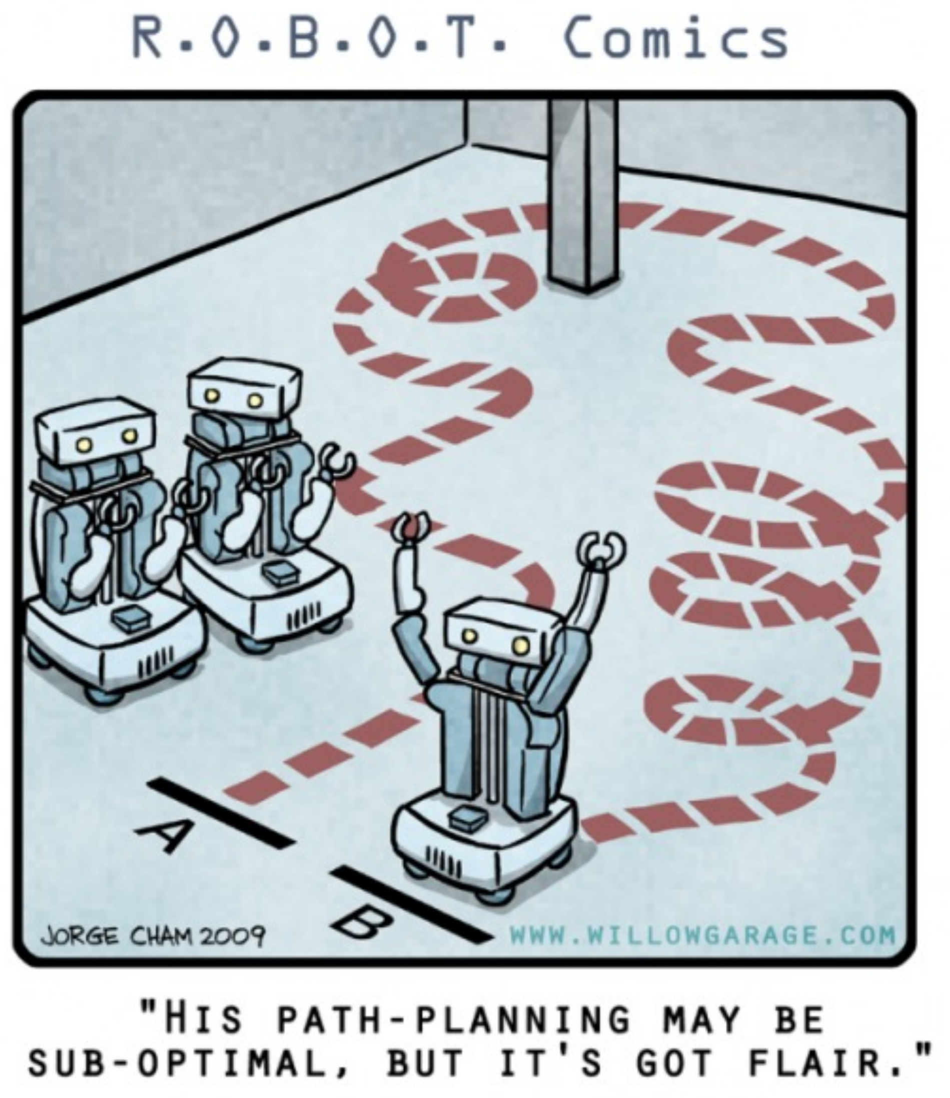
Path Planning

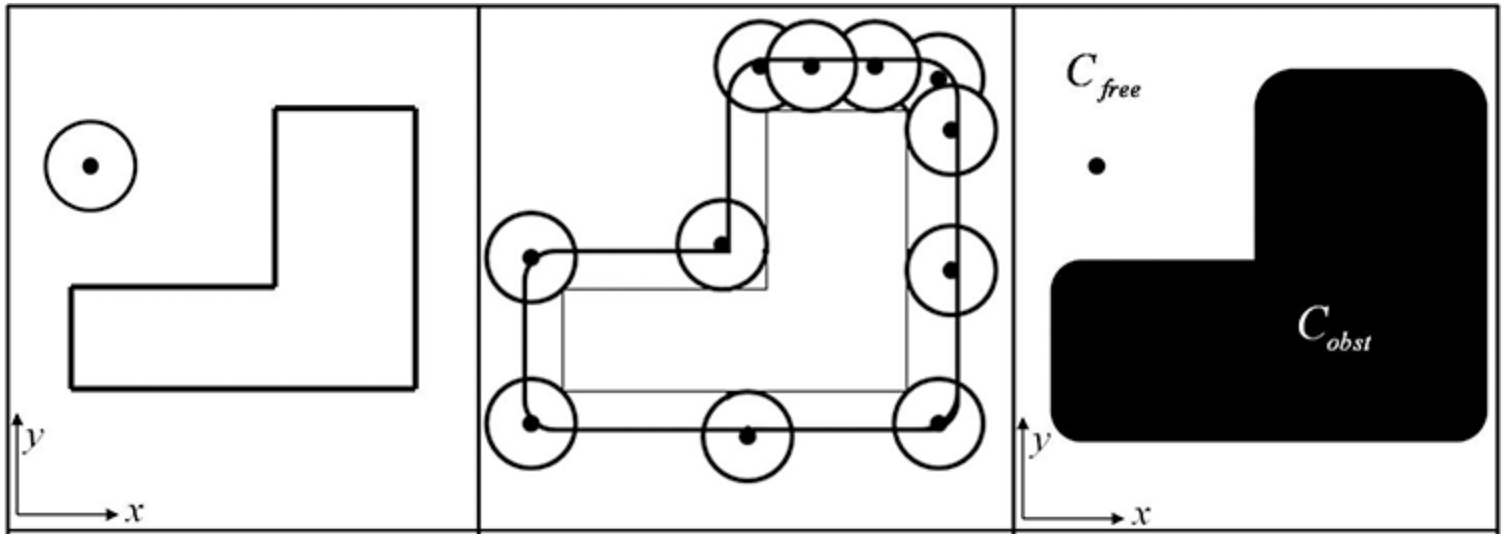
ROB550

Path Planning



Configuration Space

- Portion of space the robot can occupy.
- Reduces robot to a point.

