3
Configuration Space

In Ch. 1.1, we introduced the configuration space, C-space, and workspace mainly for point robots. In this chapter, we explain their main concepts and various issues, since the C-space is one of main components of most planners.

The C-space represents all the parameter spaces of a robot. As a result, a configuration in the C-space is a complete specification of every points of the robot. For rigid robots, dimensions of the C-space commonly correspond to set of positions and orientations of the robots. Since the robot is rigid, we do not need to explicitly represent its shape. In other words, once we specify the position and orientation of the robot, we can represent its every point in the workspace by placing its rigid shape that is transformed with the specified position and orientation.

Fig. 3.1 shows a manipulator that is fixed to the ground and has two joints. Since the manipulator is a rigid robot with two joints, we can specify its complete shape based on those two joints, θ_1 and θ_2. Typically, we concern a position of the end-effector of the manipulator. Once we specify certain values on those two joints, we can compute the exact location of the end effector. Computing such locations and orientations given joint angles is known as forward kinematics. On the other hand, computing joint angles given an end-effector pose is inverse kinematics.

Specifying a position of a robot in the workspace can have multiple corresponding configurations in the C-space, resulting in ambiguous representation.

One may consider to use the end-effector position in the workspace as an internal representation for controlling the robot. This could be an intuitive approach, but poses ambiguous representations. For example, given an end-effector position, the blue ball in the right figure of Fig. 3.1, there could be multiple combinations of joint angles, realizing the same end-effector position. Since using the end-effector position in the workspace can be ambiguous in terms of specifying joint angles of the manipulator, it is not desirable for controlling the manipulator and many other planning problems.

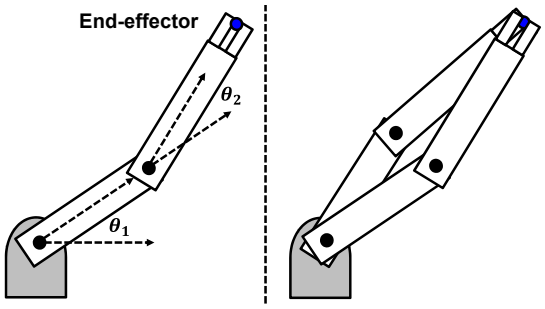

Figure 3.1: The left figure shows a manipulator with two joints, and the right figure shows that there could be multiple joint configurations whose end-effector is located at the same position in the 2D workspace.

Dimension of C-space. There can be different ways of specifying parameters for controlling a robot. Nonetheless, the dimension of the C-space is defined to have the minimum number of parameters needed to specify the robot completely. At this case, the minimum dimension is also called the number of degrees of freedom (DoFs) of the robot.

While we can use different parameters for representing C-space of a robot, its degrees of freedom (DoFs) are not changed.

Suppose a 2D robot that we can change its position, (x, y), and its orientation, θ. In this case, the number of DoFs of the robot is two. Instead of this representation, we can specify the orientation by a point, (u, v), in a unit circle, where $u^2 + v^2 = 1$, and $u = \cos\theta, v = \sin\theta$. While we use four parameters, (x, y, u, v) for the position and orientation of the robot, the number of DoFs cannot be changed and thus still is two. Intuitively speaking, the DoF of a robot does not depend on its parametrization.

3.1 *Constraints*

There can be many types of constraints on paths that we need to compute for maneuvering robots. At a high level, these constraints can be classified as the following:

- **Local constraint.** Local constraints are related to ones that require only local information to satisfy them. The most common one is to avoid collision that requires to check whether we have collisions between the robot and its local neighboring region.
- **Global constraint.** Computing a path with a minimum distance requires global information along the path and explores various regions of workspace and C-space. Satisfying such a global constraint tends to require higher computational cost than that of local constraint.
- **Holonomic constraint.** Holonomic constraint has the following relationship between the motion of a robot, $(q_1, q_2, \cdots)$, and the time t:

$$f(q_1, q_2, \cdots, t) = 0. \tag{3.1}$$

An example is a particle moving along a surface or a simple pendulum.

- **Non-holonomic constraint.** Non-holonomic constraints have the following relationship:

$$f(q_1, q_2, \cdots, q_1', q_2', \cdots, t) = 0, \tag{3.2}$$

where $q_1', q_2, \cdots$ are the first derivative, i.e., velocity, along $q_1, q_2, \cdots$. An example of this non-holonomic constraint is related to describing the motion of a kinematic car model that is restricted due to its kinematics, i.e., shape, of the car without considering any force on the car (Ch. 3.1.1). Handling non-holonomic constraints requires us to deal with the first derivatives of coordinates of robots. To do that, we commonly extend our C-space to include positions and velocity of the robot, resulting in a higher space and thus higher complexity.

