Probabilistic Methods for Kinodynamic Path Planning

Based on Past Student Lectures by:
Paul Elliott, Aisha Walcott,
Nathan Ickes and Stanislav Funiak

Lecturer:
Prof. Brian C. Williams

How do we maneuver or manipulate?

courtesy NASA JSC

courtesy NASA Ames

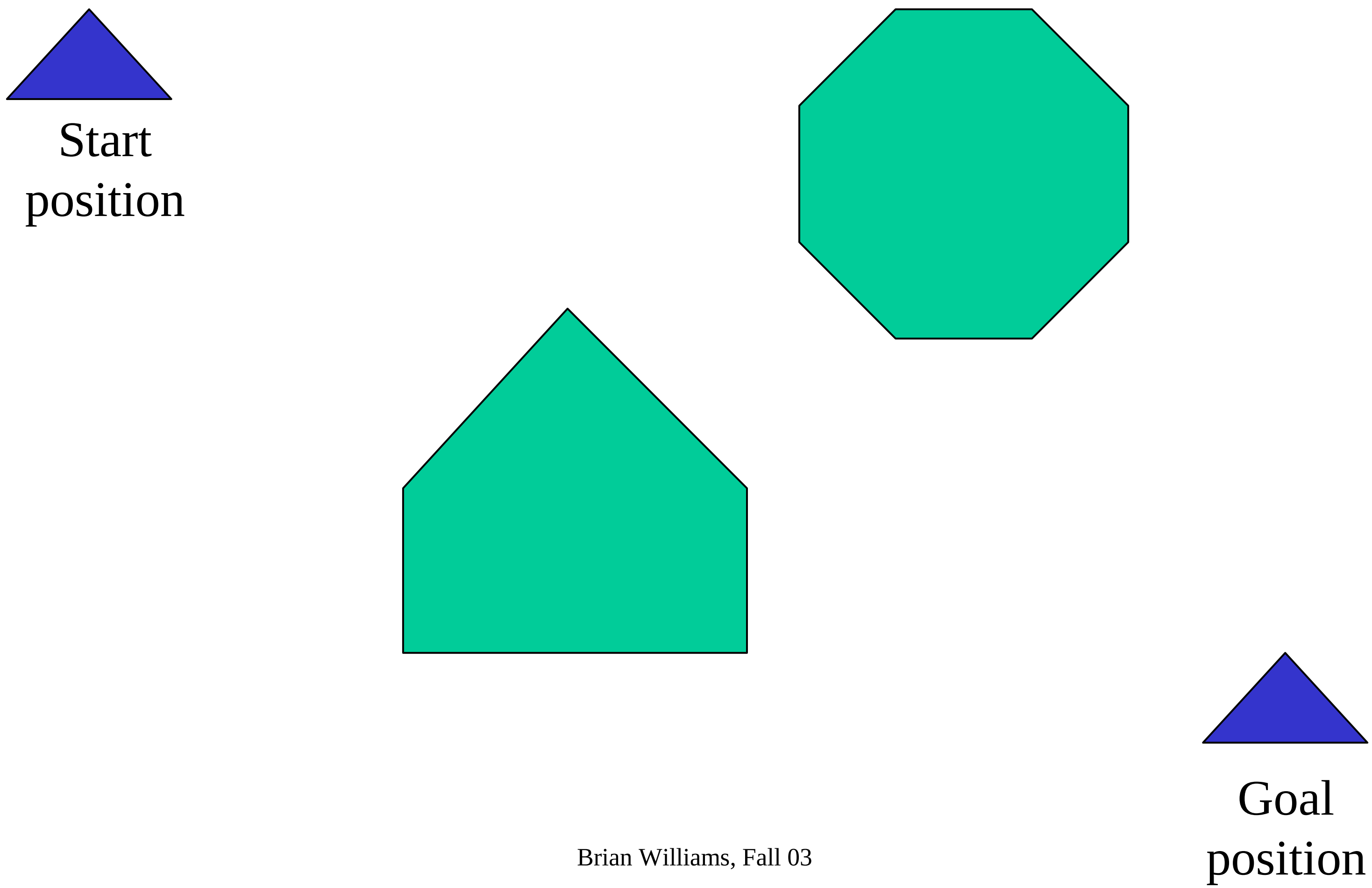
Outline

- Roadmap path planning
- Probabilistic roadmaps
- Planning in the real world
- Planning amidst moving obstacles
- RRT-based planners
- Conclusions

Outline

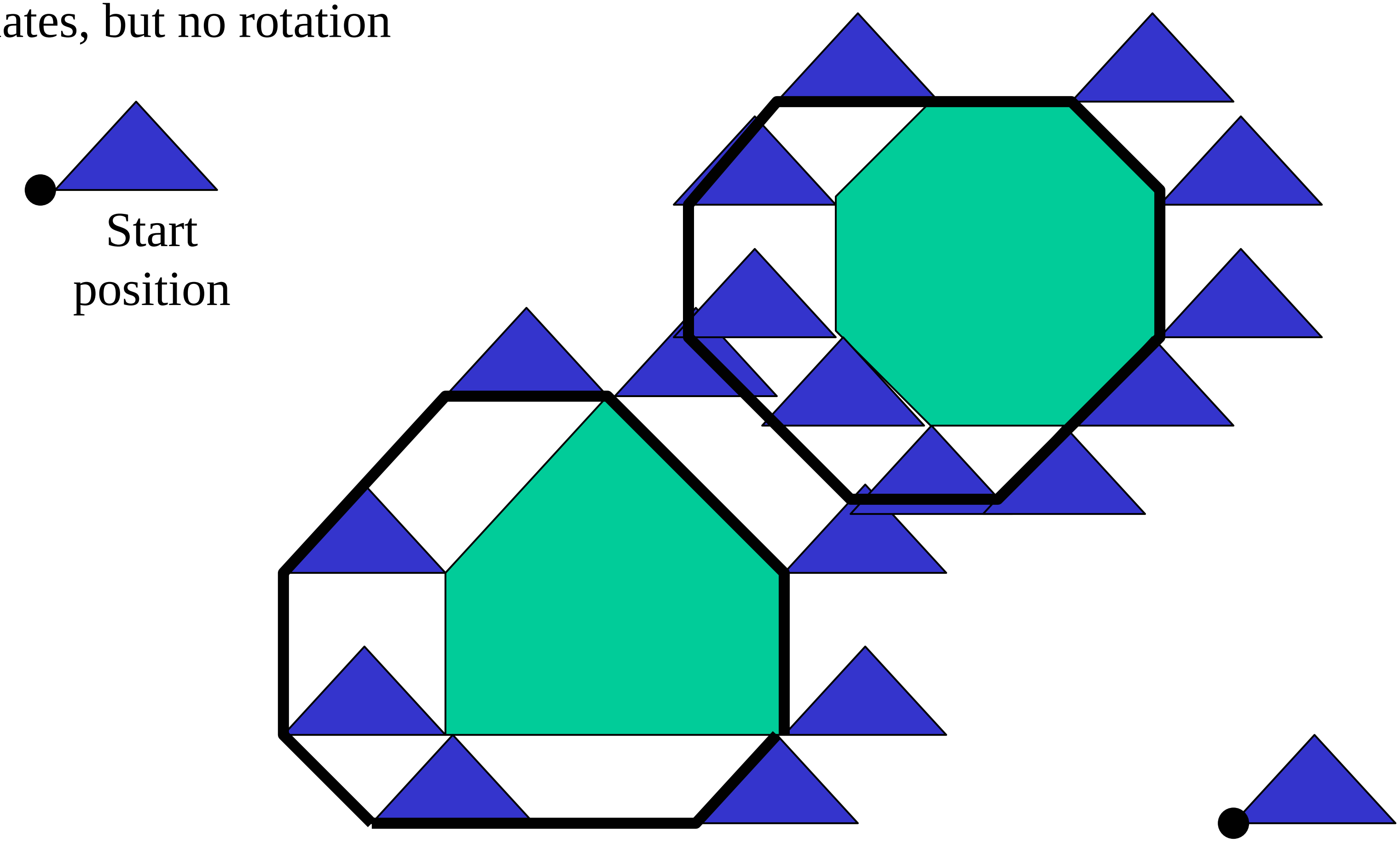
- Roadmap path planning
- Probabilistic roadmaps
- Planning in the real world
- Planning amidst moving obstacles
- RRT-based planners
- Conclusions

Path Planning through Obstacles



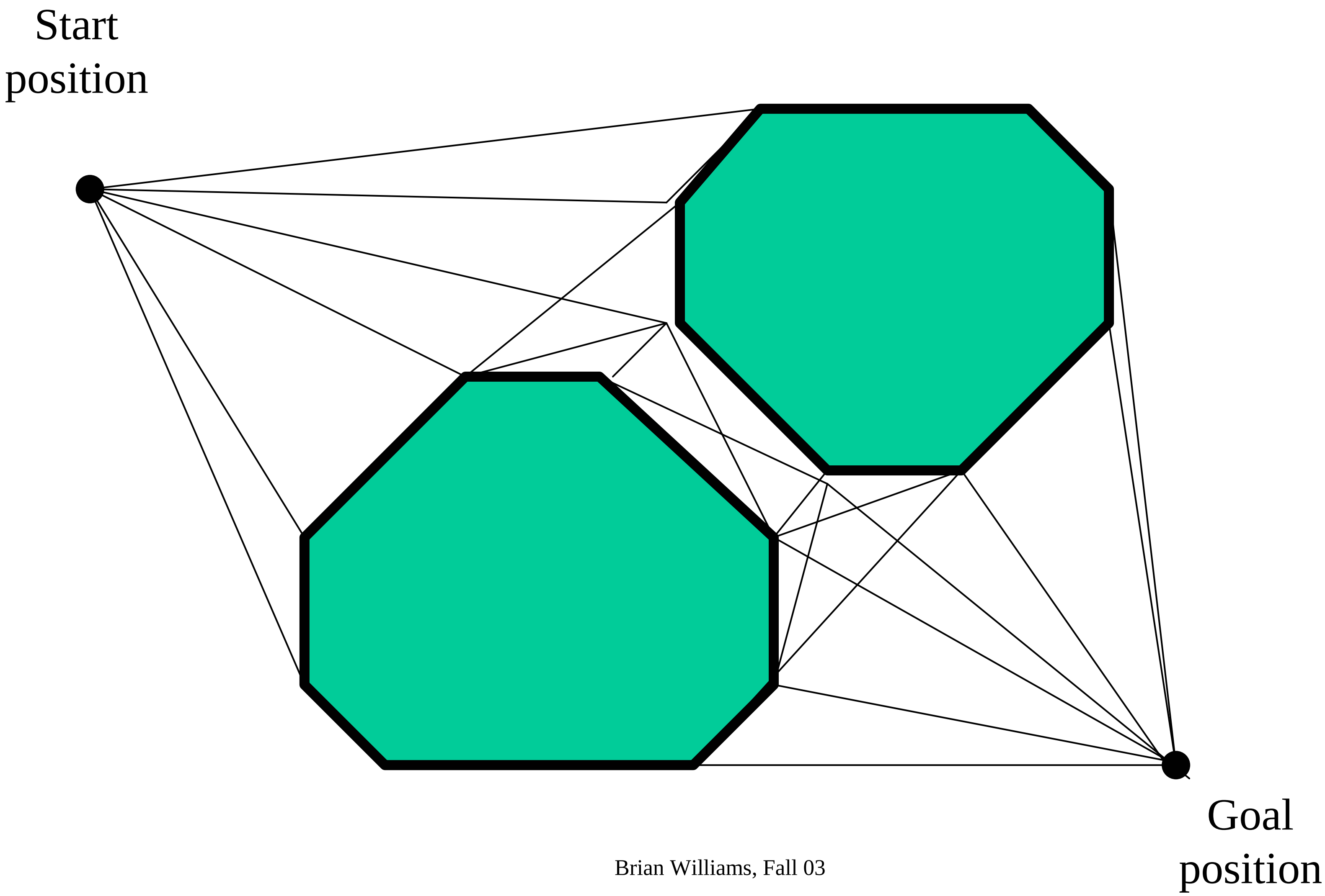
1. Create Configuration Space

Assume: Vehicle translates, but no rotation

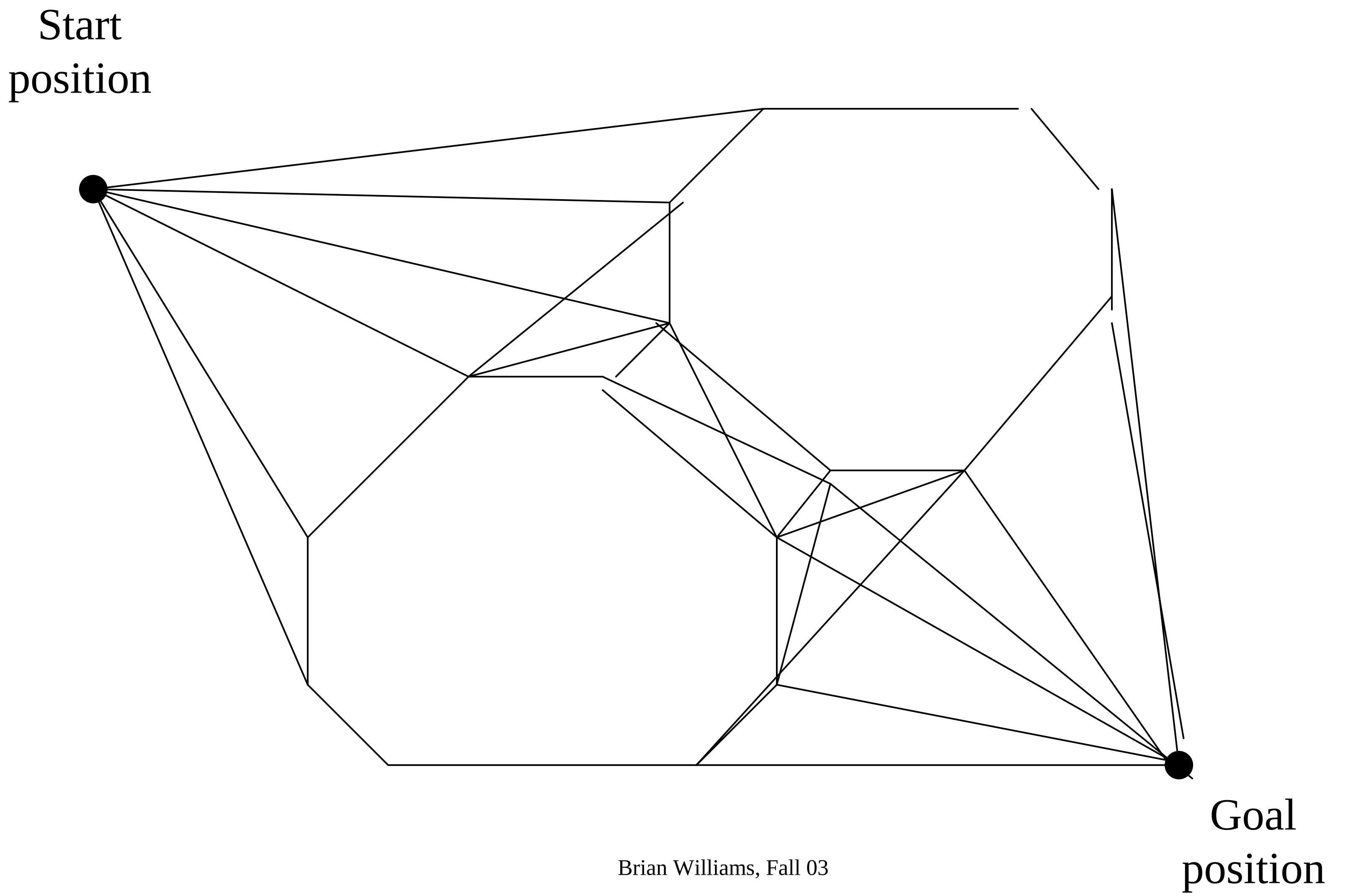


Idea: Transform to equivalent problem of navigating a point.

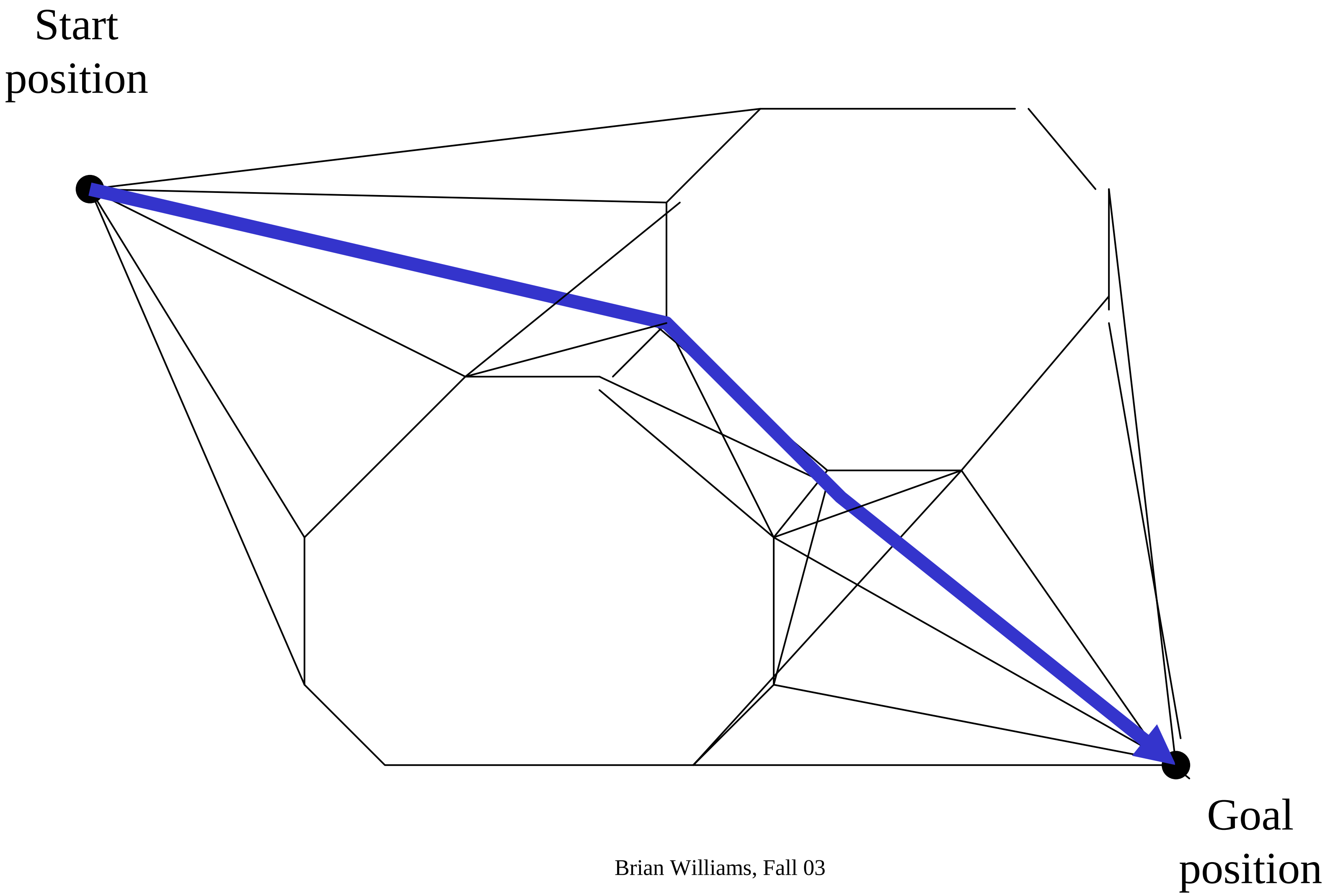
2. Map From Continuous Problem to a Roadmap: Create Visibility Graph



