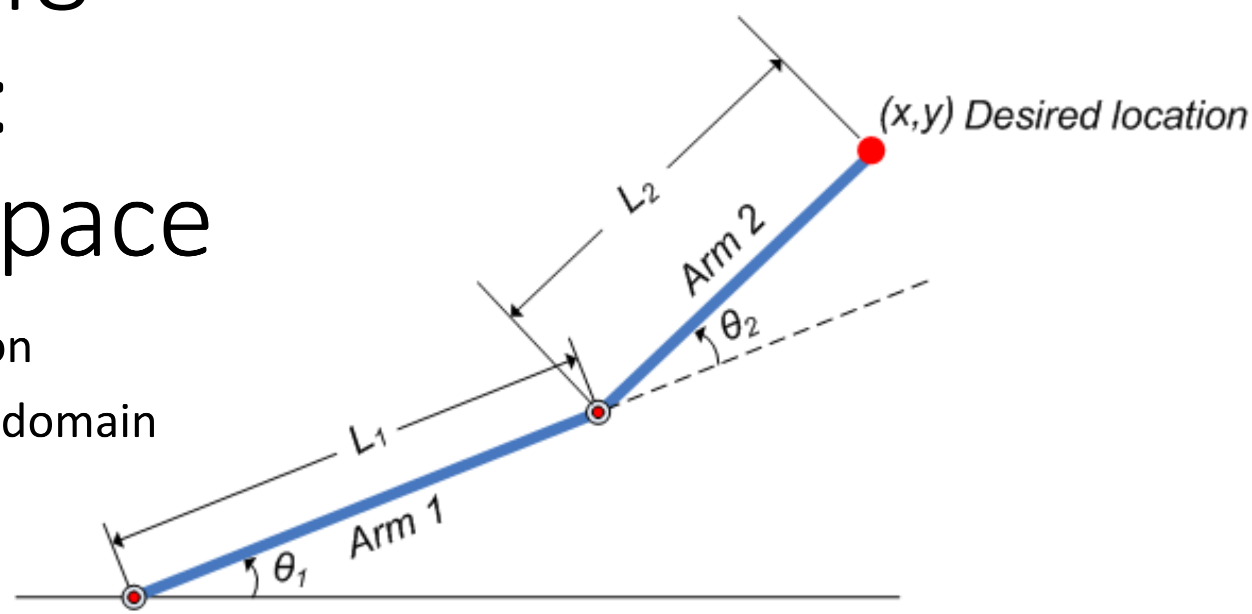
CS440/ECE448 Lecture 17: Configuration Space

Mark Hasegawa-Johnson



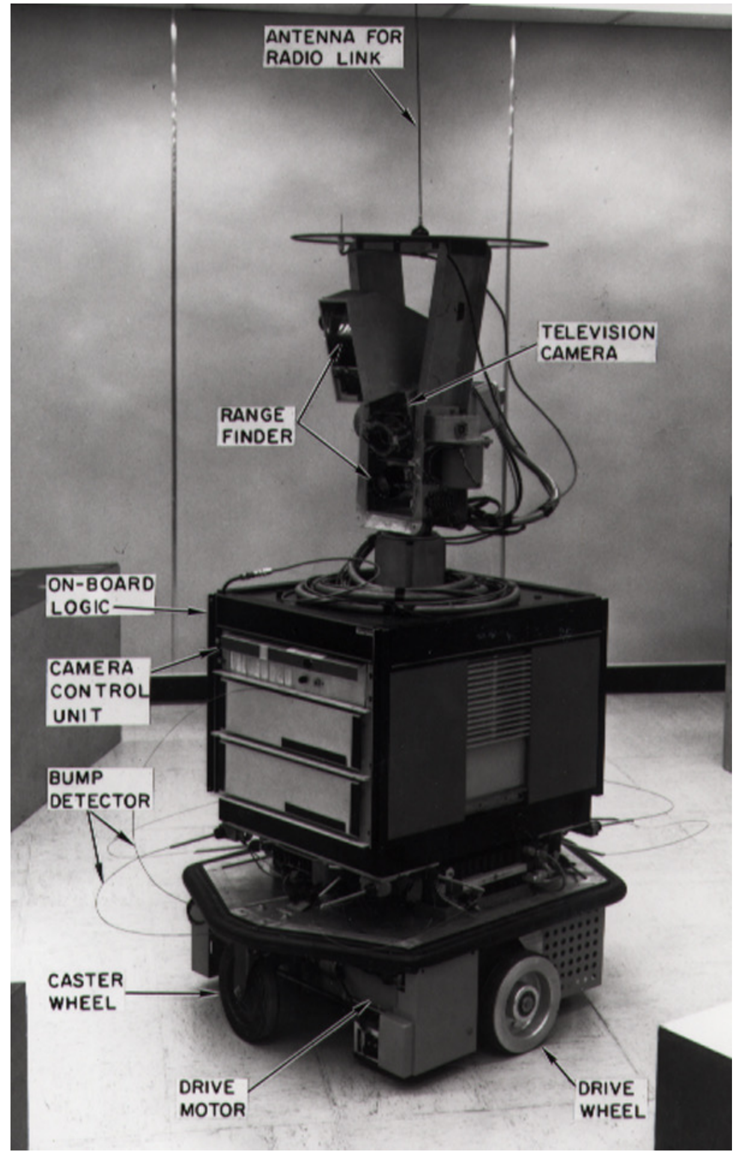
Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning: What is a good path?
- Path planning: How do we find the shortest path?
  - Discretized $\mathcal{C}$-space
  - Visibility graph
  - Rapid random trees

What is a “Robot”?

Example: Shaky the robot, 1972
https://en.wikipedia.org/wiki/Shakey_the_robot

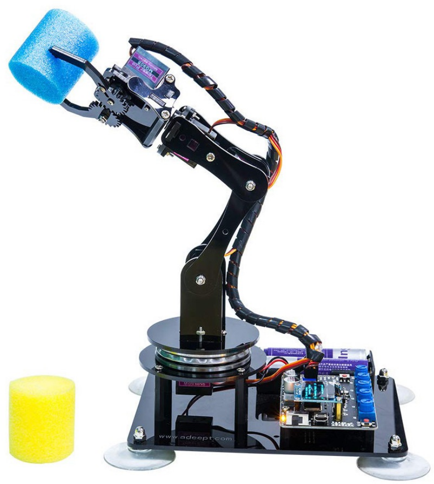
- Planning
  - Antenna for radio link
  - On-board logic
  - Camera control unit
- Perceiving
  - Range finder
  - Television camera
  - Bump detector
- Acting
  - Caster wheel
  - Drive motor
  - Drive wheel



