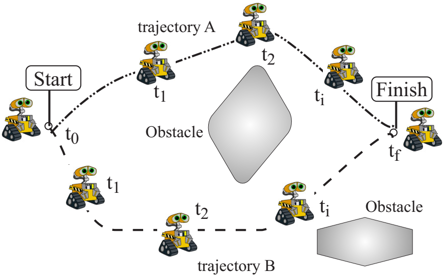
Mobile Robots

Path Planning

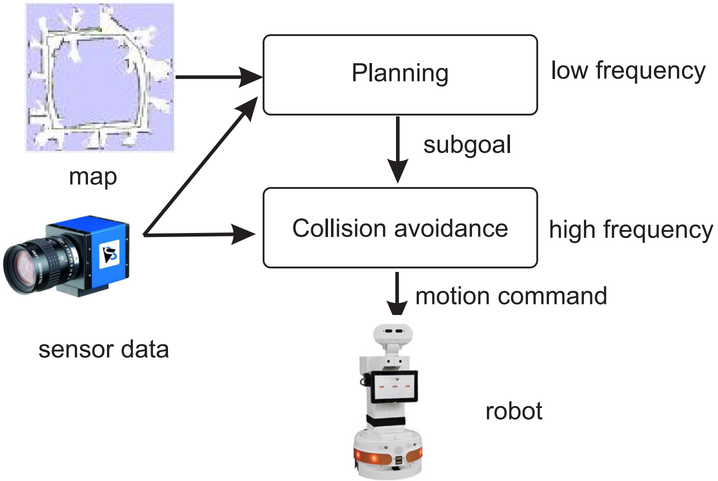
Lecture 11

Motion Planning for Mobile Robots



- Answer to questions: Where am I going? How do I get there?
- Most mobile robots are non-holonomic, however, most motion planning methods assume holonomic mobile bases.
- All robots with conventional wheels are non-holonomic, while robots with omnidirectional wheels may have a holonomic behaviour.
- Non-holonomic systems are characterized by constraint equations involving the time derivatives of the system configuration variables.

Two-layered structure



- In classical approaches to robot motion planning there is no uncertainty (they are deterministic).
- The following assumptions are implicit in the classical paradigm:
  - ✓ The knowledge of a global and accurate map of the environment is assumed.
  - ✓ The robot current pose and the goal pose are known.
  - ✓ The results of robot actions can be predicted.
- Such hypotheses are too strong in practice.
- The knowledge about the environment and situation is usually only partially known and is uncertain.

Motion Planning Problem for Non-Holonomic Mobile Robots

Given:

- a map of the environment with obstacles in the workspace,
- a robot subject to non-holonomic constraints,
- an initial configuration, and a goal configuration.

Find:

- an admissible collision-free path between the initial and the goal configuration.

How to solve:

- Solving this problem we must take into account both the configuration space constraints due to obstacles and the non-holonomic constraints.
- The tools developed to address this problem thus combine motion planning and control theory techniques.
- Such a combination is possible for the class of so-called *small-time controllable* systems.

Motion Planning Problem

Motion planning raises two problems:

- The **existence** of a collision-free admissible path (the decision problem): This is equivalent to determining whether the configurations lie in the same connected component of the collision-free configuration space.
- The **computation** of such a path (the complete problem).

In practice:

- The most common approach is to assume for path-planning purposes that the robot is in fact holonomic.
- This is especially common for differential-drive robots because they can rotate in place thus a holonomic path can be easily imitated if the orientation of the robot is not critical.
- Typically, obstacles are represented as polygons.
- Furthermore, it is often assumed that the robot is simply point (in this way the configuration space is reduced to 2D (x, y)).
- Because the robot is reduced to the point, we have to enlarge each obstacle by the size of the robot radius to compensate.

- The workspace $\mathcal{W} \subset \mathbb{R}^m$ consists of the **free space** and the **set of obstacles** $\mathcal{O} \in \mathcal{W}$.
- Typically, the path planning problem is formulated in the **configuration space**.
- A robot configuration $\boldsymbol{q}$ is a specification of the positions of all robot points relative to a fixed coordinate system.
- The configuration space (also called C-space) is the space of all possible con!gurations.
- The C-space $\mathcal{C}$ is described as a topological manifold and it consists of free space $\mathcal{C}_{free}$ and obstacle regions $\mathcal{C}_{obs}$.

$$\mathcal{C}_{free} = \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O} = \emptyset\},$$

$$\mathcal{C}_{obs} = \mathcal{C}/\mathcal{C}_{free},$$

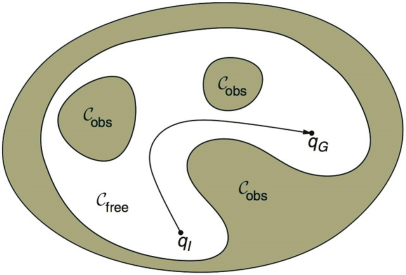
where $\mathcal{A}(\boldsymbol{q})$ is the space occupied by the robot in the configuration $\boldsymbol{q} \in \mathcal{C}$.

Path planning in the configuration space:

- Find a continuous curve

$$\boldsymbol{p}(\cdot) : [0,\ 1] \rightarrow \mathcal{C}_{free},$$

  where $\boldsymbol{p}(0) = \boldsymbol{q}_I,\ \boldsymbol{p}(1) = \boldsymbol{q}_G$.
- Planning in the $\mathcal{C}_{free}$, with the robot being a point in C-space.



