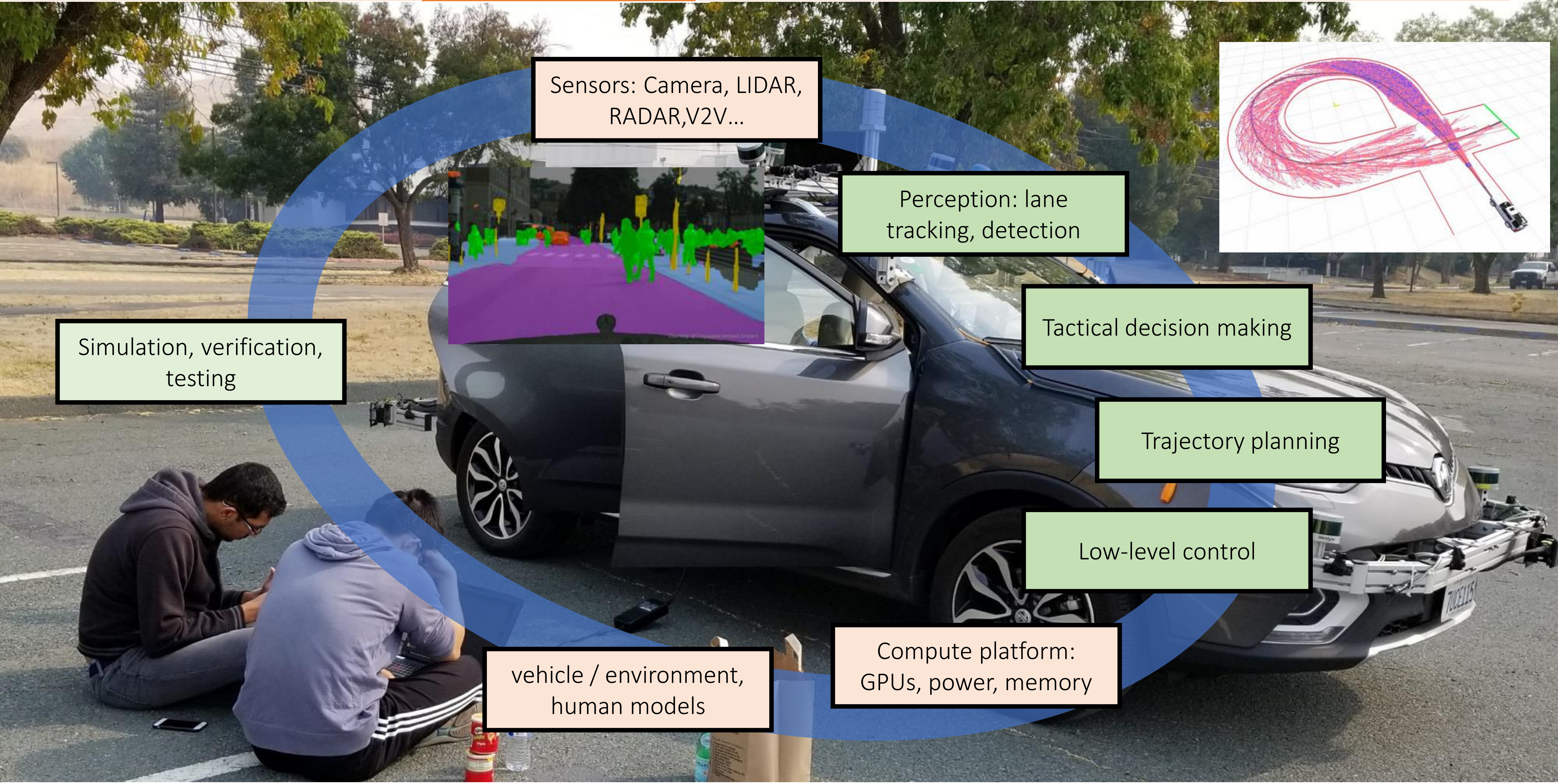
Motion Planning

ECE/CS498

Katie DC

Sensors → Perception → Tactical decision making → Trajectory planning → Low level controller → Simulation and validation

