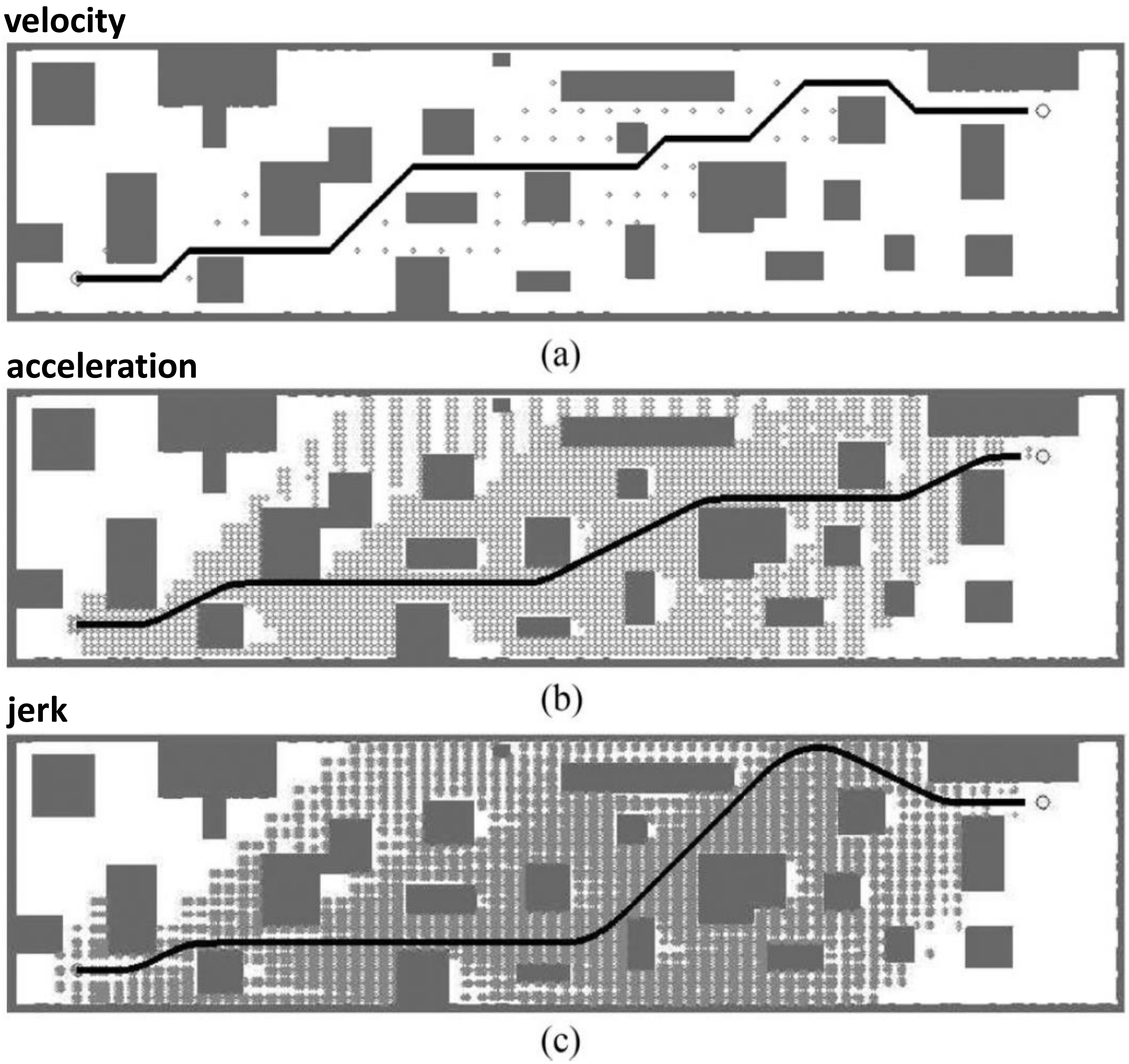


Challenges

Planning in state space like this may mean planning in high dimensions.