Path Planning

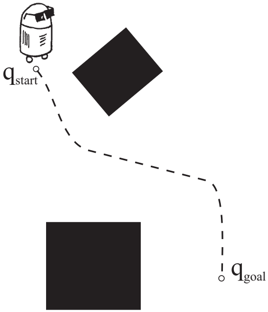
Due to motion capabilities, there are two basic types of mobile robots:

1. *Non-holonomic* robots – are subject to velocity (differential kinematic) constraints
2. *Holonomic* robots – no velocity constraints

Holonomic robot

Non-holonomic robot

- Path-planning systems use the model of the environment to compute the path.
- The model can range from continuous geometric description, through a decomposition-based geometric map to a topological map.
- Path planners transform environmental model into a discrete map suitable for the chosen path planning algorithm.



There are three general strategies for discrete decomposition:

1. Cell decomposition: discriminate between free and occupied cells.
   - exact cell decomposition
   - approximate cell decomposition
2. Road-maps: identify a set of routes within the free space.
   - deterministic
   - probabilistic
3. Potential field: create the artificial field, or gradient, across the robot's map that attracts the robot to the goal position while repelling it from obstacles.

Cellular Decomposition I

The basic, deterministic algorithm:

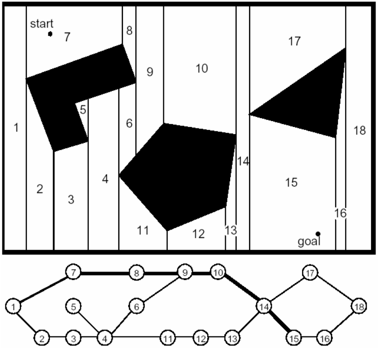
1. Divide workspace into simple, connected regions called cells.
2. Determine which open cells are adjacent and construct *a connectivity graph*.
3. Find cells containing the initial and the goal configurations and search for a path in the connectivity graph to join the initial and the goal cell.
4. From the sequence of cells found with an appropriate searching algorithm, compute a path within each cell, e.g.:
    1. passing through the midpoints of the boundaries,
    2. by a sequence of wall following motions and movements along straight lines.

Cell decomposition methods can be divided in two groups based on the placement of the boundaries between cells:

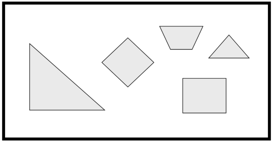
- *Exact cell decomposition* – the boundaries are placed as function of the structure of the environment (obstacles and free space), such that the decomposition is lossless.
- *Approximate cell decomposition* – grid-based approximation of the map (fixed-size grid or variable-size grid).

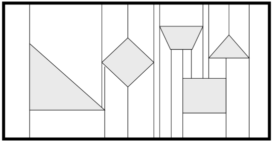
Exact Cell Decomposition

- The structures represent the free space by the union of cells.
- Two cells are *adjacent* if they share a common boundary.
- An *adjacency graph* encodes adjacency relationships of the cells, where a node corresponds to the cell and an edge connects nodes of adjacent cells.
- Computational complexity directly depends on density and complexity of elements in the environment.



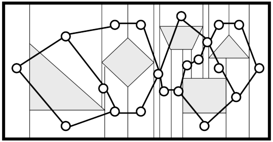
Exact Cell Decomposition: Trapezoidal Decomposition

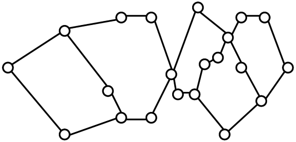




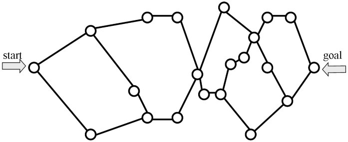
Simply draw a vertical line from each vertex until you hit an obstacle.

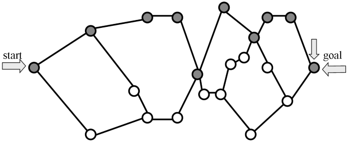
This reduces the world to a union of trapezoid-shaped cells.





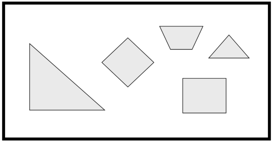
By reducing the world to cells, the world is abstracted to a graph.

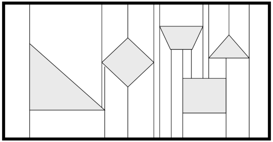




With an adjacency graph, a path from start to goal can be found by simple traversal.

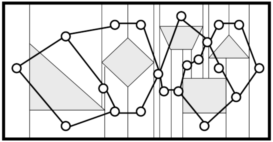
Exact Cell Decomposition: Trapezoidal Decomposition

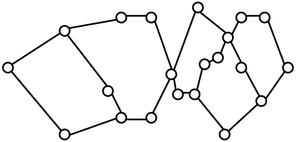




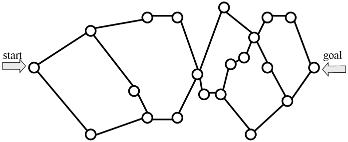
Simply draw a vertical line from each vertex until you hit an obstacle.

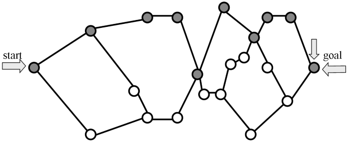
This reduces the world to a union of trapezoid-shaped cells.