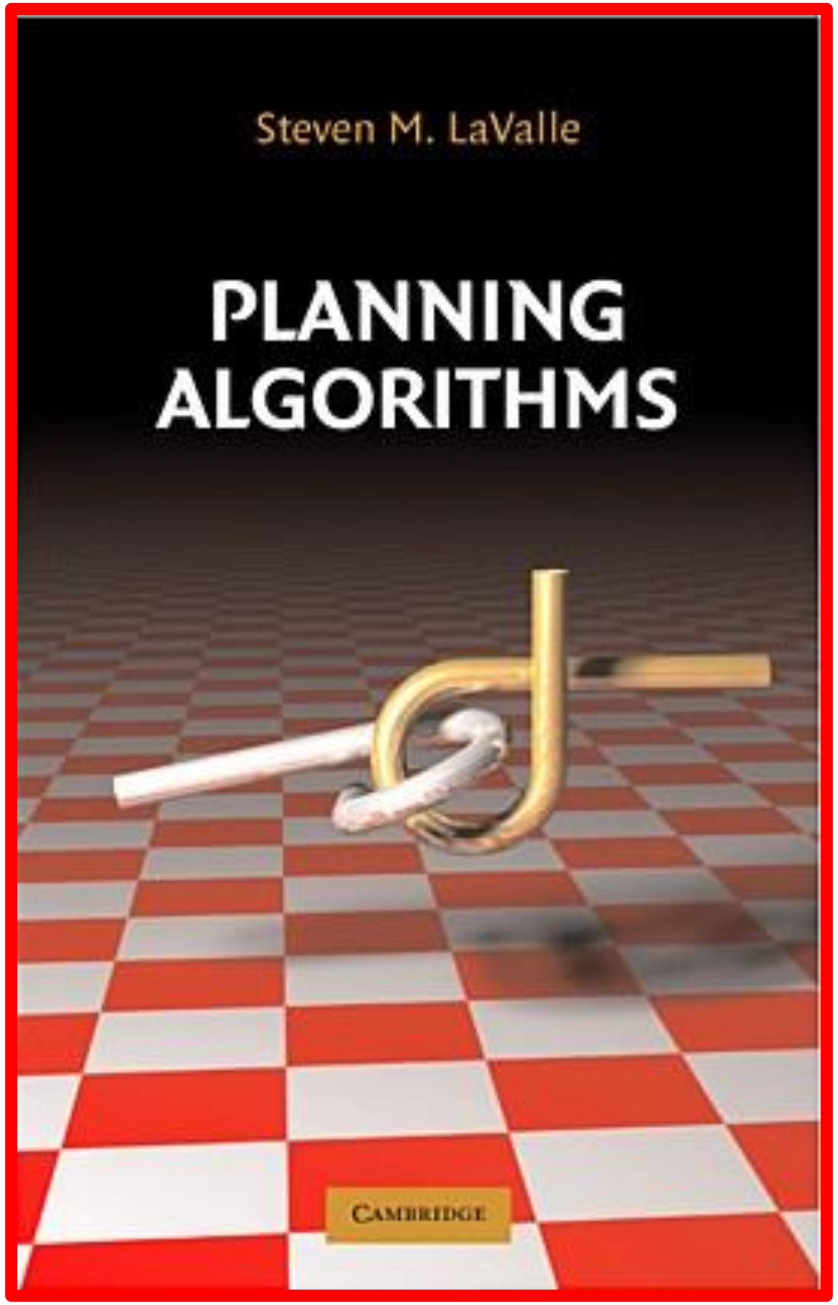


CSL Prof. LaValle central to Oculus' $2 billion success

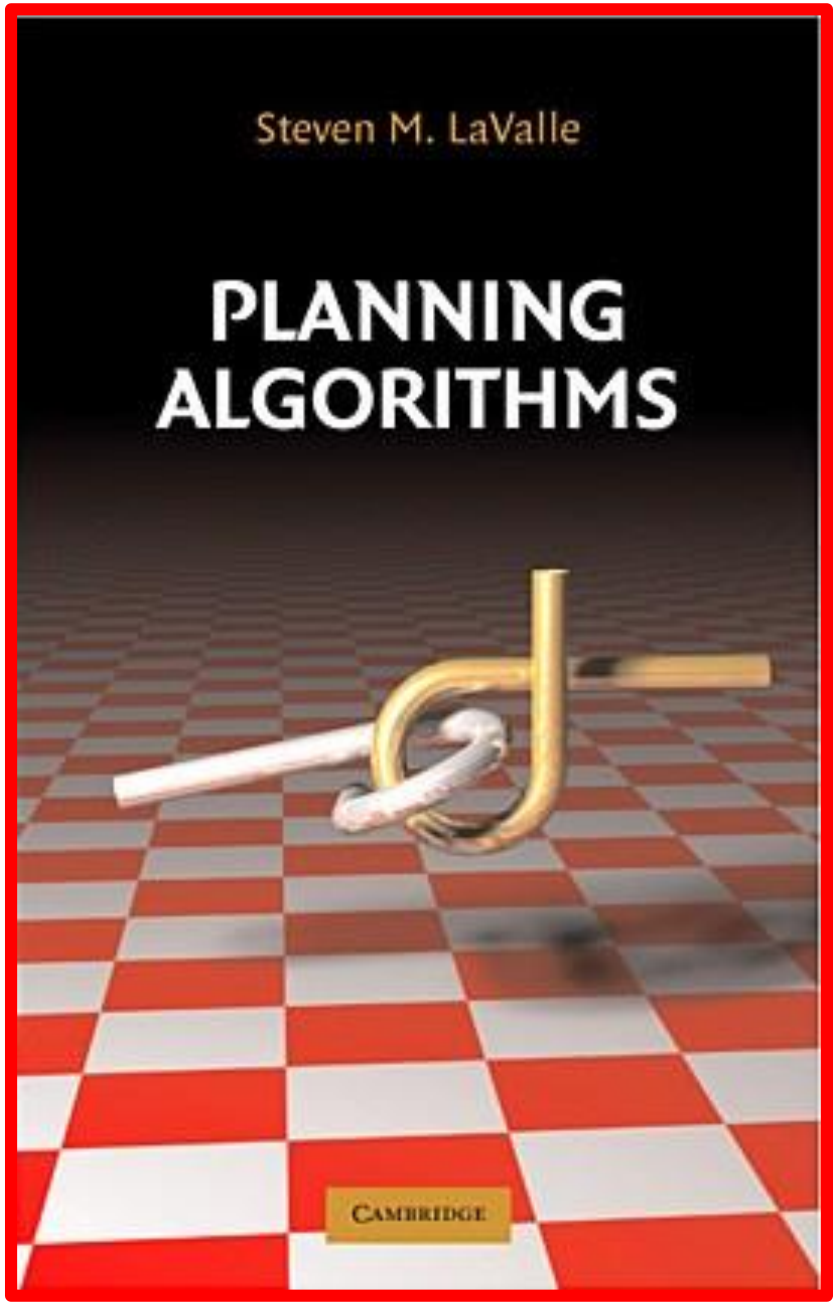
Apr 17, 2014

Rick Kubetz, Engineering Communications Office

On March 25, both the business and technology news pages excitedly announced Facebook's $2 billion acquisition of Oculus VR, the maker of a virtual reality gaming headset called Oculus Rift.




Intelligent and Mobile Robotics Group
http://imr.felk.cvut.cz
Alpha Puzzle 1.0


Steven M. LaValle
PLANNING ALGORITHMS
CAMBRIDGE

Motion Planning

Given initial state x_{init} and a goal X_G, what is the path or sequence of control inputs that will lead us from start to goal?

Possible Issues:

- Obstacle avoidance
- Nonholonomic systems
- Computationally intensive
  - Nonconvex optimization
  - Large number of samples required in real-time

Two approaches: optimization-based & sampling-based techniques

Open-Loop Optimal Control

$$\min_{x,u} \sum_{t=0}^{H} C_t(x_t, u_t)$$

$$\text{subject to} \quad x_0 = x_{init}, x_H \in X_G, x_t \in \mathcal{X}_T, u_t \in \mathcal{U}_T$$

$$x_{t+1} = f(x_t, u_t), t = 0, \ldots, H-1$$

This gives the trajectory and sequence of inputs to follow.

In general, this is a nonconvex optimization problem, so it must be solved with a nonconvex solver or with sequential convex programming

What if there is noise or uncertainty?

Model Predictive Control

Given x_{init}

For k = 0,…,T

Solve:

$$\min_{x,u} \sum_{t=k}^{k+H} C_t(x_t, u_t)$$

$$\text{subject to} \quad x_k = \bar{x}_k,\, x_H \in X_G,\, x_t \in \mathcal{X}_T,\, u_t \in \mathcal{U}_T$$

$$x_{t+1} = f(x_t, u_t), t = 0, \ldots, H-1$$

execute u_t

observe $\bar{x}_k$

Collocation versus Shooting

Collocation

- Optimize over the trajectory x and the input u

Shooting

- Optimize directly over u
  - The next state and cost is a computed directly as function of u, following the dynamics
- Why is this nice?
  - We directly improve the control sequence, which is what we put into the system
  - Less likely to converge to a local optima that infeasible
- Why is this difficult?
  - We often need to compute a derivative with respect to u, which is often numerically unstable to compute (especially in case of unstable dynamical systems)
  - Not intuitive to initialize

Sampling-based Motion Planners

Instead of optimizing the entire trajectory and solving a difficult optimization problem, let's try a random sampling approach!

- Probabilistic Road Maps
- Rapidly exploring Random Trees

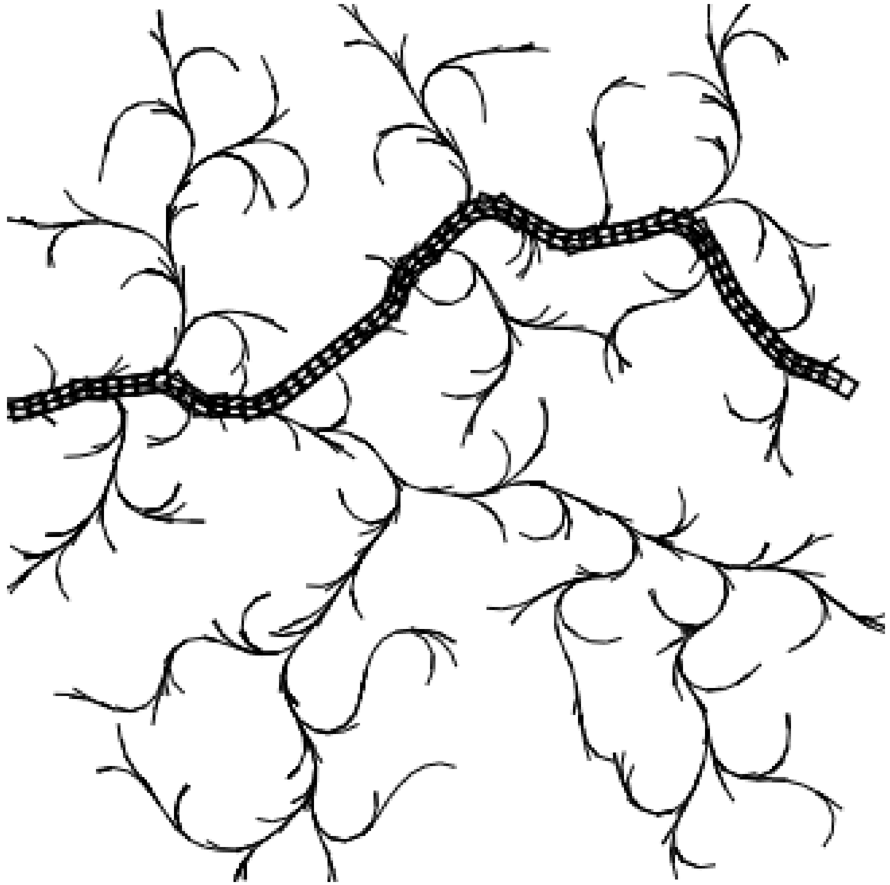


Image Credit: S. Lavalle

Probabilistic Roadmaps (PRM)

