CS 294-115 Algorithmic Human-Robot Interaction **Fall 2015**

Lecture 2: Motion Planning

Scribes: Samaneh, Peggy

In this session we cover the functional robotics side of motion planning, and optimization will be discussed in the next session. We will have discussion on the papers from psychology, robotics, and HRI sides thereafter in the course.

2.1 Outline

- Configuration spaces
- Problem statement
- Algorithms: from 60s to 2000
  - Geometric algorithms
  - Grid-based search
  - Randomized sampling method

2.2 Configuration Spaces

Configuration space is denoted by C, and elements in the configuration space $q \in C$ contain everything that we need in order to know where the robot is.

Let A be a function that takes a configuration and gives the set of points making up the robot in the world, when the robot is in that configuration:

$$A : C \to P(w)$$

where P is a power set and W denotes the world in $\mathbb{R}^2$ or $\mathbb{R}^3$. $A(q)$ represents every point x that is in the world and can be occupied by the robot in position q. It can be defined as the following set:

$$A(q) = \{x | x \in W, x \text{ occupied by the robot in } q\}$$

Example 1: If we have an $x - y$ plane on which the robot can translate, the configuration space is two-dimensional, $q = (x, y)$:

Figure 2.2a shows $A(0)$.

Example 2: When a robot can both be translated and rotated in the space with 3 degrees of freedom (DOF), $q = (x, y, \theta)$, $A(q)$ is shown as Figure 2.2b.

Example 3: For a 2 link planar arm, $q = (\theta_1, \theta_2)$, $A(q)$ is shown as Figure 2.2c.

Note: Robot arms that manipulate rigid body objects have at least 6DOFs. Human arms have 7DOFs.

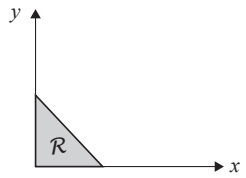
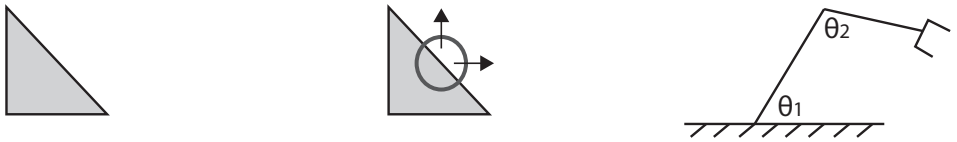


Figure 2.1: A robot R that moves in x and y.



(a) $A(q)$ when a robot translates in x-y.

(b) $A(q)$ when a robot translates and rotates in x-y.

(c) $A(q)$ when for a robot arm that rotates.

Figure 2.2: $A(q)$ with different robot behaviors.

2.2.1 A World with Obstacles

Real worlds have obstacles. We denote obstacle regions by O ($O \subset W$) and the obstacle region in the configuration space by C_{obs},

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

and $C_{free} = C \setminus C_{obs}$ is the free space where we can keep the robot without collisions.

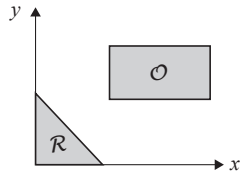


Figure 2.3: A robot R that moves in x and y with an obstacle O.

Note: C_{obs} is a closed set because it contains its limit points. Limit points are the touching configurations. C_{free} is an open set.

Figure 2.4 shows the configuration space for a robot when considering an obstacle.

2.2.2 Minkowski Difference

In 2D when we only allow translations, C_{obs} can be computed as:

$$C_{obs} = O \ominus A(0) = \{o - a | o \in O, a \in A(0)\}$$

where $A(0)$ is A at the origin (what points comprised the robot when it is located at $(0,0)$), $\ominus$ is the Minkowski difference, i.e. the set comprising the difference between all possible pairs of $o \in O$ and $a \in A(0)$.

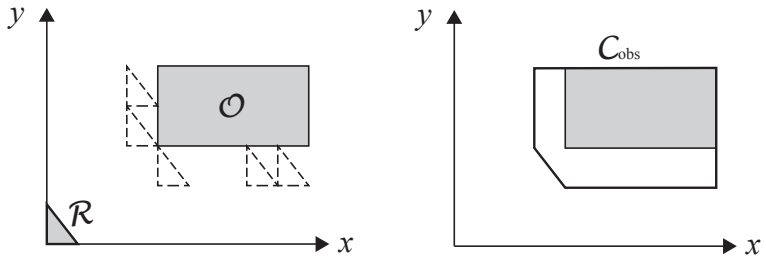


Figure 2.4: C_{obs} for a robot R that translates in x-y with a rectangle obstacle O

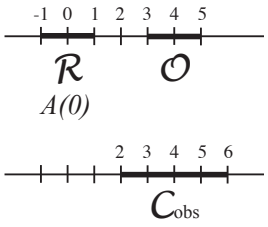


Figure 2.5: Minkowski difference

Theorem: If O and $A(0)$ are convex, then C_{obs} is convex.

