Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Fast Robots

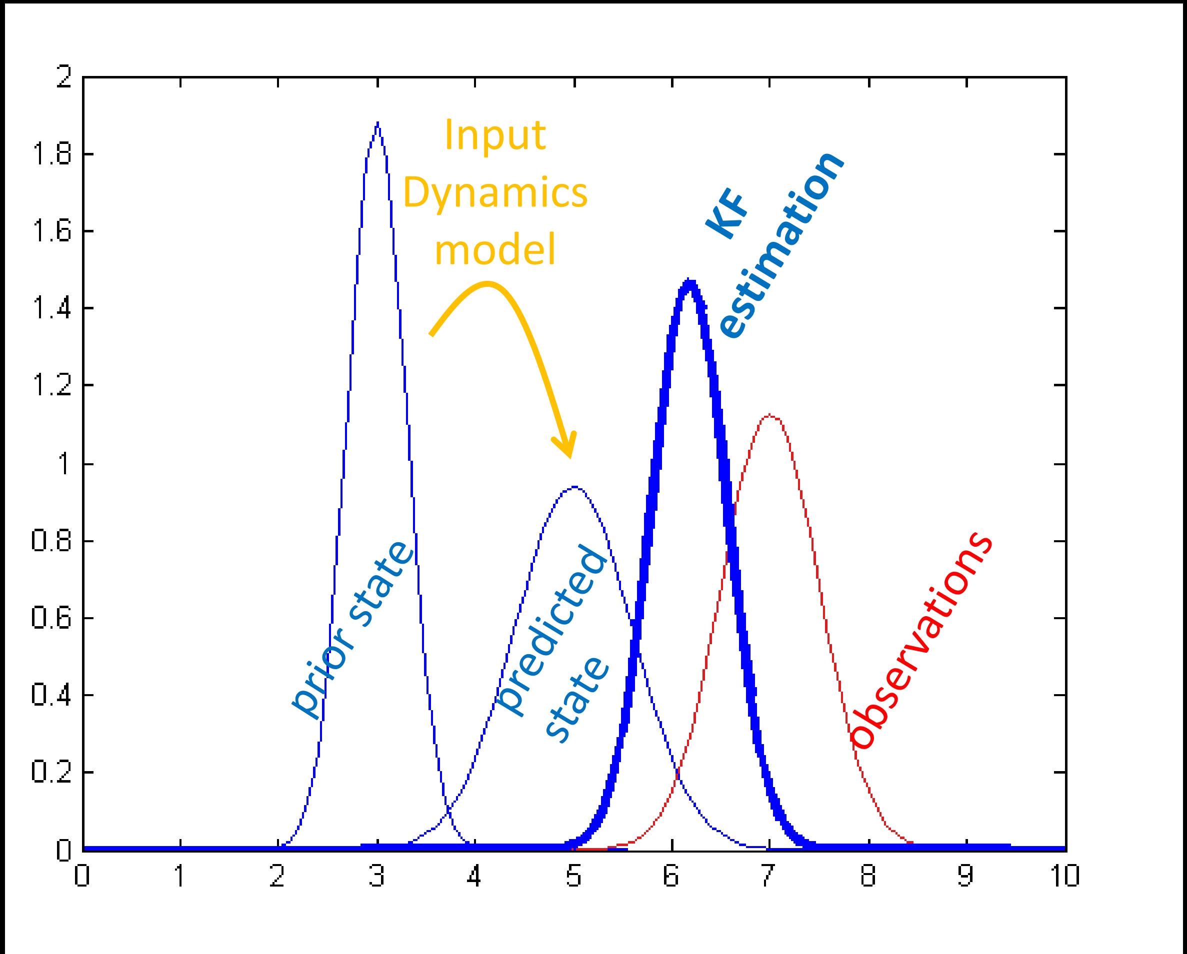
(Lecture 14 KF-Graph Construction)

Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Kalman Filter (one last example)

Kalman Filter

- Incorporate uncertainty to get better estimates based on inputs and observations



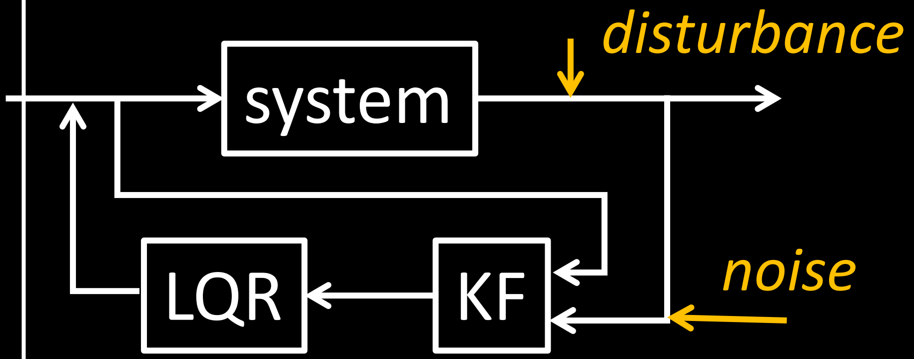
Kalman Filter Implementation

State estimate: $\mu(t)$
State uncertainty: $\Sigma(t)$
Process noise: Σ_u
Kalman filter gain: K_{KF}
Measurement noise: Σ_z

Kalman Filter ($\mu(t-1)$, $\Sigma(t-1)$, u(t), z(t))

1. $\mu_p(t) = A\,\mu(t-1) + B\,u(t)$
2. $\Sigma_p(t) = A\,\Sigma(t-1)\,A^T + \Sigma_u$
3. $K_{KF} = \Sigma_p(t)\,C^T\,(\,C\,\Sigma_p(t)\,C^T + \Sigma_z)^{-1}$
4. $\mu(t) = \mu_p(t) + K_{KF}\,(\,z(t) - C\,\mu_p(t)\,)$
5. $\Sigma(t) = (\,I - K_{KF}\,C)\,\Sigma_p(t)$
6. Return $\mu(t)$ and $\Sigma(t)$

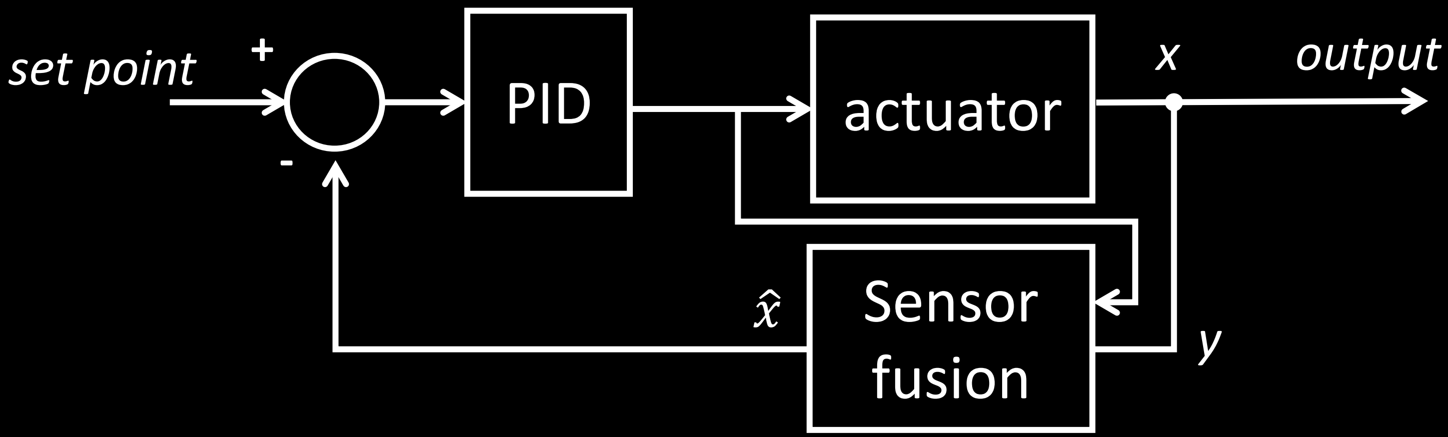
Steps 1–2: prediction
Steps 3–5: update



$$\Sigma_u = \begin{bmatrix} {\sigma_1}^2 & 0 & 0 \\ 0 & {\sigma_2}^2 & 0 \\ 0 & 0 & {\sigma_3}^2 \end{bmatrix}, \Sigma_z = \begin{bmatrix} {\sigma_4}^2 & 0 \\ 0 & {\sigma_5}^2 \end{bmatrix}$$

Lab 6-8: PID control – Sensor Fusion - Stunt

- Task A: Don’t Hit the Wall!
- Task B: Drift much?
- Task C: Thread the Needle!
  - Benefit: Best use of a Kalman Filter and LQG



Lab 6-8: PID control – Sensor Fusion - Stunt

- Task A: Don’t Hit the Wall!
- Task B: Drift much?
- Task C: Thread the Needle!

Procedure

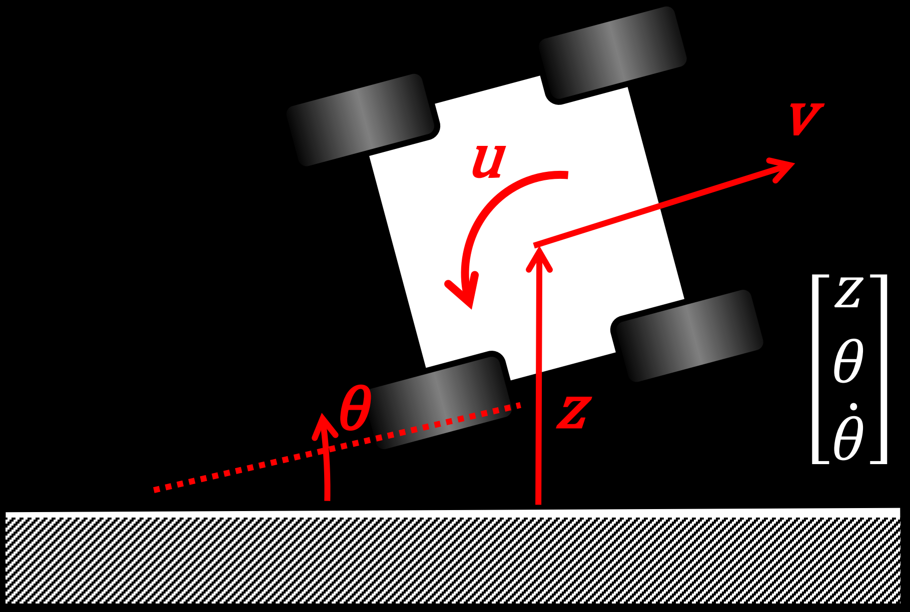
- Lab 6: Get basic PID to work
- Lab 7: Sensor Fusion
  - Approximate the state space equations
    - Step response
  - Implement Kalman Filter
    - Determine process and measurement noise
    - Try it offline on solution from lab 6
    - Try it online on your robot
- Lab 8: Use KF and PID control to execute fast stunts

Lab 7, Task C: State Space Equations

- Equations of Motion
  - $\dot{z} = v\sin(\theta)$
    - Small-angle appr.: $\dot{z} = v\theta$
  - Input, *u*, is a torque
    - $u - d\dot{\theta} = I\ddot{\theta}$
    - $\frac{u}{I} - \frac{d}{I}\dot{\theta} = \ddot{\theta}$ **v, d, I?**

$$\begin{bmatrix}\dot{z}\\ \dot{\theta}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix}0 & v & 0\\ 0 & 0 & 1\\ 0 & 0 & -\frac{d}{I}\end{bmatrix}\begin{bmatrix}z\\ \theta\\ \dot{\theta}\end{bmatrix} + \begin{bmatrix}0\\ 0\\ \frac{1}{I}\end{bmatrix}u$$

- Find *d* at steady state
  - $\ddot{\theta}_{ss} = 0$
  - $\frac{u_{step}}{I} - \frac{d}{I}\dot{\theta}_{ss} = 0$
  - $d = \frac{u_{step}}{\dot{\theta}_{ss}}$



$$\begin{bmatrix}z\\ \theta\\ \dot{\theta}\end{bmatrix}$$

Lab 7, Task C: State Space Equations

- Equations of Motion
  - $\dot{z} = v\sin(\theta)$
    - Small-angle appr.: $\dot{z} = v\theta$
  - Input, *u,* is a torque
    - $u - d\dot{\theta} = I\ddot{\theta}$
    - $\frac{u}{I} - \frac{d}{I}\dot{\theta} = \ddot{\theta}$ **v, d, I?**

$$\begin{bmatrix}\dot{z}\\ \dot{\theta}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix}0 & v & 0\\ 0 & 0 & 1\\ 0 & 0 & -\frac{d}{I}\end{bmatrix}\begin{bmatrix}z\\ \theta\\ \dot{\theta}\end{bmatrix} + \begin{bmatrix}0\\ 0\\ \frac{1}{I}\end{bmatrix}u$$

Use the 90% rise time to determine I

- Pretend $\dot{\theta}$=x:
  - $\dot{x} = -\frac{d}{I}x + \frac{u_{step}}{I}$
  - $\dot{x} + \frac{d}{I}x = \frac{u_{step}}{I}$
- $x = 1 - e^{-\frac{d}{I}t_{0.9}} \leftrightarrow 1 - x = e^{-\frac{d}{I}t_{0.9}}$
- $\ln(1-x) = -\frac{d}{I}t_{0.9}$
- $I = \frac{-dt_{0.9}}{\ln(0.1)}$

(unit step response)

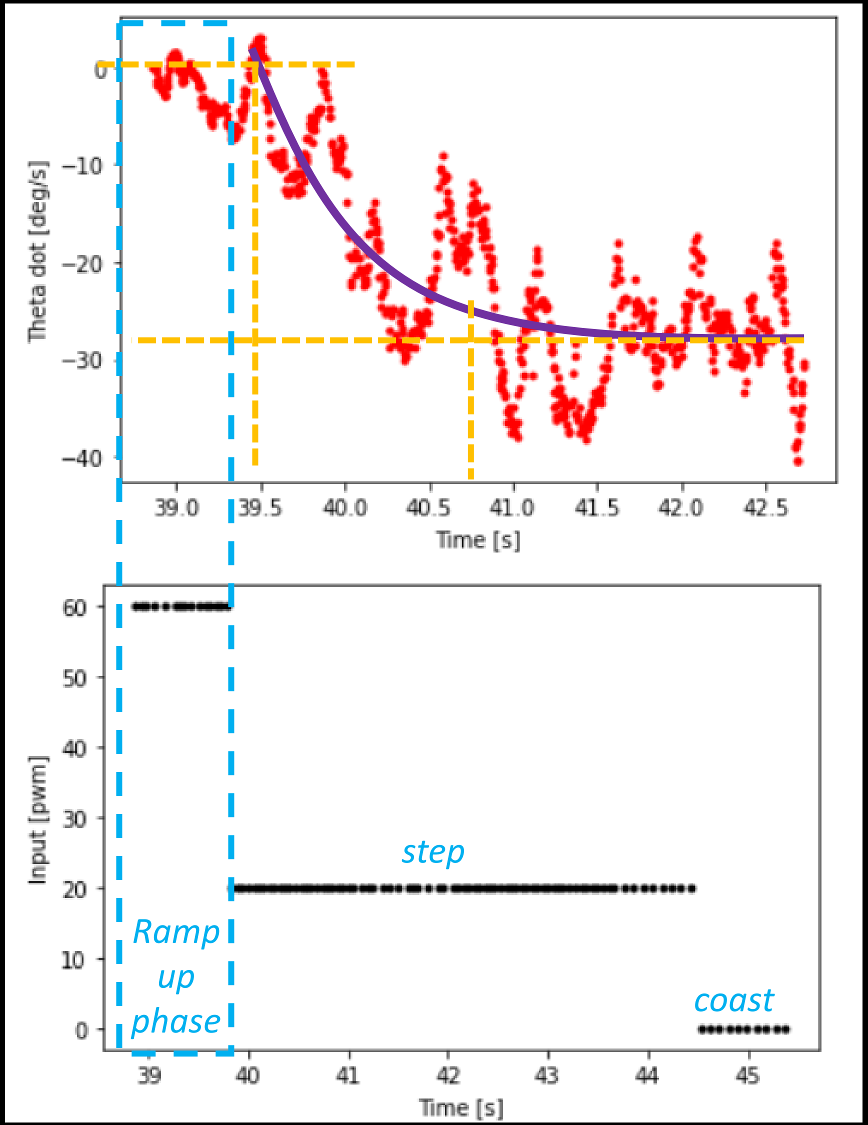
1st order system:

$$\frac{dx(t)}{dt} + \frac{1}{\tau}x(t) = y(t)$$

Step response solution:

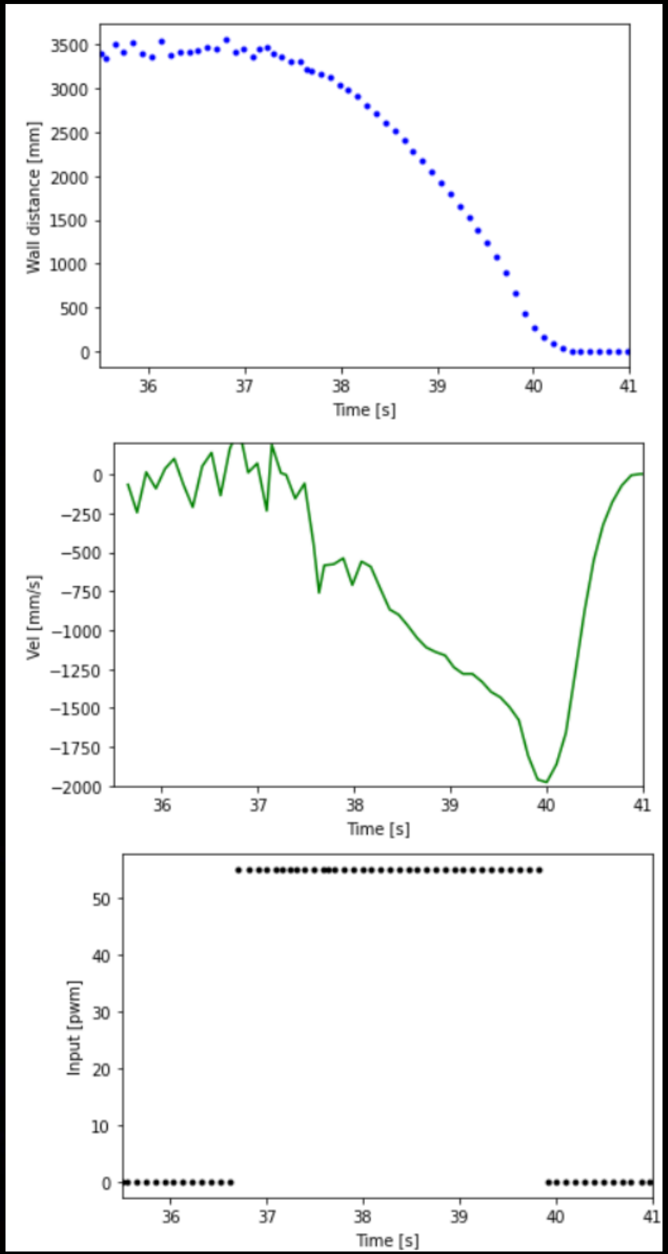
$$x(t) = 1 - e^{-\frac{t}{\tau}}$$

Lab 7, Task C: Kalman Filter



- $d = \frac{u_{Step}}{\dot{\theta}_{SS}} = \frac{-1}{-28\pi/180} = 2.047$
- $I = \frac{-d t_{0.9}}{\ln(0.1)} = \frac{-2.047 \cdot 1.3}{-2.3026} = 1.156$
- $\Sigma_u = \begin{bmatrix} {\sigma_1}^2 & 0 & 0 \\ 0 & {\sigma_2}^2 & 0 \\ 0 & 0 & {\sigma_3}^2 \end{bmatrix}$
  - $\sigma_1 = \sqrt{5^2 \cdot \frac{1}{0.05}} = 22mm$
  - $\sigma_2 = 0.1rad = 5.7deg,\ \sigma_3 = 0.1\frac{rad}{s} = 5.7\frac{deg}{s}$
- $\Sigma_z = \begin{bmatrix} {\sigma_4}^2 & 0 \\ 0 & {\sigma_5}^2 \end{bmatrix}$
  - $\sigma_4 = 5mm, \sigma_5 = 0.4\frac{rad}{s}$
- Initial covariance: $\Sigma = \begin{bmatrix} 5^2 & 0 & 0 \\ 0 & 0.1^2 & 0 \\ 0 & 0 & 0.05^2 \end{bmatrix}$

Lab 7, Task C: Kalman Filter



- What about v?
  - Drive towards a wall at base speed and use ToF data
    - Max speed
      - Appr. 1750mm/s
  - Check it visually in our video
    - Max speed
      - Appr. 6000mm/8s = 750mm/s
  - Why??

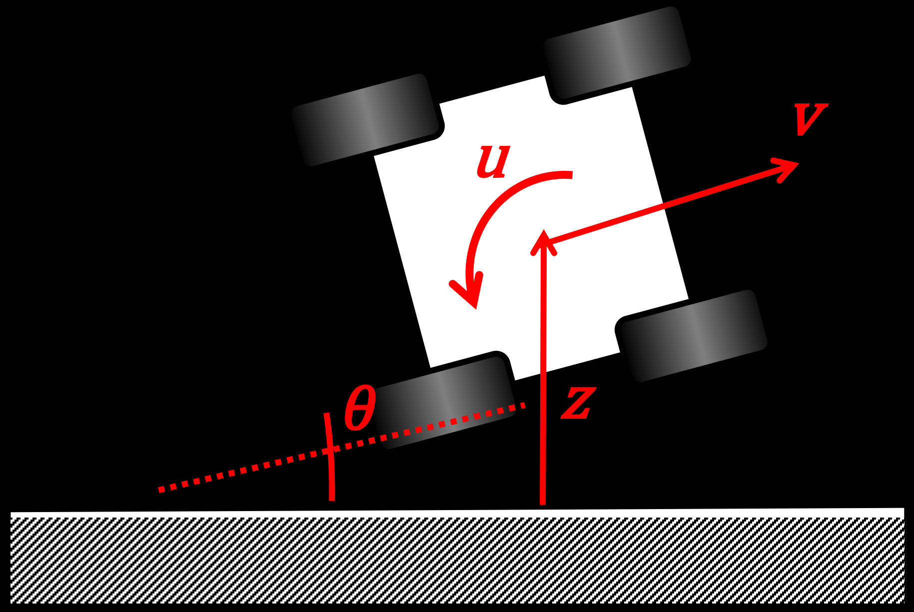
Lab 7, Task C: State Space Equations

(measured by driving at base speed towards a wall)

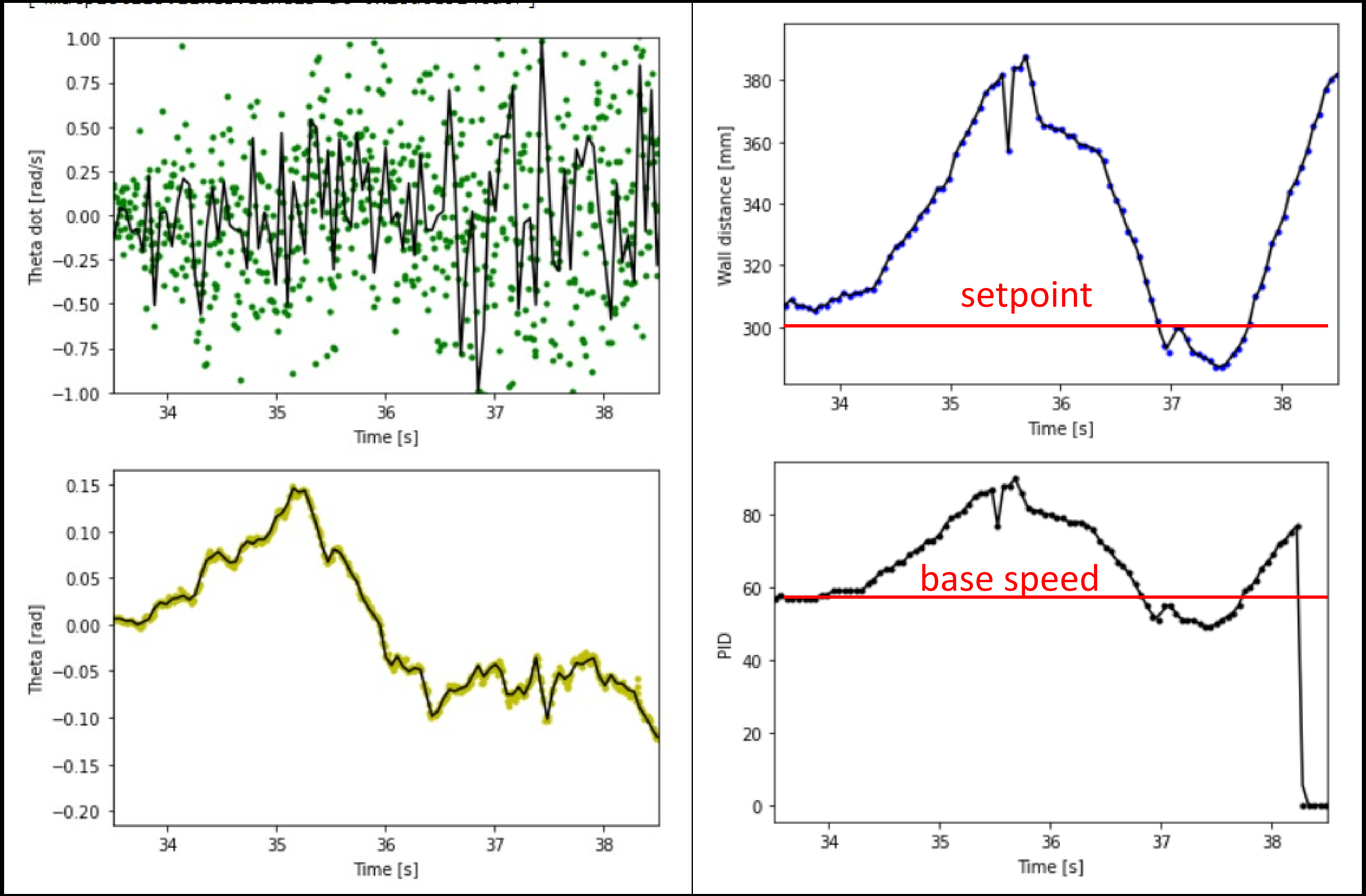
- Equations of Motion
  - $\dot{z} = v sin(\theta)$
    - Small-angle appr.: $\dot{z} = v\theta$
  - Input, *u*, is a torque
    - $u - d\dot{\theta} = I\ddot{\theta}$
    - $\frac{u}{I} - \frac{d}{I}\dot{\theta} = \ddot{\theta}$

$$\begin{bmatrix} \dot{z} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & v & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -\frac{d}{I} \end{bmatrix} \begin{bmatrix} z \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{I} \end{bmatrix} u$$

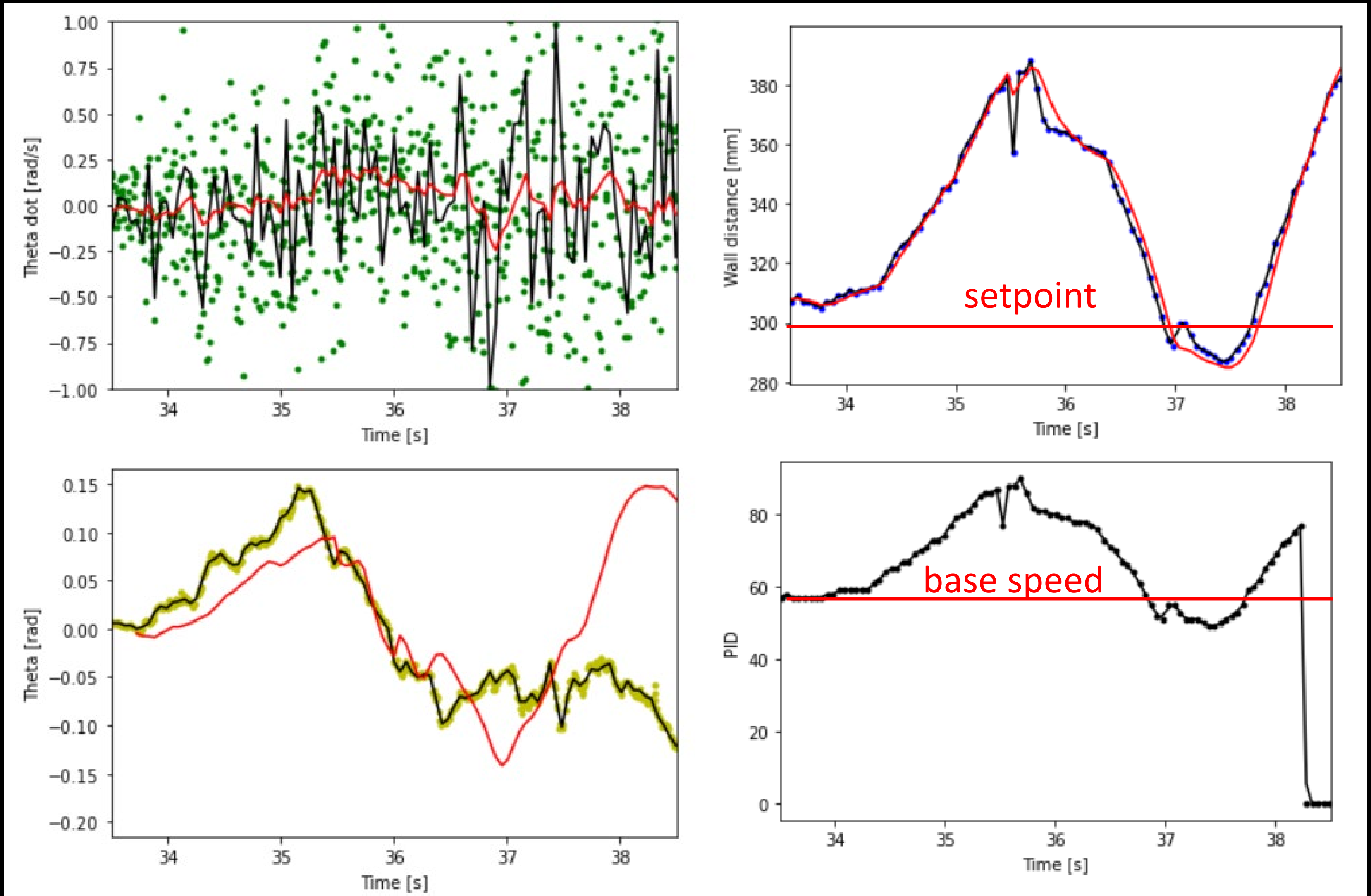
- We know A and B, we measured (d, I, v)
- $C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- We estimated:
  - $\Sigma_u, \Sigma_z, \Sigma$
- Convert from A, B to A_d, B_d
- Convert from unit input to real input



Lab 7, Task C: PID control and Kalman Filter

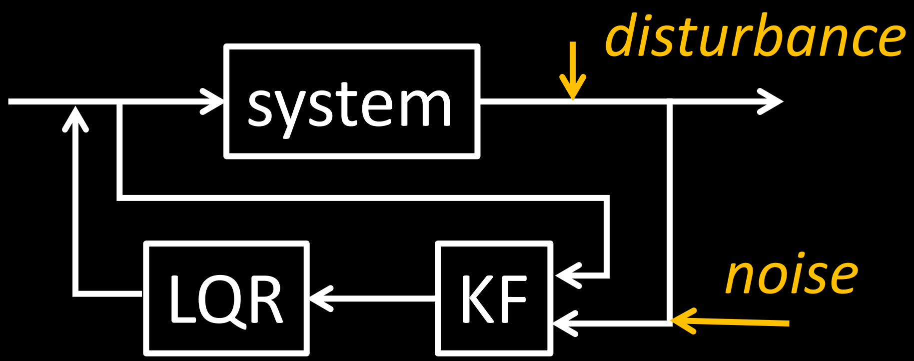


Lab 7, Task C: PID control and Kalman Filter



*We could run this both when TOF- and when gyroscope measurements come in.