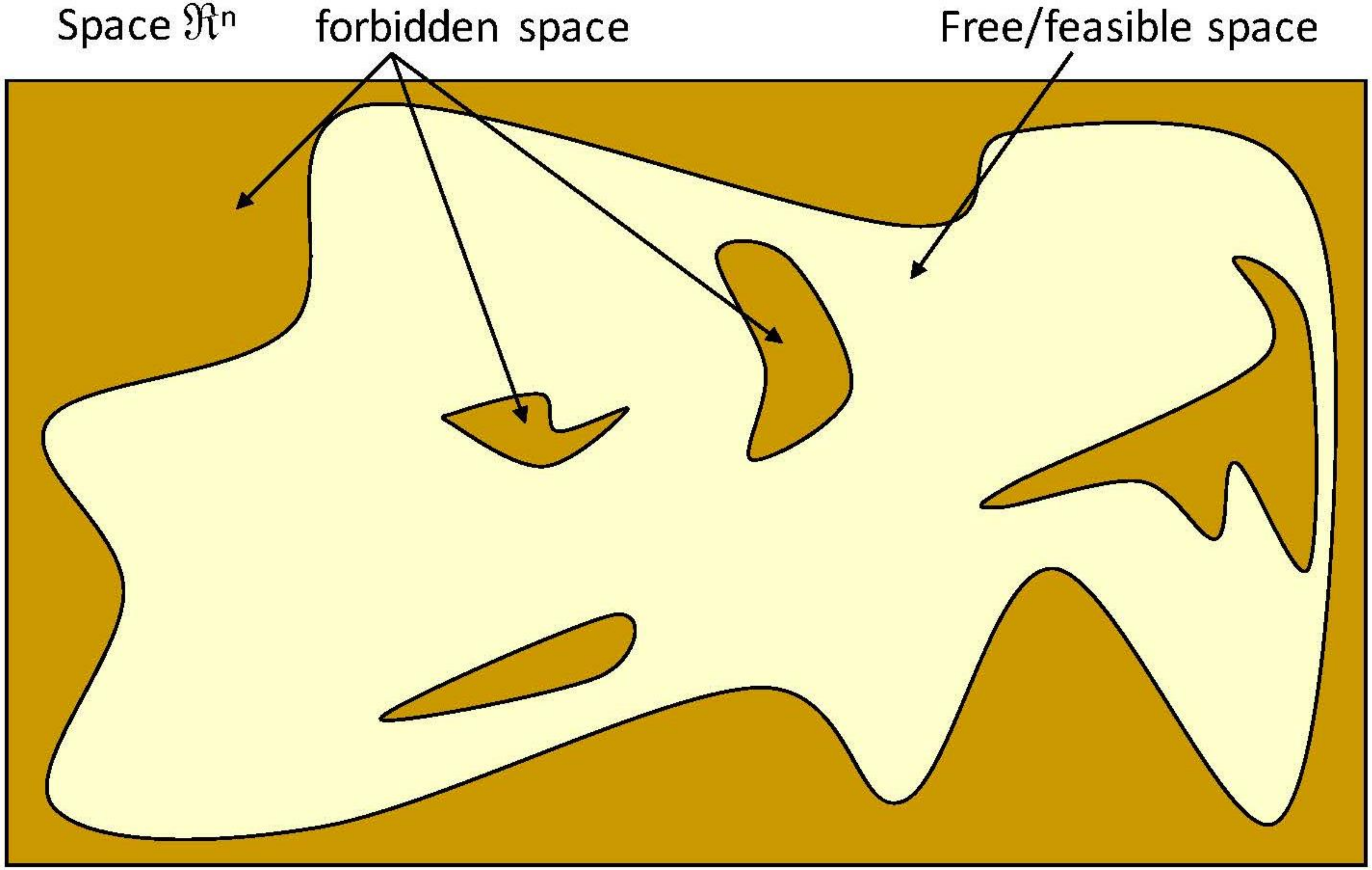


Image Credit: P. Abbeel

Probabilistic Roadmaps (PRM)

Configurations are sampled by picking coordinates at random

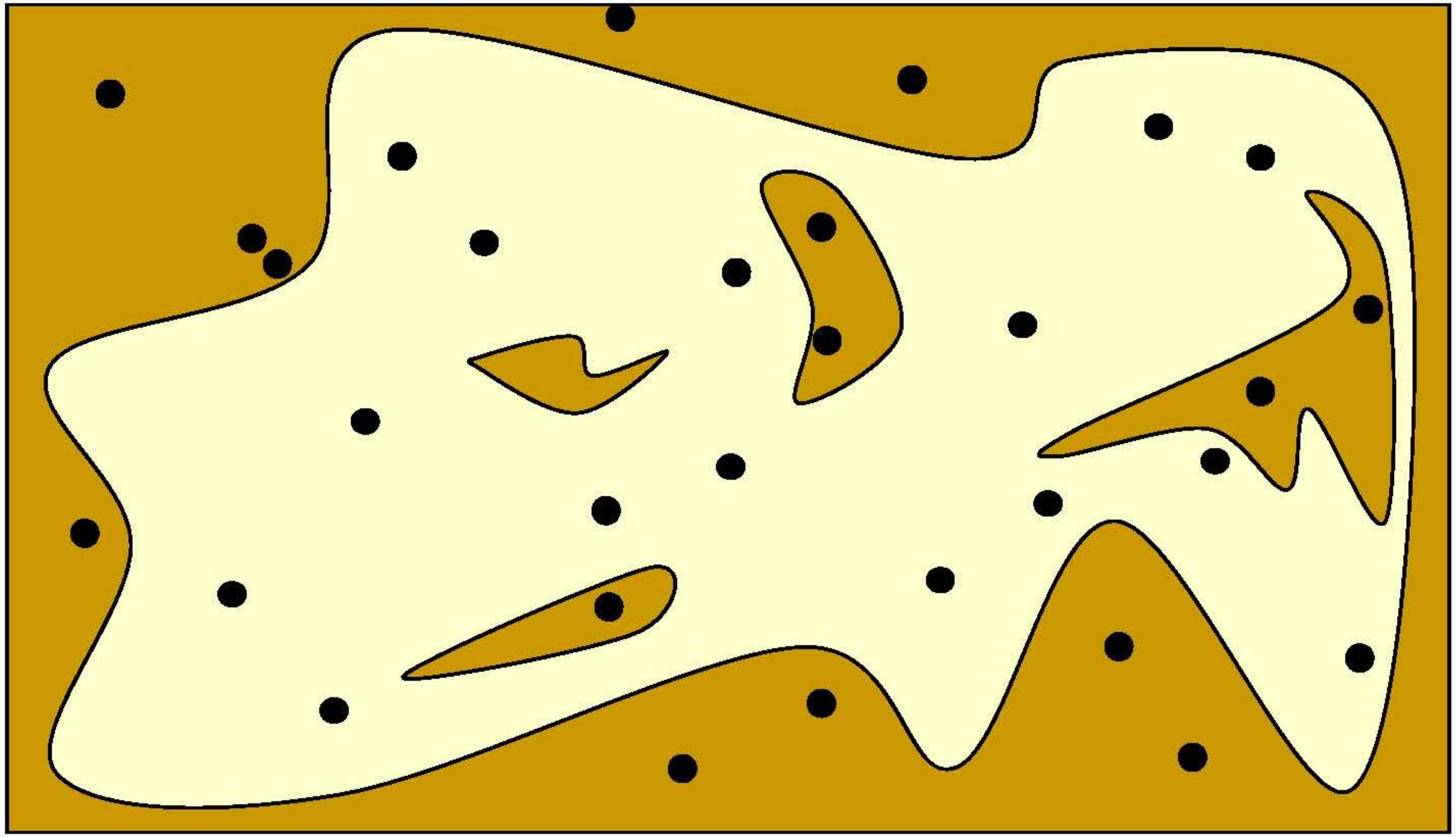


Image Credit: P. Abbeel

Probabilistic Roadmaps (PRM)