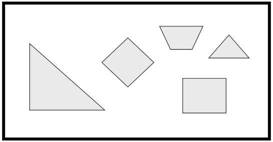
By reducing the world to cells, the world is abstracted to a graph.

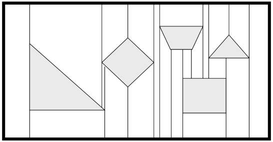




With an adjacency graph, a path from start to goal can be found by simple traversal.

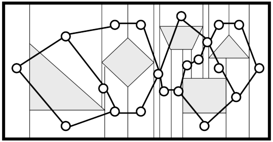
Exact Cell Decomposition: Trapezoidal Decomposition

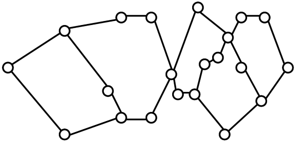




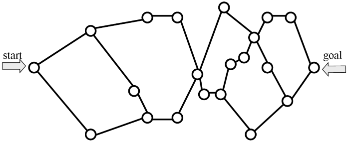
Simply draw a vertical line from each vertex until you hit an obstacle.

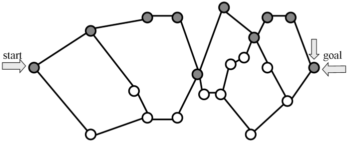
This reduces the world to a union of trapezoid-shaped cells.





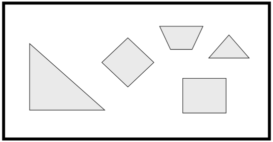
By reducing the world to cells, the world is abstracted to a graph.

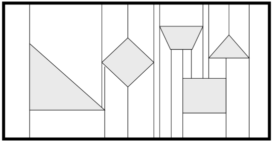




With an adjacency graph, a path from start to goal can be found by simple traversal.

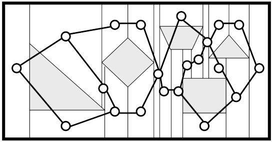
Exact Cell Decomposition: Trapezoidal Decomposition

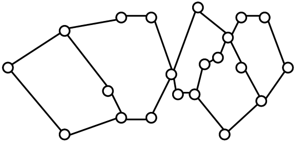




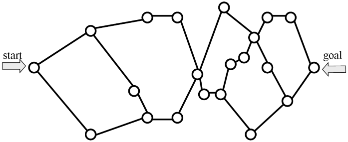
Simply draw a vertical line from each vertex until you hit an obstacle.

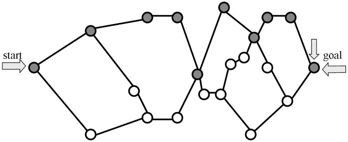
This reduces the world to a union of trapezoid-shaped cells.





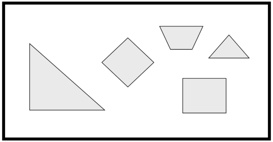
By reducing the world to cells, the world is abstracted to a graph.

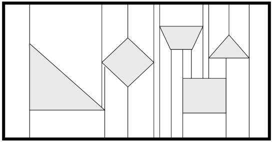




With an adjacency graph, a path from start to goal can be found by simple traversal.

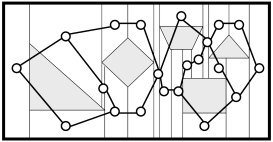
Exact Cell Decomposition: Trapezoidal Decomposition

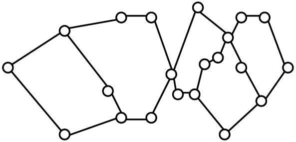




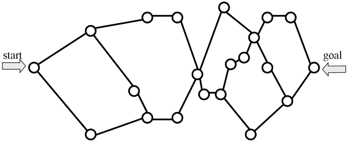
Simply draw a vertical line from each vertex until you hit an obstacle.

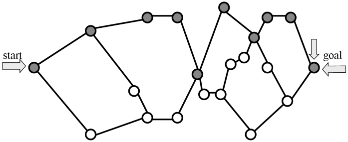
This reduces the world to a union of trapezoid-shaped cells.





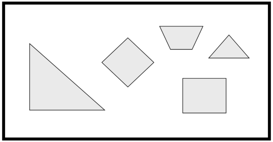
By reducing the world to cells, the world is abstracted to a graph.

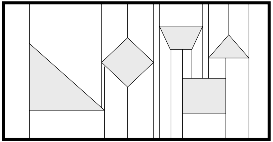




With an adjacency graph, a path from start to goal can be found by simple traversal.

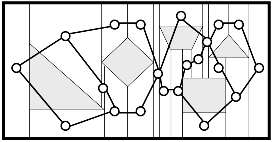
Exact Cell Decomposition: Trapezoidal Decomposition

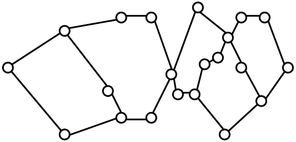




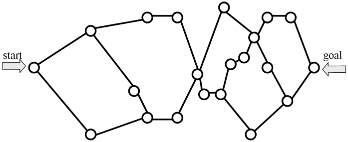
Simply draw a vertical line from each vertex until you hit an obstacle.

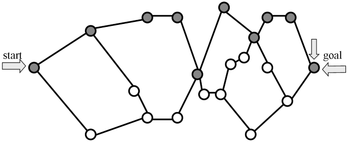
This reduces the world to a union of trapezoid-shaped cells.