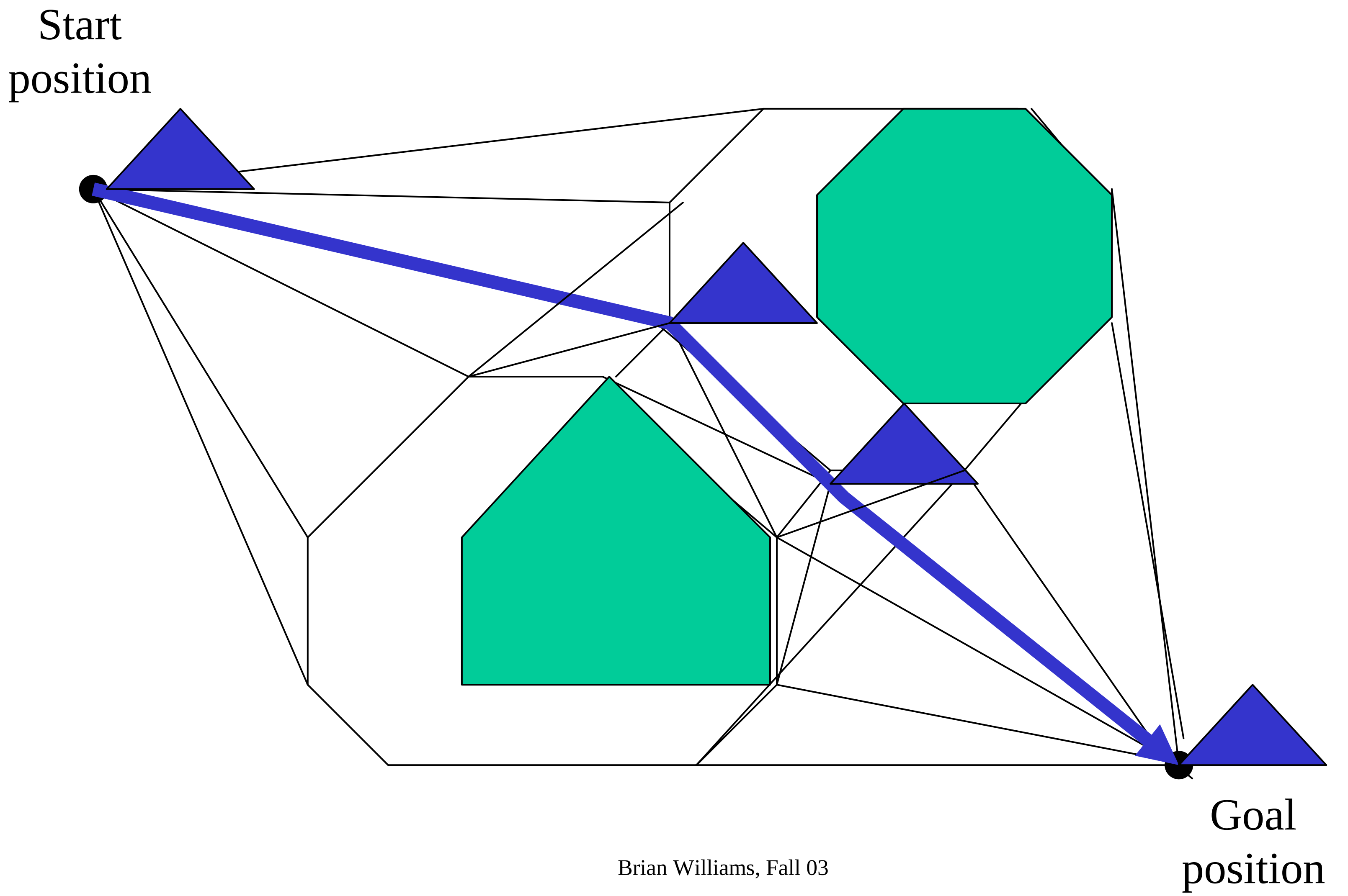
2. Map From Continuous Problem to a Roadmap: Create Visibility Graph



3. Plan Shortest Path



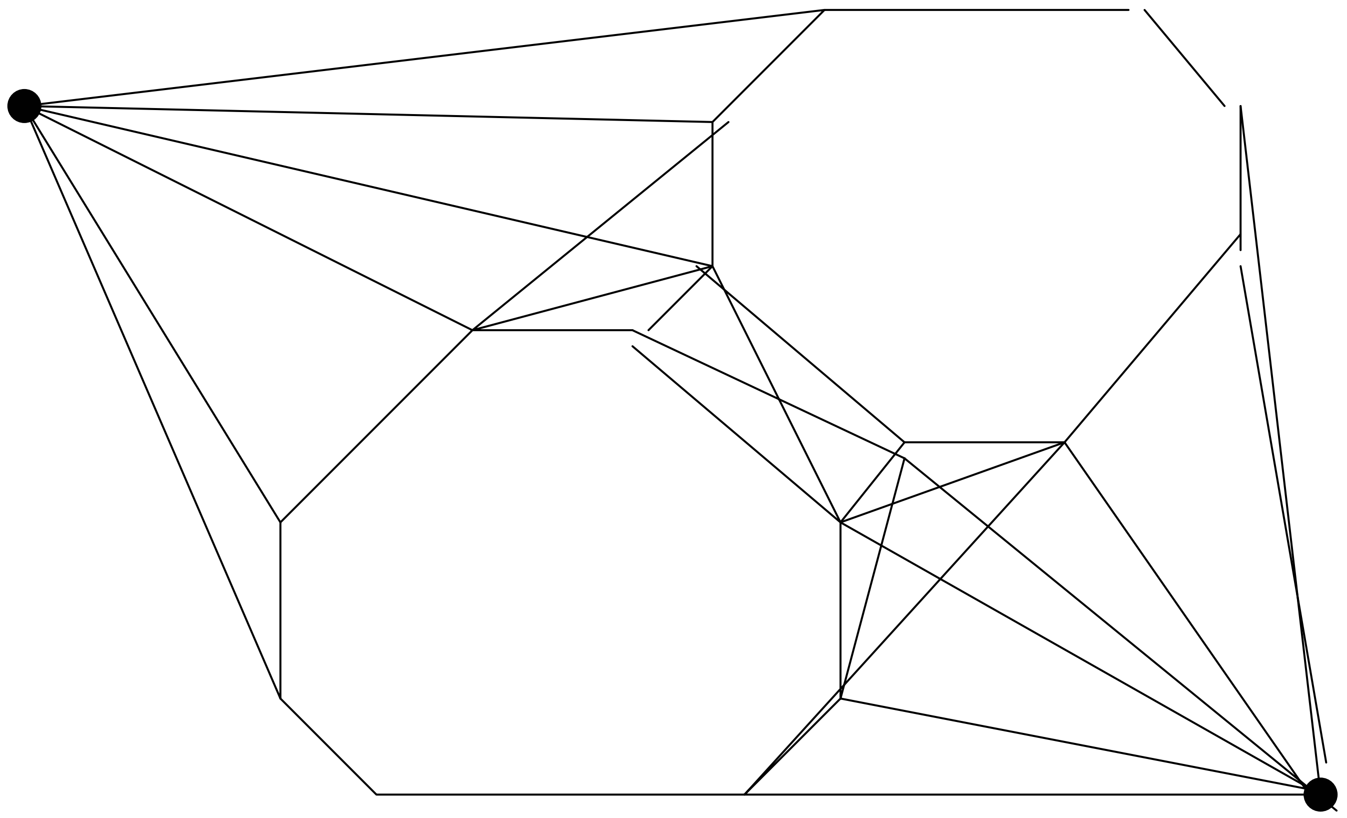
Resulting Solution



A Visibility Graph is One Kind of Roadmap

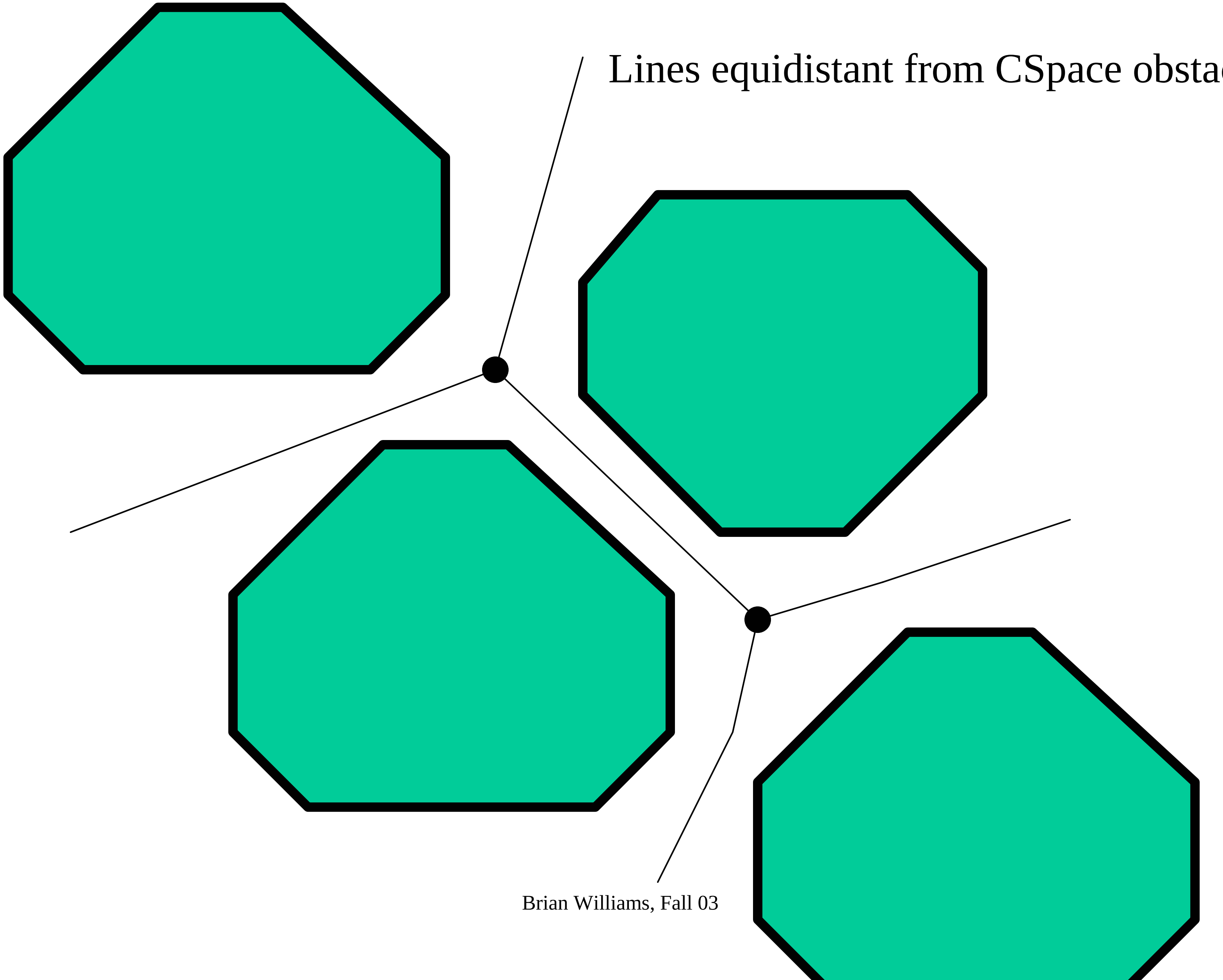
What are some other types of roadmaps?

Start position

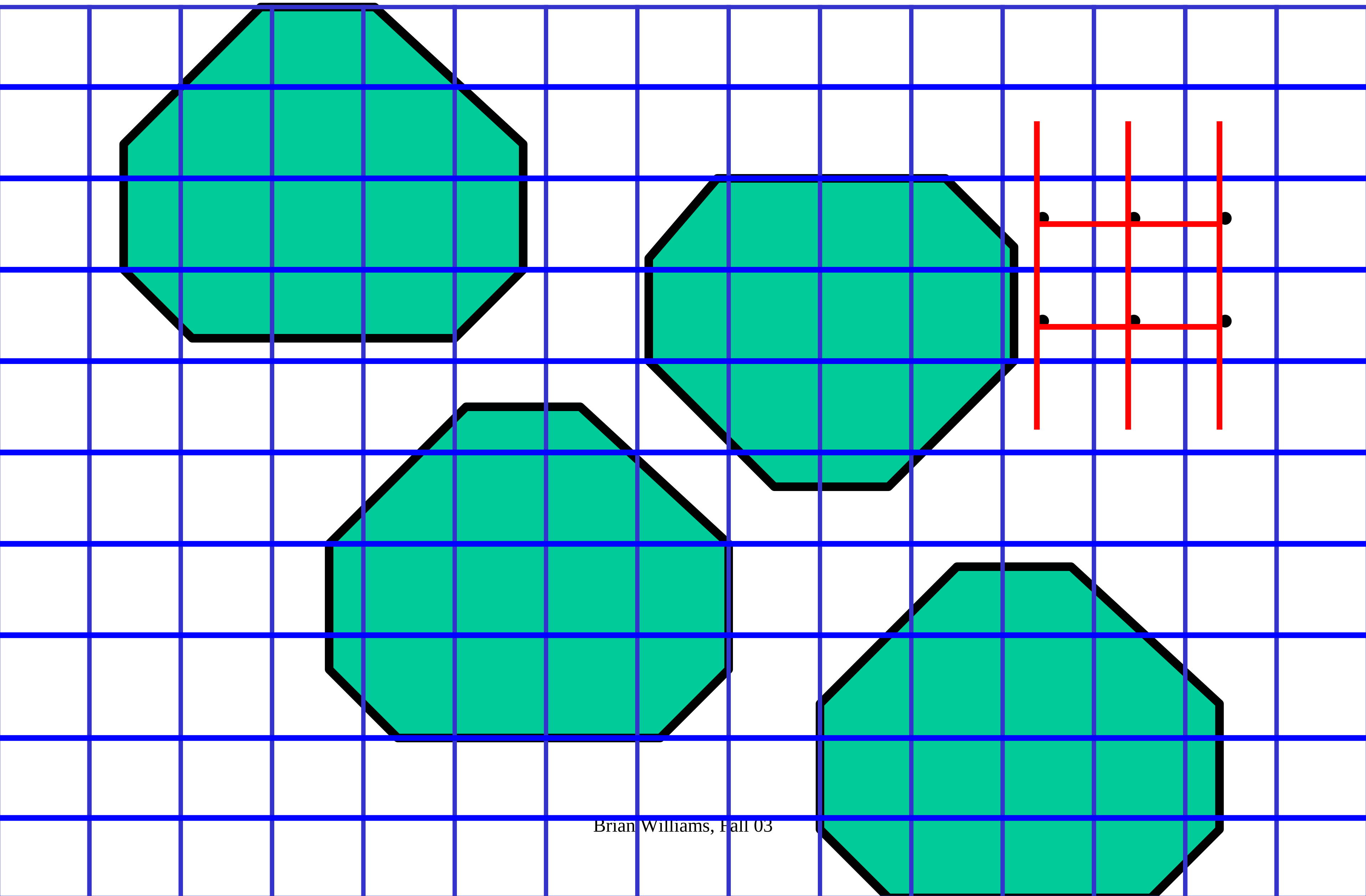


Goal position

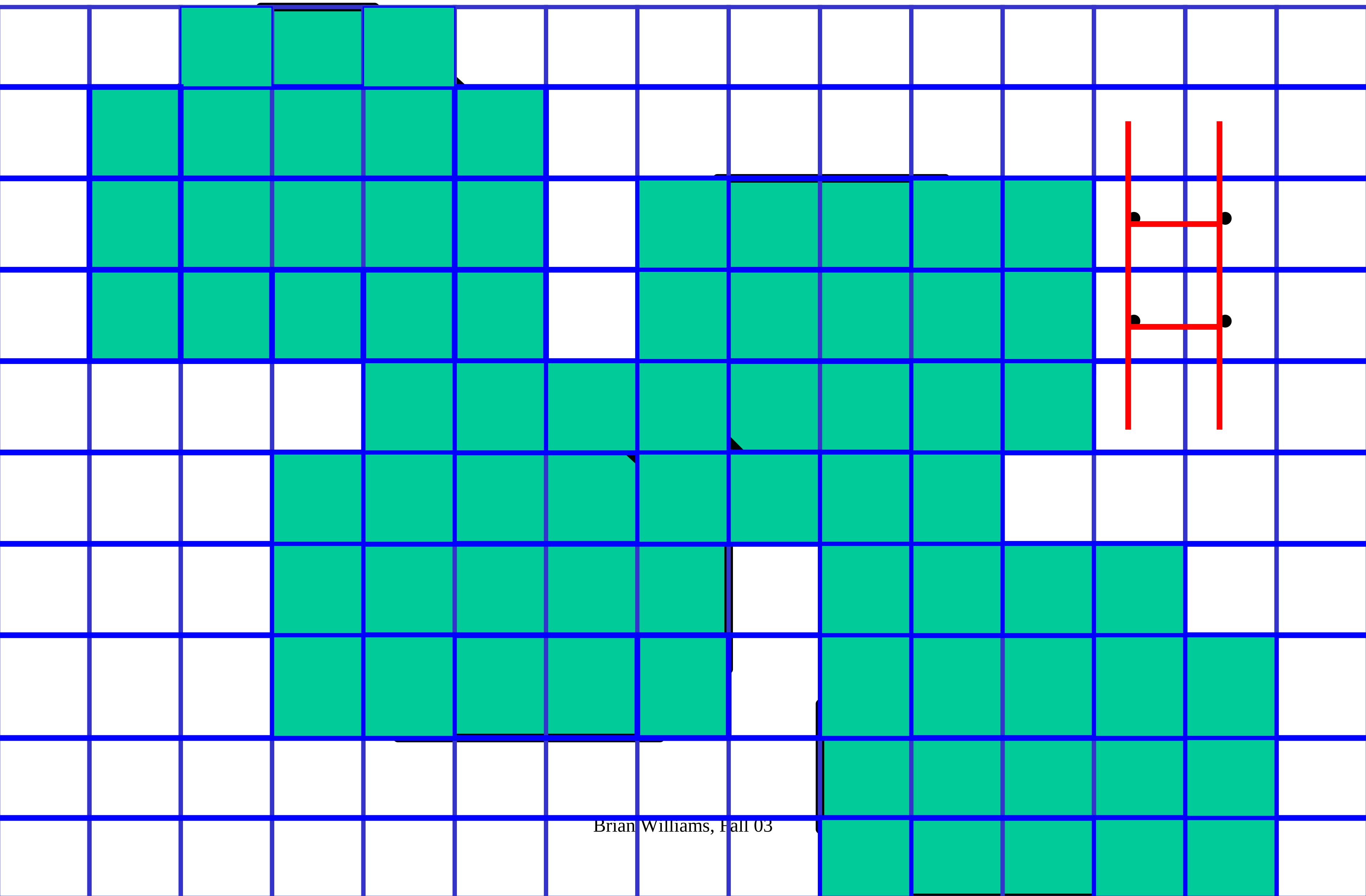
Roadmaps: Voronoi Diagrams



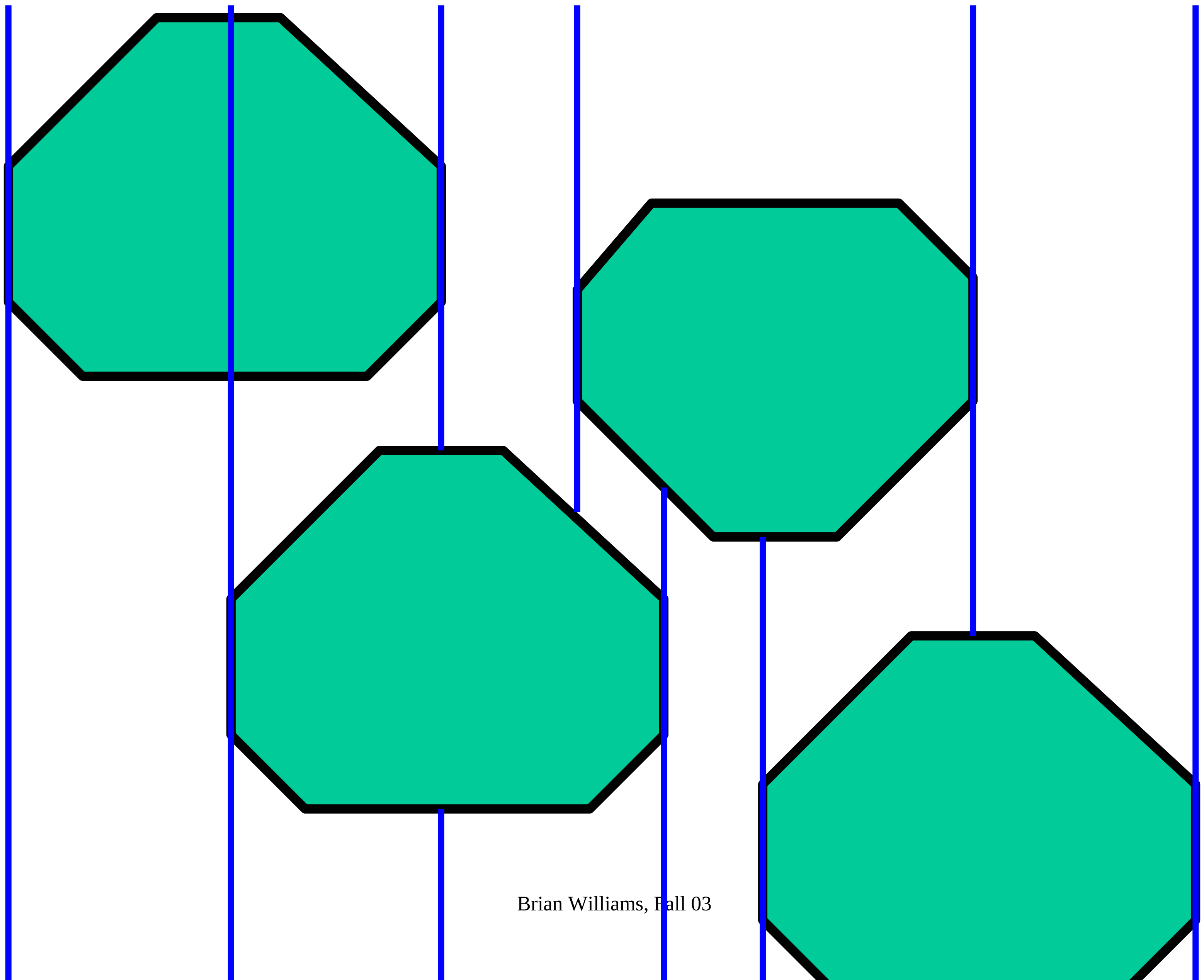
Roadmaps: Approximate Fixed Cell



Roadmaps: Approximate Fixed Cell

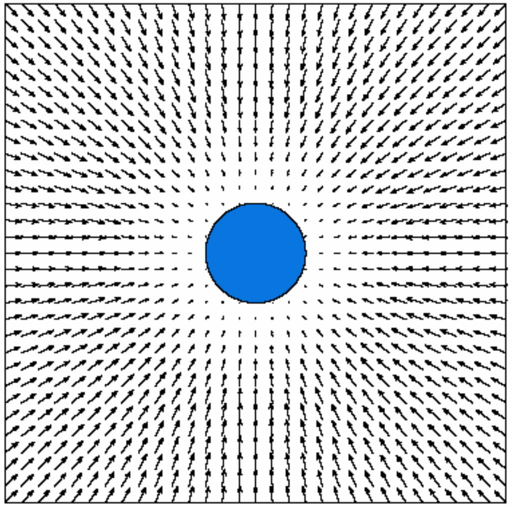


Roadmaps: Exact Cell Decomposition

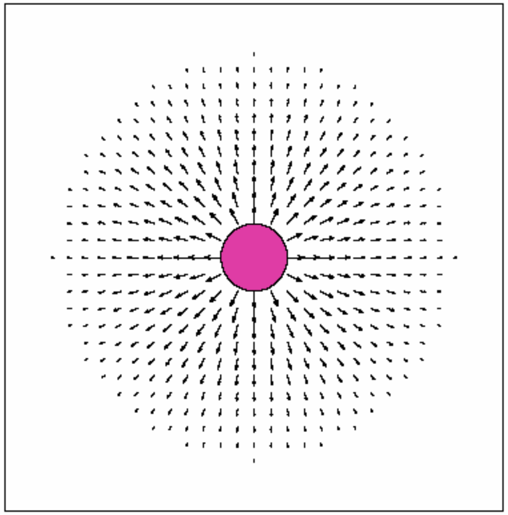


Potential Functions

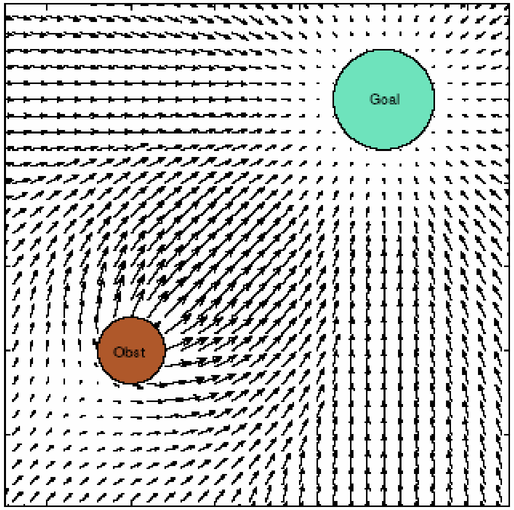
Khatib 1986
Latombe 1991
Koditschek 1998



