CS 333: Safe and Interactive Robotics Fall 2017

Lecture 1: Motion Planning

Scribes: Laura Blumenschein and Charles Lu

1.1 Overview

This lecture covers some fundamentals in the field of motion planning, or finding a sequence of motions to allow a robot to achieve a desired position under constraints, as well as some of the standard algorithms for solving the motion planning problem.

The major topics we will cover are:

1. Configuration space
2. Motion planning problem formulation
3. Motion planning techniques

1.2 Configuration Space

The configuration space is an important foundation to understand when beginning to talk about motion planning problems. A configuration, q, is a mathematical quantity that completely defines a robot's position and pose. Configuration space then, represented mathematically by C, is the set of all possible configurations, $q \in C$.

World space, denoted by W, is distinct from configuration space and is $\mathbb{R}^2$ or $\mathbb{R}^3$ for the majority of problems. A mapping of the robot into the world frame is:

$$R : C \to \mathbb{P}(W) \qquad R(q) = \{x \mid x \in W, x \in \text{robot}\} \tag{1.1}$$

where $\mathbb{P}(W)$ is the power set, or set of all subsets, of the world frame. Essentially, the function R takes a robot's configuration and gives the set of points occupied by the robot in the world frame.

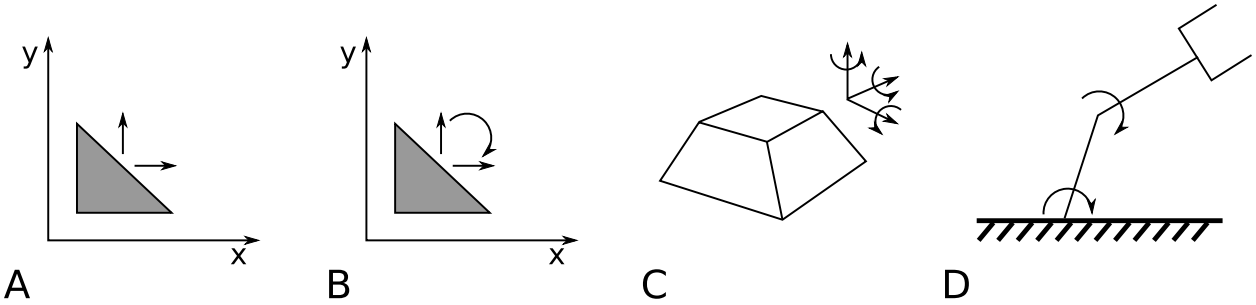

Figure 1.1: Example robots with various configuration spaces

The configuration space is highly dependent on the robot and its degrees of freedom. Some examples of robots can be seen in Fig. 1.1:

- Robot A: Translation of a robot in $\mathbb{R}^2$, the configuration is $q : (x, y)$ and configuration space $C = \mathbb{R}^2$
- Robot B: Translation and rotation of a robot in $\mathbb{R}^2$, the configuration is $q : (x, y, \theta)$ and configuration space $C = \mathbb{R}^2 \times SO(2)$
- Robot C: Rigid body motion of a robot in $\mathbb{R}^3$, the configuration is $q : (x, y, z, \alpha, \beta, \gamma)$ and configuration space $C = \mathbb{R}^3 \times SO(3) = SE(3)$
- Robot D: A two degree of freedom robotic manipulator in $\mathbb{R}^2$, the configuration is $q : (\theta_1, \theta_2)$ and the configuration space $C = SO(2) \times SO(2)$

1.2.1 Forward and Inverse Kinematics

Forward kinematics (FK) refers to the task of mapping a configuration q to a point in the world. Specifically, we define a function $\phi : C \to W$, i.e. $\phi(q) = x$ where $q \in C$ and $x \in W$. ϕ is defined for a specific point on a robot; for instance, in the example of the robot with a gripper and two joints, we might have ϕ map the robot's configuration to the position of the robot's gripper: $\phi(\theta_1, \theta_2) = (x, y)$.