2.2.3 Considering Translations and Rotations

If we can also rotate the robot, then C_{obs} If we have translations and rotations possible for the robot, then the obstacle can be represented in a 3-D space as a volume. Each orientation value induces a slice of the obstacle, computed as the Minkovski difference when we fix the orientation to that value. The figure below shows the slice for $\theta = 0$.

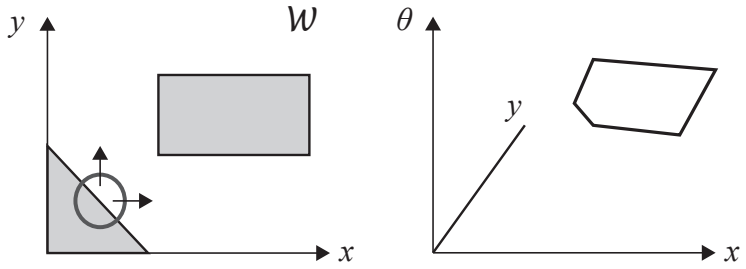
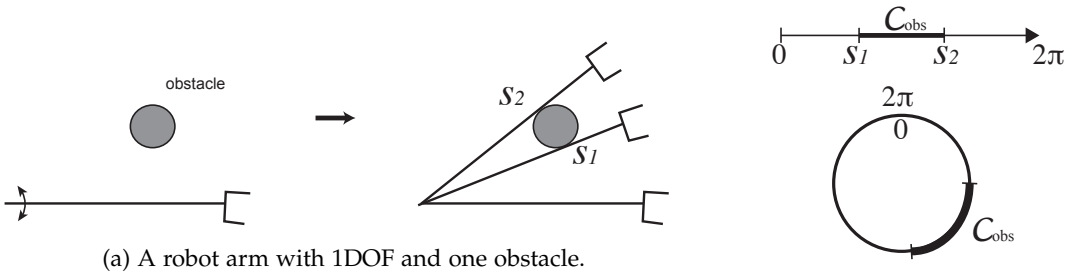


Figure 2.6: Considering translations and rotations

For an arm with one degree of freedom, C_{obs} looks like as:

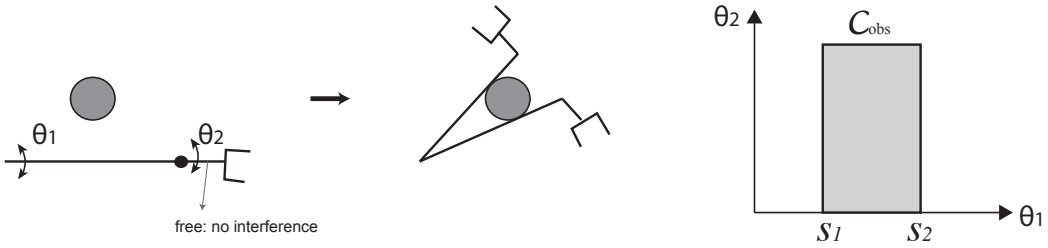


(a) A robot arm with 1DOF and one obstacle.

(b) C_{obs} for this case.

Figure 2.7: C_{obs} considering an arm with 1DOF.

For an arm with two degrees of freedom, C is similar to a donut shape and C_{obs} is some part of that as:



(a) A robot arm with 2DOF and one obstacle.

(b) C_{obs} for this case.

Figure 2.8: C_{obs} considering an arm with 2DOF.

If we have a 2-linked planar arm, C_{obs} looks like:

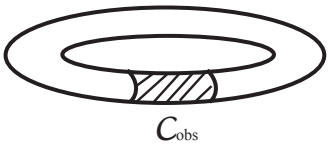


Figure 2.9: C_{obs} for a 2-linked planner robot arm.

2.3 Motion Planning

Motion planning is about moving from one configuration to another while avoiding obstacles.