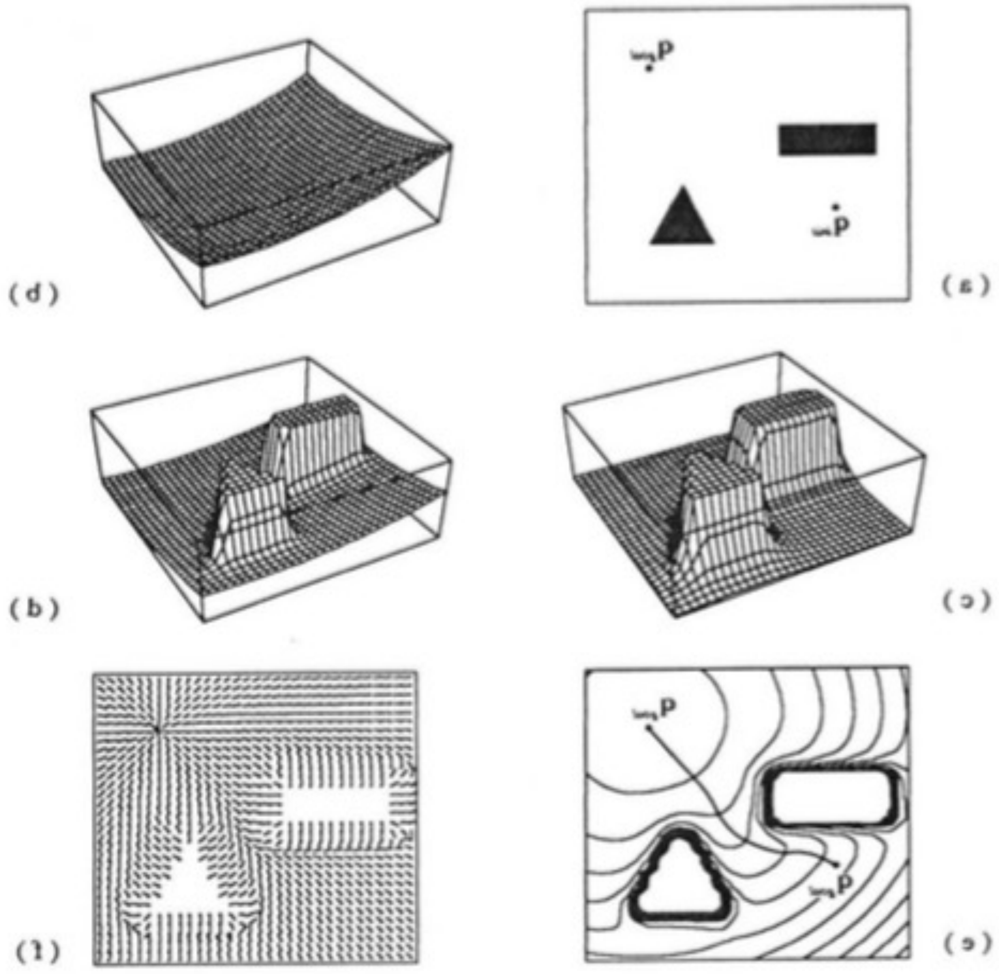


Mobile robot workspace.	The process of constructing the C-space.	Robot C-space.

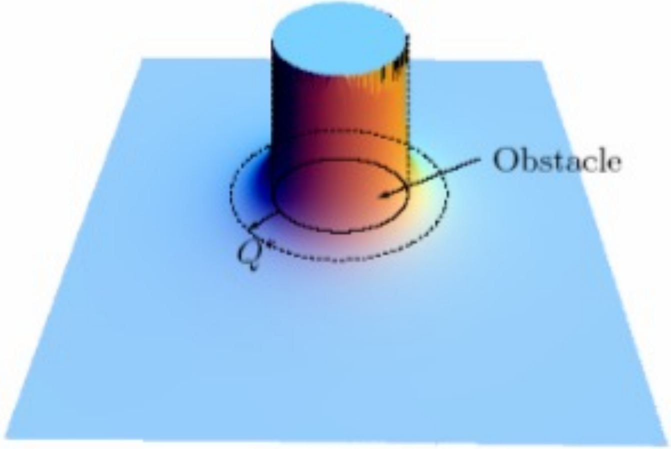
Potential Field Planning

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

$$F(q) = -\nabla U(q)$$



Repulsive Potential for Obstacles



$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(q)} - \frac{1}{Q^*}\right)^2, & D(q) \leq Q^*, \\ 0, & D(q) > Q^*, \end{cases}$$

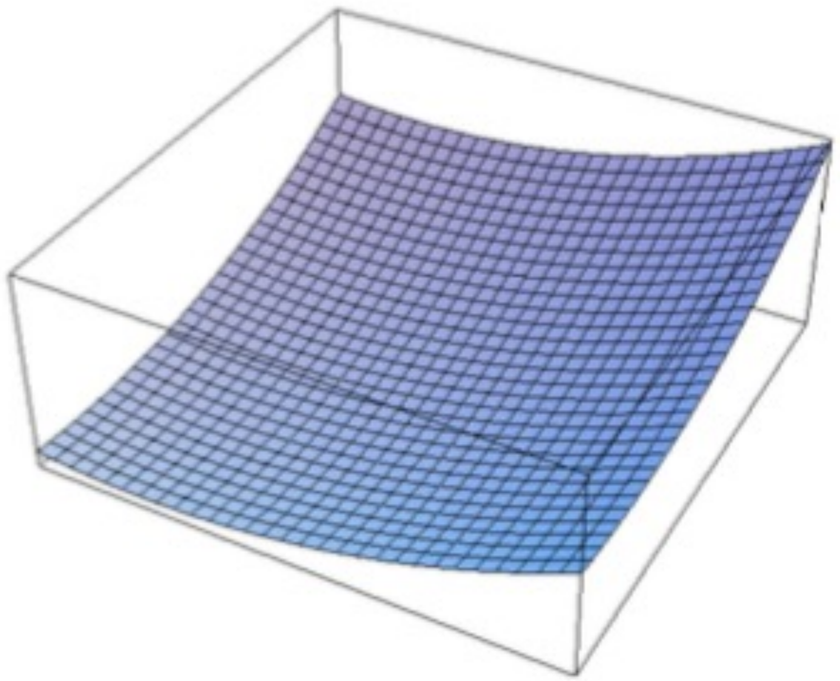
whose gradient is

$$\nabla U_{\text{rep}}(q) = \begin{cases} \eta\left(\frac{1}{Q^*} - \frac{1}{D(q)}\right)\frac{1}{D^2(q)}\nabla D(q), & D(q) \leq Q^*, \\ 0, & D(q) > Q^*, \end{cases}$$

