CMSC 425: Lecture 17
Motion Planning: Basic Concepts

Reading: Today's material comes from various sources, including "AI Game Programming Wisdom 2" by S. Rabin and "Planning Algorithms" by S. M. LaValle (Chapts. 4 and 5).

Recap: Previously, we discussed navigation meshes as a technique for planning the motion of walking agents in games. In this and future lectures, we will delve deeper into the theory and practice of motion planning as it relates to game programming.

Configuration Spaces: To begin, let us consider the problem of planning the motion of a single agent among a collection of obstacles. Since the techniques that we will be discussing originated in the field of robotics, henceforth we will usually refer to a moving agent as a "*robot*". The environment in which the agent operates is called its *workspace*, which consists of a collection of geometric objects, called *obstacles*, which the robot must avoid. We will assume that the workspace is static, that is, the obstacles do not move.[1] We also assume that a complete geometric description of the workspace is available to us.[2]

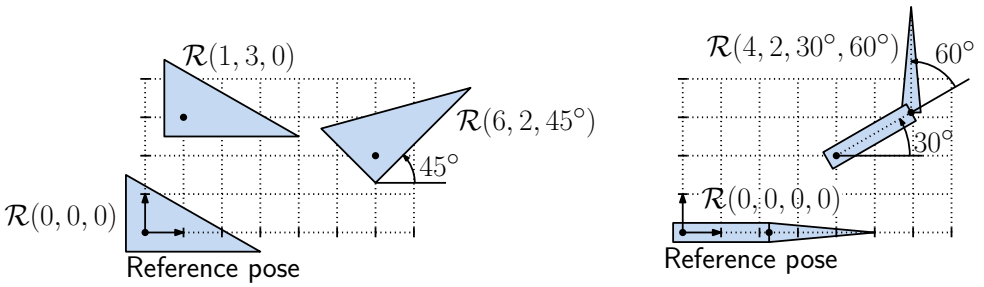
For our purposes, a *robot* will be modeled by two main elements. The first element is the robot's geometric model, say with respect to its reference pose (e.g., positioned at the origin). The second is its *configuration*, by which we mean a finite sequence of numeric parameters that fully specifies the position of the robot. Combined, these two elements fully define the robot's exact shape and position in space.

For example, suppose that the robot is a 2-dimensional polygon that can translate and rotate in the plane (shown as a triangle in Fig. 1(a)). Its geometric representation might be given as a sequence of vertices, relative to its reference position. Let us assume that this reference pose overlaps the origin. We refer to the location of the origin as the robot's *reference point*. The robot's configuration may be described by its translation, which we can take to be the (x, y) coordinates of its reference point after translation and an angle θ that represents the counterclockwise angle of rotation about its reference point (see Fig. 1(a)). Thus, the configuration is given by a triple (x, y, θ). We define the space of all valid configurations to be the robot's *configuration space*. For any point $p = (x, y, \theta)$ in this space, we define $\mathcal{R}(p)$ to be the corresponding *placement* of the robot in the workspace. The dimension of the configuration space is sometimes referred to as the robots number of *degrees of freedom* (DOF).

In 3-dimensional space, a similarly rigid object can be described by six parameters, the (x, y, z)-coordinates of the object's reference point, and the three Euler angles (θ, ϕ, ψ) that

[1] The assumption of a static workspace is not really reasonable for most games, since agents move and structures may change. A common technique for dealing with dynamic environments is to separate the static objects from the dynamic ones, plan motion with respect to the static objects, and then adjust the plan incrementally to deal with the dynamic ones.

[2] The assumption of a known workspace is reasonable in computer games. Note that this is not the case in robotics, where the world surrounding the robot is either unknown or is known only approximately based on the robots limited sensor measurements.



(a) (b)

Fig. 1: Configurations of: (a) translating and rotating robot and (b) a translating and rotating robot with a revolute joints.

define its orientation in space.)[3]

A more complex example would be an *articulated arm* consisting of a set of links, connected to one another by a set of *revolute joints*. The configuration of such a robot would consist of a vector of joint angles (see Fig. 1(b)). The geometric description would probably consist of a geometric representation of the links. Given a sequence of joint angles, the exact shape of the robot could be derived by combining this configuration information with its geometric description.