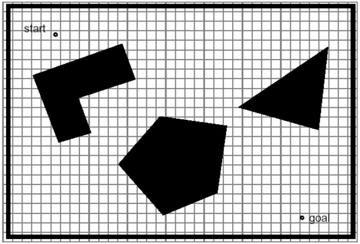
By reducing the world to cells, the world is abstracted to a graph.

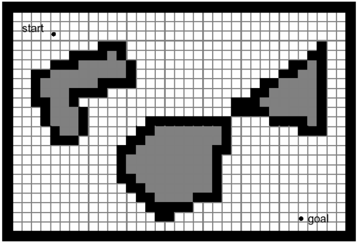




With an adjacency graph, a path from start to goal can be found by simple traversal.

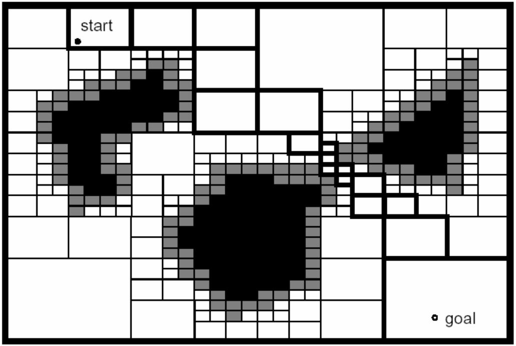
- One of the most popular techniques due to popularity of grid-based maps.
- Low computational complexity.
- Potentially large memory requirements.
- Cell size is not dependant on the particular object.
- Narrow passageways can be lost due to inexact tessellation.



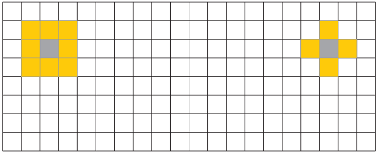


Approximate Cell Decomposition: Variable-Size

- Free space is externally bounded by a rectangle and internally bounded by three polygons.
- The rectangle is recursively decomposed into four identical smaller rectangles.
- At each level of resolution only free cells are used to construct connectivity graph.
- Efficient representation of space.



- A common algorithm used to determine the shortest paths between two points. In essence, a breadth first search of a graph.
- Typically applied to closed and stationary environments.
- Well-suited for grid representations. The world is represented as a two-dimensional grid.
- Distance is reduced to discrete steps. For simplicity, we'll assume distance is uniform.
- Direction is limited from one adjacent cell to another.



8-Point Connectivity or 4-Point Connectivity

The Wavefront Planner

Two-phase algorithm:

Phase 1: Propagate wave from goal to start.

1. Start with binary grid; 0's represent free space, 1's represent obstacles
2. Label goal cell with "2"
3. Label all 0-valued grid cells adjacent to the "2" cell with "3"
4. ...
5. Label with "n" all unlabeled cells neighboring (n - 1)-labeled cells
6. Continue until wave front reaches the start cell

Phase 2: Extract path using gradient descent.

1. Given label of start cell as "x", find neighboring grid cell labeled "x-1"; mark this cell as a waypoint
2. Then, find neighboring grid cell labeled "x-2"; mark this cell as a waypoint
3. Continue, until reach cell with value "2" (this is the goal cell)

The Wavefront Planner – Phase 1

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

Initial labels

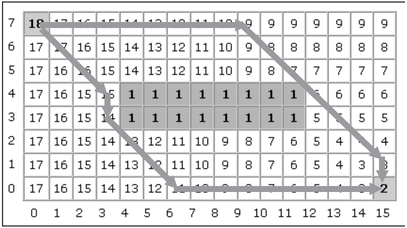
7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	4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

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

Cell labelling – wavefront propagation

To save processing time, can use dual wavefront propagation, where you propagate from both start and goal locations.

- To find the shortest path, according to a given metric, simply always move toward a cell with a lower number.
- The path is defined by any uninterrupted sequence of decreasing numbers that lead to the goal.



Sample paths

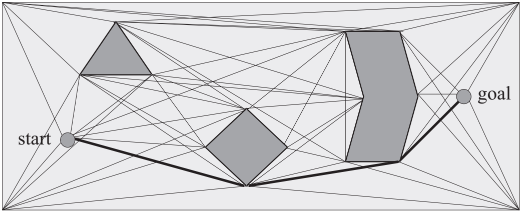
Road-Maps

- Describe the robot's free space Q_{free} as a network of lines and/or curves called road-maps.
- A road-map is a union of curves such that for all start and goal points in Q_{free} that can be connected by a path:
  - *Accessibility*: there exists a collision-free path from the start to the road-map.
  - *Departability*: there exists a collision-free path from the road-map to the goal.
  - *Connectivity*: there exists a collision-free path from the start to the goal (on the roadmap).
  - *One dimensional*.

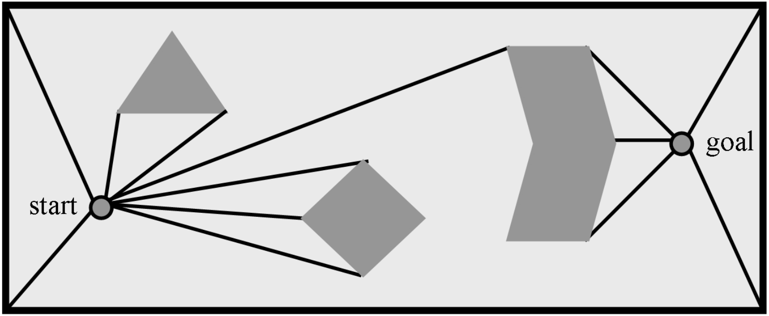
1. Build the road-map:
   a) nodes are points in Q_{free} (or its boundary),
   b) two nodes are connected by an edge if there is a free path between them.
2. Connect start and goal points to the road map at point q' and q'', respectively
3. Find a path on the road-map between q' and q''. The result is a path in Q_{free} from start to goal.