Sampled configurations are tested for collision

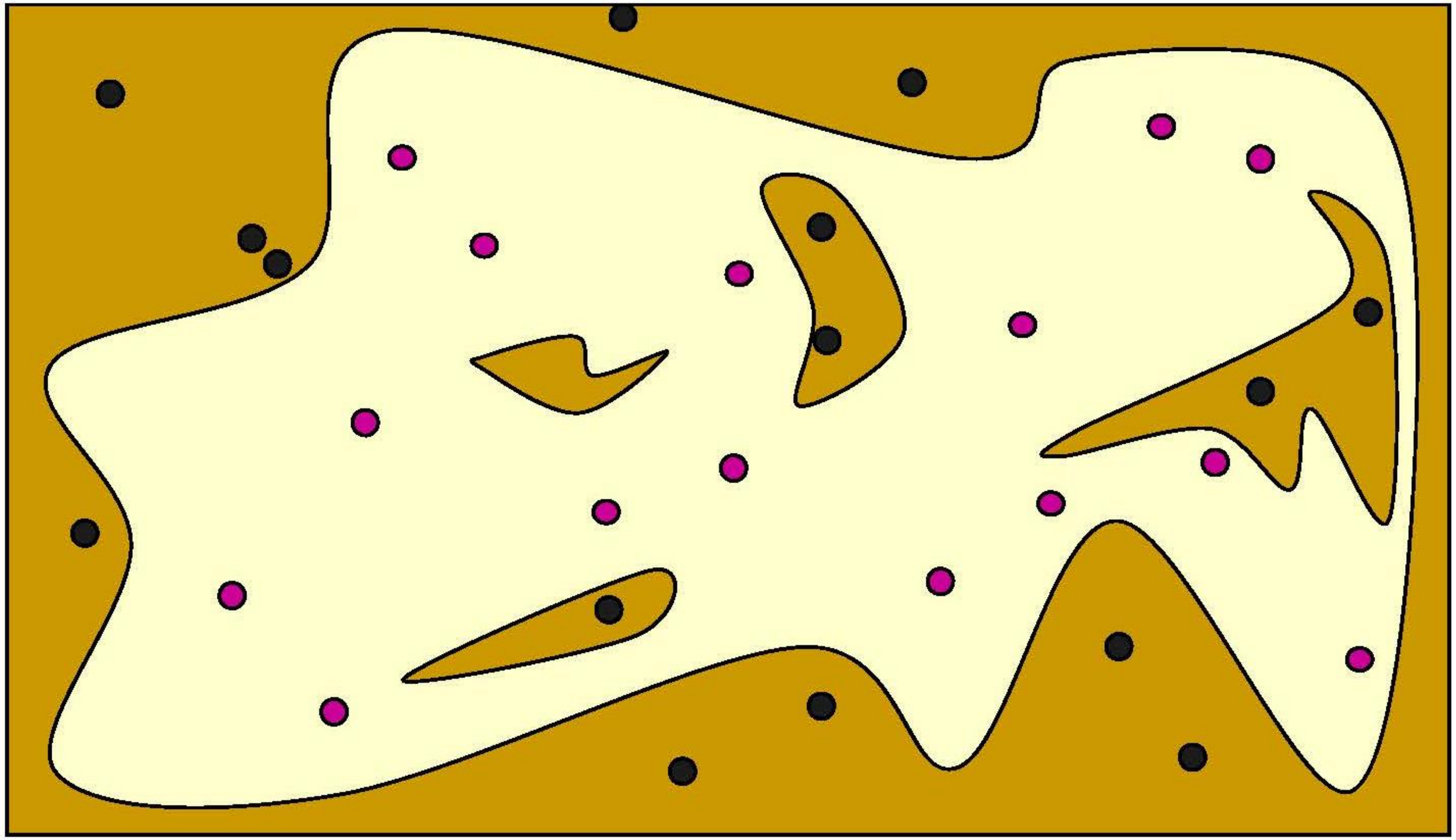


Image Credit: P. Abbeel

Probabilistic Roadmaps (PRM)

The collision-free configurations are retained as milestones

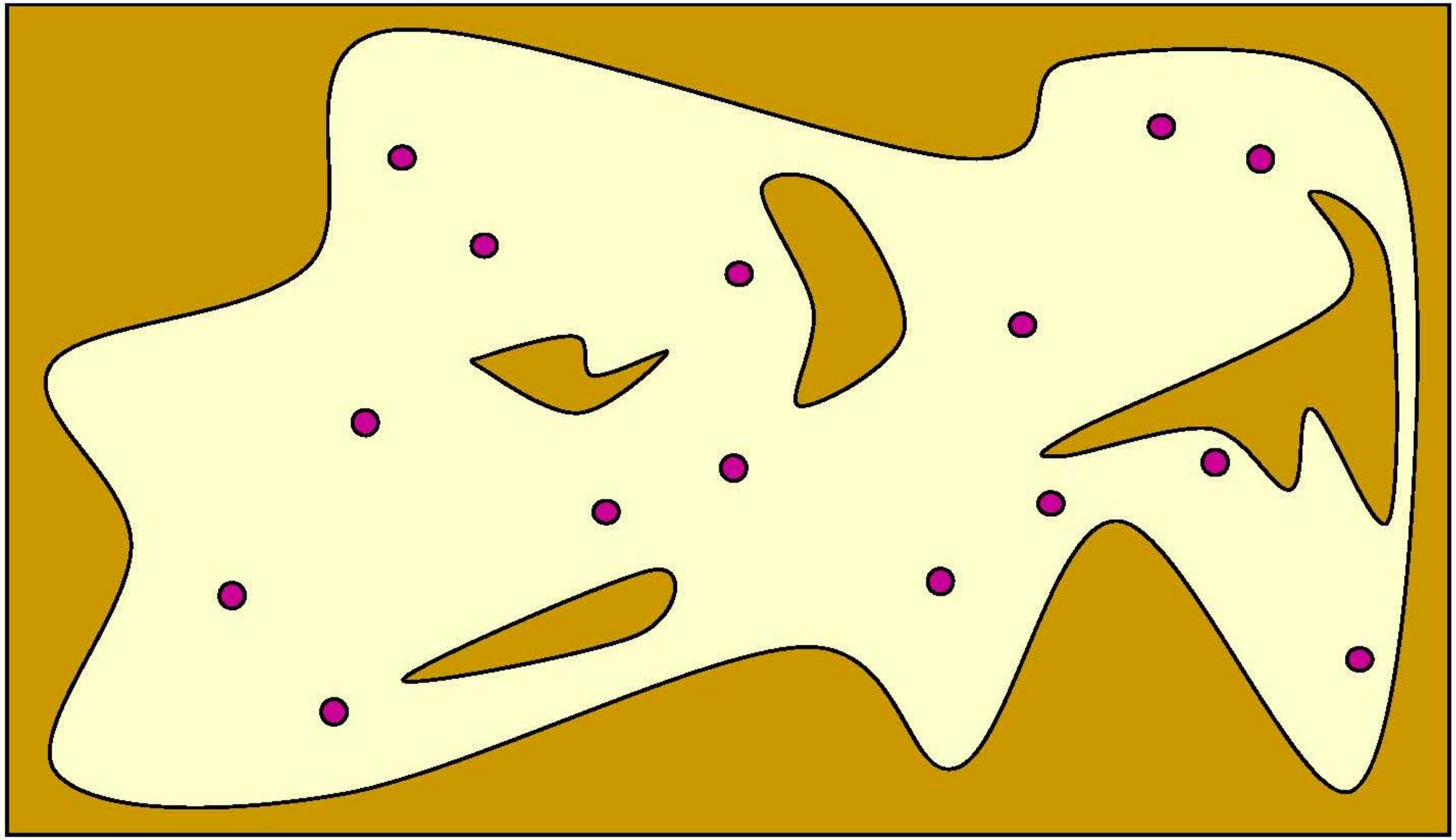


Image Credit: P. Abbeel

Probabilistic Roadmaps (PRM)

Each milestone is linked by straight paths to its nearest neighbors

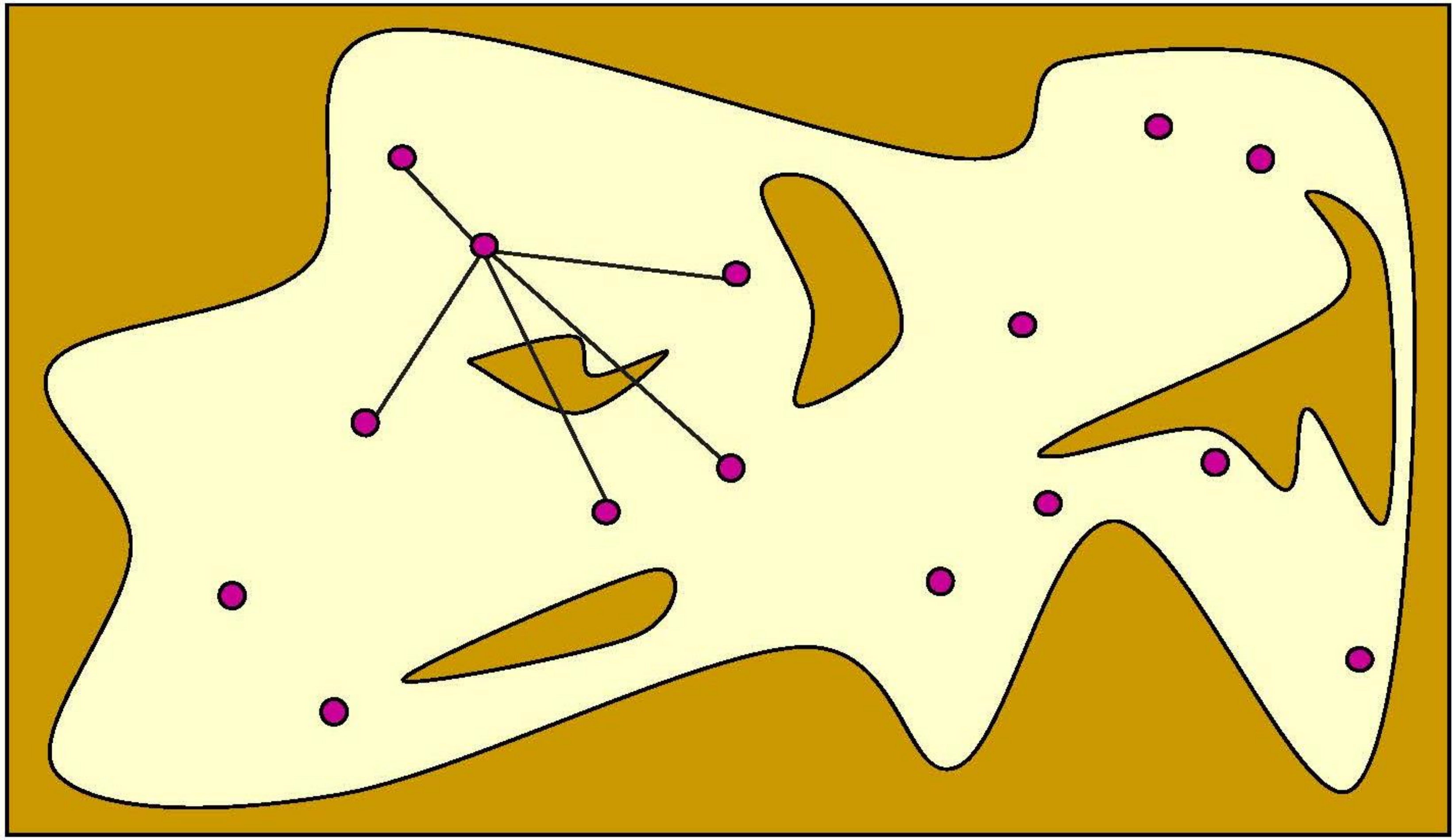


Image Credit: P. Abbeel

Probabilistic Roadmaps (PRM)