Inverse kinematics (IK) is a more difficult task: given a point in the world $x \in W$, return a set $\{q \mid \phi(q) = x\}$. Generally, inverse kinematics is performed through optimization-based or numerical-based methods, e.g. IKFast.[ikf]

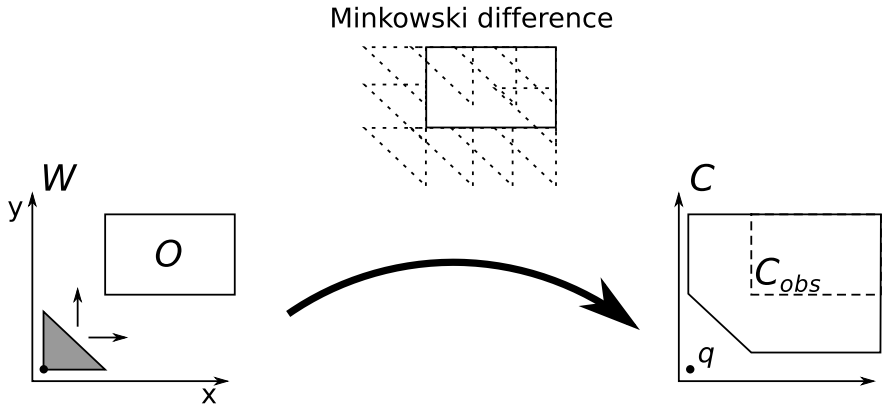

Figure 1.2: Calculation of obstacles in configuration space using the Minkowski difference. The Minkowski difference can be calculated by taking the inverse robot shape, $R' = -R$, and adding it to the borders of the obstacles.

1.2.2 Obstacles

Obstacles must be considered in many motion planning problems. We define the set of obstacles in the world as $O \subset W$.

Taking the obstacles from W-space to C-space is difficult. We define $C_{obs} = \{q \in C \mid R(q) \cap O \neq \emptyset\}$; in other words, C_{obs} is the set of configurations q in configuration space such that the robot collides with an obstacle. We can also define $C_{free} = C \setminus C_{obs}$.

When dealing with 1D and 2D translations of the robot, C_{obs} can also be defined as $C_{obs} = O \ominus R(\emptyset) = \{o - r \mid o \in O, r \in R(\emptyset)\}$, where $\ominus$ refers to the Minkowski difference. An example of using the Minkowski difference to find C_{obs} can be seen in Fig. 1.2

We now introduce a theorem related to the convexity of C_{obs}:

Theorem. If O and $R(\emptyset)$ are convex, then C_{obs} is convex (if they are in the same space).

Proof. We must prove that for any $t_1, t_2 \in C_{obs}$ and $\lambda \in [0,1]$, we have $\lambda t_1 + (1-\lambda)t_2 \in C_{obs}$. Let $R' = -R$. We have $C_{obs} = O \ominus R(\emptyset) = O \oplus R'$. Then $t_1 = o_1 + r_1, t_2 = o_2 + r_2$ with $o_1, o_2 \in O, r_1, r_2 \in R'$. Since O and $R(\emptyset)$ are convex, then $\lambda o_1 + (1-\lambda)o_2$ and $\lambda r_1 + (1-\lambda)r_2$. By adding these two equations we achieve the desired result.

Examples for the configuration space obstacles of other robot types can be seen in Fig. 1.3. The 1-joint manipulator has a configuration space on the real line from $[0, 2\pi]$ and an obstacle is translated to a section of the line corresponding to angles where the robot body is in collision. Note, the part of the configuration space on the "other side" of the obstacle is not in C_{obs}, even if the manipulator cannot reach this part of configuration space. C_{obs} only tells us where the obstacles are. A 2-joint manipulator has a configuration space in 2D from $[0, 2\pi]$ in each direction. For this example, since only the first link's angle determines collisions, the obstacle in configuration space extends to cover all values of the second link angle.