Kalman Filter Implementation



Why KF?

- Not full state feedback
- Bad sensors
- Slow feedback

Kalman Filter (μ(t-1), Σ(t-1), u(t), z(t))

1. $\mu_p(t) = A\,\mu(t-1) + B\,u(t)$
2. $\Sigma_p(t) = A\,\Sigma(t-1)\,A^T + \Sigma_u$
3. $K_{KF} = \Sigma_p(t)\,C^T\,(\,C\,\Sigma_p(t)\,C^T + \Sigma_z)^{-1}$
4. $\mu(t) = \mu_p(t) + K_{KF}\,(\,z(t) - C\,\mu_p(t)\,)$
5. $\Sigma(t) = (\,\mathbf{I} - K_{KF}\,C)\,\Sigma_p(t)$
6. Return μ(t) and Σ(t)

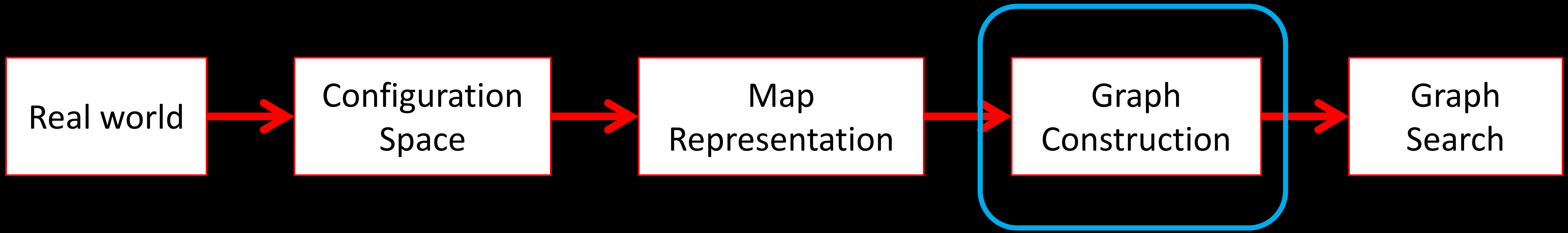
Steps 1–2: prediction

Steps 3–5: update

Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

Constructing Graphs

Global Motion Planning with Maps



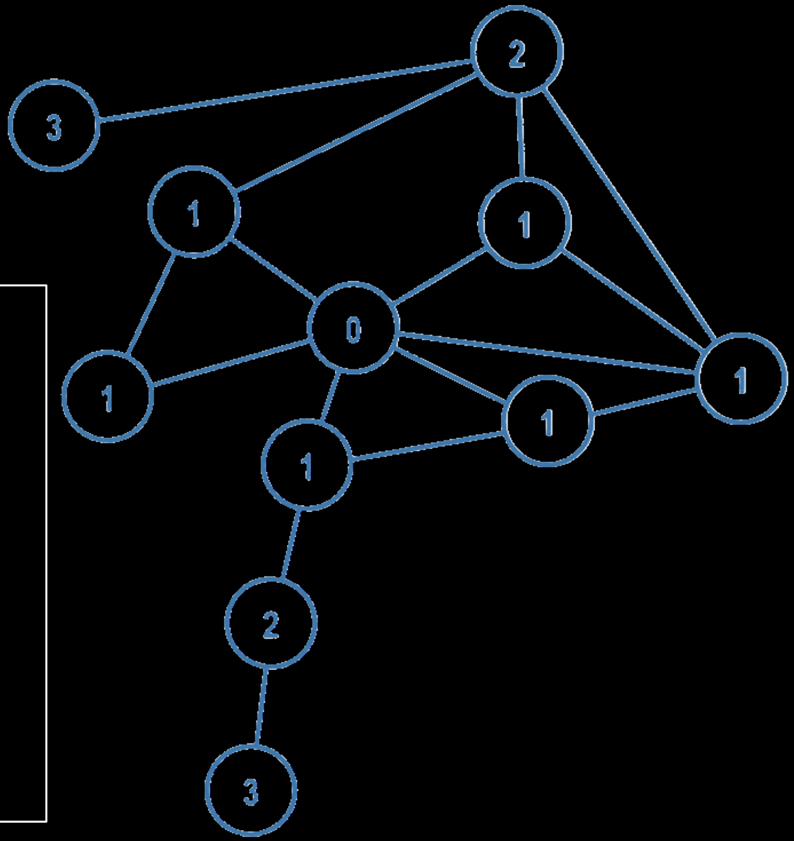
- Topological Graphs
- Cell decomposition
- Visibility Graphs
- RRT
- PRM

Common alternatives

- Optimal control
- Potential fields

Graph Construction

- Transform continuous/discrete/topological maps to a discrete graph
- Why?
  - Model the path planning problem as a search problem
  - Graph theory has lots of tools
  - Real-time capable algorithms
  - Can accommodate for evolving maps

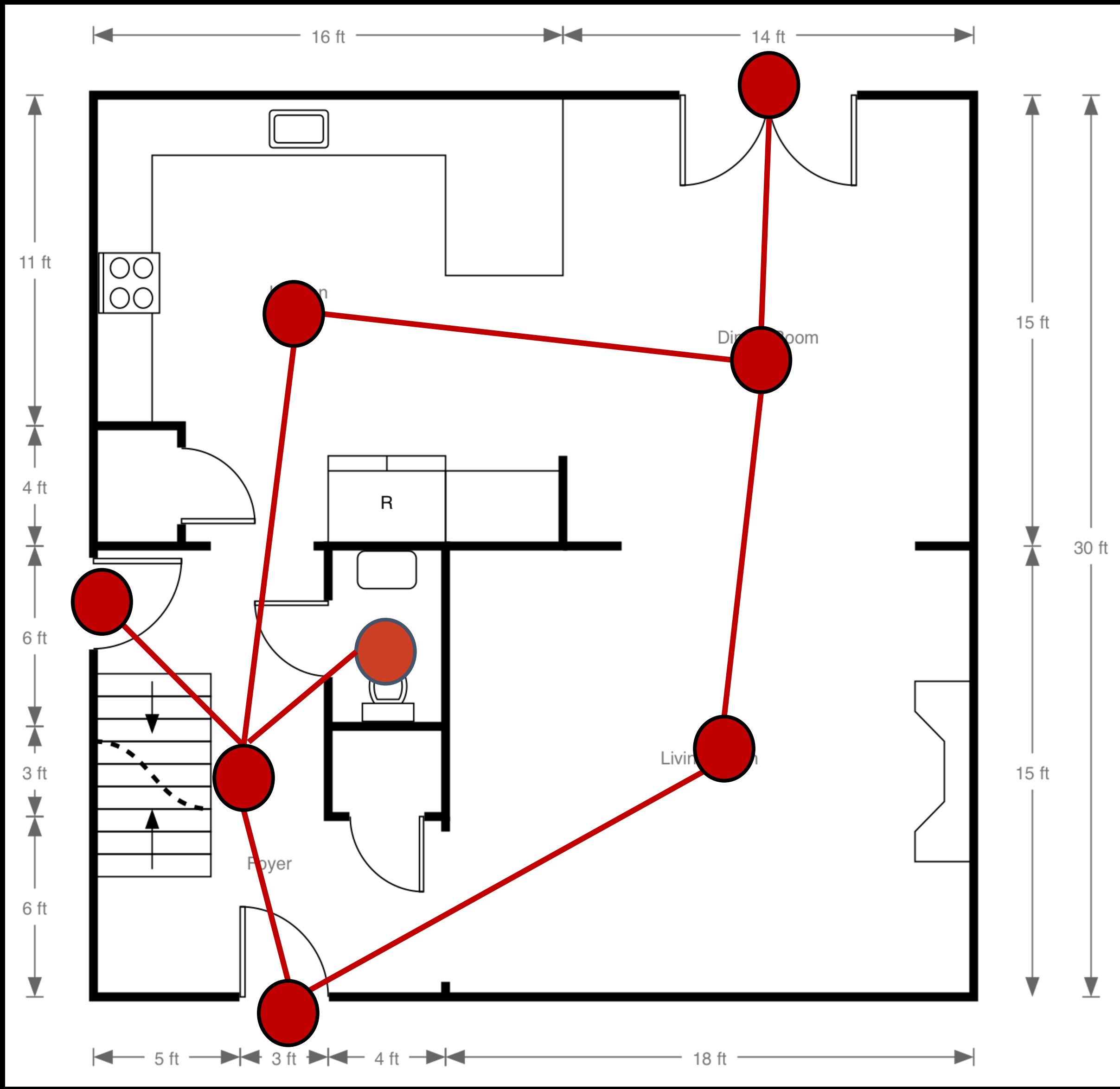


1. Divide space into simple, connected regions, or “cells”
2. Determine adjacency of open cells
3. Construct a connectivity graph
4. Find cells with initial and goal configuration
5. Search for a path in the connectivity graph to join them
6. From the sequence of cells, compute a path within each cell
   - e.g. passing through the midpoints of cell boundaries or by sequence of wall following movements

Geometry-Based Planners

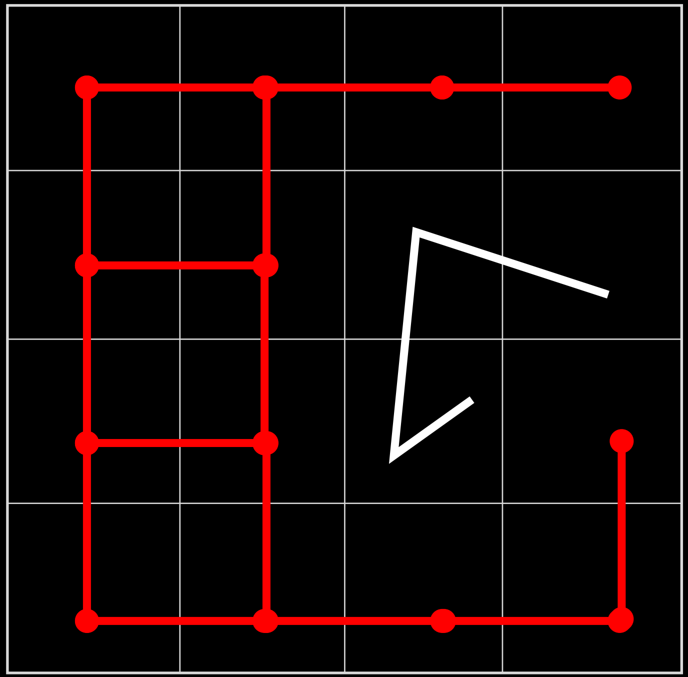
Topological Maps

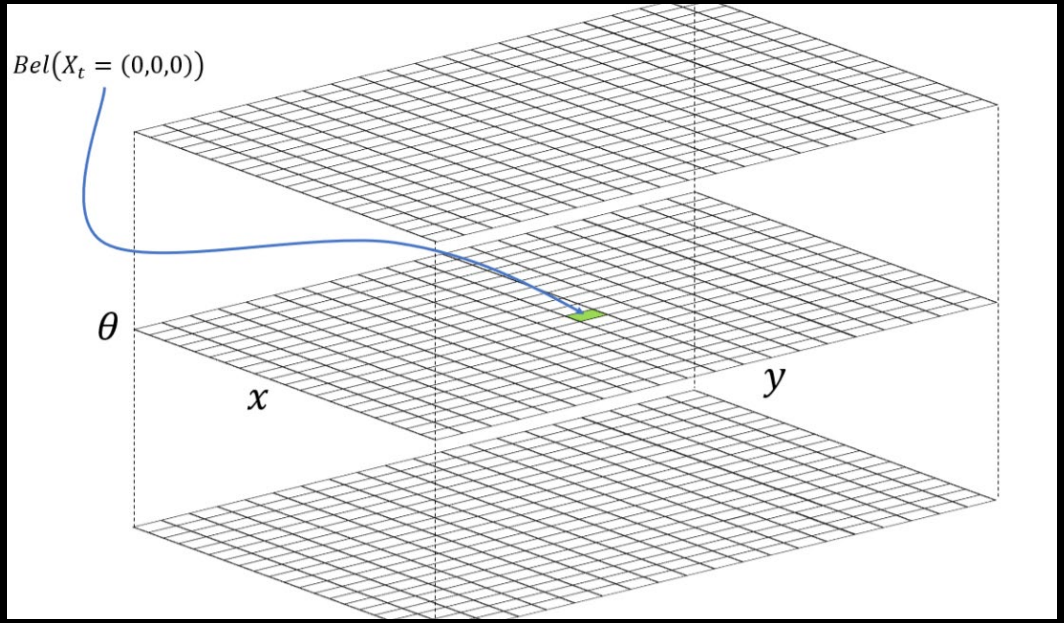
- Good abstract representation
- Tradeoff in # of nodes
  - Complexity vs. accuracy
  - Efficient in large, sparse environments
  - Loss in geometric precision
- Edges can carry weights
- Limited information



