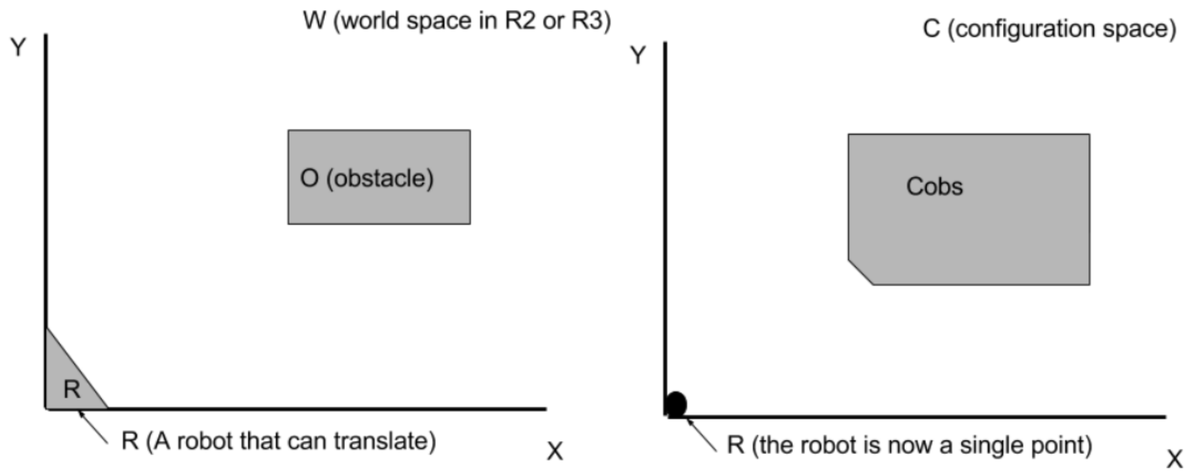
CS 294-115 Algorithmic Human-Robot Interaction **Fall 2017**

Lecture 3: Motion Planning 2

Scribes: David Chan and Liting Sun - Adapted from F2016 Notes by Molly Nicholas and Chelsea Zhang

3.1 Previously

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


W (world space in R2 or R3)
Y
O (obstacle)
R
R (A robot that can translate)
X
C (configuration space)
Y
Cobs
R (the robot is now a single point)
X

Recall that we defined an obstacle in the world as a set $O \subset W$ (In many cases, this is a set of world-space points which we would like to avoid when motion planning). We defined $C_{obs} \subset C$ to be the set of all configurations $q \in C$ such that when the robot's configuration is represented in the world space (with the kinematics function $R : C \rightarrow \mathbb{P}(W)$) the robot does not conflict with the obstacle O. In other words, we defined C_{obs} as:

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

Additionally, we defined C_{free} as the configurations not in C_{obs} (that is, $C_{free} = C_{obs}^c = C \setminus C_{obs}$).

3.2 The Motion Planning Problem

3.2.1 Problem Statement

Motion Planning Problem Statement: How you get from A (a particular configuration) to another configuration (B) without intercepting C_{obs}. Also known as the Piano Movers Problem. (Side note: This was named because moving a piano is actually quite difficult, taking into account obstacles and so on).

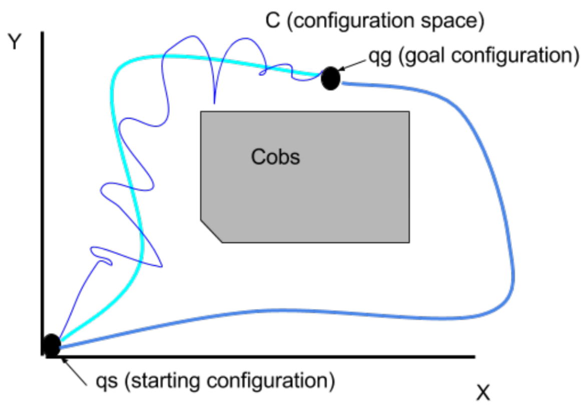
The problem is stated as follows:

Given:

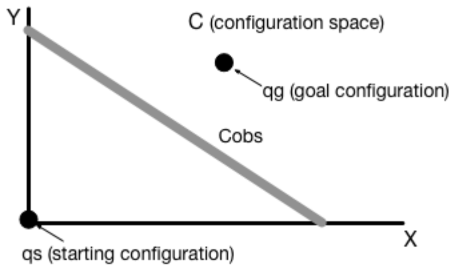
1. World Space W ($\mathbb{R}^2$ or $\mathbb{R}^3$)
2. Obstacle Region $O \subset W$
3. Robot, C-Space C and Kinematics $R : C \to \mathbb{P}(W)$
4. Starting Configuration $q_s \in C$
5. Goal Configuration $q_g \in C$

Task: Compute a continuous path $\tau : [0,1] \to C_{free}$ such that $\tau(0) = q_s$ and $\tau(1) = q_g$.

Notice in the above that τ is a function that maps from the interval $[0,1]$ to the legal configuration space given O. It's only a path, not a timed path (such a timed path is called a trajectory) and focuses on the geometry of the space. The path is unitless, and may not necessarily be optimal by any metric. A solution to the motion planning problem for any give W, O, C, R, q_s, q_g may not exist. Notice in the figure below that there are a large number of solutions, each optimizing a different metric. They are all valid solutions.



On the other hand, in the figure below, there is no solution at all, which means that this motion planning problem is not feasible.



Complexity: In 1985, Reif and Sharir showed that motion planning was NP-Hard. In 1988, Canny showed it to be NP-Complete[1].

3.2.2 Alt-Motion Planning

There are a number of possible extensions to the motion planning problem. You might consider:

- ◇ Maximizing a utility function u over τ
- ◇ Changing τ to be a function of time instead of over the interval $[0, 1$
- ◇ Replacing $q_g \in C$ with a goal state in W
- ◇ Having a set of possible goals instead of a single one
- ◇ Moving obstacles or goals
- ◇ Multi-Step planning
- ◇ The addition of constraints (in W or in C
- ◇ The addition of uncertainty

This is by no means a complete list, however it does provide some insight into extensions and reformulations of the base motion planning problem.

3.3 Algorithms

We will examine three types of algorithms for solving the Motion Planning Problem:

1. Analytic / Geometric
2. Grid search
3. Sampling-based

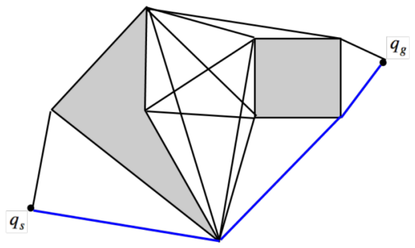
We will cover the first two motion planning algorithms today. Both try to find not just any path τ, but a τ that minimizes Euclidean distance (more or less). Its worth noting that Euclidean distance in configuration space may not be the best metric to optimize perhaps moving the robot wrist is preferable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

[1]For more information on NP-Hard and NP-Complete consult chapter 34 of *Cormen, Thomas H. Introduction to algorithms. MIT press, 2009.*

3.3.1 Visibility Graphs

3.3.1.1 Algorithm Statement

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 below for an example.



A path is found from q_s to q_g by connecting mutually visible polygonal vertices and running graph search

More formally, we first assume that C_{obs} is made up of polygonal connected sets in C-Space. Let $V_{obs} \subset C$ be the set of vertices of the polygons making up C_{obs}, and let $V = \{q_s, q_g\} \cup V_{obs}$. We construct a graph $G = (V, E)$ on the vertex set V with the rule that an edge $e = (v_1, v_2), v_1, v_2 \in V$ is in E if and only if the line from v_1 to v_2 does not intersect ${C_{obs}}^{o}$. In other words:

$$E = \{(u, v) \in V \times V | \forall t \in [0, 1], (1-t)u + (t)v \in C_{free}\}$$

We then run a graph search algorithm (DFS, BFS, A^*, etc.) to compute a path between q_s and q_g if such a path exists.