Attractive Potential for goals



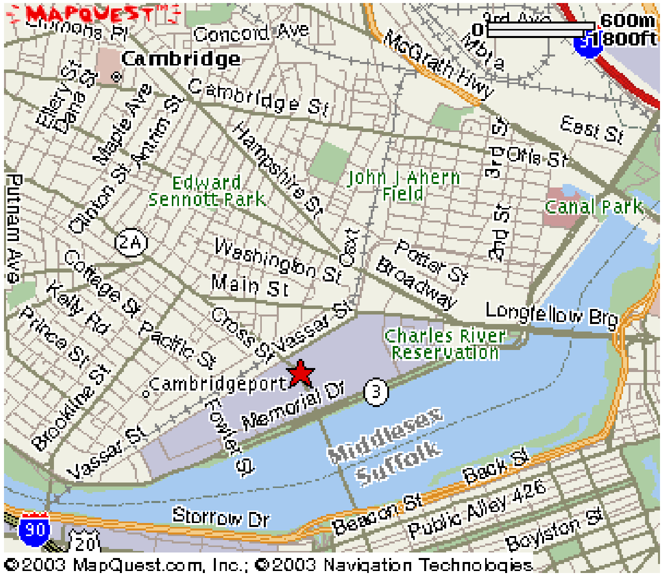
Repulsive Potential for obstacles



Combined Potential Field

Move along force: $F(x) = \nabla U_{att}(x) - \nabla U_{rep}(x)$

Exploring Roadmaps



- Shortest path
  - Dijkstra’s algorithm
  - Bellman-Ford algorithm
  - Floyd-Warshall algorithm
  - Johnson’s algorithm

- Informed search
  - Uniform cost search
  - Greedy search
  - A* search
  - Beam search
  - Hill climbing

Robonaut Teamwork: Tele-robotic

- High dimensional state space
- Controllability and dynamics
- Safety and compliance

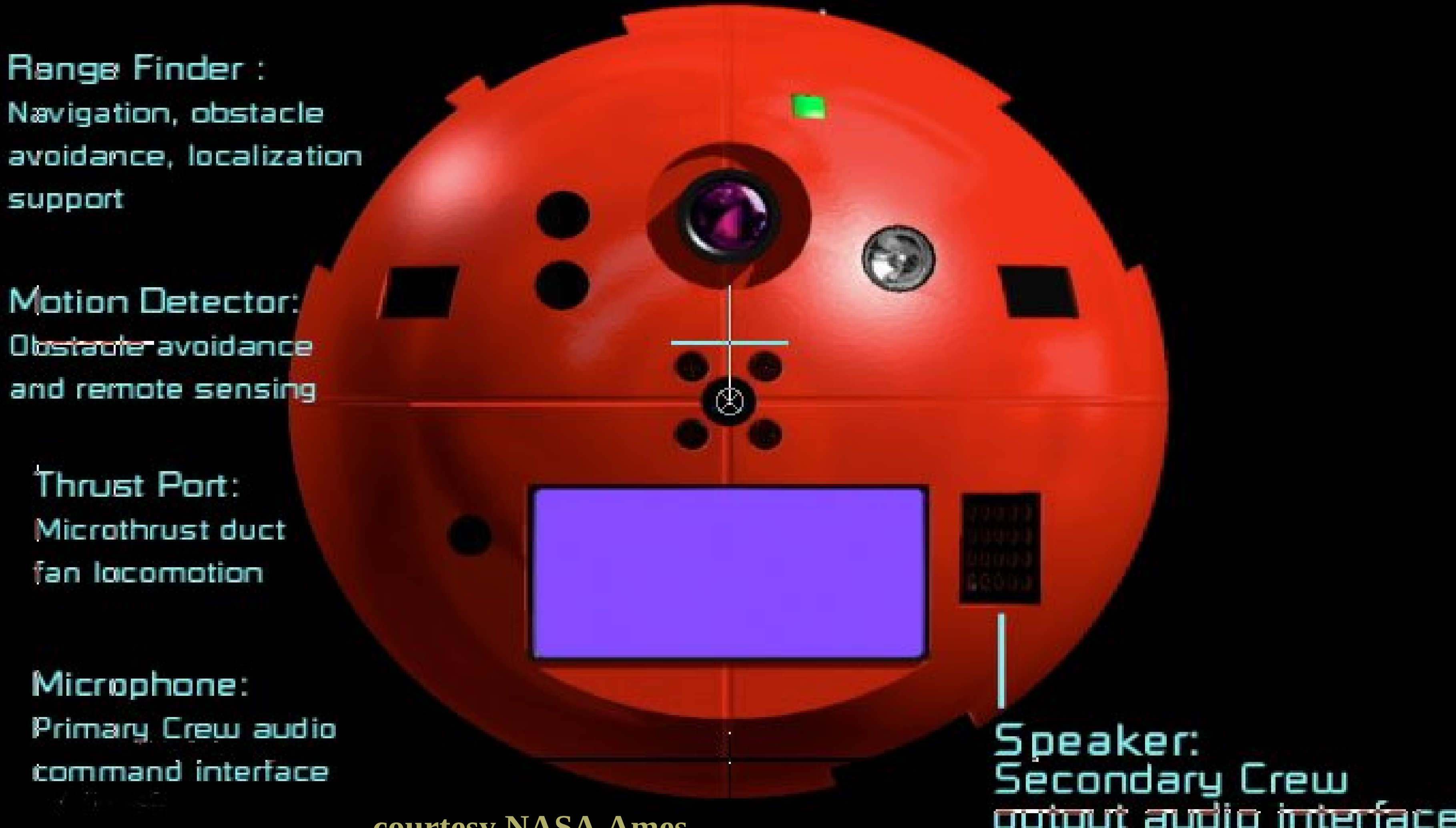
Outline

- Roadmap path planning
- Probabilistic roadmaps
- Planning in the real world
- Planning amidst moving obstacles
- RRT-based planners
- Conclusions

Applicability of Lazy Probabilistic Road Maps to Portable Satellite Assistant

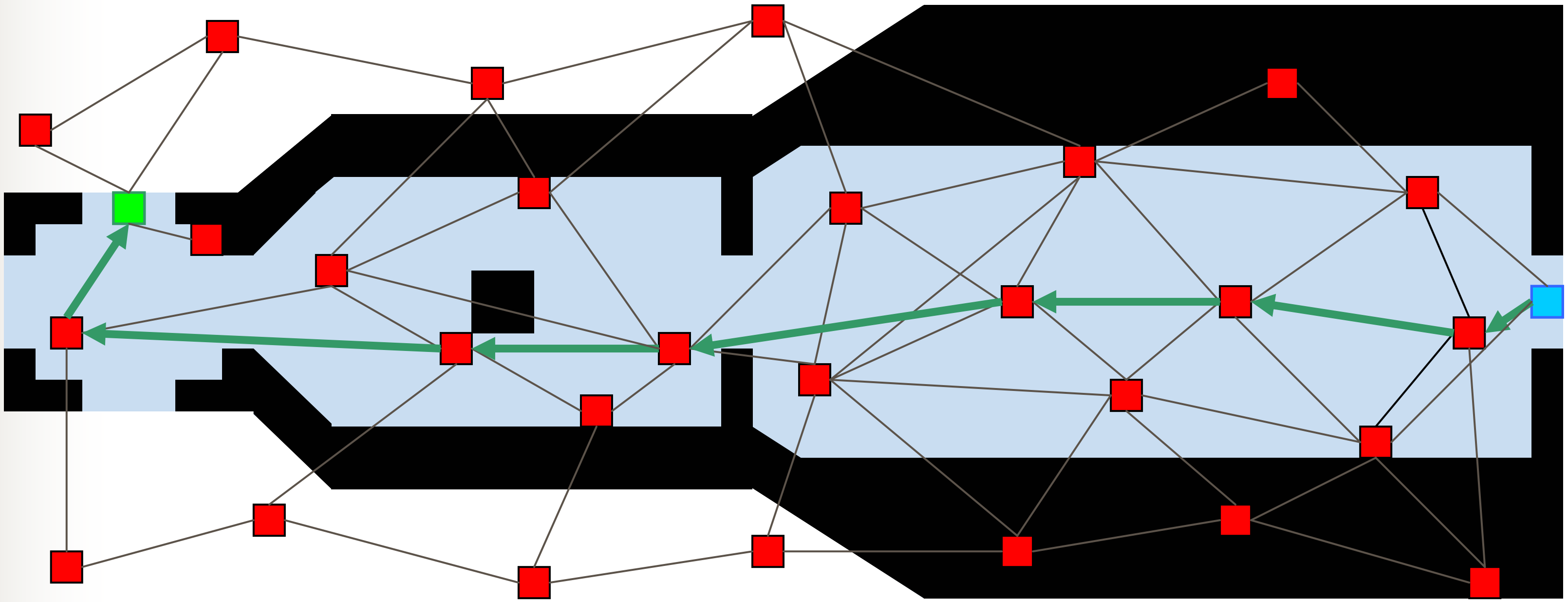
By Paul Elliott

Portable Satellite Assistant



courtesy NASA Ames

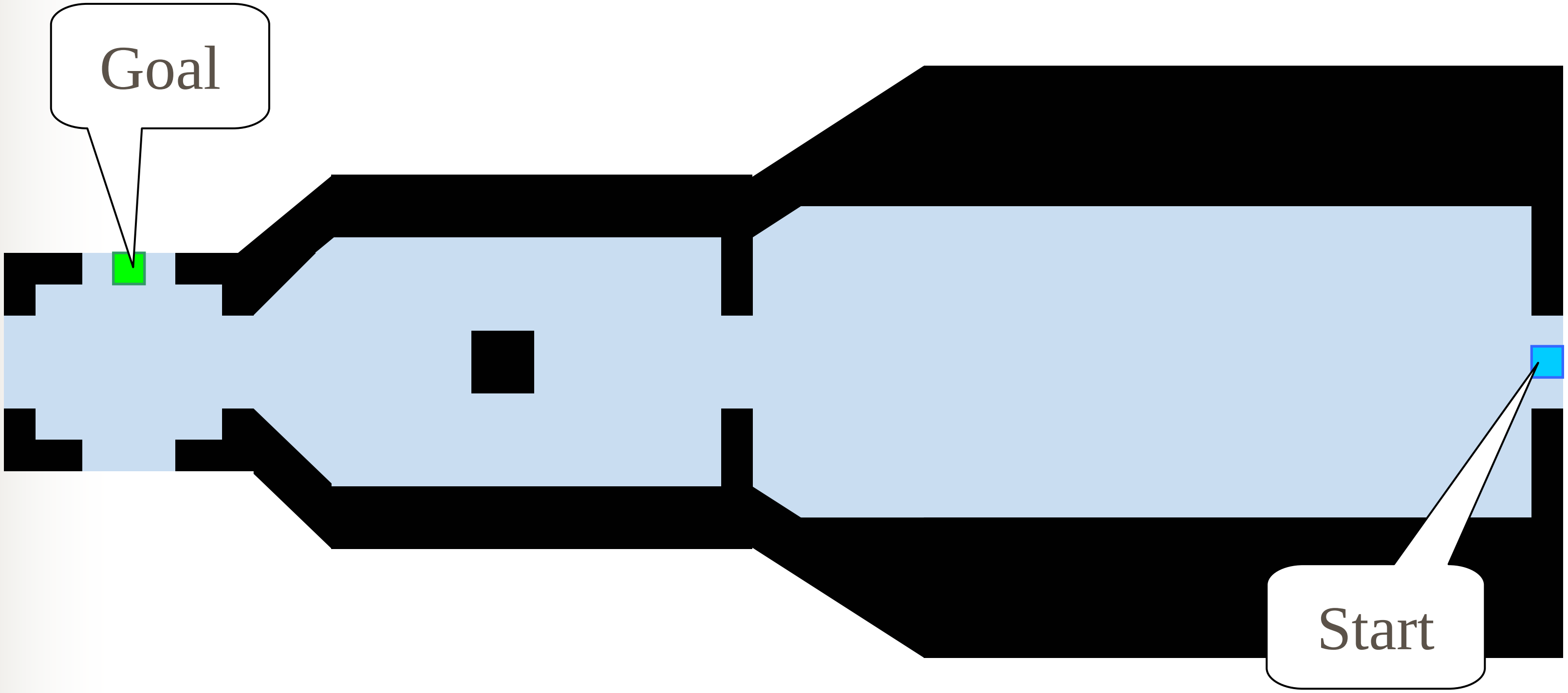
Zvezda Service Module



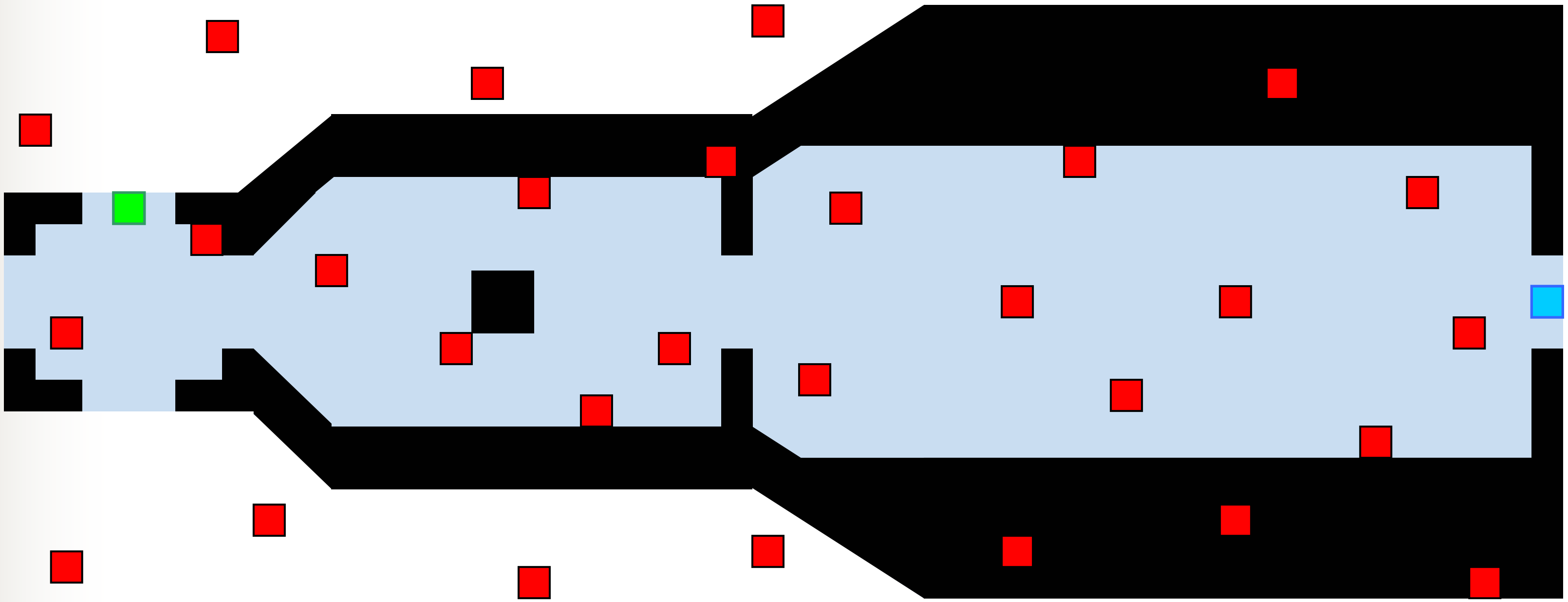
Idea: Probabilistic Roadmaps

- Search randomly generated roadmap
- Probabilistically complete
- Trim infeasible edges and nodes lazily