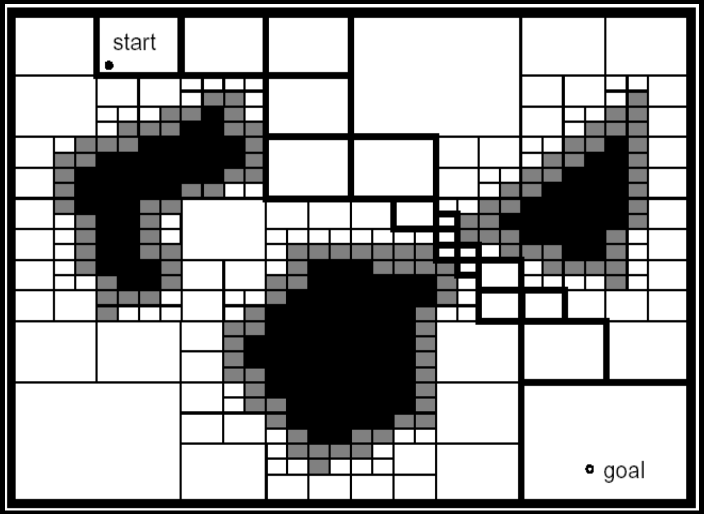
Fixed Cell Decomposition

(Lab 9-13)

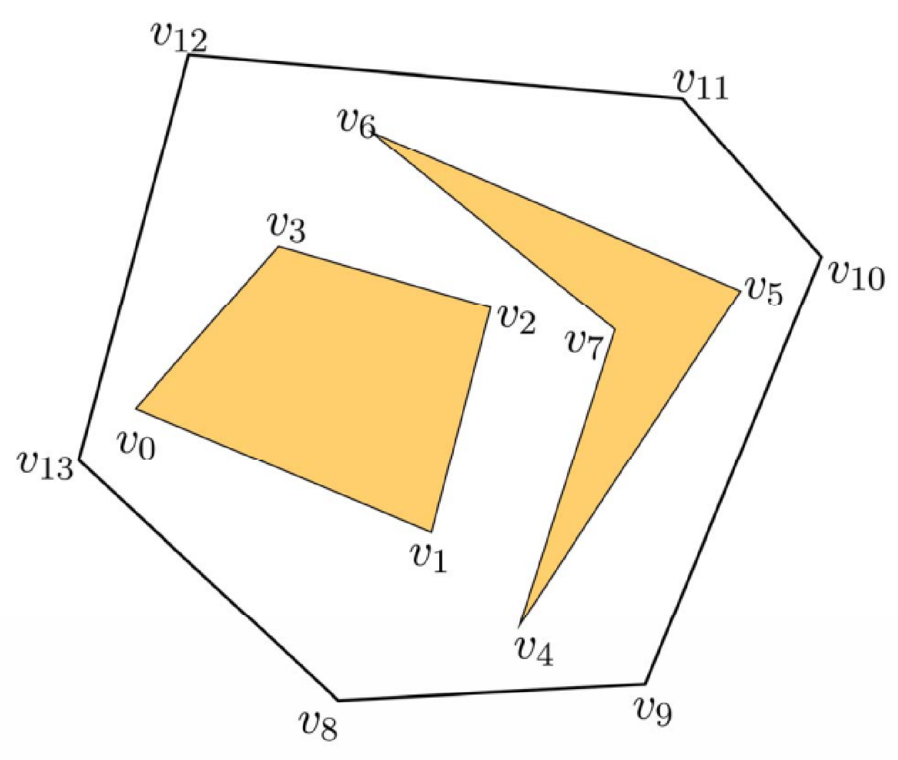




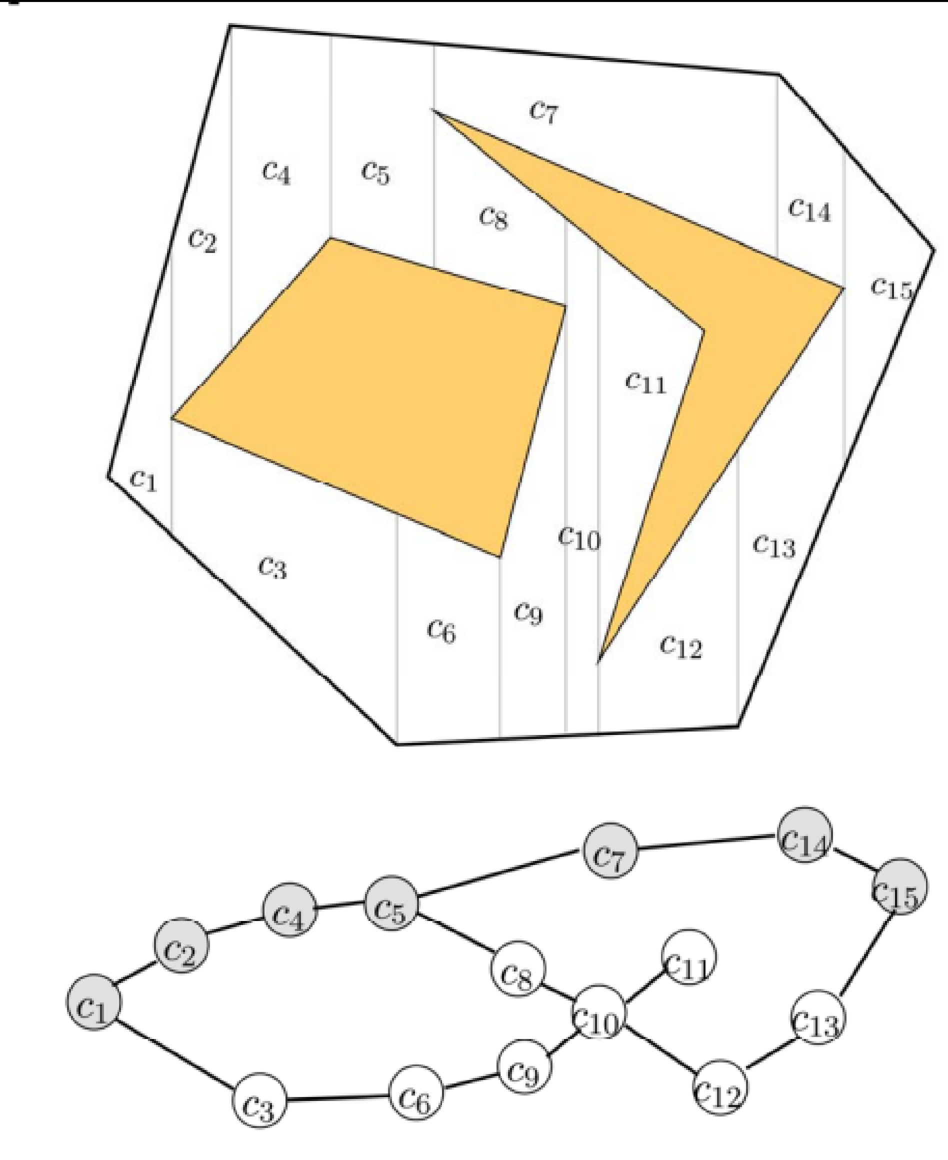
Adaptive Cell Decomposition



Trapezoidal Cell Decomposition

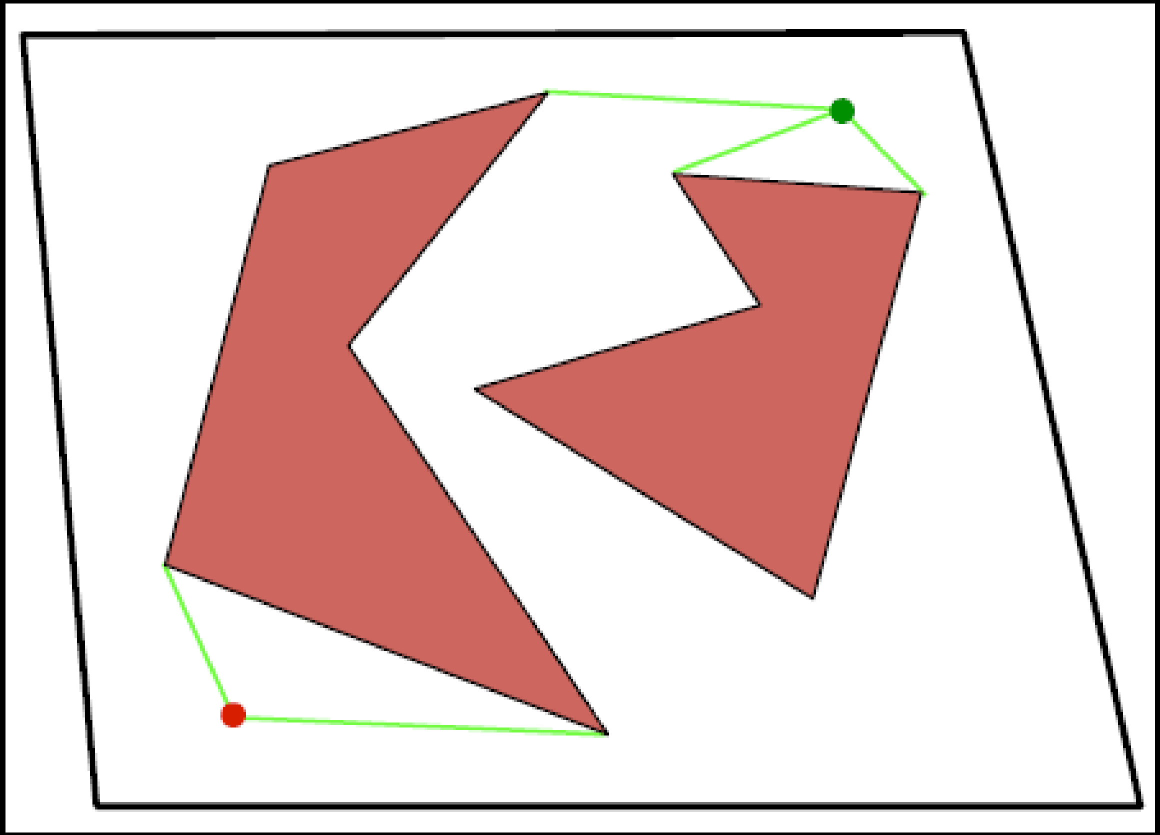


RI 16-735 Howie Choset



Visibility Graphs

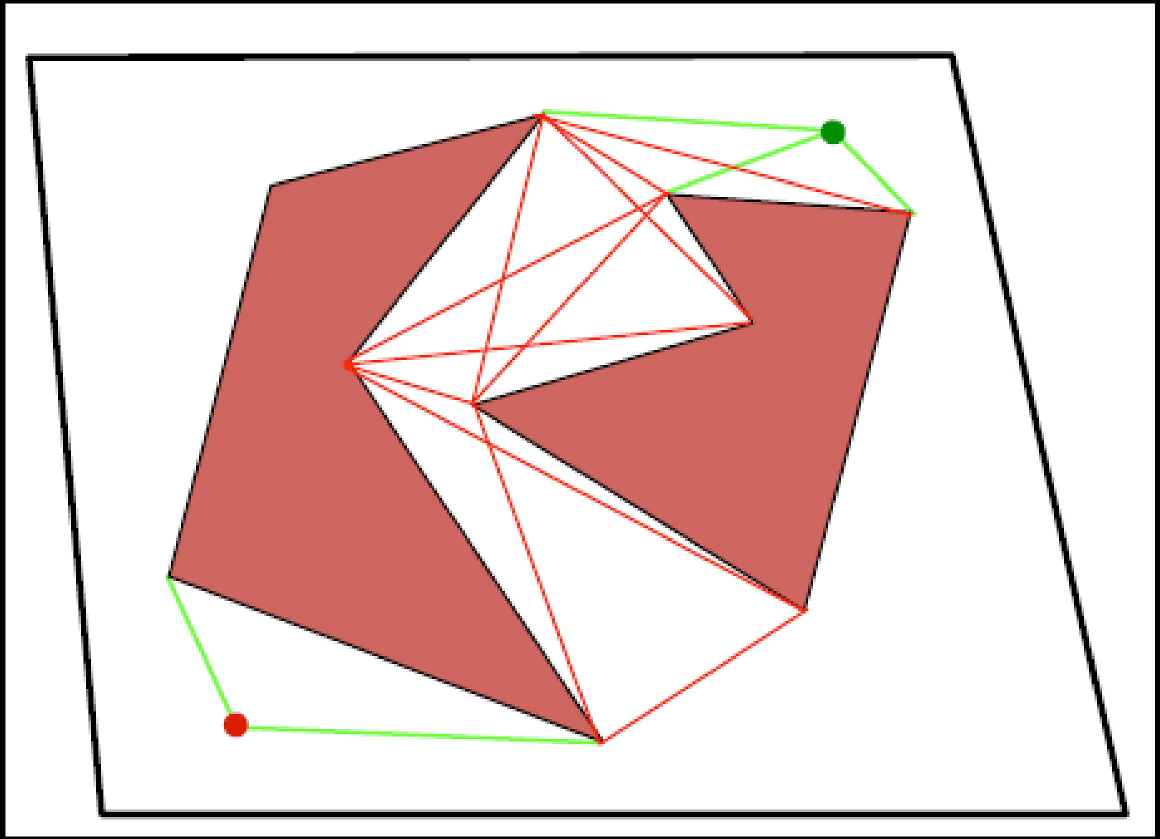
- Connect initial and goal locations with all visible vertices



Ioannis Rekleitis, South Carolina

Visibility Graphs

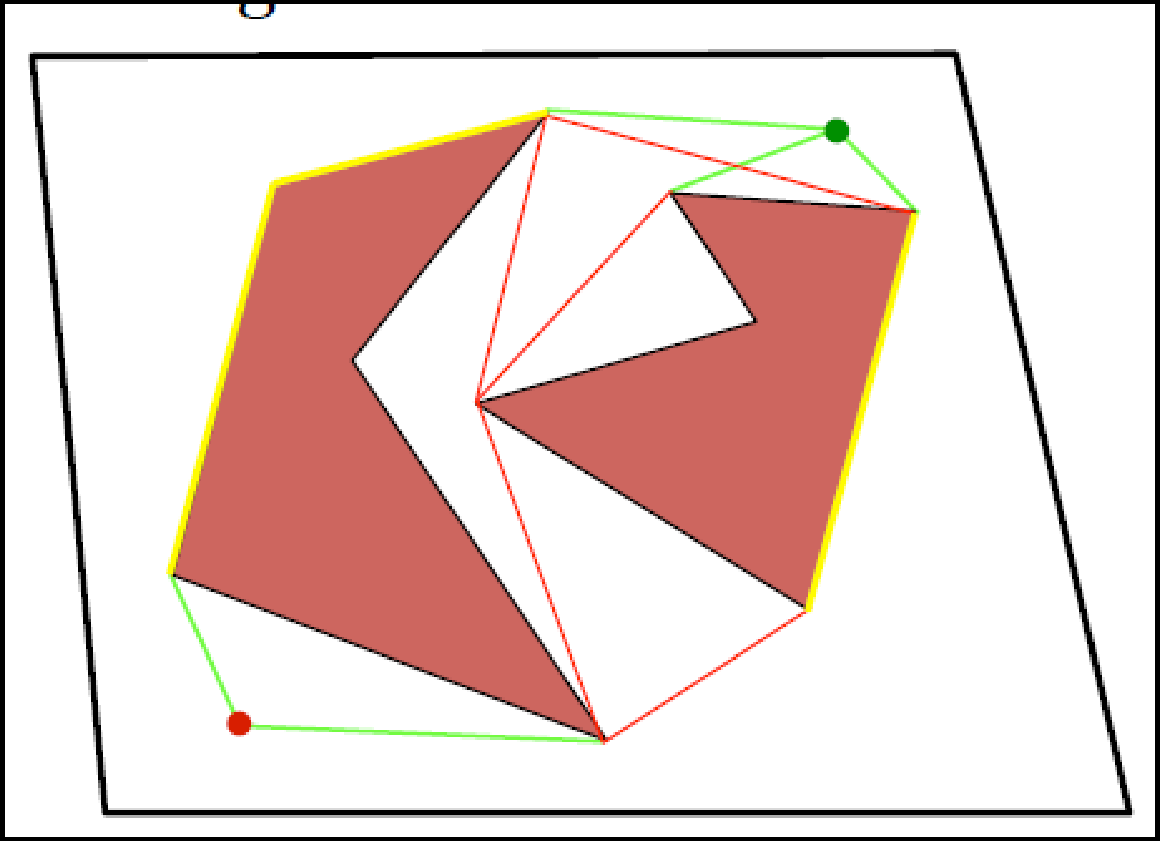
- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex



Ioannis Rekleitis, South Carolina

Visibility Graphs

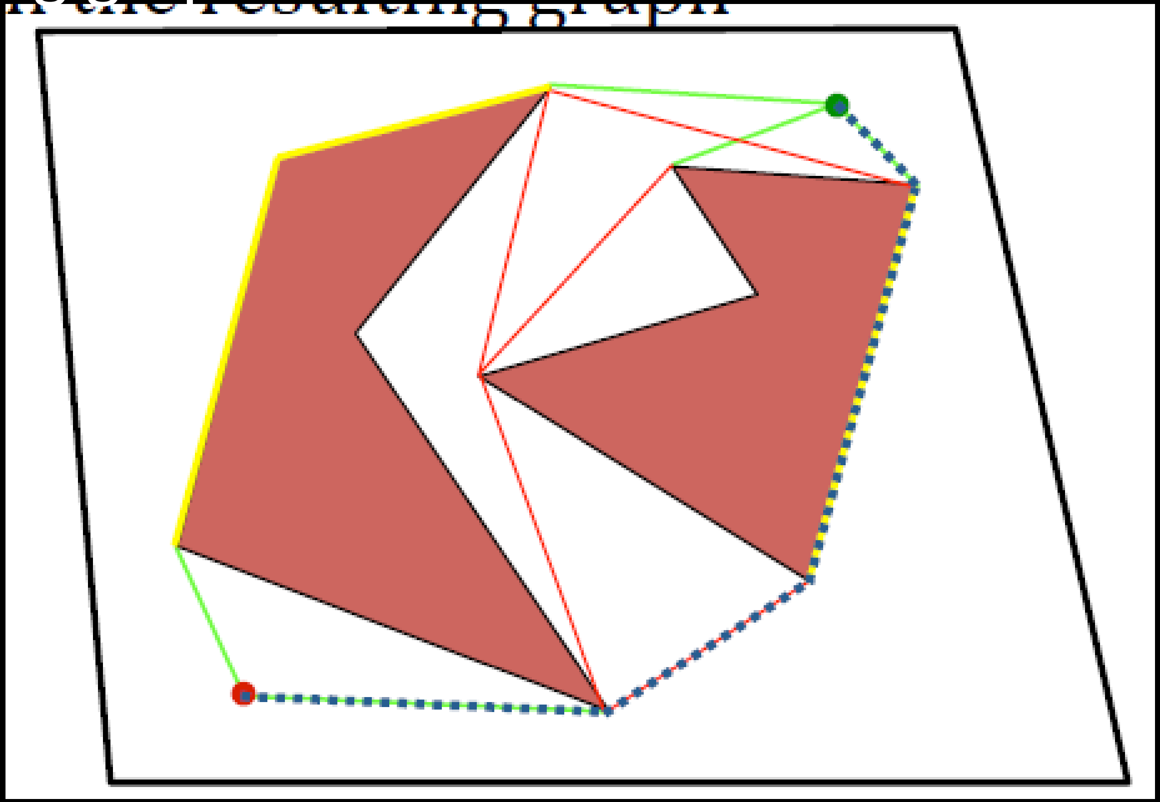
- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle



Ioannis Rekleitis, South Carolina

Visibility Graphs

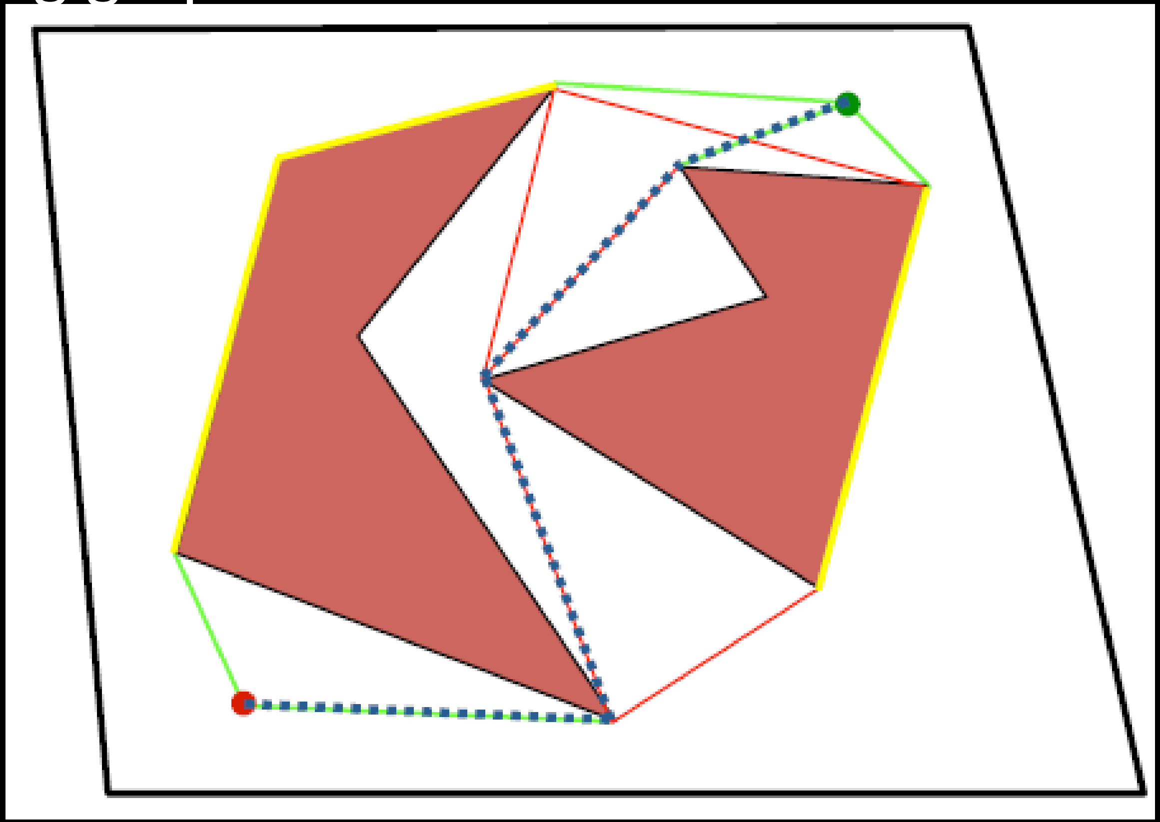
- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle
- Plan on the resulting graph



Ioannis Rekleitis,
South Carolina

Visibility Graphs

- Connect initial and goal locations with all visible vertices
- Connect each obstacle vertex to every visible obstacle vertex
- Remove edges that intersect the interior of an obstacle
- Plan on the resulting graph



Ioannis Rekleitis, South Carolina

Sampling-Based Planners

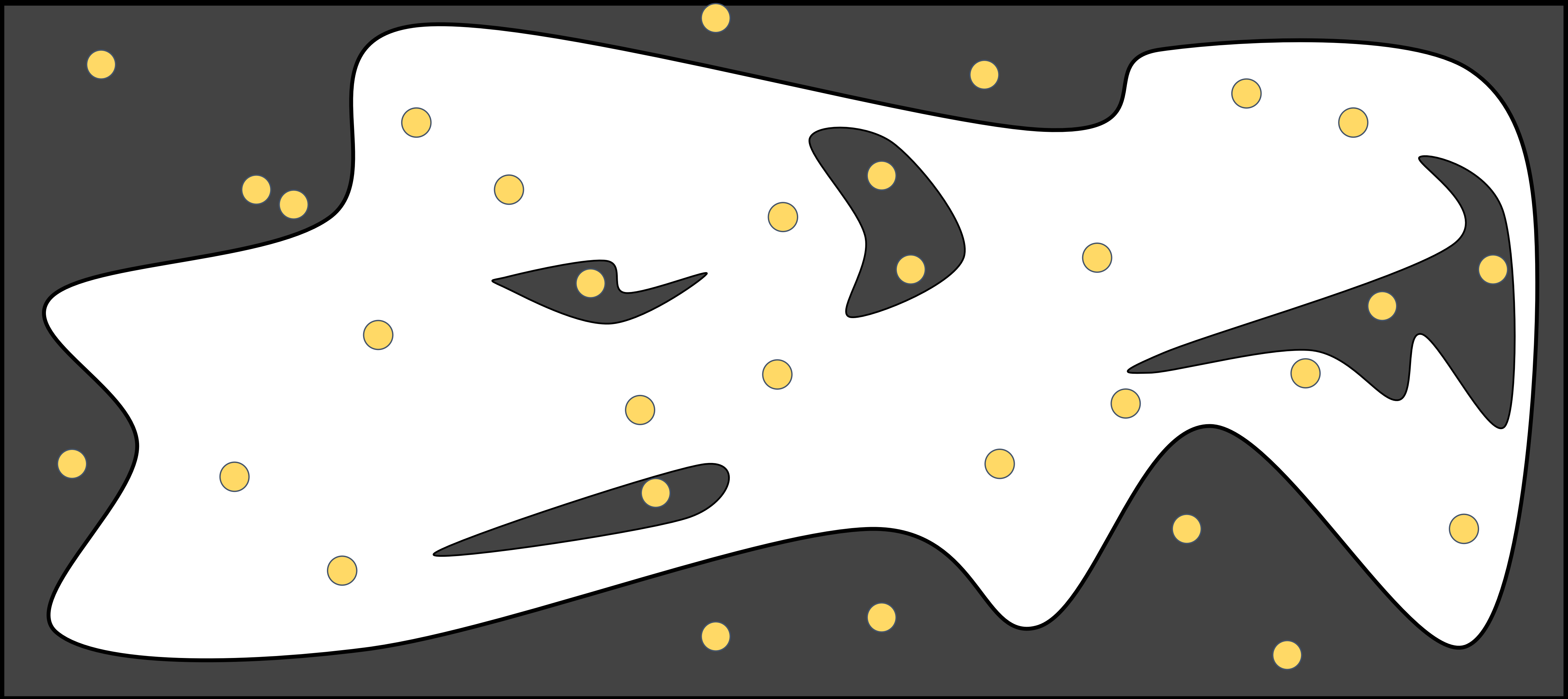
- Explicit geometry-based planners are impractical in high dimensional spaces
- Sampling-based planners
  - Often efficient in high dimensional spaces
  - Rather than computing the C-Space explicitly, we sample it
  - Compute if a robot configuration is in collision
    - Just need forward kinematics for each configuration
    - (Local path plans between each configuration)
- Examples
  - Probabilistic Roadmaps (PRM)
  - Rapidly Exploring Random Trees (RRT)