- **Dynamic constraint.** Dynamic constraints has the following form:

$$f(q_1, \cdots, q_1', \cdots, q_1'', \cdots t) = 0, \tag{3.3}$$

where $q_1'', \cdots$ are the second derivative to $q_1, \cdots$ of the robot. These dynamic constraints can be reduced to the non-holonomic ones by using state space (Ch. **??** YOON: fix). An example of dynamic constraints is planning quadruped robots while considering various forces (e.g., gravity and repulsion force from each foot of the robot).

3.1.1 *Kinematic Car Model*

In this section, we study a simple car model considering a basic kinematic structure of the car model. Fig. 3.2 shows such a simple 2D model. The C-space of the car model is (x, y, θ, v), where x, y, θ are 2D positions of the reference point of the car and its orientation angle. v is the velocity of the car. In this case, we have two control parameters v and a steering angle ϕ.

Our goal is to derive its motion equation. Given its orientation angle θ, we have the following equations:

$$\begin{aligned} \tan(\theta) = \frac{\sin(\theta)}{\cos(\theta)} = \frac{dy}{dx}, \\ \sin(\theta)dx - \cos(\theta)dy = 0, \end{aligned} \tag{3.4}$$

where (dx, dy) is the differential changes along X and Y directions. This is an implicit equation that belongs to the category of the non-holonomic constraint.

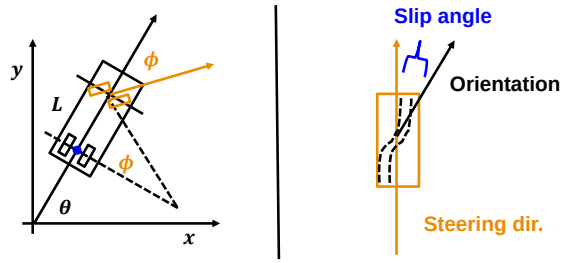

Figure 3.2: The left figure shows a simple, kinematic car model. The right image illustrates the slip angle made by the angle between the steering direction and the heading direction of the car.

While the implicit equation is useful, it does not directly predict how the configuration of the car changes depending on our input changes v and ϕ. The differential units for x and y are computed as the following:

$$x' = \frac{dx}{dt} = v\cos(\theta),$$
$$y' = \frac{dy}{dt} = v\sin(\theta). \tag{3.5}$$

Let's consider the angular velocity, the velocity change rate per unit time, θ'. According to the definition of the solid angle, we have the following equation:

$$\frac{w}{r} = \theta \rightarrow \frac{dw}{r} = d\theta, \tag{3.6}$$
$$\tan\phi = \frac{L}{r}, \tag{3.7}$$

where w is the traveled distance of the car, and r is the distance from the rotation center of the car to the reference point of the car. By merging these two equations, we get the following equation:

$$d\theta = \frac{dw}{L}\tan\phi,$$
$$\theta' = \frac{d\theta}{dt} = \frac{v}{L}\tan\phi. \tag{3.8}$$

Note that x', y', θ' are explicit equations, where we know their differential values given a control (e.g., steering angle ϕ). We later see how they are used for motion planning algorithms (Ch. **??** YOON: Fix).

Extensions. The aforementioned simple car considers only the kinematics of the car, not skidding occurring when turning at a high speed nor the slip angle. For example, when we do not consider the slip angle, we can have understeering, the car does not steer much as we requested, or oversteering. As a result, the simple model works well only with low speed. When we use this simple model for such