Attractive Potential for Goal

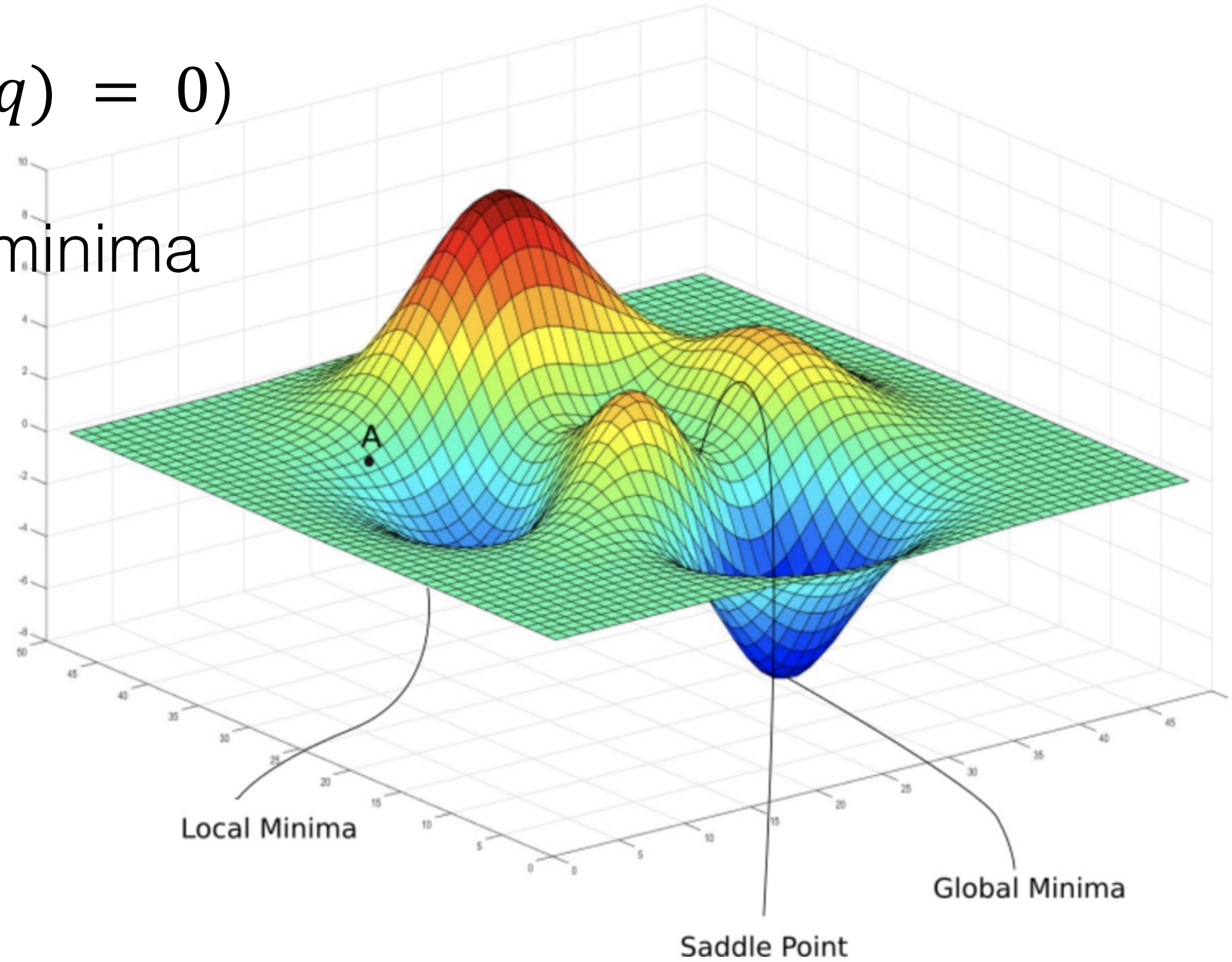
$$U_{\text{att}}(q) = \begin{cases} \frac{1}{2}\zeta d^2(q, q_{\text{goal}}), & d(q, q_{\text{goal}}) \le d^*_{\text{goal}}, \\ d^*_{\text{goal}}\zeta d(q, q_{\text{goal}}) - \frac{1}{2}\zeta (d^*_{\text{goal}})^2, & d(q, q_{\text{goal}}) > d^*_{\text{goal}}. \end{cases}$$

$$\nabla U_{\text{att}}(q) = \begin{cases} \zeta(q - q_{\text{goal}}), & d(q, q_{\text{goal}}) \le d^*_{\text{goal}}, \\ \frac{d^*_{\text{goal}}\zeta(q - q_{\text{goal}})}{d(q, q_{\text{goal}})}, & d(q, q_{\text{goal}}) > d^*_{\text{goal}}, \end{cases}$$



Potential Field

- Critical points (where $\nabla U(q) = 0$) are problematic
- Represent local maxima, minima and saddle points.



Using Potential Field Planners

- Navigation Functions are special potential field constructions with no local minima, only one minimum point for the goal
- Typically use Potential Field planner only for very local planning
- A Wavefront planner is a simple potential field planner on a grid, is not optimal, but does avoids critical points.

Wavefront Planner

- Assign obstacles in occupancy grid a value of 1 and goal a value of 2.