How can we make this practical?

- Good primitives.
- Good heuristics.
- Hierarchical planning.



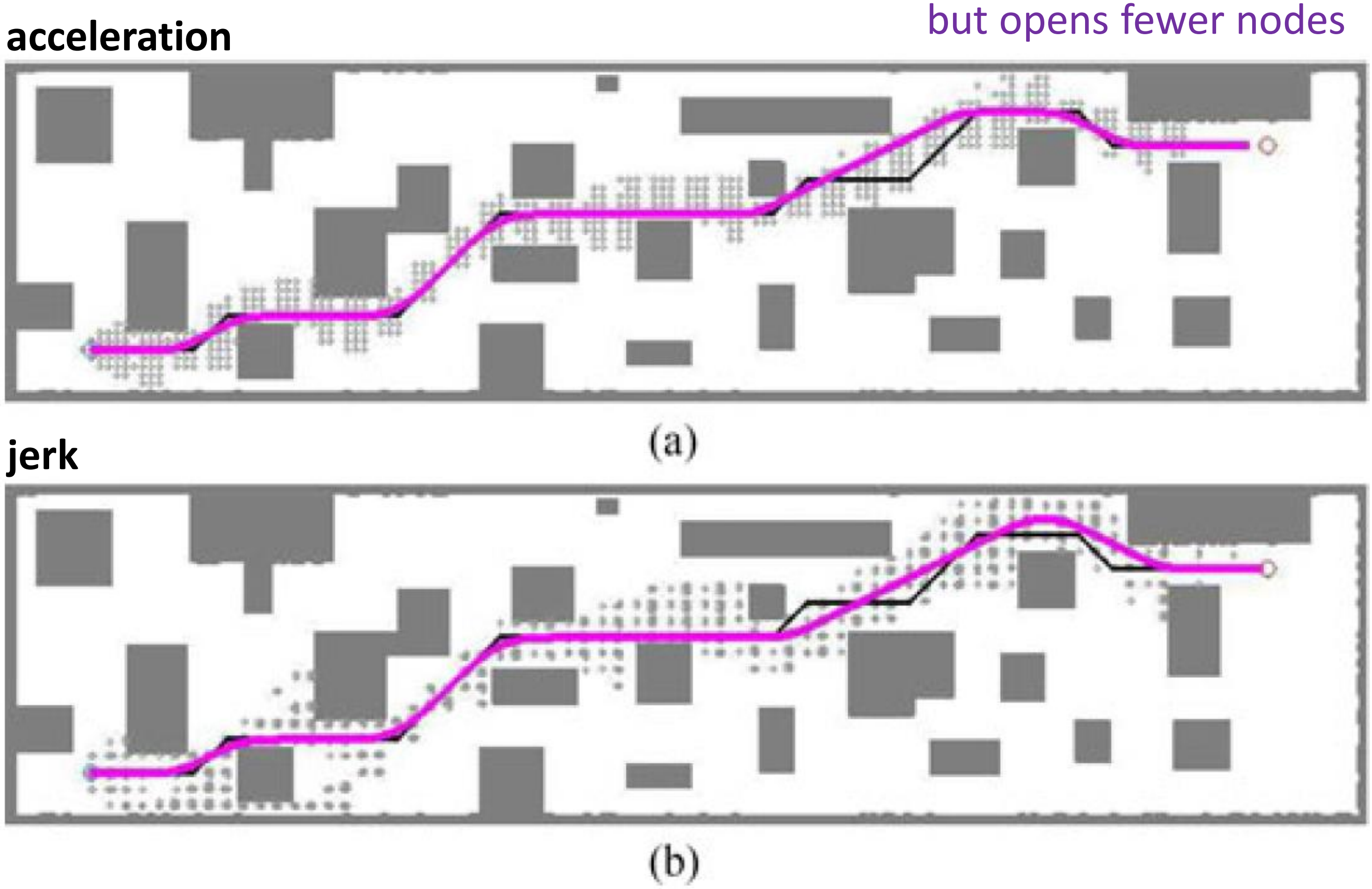
Each higher derivative input is a more constrained version of the previous problem, and must have a higher cost.