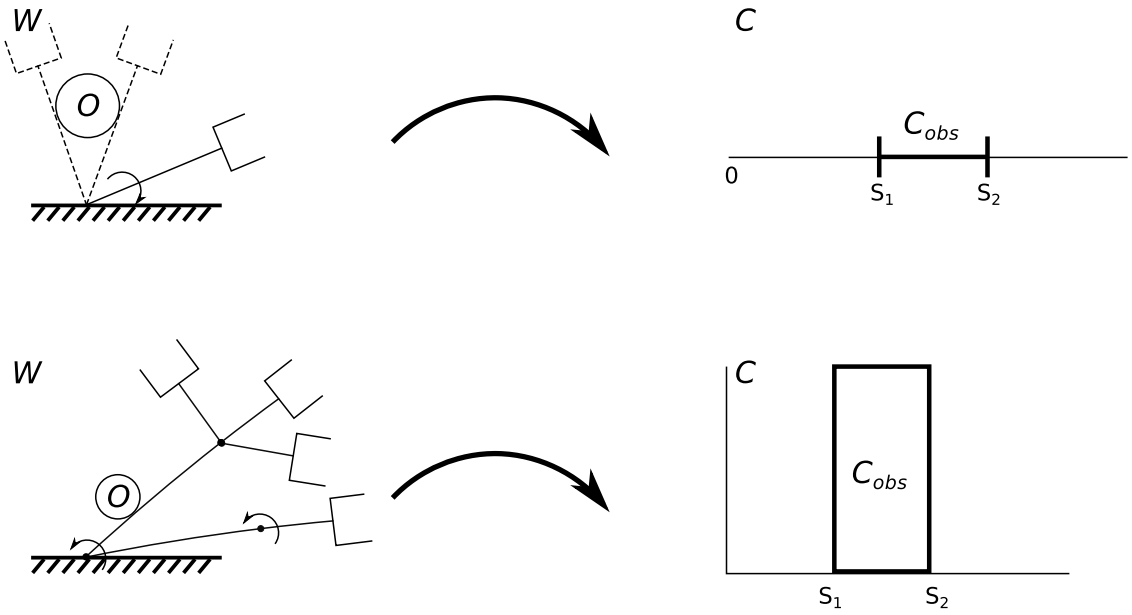

Figure 1.3: Examples showing C_{obs} for 1-joint and 2-joint robotic manipulators.

1.3 The Motion Planning Problem

We now define the motion planning problem (Fig. 1.4).

Given:

- world space W (in $\mathbb{R}^2$ or $\mathbb{R}^3$)

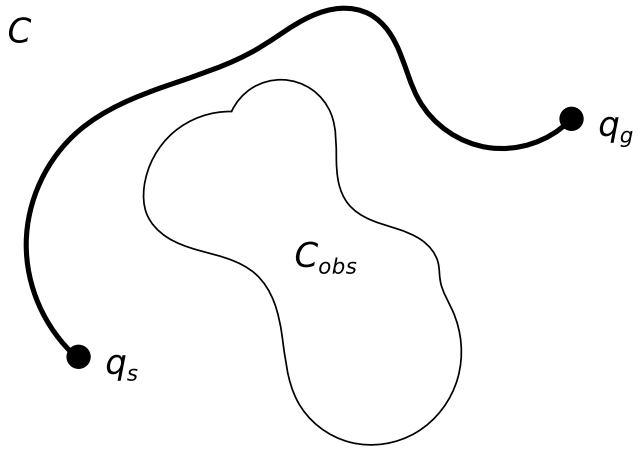

Figure 1.4: The motion planning problem

- obstacle region $O \in W$
- robot, C-space ($R : c \to \mathcal{P}(W)$)
- starting and goal configurations q_s, q_g

Goal: Find a function $\tau : [0,1] \to C_{free}$ representing the path of the robot in configuration space, where $\tau(0) = q_s$ and $\tau(1) = q_g$.