Each milestone is linked by straight paths to its nearest neighbors

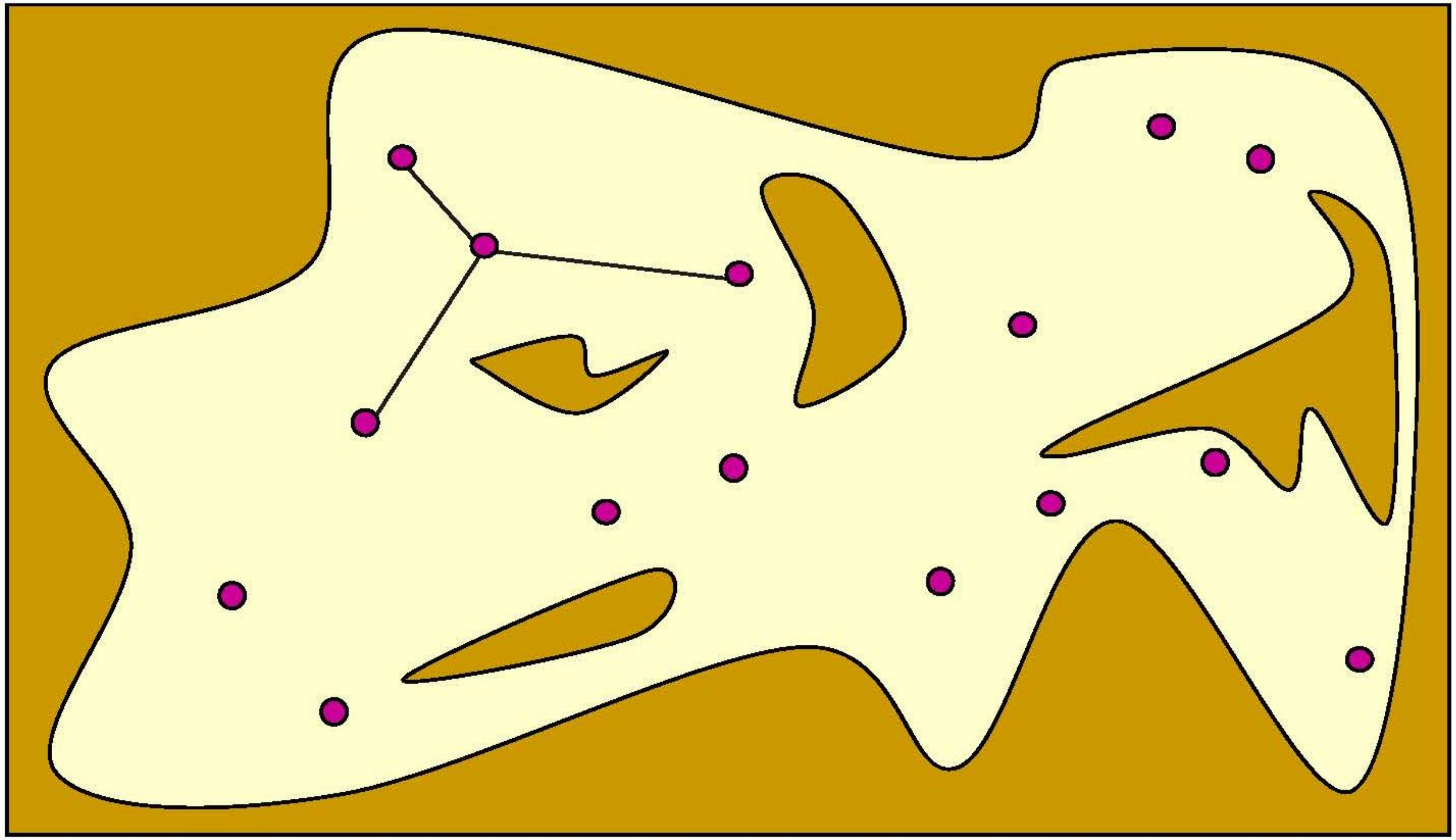


Image Credit: P. Abbeel

Probabilistic Roadmaps (PRM)

The collision-free links are retained as local paths to form the PRM

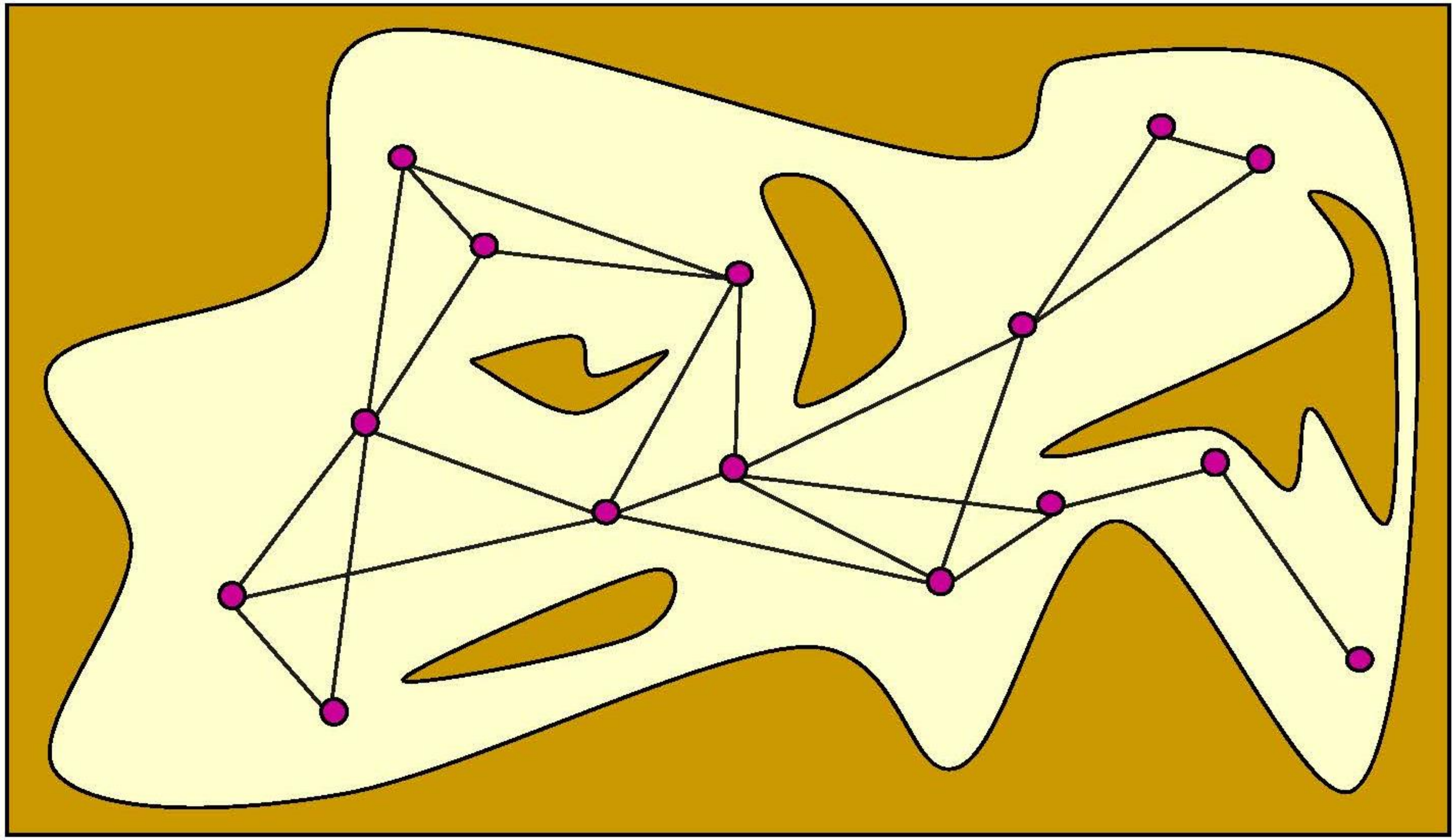


Image Credit: P. Abbeel

Probabilistic Roadmaps (PRM)