Piano Mover's Problem:

We assume a world W in $\mathbb{R}^2$ or $\mathbb{R}^3$, an obstacle region in the world ($O \subset W$), a robot with a configuration space C and its corresponding A function, $C_{free} = C \setminus C_{obs}$, $q_s \in C_{free}$ is the starting configuration, and $q_g \in C_{free}$ is the goal configuration.

The goal is to compute a path $\tau : [0,1] \to C_{free}$ such that $\tau(0) = q_s$ and $\tau(1) = q_g$.

Computational complexity: motion planning is NP-hard.

Visibility Graphs (Nilsson '69)

- Assume polygonal obstacles in C-space
- Connect all vertices of all polygons that can "see" each other (straight line paths are collision free).

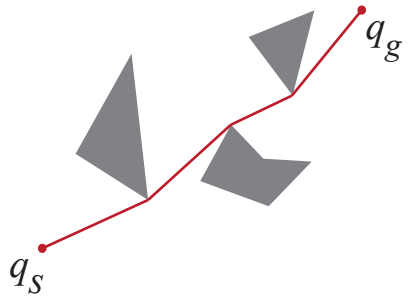


Figure 2.10: A visibility graph that finds a path between q_s and q_g with three obstacles.

Pros and Cons: This method is complete which means in a finite time, it will find and return the solution if it exists and returns failure otherwise. It is also optimal meaning it finds the shortest path. However, it assumes explicit representation of C_{obs}, and only works in 2D.

Collision Checking Instead of Representing C_{obs}

A collision checker is a query mechanism that returns whether the robot is in collision or not at a particular configuration:

$$\gamma : q \mapsto \begin{cases} 0, & \text{no collision,} \\ 1, & \text{otherwise} \end{cases}$$

Grid Search:

Idea: grid up the configuration space C by discretizing every dimension resulting in vertices (for the points in C_{free}) connected by edges.

Motion planning is then a graph search problem, solvable for instance by A^* search: prioritize nodes by combining cost-to-come with a heuristic of cost-to-go.

Pros and Cons: This method is not complete because if the resolution is too coarse, there might be a solution but the algorithm might not be able to find it. It is only "resolution-complete". It is also optimal up any inefficiencies induced by having a resolution that is too coarse. This method does not scale well to high-dimensional spaces since the number of nodes is exponential in the number of DOFs.

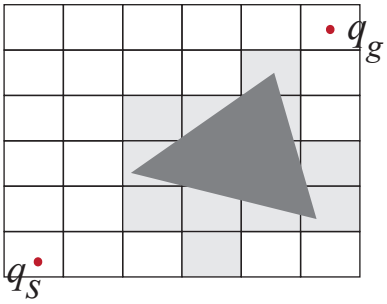


Figure 2.11: A grid-search graph problem with one obstacle.

Sampling-Based Planners:

Idea: represent C_{free} via sampled configurations. (simple, works remarkably well)

Probabilistic Road Maps (PRMs) [Kavraki '96]

We assume a collision checker γ and a simple planner B_s which returns a solution fast if it finds one, but is not complete.

$$B_s(q_1, q_2) \rightarrow \begin{cases} \text{a path if it finds one quickly} \\ \text{failure otherwise} \end{cases}$$

An example of B_s is collision-checking the straight line, and returning it if collision-free, returning failure otherwise.

1. Start with q_s, q_g,
2. Sample M more milestones in C_{free} (rejection sampling),
3. Try to connect using B_s:
   - all pairs
   - everything in an r-disc around each configuration
   - k-nearest neighbors of each configuraiton: k-PRM
4. Try to find a path on the graph (called a roadmap) from q_s to q_g
   - If success, return the path.
   - Else, go back to step 2.

Idea 1: $M = 1$ (no more batches!) stop when q_s and q_g are in the same connected component.

Idea 2: Keep track of the connected components of q_s and q_g (denoted as G_s and G_g). Only try to connect to G_s and G_q.

Idea 3: 1-PRM.

This is now the Bidirectional Rapidly-Exploring Random Tree (non-incremental version): Bi-RRT

The RRT was introduced by [Kuffner and LaValle 2001]. Having a tree instead of a graph allows for kinodynamic planning.

Pros and Cons. Not complete, only probabilistically complete (probability of success approaches 1 as time goes to infinity). Not optimal, usually there is a post-processing stage to shortcut it. But, works in high-dimensional spaces.