Free Space: Because of limitations on the robot's physical structure and the obstacles, not every point in configuration space corresponds to a legal placement of the robot. Some configurations may be illegal because:

- The joint angle is outside the joint's operating range. (E.g., you can bend your knee backwards, but not forwards . . . ouch!)
- The placement associated with this configuration intersects some obstacle (see Fig. 2(a)).

Such illegal configurations are called a *forbidden configurations*. Given a robot $\mathcal{R}$ and workspace S, the set of all forbidden configurations is denoted $C_{\text{forb}}(\mathcal{R}, S)$, and all other placements are called *free configurations*, and the set of these configurations is denoted $C_{\text{free}}(\mathcal{R}, S)$, or *free space*. These two sets partition configuration space into two distinct regions (see Fig. 2(b)).

C-Obstacles and Paths in Configuration Space: *Motion planning* is the following problem: Given a workspace S, a robot $\mathcal{R}$, and initial and final configurations $s, t \in C_{\text{free}}(\mathcal{R}, S)$, determine whether it is possible to move the robot from one configuration by a path $\mathcal{R}(s) \rightarrow \mathcal{R}(t)$consisting entirely of free configurations (see Fig. 3(a)).

Based on the definition of configuration space, it is easy to see that the motion planning problem reduces to the problem of determining whether there is a path from s to t in con-

[3]A quaternion might be a more reasonable representation of the robot's angular orientation in space. You might protest that the use of a quaternion will involve four parameters rather than three. But remember that the quaterions used for representing rotations are *unit quaternions*, meaning that once three of the parameters are given, the fourth one is fixed.

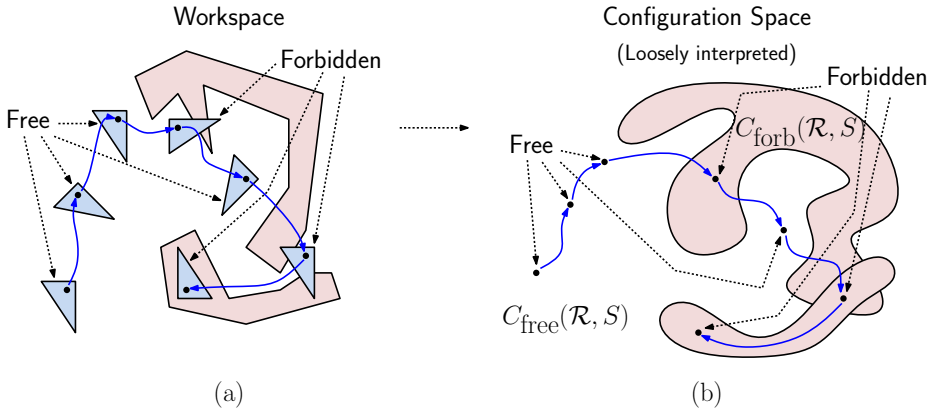


Fig. 2: Workspace showing free and forbidden configurations and a possible configuration space.

figuration space (as opposed to the robot's workspace) that lies entirely within the robot's free configuration subspace (see Fig. 3(b)). Thus, we have reduced the task of planning the motion of a robot in its workspace to the problem of finding a path for a single point through free configuration space.