Certain motion planning problems may involve variations on the original motion planning problem:

- maximize $u(\tau)$
- change τ to be timed
- multiple goals
- dynamic obstacles/goals
- additional constraints on W or C
- uncertainty
- multi-step planning

1.4 Motion Planning Techniques

1.4.1 Geometric Methods

1.4.1.1 Visibility Graph

The visibility graph method, initially introduced by Nils Nilsson in 1969, is a geometric method for motion planning.[Lozano-Pérez and Wesley, 1979]

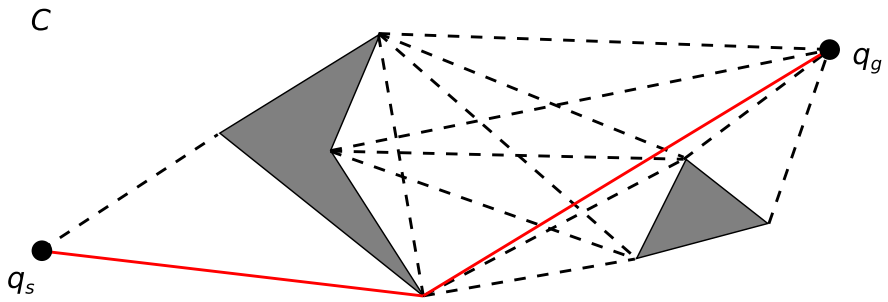

Figure 1.5: Example of using visibility graphs to solve a motion planning problem. Obstacles must be polygonal in configuration space for this method.

The method relies on the assumption that all obstacles are polygonal in C-space. It is important to note that any robots whose configurations involve rotations therefore likely cannot use this method.

First, all vertices of obstacles, in addition to the start and goal points, are connected to one another if they "see" each other in C-space, i.e. there is no obstacle along the straight line between the two vertices. This forms a graph, upon which a graph search algorithm like A*, BFS, etc. can be used to obtain a motion plan (Fig. 1.5).

Formally:

1. Let $V_{obs} \subset C$ be the set of all vertices of obstacles.
2. Let $V = V_{obs} \cup (q_s, q_g)$.
3. Form $G : (V, E)$ where $E = \{(v, u) \in V \times V \mid \forall t \in [0, 1] : (1 - t)v + tu \in C_{free}\}$.
4. Use any graph search algorithm to find path from q_s to q_g.

The advantages of the visibility graph method are:

- **Complete:** if there exists some τ, the algorithm will return it in finite time; if not, it will fail.
- **Optimal:** the algorithm is optimal under the shortest Euclidean path metric.

Disadvantages are:

- Only addresses subset of planning problems due to limitation of polygonal obstacles.
- Assumes explicit access to C_{obs}.

1.4.2 Grid-Based Methods

1.4.2.1 Grid Search Algorithm

Grid-based methods rely on two essential components:

- Discretization of the C-space

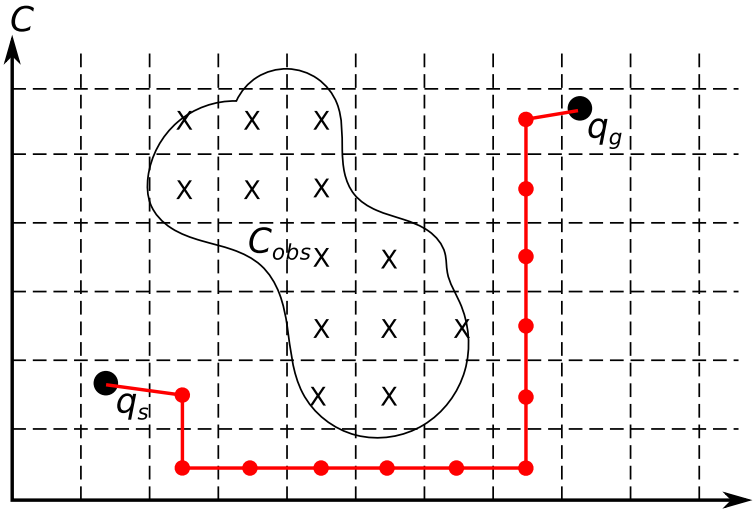

Figure 1.6: Example of using grid search to solve a motion planning problem. Grid points in collision with obstacles are removed and a path is found in the remaining grid points.