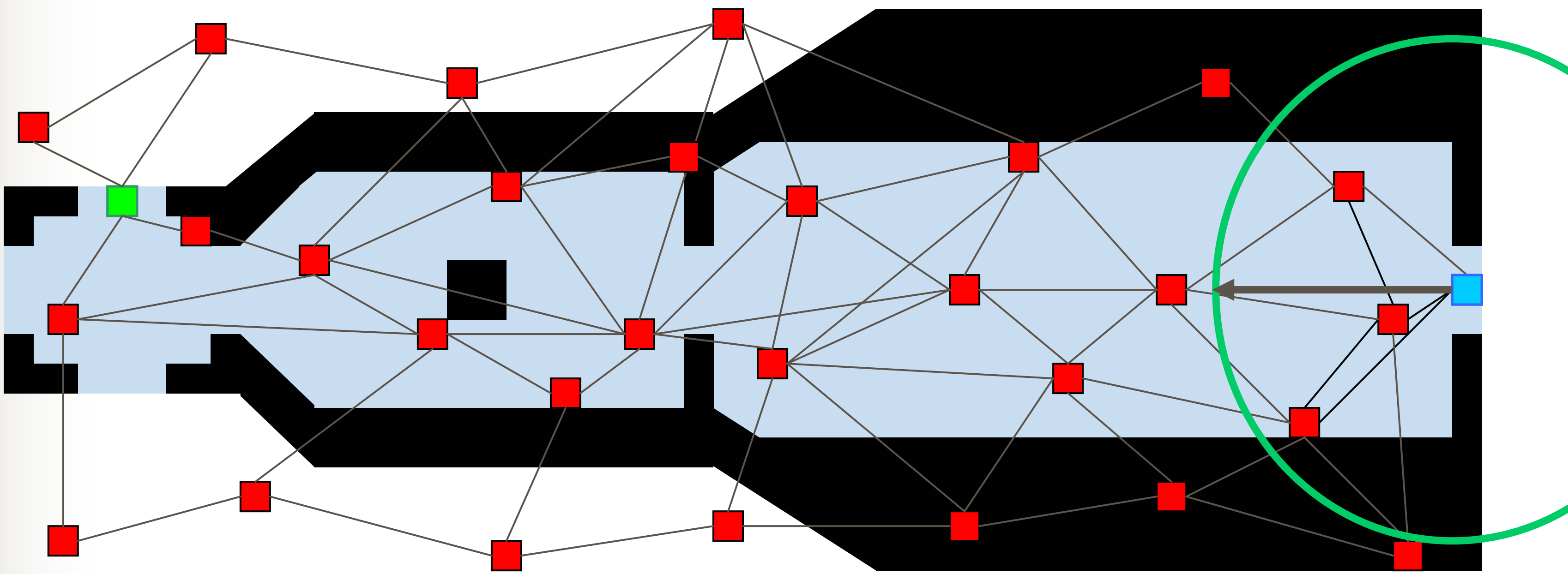
Place Start and Goal



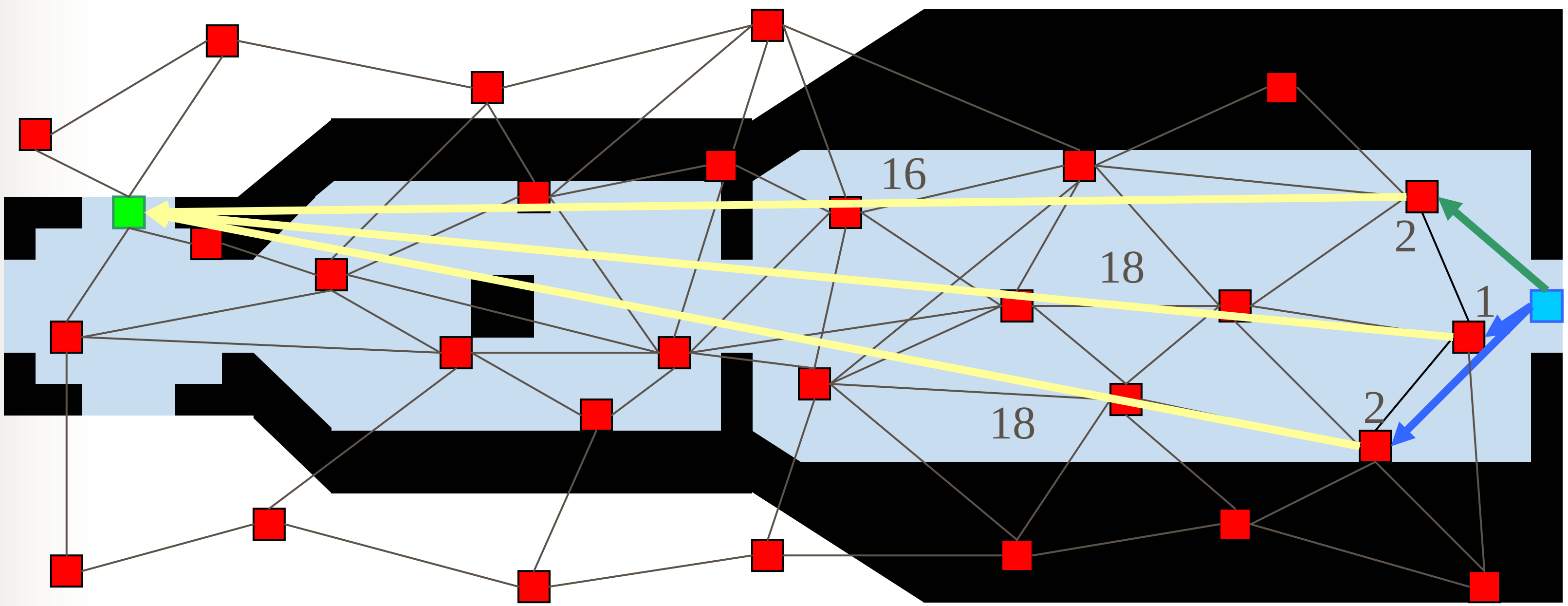
Place Nodes Randomly



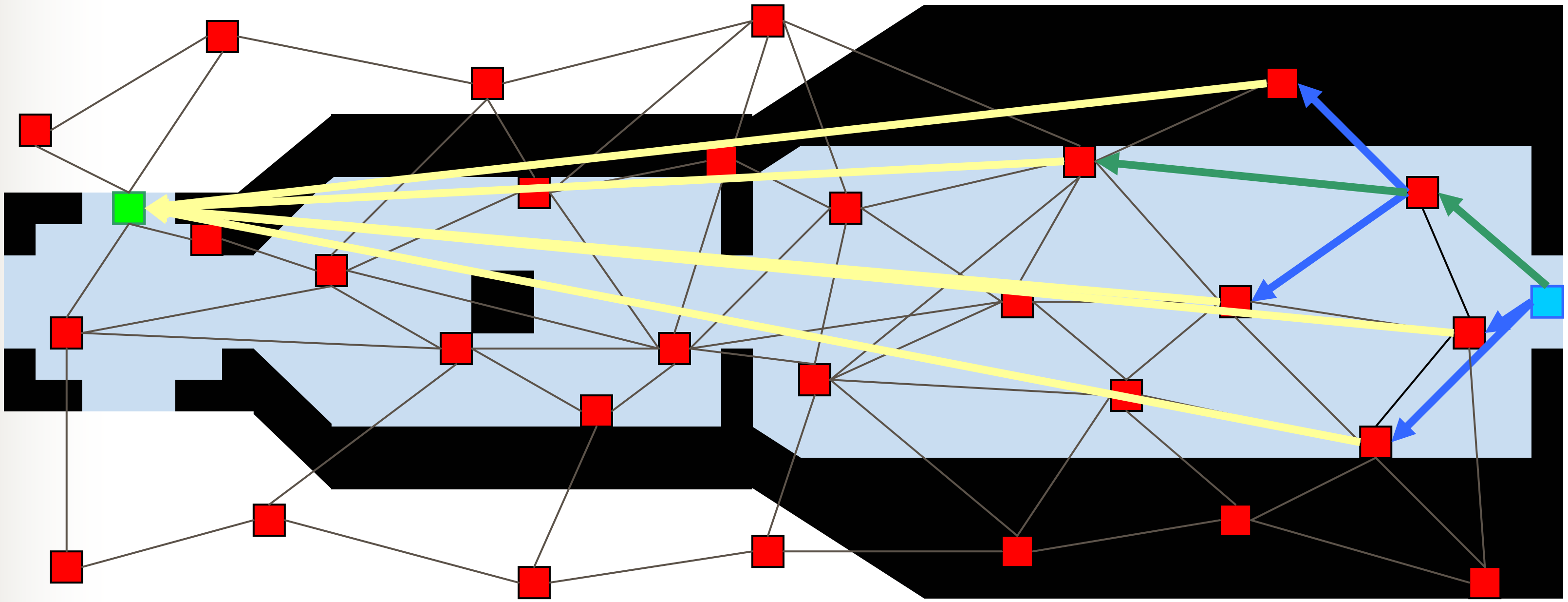
Select a Set of Neighbors



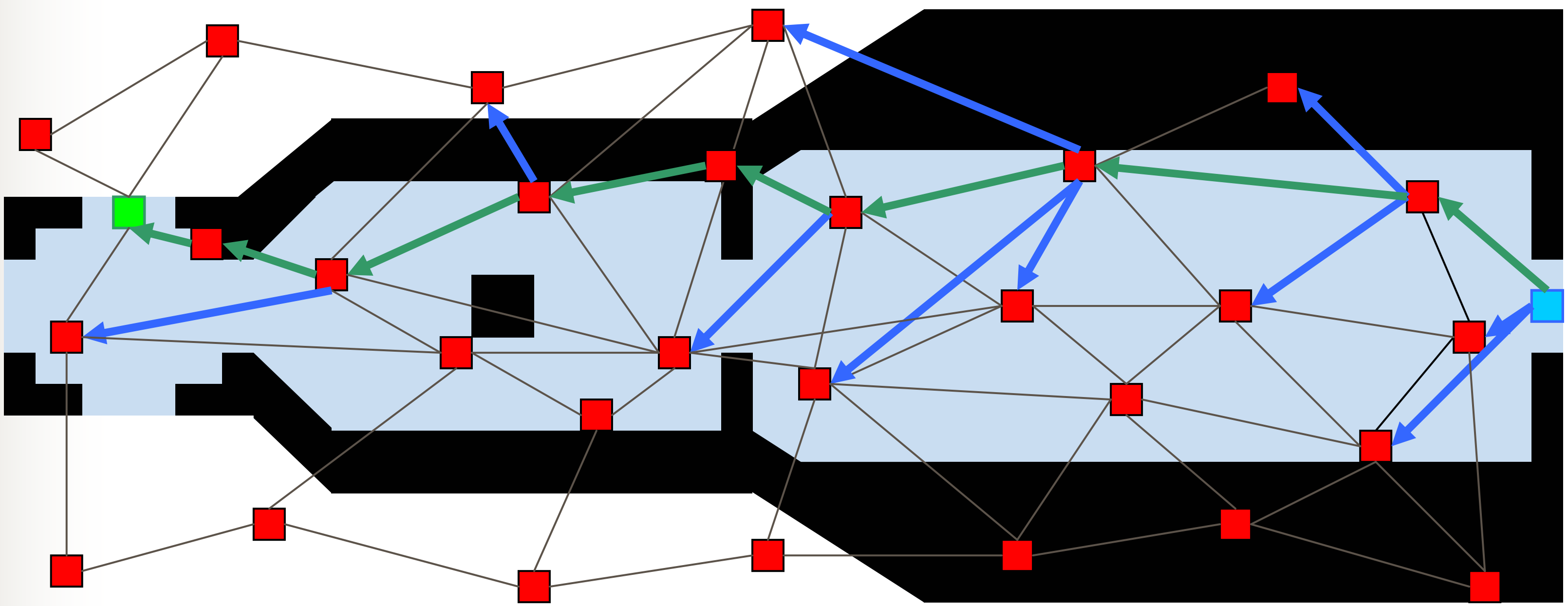
A* Search



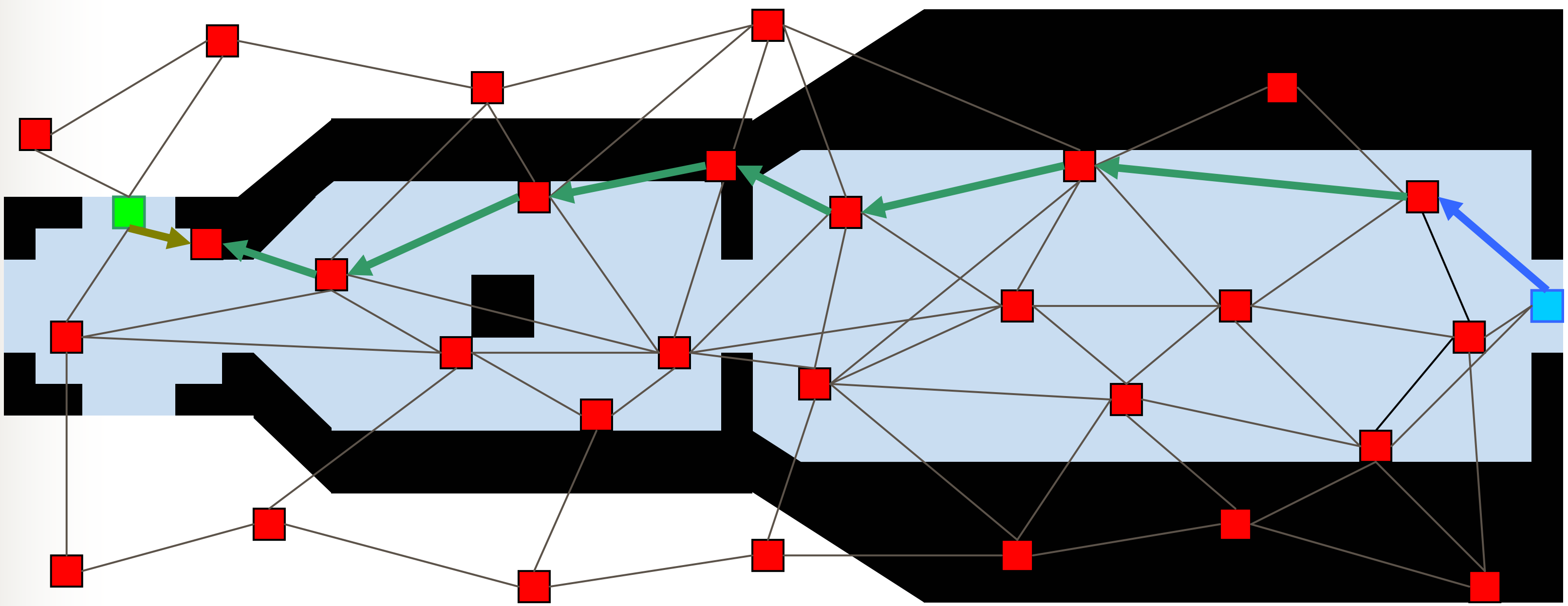
A* Search



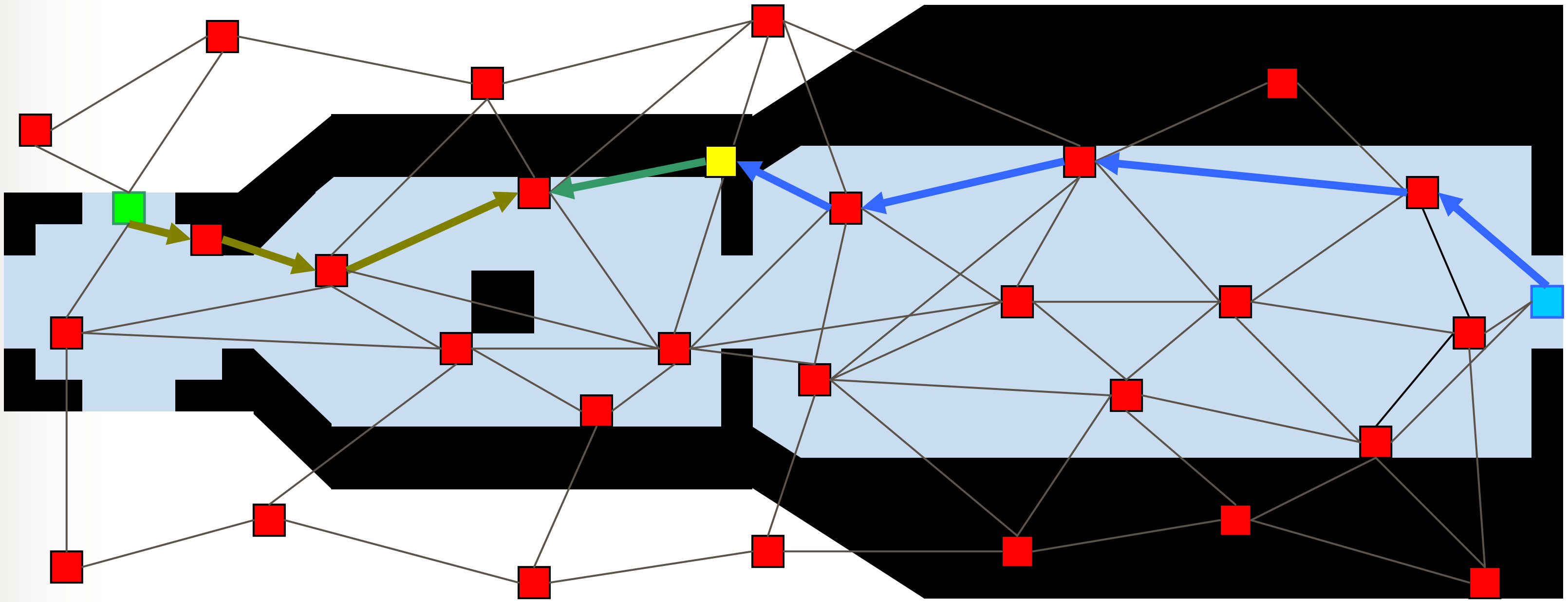
A* Search



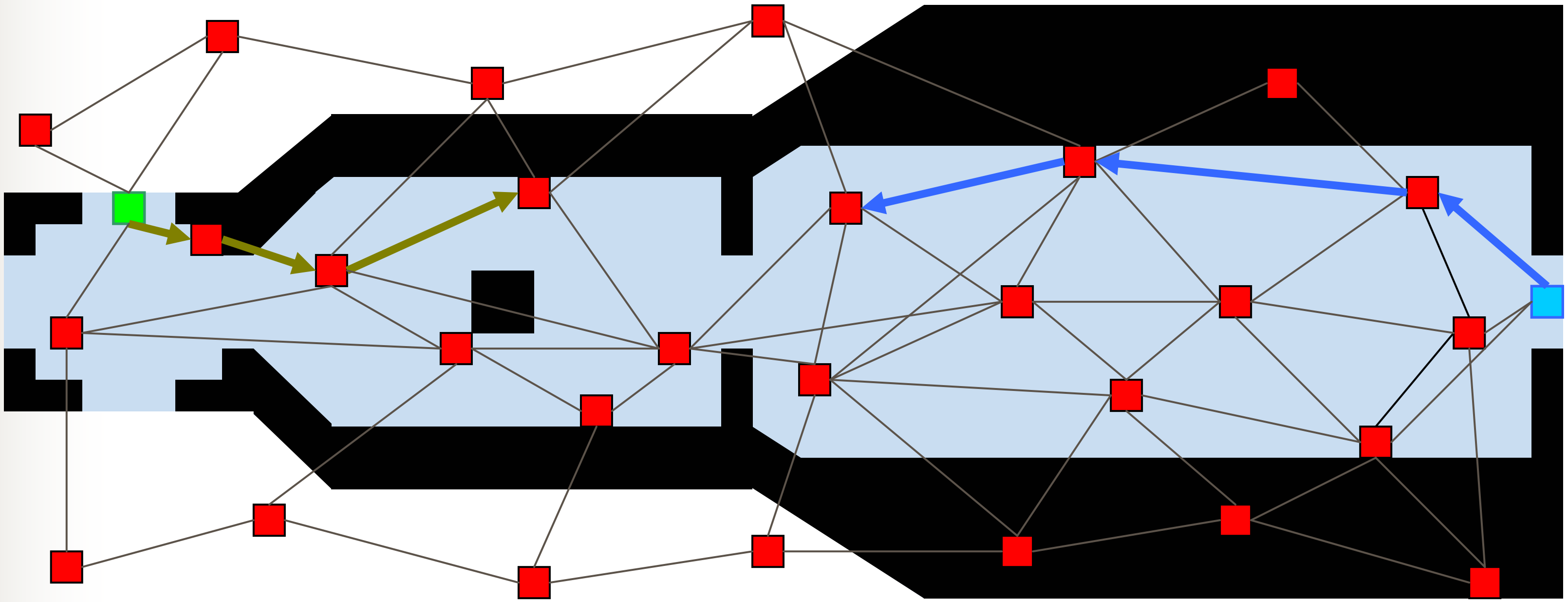
Check Feasible Nodes



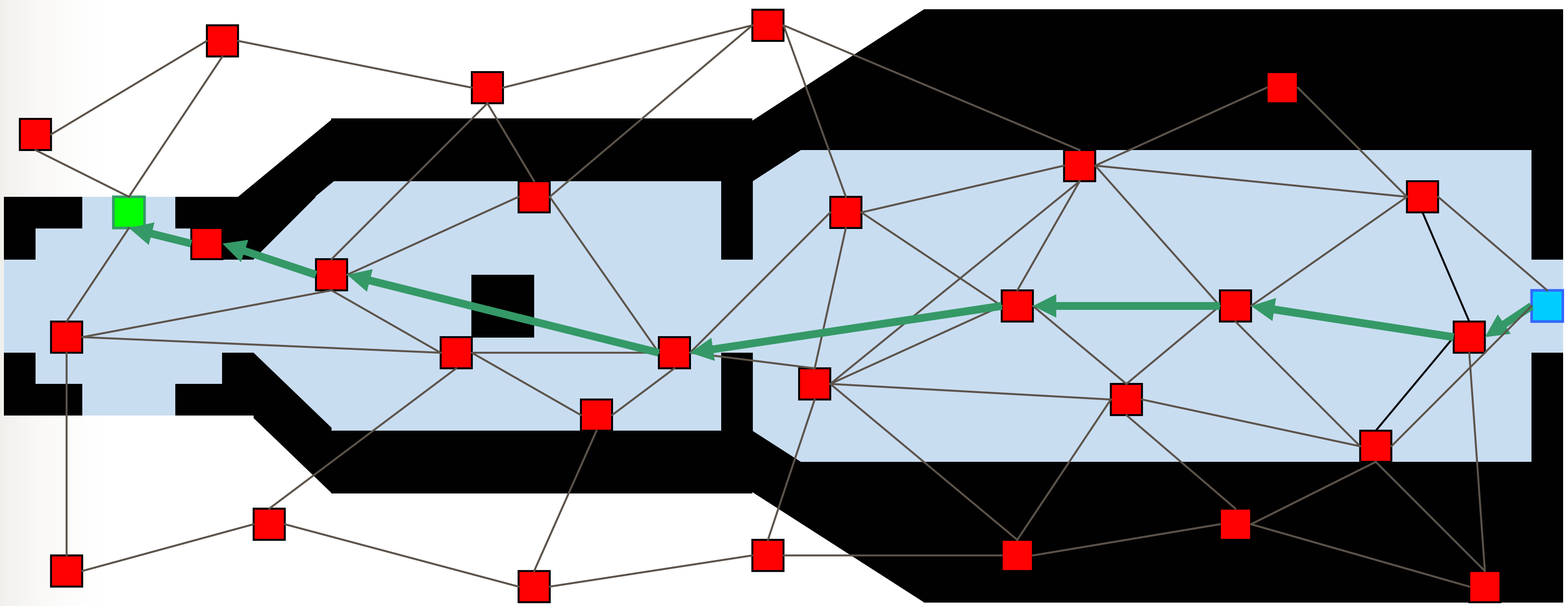
Check Feasible Nodes



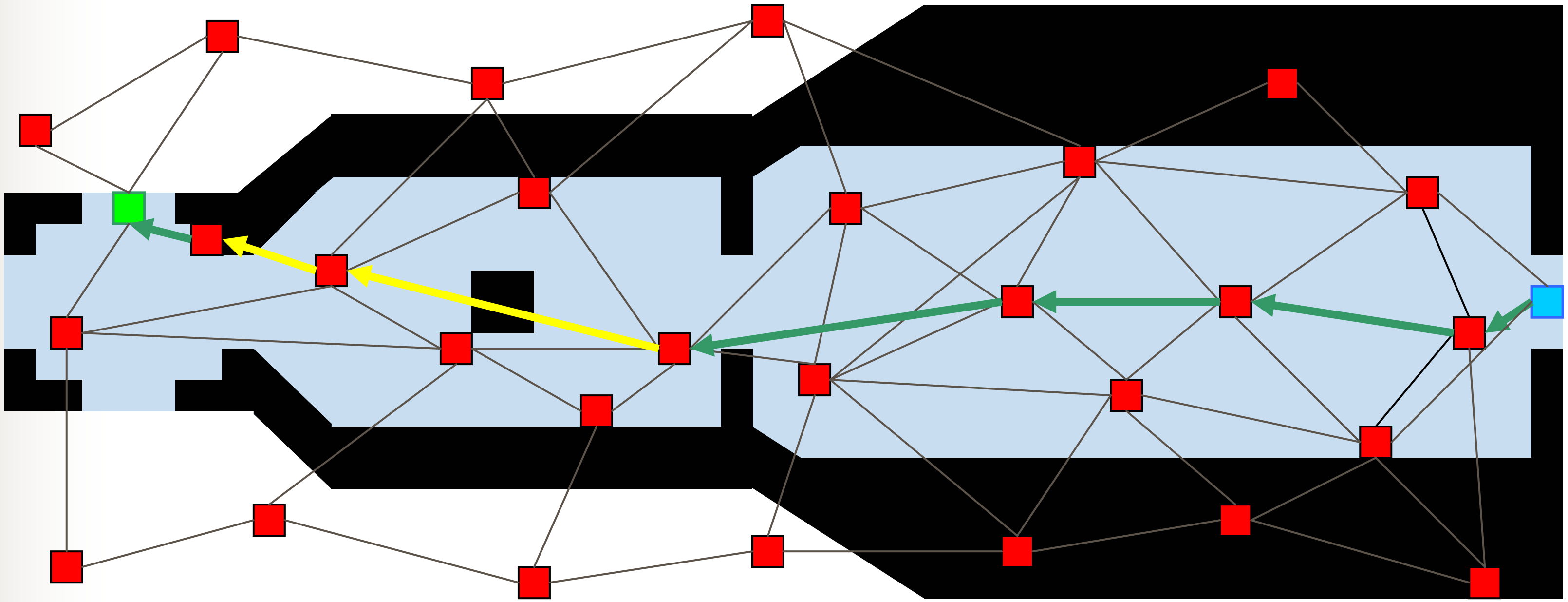
Check Feasible Nodes



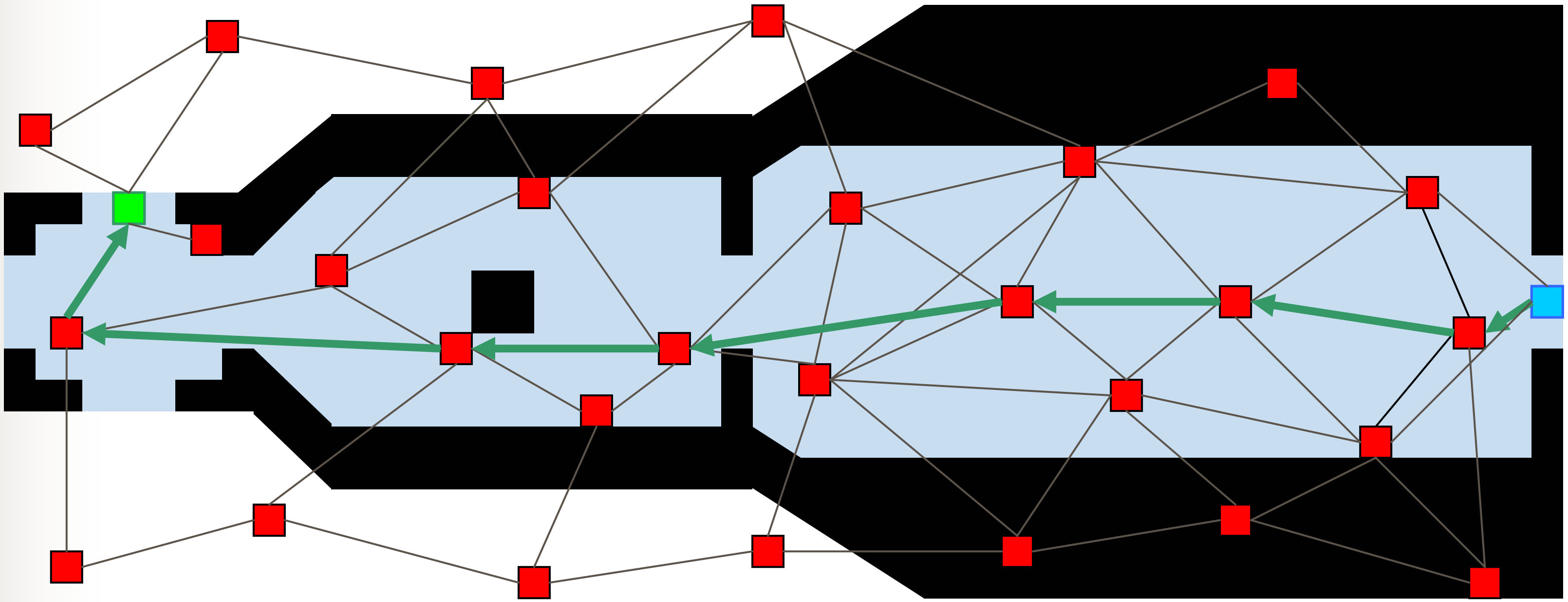
A* Search



Check Feasible Edges

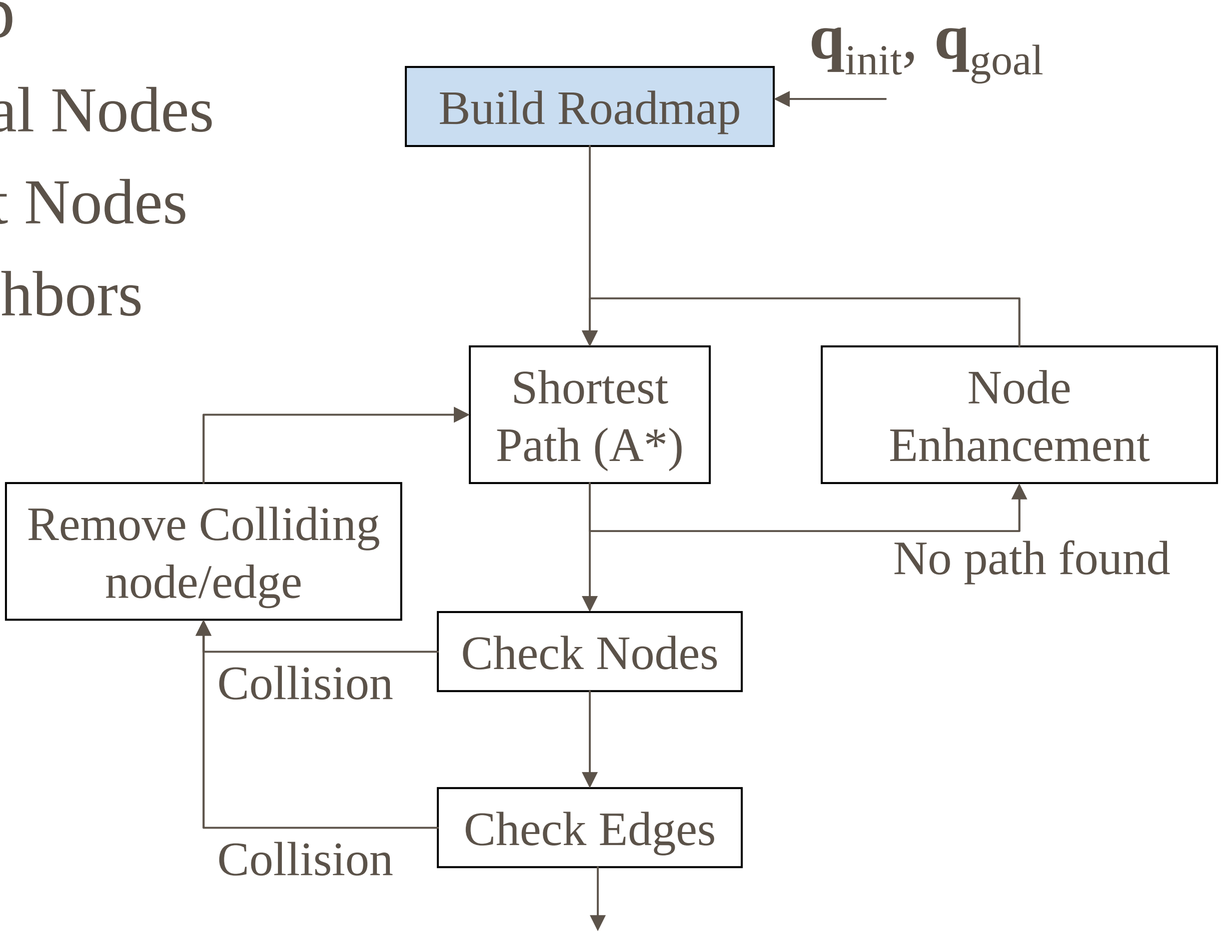