Heuristics from Hierarchical Planning

A ‘relaxed’ version of the problem is often a good placed to seek a heuristic.

Liu 2018 uses the lower-order solution cost as *inadmissible* heuristic.

heuristic guides search towards suboptimal path but opens fewer nodes



Aside: A* and Heuristics:

Admissible and consistent:

- *Solutions are optimal.*

Admissible, not consistent:

- Possibly suboptimal solutions unless re-opening closed nodes is allowed.

Not admissible, not consistent:

- Possibly suboptimal solutions.

recall a consistent heuristic is always admissible.

RRTs with Motion Primitives

Motion primitives are naturally suited to RRTs as well.

Key Idea: Grow tree by sampling from controls or 'motion primitives.'

SIMPLE_RDT_WITH_DIFFERENTIAL_CONSTRAINTS(x_0)

```
G.init(x_0);
for i = 1 to k do
    x_n ← NEAREST(S(G), α(i));
    (ũ^p, x_r) ← LOCAL_PLANNER(x_n, α(i));
    G.add_vertex(x_r);
    G.add_edge(ũ^p);
```

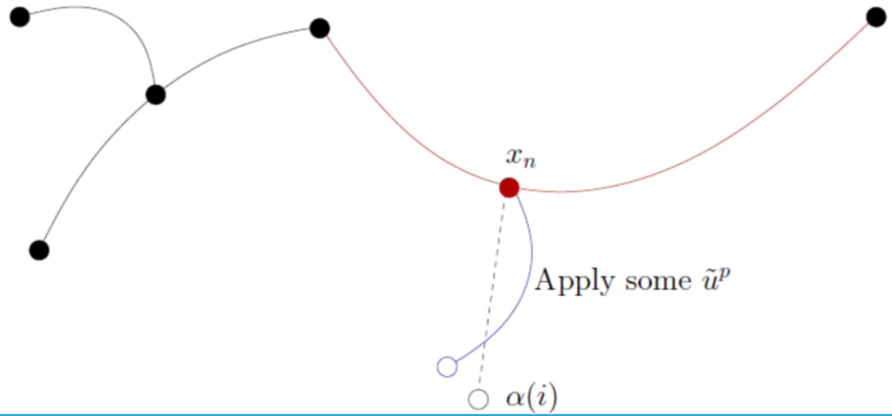
start graph from origin

sample a point α, and

find nearest point x_n on the *swath* S

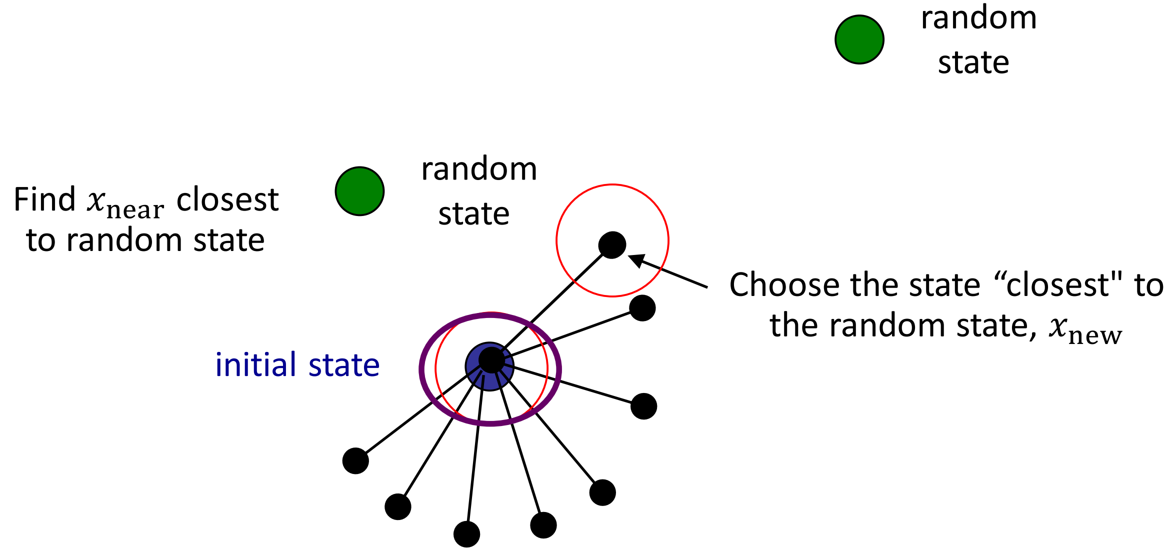
sample motion primitives to reach from x_n towards α