- A collision checker: $\gamma : C \to 0, 1$ where

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

The grid search algorithm simply gets the center of each discretized cell, uses the collision checker to find the points in C_{free}, and connects neighboring nodes in C_{free} (Fig. 1.6). A similar graph search algorithm like BFS can be used to create a motion plan from q_s to q_g.

It is possible for obstacles to exist on the path between two cell centers, despite the two centers being in C_{free}. A simple way to address this problem is to assume that the minimum size of obstacles is w, and to check every point between the center of cells with interval λw where $\lambda < 1$.

This method is both resolution complete and resolution optimal, meaning that if a solution exists it will be found and become optimal as the resolution of the graph increases. But grid search is not actually complete or optimal like the visibility graphs algorithm is. Another advantage is that classic graph algorithms like BFS, DFS, A*, etc. can be used for the graph search portion of the algorithm. However, grid-based methods suffer from the curse of dimensionality when the C-space is very high dimensional.

1.4.3 Sampling-Based Methods

1.4.3.1 Probabilistic Roadmap (PRM)

The probabilistic roadmap planner [Kavraki et al., 1996] is a sampling-based method for motion planning (Fig. 1.7). The basic outline is as follows:

1. Randomly sample points from the configuration space, along with q_s and q_g.
2. Keep samples which are in C_{free}.

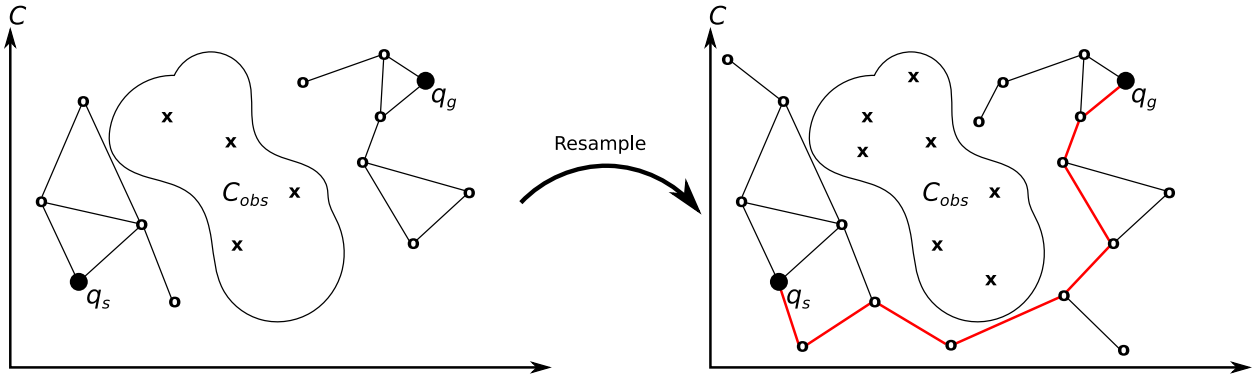

Figure 1.7: Example of using probabilistic roadmaps (PRMs) to solve a motion planning problem.

3. Connect samples to each other using k-nearest neighbors or some other method, forming connected components of samples.
4. If q_s and q_g are not in the same connected component, sample more points in (1).
5. Otherwise, use a graph search algorithm to find τ.

PRM is probabilistically complete, meaning if a solution exists, it will be found as the number of samples increases (though if no solution exists, the algorithm will not be able to return a definitive failure in finite time like a truly complete planner).

1.4.3.2 Rapidly-exploring Random Tree (RRT)

The rapidly-exploring random tree algorithm [LaValle, 1998] is another sampling-based method.

References

Ikfast: The robot kinematics compiler. URL `http://openrave.org/docs/latest_stable/openravepy/ikfast/`.

Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4): 566–580, 1996.

Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.