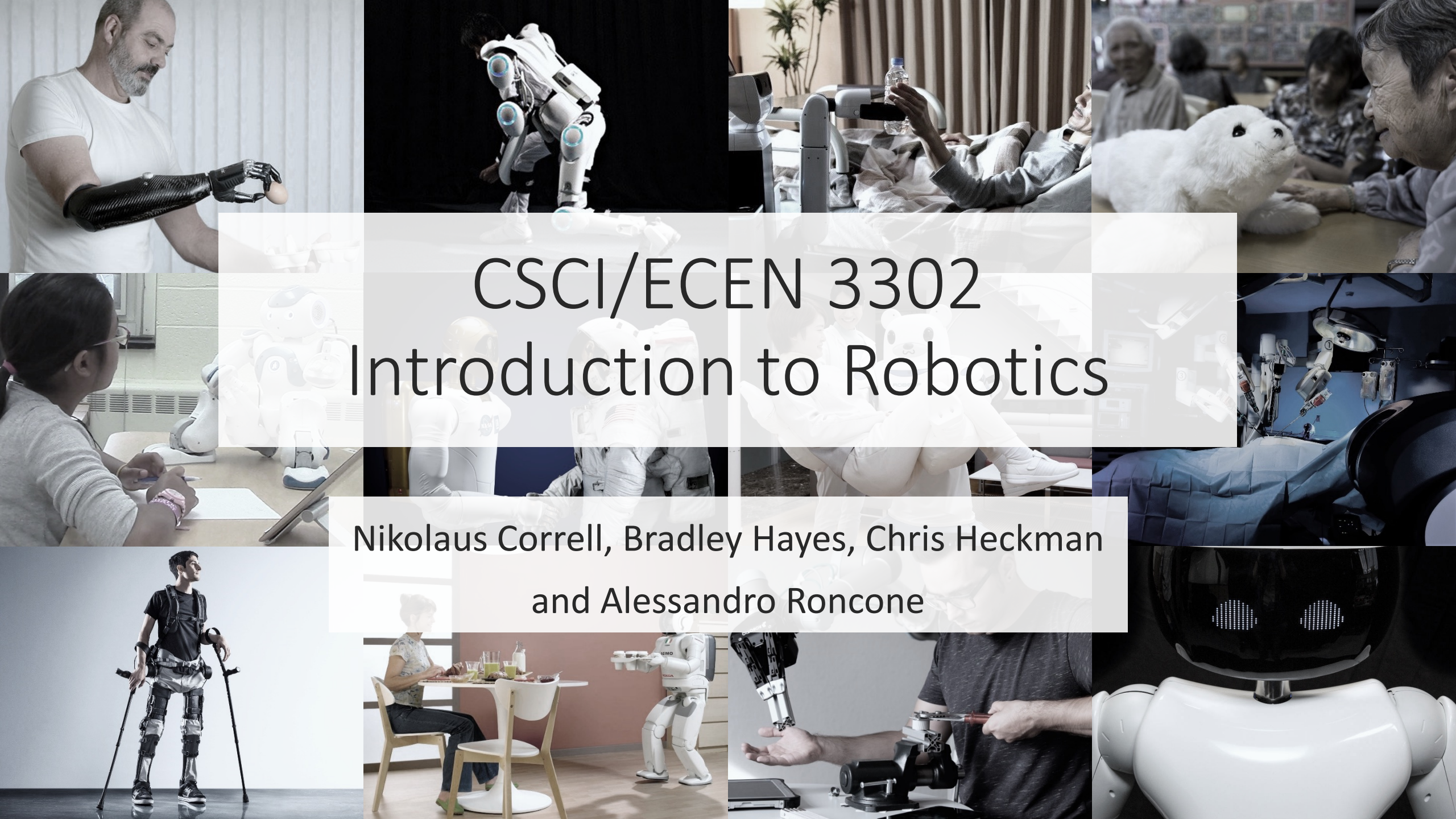


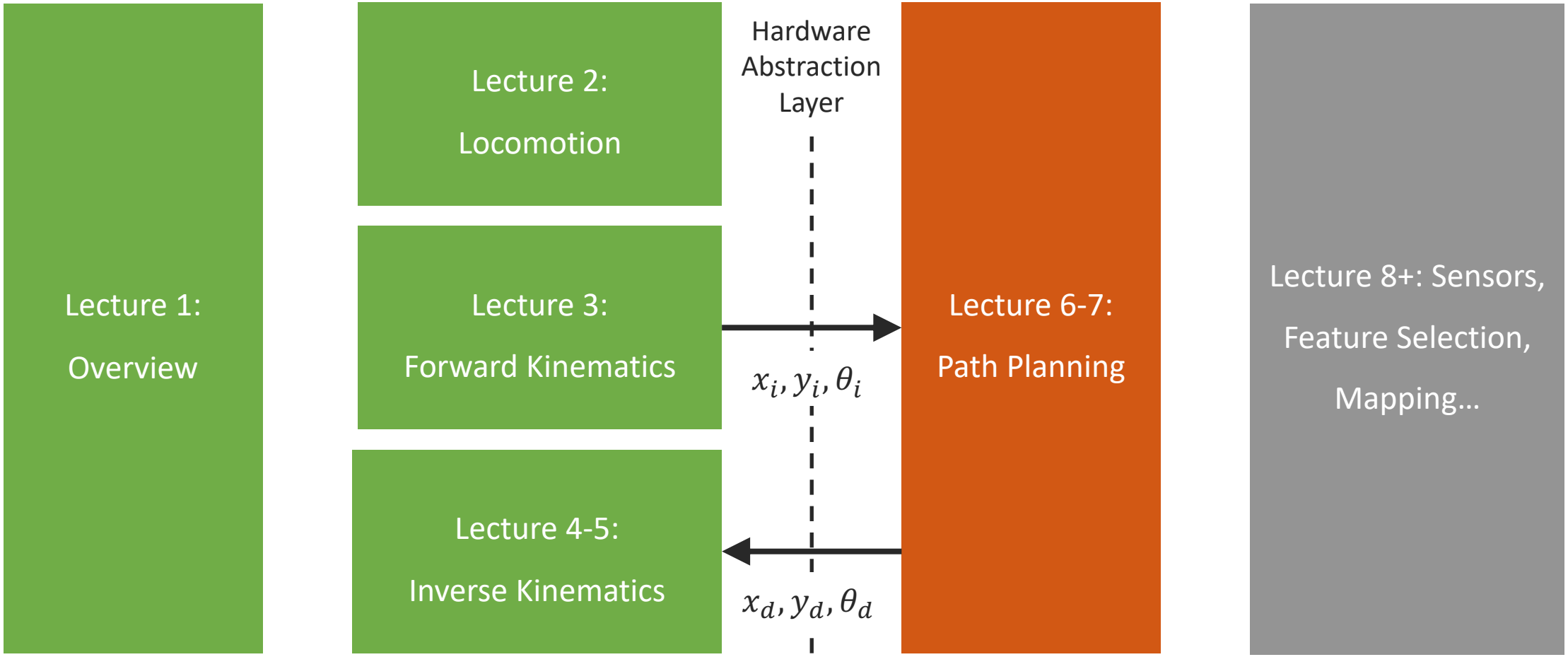
CSCI/ECEN 3302
Introduction to Robotics

Nikolaus Correll, Bradley Hayes, Chris Heckman
and Alessandro Roncone

Administrivia

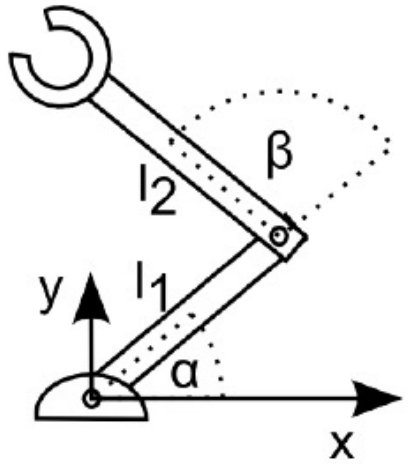
- Lab 3: Inverse Kinematics – Lecture 5 content
  - Deadline October 4th
- Lab 4: Mapping – Lecture 6 (today) content
  - One week long
- Homework 2 is out!

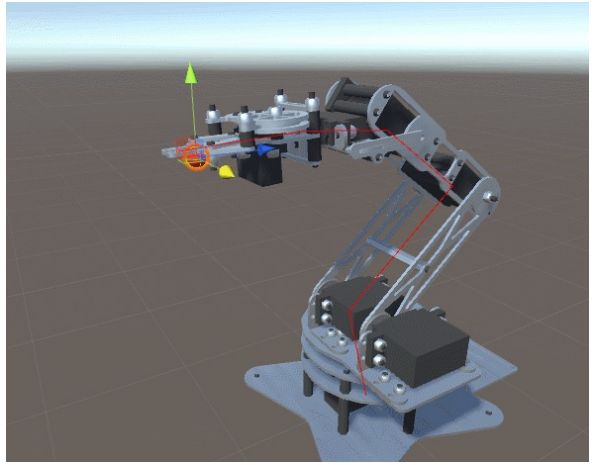
Roadmap


Lecture 1:
Overview
Lecture 2:
Locomotion
Lecture 3:
Forward Kinematics
Lecture 4-5:
Inverse Kinematics
Hardware
Abstraction
Layer
x_i, y_i, θ_i
x_d, y_d, θ_d
Lecture 6-7:
Path Planning
Lecture 8+: Sensors,
Feature Selection,
Mapping…

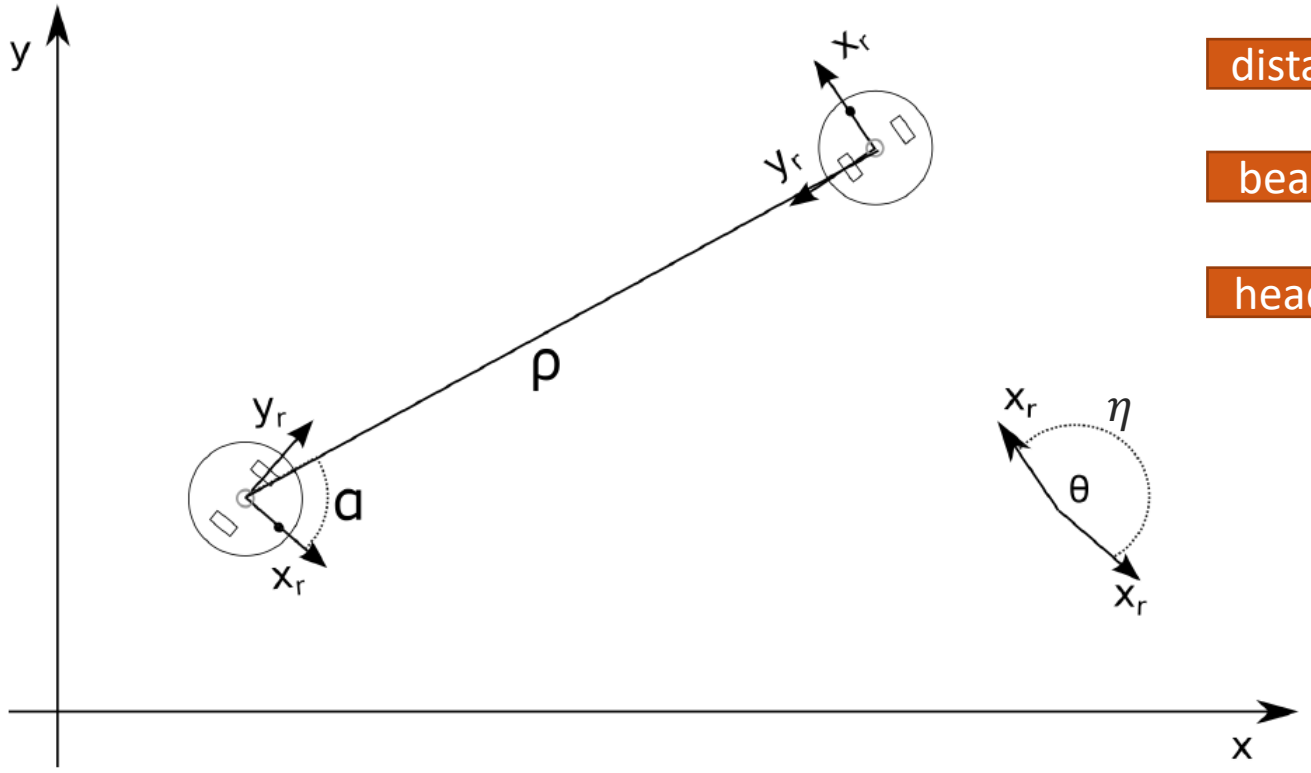
Chapters 12-13

RECAP
Motion and Trajectory Planning





Position Change Using Feedback Control



Error Terms

distance $\rho = \sqrt{(x_r - x_g)^2 + (y_r - y_g)^2}$

bearing $\alpha = \tan^{-1}\left(\frac{y_g - y_r}{x_g - x_r}\right) - \theta_r$

heading $\eta = \theta_g - \theta_r$

Update Rules

translation $\dot{x} = p_1\rho$

rotation $\dot{\theta} = p_2\alpha + p_3\eta$

p_1, p_2, p_3 are controller gains

Jacobian: relate joint to operational velocities

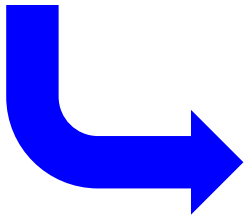
$$\begin{bmatrix} \dot{x}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} \frac{r\dot{\phi}_l}{2} + \frac{r\dot{\phi}_r}{2} \\ \frac{\dot{\phi}_r r}{d} - \frac{\dot{\phi}_l r}{d} \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ -\frac{r}{d} & \frac{r}{d} \end{bmatrix} \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix} = \begin{bmatrix} \frac{\partial \dot{x}_R}{\partial \dot{\phi}_l} & \frac{\partial \dot{x}_R}{\partial \dot{\phi}_r} \\ \frac{\partial \dot{\theta}_R}{\partial \dot{\phi}_l} & \frac{\partial \dot{\theta}_R}{\partial \dot{\phi}_r} \end{bmatrix} \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix}$$

$$\frac{dp_e}{dt} = \boldsymbol{J} \frac{d\boldsymbol{q}}{dt}$$

Inverse kinematics

$$\begin{bmatrix} \dot{x}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ -\frac{r}{d} & \frac{r}{d} \end{bmatrix} \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

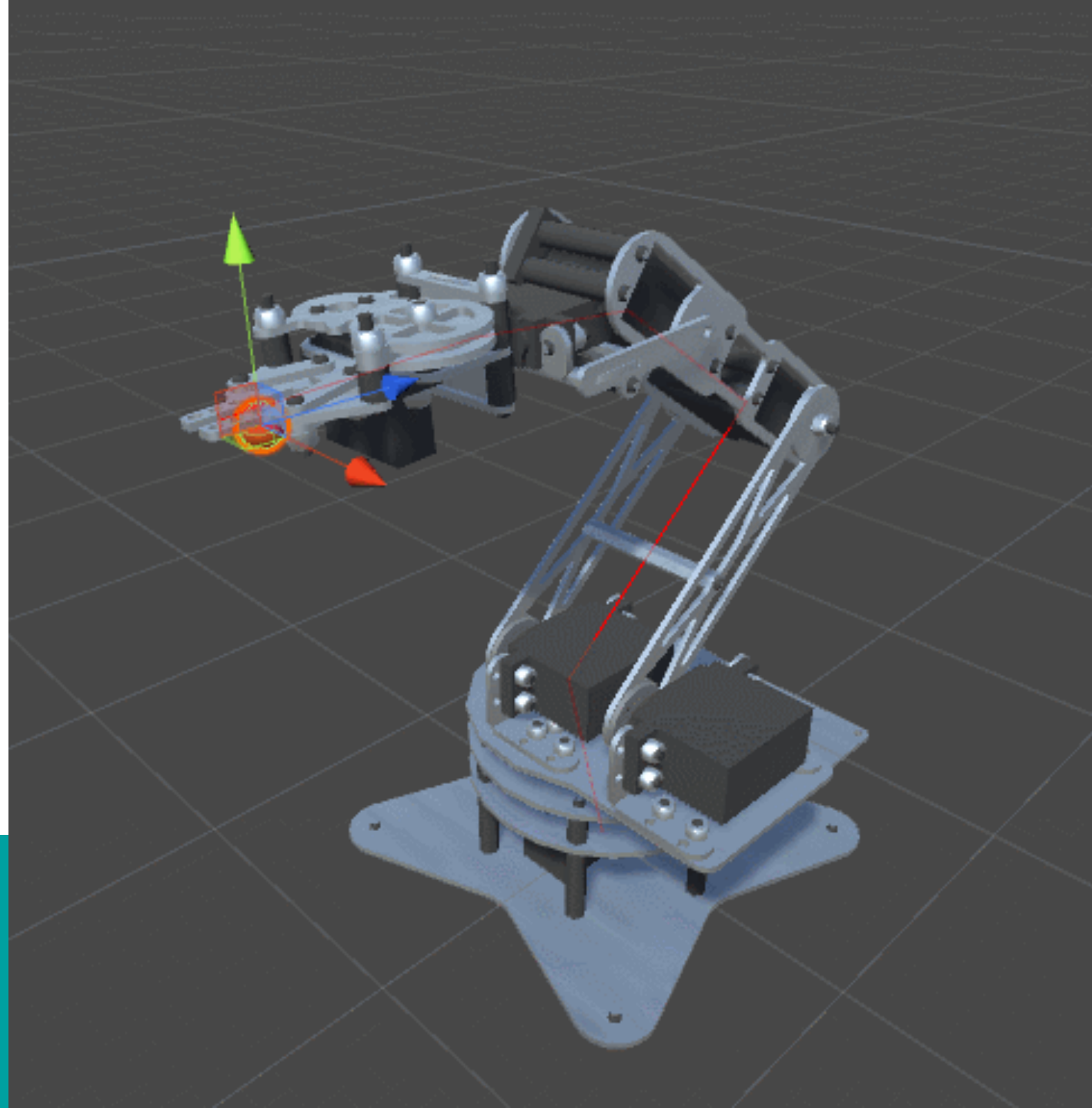


$$\begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ -\frac{r}{d} & \frac{r}{d} \end{bmatrix}^{-1} \begin{bmatrix} \dot{x}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -d/2 \\ 1 & d/2 \end{bmatrix} \begin{bmatrix} \dot{x}_R \\ \dot{\theta}_R \end{bmatrix}$$