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Wavefront Planner

- Starting with the goal set adjacent cells to +1 and repeat…

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

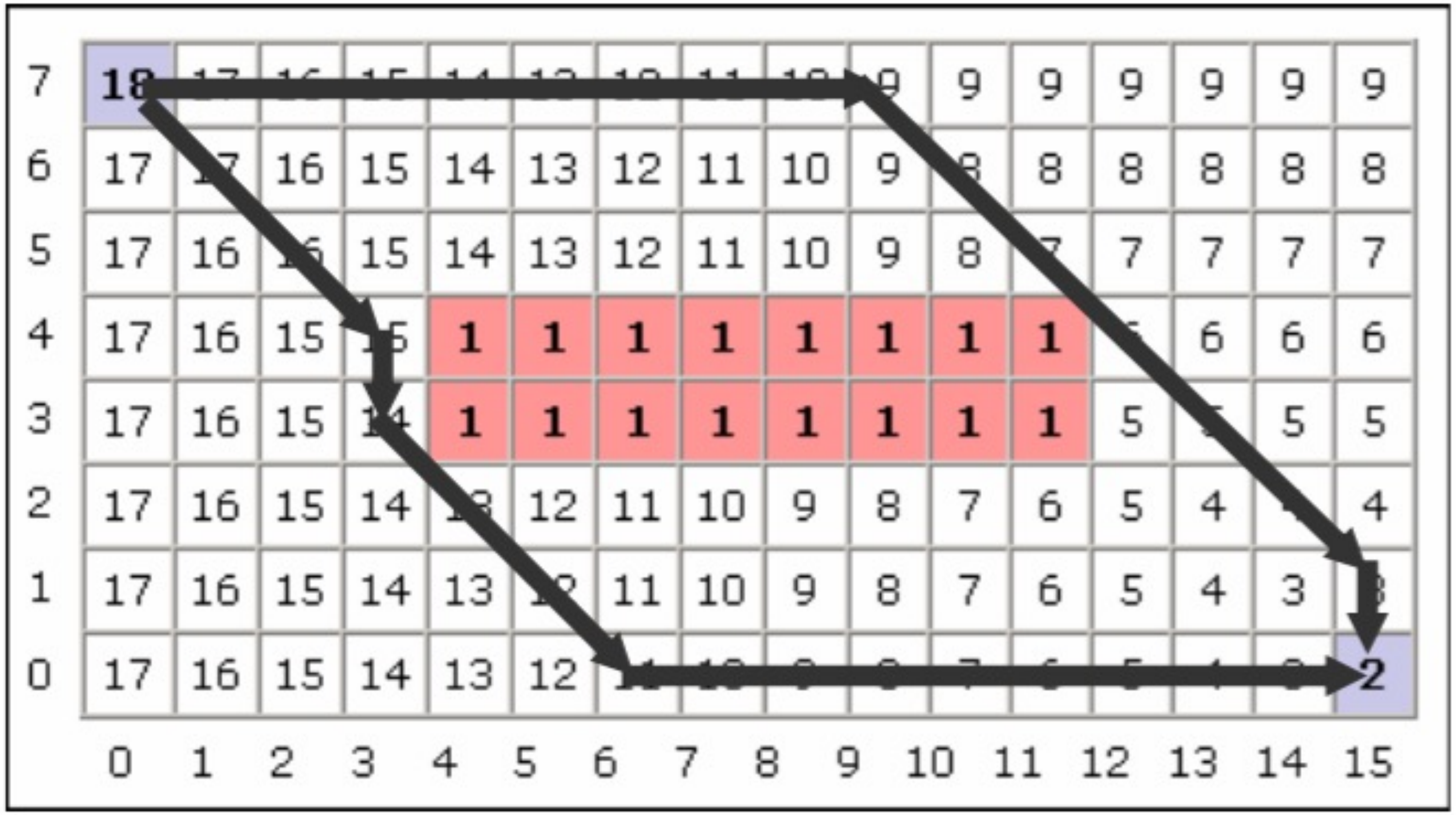
Wavefront Planner

- Completed when start cell has a number.
- Only non-navigable areas should be 1.

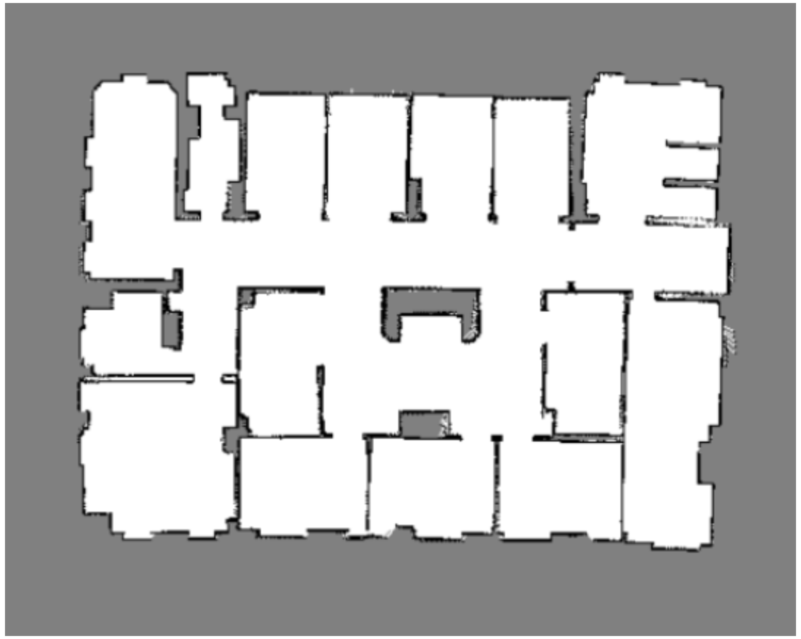
7	**18**	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	17	16	15	14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Wavefront Planner

- Shortest path is found by always moving to a cell with a lower value.

