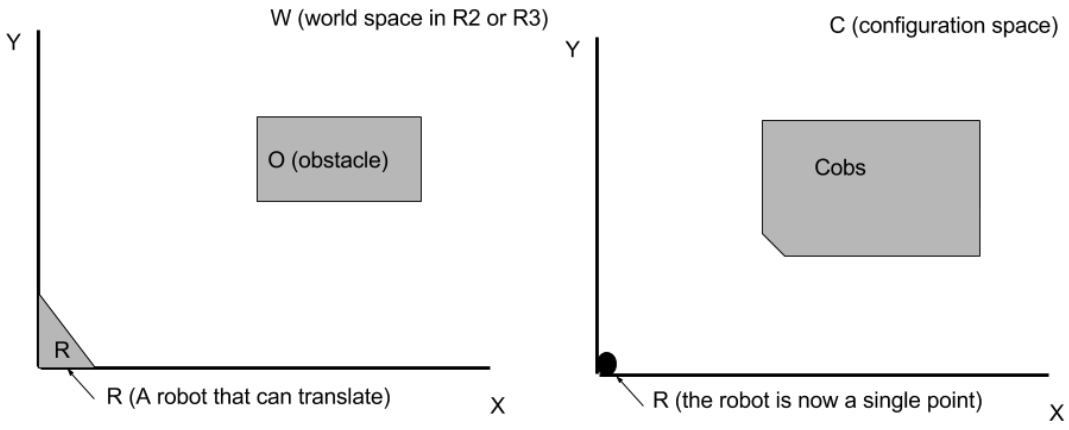
**CS 294-115 Algorithmic Human-Robot Interaction** **Fall 2016**

# Lecture 3: Motion Planning (cont.)

*Scribes: Molly Nicholas, Chelsea Zhang*

## 3.1 Previously in class...

Recall that we defined configuration spaces. Take the below example, where we have the robot in the World space ($\mathbb{R}^2$ or $\mathbb{R}^3$) on the left, and the Configuration space on the right. Note that C-space is not always the same dimensionality as the World space, but it does allow us to represent the robot as a single point.




$C_{obs}$ is a **set**, defined as all configurations (q) in Configuration space (C) such that R(q) intersected with the Obstacles is not the empty set:

$$C_{obs} = \{q \in C | R(q) \cap O \neq \emptyset\} \tag{3.1}$$

Recall that R(q) takes us from Configuration space (C-space) to World space. R(q) gives you the set of points in the real world.

## 3.2 Motion Planning Problem Statement

**Motion Planning Problem Statement:** How do you get from A (a particular configuration) to another configuration (B) without intercepting $C_{obs}$. Also known as the "Piano Mover's Problem". (Side note: This was named because moving a piano is actually quite difficult, taking into account obstacles and so on). We will examine three types of algorithms for solving the Motion Planning Problem:

- Analytic / Geometric

- Grid search
- Sampling-based

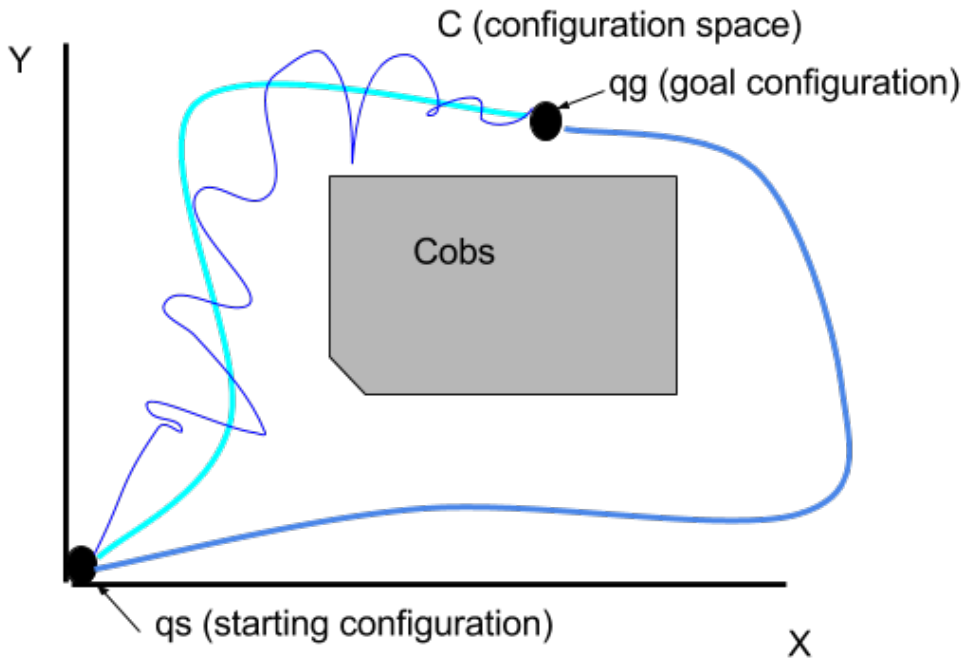
The problem is laid out as follows. Assume we are given:

- World space $\mathbb{R}^2$ or $\mathbb{R}^3$
- Obstacle regions $O \in W$
- Robot R $\rightarrow$ Cspace C (assume's R's kinematics are known)
- $q_s$: A starting configuration
- $q_g$: A goal configuration

Our goal is to compute a path

$$\tau : [0,1] \rightarrow C_{free} \tag{3.2}$$
$$\tau(0) = q_s \tag{3.3}$$
$$\tau(1) = q_g \tag{3.4}$$

$\tau$ is a function that maps from the interval [0, 1] to a configuration. It's just a path, not a timed path (that's called a trajectory). The focus is on the geometry. A path is unitless. The path is not necessarily optimal by any metrics, it's just a correct solution. All the below paths would be considered a correct solution, even though one of them is pretty wacky.



In 1985, Reif and Sharir showed that motion planning was NP-hard. Canny showed it to be NP-complete in 1988.

Note: NP stands for "Nondeterministic polynomial". NP hard means that, if I can randomly generate a solution, I can quickly (within polynomial time) verify that it's correct, but I cannot generate a solution in polynomial time. A decision problem is "NP complete" if it is in NP, and every other problem in NP is reducible to it in polynomial time.

## 3.3 Algorithms

We will cover two motion planning algorithms today. Both try to find not just any path $\tau$, but a $\tau$ that minimizes Euclidean distance (more or less). It's worth noting that Euclidean distance in configuration space may not be the best metric to optimize – perhaps moving the robot wrist is preferable to moving the shoulder, since the shoulder carries all the arm weight.

### 3.3.1 Visibility graphs

The simplest motion planning algorithm we cover uses a visibility graph (Nilsson, 1969). This algorithm assumes polygonal obstacles in C-space. Intuitively, it connects all vertices that can "see" each other, i.e. all vertices for which the straight line connecting them lies in $C_{free}$. Then it runs a graph search algorithm to find the shortest path. See Figure 3.1 for an example.

A few more details: the set of vertices is $V = \{q_s, q_g\} \cup \{v : v \text{ is a vertex of } C_{obs}\}$. The edges are $E = \{(u, v) : (1-t)u + tv \in C_{free}\ \forall u, v \in V, t \in [0, 1]\}$. To allow paths along the edges of obstacles, $C_{free}$ would need to include those edges (it would need to be closed). Checking whether edges are collision-free can be done efficiently since all the obstacles are polygons.

#### 3.3.1.1 Pros of this approach

- The algorithm will find the optimal (shortest) path.
- Graph search for the optimal path is probably fast.
- The algorithm satisfies *completeness*: if $\tau$ exists, it returns $\tau$ in finite time. Otherwise, it returns failure in finite time.

#### 3.3.1.2 Cons

- It solves a subset of the motion planning problem; the assumption of polygonal obstacles is quite restrictive. Note that robot arms, and rotation-capable robots in general, break the assumption of polygonal obstacles. For arms, a point obstacle in the world becomes a curve in C-space.
- It assumes an *explicit* representation of $C_{obs}$. But it can be very difficult to map obstacles in the world to obstacles in C-space.

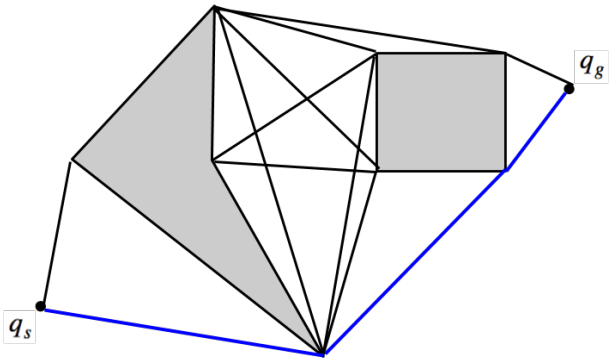



Figure 3.1: The edges and shortest path in a sample visibility graph.