Example: Robot Arm

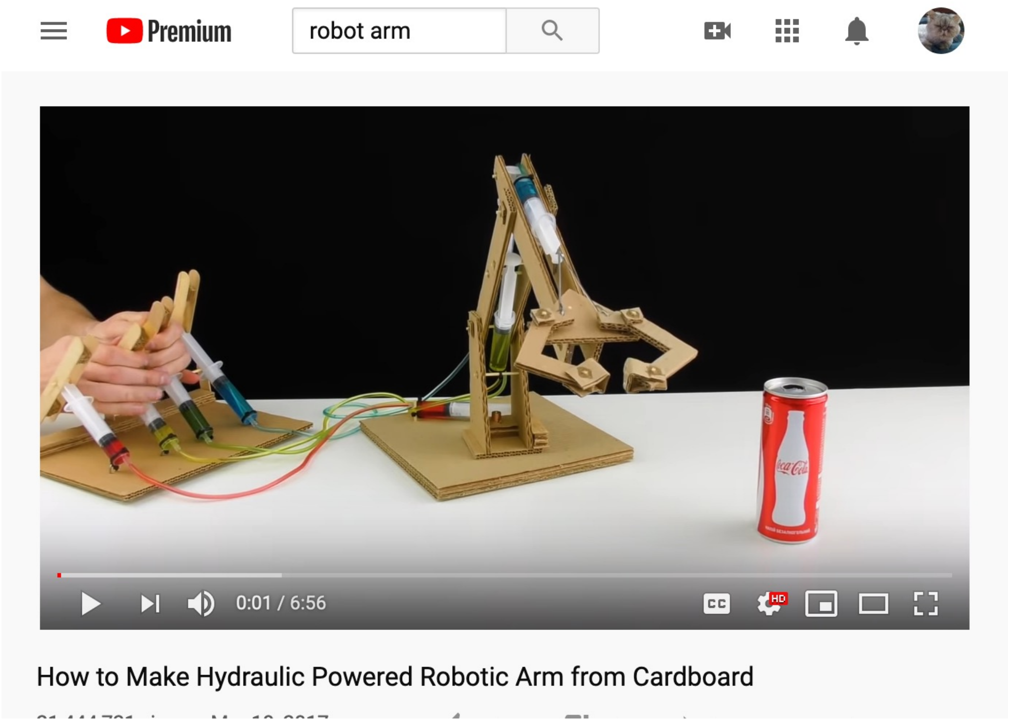
Adeept robot arm for Arduino (from Amazon)

- How does the robot arm decide when it has successfully grasped a cup?
- How does it find the shortest path for its hand?



Configuration Space Example: Robot Arm

https://www.youtube.com/watch?v=P2r9U4wkjcc



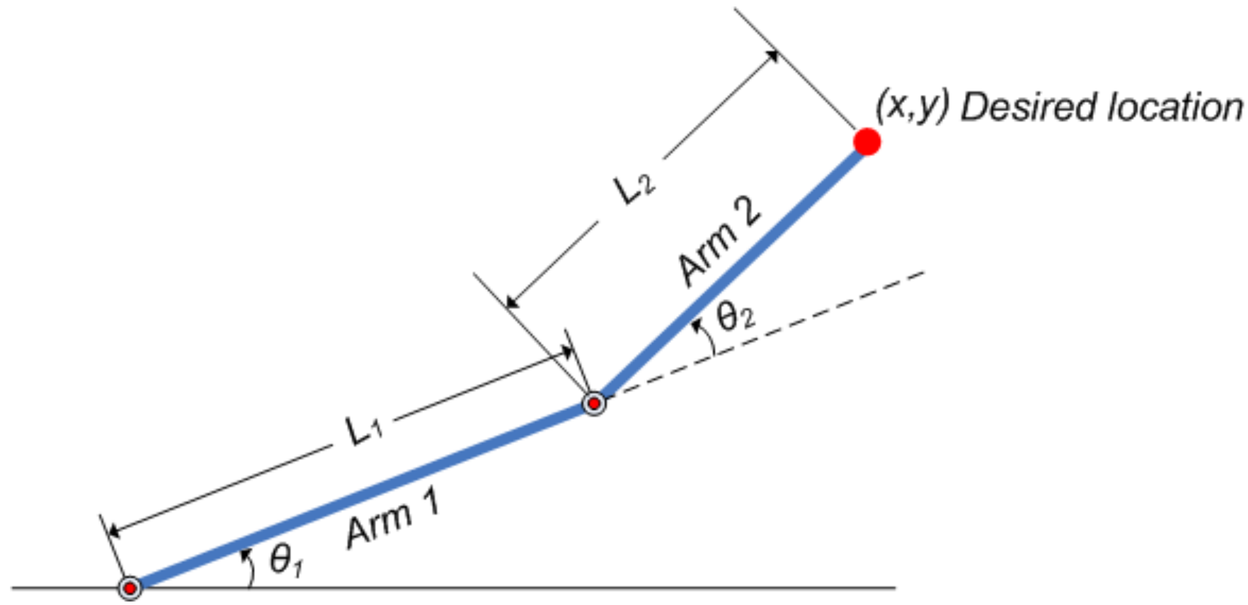
Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning: What is a good path?
- Path planning: How do we find the shortest path?
  - Discretized $\mathcal{C}$-space
  - Visibility graph
  - Rapid random trees

The Robot Arm Reaching Problem

https://www.mathworks.com/help/fuzzy/modeling-inverse-kinematics-in-a-robotic-arm.html

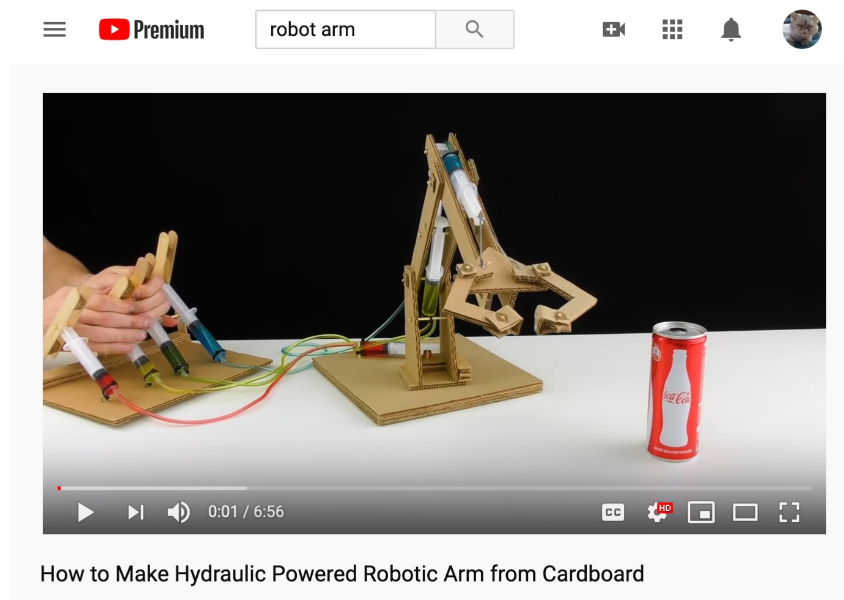
- Our goal is to reach a particular location (x,y)
- But we can't control (x,y) directly! What we actually control is (θ_1, θ_2).