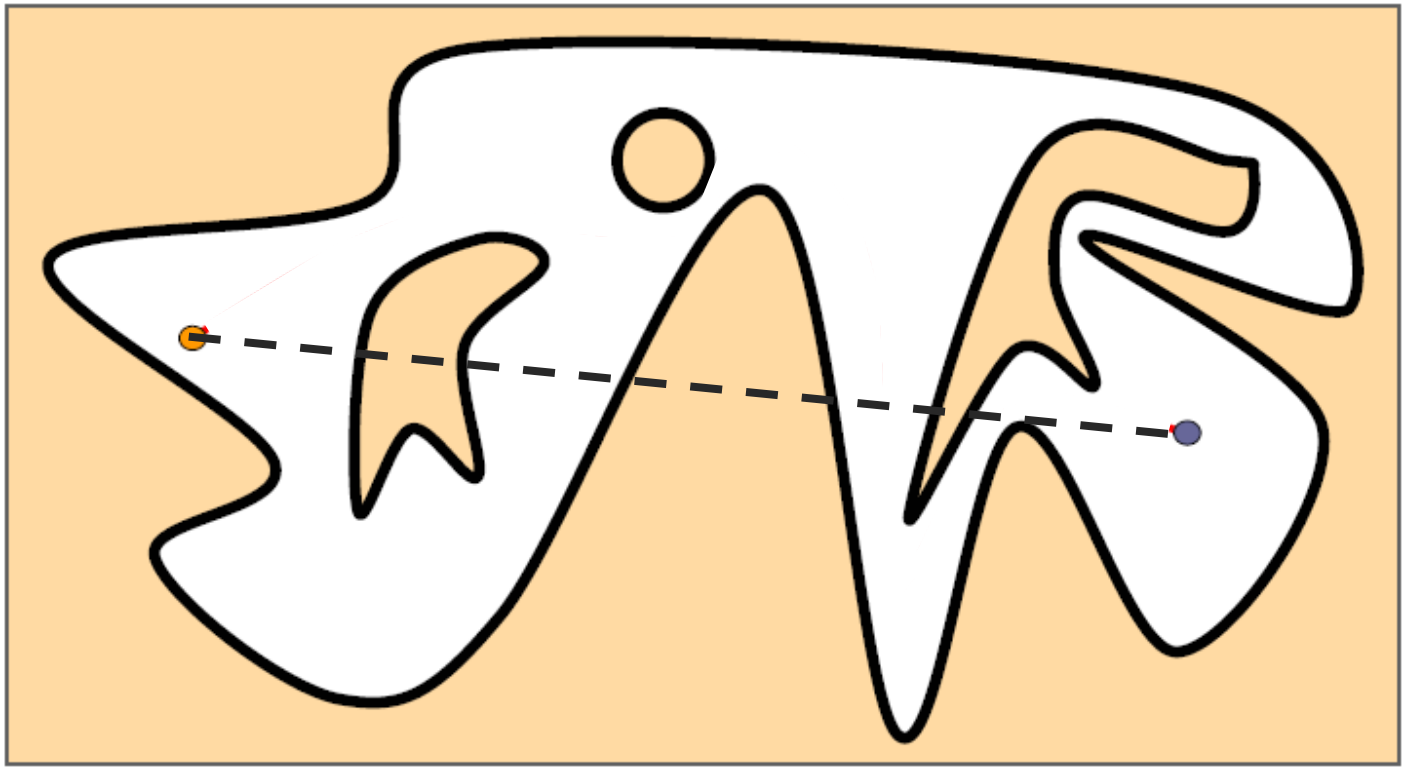
Why we need planning?

Joint Angle Limitation Problems!

How do we fix this?



Why we need planning? Obstacles.



Using distance and angle as ”error” is not sufficient anymore.

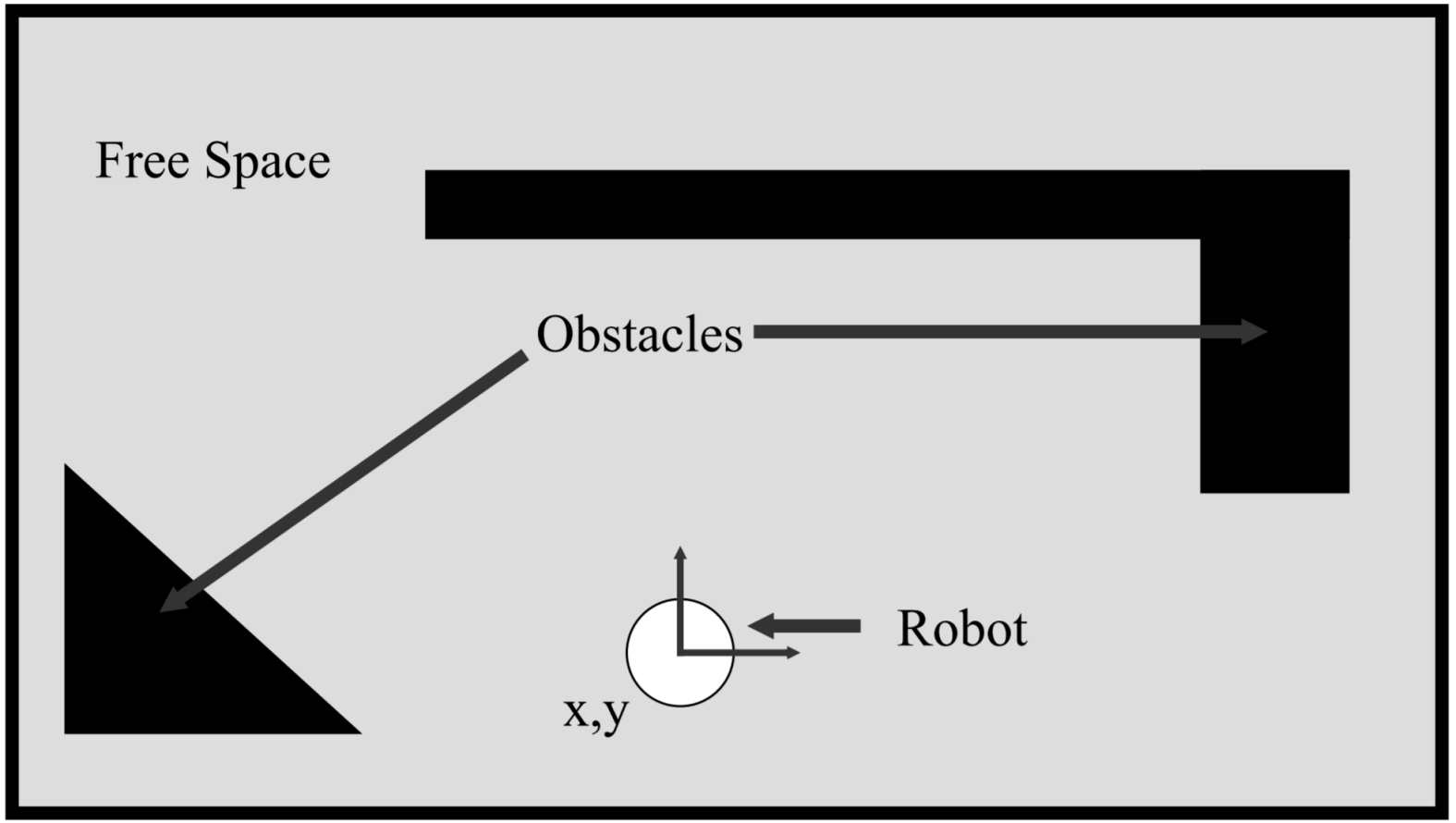
Motion Planning

- **Goal: Find a trajectory in configuration space from one point to another**
  - A trajectory is usually denoted as $\xi: t \in [0, T] \rightarrow C$
  - *(A function mapping time to robot configurations)*

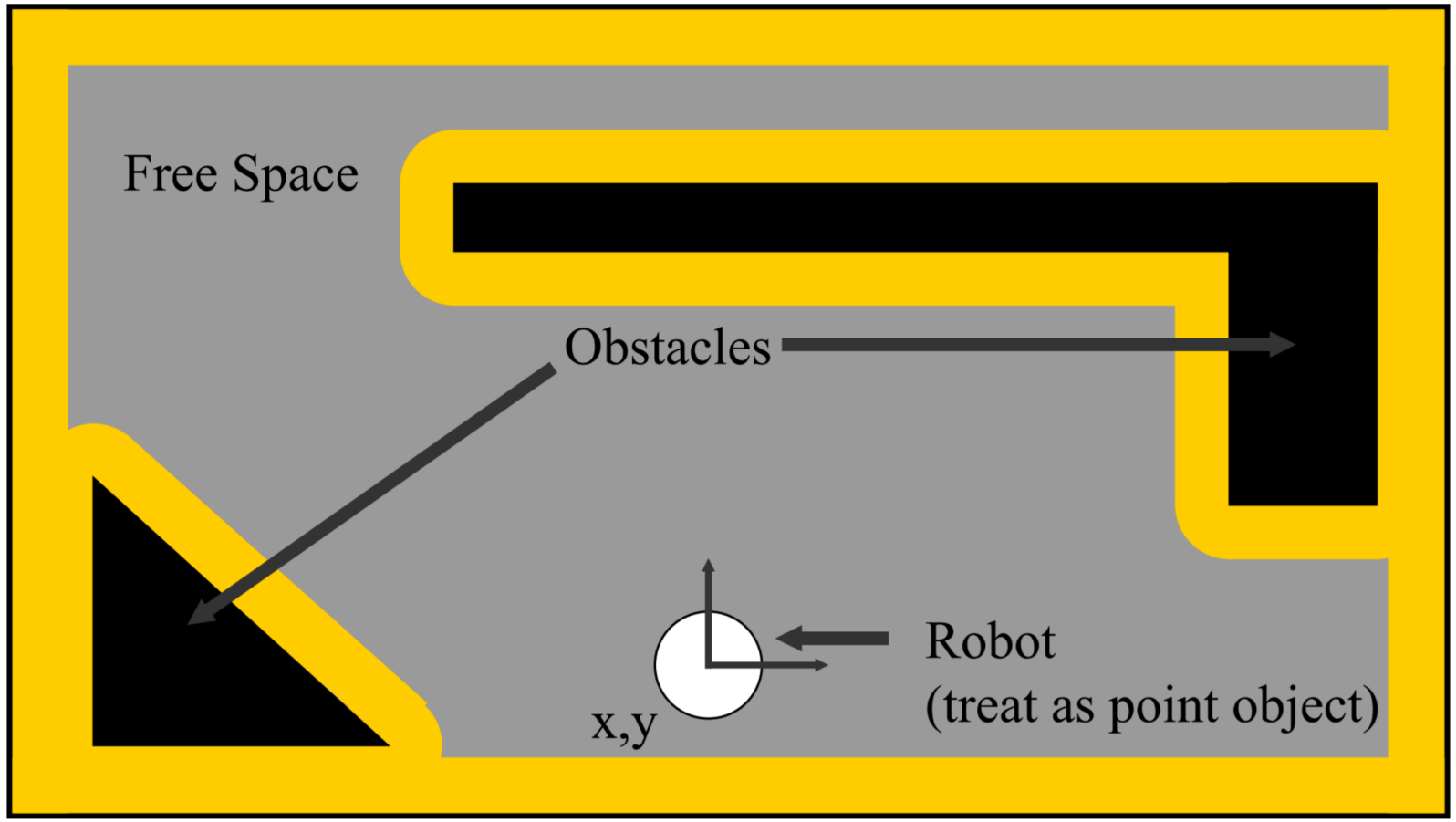
Trajectory: execution of path r over time $r(t)$

- The Bad News:
  - All complete search algorithms scale exponentially!!
  - Motion planning problems are high dimensional, e.g., big exponent.

Configuration Space (operational space)



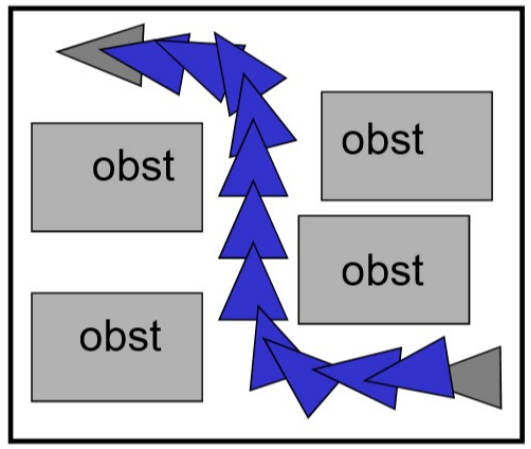
Configuration Space



Configuration Space

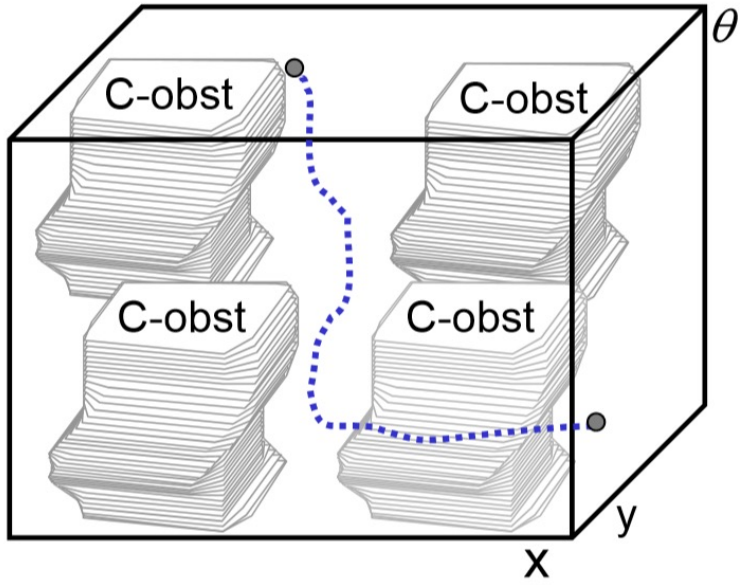
Each step has to check for collisions between the robot's body and obstacles!

Workspace
(*x*, *y*)



Robot

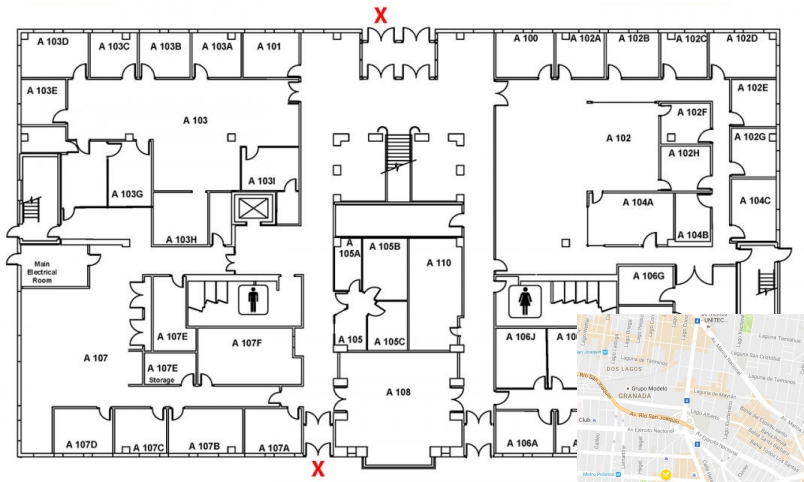
C-space
(*x*, *y*, θ)

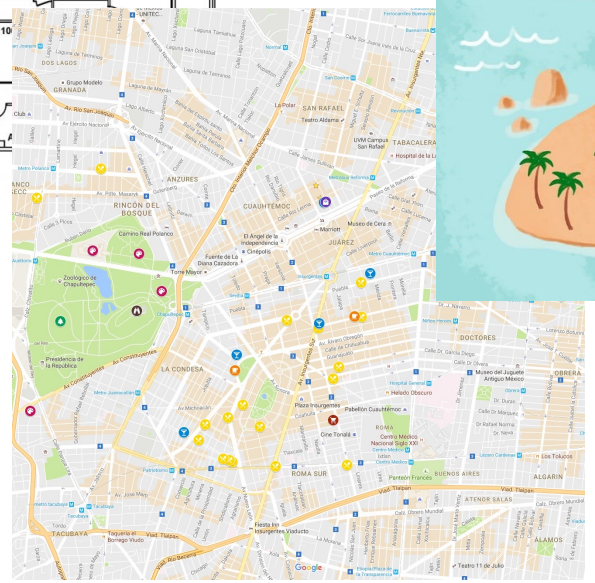


Robot

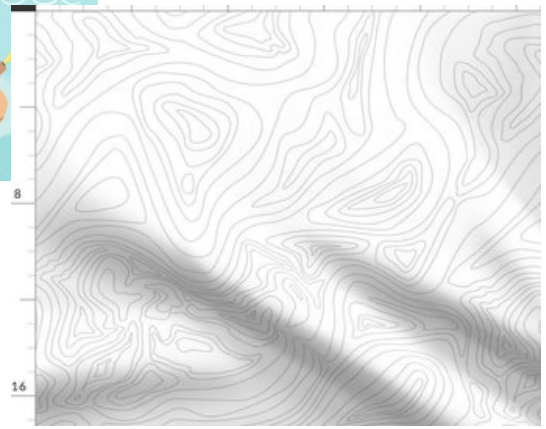
The robot's path is just a path through space – as a single point, collision checks are easy!