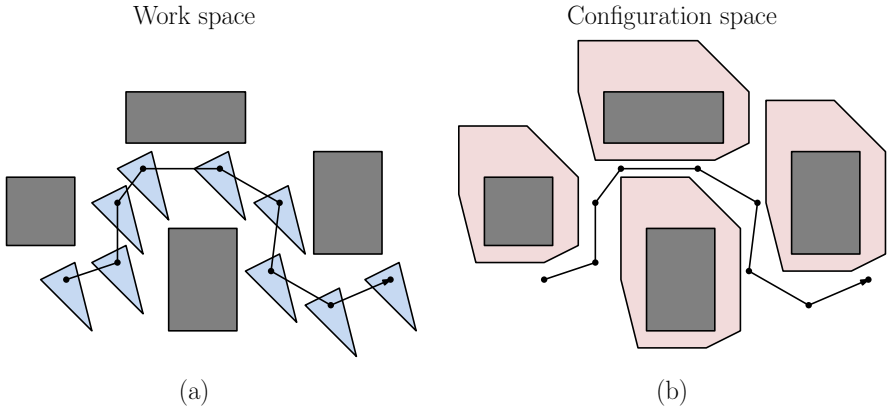


Fig. 3: Motion planning: (a) workspace with obstacles and (b) configuration space and C-obstacles.

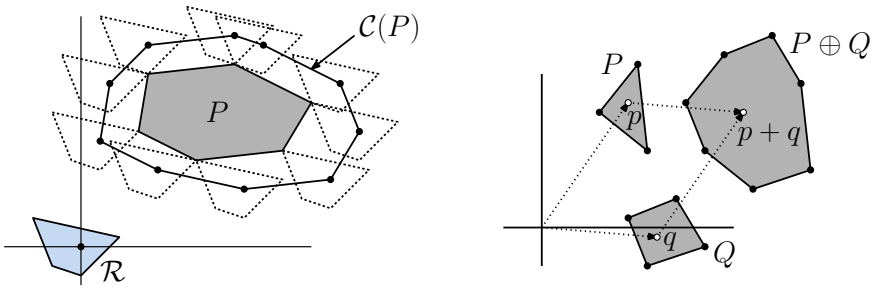
Configuration Obstacles and Minkowski Sums: Since high-dimensional configuration spaces are difficult to visualize, let's consider the simple case of translating a convex polygonal robot in the plane amidst a collection of polygonal obstacles. In this cased both the workspace and configuration space are two-dimensional. We claim that, for each obstacle in the workspace, there is a corresponding *configuration obstacle* (or *C-obstacle*) that corresponds to it in the sense that if $\mathcal{R}(p)$ does not intersect the obstacle in the workspace, then p does not intersect the corresponding C-obstacle.

For simplicity, let us assume that the reference point for our robot $\mathcal{R}$ is at the origin. Let $\mathcal{R}(p)$ denote the *translate* of the robot so that its reference point lies at point p. Given a polygonal obstacle P, the corresponding C-obstacle is formally defined to the set of placements of $\mathcal{R}$

that intersect P, that is

$$\mathcal{C}(P) = \{p \ : \ \mathcal{R}(p) \cap P \neq \emptyset\}.$$

One way to visualize $\mathcal{C}(P)$ is to imagine "scraping" $\mathcal{R}$ along the boundary of P and seeing the region traced out by $\mathcal{R}$'s reference point (see Fig. 4(a)).



(a) (b)

Fig. 4: Minkowski sum of two polygons.

Given $\mathcal{R}$ and P, how do we compute the configuration obstacle $\mathcal{C}(P)$? To do this, we first introduce the notion of a *Minkowski sum*. Let us think of points in the plane as vectors. Given any two sets P and Q in the plane, define their *Minkowski sum* to be the set of all pairwise sums of points taken from each set (see Fig. 4(b)), that is,

$$P \oplus Q \ = \ \{p + q \ : \ p \in P, \ q \in Q\}.$$

Also, define $-S = \{-p \ : \ p \in S\}$. (In in the plane $-S$ is just the 360° rotation of S about the origin, but this does not hold in higher dimensions.) We introduce the shorthand notation $\mathcal{R} \oplus p$ to denote $\mathcal{R} \oplus \{p\}$. Observe that the *translate* of $\mathcal{R}$ by vector p is $\mathcal{R}(p) = \mathcal{R} \oplus p$. The relevance of Minkowski sums to C-obstacles is given in the following claim.

Claim: Given a translating robot $\mathcal{R}$ and an obstacle P, $\mathcal{C}(P) = P \oplus (-\mathcal{R})$ (see Fig. 5).