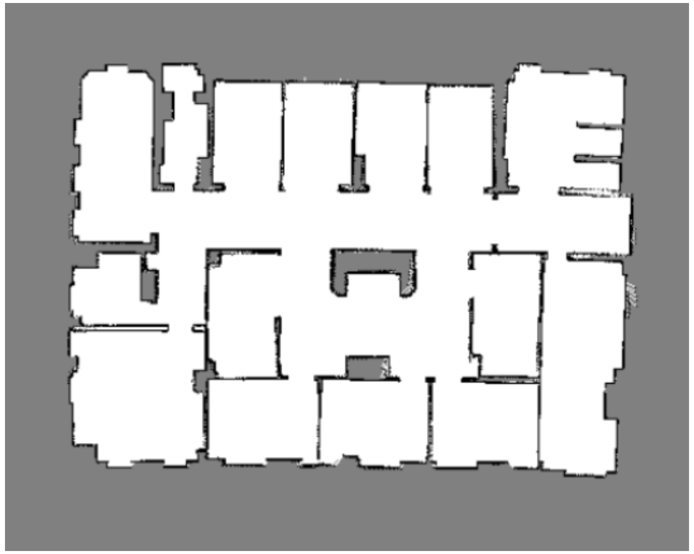


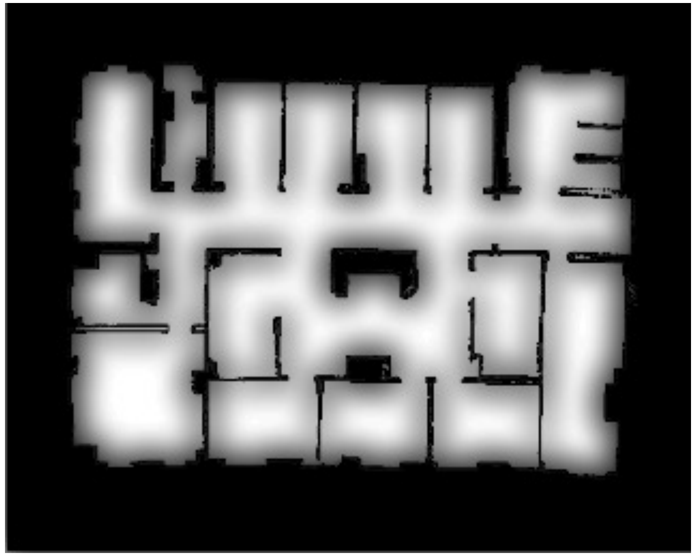
C-Space in Grid



- Grow all occupied cells by radius of the robot

Obstacle Distance Grid

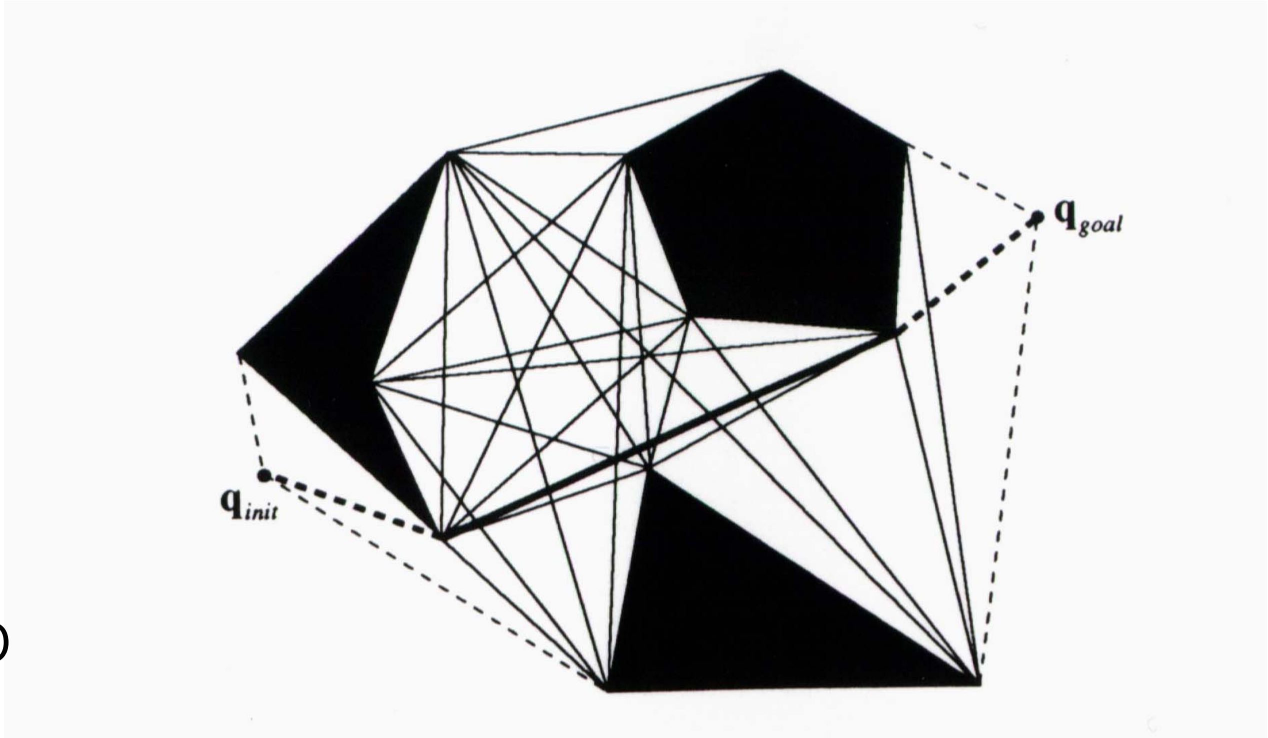




- Store distance to nearest obstacle in each cell
- If distance < robot radius, cannot traverse
- Allows taking obstacle distance into heuristic
- Brushfire Algorithm using priority queue

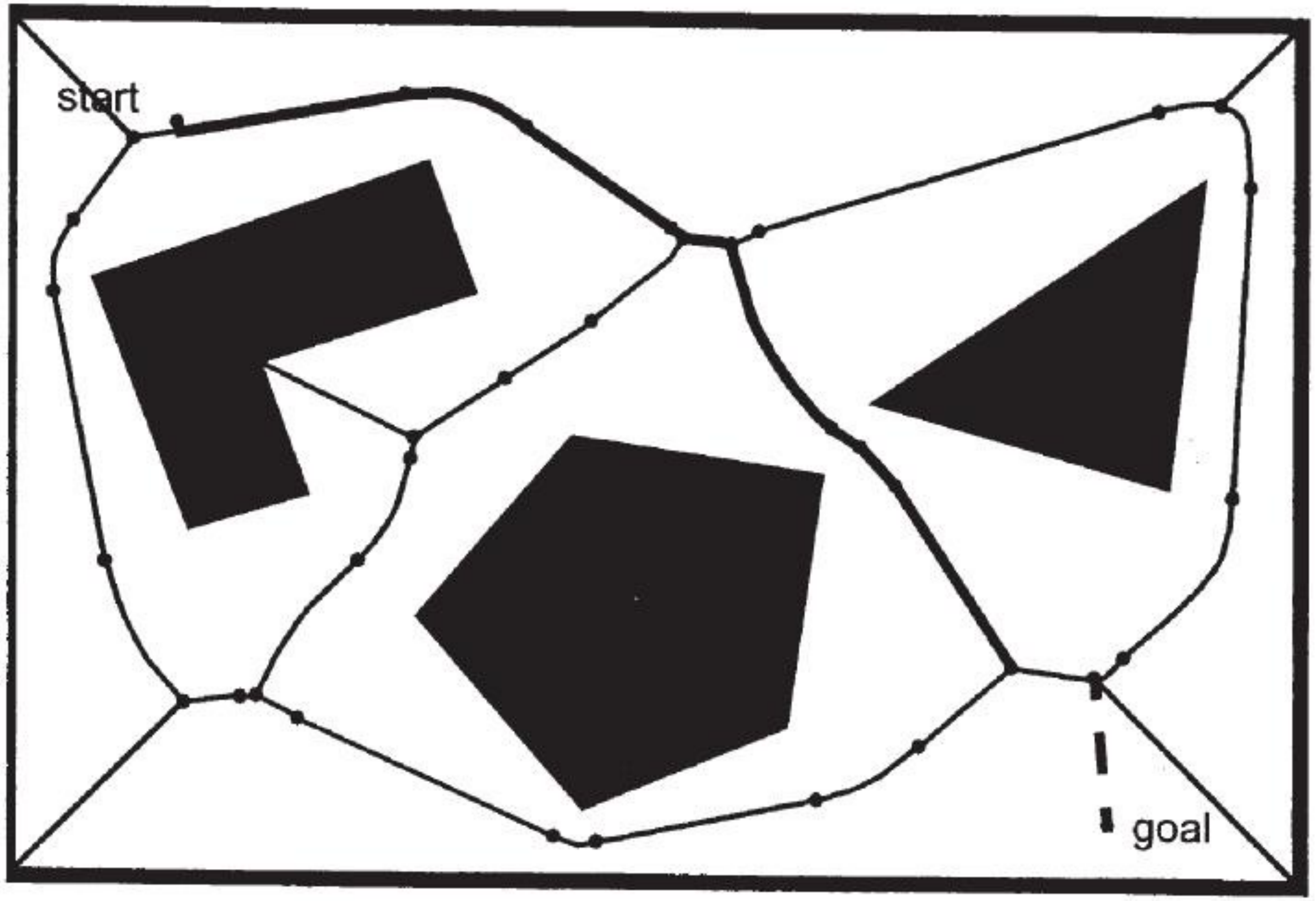
Visibility Graph

- Make obstacles in C-space polygons
- Set node at each vertex
- Set node at start and goal
- Edges between nodes based on visibility
- Must use collision detection to build edges