Maps


X
A 103D
A 103C
A 103B
A 103A
A 101
A 103E
A 103
A 103I
A 103G
A 103H
Main Electrical Room
A 107E
A 107F
A 107
A 107E Storage
A 107D
A 107C
A 107B
A 107A
A 105A
A 105B
A 110
A 105
A 105C
A 108
A 100
A 102A
A 102B
A 102C
A 102D
A 102E
A 102F
A 102
A 102H
A 102G
A 104A
A 104C
A 104B
A 106G
A 106J
A 106A
X


UNITEC
DOS LAGOS
GRANADA
SAN RAFAEL
TABACALERA
ANZURES
RINCON DEL BOSQUE
CUAUHTEMOC
JUAREZ
DOCTORES
LA CONDESA
ROMA SUR
BUENOS AIRES
ATENOR SALAS
ALGARIN
TACUBAYA
ALAMOS
Google

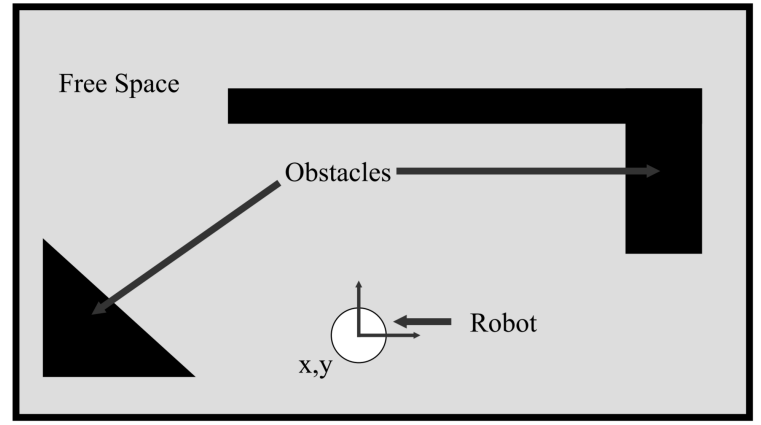

Balloontopia
X
Puzzle Palace
Zany Zoo
Baking Boulevard

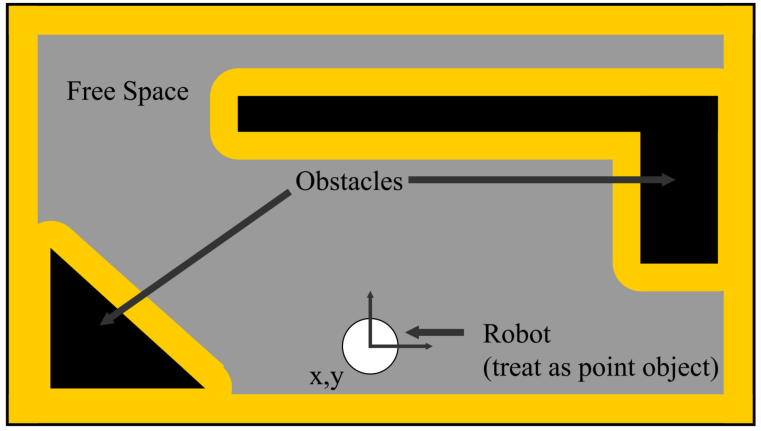

8
16

Questions?

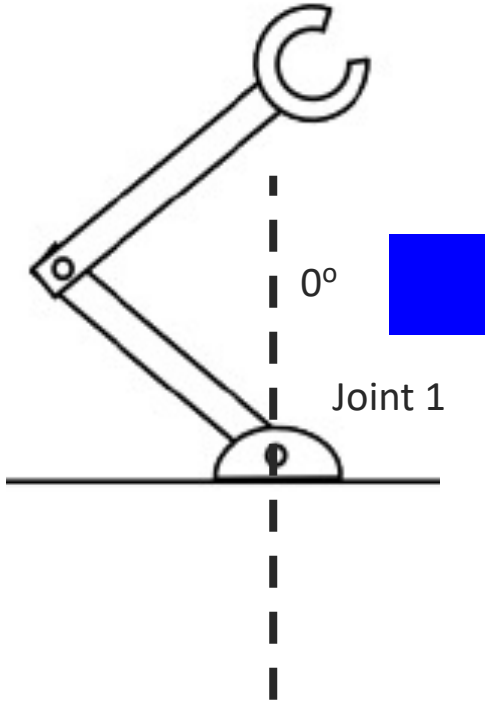
In practice…

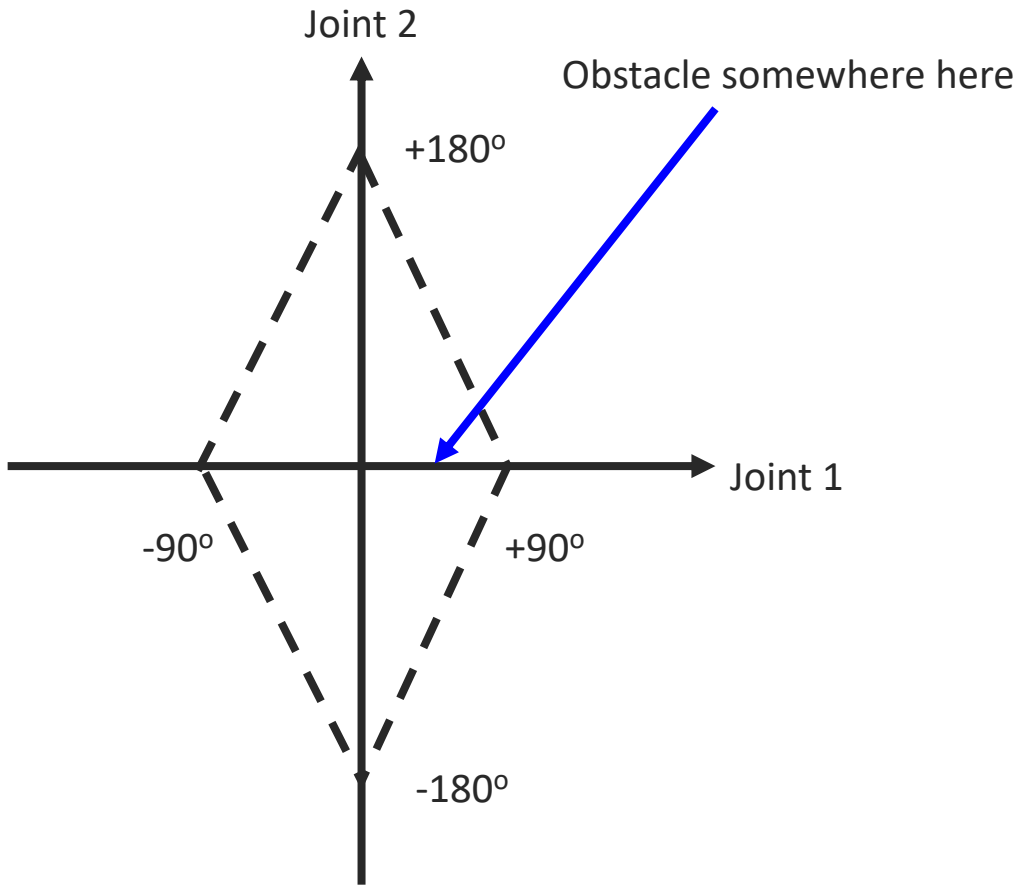
- Optimally, maps are in configuration space
- In practice: combination of c-space representation and on-the-spot checking





Example: C-Space for Robotic Arm





C-Space without obstacles

Chapter 4

Module 2 – COMPUTATION

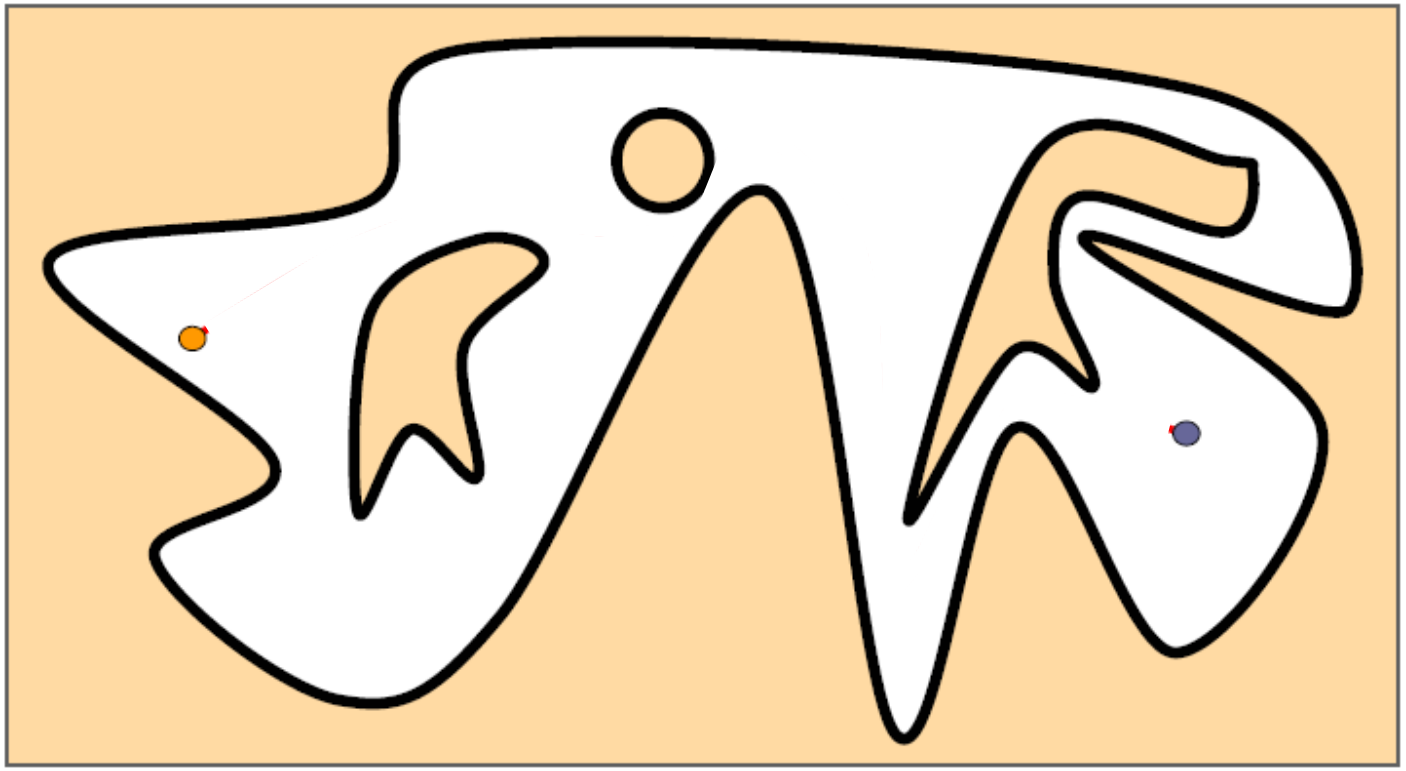
Part II –World representations

Objective: Compute a collision-free path for a mobile robot among static obstacles

- Inputs required
  - Geometry of the robot, any obstacles in the environment
  - Kinematics of the robot (DoF, Joint links and types)
  - Initial and goal robot configurations (positions & orientations)

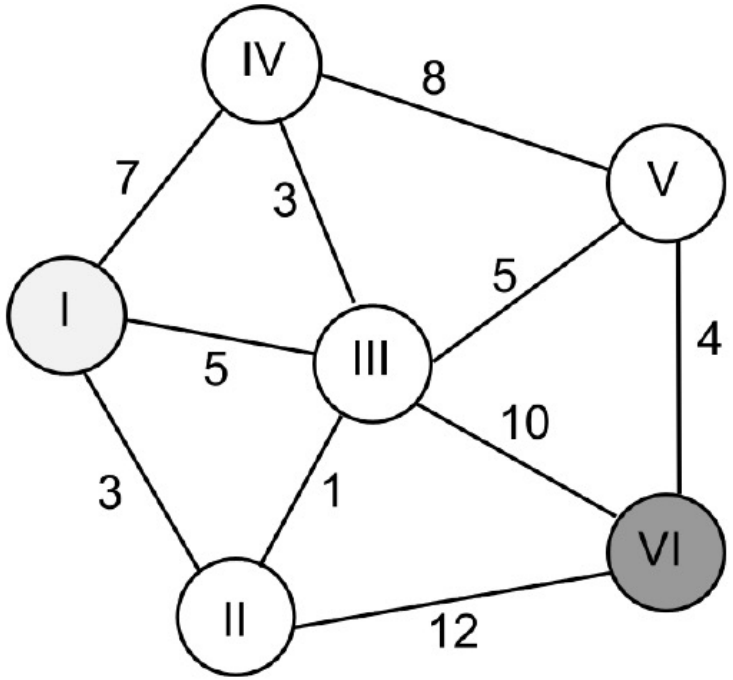
- Expected Result
  - Continuous sequence of collision-free robot configurations
  - The sequence connects the initial and goal configurations
  - We can follow this sequence of configurations to traverse the path

Moving from Start to Goal state

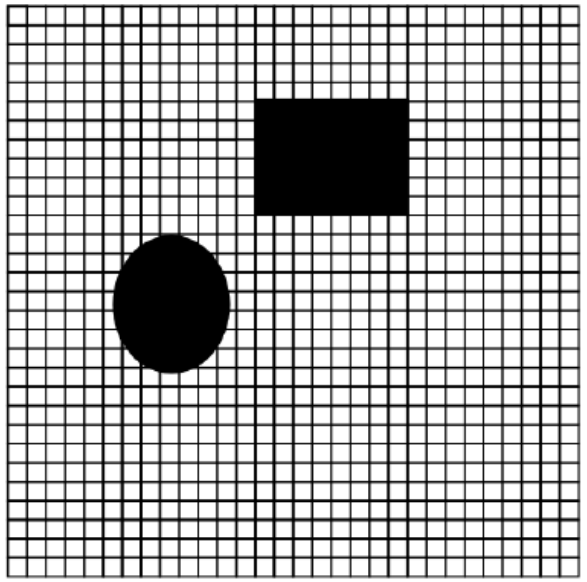


- General approach: Reduce what amounts to an intractable problem in continuous C-space to a tractable problem in a discrete space.
- Tools:
  - Environment Representations
  - Search Algorithms (next lectures)

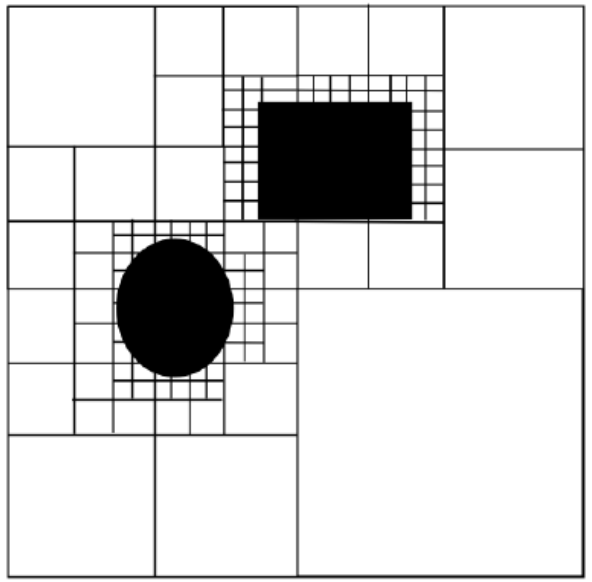
Map Representations



Topological Map
(Graph)



Grid Map
(Discrete Coordinates)

