Lecture 18
Kinematic Chains

Instructor: Jingjin Yu
Outline

What are kinematic chains?
C-space for kinematic chains
Degrees of freedom
Forward and inverse kinematics
The Jacobian
Planning for kinematic chains
Kinematic Chains

First, it is a **chain**
- 2D can be revolute or prismatic
- In 3D, many possibilities

Then, it only considers **kinematics**
- Focusing on the geometric aspects of motion
  - Is the motion feasible geometrically?
- In particular, it does not consider masses of the chains nor forces acting on them
  - Sometimes kinematics is not enough, e.g., balancing an inverted pendulum (many videos, e.g., [https://www.youtube.com/watch?v=cyN-CRNrb3E](https://www.youtube.com/watch?v=cyN-CRNrb3E))

- But, it works well when joints can be controlled
- Unless specified, we assume the chain is NOT closed

Image source: Planning Algorithms
Examples

Kinematic chains can be used to model many mechanical systems, e.g.

⇒ Industrial manipulators
  ⇒ The tip can be a hand, often called end effectors
⇒ Legged robots
⇒ Humanoid robots
  ⇒ Or for that matter, humans

A hexapod robot

Bigdog

NASA Valkerie

https://www.youtube.com/watch?v=M8YjvHYbZ9w
https://www.youtube.com/watch?v=_luhn7TLfWU
The C-Space of Kinematic Chains (I)

What is the configuration space of a kinematic chain?

- Let’s look at some simple 2D examples
  - Revolute joint: $S^1$
  - Prismatic joint: $\mathbb{R}$

- Now, what if we combine these?
  - Yes, we get $\mathbb{R} \times S^1$
  - In general, for $k$ joints with individual configuration space $C_1, \ldots, C_k$, the overall configuration space is the Cartesian product: $C = C_1 \times C_2 \times \cdots \times C_k$

- For $k$ revolute joints, that is $S^1 \times \cdots \times S^1 = T^k$
  - $k$ of these

- This is also known as $k$-dimensional torus

- For a spherical joint and a revolute joint, that is $SO(3) \times S^1 \neq T^4$

Image source: Planning Algorithms, Wolfram Mathworld
The C-Space of Kinematic Chains (II)

In practice, part of the C-space (manifold) is unreachable

⇒ E.g., a real ball joint cannot reach all of $SO(3)$
⇒ E.g., a robotic arm generally cannot do 360 degrees continuously

Closed kinematic chains are the most tricky ones

⇒ Obviously, all links can move somewhat
⇒ But the C-space is lower dimensional

Image source: Planning Algorithms, Mathworks, Kuka
Determining the Degrees of Freedom (DOFs)

Deciding the DOF

⇒ 2D kinematic chains
  ⇒ Base link is 3D ($\mathbb{R}^2 \times S^1$)
  ⇒ If fixed, then often 1D
  ⇒ Adding joints generally adds one more dimension

⇒ 3D kinematic chains
  ⇒ Base link is 6D ($\mathbb{R}^3 \times SO(3)$)
  ⇒ If fixed, depending on the joint
  ⇒ Then add the DOF of each additional joint

⇒ Closed kinematic chains
  ⇒ We have a formula!
  ⇒ $N$: 6 for 3D, 3 for 2D
  ⇒ $k$: # of links (including the ground link)
  ⇒ $n$: the number of joints
  ⇒ $f_i$: DOF of the joint
  ⇒ Examples
    ⇒ 2D, 3 links
    ⇒ 2D, 4 links
    ⇒ 2D, 6 links

$$DOF = N(k - 1) - \sum_{i=1}^{n} (N - f_i) = N(k - n - 1) + \sum_{i=1}^{n} f_i$$
Forward Kinematics

Forward kinematics: given the joint angles

⇒ Where is the end effector?
⇒ Or for that matter, where is an arbitrary fixed point on the chain?