Voronoi Graph

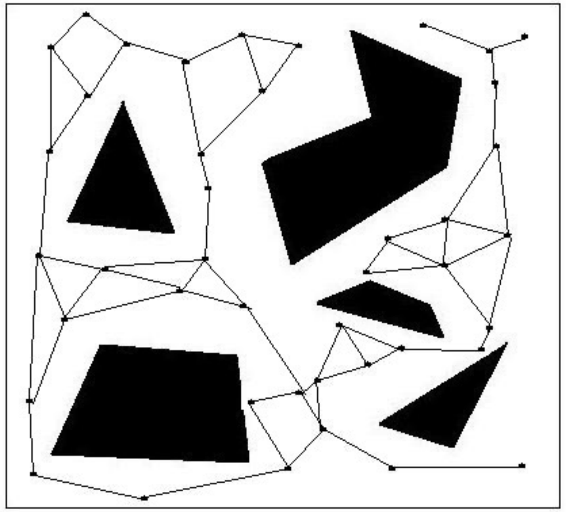
- Paths are points where distance to the two nearest obstacles are the same
- Paths favor moving the robot through the middle of large open spaces

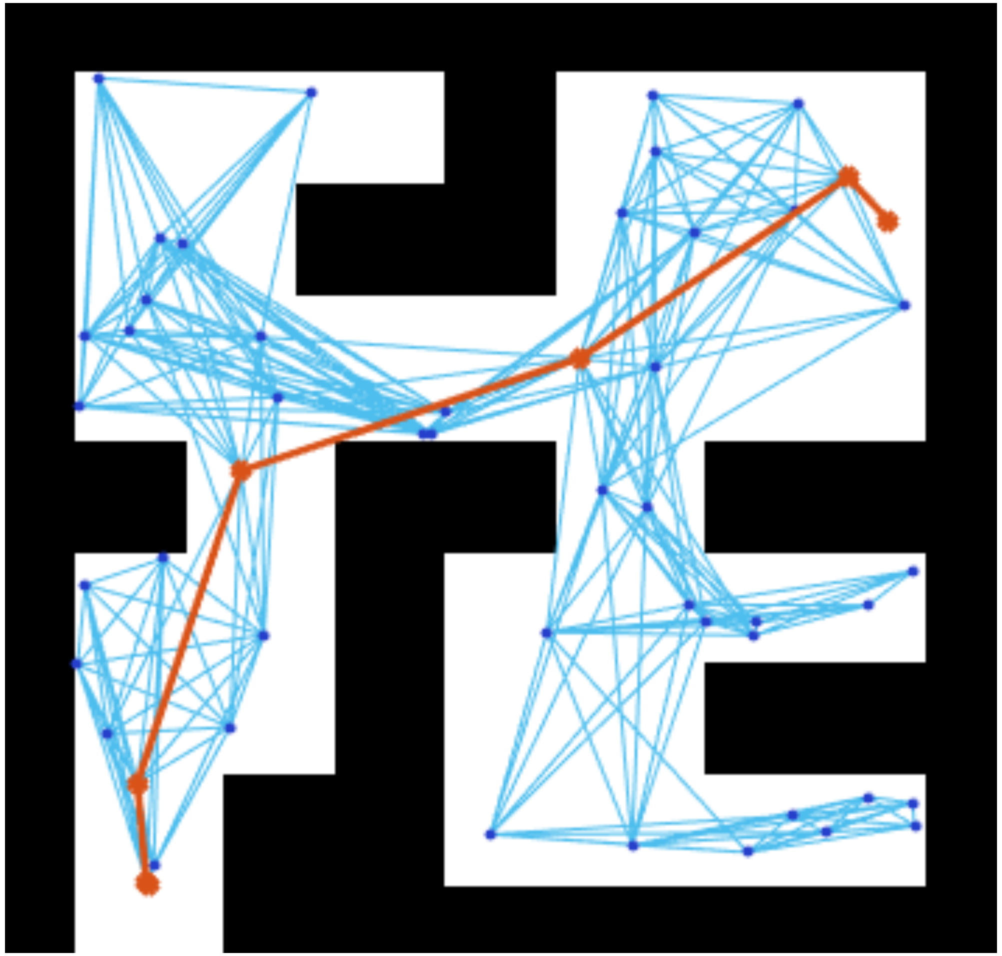


Voronoi Graph

Probabilistic Roadmap (PRM)

- Randomly sample free points in C-space to generate nodes
- Connect edges to visible nodes within a horizon





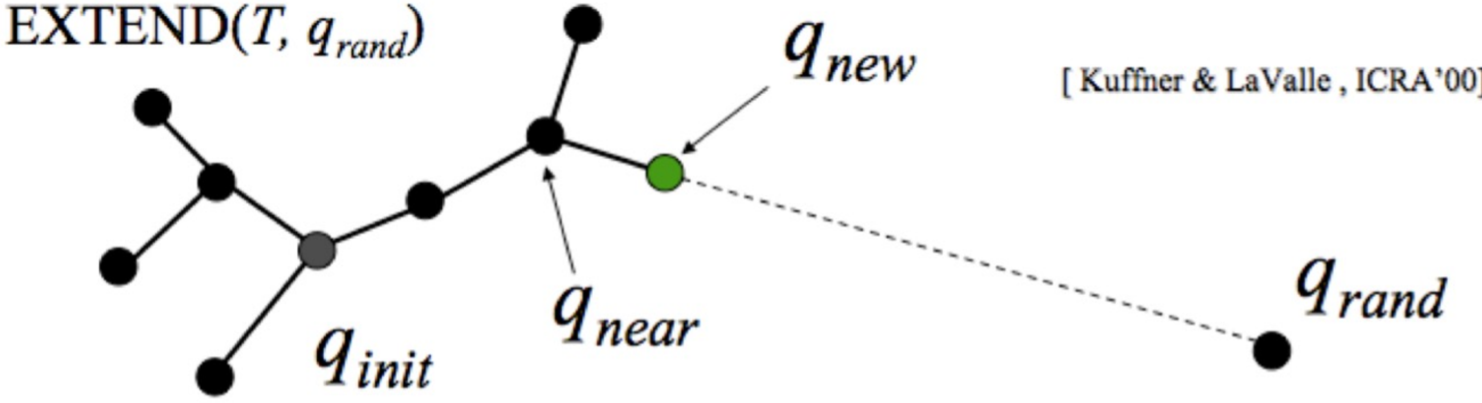
Rapidly Exploring Random Tree (RRT)