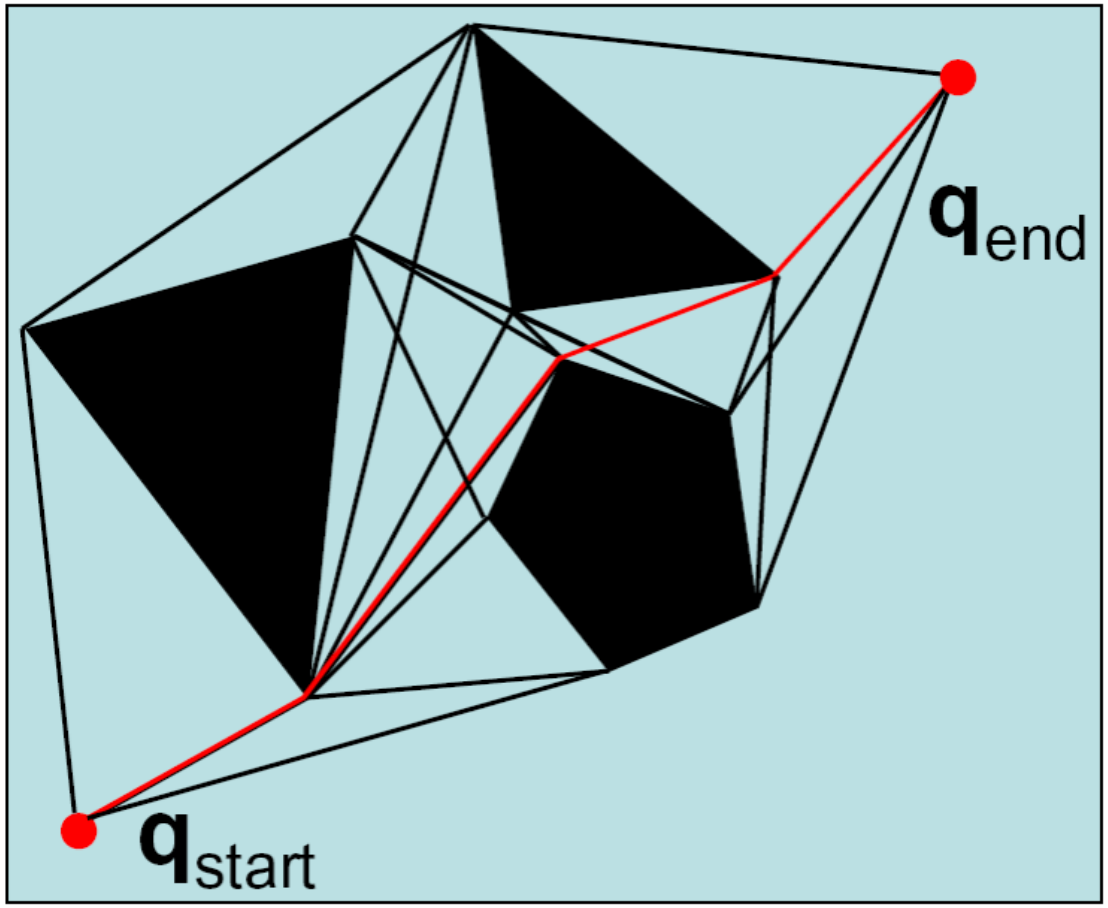


K-d Tree
(Quadtree)

Planning across length scales

Some Well-Known Representations

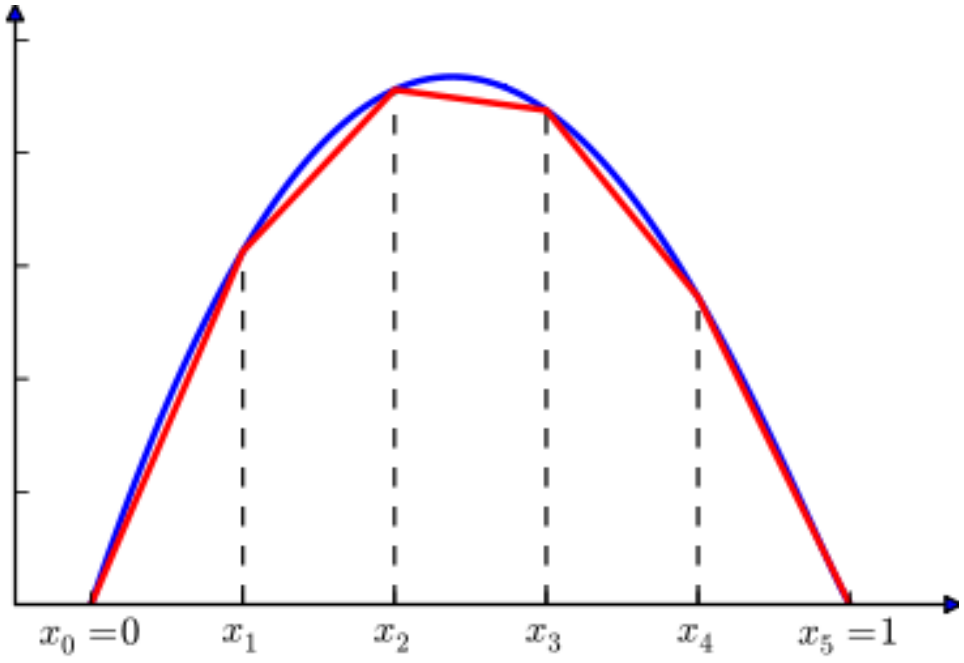
- **Visibility Graphs**
- Cell Decomposition
- Probabilistic Road Map



Visibility Graph Method

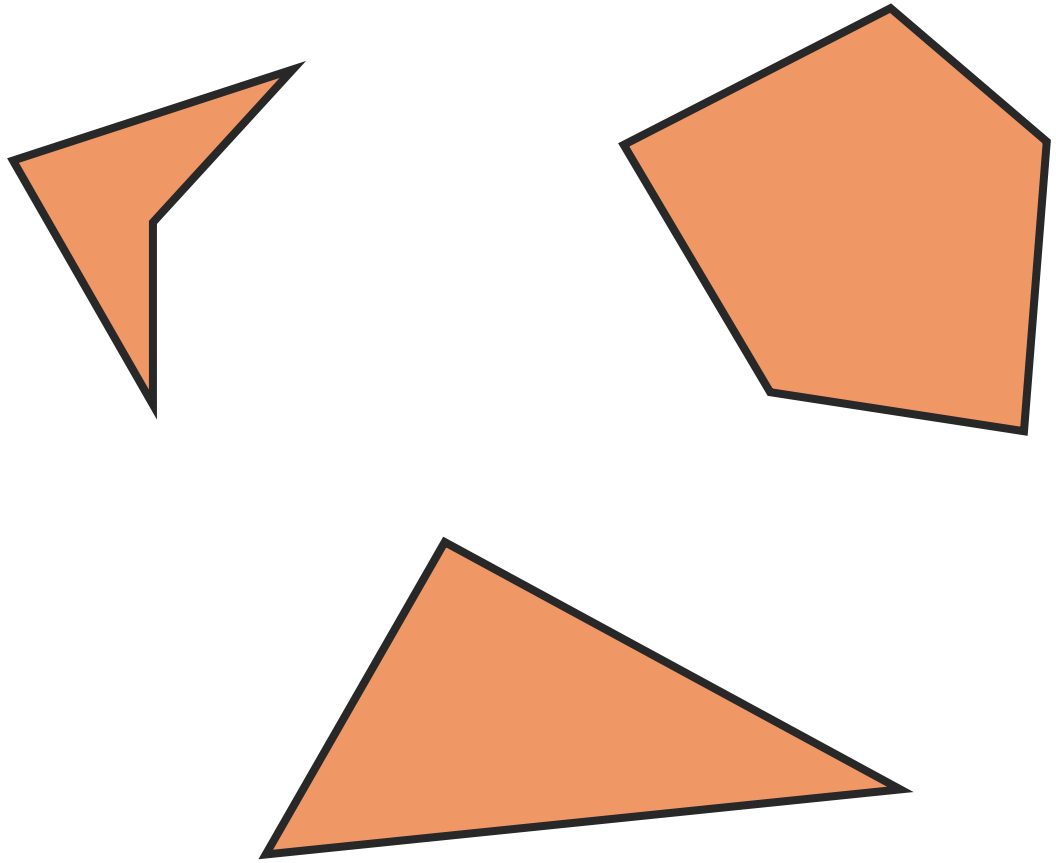
- If there is a collision-free path between two points, then there is a polygonal path that bends only at the obstacles vertices.

- A polygonal path is a piecewise linear curve:



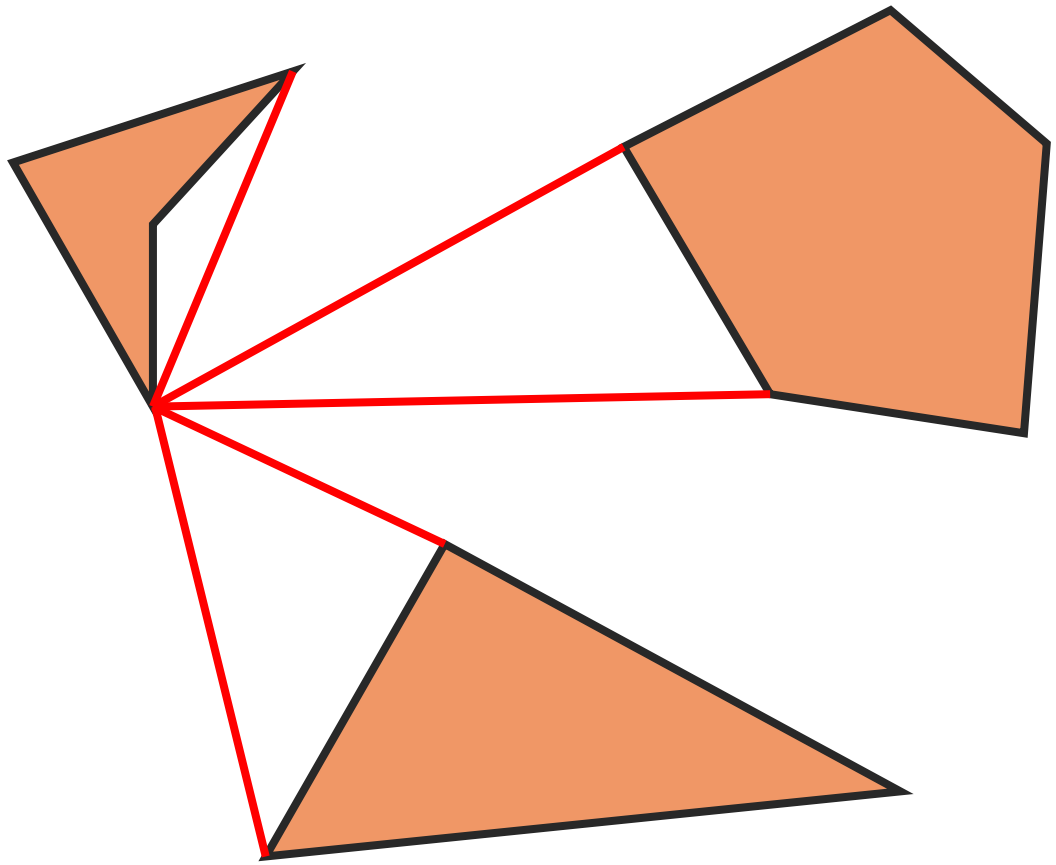
Visibility Graph Method

- Create edges to all polygon corners that are “visible” from each other



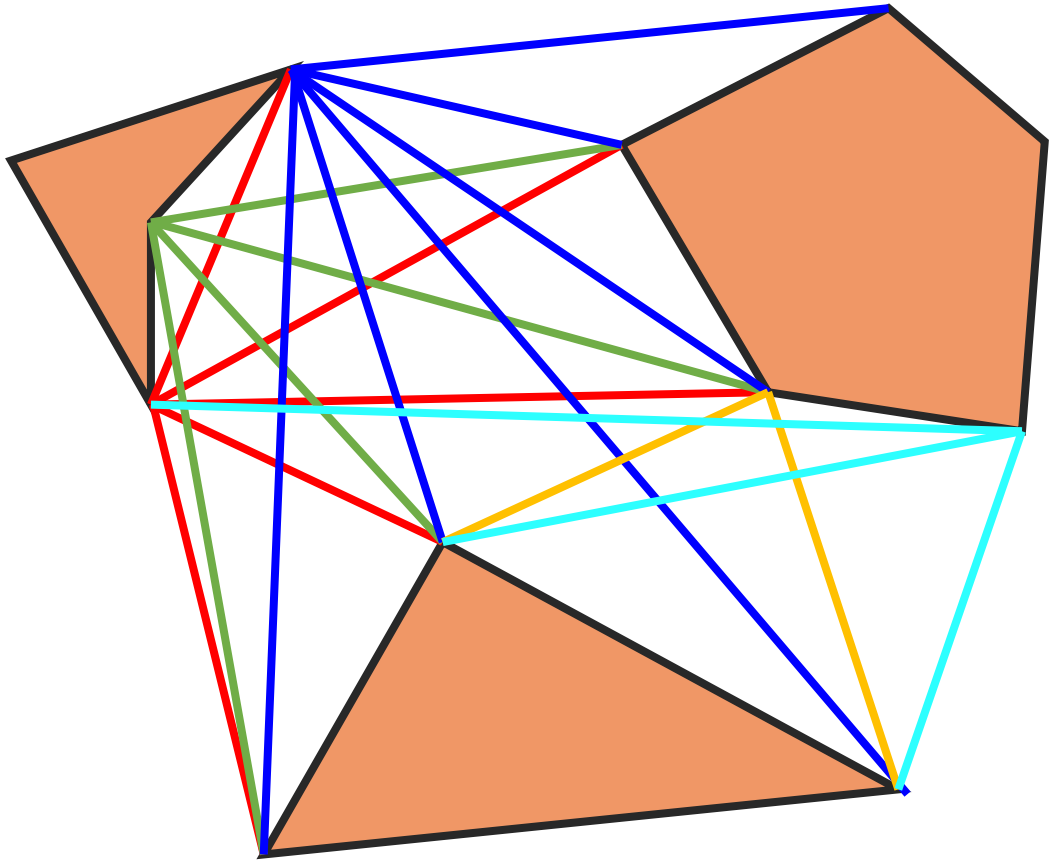
Visibility Graph Method

- Create edges to all polygon corners that are “visible” from each other
- Include edges along polygons



Visibility Graph Method

- Create edges to all polygon corners that are “visible” from each other
- Include edges along polygons



Visibility Graph

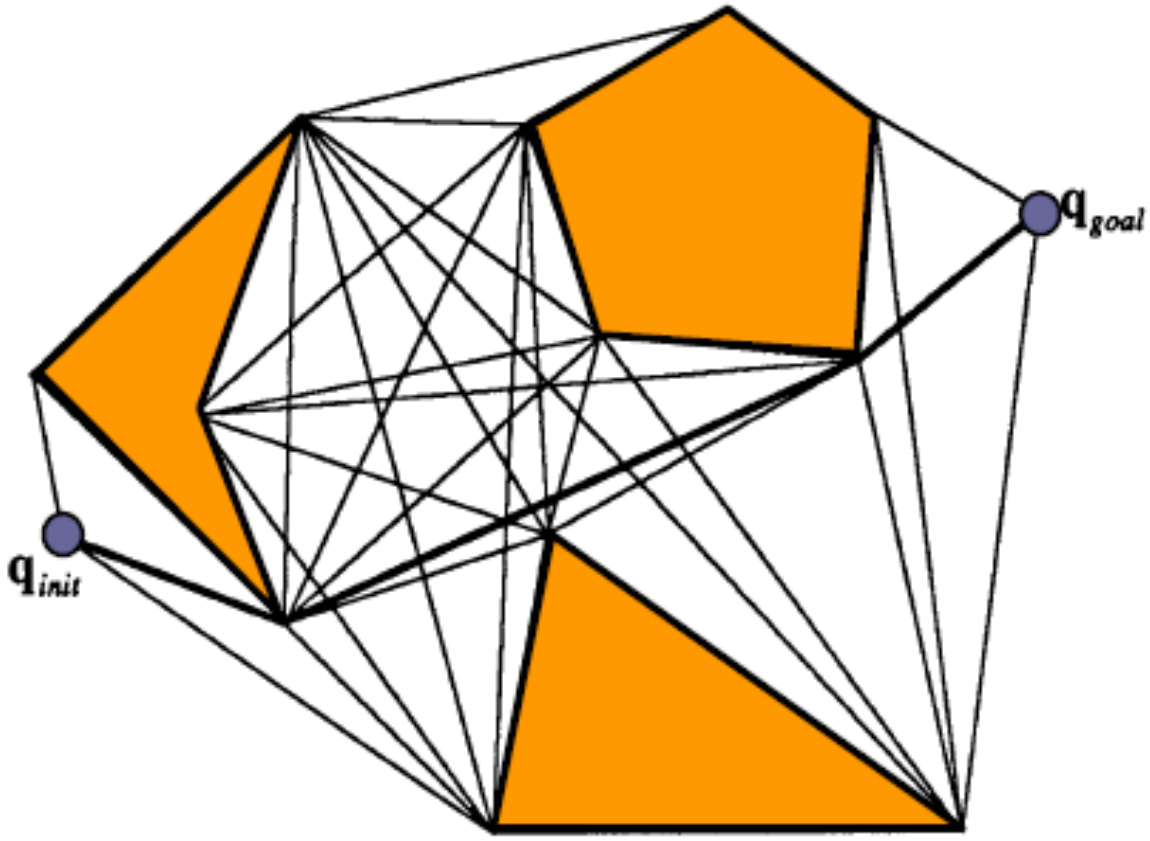
- A visibility graph is a graph such that
  - Nodes: q_{init}, q_{goal}, or an obstacle vertex.
  - Edges: An edge exists between nodes u and v if the line segment between u and v is an obstacle edge or it does not intersect the obstacles.