Proof: Observe that $q \in \mathcal{C}(P)$ iff $\mathcal{R}(q)$ intersects P, which is true iff there exist $r \in \mathcal{R}$ and $p \in P$ such that $p = r + q$ (see Fig. 5(a)), which is true iff there exist $-r \in -\mathcal{R}$ and $p \in P$ such that $q = p + (-r)$ (see Fig. 5(b)), which is equivalent to saying that $q \in P \oplus (-\mathcal{R})$. Therefore, $q \in \mathcal{C}(P)$ iff $q \in P \oplus (-\mathcal{R})$, which means that $\mathcal{C}(P) = P \oplus (-\mathcal{R})$, as desired.

It is an easy matter to compute $-\mathcal{R}$ in linear time (by simply negating all of its vertices) the problem of computing the C-obstacle $\mathcal{C}(P)$ reduces to the problem of computing a Minkowski sum of two convex polygons. We'll show next that this can be done in $O(m+n)$ time, where m is the number of vertices in $\mathcal{R}$ and n is the number of vertices in P.

Note that the above proof made no use of the convexity of $\mathcal{R}$ or P. It works for any shapes and in any dimension. However, computation of the Minkowski sums is most efficient for convex polygons. We will not present the algorithm formally here, but here is an intuitive explanation. First, compute the vectors associated with the edges of each polygon and merge them into a single list, sorted by angular order. Then link them together end-to-end (see Fig. 6). (It is not immediately obvious that this works, but it can be proved to be correct.)

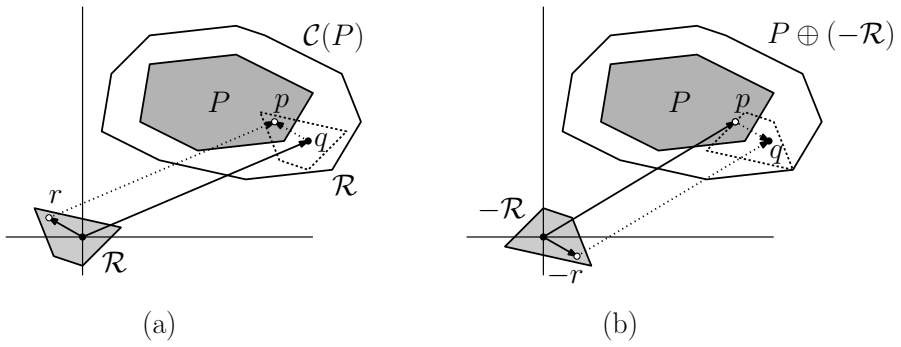


Fig. 5: Configuration obstacles and Minkowski sums.

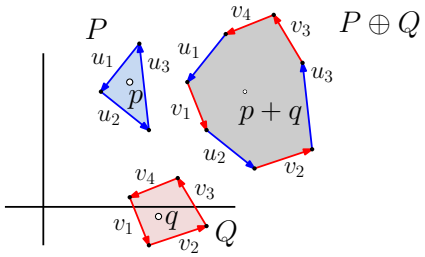


Fig. 6: Computing the Minkowski sum of two convex polygons.

C-Obstacles for Rotating Robots: When rotation is involved, this scraping process must consider not only translation, but all rotations that cause the robot's boundary to touch the obstacle's boundary. (One way to visualize this is to fix the value of θ, rotate the robot by this angle, and then compute the translational C-obstacle with the robot rotated at this angle. Then, stack the resulting C-obstacles on top of one another, as θ varies through one complete revolution. The resulting "twisted column" is the C-obstacle in 3-dimensional space.) Note that because the configuration space encodes not only translation, but the joint angles as well. Thus, a path in configuration space generally characterizes both the translation and the individual joint rotations. (This is insanely hard to illustrate, so I hope you can visualize this on your own!)

When dealing with polyhedral robots and polyhedral obstacles models under translation, the C-obstacles are all polyhedra as well. However, when revolute joints are involved, the boundaries of the C-obstacles are curved surfaces, which require more effort to process than simply polyhedral models. Complex configuration spaces are not typically used in games, due to the complexity of processing them. Game designers often resort to more ad hoc tricks to avoid this complexity, and the expense of accuracy.