- Randomly explore space, biased towards the goal
- Tree will eventually fill space efficiently
- Can grow from start and goal and stop when trees meet
- Could also take into account motion equations as constraints

Building an RRT

```
BUILD_RRT (q_init) {
  T.init(q_init);
  for k = 1 to K do
    q_rand = RANDOM_CONFIG();
    EXTEND(T, q_rand)
}
```

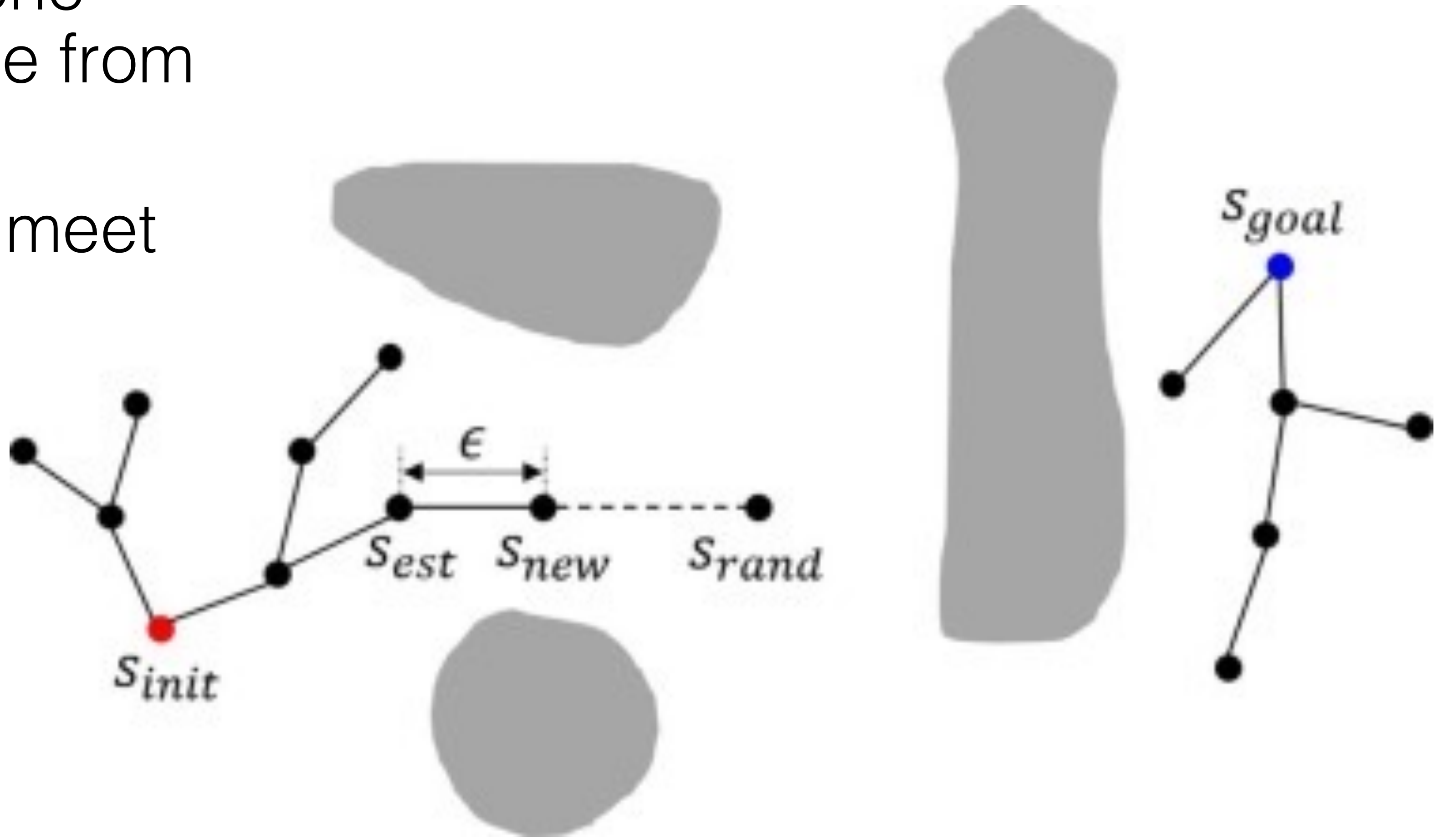
- Random sample connects to the nearest node so far
- If the nearest point lies on an edge, the edge is split in two