Road-Map Path Planning: Visibility Graph

- Defined for polygonal obstacles.
- A visibility graph consists of the set of edges obtained by joining all pairs of vertices that can see each other (including the start and goal vertices).
- Path planning can be achieved by determining the start and end points and applying standard algorithms from graph theory.
- Visibility graphs are easy to implement (when obstacles are polygons) and generate optimal (shortest possible length) paths.
- However, these paths skirt the edges of obstacles, possibly endangering the robot. Possible solution is to enlarge obstacles by more than the robot's radius.

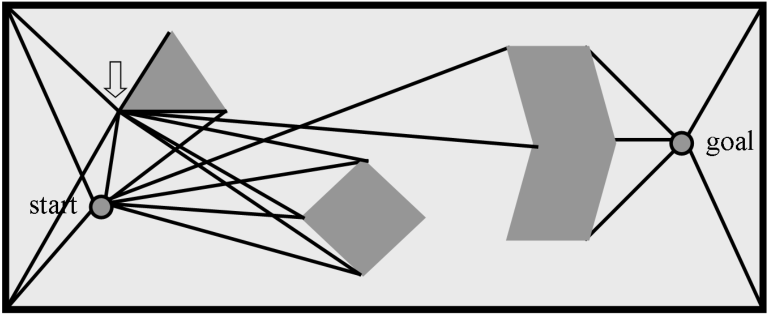


- First, draw lines of sight from the start and goal to all "visible" vertices and corners of the world.

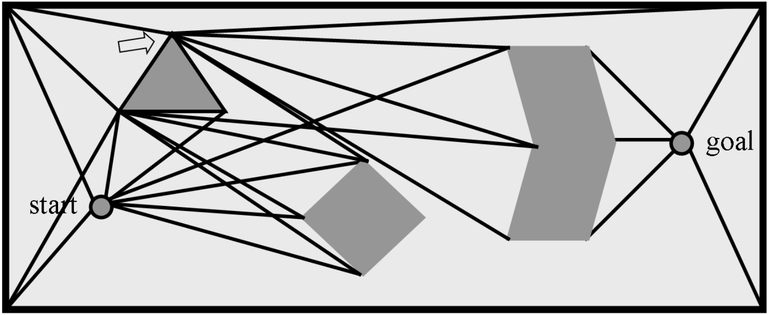


$$e_{ij} \neq \emptyset \;\Leftrightarrow\; sv_i + (1-s)v_j \in \mathrm{cl}Q_{free} \quad \forall s \in (0,1)$$

- Second, draw lines of sight from every vertex of every obstacle like before. Remember, lines along edges are also lines of sight.

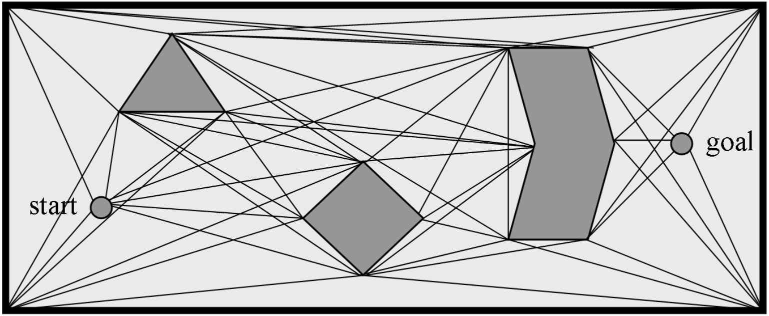


- Second, draw lines of sight from every vertex of every obstacle like before. Remember, lines along edges are also lines of sight.



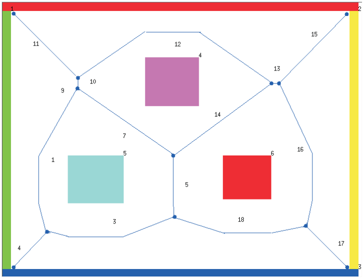
Visibility Graph Construction

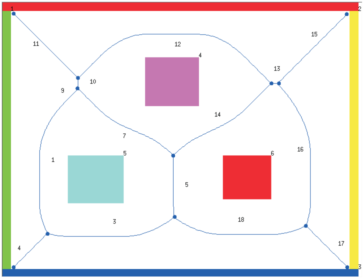
- Repeat until graph is done.



Road-Map Path Planning: Voronoi Diagram

- The Voronoi diagram consists of all points in free space which are equidistant to the two closest obstacles.
- In contrast to the visibility graph approach tends to maximize the distance between robot and obstacles.
- Using range sensors like laser or sonar, a robot can navigate along the Voronoi diagram using simple control rules.
- When obstacles are polygons, the Voronoi map consists of straight and parabolic segments.





Probabilistic Road-Maps (PRM) I

- When exact computation is too hard, it may still be possible to achieve good results by approximate, sampling-based algorithms.
- Search for collision free path not by explicitly constructing the collision free space, but by sampling the configuration space.
- PRM does not represent the entire free configuration space, but rather a road-map through it.
- The key idea in sampling-based planning is to exploit advances in collision detection algorithms that compute whether a single configuration is collision free.
- This road-map has the form of an undirected, acyclic graph $G = (V, E)$, with vertices as configurations and edges as paths.
- PRM does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists).
- It offers probabilistic completeness: when a solution exists, a probabilistically complete planner finds a solution with probability 1 as time goes to infinity.