3.3.1.2 Pros & Cons

Pros:

+ The Visibility Graph algorithm is *complete*[2].
+ The Visibility Graph algorithm is "Optimal" (under the shortest euclidean path metric).

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.
- You need explicit access to the polygonal obstacles in C-Space (that is, you need an explicit definition of C_{obs} which is hard to construct from O *and* these obstacles need to be polygonal).

[2]An algorithm is *complete* iff the algorithm terminates in finite time with a solution should such a solution exists, or will terminate in finite time with an answer of "No Solution"

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. While it initially seems like such an algorithm does not resolve either issue we had with visibility graphs, we will introduce an *implicit* method of determining intersection with C_{obs} which does not require constructing each of the obstacles in C-Space which significantly reduces the time to solution for the algorithm.

3.3.2.1 Algorithm Statement

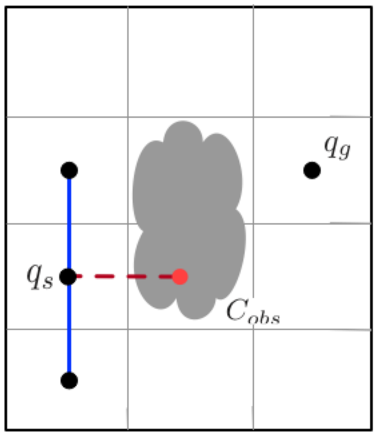
The Grid Search algorithm relies on two essential components - a discritization of C-Space, and a collision checker $\gamma : C \to \{0, 1\}$. The boolean function γ has the role of determining if a point in C is in collision with C_{obs}, that is:

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

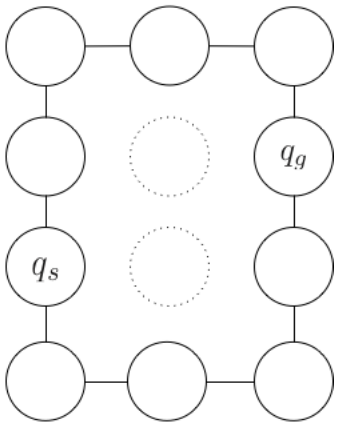
In grid search, the first step is to discretize C-Space (The traditional algorithm uses a grid formed by segmenting each dimension in C-Space). Then, for each discretized cell, a point inside that cell is selected to be the representative point of the cell (this is often the center of the cell, but it is up to the person who is implementing the algorithm).

We then form a graph G by connecting the representative point from each cell to the representative points from each neighboring cell (In $\mathbb{R}^2$, this can create either a 4 or 8 connected grid, depending on the choices made my the algorithm designer).

Then, for each cell D with representative point p, if $\gamma(p) = 0$, then D is considered a "free" cell, and remains in the graph G. On the other hand, if $\gamma(p) = 1$, then D is considered occupied, and removed from G. One example is given below.



a sampled configuration space



the constructed graph G

Finally, graph search is run on the graph G, and if a path is found it is presented as a solution. Otherwise, the algorithm returns that no solution can be found.

3.3.2.2 Pros & Cons

Pros:

+ Only need to check the vertices of G for a collision, which uses an implicit representation of C_{obs}.
+ Solvable by BFS, DFS, A^* and other traditional graph search algorithms
+ Resolution Complete. This means that if a solution exists that falls within the resolution of our discritization, it will be presented in finite time, otherwise the algorithm will terminate in finite time.
+ Resolution Optimal (Exactly what you think it means)

Cons:

- The algorithm may not find paths that are too fine for the level of discritization chosen by the algorithm designer.
- Discritization is expensive (curse of dimension), which means that the number of grids increases exponentially as the increase of the number of DOFs. (Side note: this can now be solved using some advanced search algorithms for high DOFs robots.)

3.4 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.