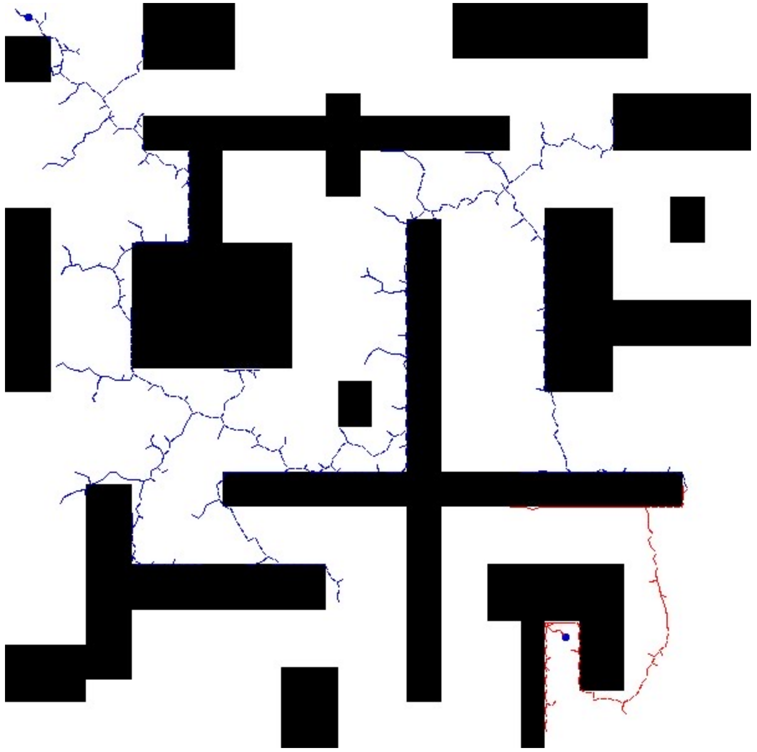
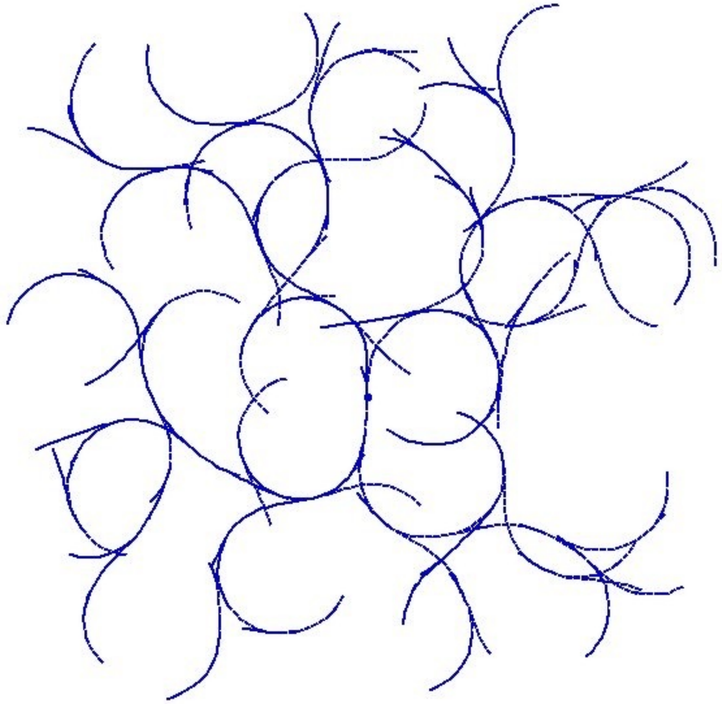
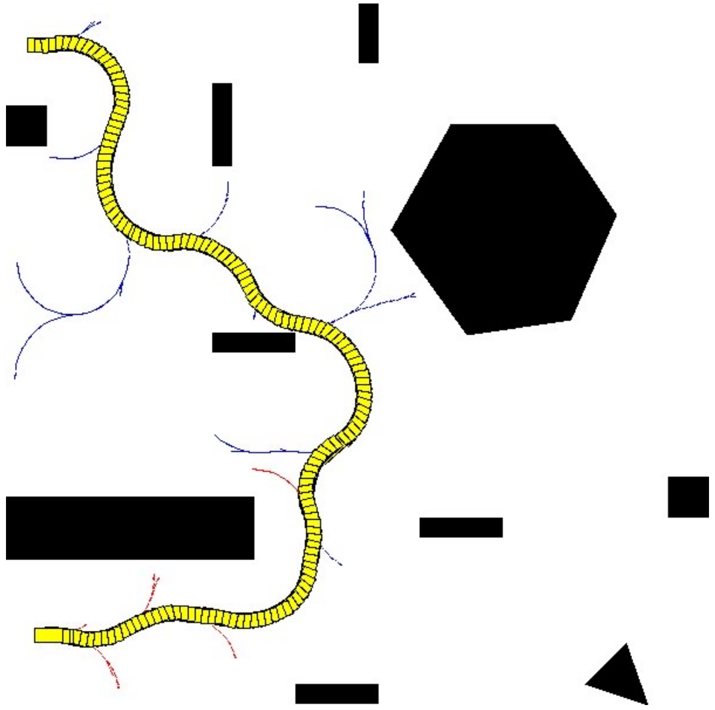
[Kuffner & LaValle , ICRA'00]

RRT for Planning

- Build two trees, one from start and one from goal
- Extend until they meet

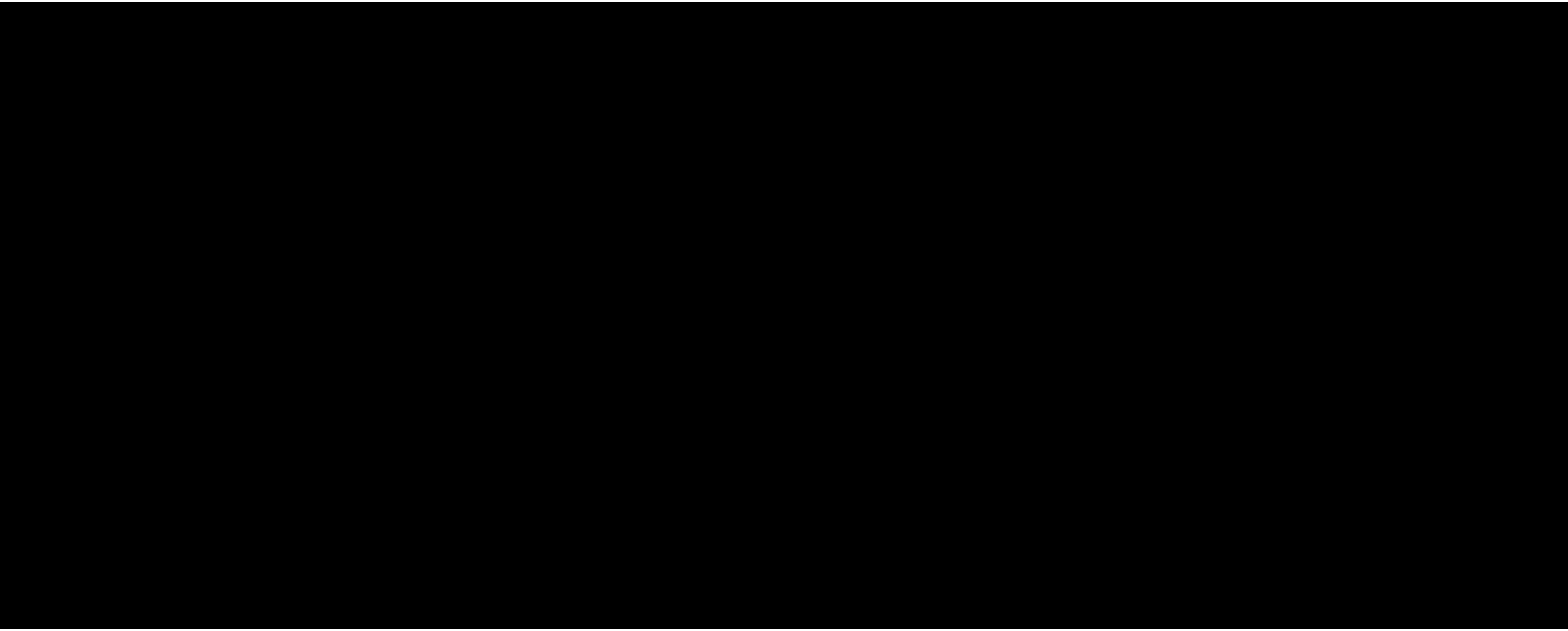


Other RRTs

http://msl.cs.uiuc.edu/rrt/gallery.html

Dynamic RRT Trajectory Planner



Amazon Astro

- Astro’s motion planner developed by UM alumni Jong Jin Park
- https://www.amazon.science/blog/astros-intelligent-motion-brings-state-of-the-art-navigation-to-the-home