Visibility Graph Algorithm

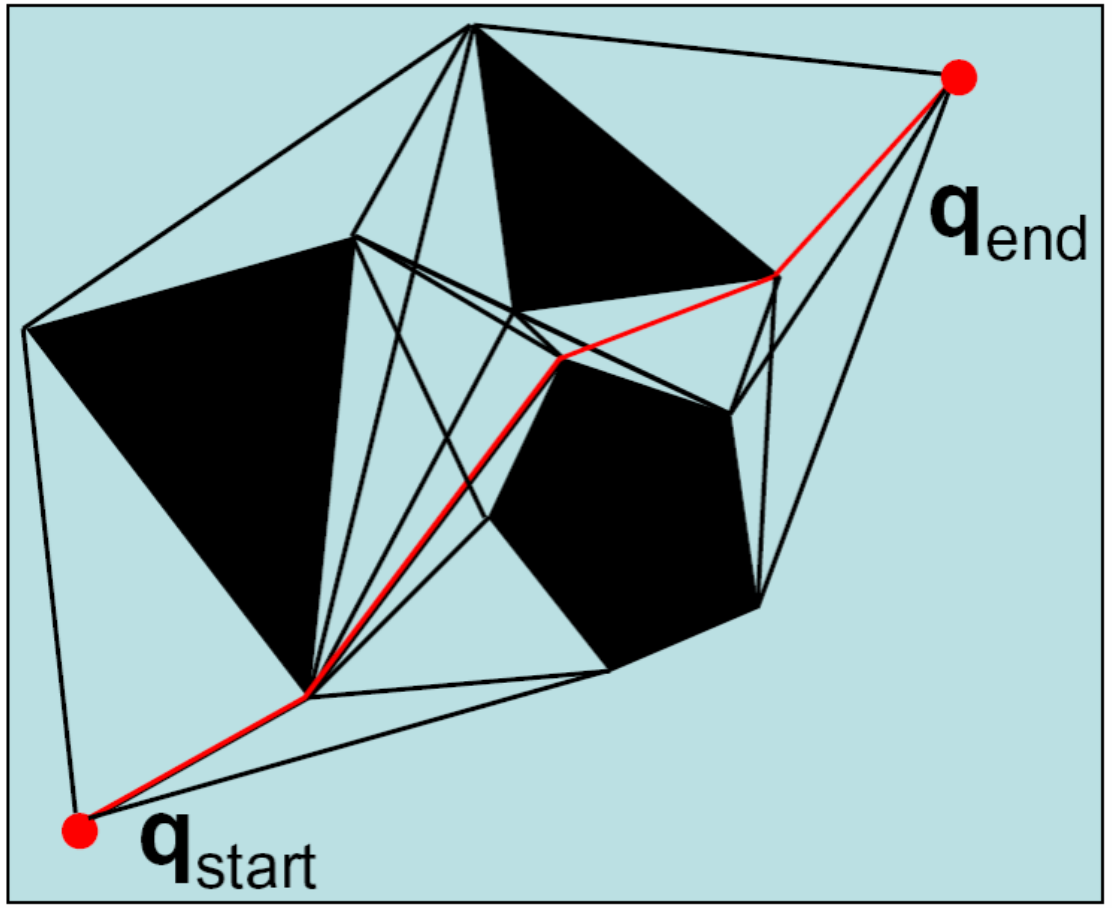
Input: q_{init}, q_{goal}, polygonal obstacles

Output: visibility graph G

```
1.    for every pair of nodes u,v
2.      if segment (u,v) is an obstacle edge then
3.        insert edge (u,v) into G;
4.      else
5.        for every obstacle edge e
6.          if segment (u,v) intersects e
7.            go to (1);
8.        insert edge (u,v) into G.
```

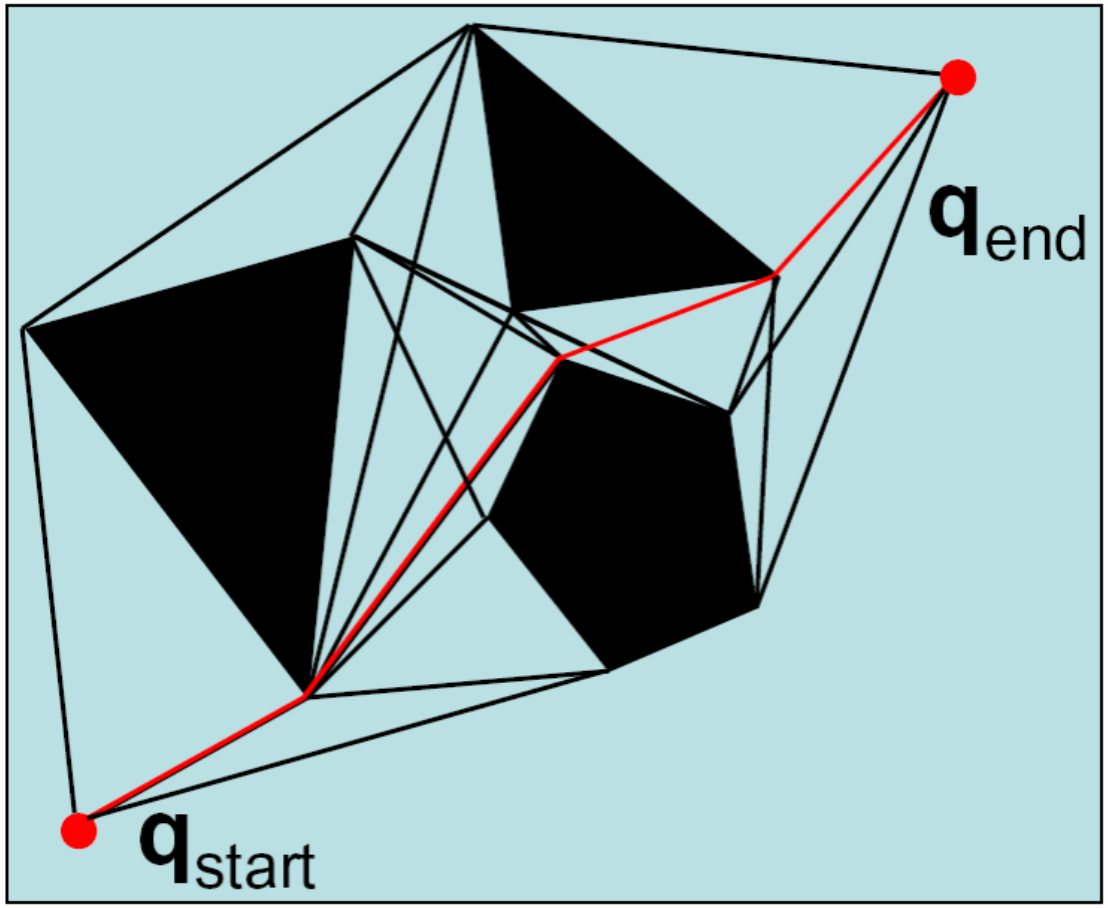


Visibility Graph: Strengths/Weaknesses?

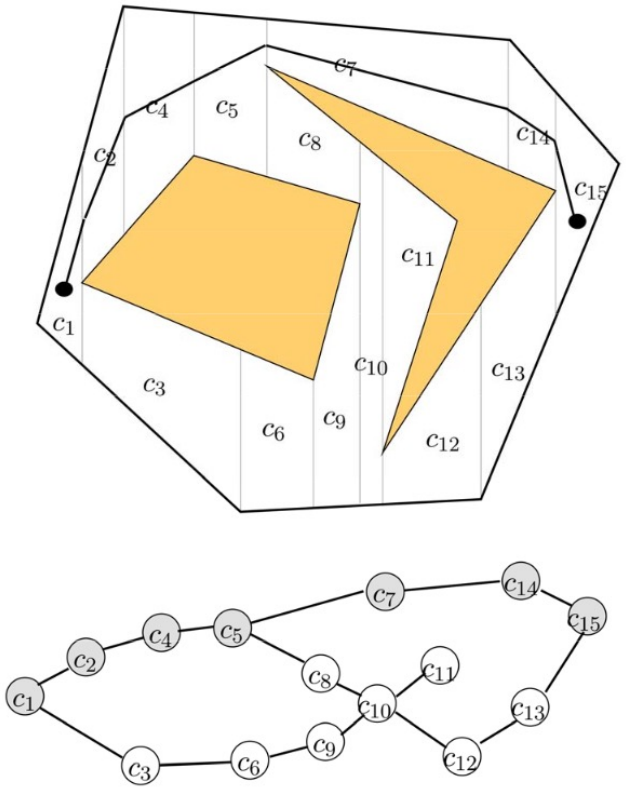


Some Well-Known Representations

- Visibility Graphs
- **Cell Decomposition**
- Probabilistic Road Map



Cell Decomposition



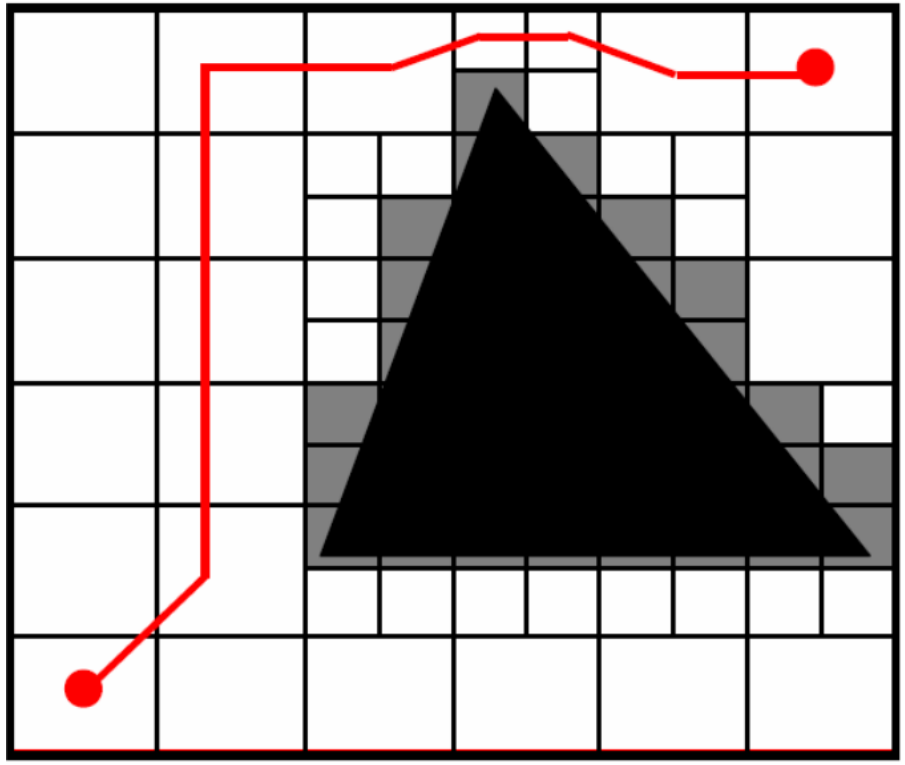
Exact cell decomposition

- Divides a space **F** precisely into sub-units

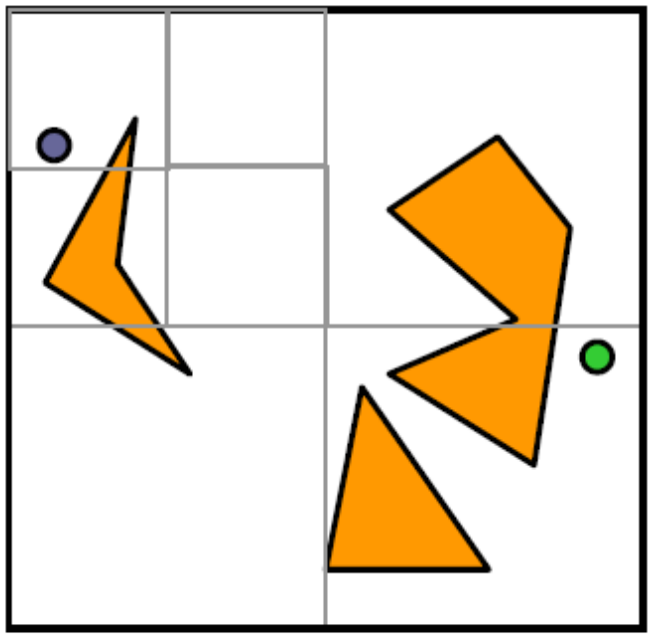
Approximate cell decomposition

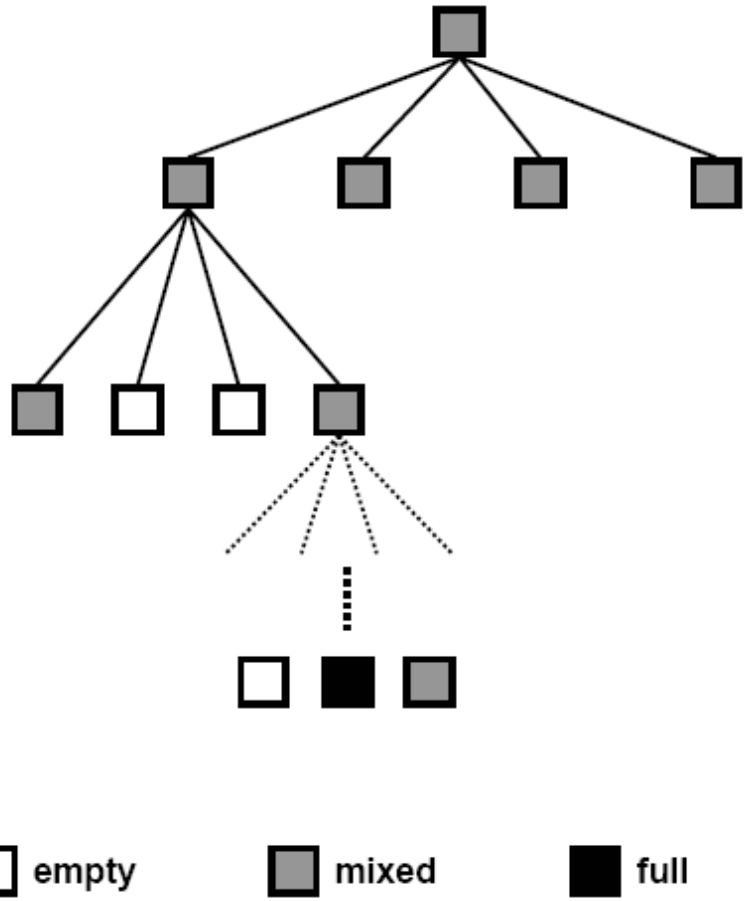
- ***F*** is represented by a collection of non-overlapping cells whose union is contained in ***F*.**
- Cells usually have simple, regular shapes, e.g., rectangles, squares.
- Facilitates hierarchical space decomposition