Workspace vs. Configuration space

- A robot's **workspace**, $\mathcal{W}$, is the physical landscape in which it operates, $\mathcal{W} \subset \mathbb{R}^3$.
- **Configuration space**, $\mathcal{C}$, is the set of joint angles that govern the robot's shape. For example, if we have four angles to control, then $\mathcal{C} \subset \mathbb{R}^4$:

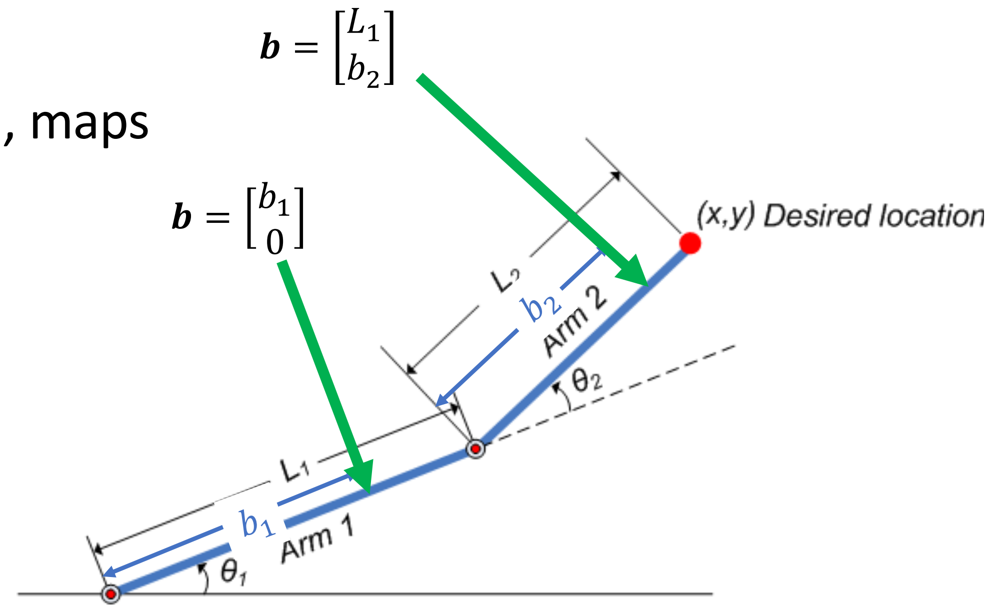
$$\boldsymbol{c} = \begin{bmatrix} \text{shoulder azimuth} \\ \text{shoulder elevation} \\ \text{elbow elevation} \\ \text{gripper opening} \end{bmatrix} \in \mathcal{C} \subset \mathbb{R}^4$$



Forward kinematics

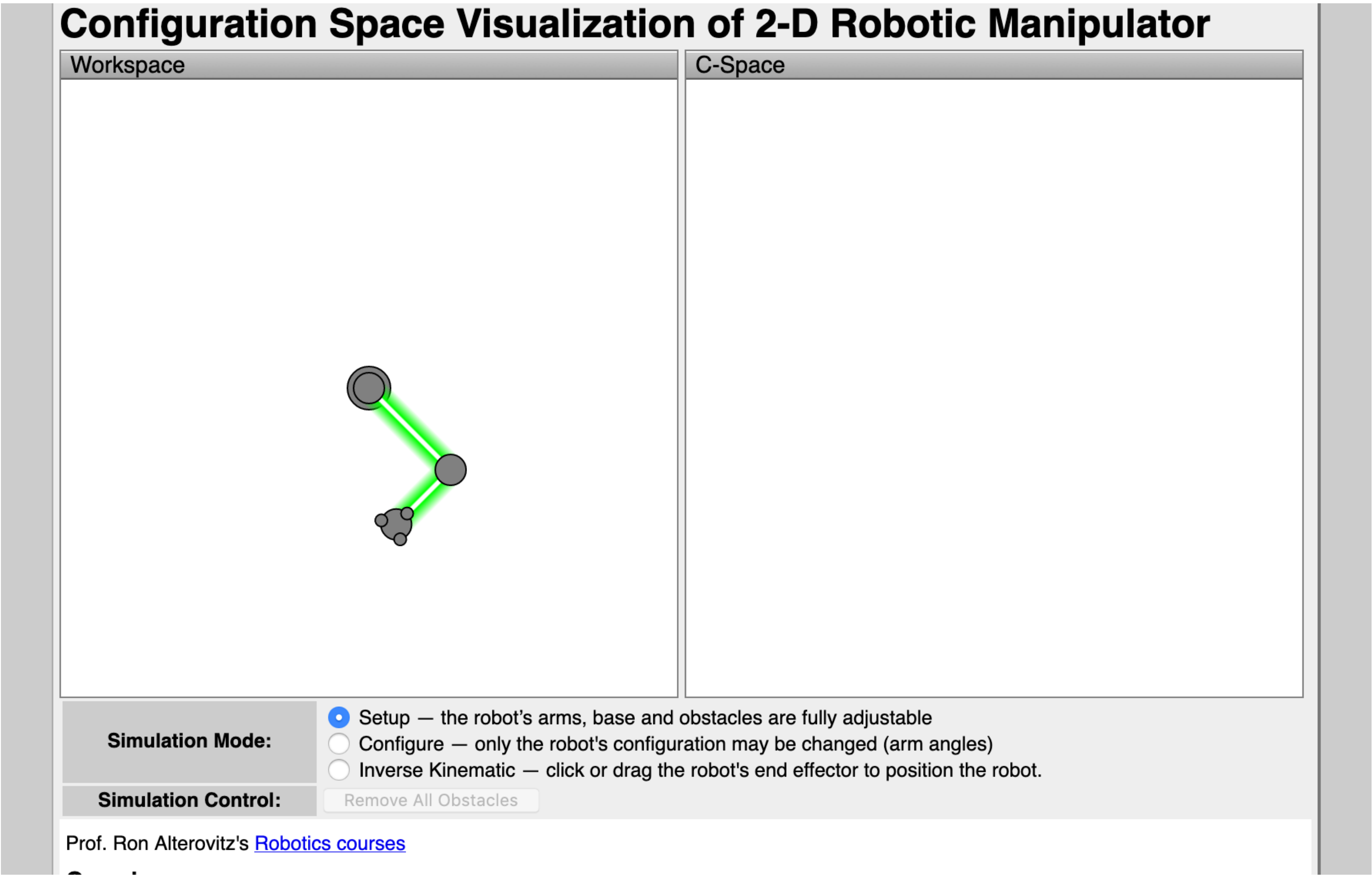
- The **forward kinematics** function, $\varphi(\boldsymbol{b}, \boldsymbol{c})$, maps (point on robot × configuration space)→(workspace).
- Forward kinematics is determined by the robot's geometry. For example:
  - $\boldsymbol{b} = [b_1, b_2]^T$, where b_1 is distance from the shoulder, b_2 is distance from the elbow.