Probabilistic Roadmaps

Lydia Kavraki, 1996
Rice Univeristy

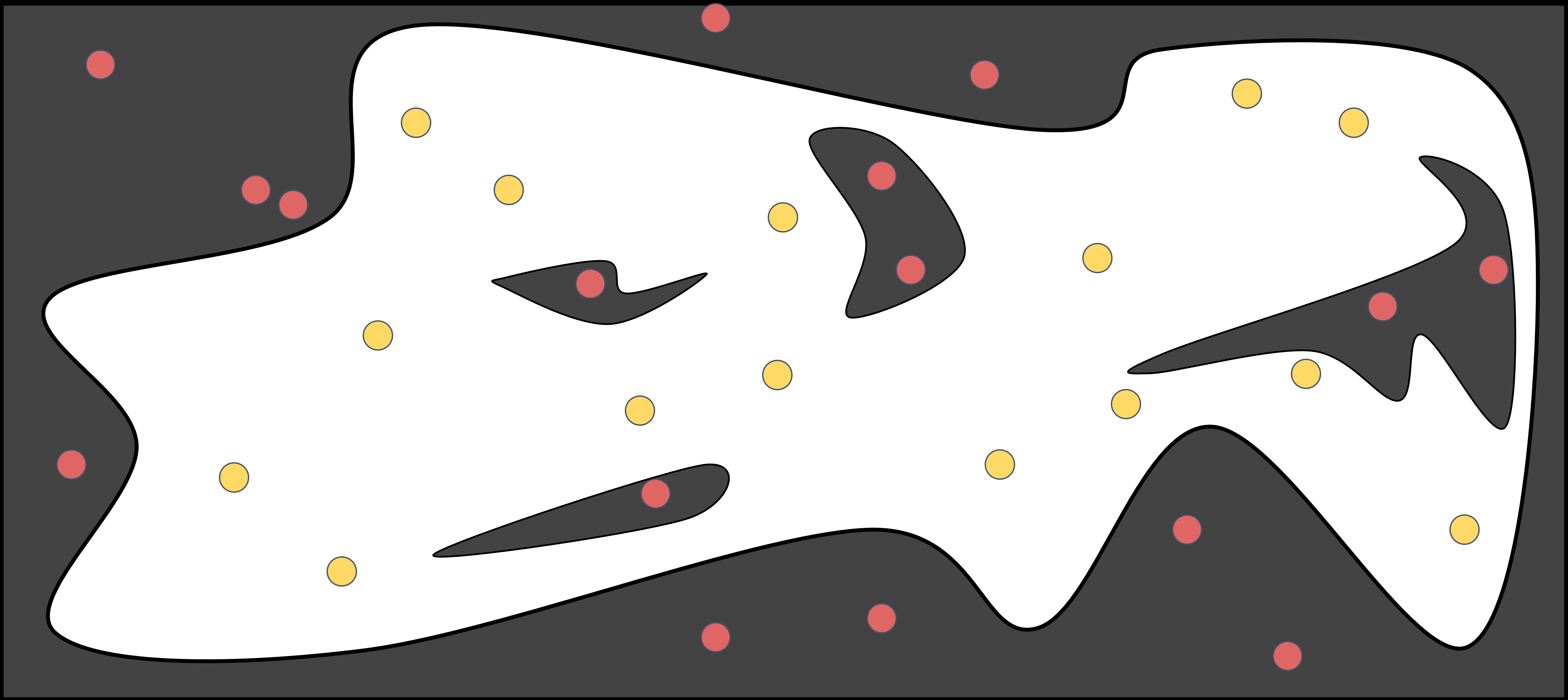
Probabilistic Roadmaps

Configurations are sampled by picking coordinates at random



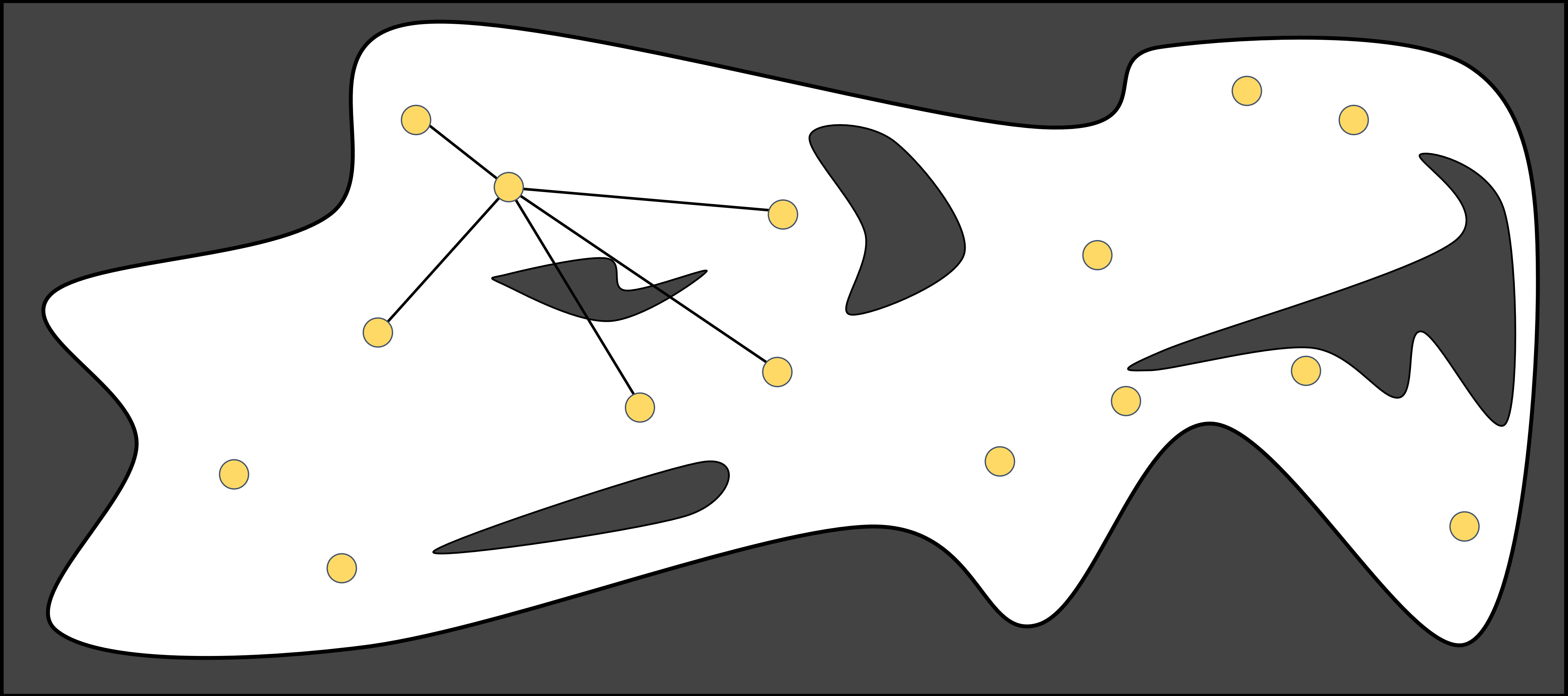
Probabilistic Roadmaps

Sampled configurations are tested for collision



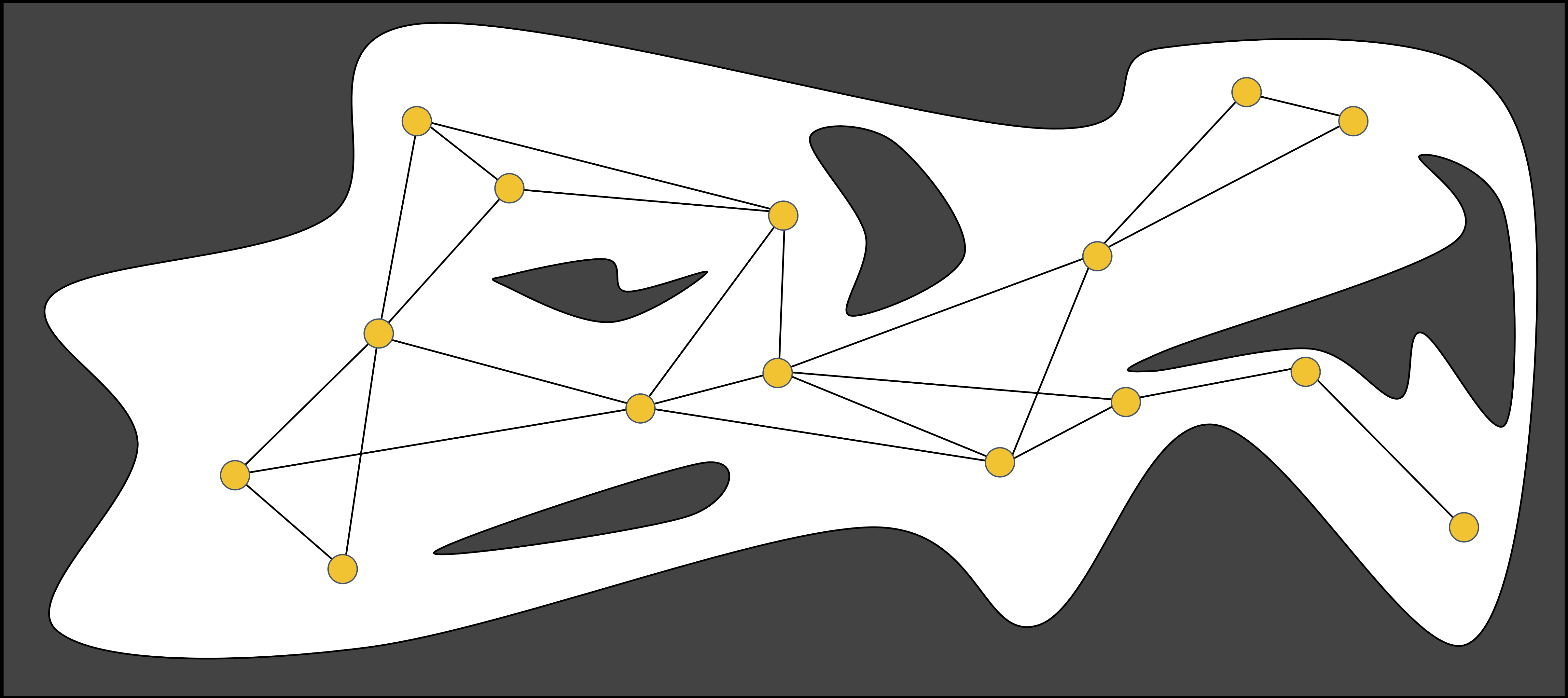
Probabilistic Roadmaps

Each configuration is linked by straight paths to its nearest neighbors



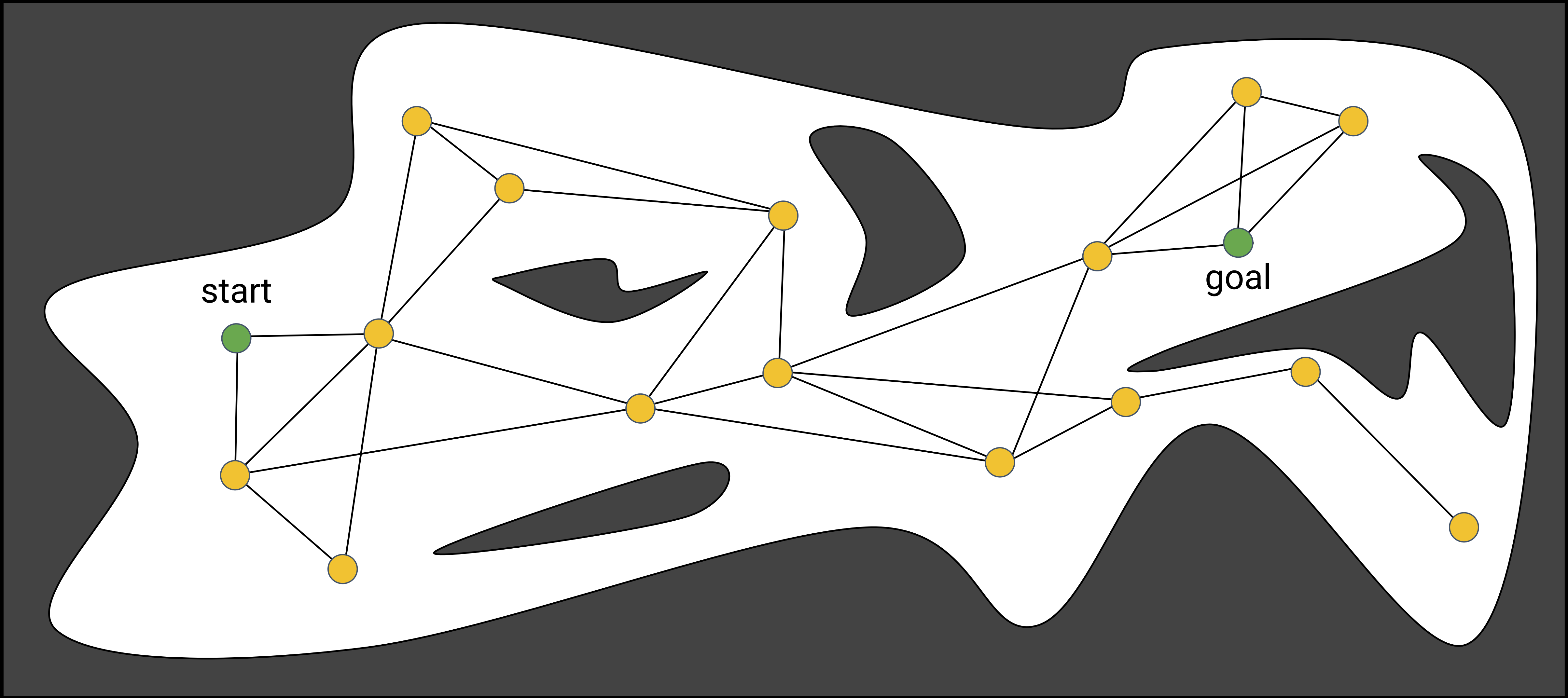
Probabilistic Roadmaps

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



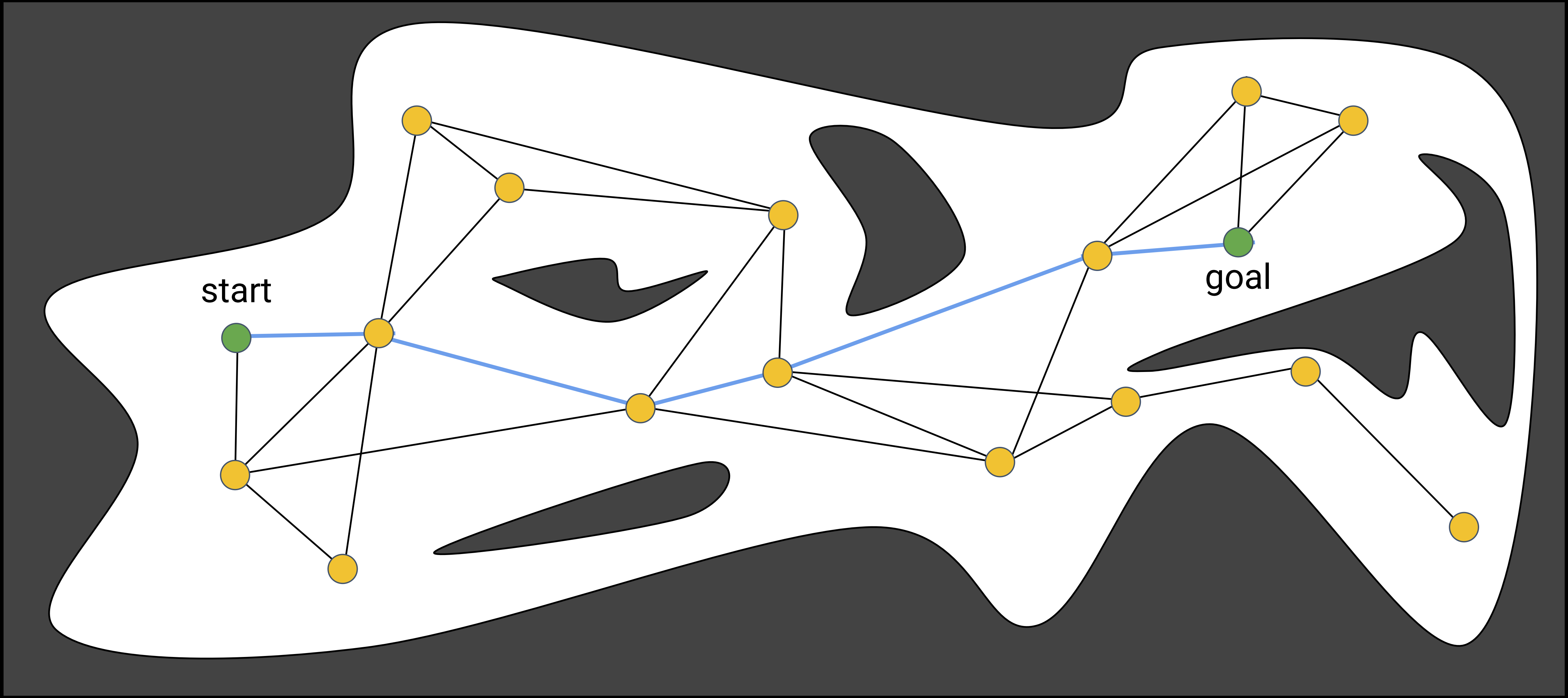
Probabilistic Roadmaps

The start and goal configurations are included as milestones



Probabilistic Roadmaps

The PRM is searched for a path from start to goal



Probabilistic Roadmaps

Constructing the graph

- Initially empty Graph G
- A configuration q is randomly chosen
- If $q \in Q_{free}$ then add to G
  - <need collision detection>
- Repeat until N vertices chosen
- For each q, select k closest neighbors
- Local planner, Δ, connects q to neighbor q’
- If connection is collision free, add edge (q, q’)

Algorithm 6 Roadmap Construction Algorithm

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

```
V ← Ø
E ← Ø
while |V| < n do
   repeat
      q ← a random configuration in Q
   until q is collision-free
   V ← V ∪ {q}
end while
for all q ∈ V do
   N_q ← the k closest neighbors of q chosen from V according to dist
   for all q′ ∈ N_q do
      if (q, q′) ∉ E and Δ(q, q′) ≠ NIL then
         E ← E ∪ {(q, q′)}
      end if
   end for
end for
```

Probabilistic Roadmaps

Finding the Path

- Connect q_{init} and q_{goal} to the roadmap
- Find k nearest neighbors of q_{init} and q_{goal} in roadmap, plan local path Δ
- Compute cost of path
- Repeat until graphs are connected
- Choose cheapest path

Algorithm 7 Solve Query Algorithm

Input:

q_{init}: the initial configuration
q_{goal}: the goal configuration
k: the number of closest neighbors to examine for each configuration
$G = (V, E)$: the roadmap computed by algorithm 6

Output:

A path from q_{init} to q_{goal} or failure

$N_{q_{\text{init}}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
$N_{q_{\text{goal}}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
$V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
set q' to be the closest neighbor of q_{init} in $N_{q_{\text{init}}}$
repeat
  if $\Delta(q_{\text{init}}, q') \neq$ `NIL` **then**
    $E \leftarrow (q_{\text{init}}, q') \cup E$
  else
    set q' to be the next closest neighbor of q_{init} in $N_{q_{\text{init}}}$
  end if
until a connection was succesful or the set $N_{q_{\text{init}}}$ is empty
set q' to be the closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
repeat
  if $\Delta(q_{\text{goal}}, q') \neq$ `NIL` **then**
    $E \leftarrow (q_{\text{goal}}, q') \cup E$
  else
    set q' to be the next closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
  end if
until a connection was succesful or the set $N_{q_{\text{goal}}}$ is empty
$P \leftarrow$ shortest path$(q_{\text{init}}, q_{\text{goal}}, G)$
if P is not empty **then**
  return P
else
  return `failure`
end if

Probabilistic Roadmaps

Finding the Path

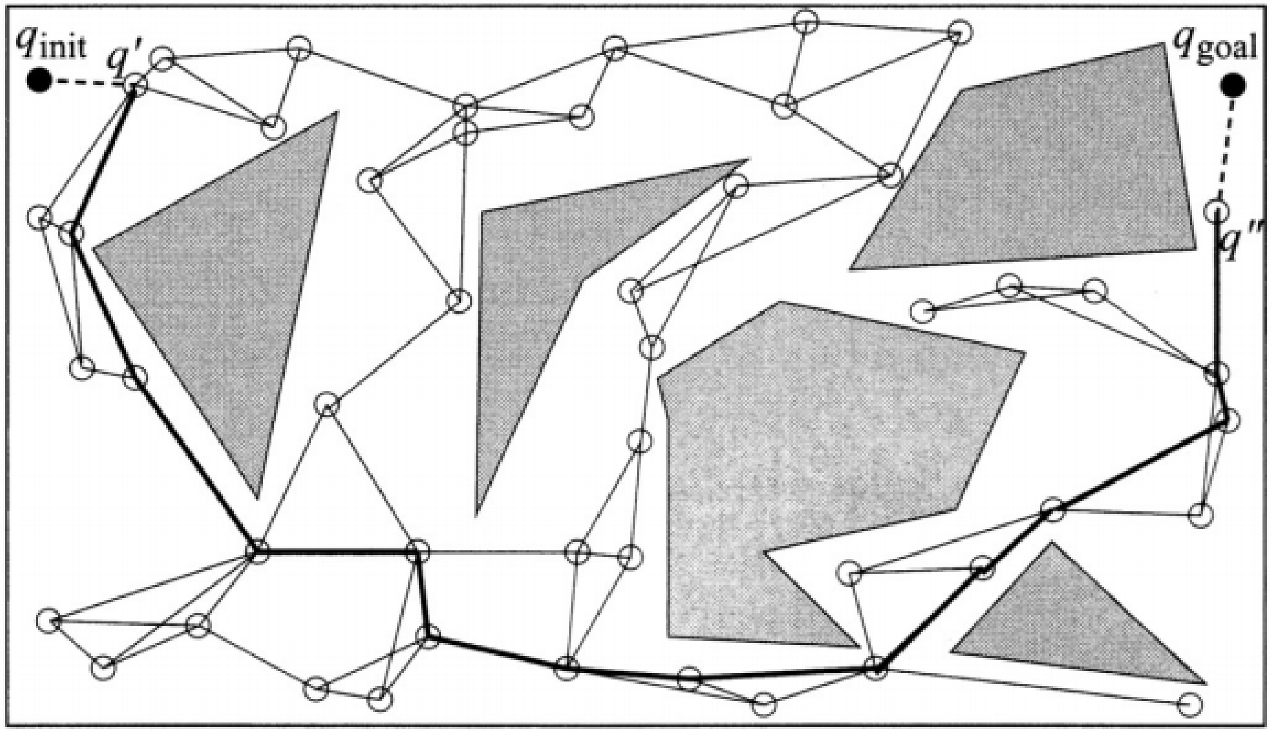
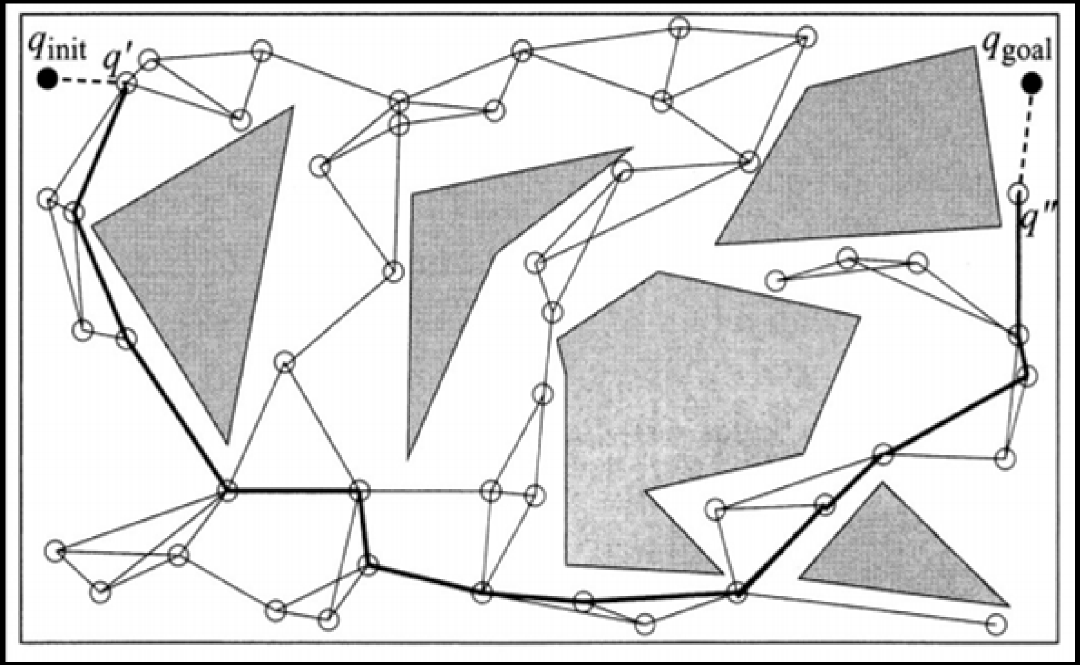


Figure 7.4 An example of how to solve a query with the roadmap from figure 7.3. The configurations q_{init} and q_{goal} are first connected to the roadmap through q' and q''. Then a graph-search algorithm returns the shortest path denoted by the thick black lines.

Probabilistic Roadmaps

Considerations

- Single query/multi query
- How are nodes placed?
  - Uniform sampling strategies
  - Non-uniform sampling strategies
- How are local neighbors found?
- How is collision detection performed?
  - Dominates time consumption in PRMs



Probabilistic Roadmaps

- "Robot Motion Planning on a Chip", Murray et al. RSS 2016
- Company: Real Time Robotics
  - PRM on an FPGA
  - Collision detection circuits on each edge in logic gates for massive parallel operation
  - 6DOF planning in <1ms

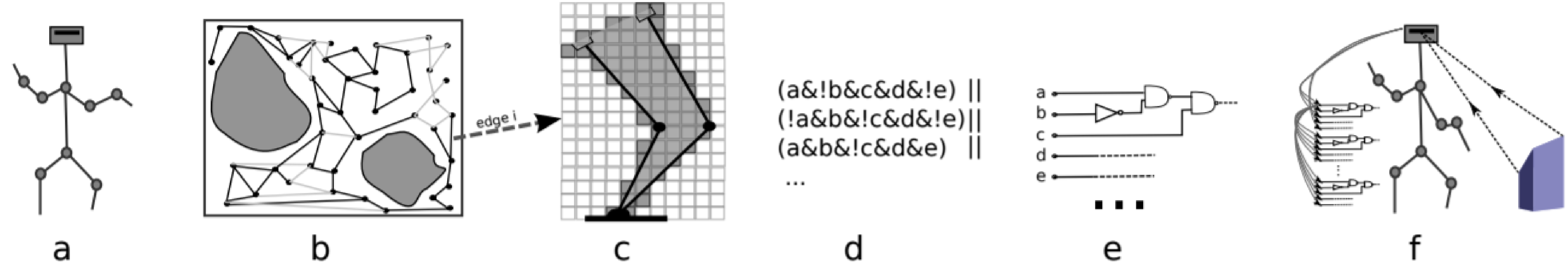


Fig. 3: Our process for producing robot-specific motion planning circuitry. Given a robot description (a), we construct a PRM (b), most likely subsampled for coverage from a much larger PRM. We discretize the robot's reachable space into depth pixels and, for each edge i on the PRM, precompute all the depth pixels that collide with the corresponding swept volume (c). We use these values to construct a logical expression that, given the coordinates of a depth pixel encoded in binary, returns `true` if that depth pixel collides with edge i (d); this logical expression is optimized and used to build a collision detection circuit (CDC) (e). For each edge in the PRM there is one such circuit. When the robot wishes to construct a motion plan, it perceives its environment, determines which depth pixels correspond to obstacles, and transmits their binary representations to every CDC (f). All CDCs perform collision detection *simultaneously, in parallel* for each depth pixel, storing a bit which indicates

Prof. Kirstin Hagelskjær Petersen
kirstin@cornell.edu

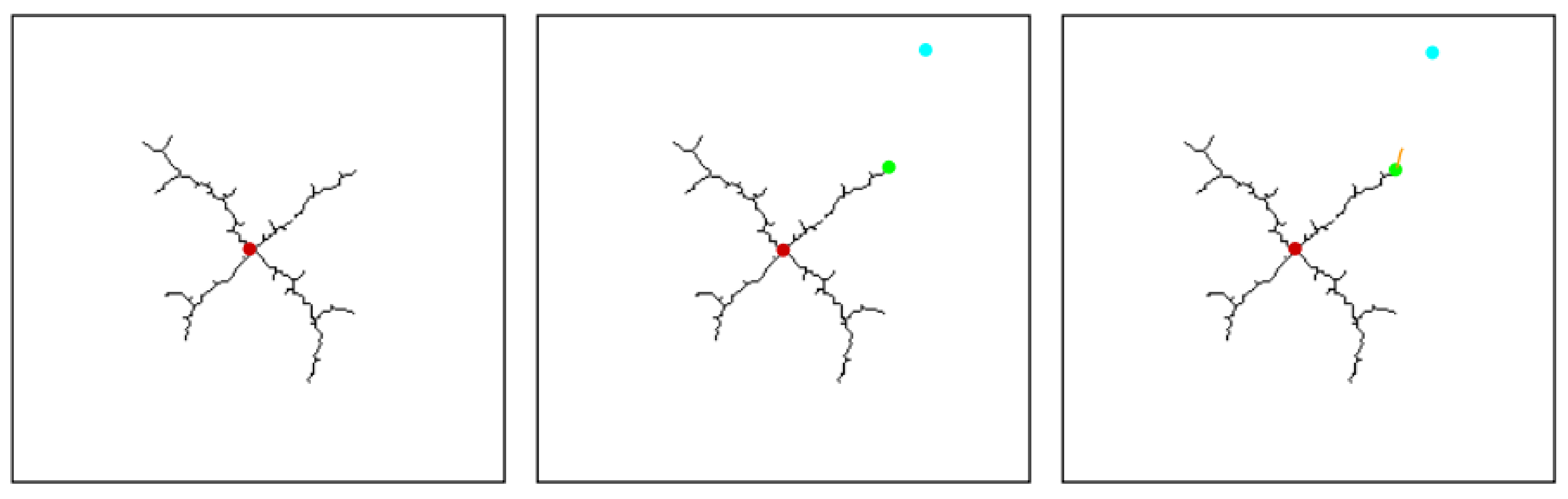
Rapidly Exploring Random Trees (RRT)

Rapidly Exploring Random Trees (RRT) – Uniform/biased sampling

Aaron Becker, UH, Wolfram Player example

S. LaValle, UIUC / Oculus

Rapidly Exploring Random Trees (RRT)



1. Maintain a tree rooted at the starting point
2. Choose a point at random from free space
3. Find the closest configuration already in the tree
4. Extend the tree in the direction of the new configuration

Rapidly Exploring Random Trees (RRT)

```
Algorithm BuildRRT
  Input: Initial configuration q_init, number of vertices K, incremental distance Δq)
  Output: RRT graph G
  G.init(q_init)
  for k = 1 to K
    qrand ← RAND_CONF()
    qnear ← NEAREST_VERTEX(qrand, G)
    qnew ← NEW_CONF(qnear, qrand, Δq)
    G.add_vertex(qnew)
    G.add_edge(qnear, qnew)
  return G
```

Rapidly Exploring Random Trees (RRT) - Considerations

- **Sensitive to step-size (Δq)**
  - Small: many nodes, closely spaced, slowing down nearest neighbor computation
  - Large: Increased risk of suboptimal plans / not finding a solution
- **How are samples chosen?**
  - Uniform sampling may need too many samples to find the goal
  - Biased sampling towards goal can ease this problem
- **How are local paths generated?**
- **How are closest neighbors found?**

Rapidly Exploring Random Trees (RRT) - Variations

- RRT Connect
  - Two trees rooted at start and goal locations
- RRT*
  - Converges towards an optimal solution
  - Aaron Becker, UH, Wolfram Player example

- A*-RRT

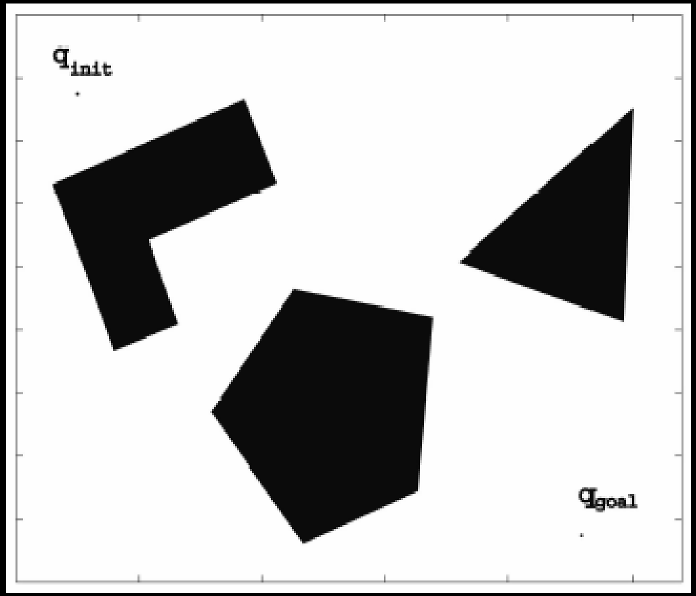
- Informed RRT*, Real-Time RRT*, Theta*-RRT, etc.