Grid-based Cell Decomposition

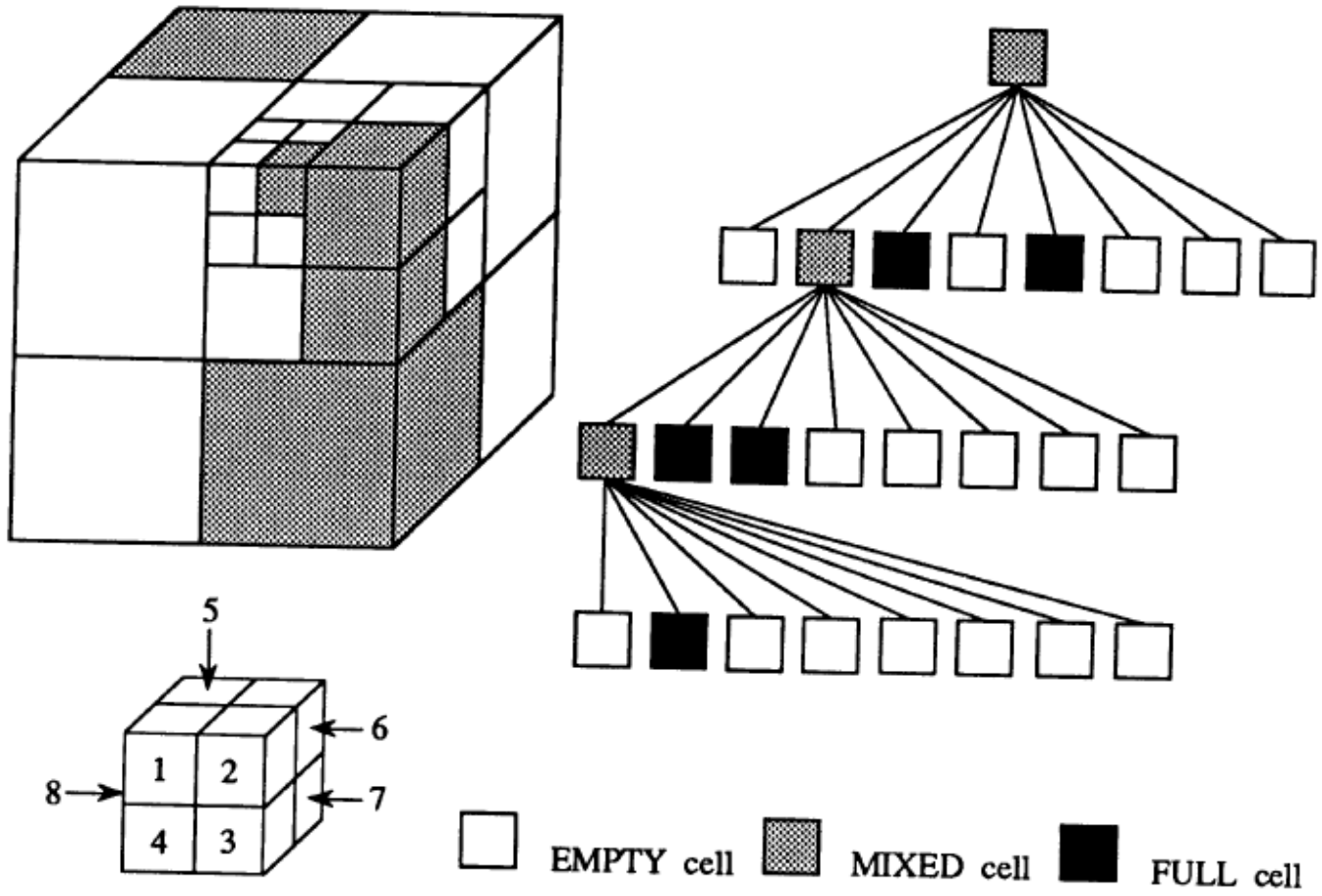


Quadtree Decomposition

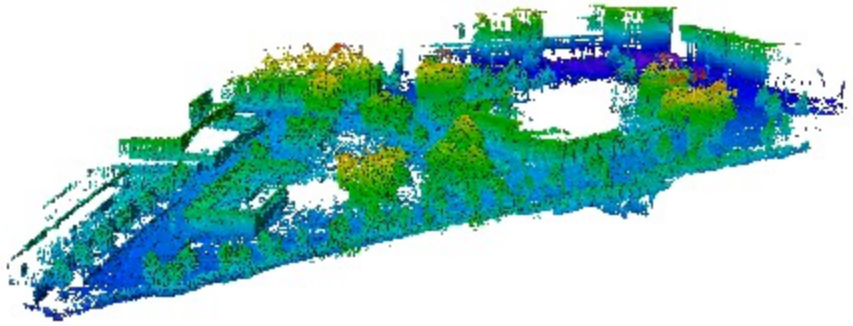




Octree Decomposition

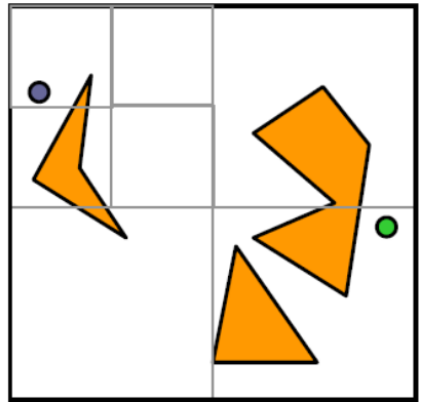


Octree

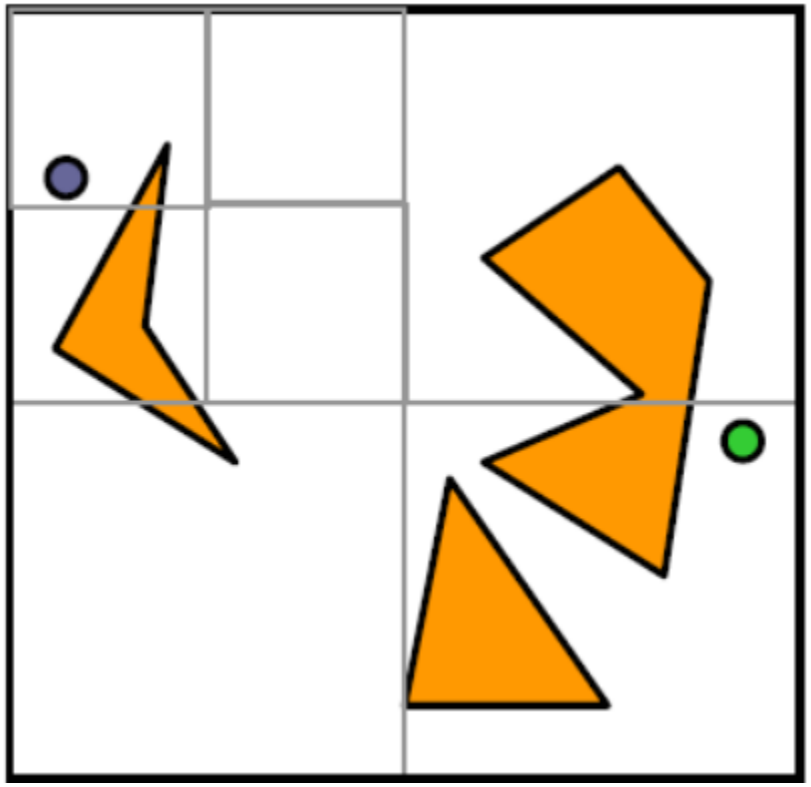


Cell Decomposition Path Planning Algorithm Outline

1. Decompose the free space F into cells.
2. Search for a sequence of **mixed** or **free** cells that connect the initial and goal positions.
3. Further decompose the mixed.
4. Repeat (2) and (3) until a sequence of **free** cells is found.

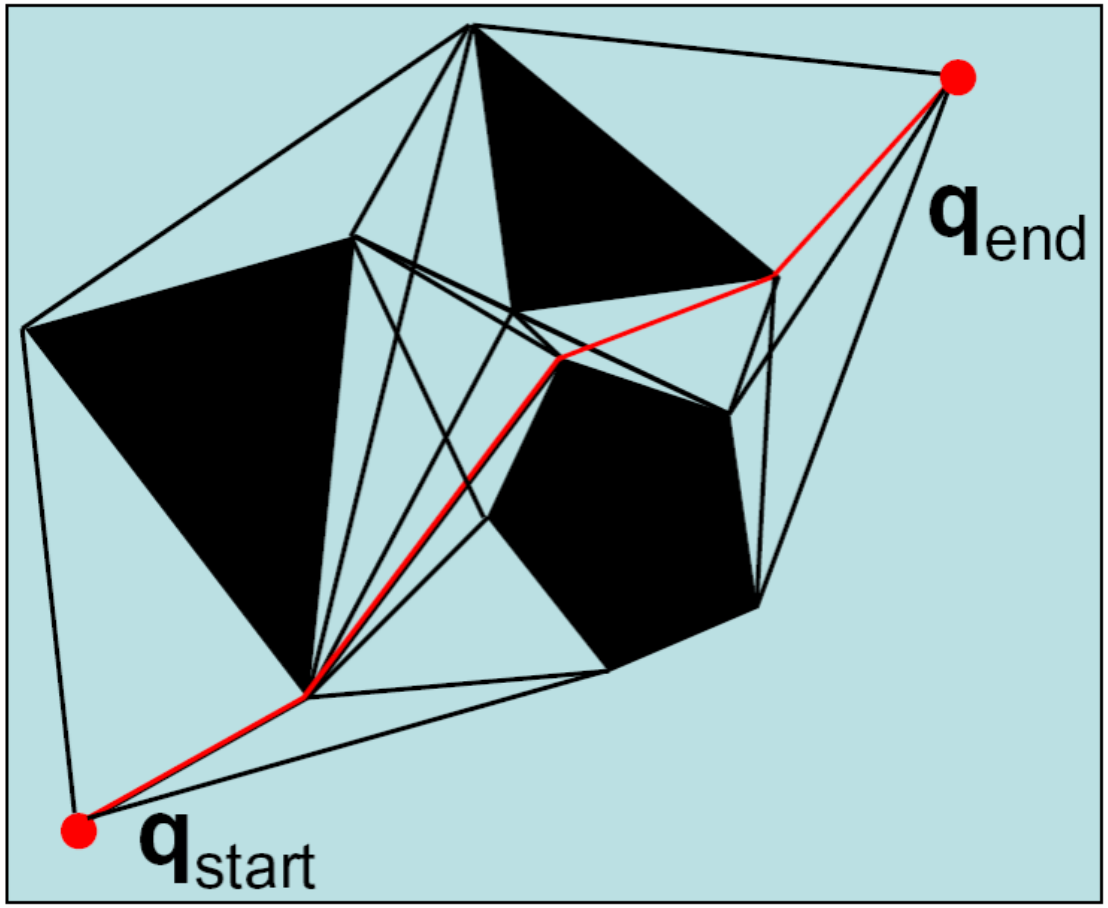


Cell Decomposition: Strengths/Weaknesses

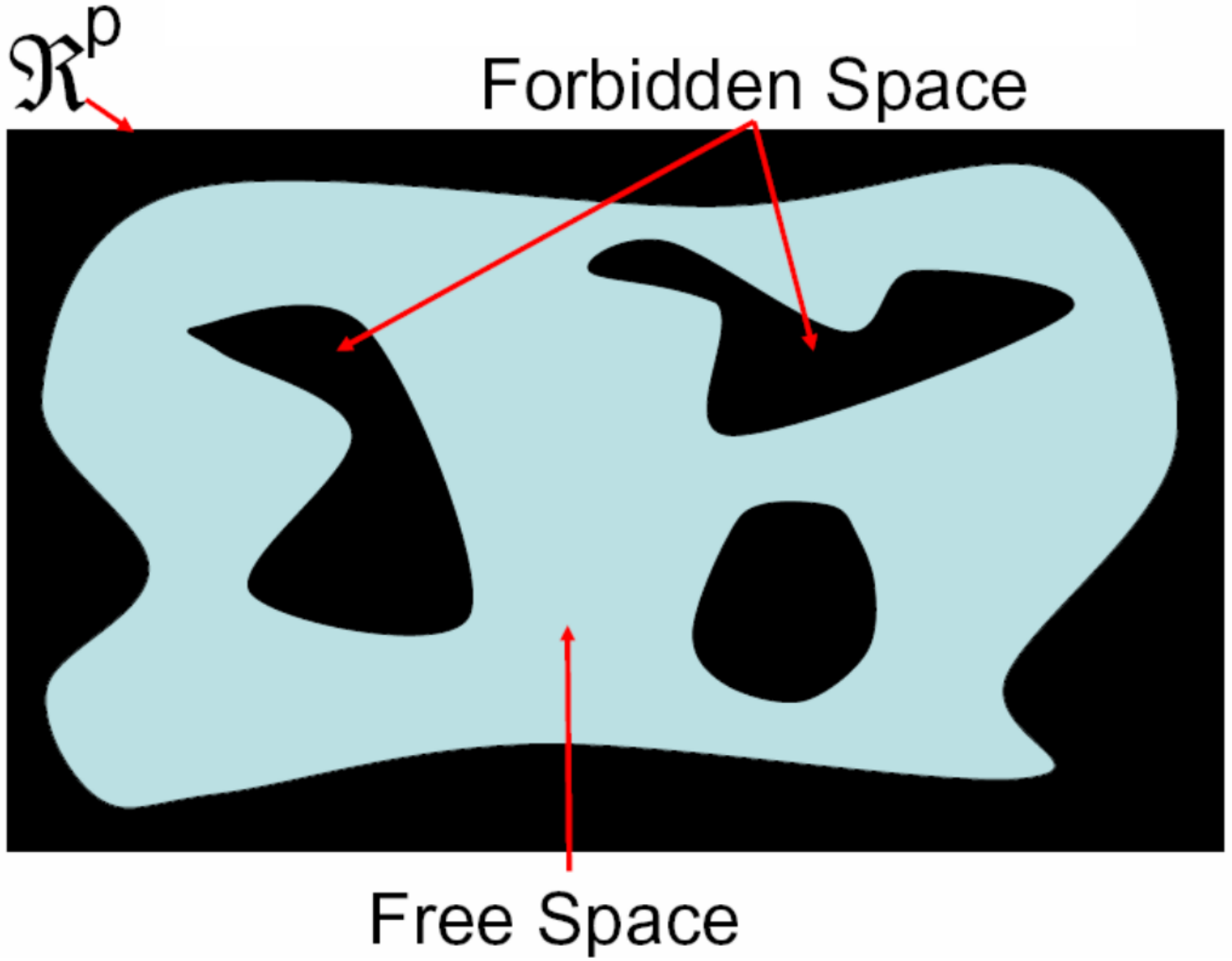


Some Well-Known Representations

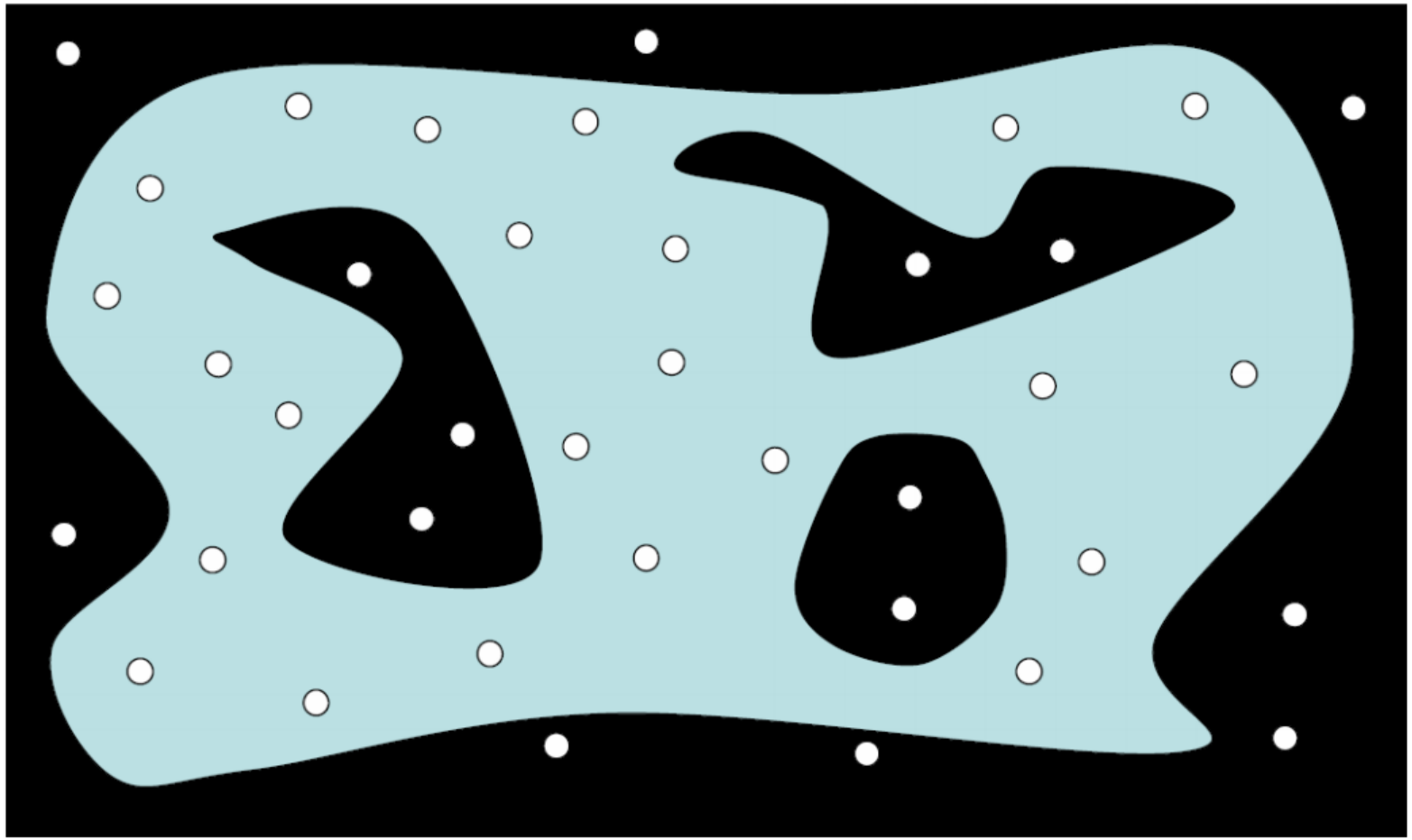
- Visibility Graphs
- Cell Decomposition
- **Probabilistic Road Map**



Probabilistic Road Map (PRM)