$$\varphi(\boldsymbol{b}, \boldsymbol{c}) = \begin{cases} \begin{bmatrix} b_1 \cos\theta_1 \\ b_1 \sin\theta_1 \end{bmatrix} & b_2 = 0 \\ \begin{bmatrix} L_1 \cos\theta_1 + b_2 \cos(\theta_1 + \theta_2) \\ L_1 \sin\theta_1 + b_2 \sin(\theta_1 + \theta_2) \end{bmatrix} & b_1 = L_1 \end{cases}$$



The Robot Arm Reaching Problem

Jeff Ichnowski, University of North Carolina, https://www.cs.unc.edu/~jeffi/c-space/robot.xhtml



Quiz

Try the quiz!

Obstacles and Inverse kinematics

- Obstacles are things in the workspace, $\mathcal{W}$, that we don't want to run into.
- We want to plan a path through configuration space, C, such that we don't run into any obstacle.
- **<u>Inverse kinematics</u>**: a function that converts obstacles in the workspace, $\mathcal{W}_{\text{obs}}$, into equivalent obstacles in configuration space, $\mathcal{C}_{\text{obs}}$.

$$\mathcal{C}_{\text{obs}} = \{\boldsymbol{c}: \exists \boldsymbol{b}: \varphi(\boldsymbol{b}, \boldsymbol{c}) \in \mathcal{W}_{\text{obs}}\}$$

- Usually: exhaustively test every point in configuration space, to see if $\exists b: \varphi(\boldsymbol{b}, \boldsymbol{c}) \in \mathcal{W}_{\text{obs}}$.

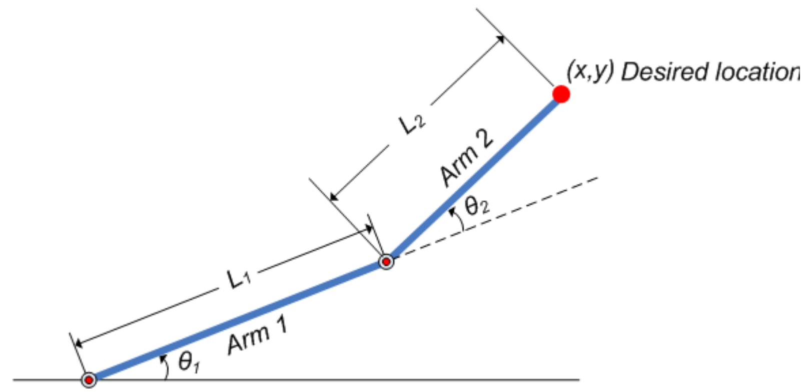
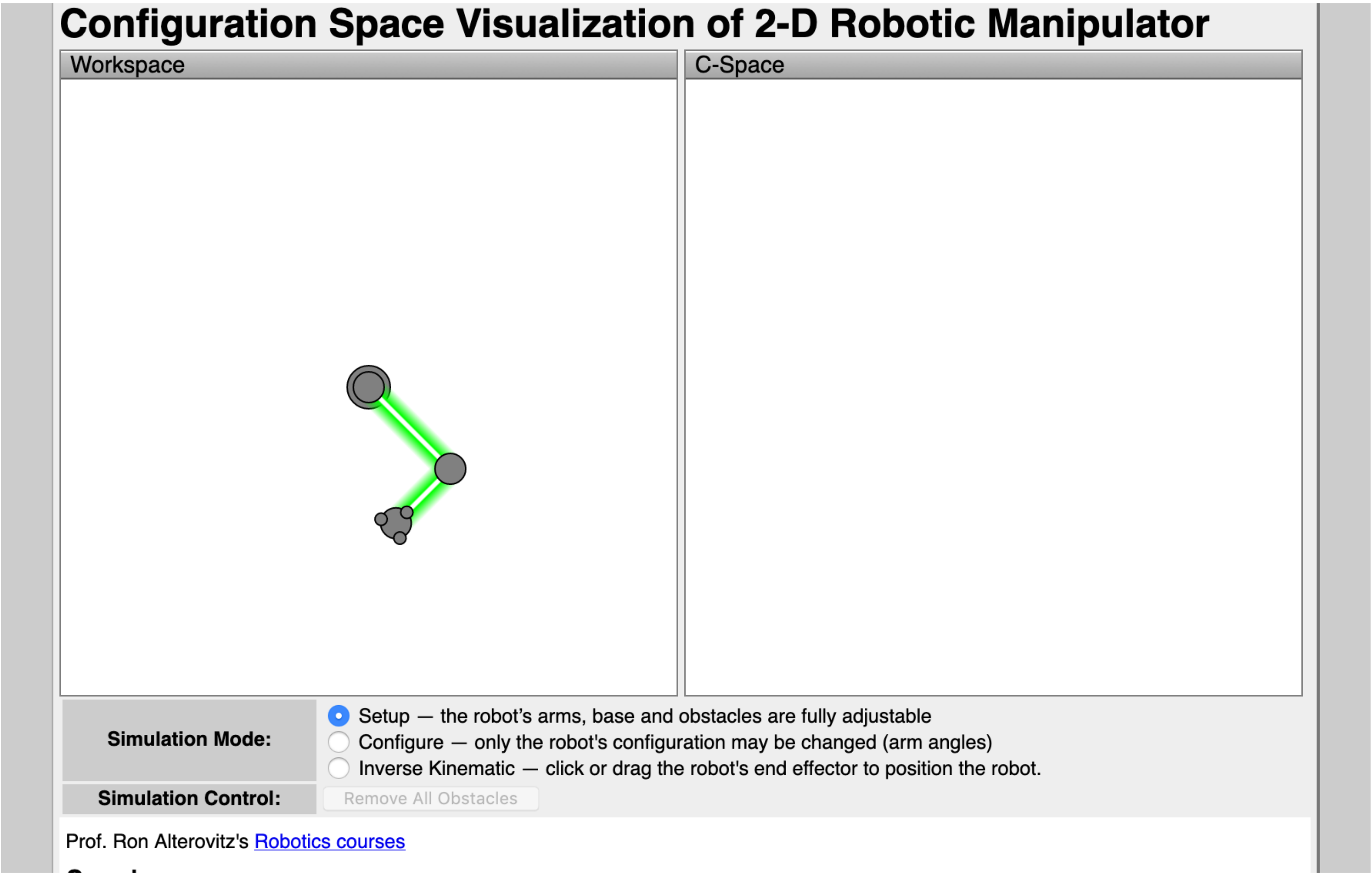