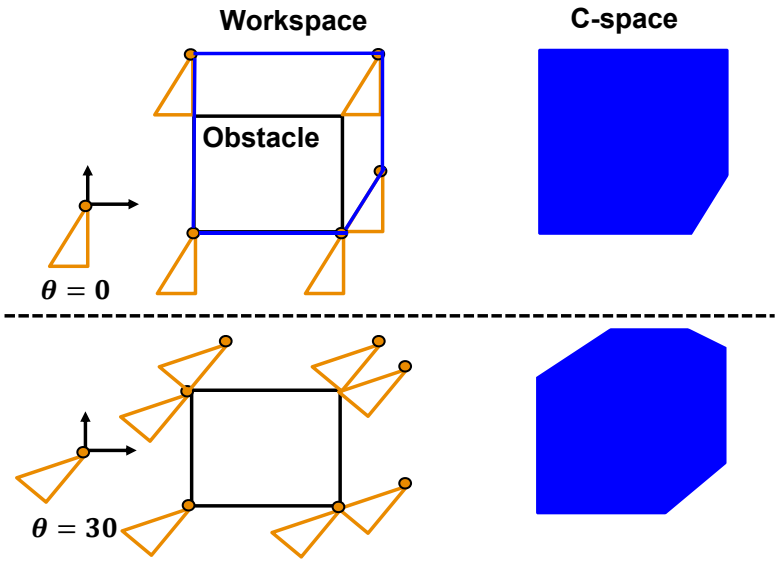

Figure 3.3: At the top row, we show a triangular robot (orange color) whose orientation is set to 0 and reference point is shown red dot filled with the orange color. In the middle, we show a few samples of robot positions that touch with the rectangular obstacle (black color). Blue lines are the trajectory of the reference point; we basically roll the robot while touching the obstacle. In the right, we show the C-obstacle space (blue region) in C-space. At the bottom row, we show similar information, but with a different orientation of the robot.

high speed cases, our low-level controller find that there are large gaps between a planned path and an actual trajectory made by the car, which may cause collisions. A car model considering the slip angle [1] and other complex models are also available.

[1] R. Pepy, A. Lambert, and H. Mounier. Path planning using a dynamic vehicle model. In *Int. Conf. on Information Communication Technologies*, 2006

3.2 *Construction of C-Obstacle*

So far, we discussed C-space with its free space and obstacle space. One can think of that if we can exactly construct the boundary of the obstacle space, i.e., C-Obstacle, we can conduct the collision detection efficiently and perform planning efficiently. This line of research was conduced early in the field of motion planning, and later it turned out that exactly computing free or obstacle spaces requires an exponential time complexity as a function of dimensions of C-space. While exactly computing such spaces is not widely attempted in these days, some of these discussions are useful for understanding the computational challenges of motion planning.

We first discuss how to construct the C-obstacle in a simple setting with a 2D obstacle and 2D robot, as shown in Fig. 3.3. In this example, we use a triangular shape robot (orange) and rectangular obstacle (black). We assume that we can translate and rotate the robot. As a result, its C-space is 3D.

To simplify our discussion, we first suppose that the robot's orientation is fixed to 0, resulting in its C-space 2D, and the reference point of the robot is defined at one of its vertices. To see how the C-obstacle space looks like, we need to see cases of robots that touch

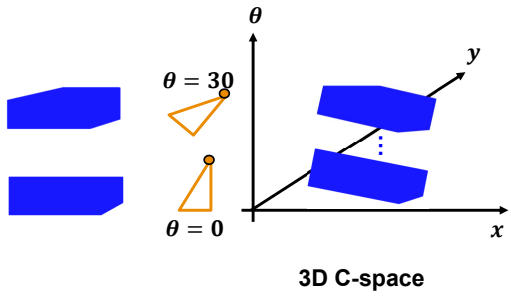

Figure 3.4: We visualize the 3D C-space how the space changes as a function of the orientation of the robot shown in Fig. 3.3.

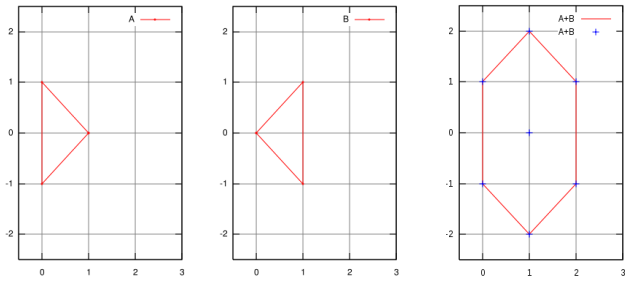

Figure 3.5: The rightmost polygon is $A \oplus B$ given two triangles of A and B. The image is adopted from wiki.

or collide with the C-obstacle. One simple method is to simply roll move the robot while it maintains the contact with the obstacle. At that case, the trajectory of the reference point of the robot (shown in blue line) is the boundary of the C-obstacle space. At the right side of Fig. 3.3, we show the C-obstacle space (blue region) at the chosen orientation angle.

When we change the orientation, e.g., 30 degree, as shown in the bottom side of Fig. 3.3, we have a different shape of C-obstacle space within the C-space. Now, when we consider the whole C-space consisting of 3D space including the orientation of the robot, we can easily that the shape of C-obstacle is very complicated.