A* Search



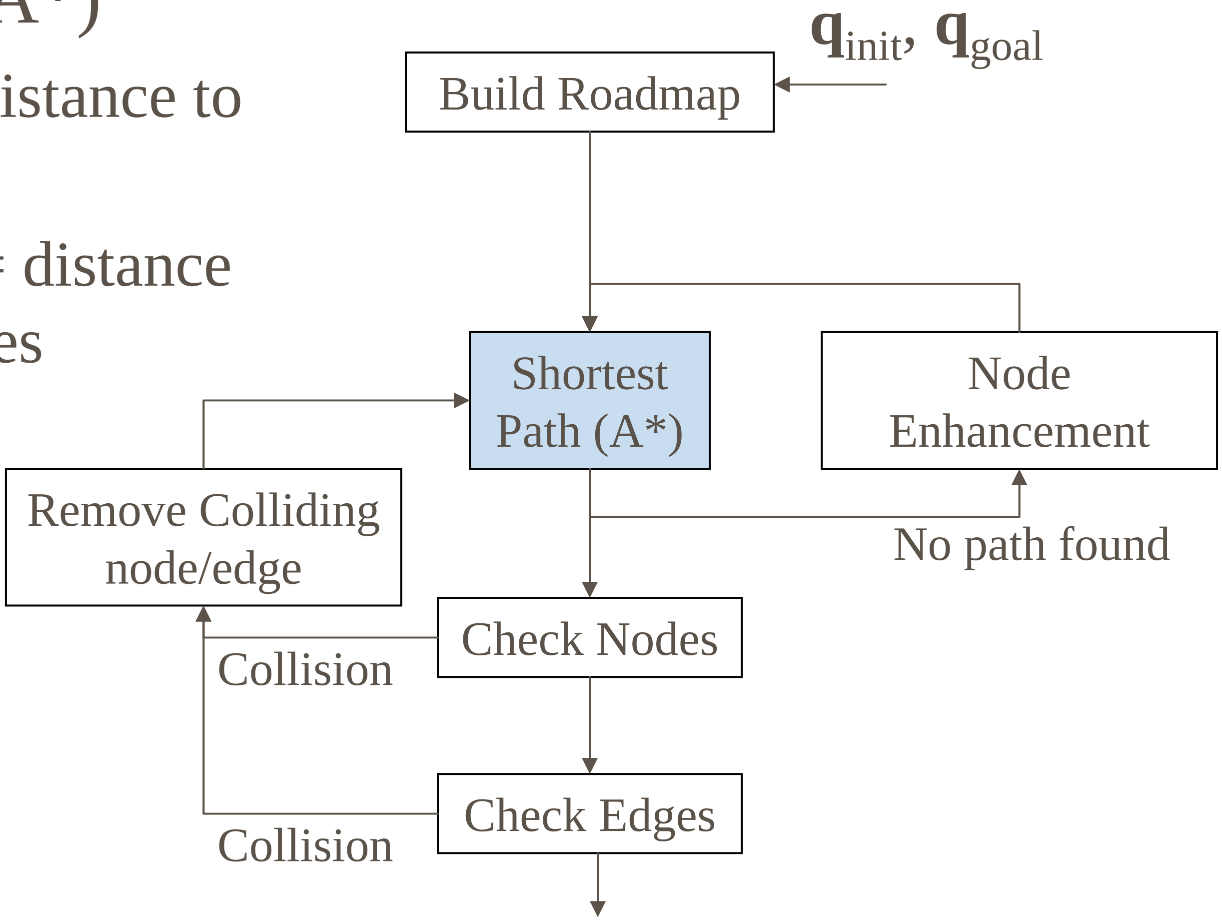
Lazy PRM Algorithm

- Build Roadmap
  - Start and Goal Nodes
  - Uniform Dist Nodes
  - Nearest Neighbors



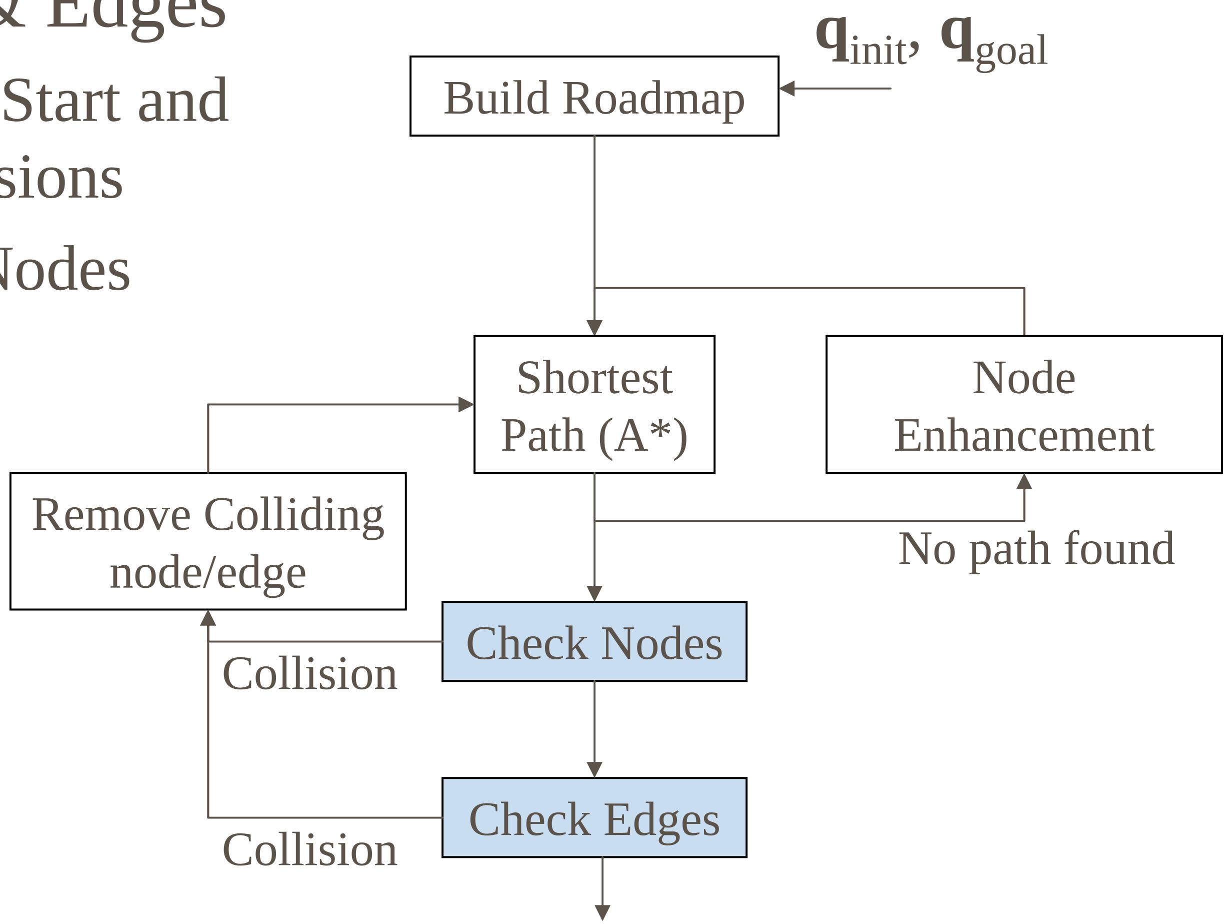
Lazy PRM Algorithm

- Shortest Path (A*)
  - Heuristic = distance to the goal
  - Path length = distance between nodes



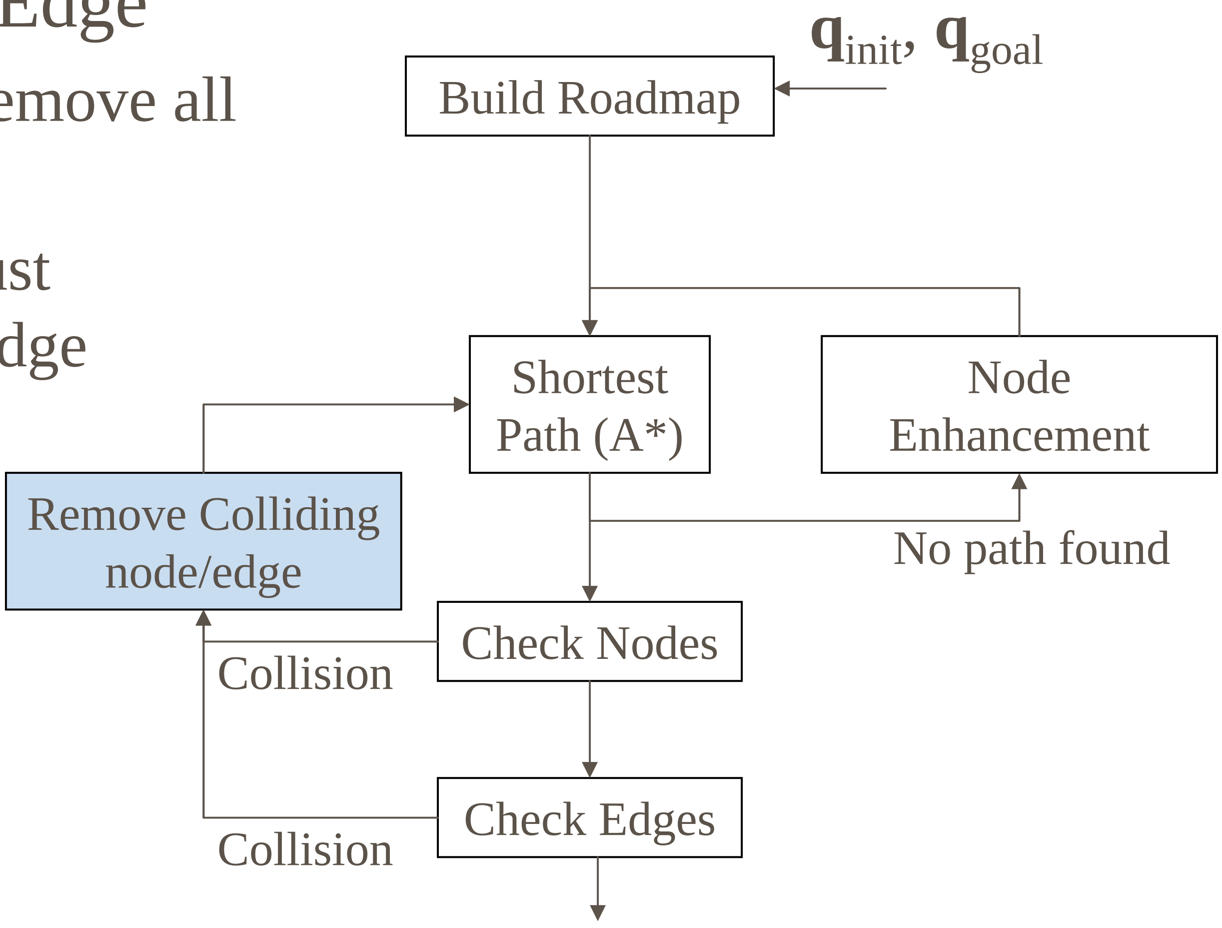
Lazy PRM Algorithm

- Check Nodes & Edges
  - Search from Start and End for collisions
  - First check Nodes then Edges



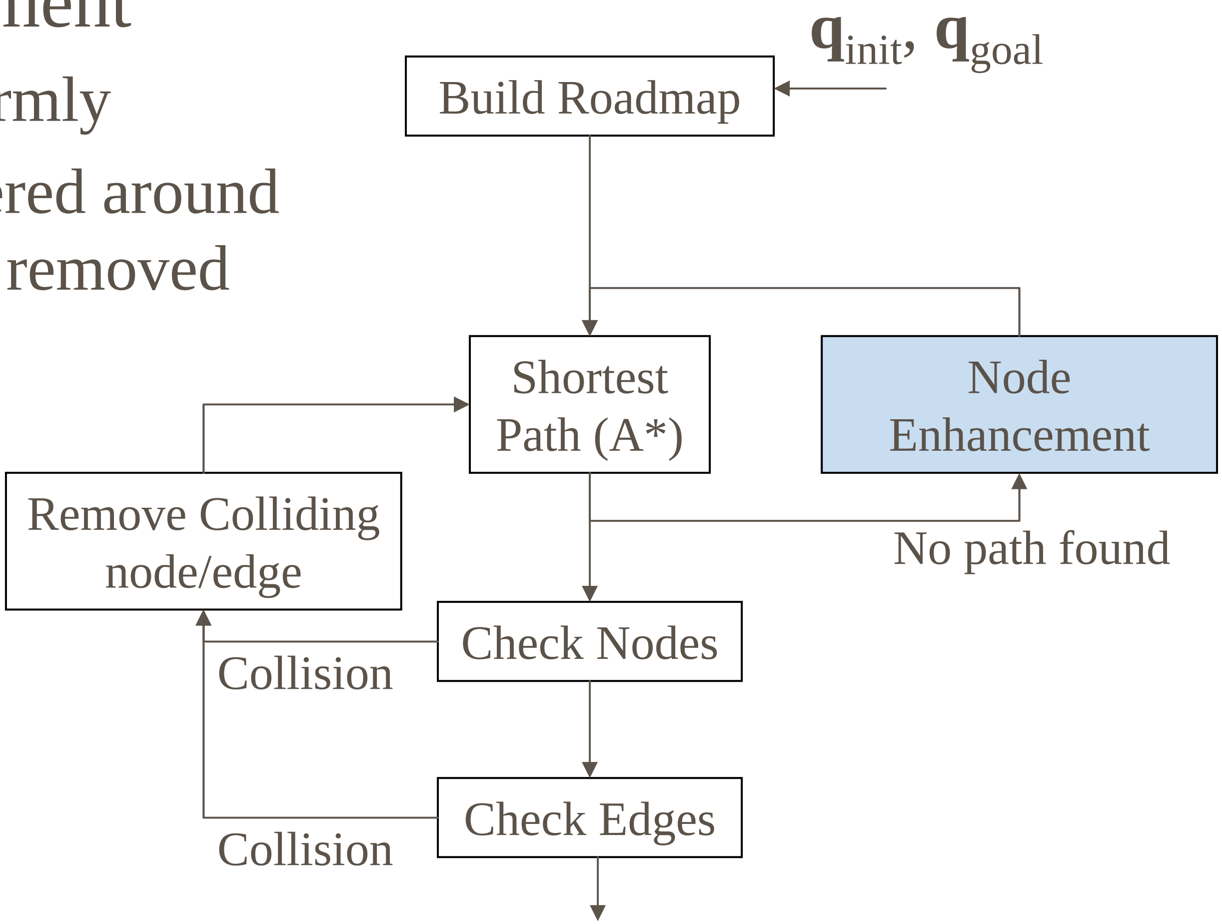
Lazy PRM Algorithm

- Remove Node/Edge
  - For Nodes, remove all edges
  - For Edges, just remove the edge