Probabilistic Road-Maps (PRM) II

- When a solution does not exists, a probabilistically complete planner may not be able to determine that a solution does not exist.

Two-phase method:

- Preprocessing phase: Construct a probabilistic road map by adding randomly distributed points in the free space and connecting them via a fast local planner.
- Query phase: Connect the start and goal configuration to the road-map and find a path in the road-map.

The preprocessing phase of PRM:

1. *Initialization*: Let $G = (V, E)$ is initially empty. Vertices of G will correspond to collision-free configurations, and edges to collision-free paths that connect vertices.
2. *Configuration sampling*: A configuration $\boldsymbol{c}$ is sampled from C_{free} and added to the vertex set V.
3. *Neighborhood computation*: Usually, a metric is defined in the C-space, $d : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$. Vertices $\boldsymbol{q}$ already in V are then selected as part of c's neighborhood if they have small distance according to d.

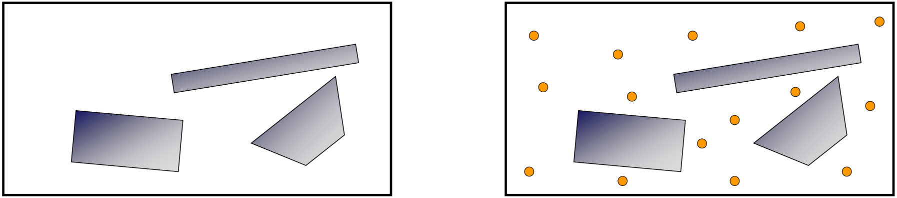
Probabilistic Road-Maps (PRM) III

4. *Edge consideration*: For those vertices $\boldsymbol{q}$ that do not belong in the same connected component of G with $\boldsymbol{c}$ the algorithm attempts to connect them with an edge.
5. *Local planner*: Given $\boldsymbol{c}$ and $\boldsymbol{q} \in \mathcal{C}_{free}$ a local planner used that attempts to construct a path $\tau_S : [0,1] \to \mathcal{C}_{free}$ such that $\tau(0) = \boldsymbol{c}$ and $\tau(1) = \boldsymbol{q}$. Using collision detection, τ_S must be checked to ensure that it does not cause a collision.
6. *Edge insertion*: Insert τ_S into E, as an edge from $\boldsymbol{c}$ to $\boldsymbol{q}$.
7. *Termination*: The algorithm is typically terminated when a predefined number of collision-free vertices N has been added in the road-map.

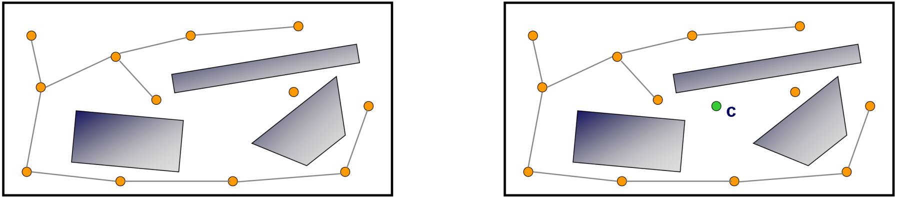
To solve a query, $\boldsymbol{q}_I$ and $\boldsymbol{q}_G$ are connected to the road-map, and graph search is performed.

Probabilistic Road-Maps (PRM) IV

Configurations are sampled by picking coordinates at random.

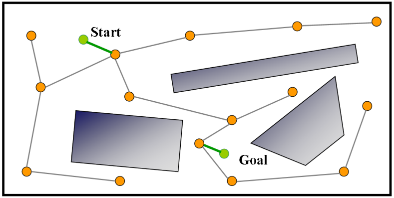


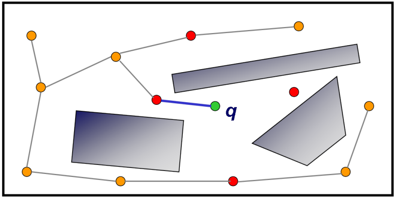
Local planner is used to determine whether a local path exists. A random free configuration c is generated and added to N.



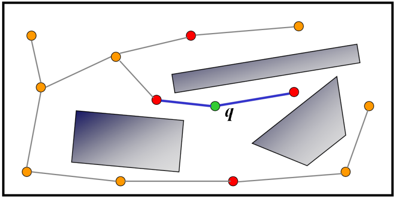
Probabilistic Road-Maps (PRM) V

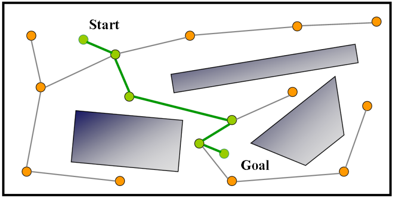
Candidate neighbors to c are partitioned from V.





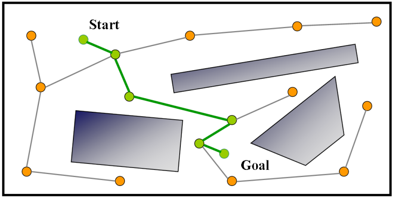
Edges are created between these neighbors and c, such that acyclicity is preserved.