The start and goal configurations are included as milestones

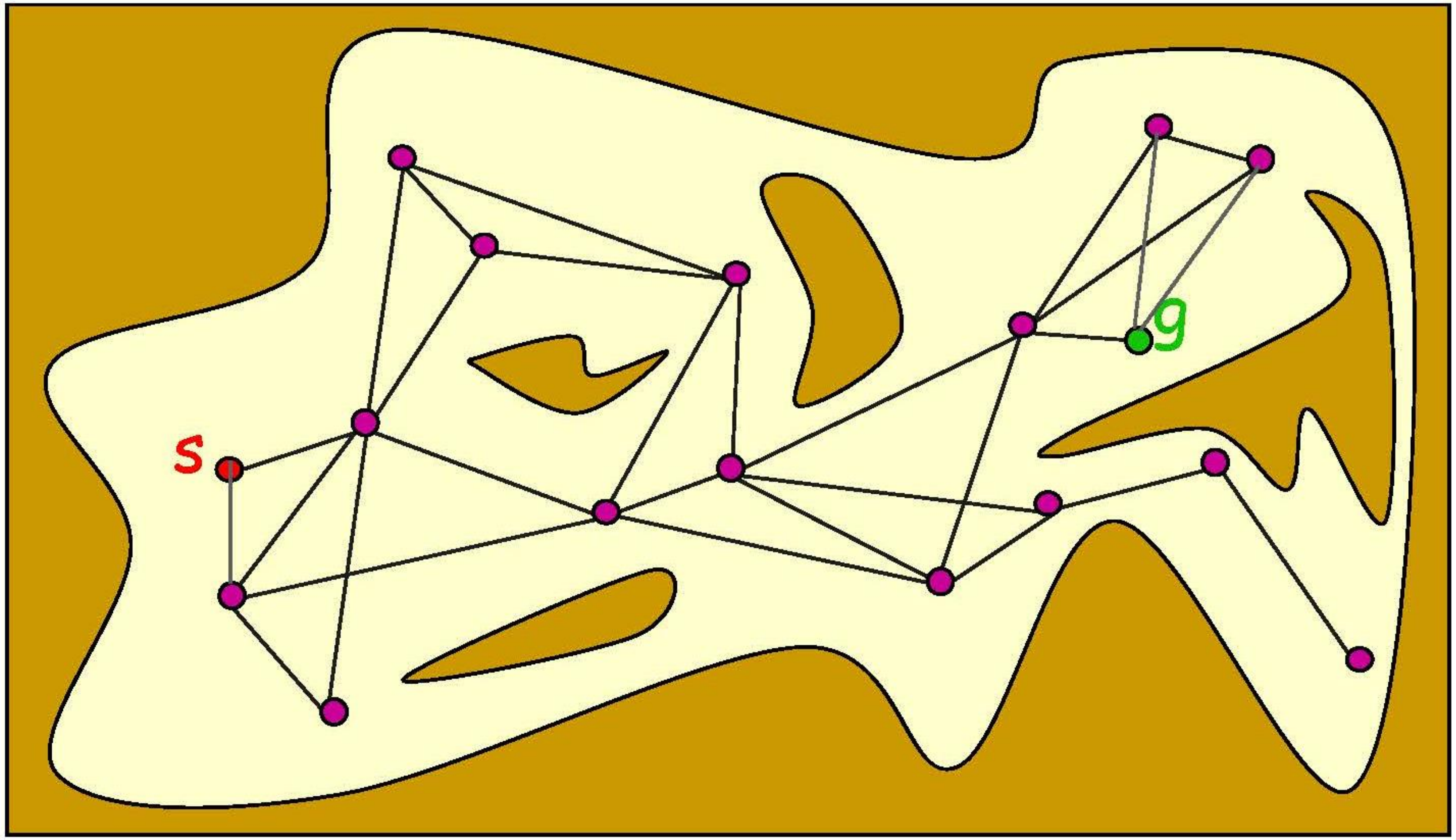


Image Credit: P. Abbeel

Probabilistic Roadmaps (PRM)

- Initialize starting point and goal point (set)
- Randomly sample points in the space
- Connect nearby points *if they are reachable*
- Find a path from start to goal, often with A*
- Generally control inputs are required for execution
- System cannot always execute an arbitrary path

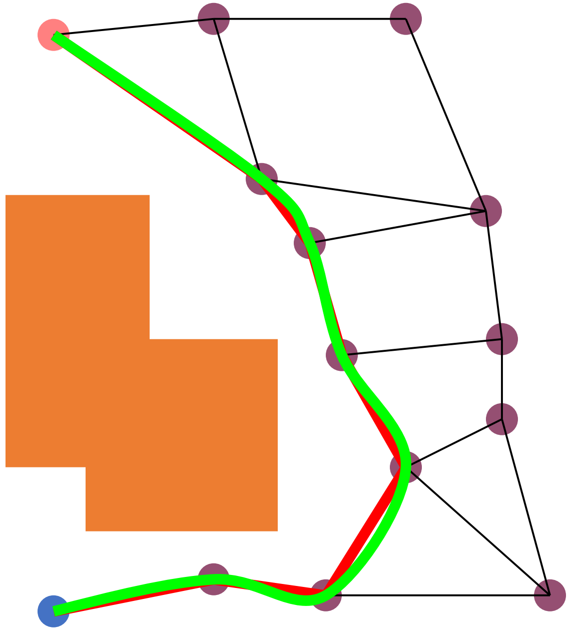


Image Credit: Aisha Walcott, Nathan Ickes, Stanislav Funiak

Rapidly exploring Random Tree (RRT)

Build tree by generating next states through the dynamics by randomly selecting inputs

Rapidly exploring Random Tree (RRT)

Build tree by generating next states through the dynamics by randomly selecting inputs

Image Credit: Wikipedia, RRT

Rapidly exploring Random Tree (RRT)

Build tree by generating next states through the dynamics by randomly selecting inputs

Generate_RRT($x_{init}, K, \Delta t$)

$\mathcal{T}$.init(x_{init})

for k = 1 to K

x_{rand} ← RANDOM_STATE()

x_{near} ← NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$)

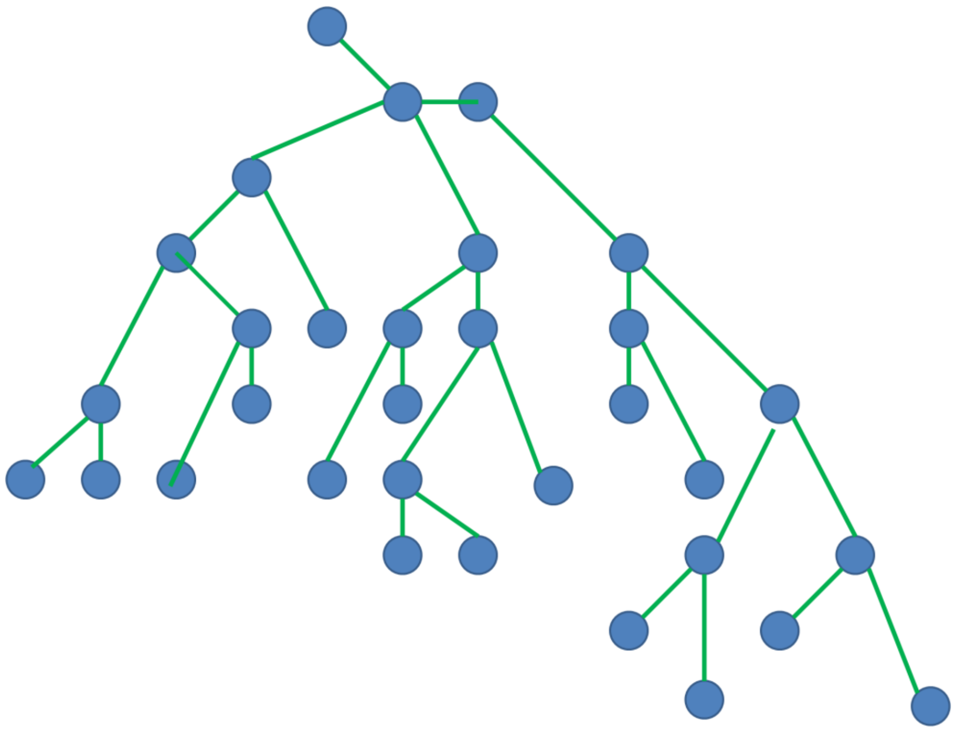
u ← SELECT_INPUT(x_{rand}, x_{near})

x_{new} ← NEW_STATE($x_{near}, u, \Delta t$)

$\mathcal{T}$.add_vertex(x_{new})

$\mathcal{T}$.add_edge(x_{near}, x_{new}, u)

Return $\mathcal{T}$



Rapidly exploring Random Tree (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*