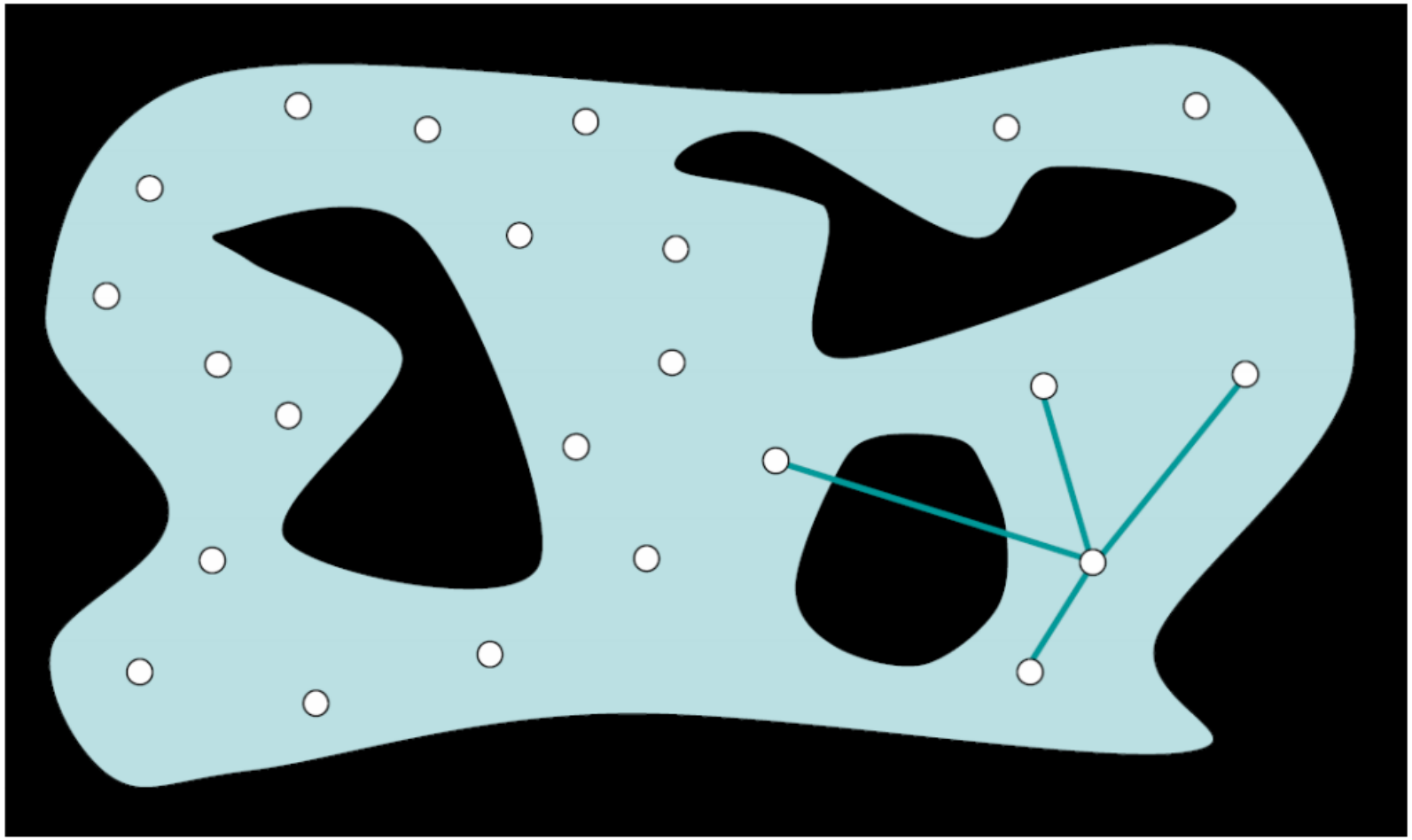


Probabilistic Road Map (PRM)



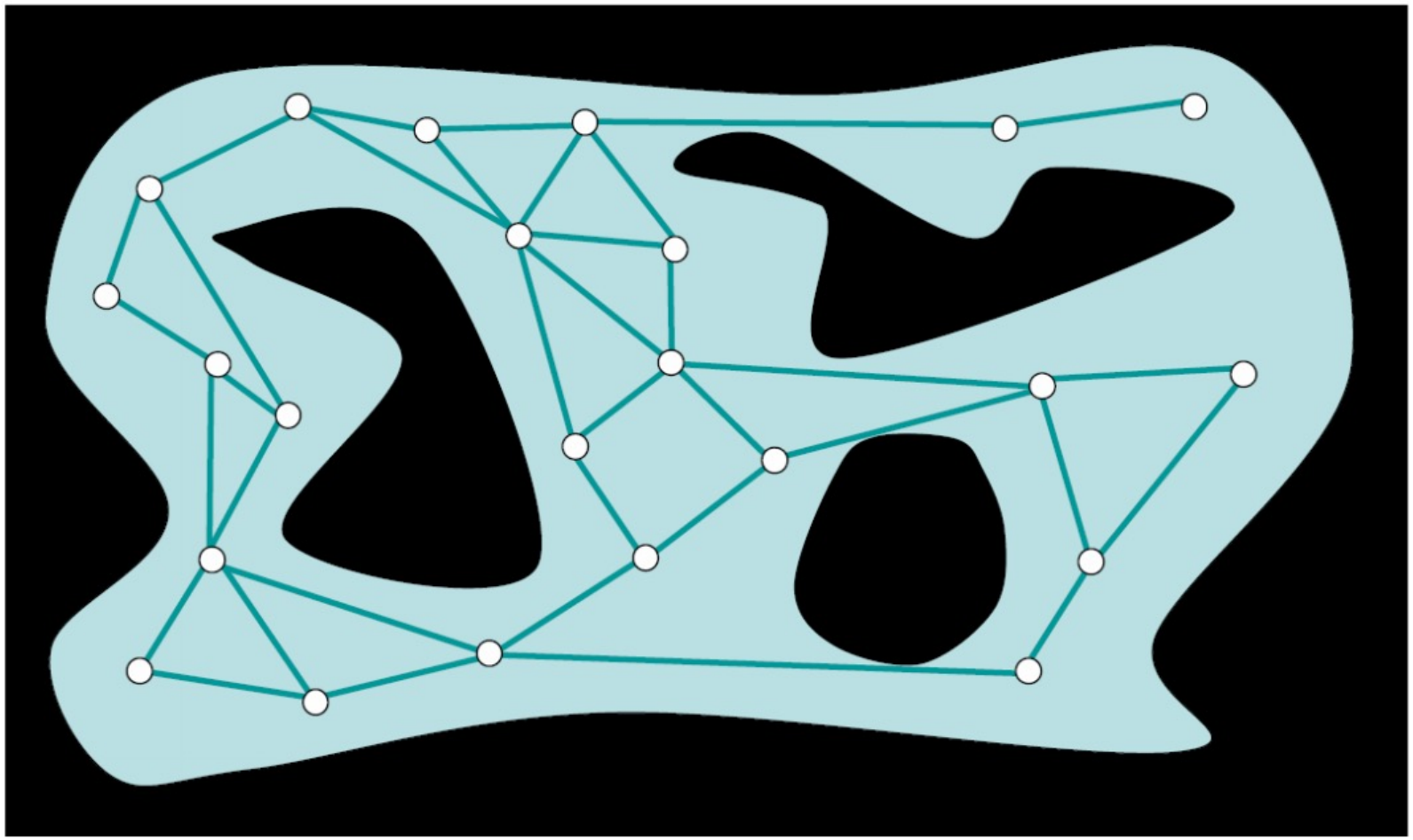
Sample random locations!

Probabilistic Road Map (PRM)



Remove points in forbidden areas
Link each point to its K nearest neighbors

Probabilistic Road Map (PRM)

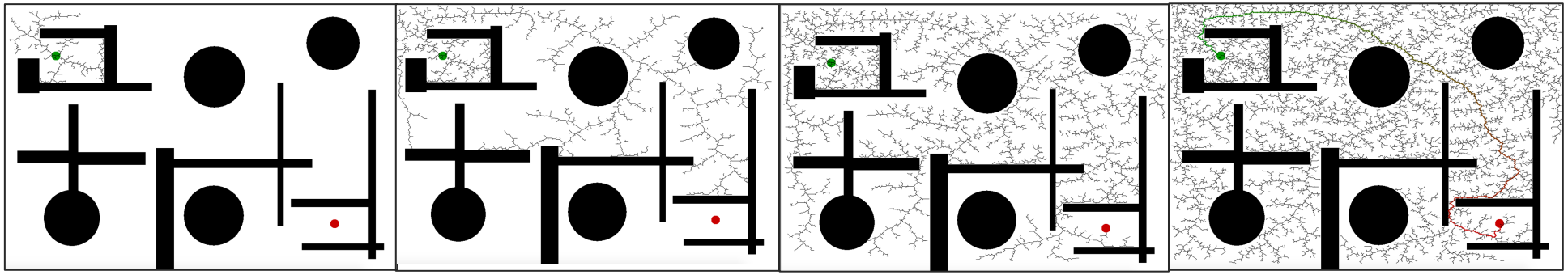


Remove edges crossing forbidden areas

How to sample points?

- Uniformly randomly
- Sample more near places with few neighbors
- Bias samples to exist near obstacles
- Use human-provided waypoints
- Something better?

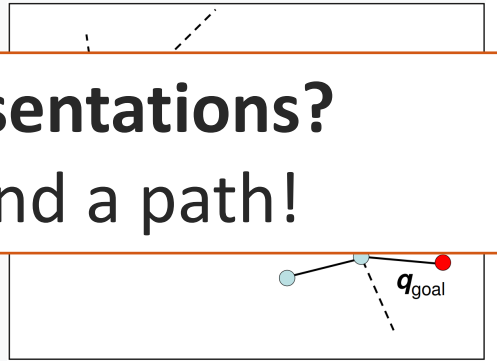
Rapidly-exploring Random Trees (RRT)



Rapidly-exploring Random Trees

```
Algorithm BuildRRT
  Input: Initial configuration q_init, number of vertices in RRT K, incremental distance Δq)
  Output: RRT graph G

  G.init(q_init)
  for k = 1 to K
    G.add_edge(q_near, q_new)
  return G
```



How can we make use of these representations?

Search algorithms provide a way to find a path!

Rapidly Exploring Random Trees [RRT]

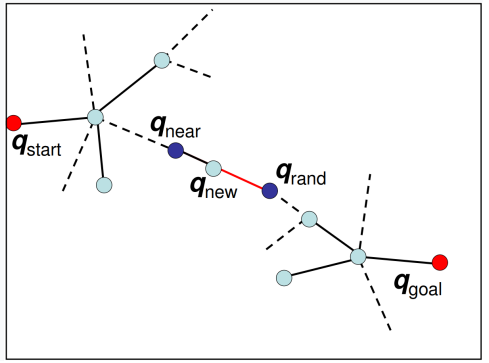
Basic idea:

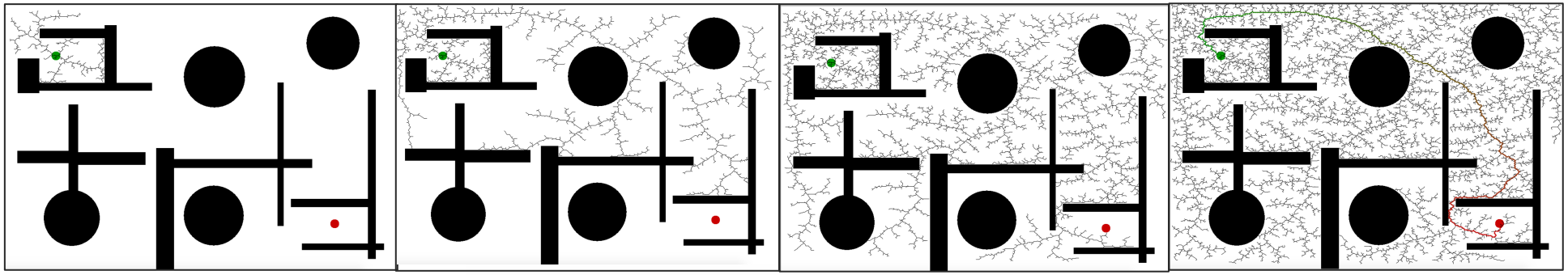
- Build up a tree through generating “next states” in the tree by executing random controls
- However: to ensure good coverage, we need something better than the above

Rapidly-exploring Random Trees

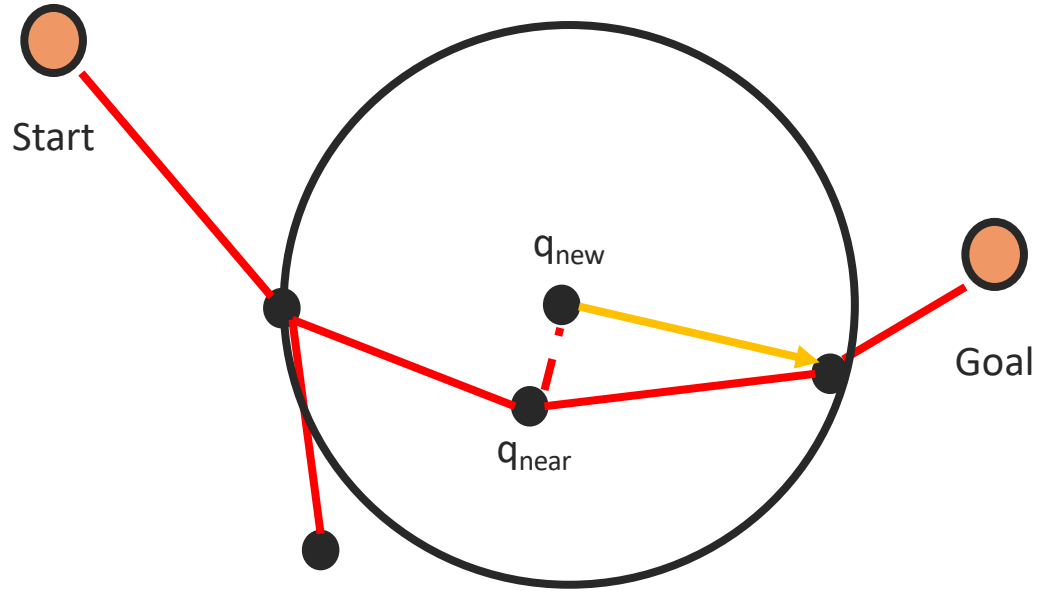
Algorithm BuildRRT
  Input: Initial configuration q_{init}, number of vertices in RRT K, incremental distance Δq)
  Output: RRT graph G

  G.init(q_{init})
  for k = 1 **to** K
    q_{rand} ← RAND_CONF()
    q_{near} ← NEAREST_VERTEX(q_{rand}, G)
    q_{new} ← NEW_CONF(q_{near}, q_{rand}, Δq)
    G.add_vertex(q_{new})
    G.add_edge(q_{near}, q_{new})
  return G



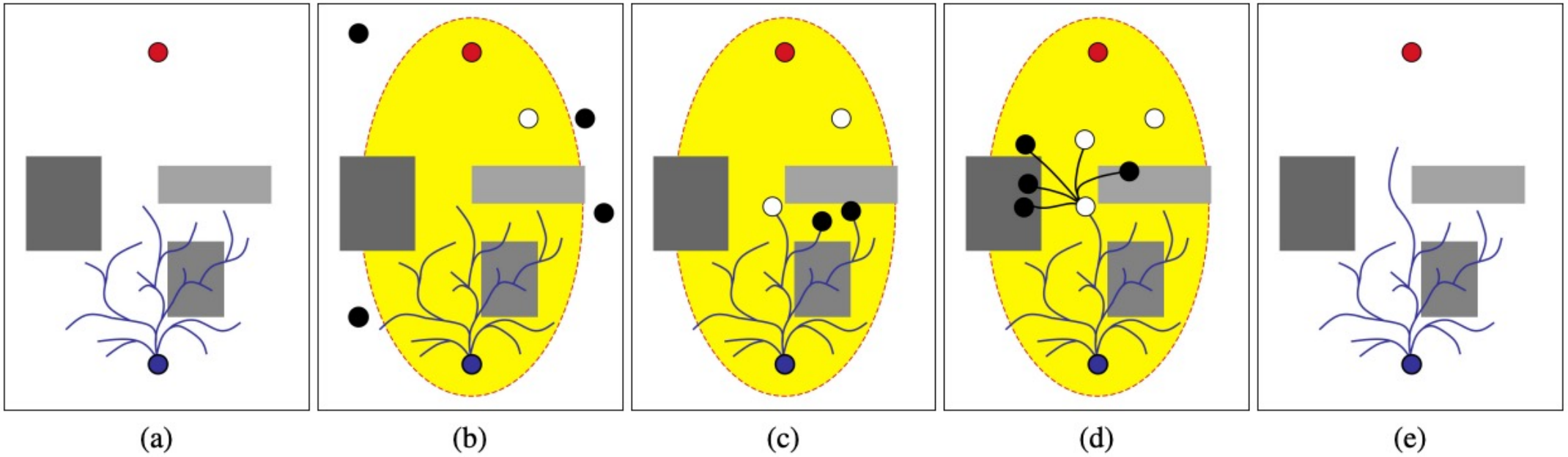


Making RRT optimal – RRT*



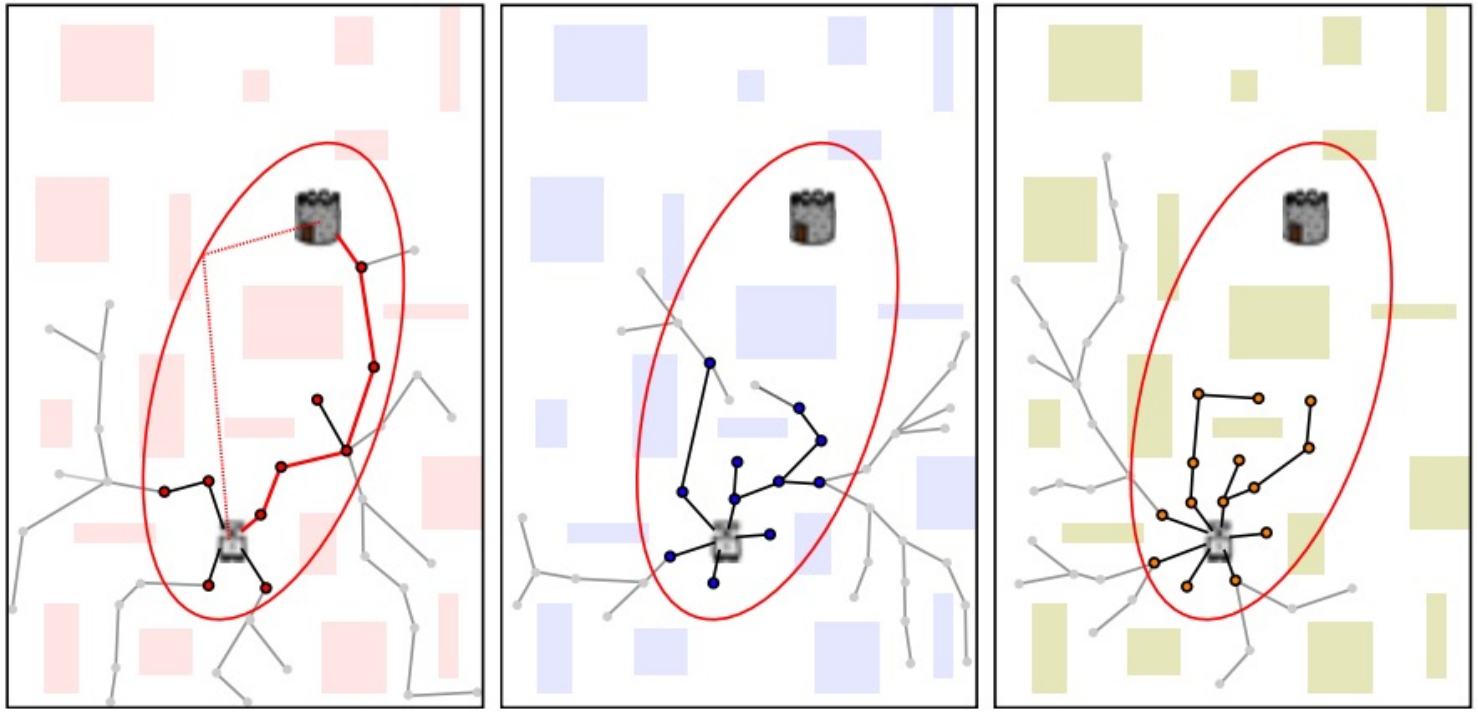
Karaman, Sertac, and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning." *The international journal of robotics research* 30, no. 7 (2011): 846-894.