Lazy PRM Algorithm

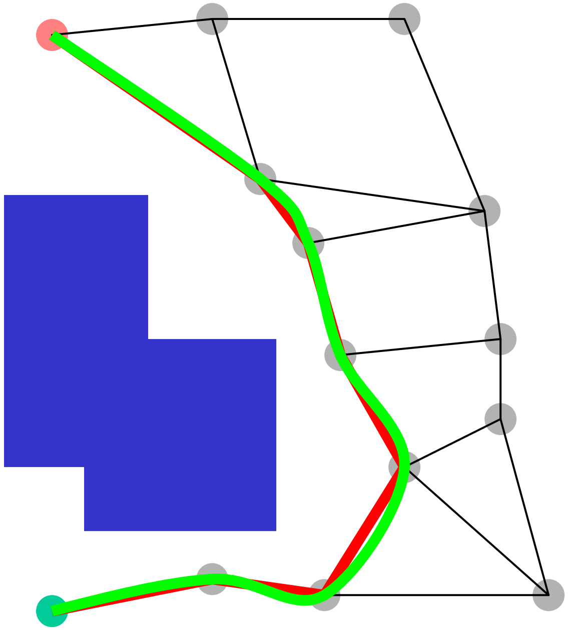
- Node Enhancement
  - Add ½ uniformly
  - Add ½ clustered around midpoints of removed edges



PRMs Fall Short For Dynamical Systems

- Using PRM
  1. Construct roadmap
  2. A* finds path in roadmap
  3. Must derive control inputs from path

- *Cannot always find inputs for an arbitrary path*



Outline

- Roadmap path planning
- Probabilistic roadmaps
- Planning in the real world
- Planning amidst moving obstacles
- RRT-based planners
- Conclusions

Path Planning in the Real World

Real World Robots

- Have inertia
- Have limited controllability
- Have limited sensors
- Face a dynamic environment
- Face an unreliable environment

Static planners (e.g. PRM) are not sufficient

Two Approaches to Path Planning

Kinematic: only concerned with motion, without regard to the forces that cause it

- Works well: when position controlled directly.
- Works poorly: for systems with significant inertia.

Kinodynamic: incorporates dynamic constraints

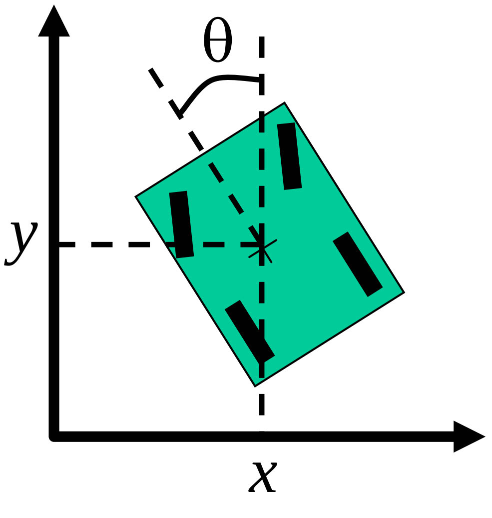
- Plans velocity as well as position

Representing Static State

- Configuration space represents the position and orientation of a robot
- Sufficient for static planners like PRM

Example: Steerable car

Configuration space
(x, y, θ)



Representing Dynamic State

- State space incorporates robot dynamic state
- Allows expression of dynamic constraints
- Doubles dimensionality

Example: Steerable car

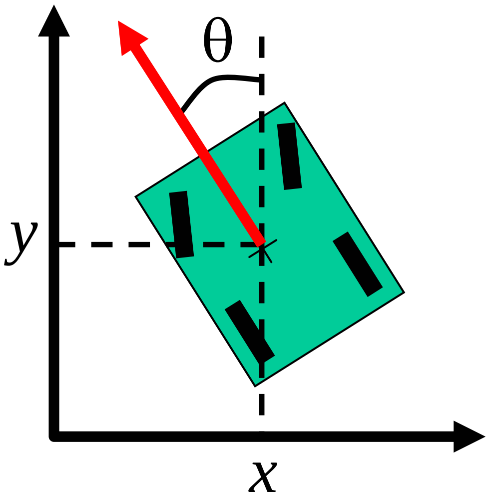
State space

$X = (x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$

Constraints

•max velocity, min turn

•car dynamics



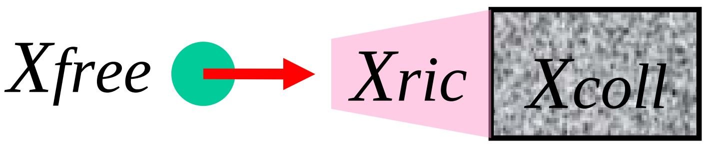
Incorporating Dynamic Constraints

- For some states, collision is unavoidable
  - Robot actuators can apply limited force

- Path planner should avoid these states

Regions in State Space

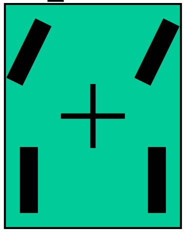
- Collision regions: X_{coll}
  - Clearly illegal
- Region of Imminent Collision: X_{ric}
  - Where robot's actuators cannot prevent a collision
- Free Space: $X_{free} = X - (X_{coll} + X_{ric})$



- Collision-free planning involves finding paths that lie entirely in X_{free}

Constraints on Maneuvering

- Nonholonomic: Fewer controllable degrees of freedom then total degrees of freedom
- Example: steerable car



  - 3 dof (x, y, θ), but only
  - 1 controllable dof (steering angle)

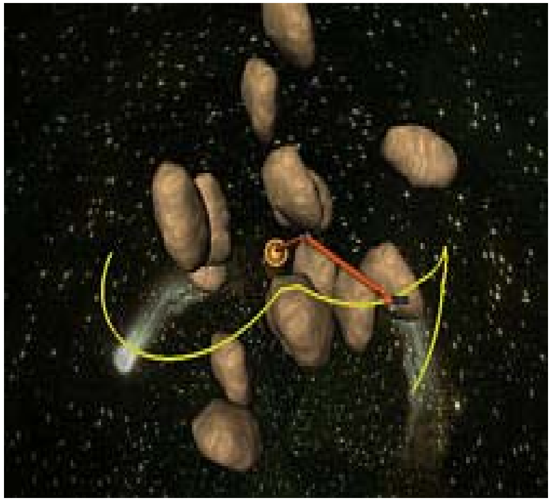
- Equation of Motion: $G(s, \dot{s}) = 0$
  - Constraint is a function of state and time derivative of state

Outline

- Roadmap path planning
- Probabilistic roadmaps
- Planning in the real world
- Planning amidst moving obstacles
- RRT-based planners
- Conclusions

Problem

- Kinodynamic motion planning amidst moving obstacles with known trajectories
- Example: Asteroid avoidance problem
- Moving Obstacle Planner (MOP)
  - *Extension to PRM*



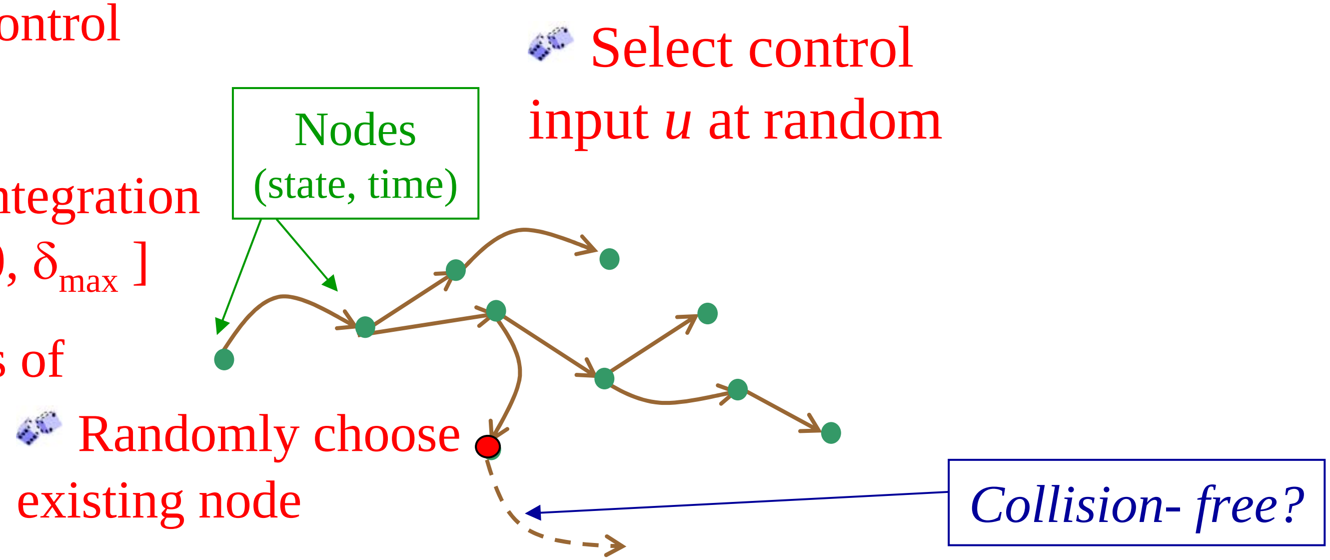
MOP Overview

Similar to PRM, except

- Does **not pre-compute** the roadmap
- **Incrementally constructs** the roadmap by extending it from existing nodes
- Roadmap is a **directed tree** rooted at initial **state × time** point and oriented along time axis

Building the Roadmap

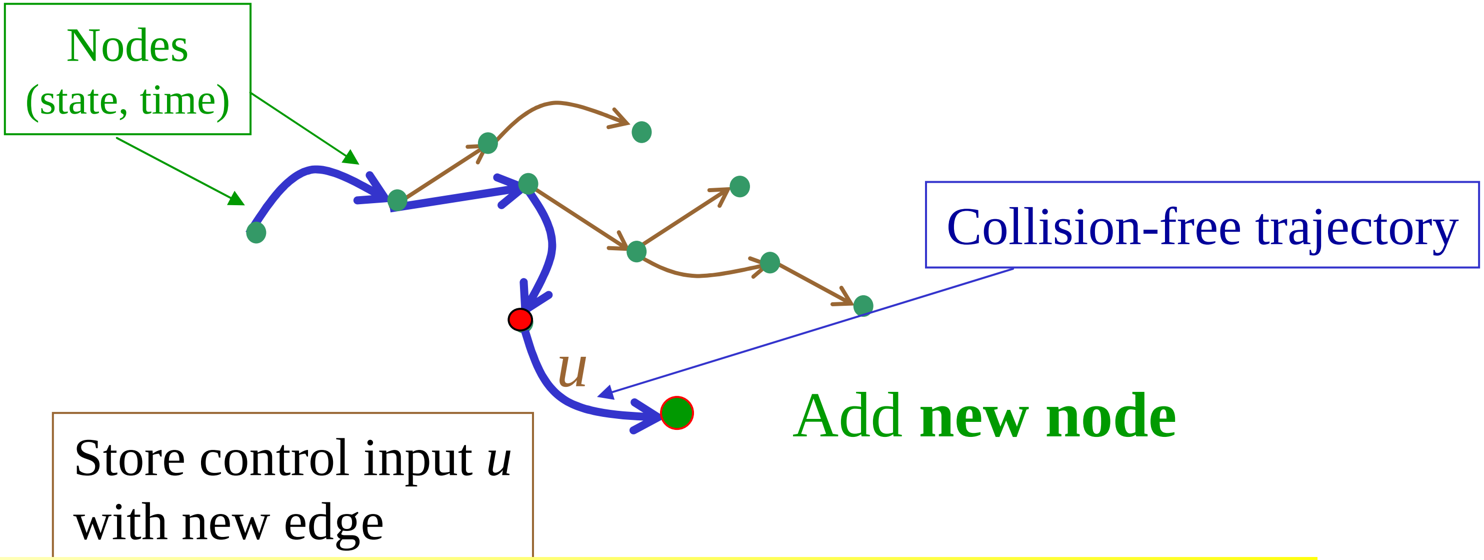
1. Randomly choose an existing node
2. Randomly select control input *u*
3. Randomly select integration time interval $\delta \in [0, \delta_{max}]$
4. Integrate equations of motion



Integrate *equations of motion* from an existing node with respect to *u* for some time interval δ

Building the Roadmap (cont.)

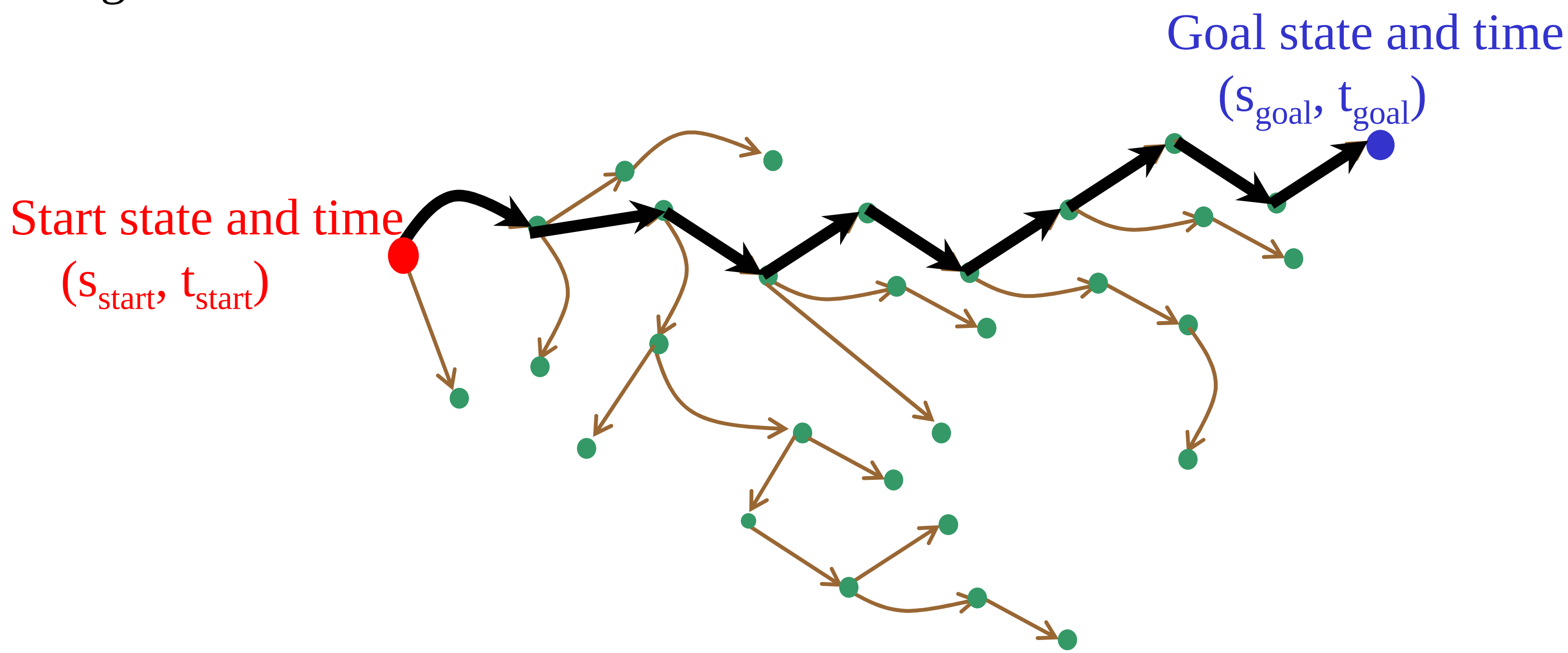
5. **If** edge is collision-free **then**
6. Store control input with new edge
7. Add new node to roadmap



<u>Result:</u> Any trajectory along tree satisfies motion constraints and is collision-free!

Solution Trajectory

1. **If** goal is reached **then**
2. Proceed backwards from the goal to the start



MOP details: Inputs and Outputs

Planning Query:

- Let (s_{start}, t_{start}) denote the robot's start point in the state × time space, and (s_{goal}, t_{goal}) denote the goal
- $t_{goal} \in I_{goal}$, where I_{goal} is some time interval in which the goal should be reached

Solution Trajectory:

- Finite sequence of fixed control inputs applied over a specified duration of time
  - Avoids moving obstacles by indexing each state with the time when it is attained
  - Obeys the dynamic constraints

MOP details: Roadmap Construction

- Objective: obtain new node (s', t')
  - s' = the new state in the robot's state space
  - t' = t + δ, current time plus the integration time

Each iteration:

1. Select an existing node (s, t) in the roadmap at random
2. Select control input *u* at random
3. Select integration time δ at random from $[0, \delta_{max}]$

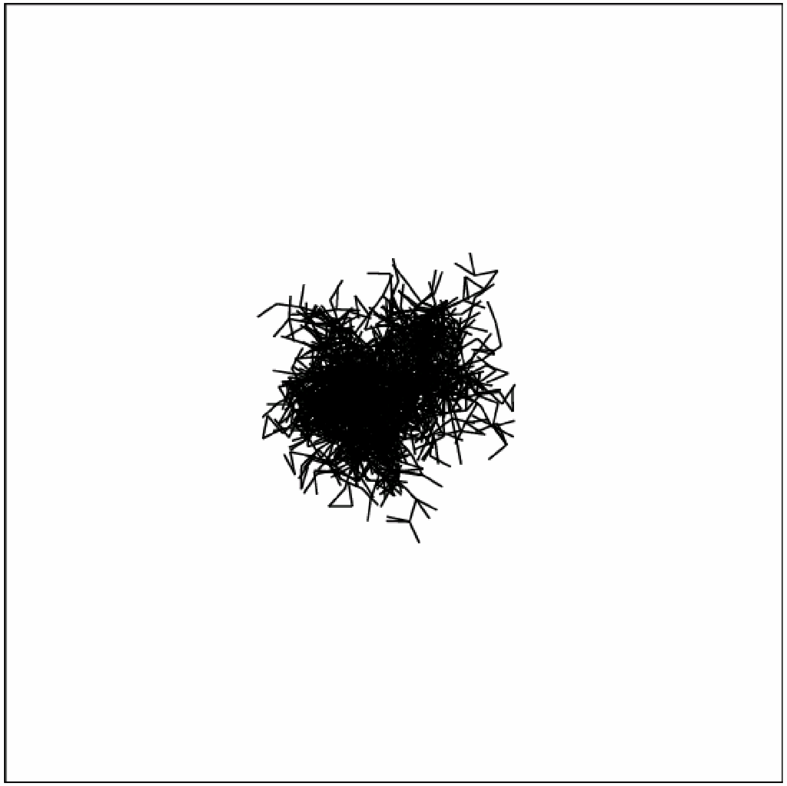
MOP details: Roadmap Construction

3. Integrate control inputs over time interval
4. Edge between (s, t) and (s’, t’) is checked for collision with static obstacles and moving obstacles
5. If collision-free, store control input *u* with the new edge
6. (s’, t’) is accepted as new node

MOP details: Uniform Distribution

Modify to Ensure Uniform Distribution of Space:

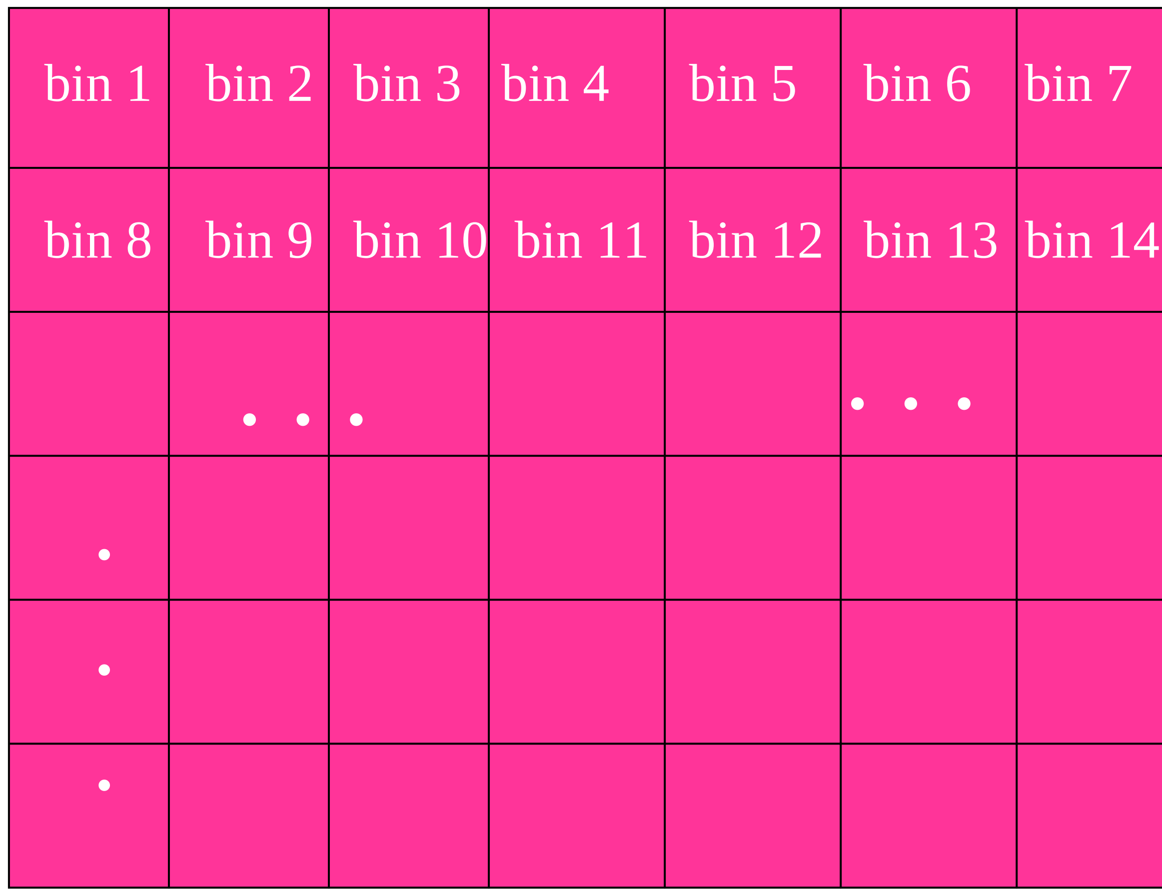
- **Why?** If existing roadmap nodes were selected uniformly, the planner would pick a node in an already densely sampled region
- Avoid oversampling of any region by dividing the state×time space into bins



Achieving Uniform Node Distribution

1. Equally divide space
2. Denote each section as a bin; number each bin

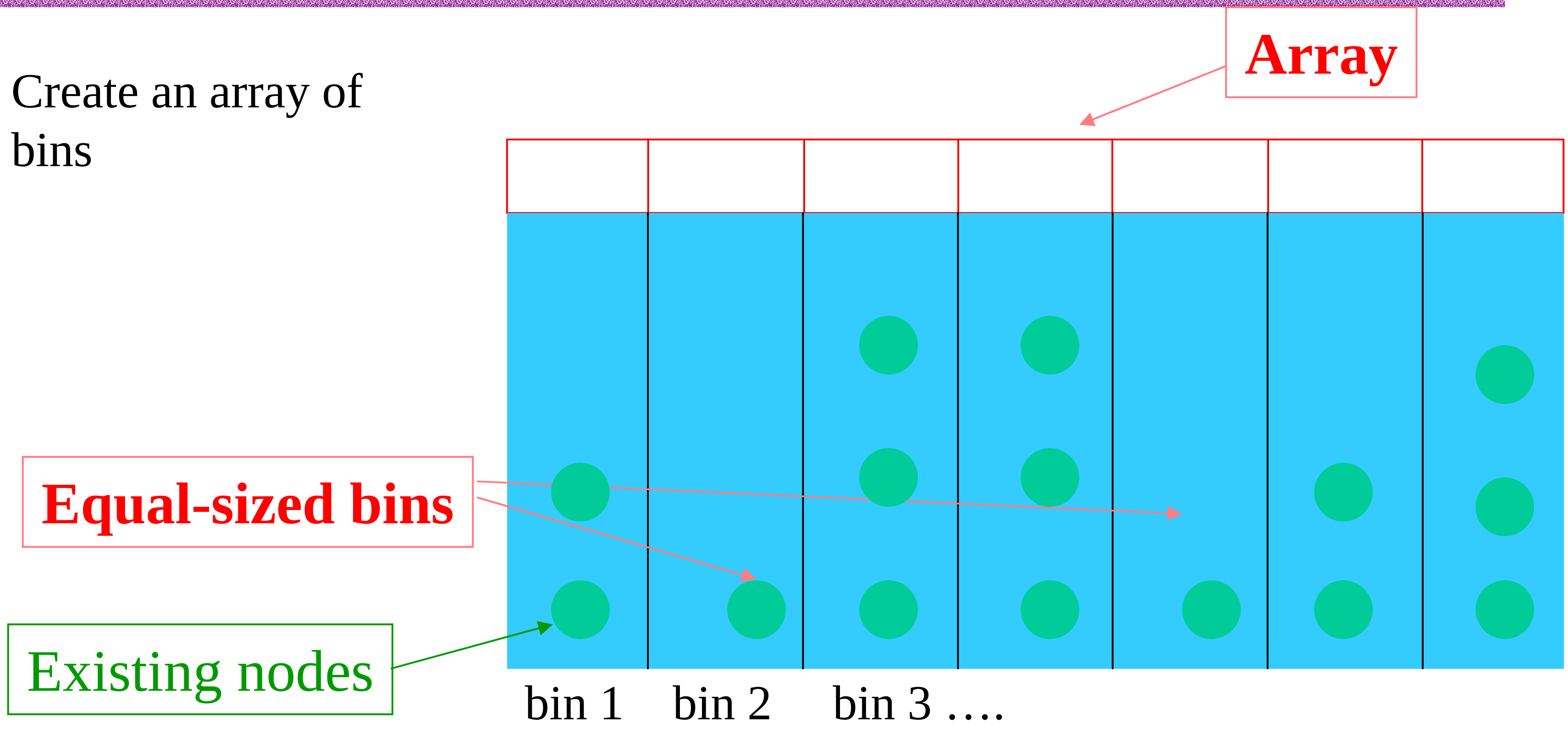
Space



*bins store roadmap nodes that lie in their region

Achieving Uniform Node Distribution

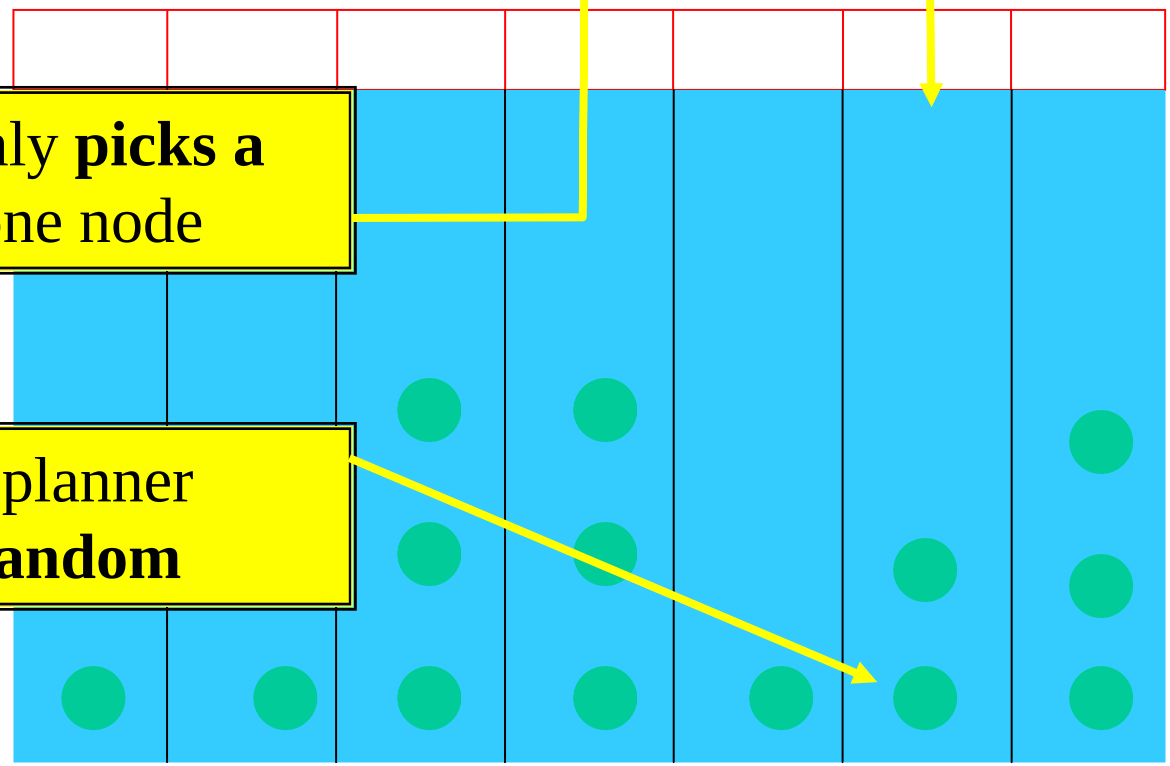
3. Create an array of bins



Achieving Uniform Node Distribution

- Planner randomly **picks a bin** with at least one node

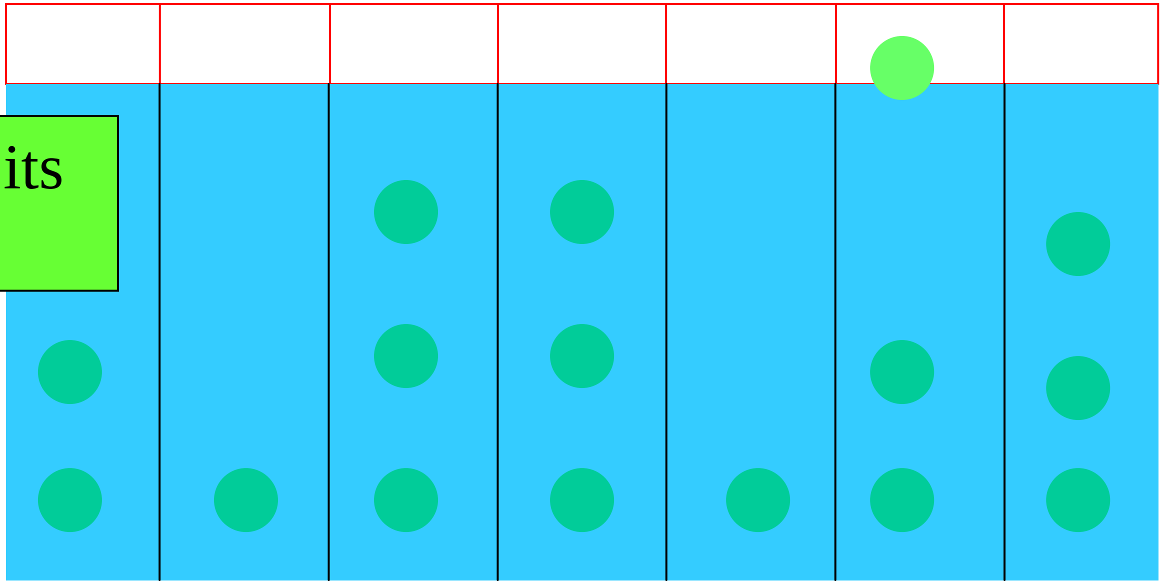
- At that bin, the planner **picks a node at random**



bin 1 bin 2 bin 3 ….

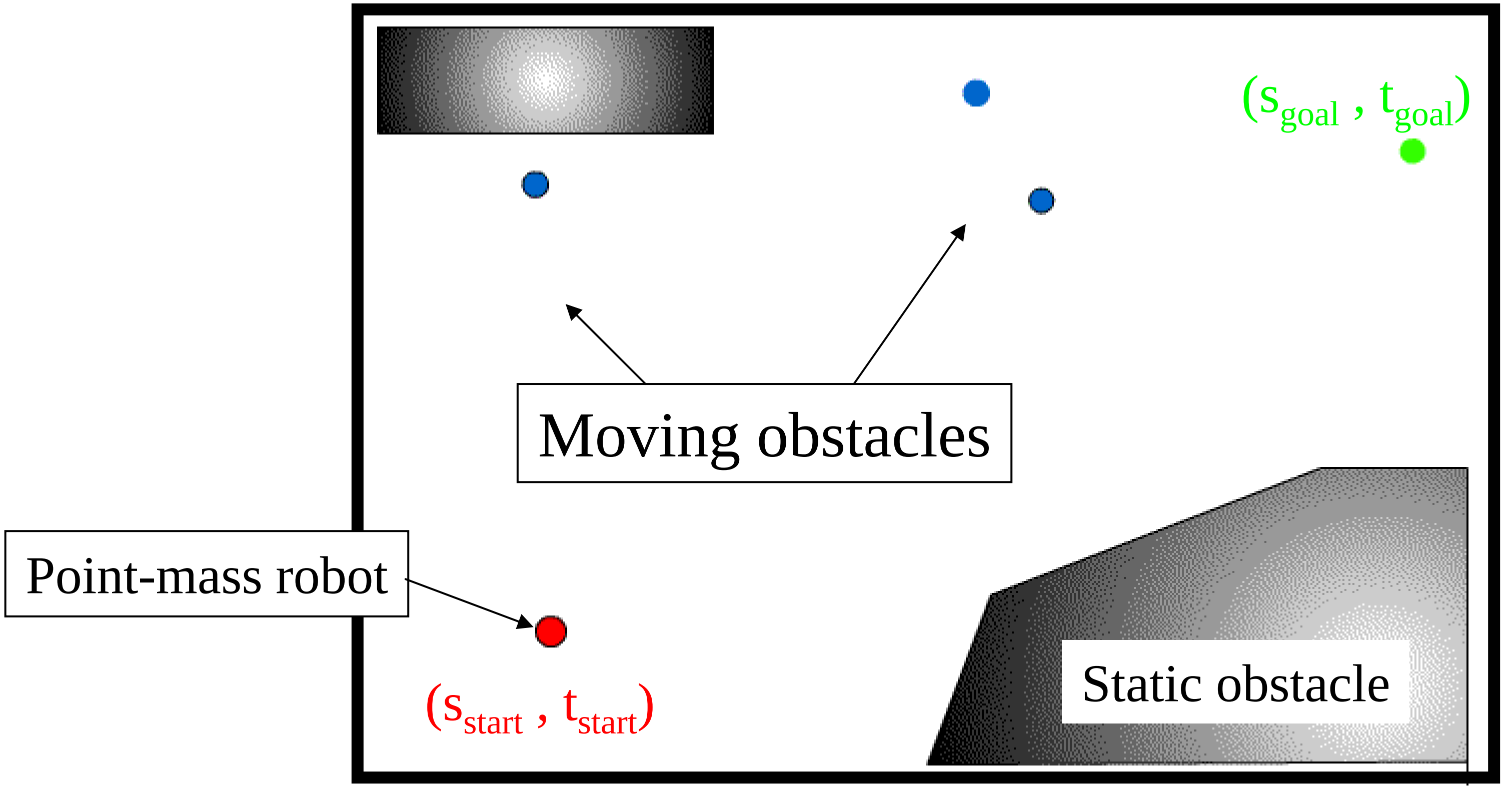
Achieving Uniform Node Distribution

Insert new node into its corresponding bin

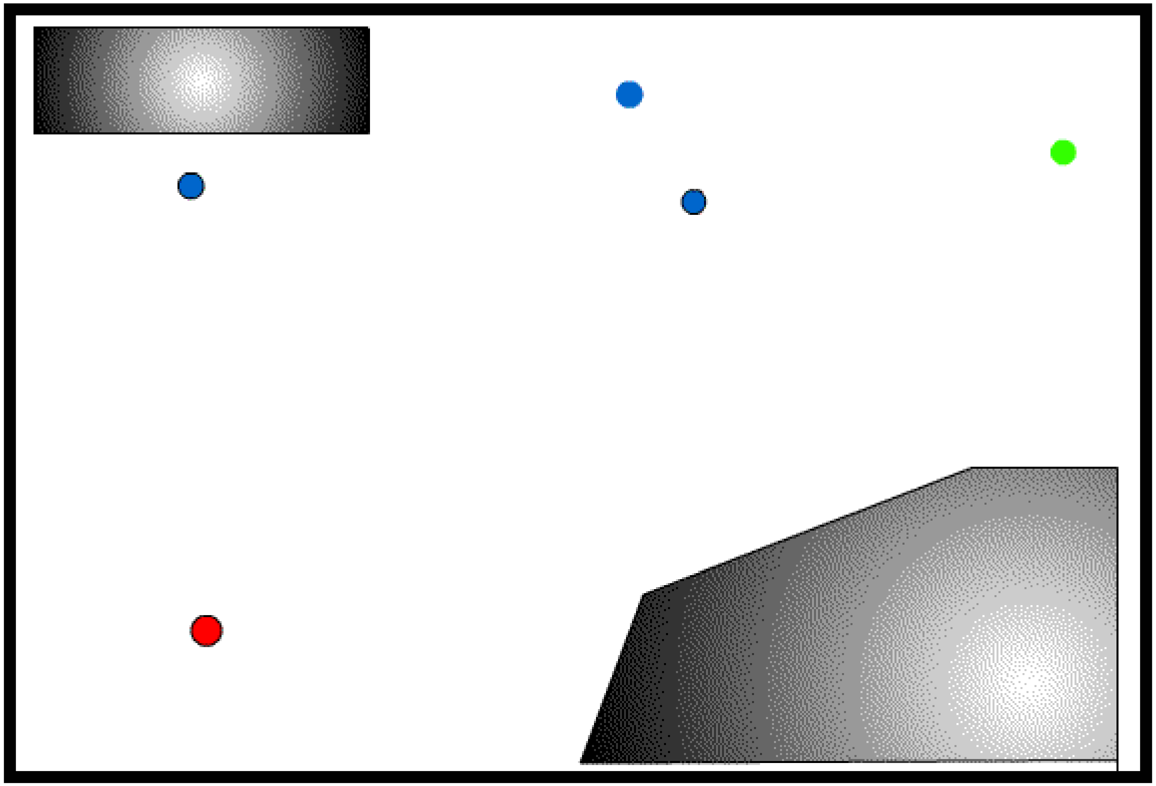


Demonstration of MOP

- *Point–mass* robot moving in a plane
  - State s = $(x, y, \dot{x}, \dot{y})$



Demonstration of MOP



Summary

- MOP algorithm incrementally builds a roadmap in the state×time space
- The roadmap is a directed tree oriented along the time axis
- By including time the planner is able to generate a solution trajectory that
  - avoids moving and static obstacles
  - obeys the dynamic constraints
- Bin technique to ensure that the space is explored somewhat uniformly

Outline

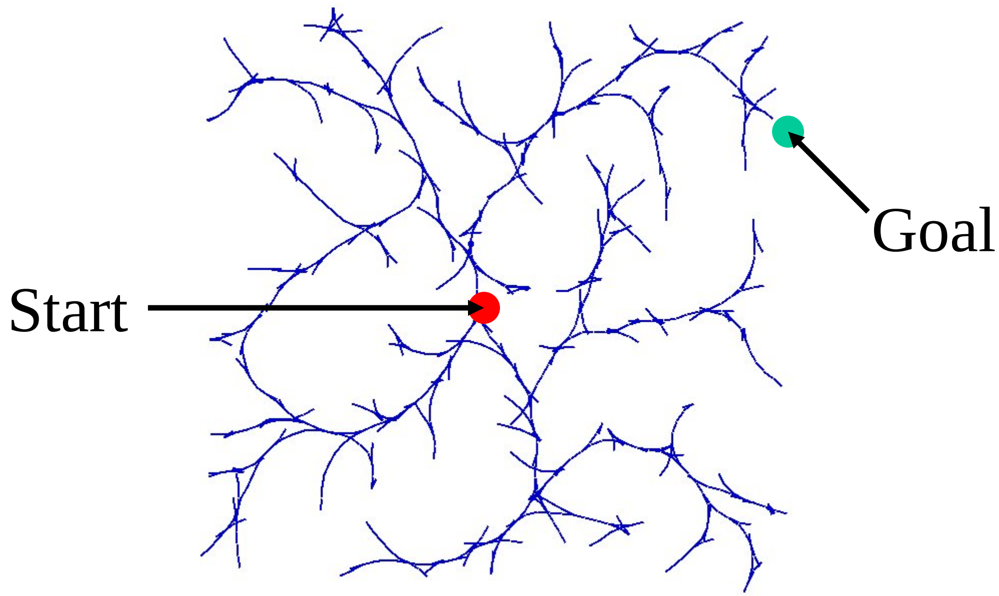
- Roadmap path planning
- Probabilistic roadmaps
- Planning in the real world
- Planning amidst moving obstacles
- RRT-based planners
- Conclusions

Planning with RRTs

- RRTs: Rapidly-exploring Random Trees
- Similar to MOP
  - **Incrementally builds** the roadmap tree
  - **Integrates** the **control inputs** to ensure that the kinodynamic constraints are satisfied
- **Informed exploration strategy** from MOP
- Extends to more advanced planning techniques