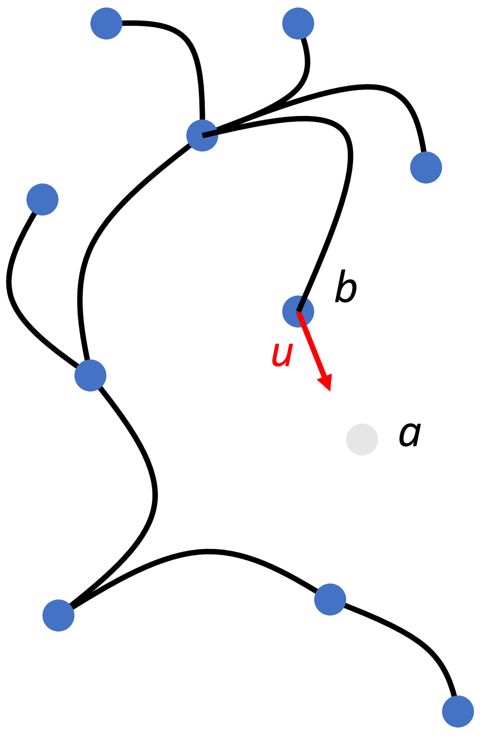


Image Credit: Aisha Walcott, Nathan Ickes, Stanislav Funiak

Rapidly exploring Random Tree (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
- Apply control inputs u for time Δt, so robot reaches c
- If no collisions occur in getting from a to c, add c to RRT and record u with new edge

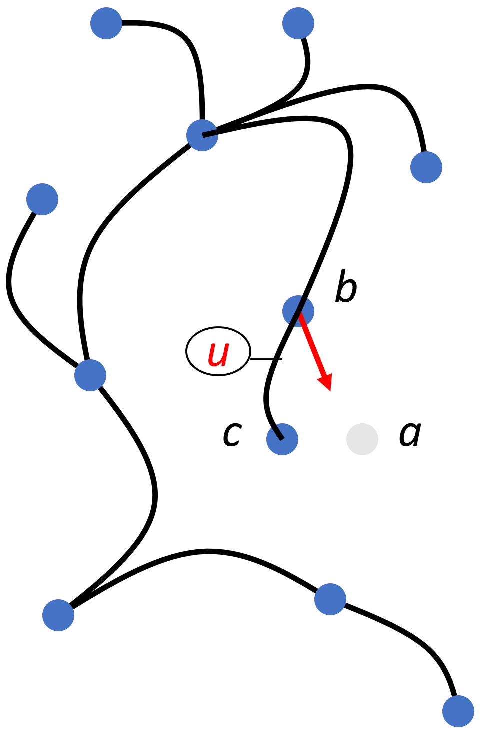


Image Credit: Aisha Walcott, Nathan Ickes, Stanislav Funiak

RRT

Generate_RRT($x_{init}, K, \Delta t$)
$\mathcal{T}$.init(x_{init})
For k = 1 to K
    x_{rand} ← RANDOM_STATE()
    x_{near} ← NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$)
    u ← SELECT_INPUT(x_{rand}, x_{near})
    x_{new} ← NEW_STATE($x_{near}, u, \Delta t$)
    $\mathcal{T}$.add_vertex(x_{new})
    $\mathcal{T}$.add_edge(x_{near}, x_{new}, u)
Return $\mathcal{T}$

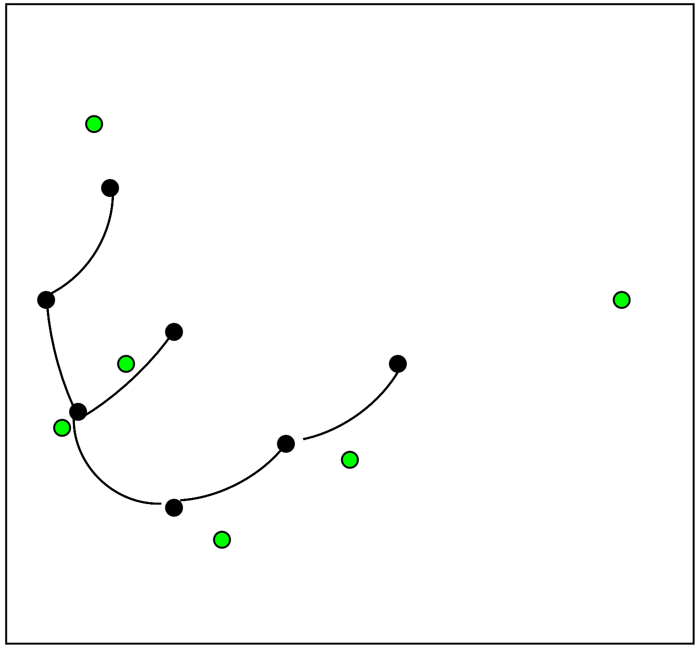


Image Credit: Aisha Walcott, Nathan Ickes, Stanislav Funiak

RRT

Once stopping condition is met, traverse back through the graph to uncover your trajectory and input sequence.

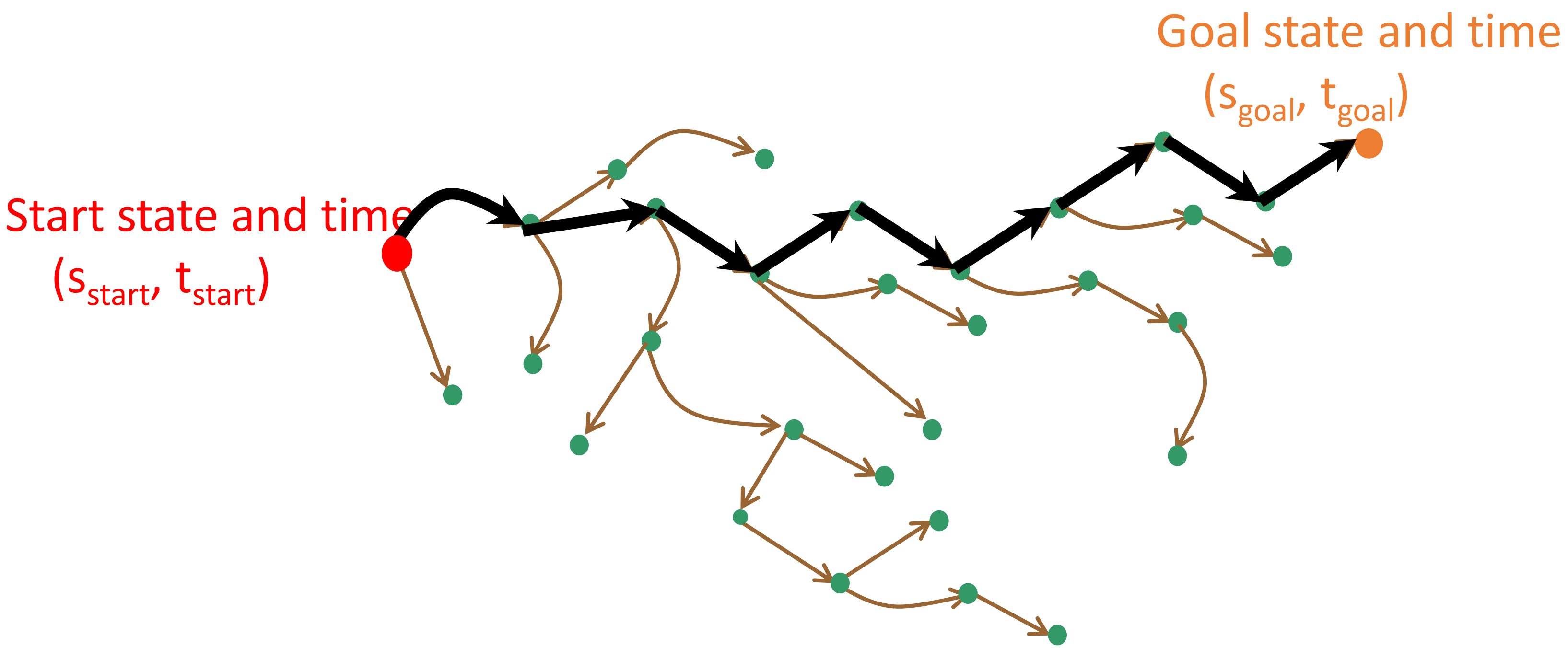
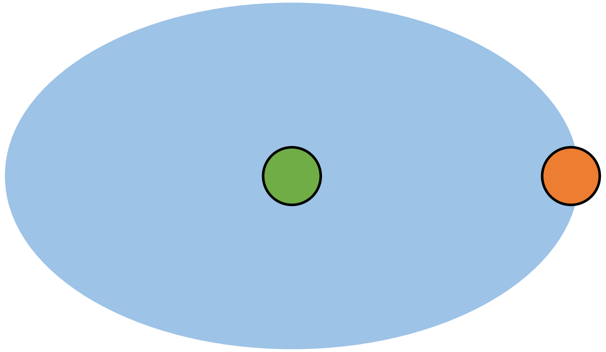


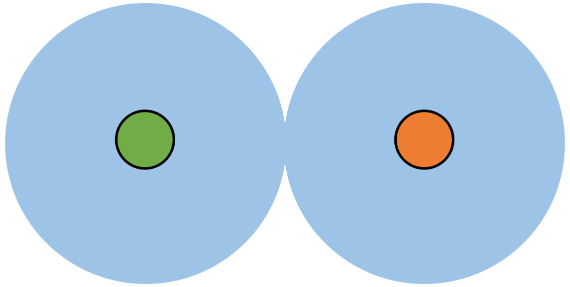
Image Credit: Aisha Walcott, Nathan Ickes, Stanislav Funiak