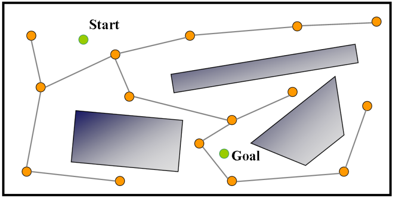




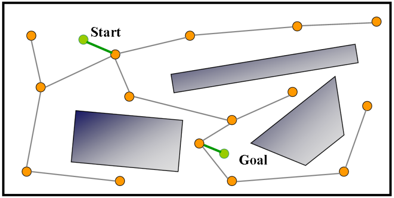
Probabilistic Road-Maps (PRM) VI

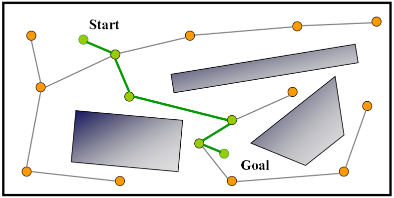
The start and goal configurations are added to the graph.





Given the start and goal configurations and the edges calculate the path connecting them.





Rapidly-Exploring Random Trees (RRT)

- Idea: aggressively probe and explore the C-space by expanding incrementally from an initial configuration q_0.
- The explored region is marked by a tree rooted at q_0.

after 45 iterations

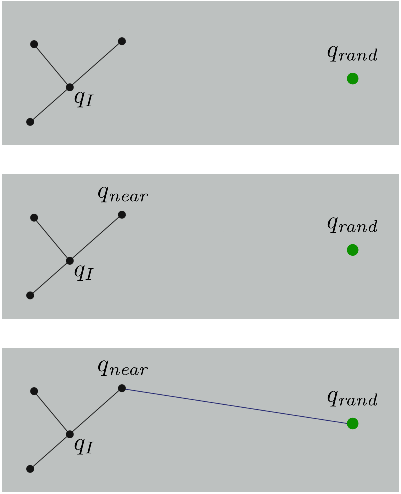
after 2345 iterations

- Basic RRT algorithm

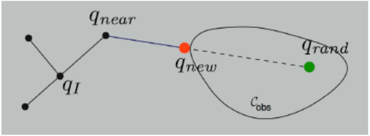
Algorithm 1: RRT

In: $\mathcal{C}$, q_I, q_G
Out: Tree $G = (V, E)$

G.add_vertex(q_I)
repeat
    $q_{rand} \rightarrow$ RANDOM_CONFIG($\mathcal{C}$)
    if *CLEAR*(q_{rand}) **then**
        $q_{near} \leftarrow$ NEAREST(G, q_{rand})
        if *LINK*(q_{rand}, q_{near}) **then**
            G.add_vertex(q_{rand})
            G.add_edge(q_{near}, q_{rand})
        end
    end
until *until* q_G *reached*



- Unlike PRMs that connect newly sampled configurations to a set of nearest neighbors, RRT selects a **single neighbor**.
- Requires the choice of a proper distance metric.
- Instead of discarding q_{rand} when the local planner reports a collision, we can also add the configuration along the local path which is closest to $\mathcal{C}_{obs}$.
- The tree can only be extended by a configuration q_{new} close to q_{rand} and the corresponding edge from q_{near} to q_{new}.
- The basic algorithm usually terminates by checking if q_{rand} (or q_{new}) is near the goal ("has reached the goal region").



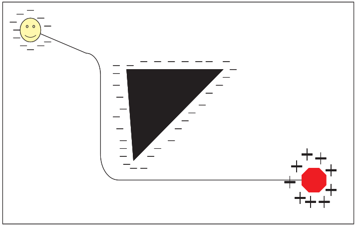
Sampling-Based Planning

- Sampling-based planners are usually more efficient but have weaker guarantees.
- They are probabilistically complete: the probability tends to one that a solution is found if one exists (when computation time goes to ∞). Otherwise they may still run forever.
- It is easy to construct examples that cause sampling-based algorithm to fail or converge slowly. In some cases (e.g. narrow passages), problem-specific heuristics can be developed.
- Problems with high-dimensional and complex C-spaces are still also hard for sampling-based methods.
- However, they have solved previously unsolved problems and have become the preferred choice for many practical problems.

Artificial Potential Fields I

- *Potential Field* method is inspired from obstacle avoidance techniques.
- It does not explicitly construct a road-map, but instead constructs a differentiable real-valued function $U : \mathbb{R}_m \rightarrow R$, called a potential function, that guides the motion of the moving object.
- The potential consists of an attractive component $U_a(\boldsymbol{q})$ ($\boldsymbol{q} = [x, y]^\mathsf{T}$), which pulls the robot towards the goal, and a repulsive component $U_r(\boldsymbol{q})$, which pushes the robot away from the obstacles.

$$U(\boldsymbol{q}) = U_a(\boldsymbol{q}) + U_r(\boldsymbol{q})$$



Artificial Potential Fields II

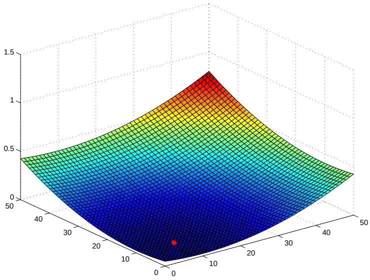
- It generates a resultant force

$$F = -\nabla U(\boldsymbol{q}) = \begin{pmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{pmatrix},$$