connect new point x_r to the graph



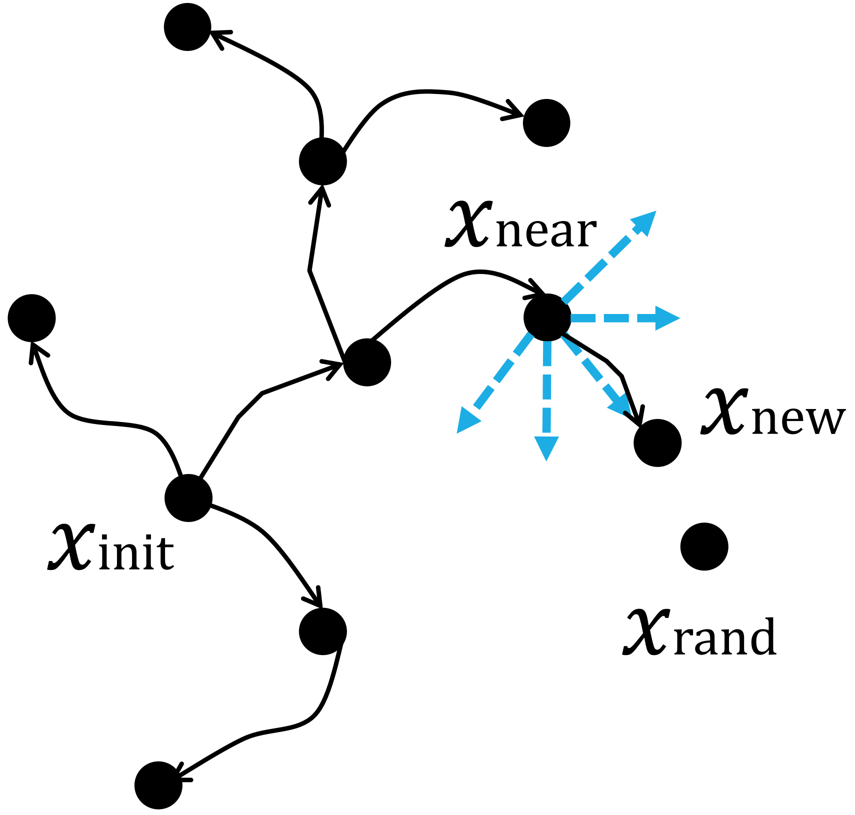
LaValle, *Planning Algorithms*, 2006

RRTs with Motion Primitives

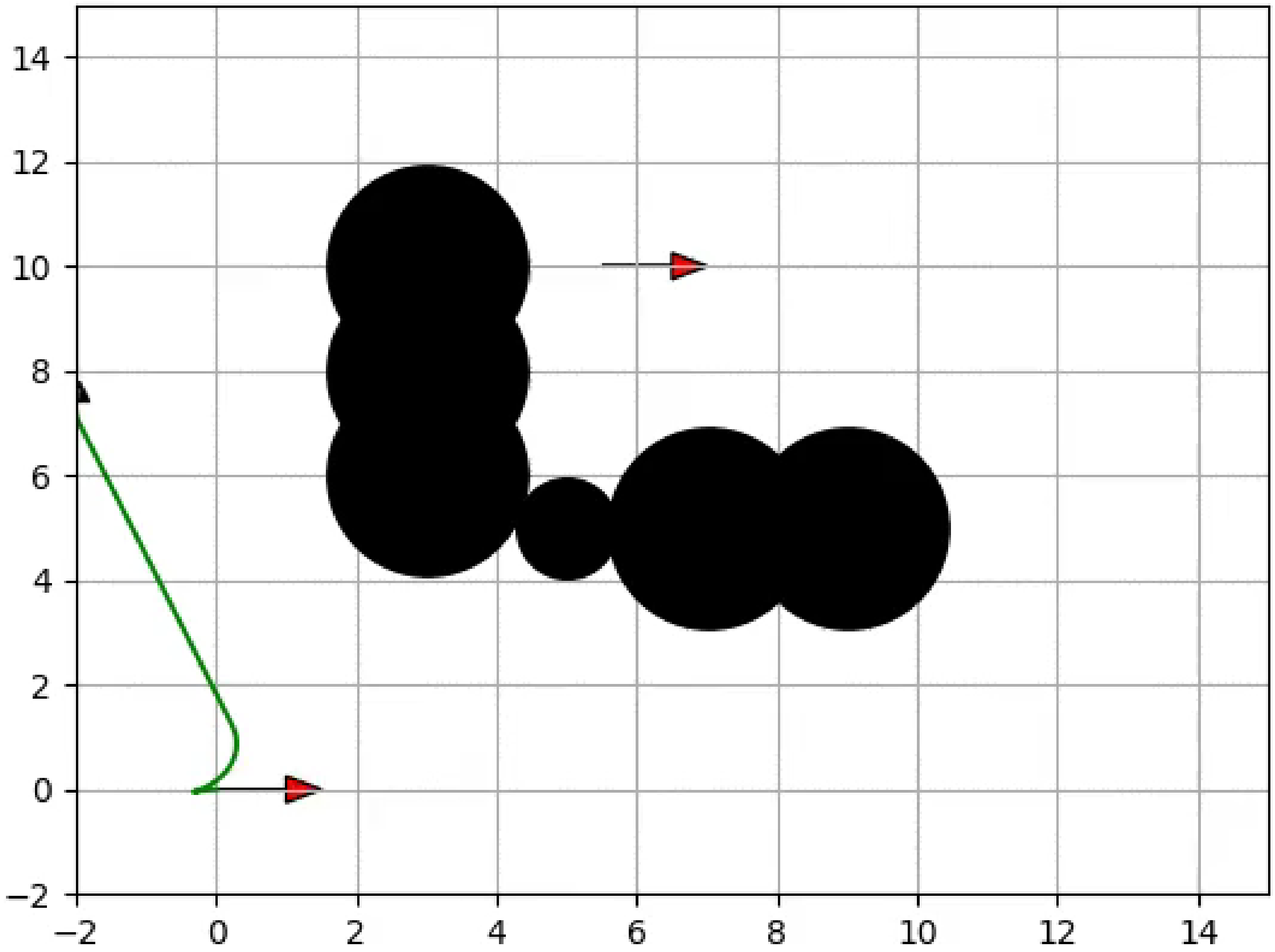


Explore motions from the chosen vertex by trying possible inputs (motion primitives)

kth Step



RRT with Reeds-Shepp Bicycle Model



Atsushi Sakai, 2018
Python Robotics

What are the challenges?

Next Lab Sessions Monday/Tuesday

Similar to last session, little preparation is required.

Does require a solid start on the project in simulation.

Your laboratory approach should be much more conservative than in simulation.

- If it ain't broke, don't fix it.

Midterm Exam

Thursday, 3/5 during class.

One sheet of notes, front and back.

No book, no calculators.

Previous exams from 2018 and 2019 are on Canvas.

Tuesday will be an in-class review. Please request questions/topics to Piazza.