Image © https://www.mathworks.com/help/fuzzy/modeling-inverse-kinematics-in-a-robotic-arm.html

The Robot Arm Reaching Problem

Jeff Ichnowski, University of North Carolina, https://www.cs.unc.edu/~jeffi/c-space/robot.xhtml

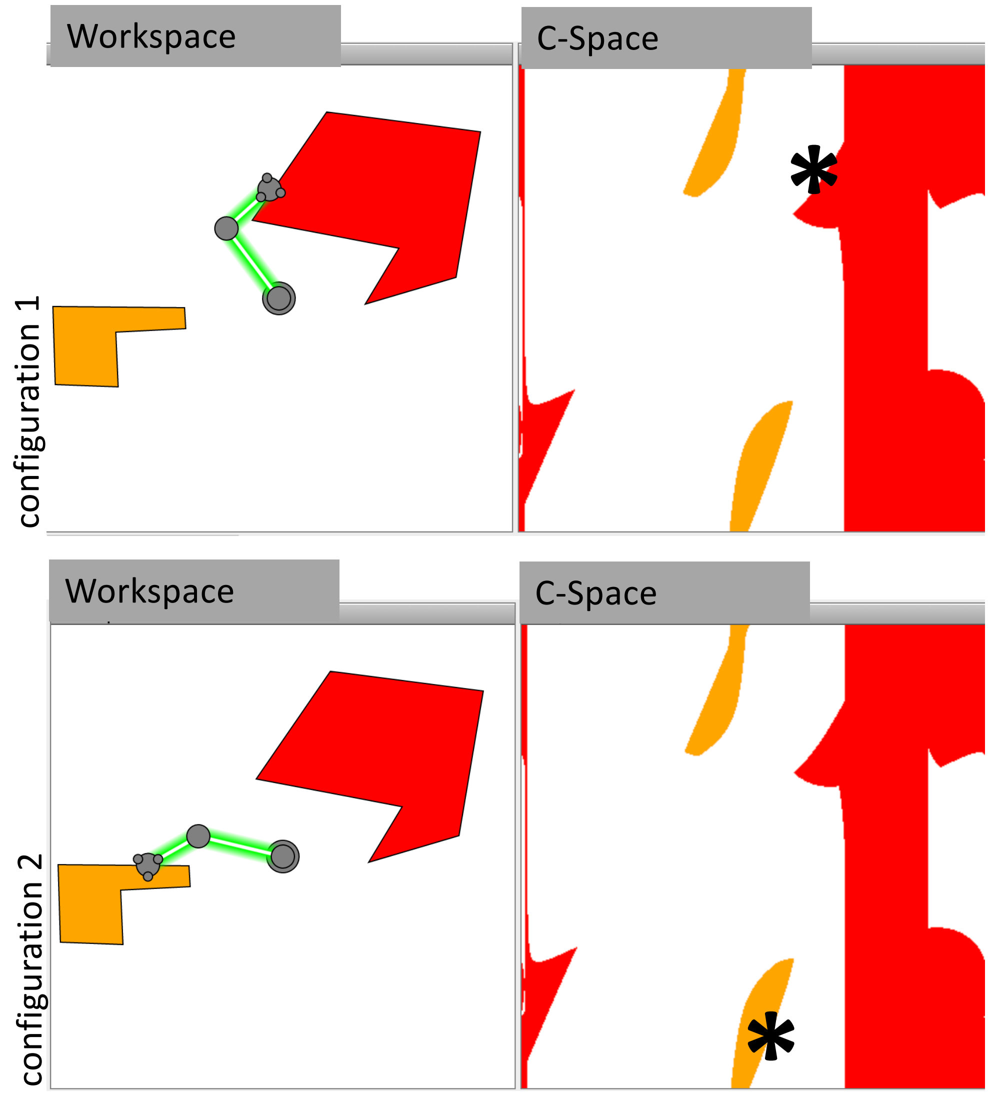


Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning: What is a good path?
- Path planning: How do we find the shortest path?
    - Discretized $\mathcal{C}$-space
    - Visibility graph
    - Rapid random trees

The planning problem

What is the best way to get from configuration 1 to configuration 2?



Criteria for a good path

- Minimum distance in $\mathcal{W}$-space
  - …means that the physical path is a straight line
  - …but who cares if the physical path is a straight line?
- Minimum distance in $\mathcal{C}$-space
  - Minimizes the total change in settings of the robot's motors
  - Smallest total energy expended!
- Keep the maximum torque within the limits of your motors
  - $\left|\frac{d^2\theta_1}{dt^2}\right| \leq max_1, \left|\frac{d^2\theta_2}{dt^2}\right| \leq max_2$
- Minimize the maximum velocity in $\mathcal{W}$-space
  - Prevents the robot from moving too quickly, which might endanger humans

Criteria for a good path

- What are some other criteria for a good path?

Criteria for a good path

We want to find a path through configuration space, $\boldsymbol{c}(t)$, that

- …minimizes one criterion (e.g., length):

$$\int f(\boldsymbol{c}(t))dt$$

- …while making sure that some other criteria (e.g., torque, velocity) remain below some maximum allowable value:

$$\begin{gathered} g_1(\boldsymbol{c}(t)) \leq G_1 \\ g_2(\boldsymbol{c}(t)) \leq G_2 \\ \vdots \end{gathered}$$

Simplest reasonable criterion: Minimum length

For this course, let's assume we want to minimize the total path length in $\mathcal{C}$-space:

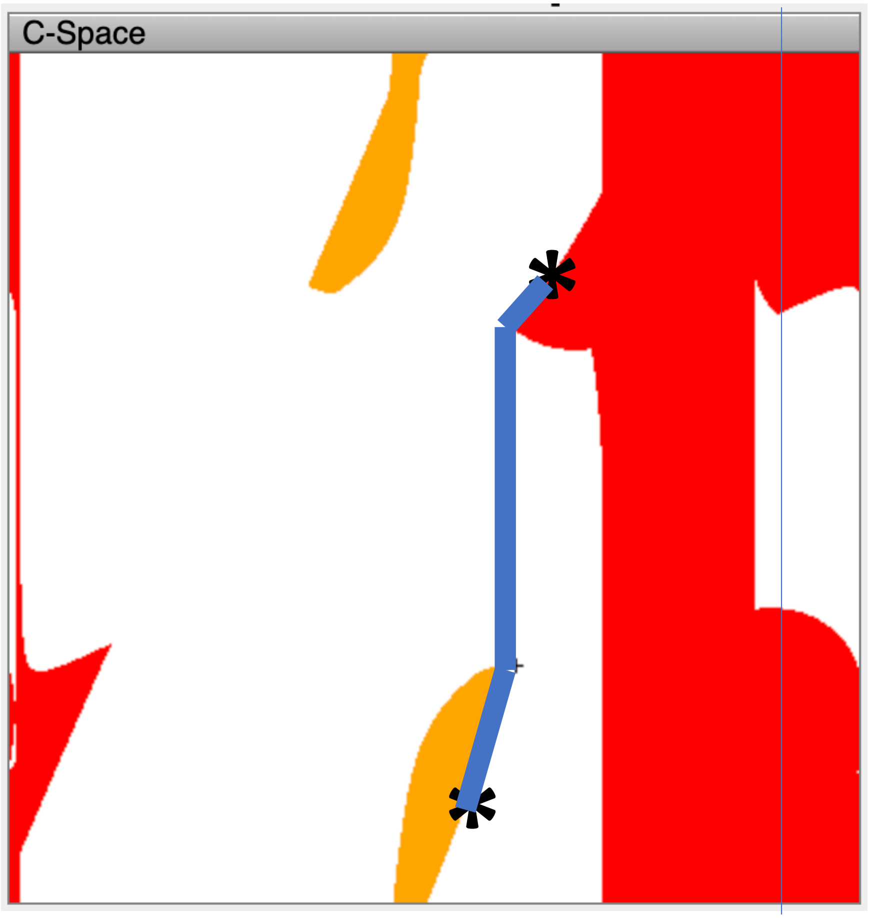
$$L = \int \sqrt{\left(\frac{\partial \theta_1}{\partial t}\right)^2 + \left(\frac{\partial \theta_2}{\partial t}\right)^2}\, dt$$

…subject to the constraint that the robot doesn't hit any obstacles:

$$\boldsymbol{c} \notin \mathcal{C}_{\text{obs}}$$

Minimum $\mathcal{C}$-space distance

The shortest path is a straight line. Therefore, we want to travel in straight lines in $\mathcal{C}$-space .



Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning: What is a good path?
- Path planning: How do we find the shortest path?
  - Discretized $\mathcal{C}$-space
  - Visibility graph
  - Rapid random trees

How do we find the shortest path?

- The problem "find the shortest path from start to finish, while avoiding obstacles" is called a **search problem**.
- If $\mathcal{C}$-space contains a discrete list of nodes, then we can search for the shortest path using Dijkstra's algorithm, which you learned in CS 225, and which we'll discuss again in the next lecture.
- … but $\mathcal{C}$-space is usually continuous, not discrete! How can we discretize it?

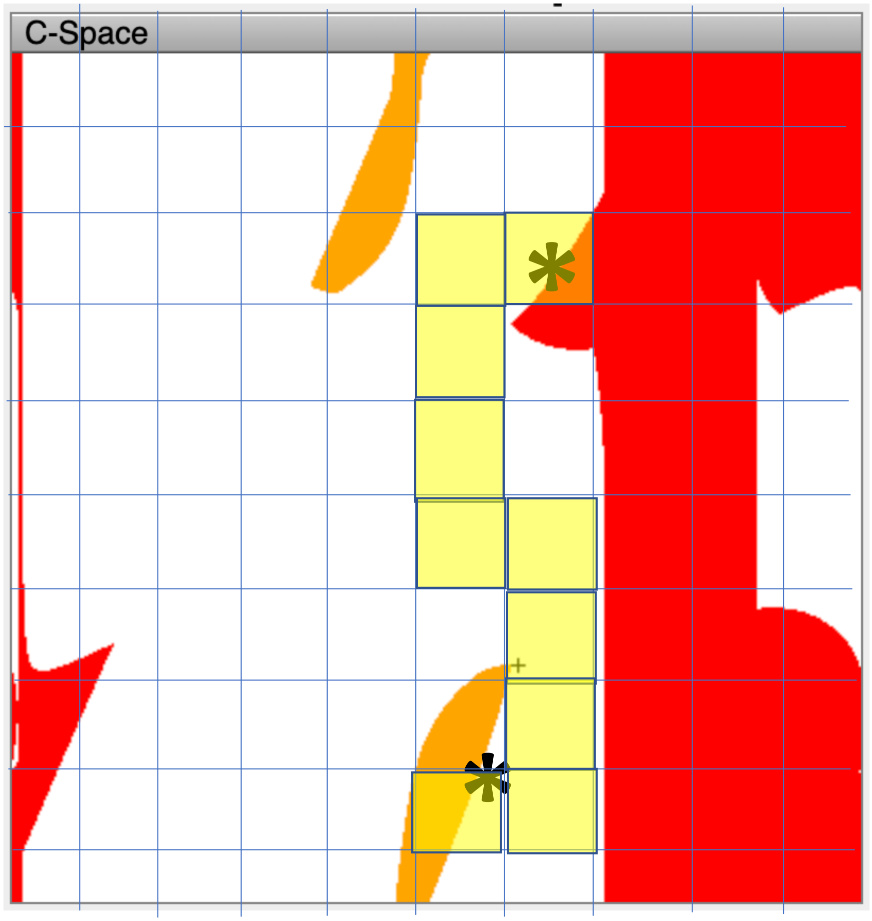
How to discretize $\mathcal{C}$-space: Three options

- Rectangular discretization
- Visibility graph
- Rapid random trees

Rectangular discretization

The most straightforward option is to just bin $\mathcal{C}$-space into rectangular bins.