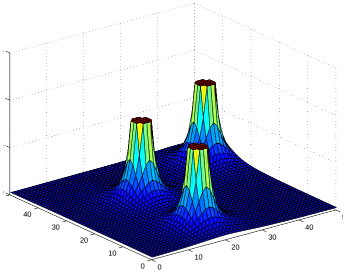
  where $\nabla U(\boldsymbol{q})$ is the gradient of the potential function.
- Energy is minimized by following the negative gradient of the potential energy function U.
- $U_a(\boldsymbol{q})$ can be modeled as a parabolic attractor

$$U_a(\boldsymbol{q}) = \frac{1}{2} k_a \, d(\boldsymbol{q}, \boldsymbol{q}_G)^2,$$

  where $k_a > 0$ is a scaling factor, and $d = ||\boldsymbol{q} - \boldsymbol{q}_G||$ is the Euclidean distance between the current configuration $\boldsymbol{q}$ and the goal configuration $\boldsymbol{q}_G$.



Goal attractor potential field



Repulsive potential field from obstacles

- The potential function is a differentiable function, thus the attractive force can be computed as

$$F_a(\boldsymbol{q}) = -\nabla U_a(\boldsymbol{q}) = -k_a(\boldsymbol{q} - \boldsymbol{q}_G),$$

that converges linearly toward 0 as the robot reaches the goal.

Artificial Potential Fields IV

- The repulsive potential can be described as a barrier function

$$U_r(\boldsymbol{q}) = \begin{cases} \frac{1}{2}k_r\,(\frac{1}{d(\boldsymbol{q},\boldsymbol{q}_p)} - \frac{1}{d_0})^2 & \text{if} \quad d(\boldsymbol{q},\boldsymbol{q}_p) \leqslant d_0 \\ 0 & \text{if} \quad d(\boldsymbol{q},\boldsymbol{q}_p) > d_0 \end{cases},$$

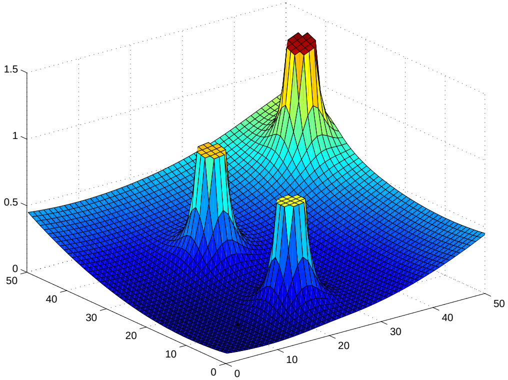
  where k_r is a scaling factor, $d(\boldsymbol{q},\boldsymbol{q}_p)$ is the minimal distance from $\boldsymbol{q}$ to the obstacle, and d_0 the distance of influence of the obstacle.

- If the obstacle boundary is convex and piecewise differentiable, this lead to the repulsive force

$$\begin{aligned} F_r(\boldsymbol{q}) &= -\nabla U_r(\boldsymbol{q}) \\ &= \begin{cases} k_r(\frac{1}{d(\boldsymbol{q},\boldsymbol{q}_p)} - \frac{1}{d_0})\frac{1}{d(\boldsymbol{q},\boldsymbol{q}_p)^2}\frac{\boldsymbol{q}-\boldsymbol{q}_p}{d(\boldsymbol{q},\boldsymbol{q}_p)} & \text{if} \quad d(\boldsymbol{q},\boldsymbol{q}_p) \leq d_0 \\ 0 & \text{if} \quad d(\boldsymbol{q},\boldsymbol{q}_p) > d_0 \end{cases} \end{aligned}$$

- The resulting force $F(\boldsymbol{q}) = F_a(\boldsymbol{q}) + F_r(\boldsymbol{q})$ acting on a point robot moves the robot away from the obstacles and toward the goal.
- Under ideal conditions, by setting robot's velocity vector proportional to the field force vector, the robot can be smoothly guided toward the goal.

Artificial Potential Fields V



Resultant potential field

Artificial Potential Fields VI

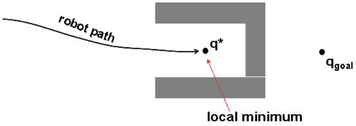
On-line path planning with potential field:

Input: Function $\nabla U(\boldsymbol{q})$
Output: Sequence $\{\boldsymbol{q}(0), \boldsymbol{q}(1), \ldots, \boldsymbol{q}(i)\}$

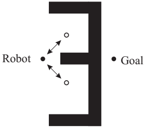
1. $\boldsymbol{q}(0) = \boldsymbol{q}_{start}$
2. $i = 0$
3. **while** $\nabla U(\boldsymbol{q})(i) \neq 0$
4. $\quad \boldsymbol{q}(i+1) = \boldsymbol{q}(i) + \alpha(i) \nabla U(\boldsymbol{q}(i))$
5. $\quad i = i + 1$
6. **end while**

Limitations of the potential field methods:

- Local min/max/saddle point can appear dependent on the obstacle shape and size (robot's motion would terminate at a critical point, $\boldsymbol{q}^*$, where $\nabla U(\boldsymbol{q}^*) = 0$).
- It possible to check this point by looking at the Hessian $\partial^2 U / \partial \boldsymbol{q}^2$ (i.e. positive-definite Hessian: local minimum).
- If the objects are concave it might lead to a situation for which several minimal distances $d(\boldsymbol{q})$ exist, resulting in oscillation between the two closest points to the object.



Local minimum trap



Oscillation between two points