A 2D arm with two revolute joints

⇒ Do it link by link

\[
(x', y') = (x_0 + \ell_1 \cos \alpha_1, y_0 + \ell_1 \sin \alpha_1)
\]

\[
(x, y) = (x' + \ell_2 \cos(\alpha_1 + \alpha_2), y' + \ell_2 \sin(\alpha_1 + \alpha_2))
\]

⇒ Put it together

\[
\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \ell_1 \begin{bmatrix} \cos \alpha_1 \\ \sin \alpha_1 \end{bmatrix} + \ell_2 \begin{bmatrix} \cos(\alpha_1 + \alpha_2) \\ \sin(\alpha_1 + \alpha_2) \end{bmatrix}
\]

The methods, which is iterative, applies to 3D as well
Inverse Kinematics

Inverse kinematics: finding joint angles from end effector position

⇒ I.e., given \( (x, y) \) (assume \( (x_0, y_0) = 0 \)), determine \( \alpha_1, \alpha_2 \)

⇒ Note that the solution is generally non-unique

⇒ More on this later...

A 2D arm with two revolute joints

⇒ First, compute \( \alpha_2 \) via

\[
x^2 + y^2 = \ell_1^2 + \ell_2^2 - 2\ell_1\ell_2 \cos\left(\frac{\pi}{2} - \alpha_2\right)
\]

⇒ Then, compute \( \alpha_1 \)

\[
\alpha_1 = \arctan\frac{y}{x} - \cos^{-1}\left[\frac{\ell_1^2 + x^2 + y^2 - \ell_2^2}{2\ell_1\sqrt{x^2 + y^2}}\right]
\]
The Jacobian

Forward kinematics is a map \( x = f(q) \) from C-space to workspace

\[
x = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + l_1 \cos \alpha_1 + l_2 \cos(\alpha_1 + \alpha_2) \\ y_0 + l_1 \sin \alpha_1 + l_2 \sin(\alpha_1 + \alpha_2) \end{bmatrix} = f(q)
\]

One can compute a relationship between the velocities via

\[
\dot{x} = \frac{\partial f}{\partial q} \dot{q} = J(q)\dot{q}
\]

\( J(q) \) is commonly known as the Jacobian

In this case

\[
J(q) = \begin{bmatrix} -l_1 \sin \alpha_1 - l_2 \sin(\alpha_1 + \alpha_2) & -l_2 \sin(\alpha_1 + \alpha_2) \\ l_1 \cos \alpha_1 + l_2 \cos(\alpha_1 + \alpha_2) & l_2 \cos(\alpha_1 + \alpha_2) \end{bmatrix}
\]
Planning for Kinematic Chains (I)

In practice, one cares about end effector (workspace) configuration $\mathbf{x}$

- This is usually a lower dimensional space
- Often, DOF of chain is larger than DOF of end effectors
- E.g., 3 links
- E.g., human skeleton has 70 or so DOF

In general, kinematic chains are (highly) **redundant**

- This is good
  - Help overcome joint angle limits (self collision)
- This is also bad
  - Multiple solutions
  - In fact, generally there are infinite number of solutions
  - Difficult to pick and choose
  - This is known as the null space of a Jacobian

Somehow, need to kill some of the dimensions
Planning for Kinematic Chains (II)

Goal: move end effector by moving the joints

- One can sample (e.g., RRT, PRM) $\mathbf{x}$ and compute how to reach them
- Need to relate $\Delta \mathbf{x}$ and $\Delta \mathbf{q}$

- E.g., $\dot{\mathbf{q}} = ? \dot{\mathbf{x}}$ (recall we have $\dot{\mathbf{x}} = \frac{\partial f}{\partial \mathbf{q}} \dot{\mathbf{q}} = J(\mathbf{q}) \dot{\mathbf{q}}$, but not the other way)

Some Jacobian based methods

- Jacobian transpose
- Pseudo inverse
- Extended Jacobian method

- Basically, impose additional optimization criteria to kill the null space
  - We will highlight how Jacobian transpose works

On can also use inverse kinematics for doing the same

- Compute the joint angles given a new end effector position
- Move to these new joint angles
  - May need to replan
Planning for Kinematic Chains (III)

Jacobian transpose has the form

$$\Delta q = \alpha J^T(q)\Delta x$$

The principle

- Try to find $\Delta q$ that decreases $\Delta x$ the most
- Essentially projecting $\Delta x$ over $q'$s dimensions

Derivation

- Seek to minimize the function $F(q) = \frac{1}{2} (x - x_G)^T (x - x_G)$
- Do this via gradient descent ($\alpha$ is a step size)

$$\Delta q = -\alpha \left( \frac{\partial F}{\partial q} \right)^T = -\alpha \left( (x - x_G)^T \frac{\partial x}{\partial q} \right)^T = -\alpha J^T(q) (x - x_G) = \alpha J^T(q)\Delta x$$