It has been identified that we can construct such C-obstacle space out of a convex polygonal robot and a convex polygonal obstacle by utilizing Minkowski sum. The Minkowski sum between two sets of vectors, P and Q, is defined as the following:

$$P \oplus Q = \{p + q | p \in P, q \in Q\}. \tag{3.9}$$

For example, let P and Q two sets of vertices of two triangles: e.g., $P = \{(0,0),(1,1),(1,-1)\}$ and $Q = \{(0,1),(0,-1),(1,0)\}$. Then, $P \oplus Q = \{(0,1),(0,-1),(1,0),(1,2),(1,0),(2,1),(1,0),(1,-2),(2,-1),\}$. If we interpret elements of $P \oplus Q$ as vertices of a polygon, the Minkowski sum of two triangles, i.e., polygons, is shown in Fig. 3.5. From this example, we can see that when P and Q have n and m vertices of two polygons, $P \oplus Q$ has $n + m$ vertices, which are computed by summing those vertices of P and Q.

Similarly, we can define the Minkowski difference between two sets as the following:

$$\begin{aligned} P \ominus Q &= \{p - q | p \in P, q \in Q\} \\ &= P \oplus -Q. \end{aligned} \tag{3.10}$$

Now, we would like to show that C-obstacle is computed by the Minkowski sum or difference. Fig. 3.6 shows the boundary of C-obstacle given the robot M and the obstacle P. In this example, we place the robot in a location where it has a collision with the obstacle. The translation amount for the robot in this 2D case is the particular position in the C-space, especially C-obstacle, of the robot. That amount is computed by adding the obstacle point p and $-m$, where m is the vector to the collision point from the robot origin. This simple example illustrates that the C-obstacle is computed by $P \ominus M$.

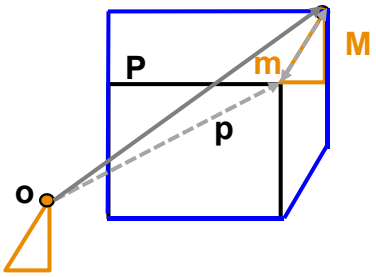

Figure 3.6: This show that C-obstacle is computed by the Minkowski difference between the obstcle P and the robot M.

C-obstacle is computed by Minkowski difference between the obstacles and the robot.

We have came to know that we can compute the boundary of C-obstacle by using the Minkowski difference. Unfortunately, its time complexity is prohibitive high and thus we cannot use for real-time robotics. For non-convex objects that are common in practice, the Minkowski sum has the time complexity of $O(n^6)$, where n is the number of features in each object, e.g., vertices. Fig. 3.7 shows an example of computing the Minkowski sum between two non-convex objects; the image is excerpted from [2].

[2] G. Varadhan and D. Manocha. Accurate minkowski sum approximation of polyhedral models. In *Pacific Conference on Computer Graphics and Applications*, pages 392–401, 2004

3.3 *Paths*

Motion planners can compute paths or trajectories. Paths include geometric paths like line segments or a continuous curve that connects two points without having collisions. On the other hand, the trajectory is a path parametrized path, and thus is associated with various concepts like velocity.

While exploring various paths, it is important to find a path that is novel to already found ones. In this regard, the homotopy concept is important. We say that two paths, τ and τ', with the same end points are homotpic, if one can be continuously deformed into the other:

$$h : U \times [0,1] \to V, \tag{3.11}$$

where $h(s,0) = \tau(s)$ and $h(s,1) = \tau'(s)$. A homotopic class of paths contains all the paths that are homotopic to another. Later, we will discuss optimal planners, and these optimal planners can explore different homotopic classes of paths.

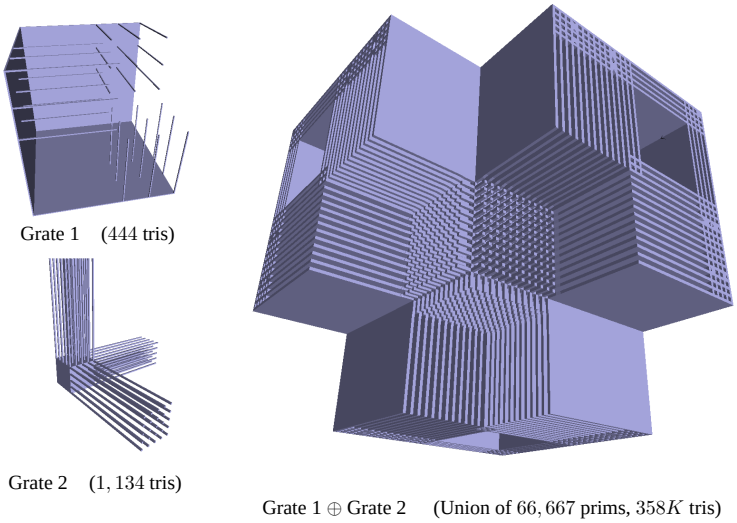

Figure 3.7: This shows an example of performing the Minkowski sum between two non-convex objects. The image is excerpted from the work of Varadhan et al.