Making RRT faster….



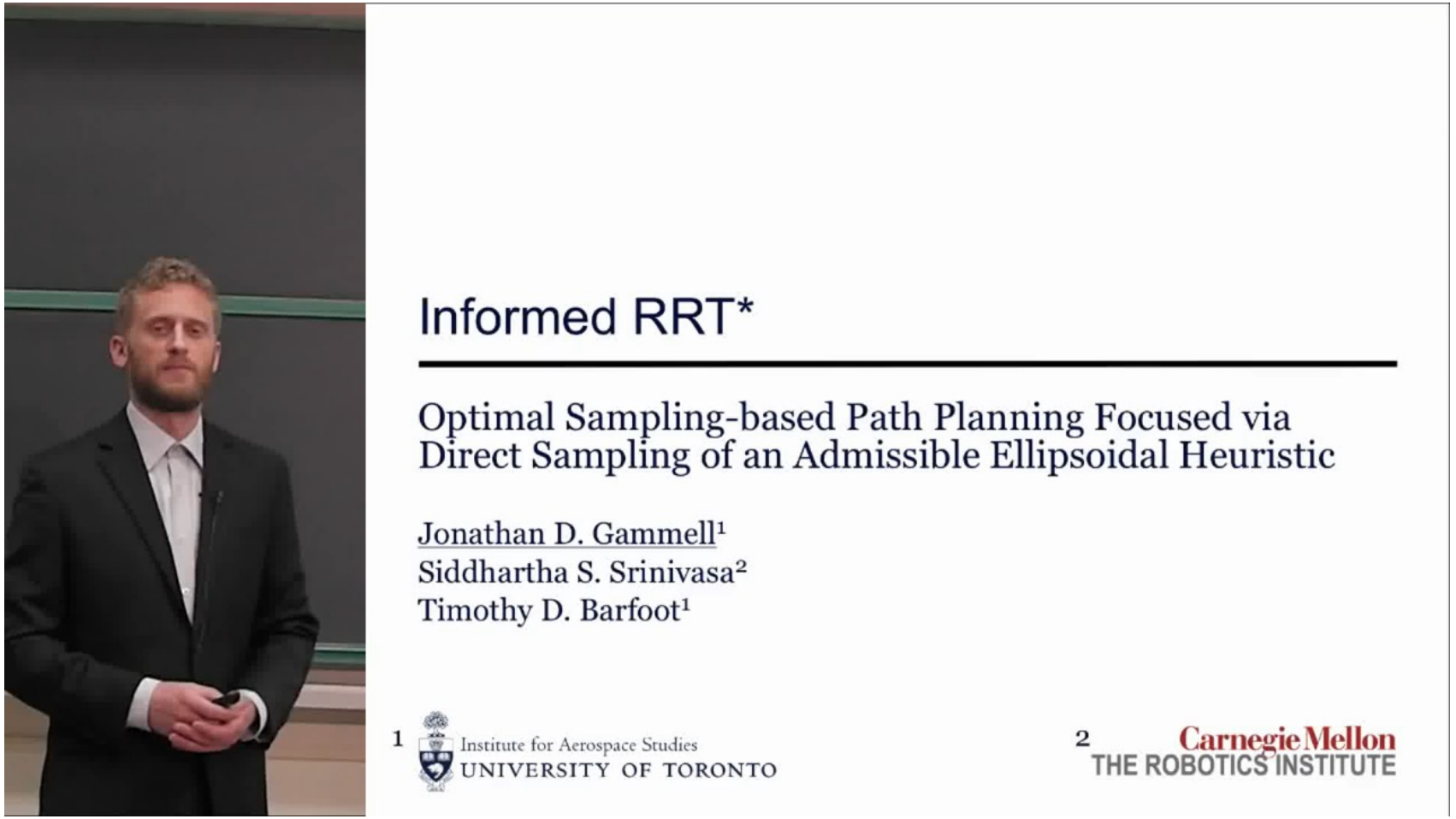
Ferguson, D. and Stentz, A., 2006, October. Anytime RRTs. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 5369-5375).

Parallelizing RRT

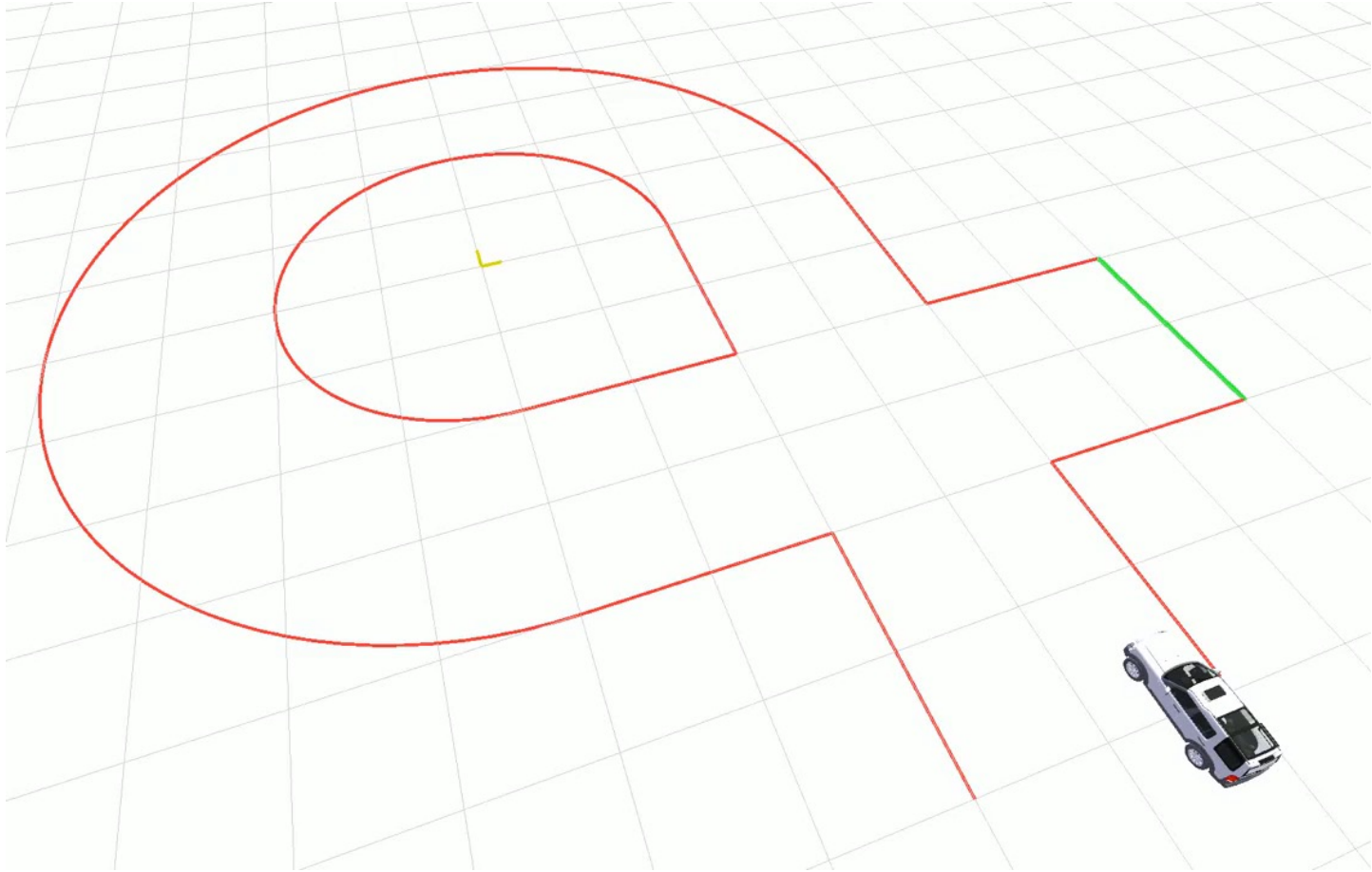


Otte, Michael, and Nikolaus Correll. "C-FOREST: Parallel shortest path planning with superlinear speedup." *IEEE Transactions on Robotics* 29, no. 3 (2013): 798-806.

Putting it together: Informed RRT*

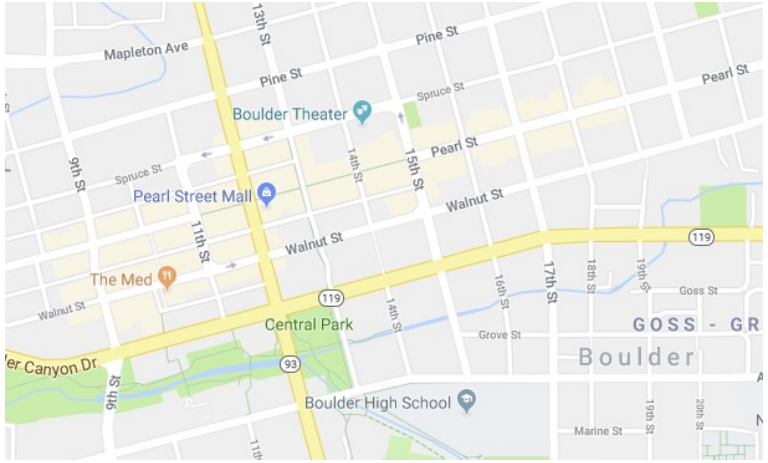


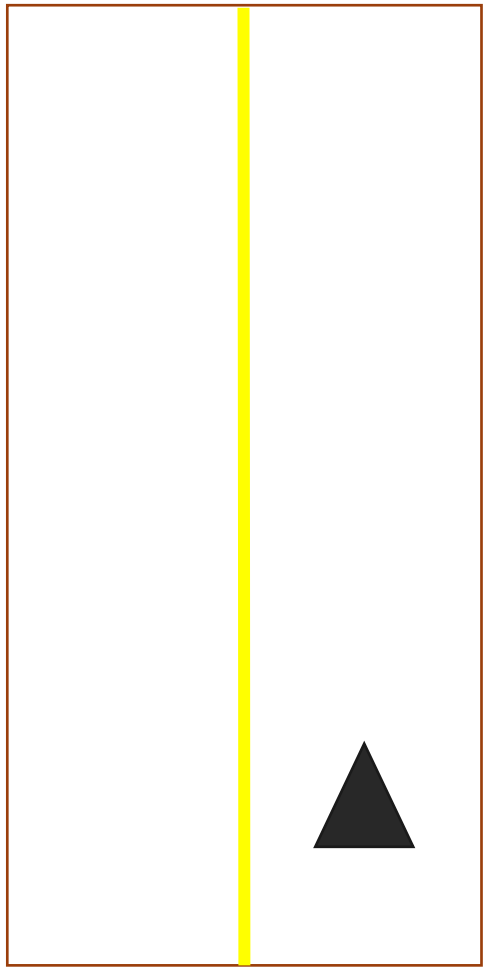
RRT with dynamics



Exercise: Building an Autonomous Car

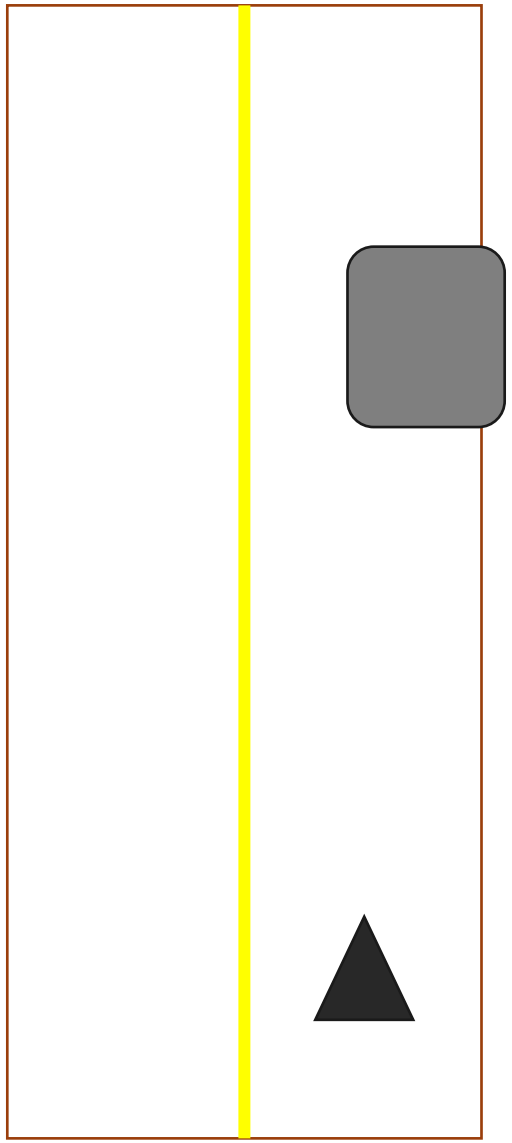
1. World Representation
2. Planning System



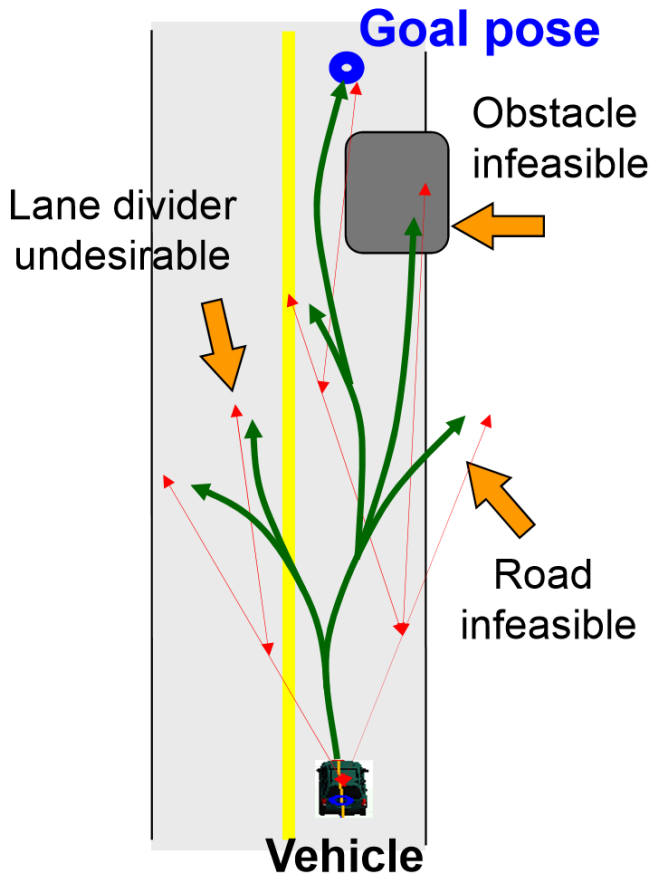


Exercise: Building an Autonomous Car

How can we get our agent to accommodate obstacles?



Exercise: Building an Autonomous Car



RRT for Autonomous Parking

How can we make use of these representations?

Search algorithms provide a way to find a path!