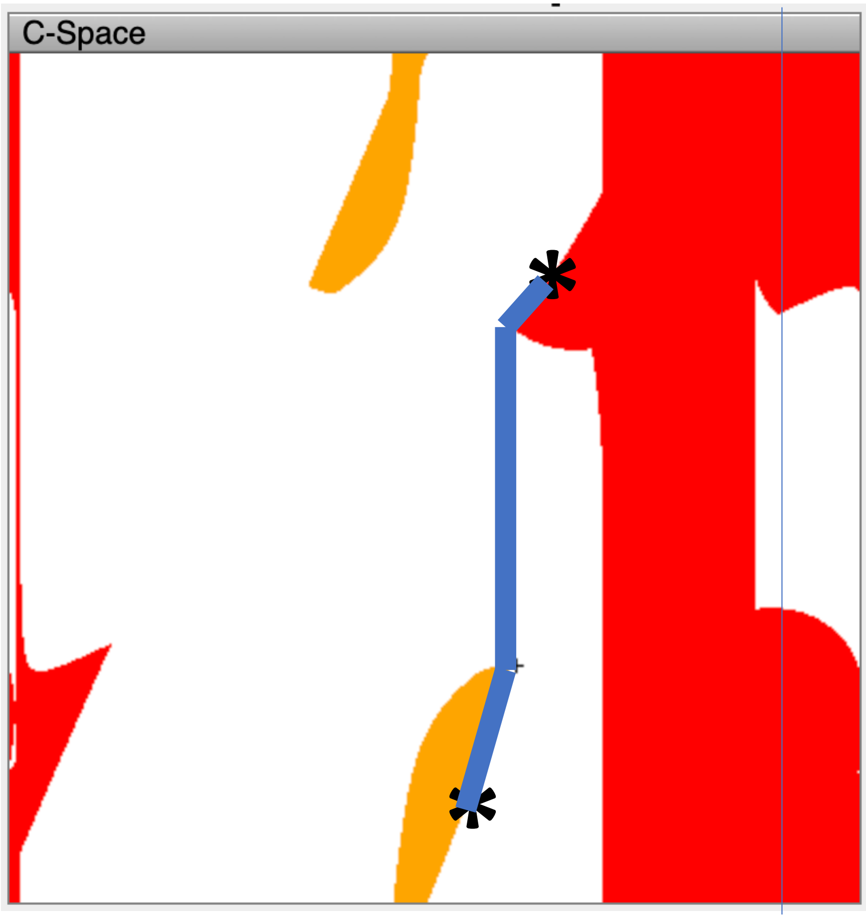
- Advantages:
  - Easy
- Disadvantages:
  - You might miss the shortest path
  - Your discretization might be so coarse that there is no path!



Visibility Graph

Suppose all the obstacles are polygons in C-space. Then the shortest path is guaranteed to be:

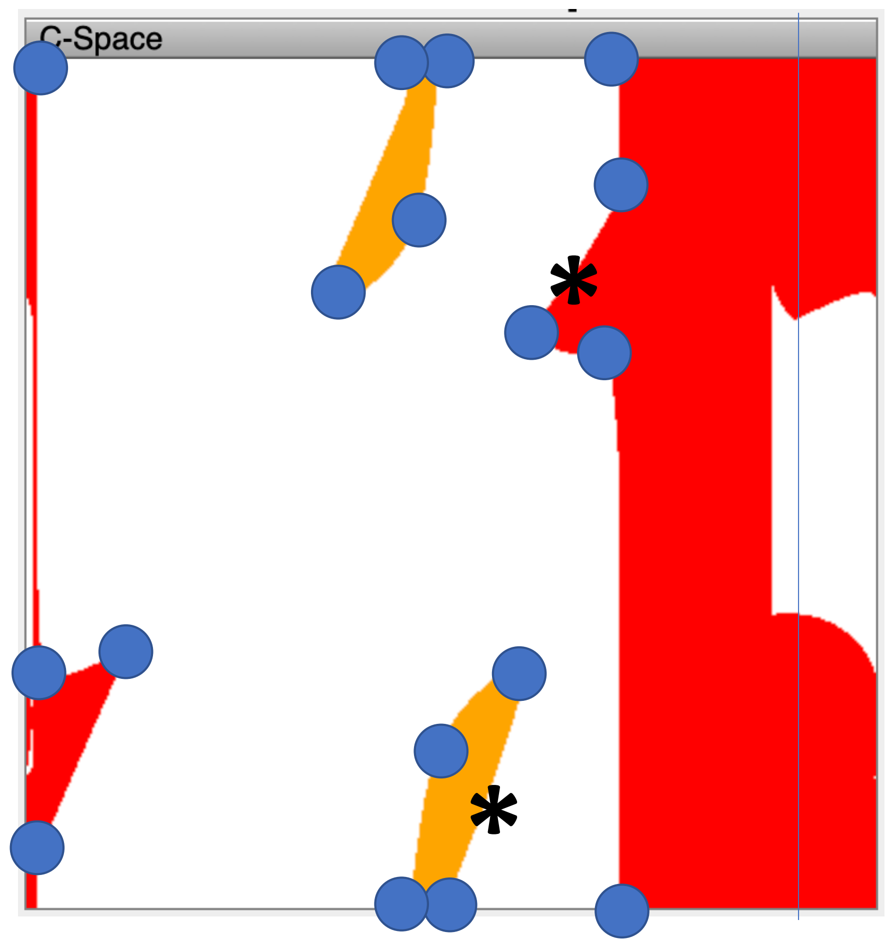
- From starting point to the corner of an obstacle, then…
- …from that corner to another corner, then….
- …from the corner of an obstacle to the goal.



Visibility Graph

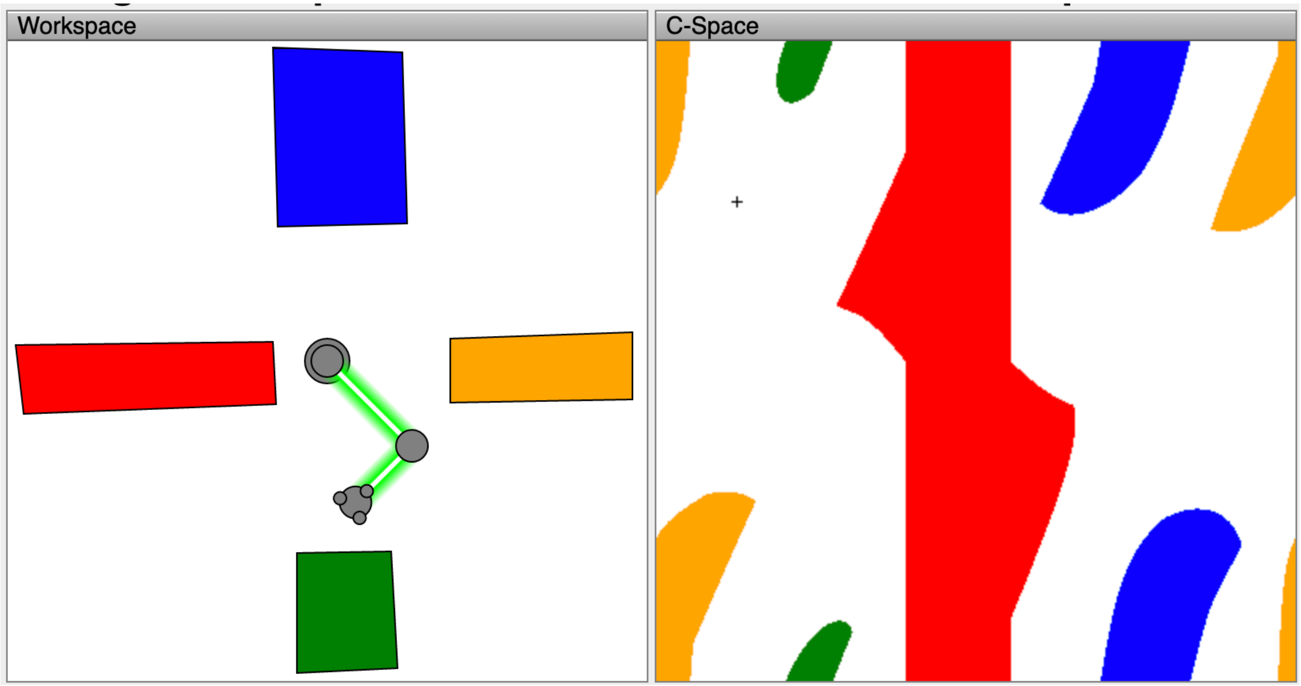
A visibility graph discretizes $\mathcal{C}$-space by finding the corners of all the polygons.