### 3.3.2 Grid search

Grid search overcomes both of these limitations, at the cost of sacrificing optimality and completeness. This approach discretizes C-space into a grid, and performs graph search on the grid vertices, discarding them if they are in collision with obstacles. Figure 3.2 shows the first step in a sample grid search. The points above and below the start configuration can be added to the fringe, but the point to the right will be disallowed by the collision checker.

In particular, this approach selects $q_s$, $q_g$ and the points along $\tau$ from the grid cells (without loss of generality, take the center of the cell). It assumes a collision checker $\gamma$ is available, where

$$\gamma(q) = \begin{cases} 0 & \text{if } q \in C_{free} \\ 1 & \text{otherwise} \end{cases}$$

Starting from $q_s$, run a graph search algorithm like BFS or $A^*$ (Hart and Nielsson, 1968). Note $A^*$ requires an admissible heuristic for the cost-to-go, which means the heuristic should underestimate actual distance to the goal. Some candidates here are Euclidean and Manhattan distance to $q_g$.

When the graph search expands the fringe (finds acceptable neighbors of the current point $q_{curr}$), it only adds neighbors that pass the collision checker. This could be a simple check for $\gamma(q_{neighbor}) = 0$. If we are concerned that the line between $q_{curr}$ and $q_{neighbor}$ is in collision, even if both points are in $C_{free}$, we could assume the obstacle has a minimum size $w$ and check all points along the line between $q_{curr}$ and $q_{neighbor}$, spaced $\lambda w$ apart, where $\lambda < 1$.

If we want, we could allow diagonal edges by including those as neighbors in the graph search, probably with a higher edge cost.

If the problem is not solvable, the algorithm can be rerun at a higher grid resolution.

#### 3.3.2.1 Pros

- It is *resolution-optimal*: an optimal path will be found for a given resolution.
- It is *resolution-complete*: if $\tau$ exists for a given resolution, it returns $\tau$ in finite time. Otherwise, it returns failure in finite time.
- $C_{obs}$ is represented implicitly, through the collision checker, rather than explicitly as in the visibility graphs. Collision checking could be expensive, but constructing $C_{obs}$ is even worse.

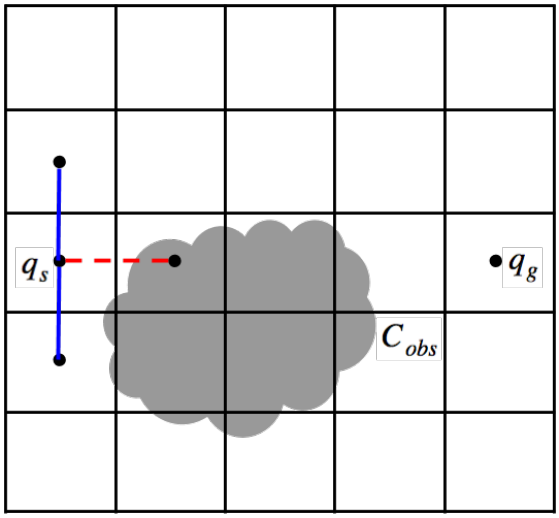



Figure 3.2: Grid search on a sample configuration space.

#### 3.3.2.2 Cons

- The resolution-optimal path is generally not optimal.
- The algorithm is not complete: $\tau$ could exist but discovering it requires an infinitely fine resolution. Or $\tau$ doesn't exist, but the algorithm keeps trying finer resolutions.
- Curse of dimensionality: the number of grid cells is exponential in the number of DOF. $A^*$ is proven in 2-D and 3-D, but gets shaky beyond that.

### 3.3.3 Sampling (next time)

Next lecture we will discuss sampling-based motion planning algorithms, which are what the vast majority of robots run nowadays. These algorithms build a graph on the fly, collecting samples until a point is found that can be added to the path without collisions. Probabilistic roadmaps (PRMs), introduced in 1996, enabled planning for robot arms in high dimensions. The rapidly exploring random tree (RRT), invented for kinodynamic planning, is the tree version of the PRM.