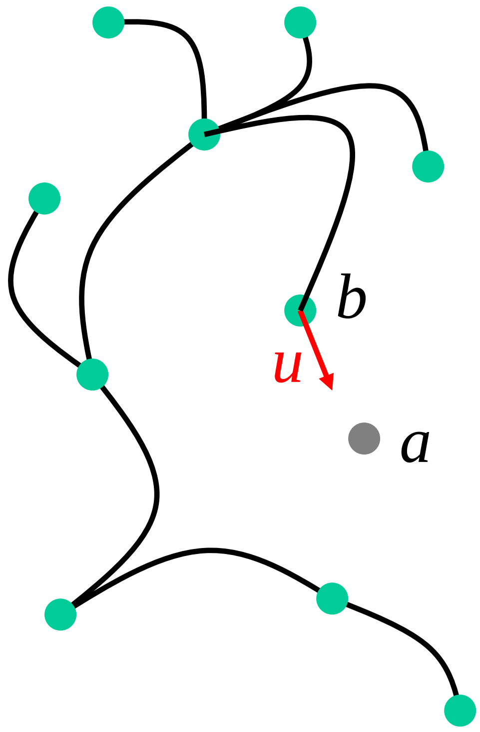
How it Works

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



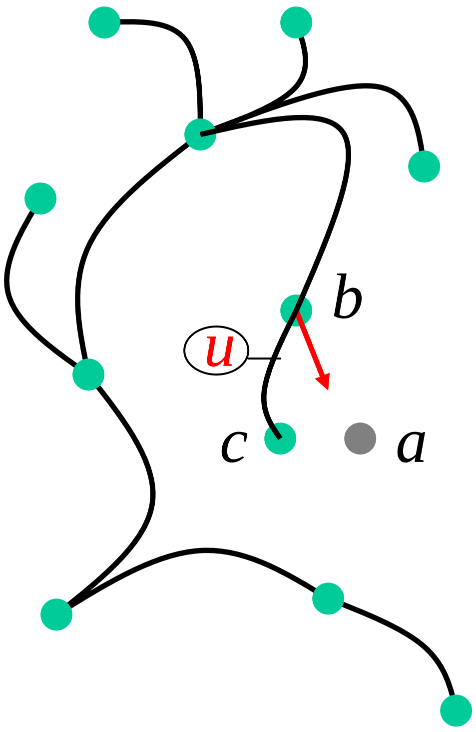
Building an RRT

- 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



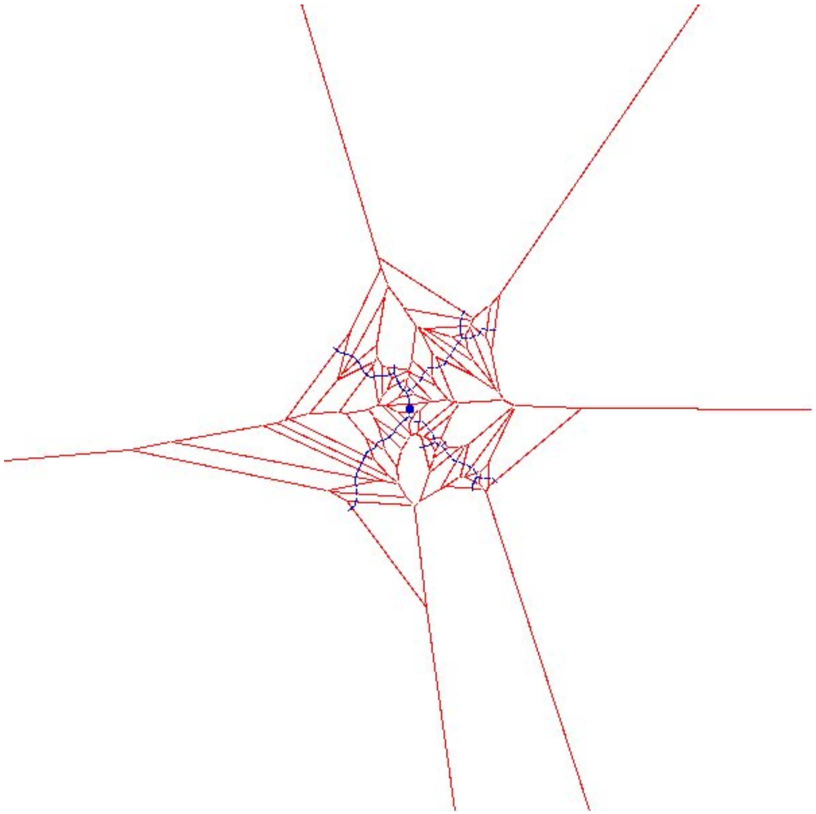
Executing the Path

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

Principle Advantage

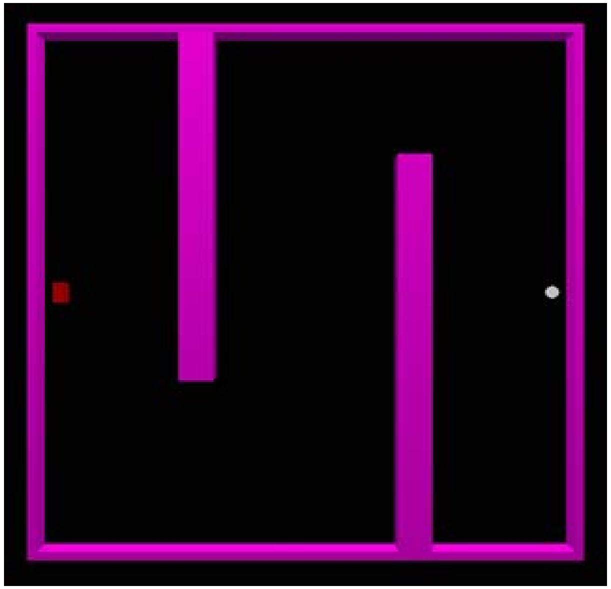
- **RRT quickly explores the state space**:
  - Nodes most likely to be expanded are those with largest Voronoi regions

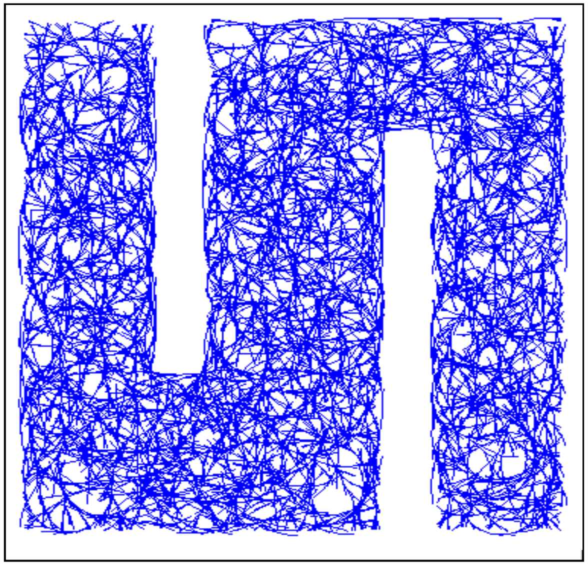


Advanced RRT Algorithms

1. Single RRT **biased towards the goal**
2. **Bidirectional** planners
3. RRT planning in **dynamic environments**

Example: Simple RRT Planner





- Problem: ordinary RRT explores *X* uniformly
  → slow convergence
- Solution: bias distribution towards the goal

Goal-biased RRT

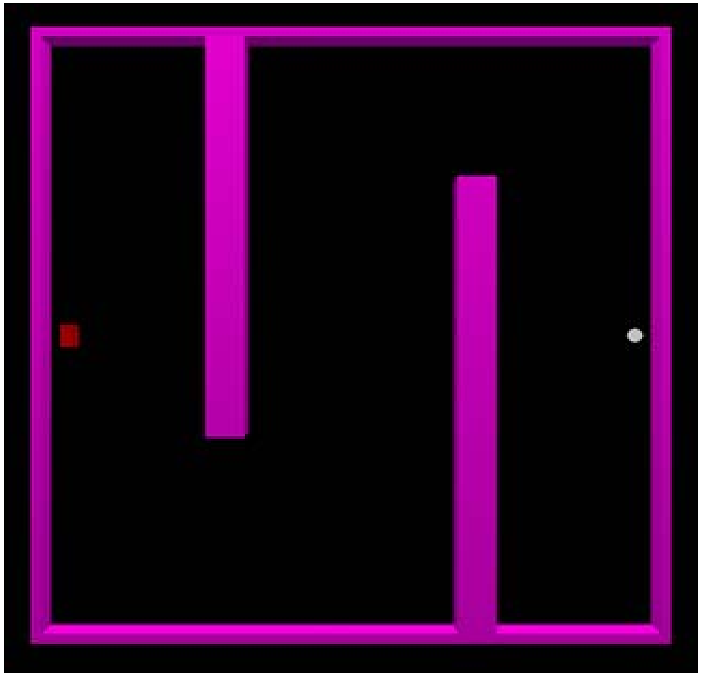
BUILD_RRT(x_{init})

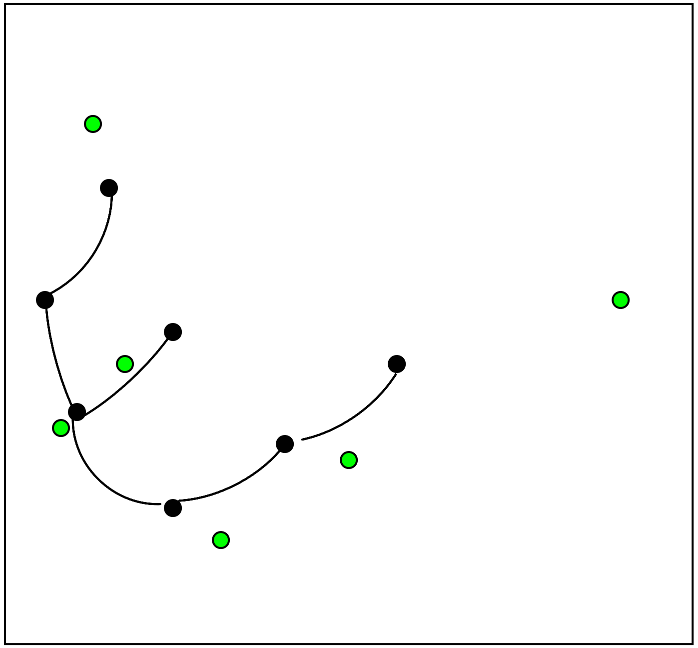
$\mathcal{T}$.init(x_{init});
for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ ~~RANDOM_STATE();~~
    EXTEND($\mathcal{T}, x_{rand}$);
Return $\mathcal{T}$

BIASED_RANDOM_STATE()

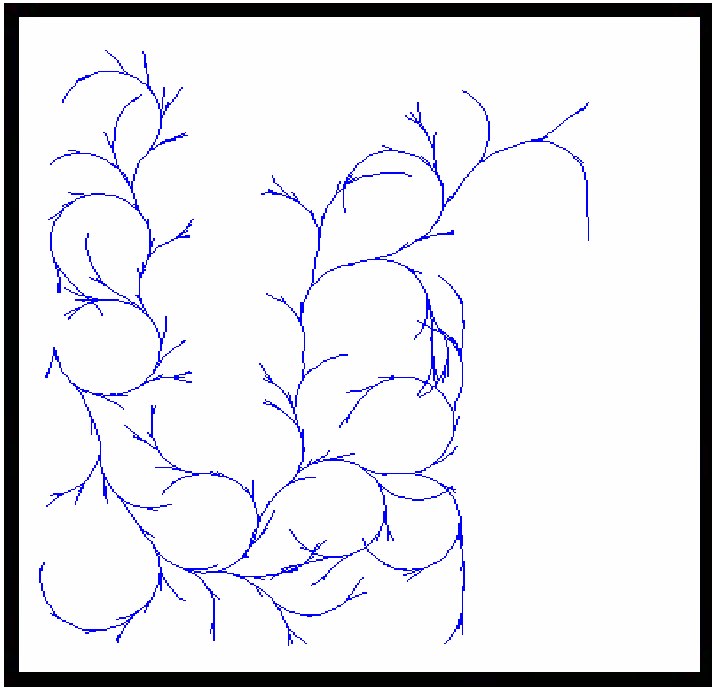
toss ← COIN_TOSS()
if *toss* = heads **then**
    Return s_{goal}
else
    Return RANDOM_STATE()

Goal-biased RRT





The world is full of… local minima



- If too much bias, the planner may get trapped in a local minimum

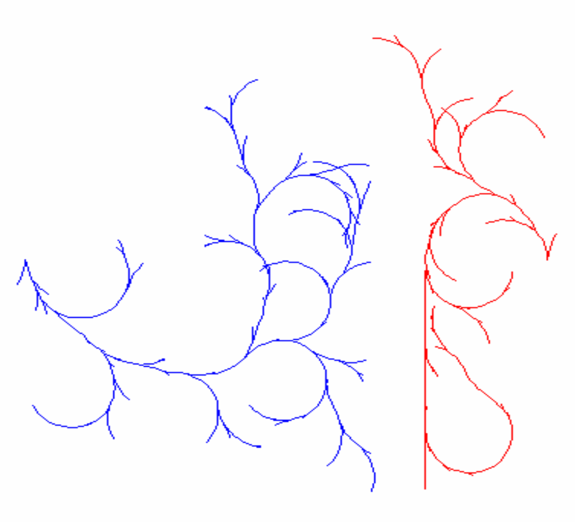
A different strategy:

- Pick RRT point near s_{goal}
- Based on distance from goal to the nearest *v* in *G*
- Gradual bias towards s_{goal}

Rather slow convergence

Bidirectional Planners

- Build **two RRTs**, from **start** and **goal** state



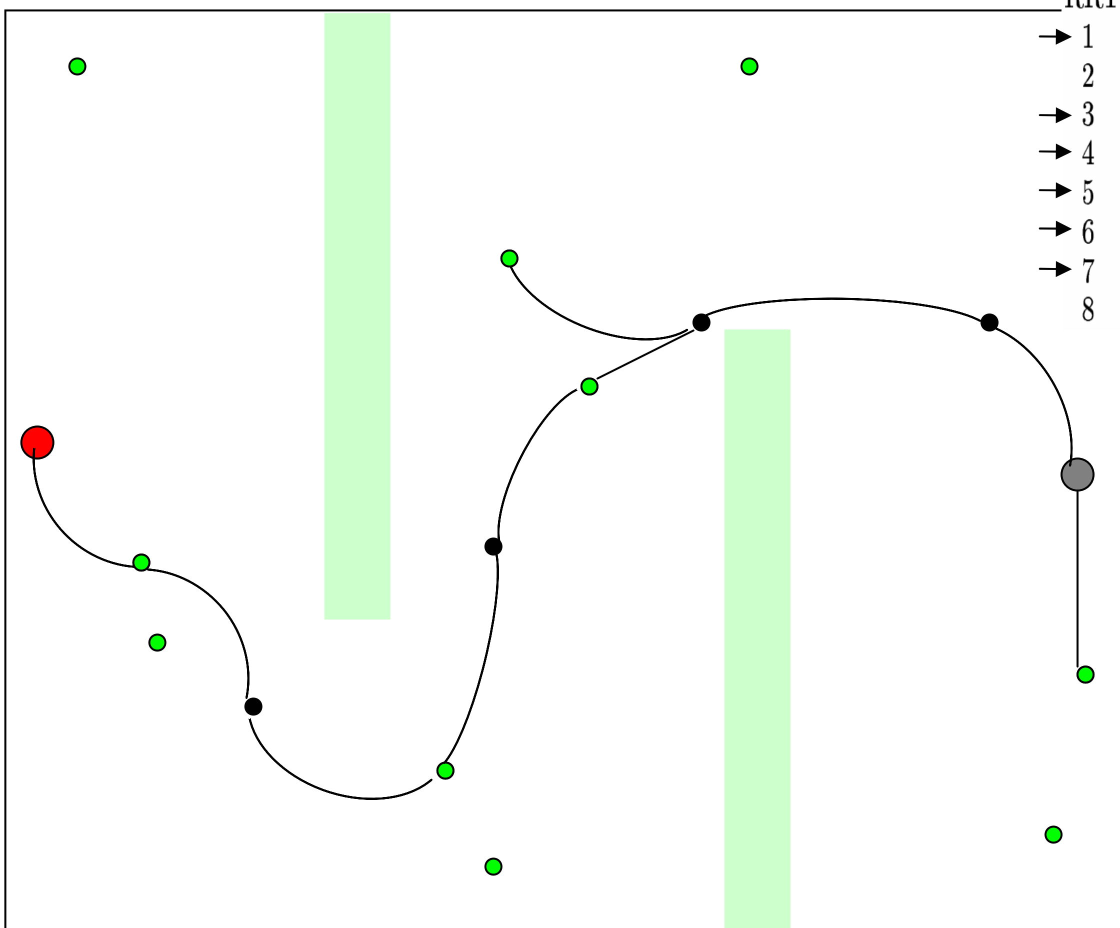
- **Complication**: need to **connect** two RRTs
  - local planner will not work (dynamic constraints)
  - bias the **distribution**, so that the **trees meet**

Bidirectional Planner Algorithm

RRT_BIDIRECTIONAL(x_{init}, x_{goal})

$\mathcal{T}_a$.init(x_{init}); $\mathcal{T}_b$.init(x_{goal});
for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    if not (EXTEND($\mathcal{T}_a, x_{rand}$) = *Trapped*) **then**
        if (EXTEND($\mathcal{T}_b, x_{new}$) = *Reached*) **then**
            Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
    SWAP($\mathcal{T}_a, \mathcal{T}_b$);
Return *Failure*

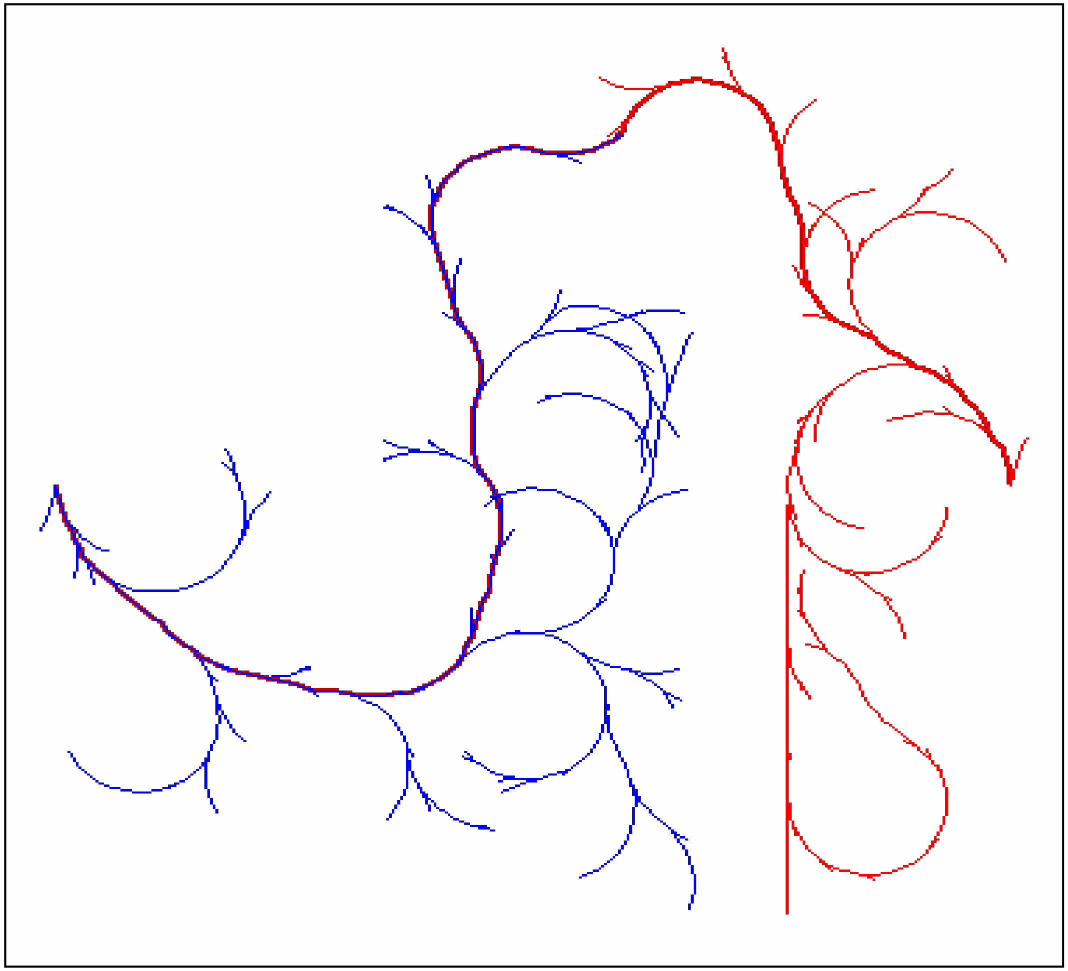
Bidirectional Planner Example



RRT_BIDIRECTIONAL(x_{init}, x_{goal})

$\mathcal{T}_a$.init(x_{init}); $\mathcal{T}_b$.init(x_{goal});
for $k = 1$ **to** K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE();
    if not (EXTEND$(\mathcal{T}_a, x_{rand})$ = *Trapped*) **then**
        if (EXTEND$(\mathcal{T}_b, x_{new})$ = *Reached*) **then**
            Return PATH$(\mathcal{T}_a, \mathcal{T}_b)$;
    SWAP$(\mathcal{T}_a, \mathcal{T}_b)$;
Return *Failure*

Bidirectional Planner Example



Conclusions

- **Path planners** for **real-world robots** must account for **dynamic constraints**
- **Building** the roadmap tree **incrementally**
  - ensures that the **kinodynamic constraints** are **satisfied**
  - avoids the need to **reconstruct control inputs** from the path
  - allows **extensions** to **moving obstacles** problem

Conclusions

- MOP and RRT planners are similar
- Well-suited for single-query problems
- RRTs benefit from the ability to steer a robot toward a point
  - RRTs explore the state more uniformly
  - RRTs can be biased towards a goal or to grow into another RRT