Bidirectional Planning

- Volume of unidirectional RRT

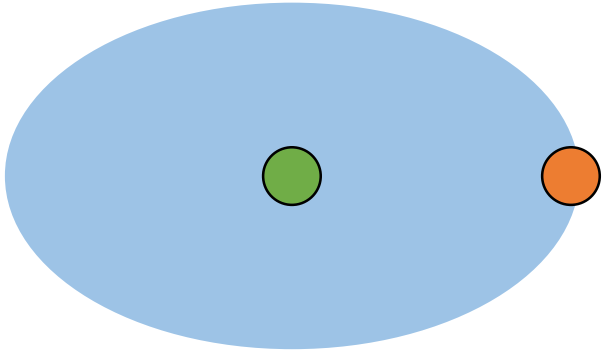


- Volume of bidirectional RRT



Bidirectional Planning

- Volume of unidirectional RRT



- Volume of bidirectional RRT

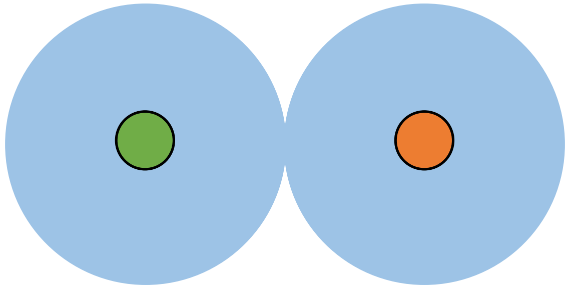
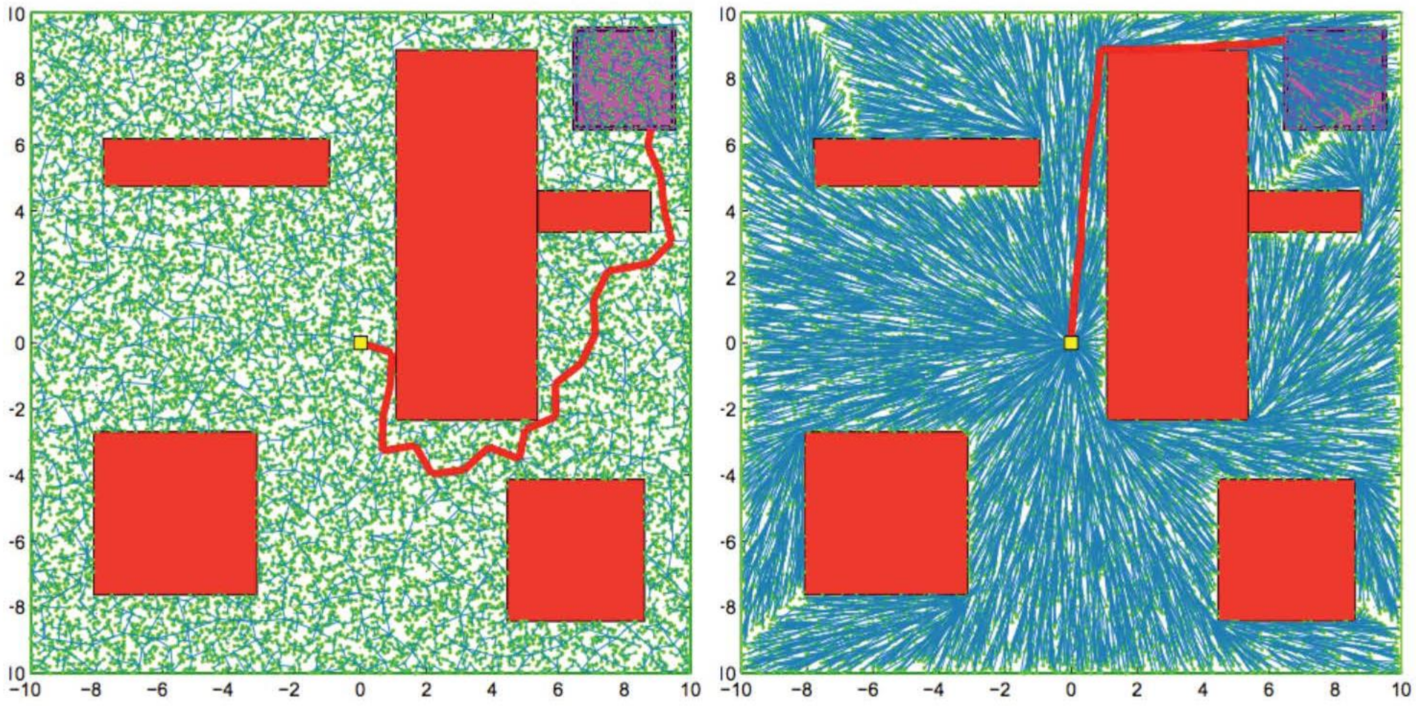


Image Credit: Aisha Walcott, Nathan Ickes, Stanislav Funiak

Asympototic Guarantees with RRT*

Swap new point in as parent for nearby vertices who can be reached along shorter path through new point than through their original (current) parent. Can be shown to be asymptotically optimal.

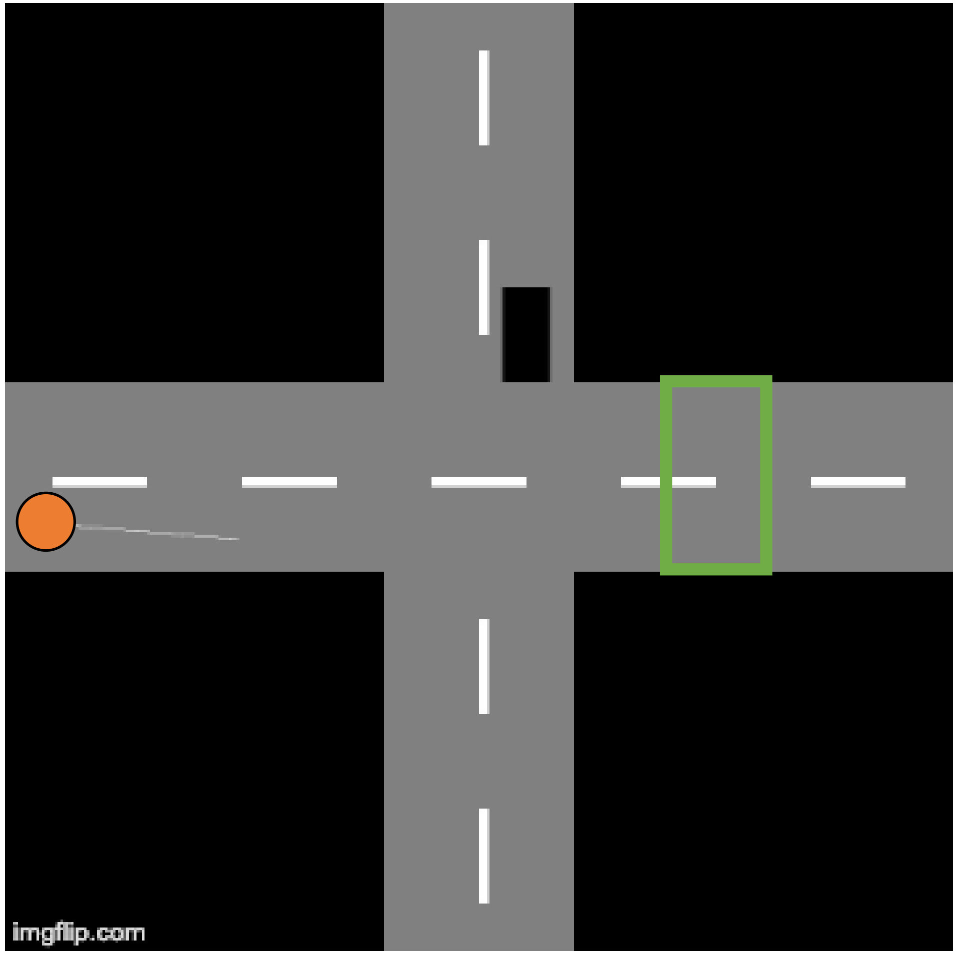
RRT RRT*



Source: Karaman and Frazzoli

A note on smoothing

- Sampling-based motion planners often give funky results
  - Typically jagged, indirect
- In practice, perform smoothing prior to execution via shortcutting or running a quick optimization



Quick Recap

- Motion planning gives use tools for finding the trajectory and sequence of inputs that will take to a goal position
- Optimization-based techniques give nice results, but are computationally difficult
- Sampling-based techniques work well in practice, but are slow and have implementation quirks
- There are many existing implementations and solvers available for you to try!