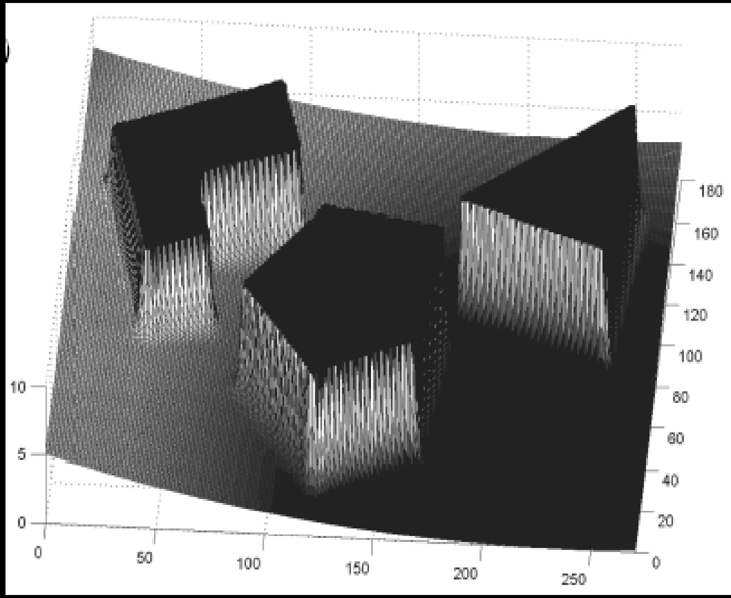
Planning using Potential Fields

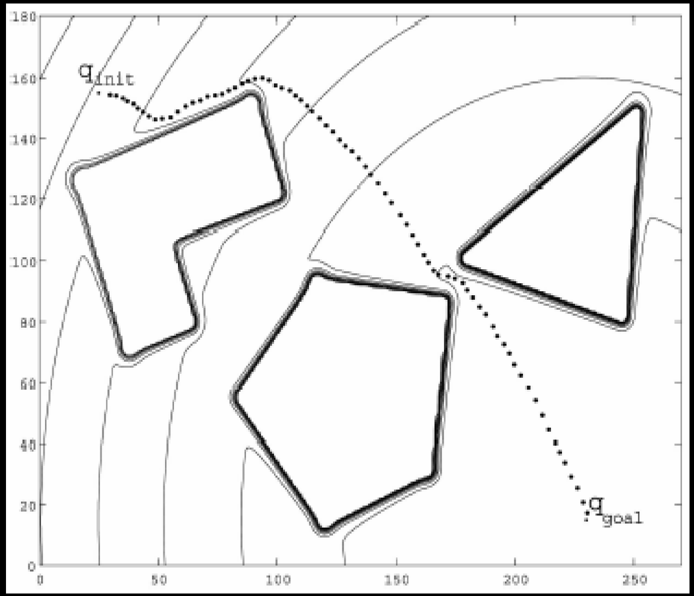
- Robot is treated as a point under the influence of a (continuous) artificial potential field
- Robot movement becomes similar to a ball rolling down a hill

Khatib, Stanford

Khatib, 1986

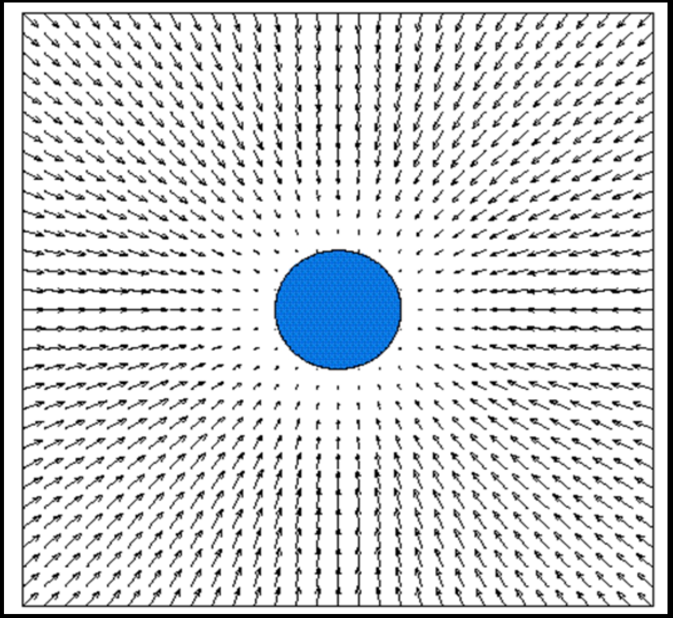


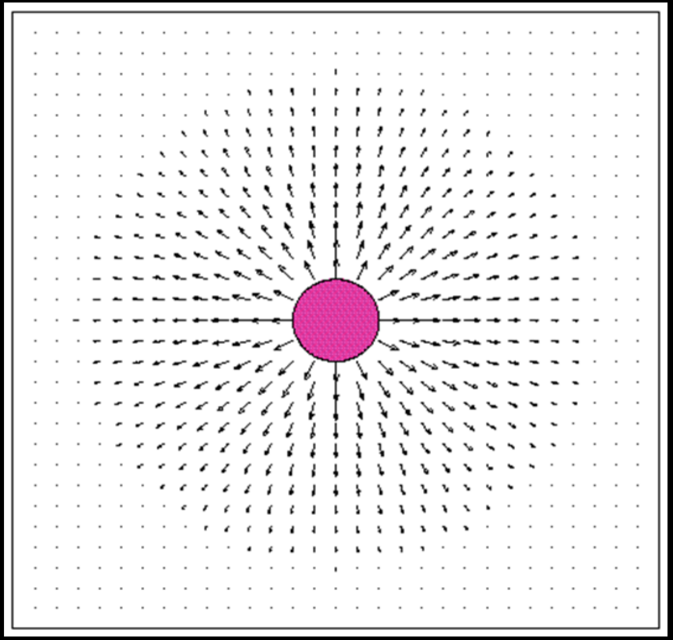




Planning using Potential Fields

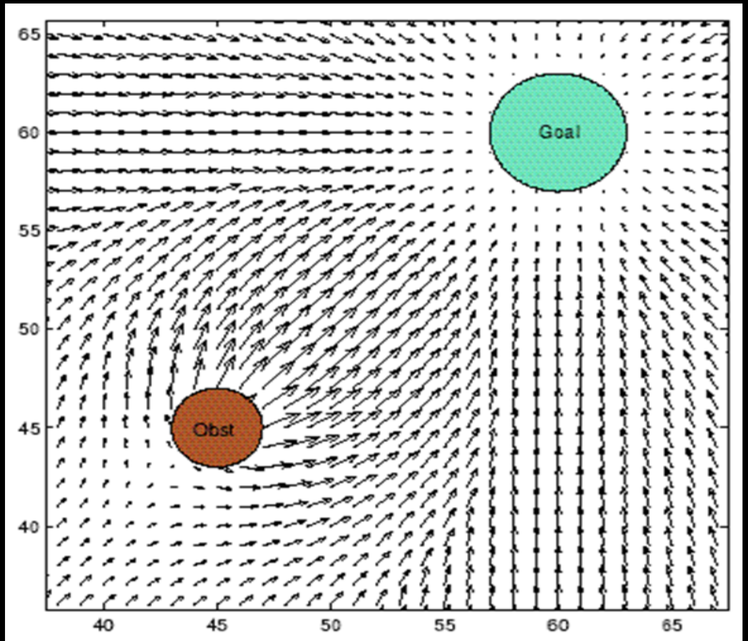
- The goal creates an attractive force
  - Modeled as a spring
  - Hooke's law: F = -kX
  - "Parabolic attractor"
  - $U_{att}(q) = k_{att}(q - q_{goal})^2$
  - $F_{att}(q) = -\nabla U_{att} = k_{att}\left(q - q_{goal}\right)$
- Obstacles are repulsive forces
  - Modeled as charged particles
  - Coulomb's law: $F = k\, q_1 q_2 / r^2$
  - $U_{rep}(q) = \begin{cases} 0.5k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & , if\ \rho(q) \leq \rho_0 \\ 0 & , if\ \rho(q) \geq \rho_0 \end{cases}$
  - $F_{rep}(q) = \begin{cases} k_{rep}\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\frac{1}{\rho(q)^2}\frac{q - q_{obst}}{\rho(q)} & , if\ \rho(q) \leq \rho_0 \\ 0 & , if\ \rho(q) \geq \rho_0 \end{cases}$

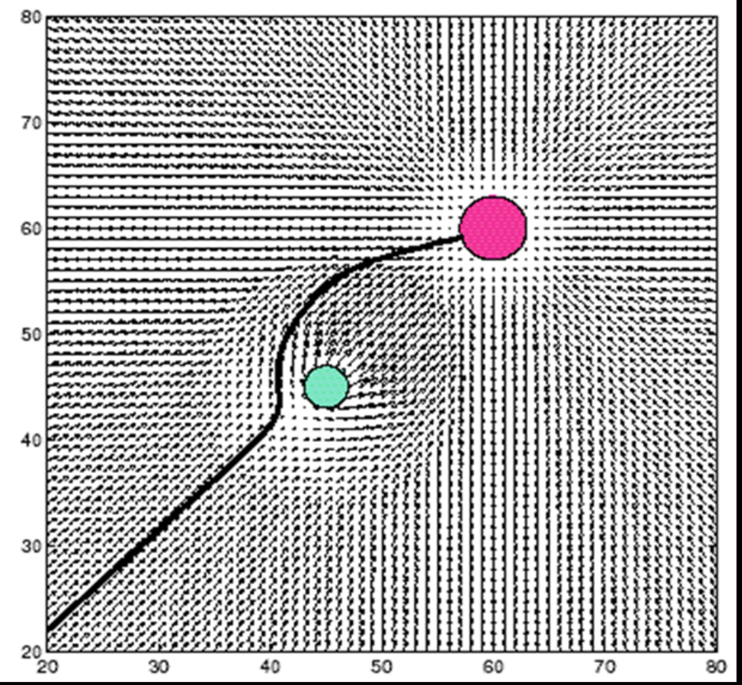




Planning using Potential Fields

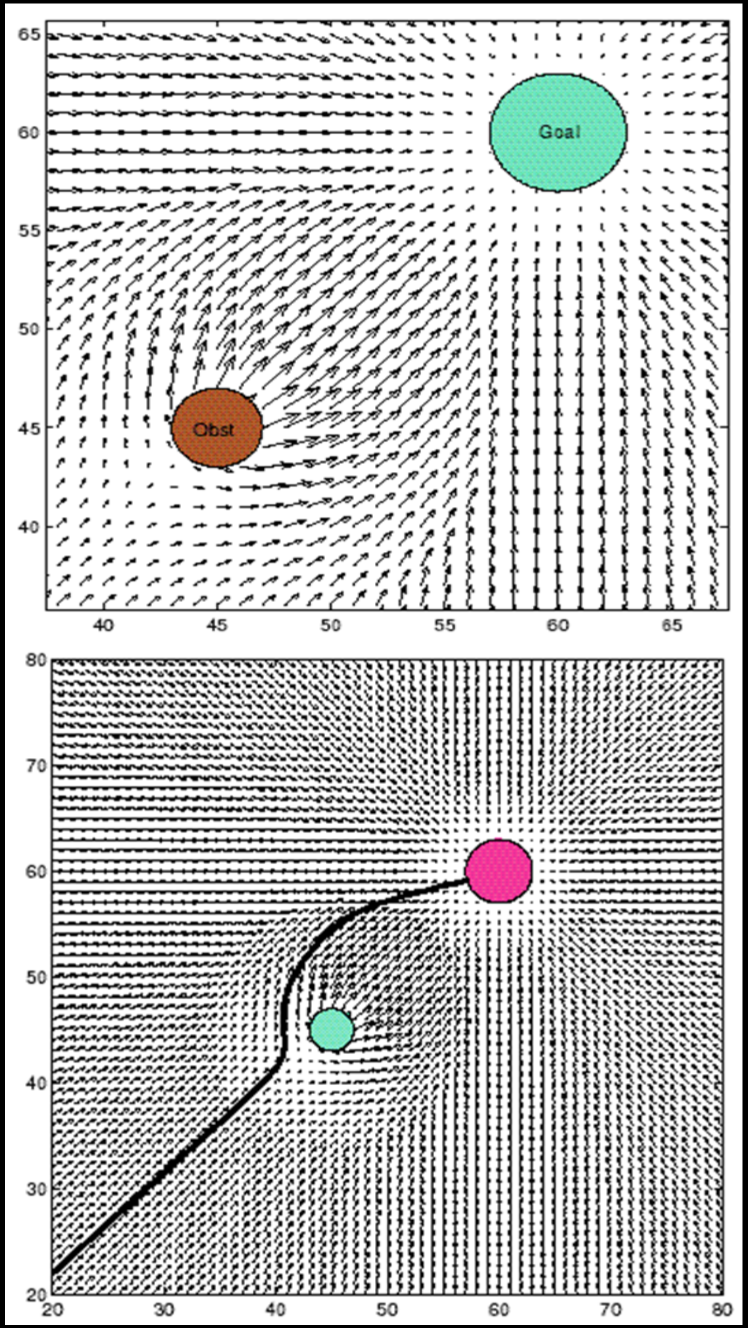
- Goal generates attractive force
  - Modeled as a spring
  - Hooke's law: F = -kX
- Obstacle are repulsive forces
  - Modeled as charged particles
  - Coulomb's law: $F = k\, q_1 q_2 / r^2$
- Model navigation as the sum of forces on the robot
  - The overall potential field
    - $U(q) = U_{goal}(q) + \sum U_{obstacles}(q)$
  - Robot motion is proportional to induced force
    - $\mathrm{F(q)} = -\nabla U(q)$
  - e.g. 2 DOF robot will experience
    - $F(q) = -\nabla U(q) = \left(\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y}\right)$



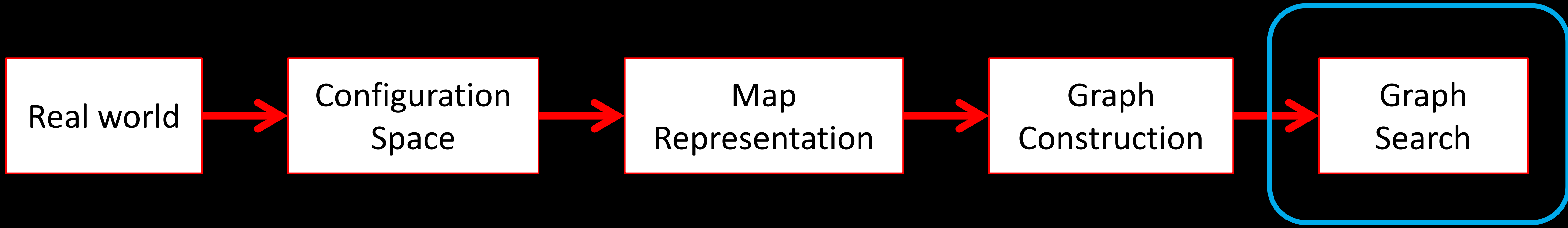


Planning using Potential Fields

- Goal generates attractive force
  - Modeled as a spring
  - Hooke’s law: $F = -kX$
- Obstacle are repulsive forces
  - Modeled as charged particles
  - Coulomb's law: $F= k\, q_1 q_2 / r^2$
- Model navigation as the sum of forces on the robot
- Pitfalls / local minima
  - U-shaped obstacles
  - Long walls
  - Solutions
    - Incorporate high-level planner
    - Incorporate procedural planner
    - Adapt the field to have gradual repulsion
    - Adding stochasticity



Global Motion Planning with Maps



https://pythonrobotics.readthedocs.io/en/latest/modules/path_planning.html#basic-rrt

- Breadth first
- Depth first
- Dijstra
- A*