If all obstacles are polygons, then the shortest path is guaranteed to be a path among these discrete points.



Limitations

The limitation of a visibility graph: it only works if the obstacles are polygons in C-space. If obstacles are arcs, they don't have corners.

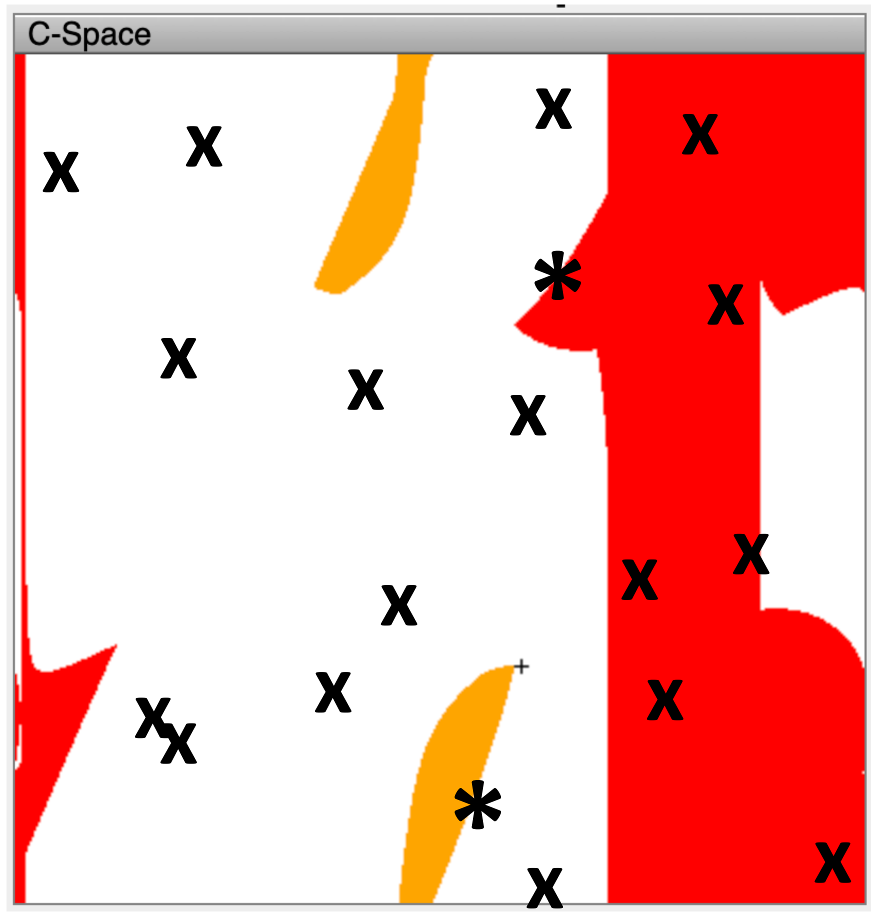


Rapid Random Trees

The "rapid random trees" algorithm (RRT) discretizes $\mathcal{C}$-space in a counter-intuitive way:

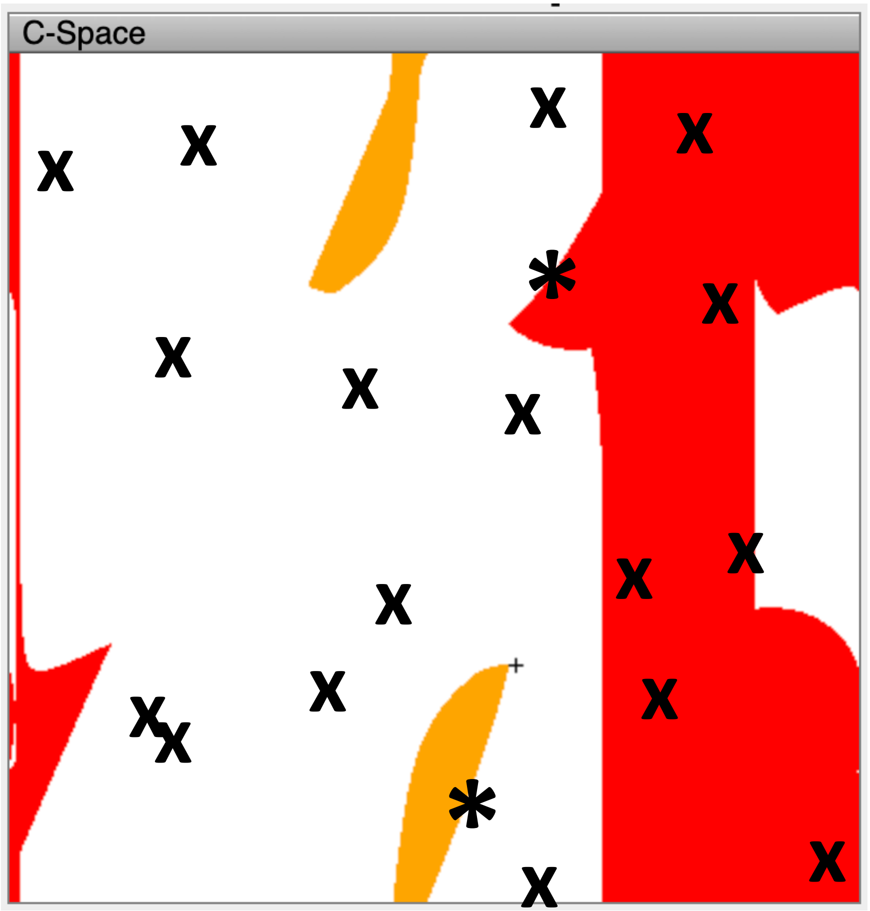
Randomly generate a set of points in $\mathcal{C}$-space.

Consider only the paths that go through those points!



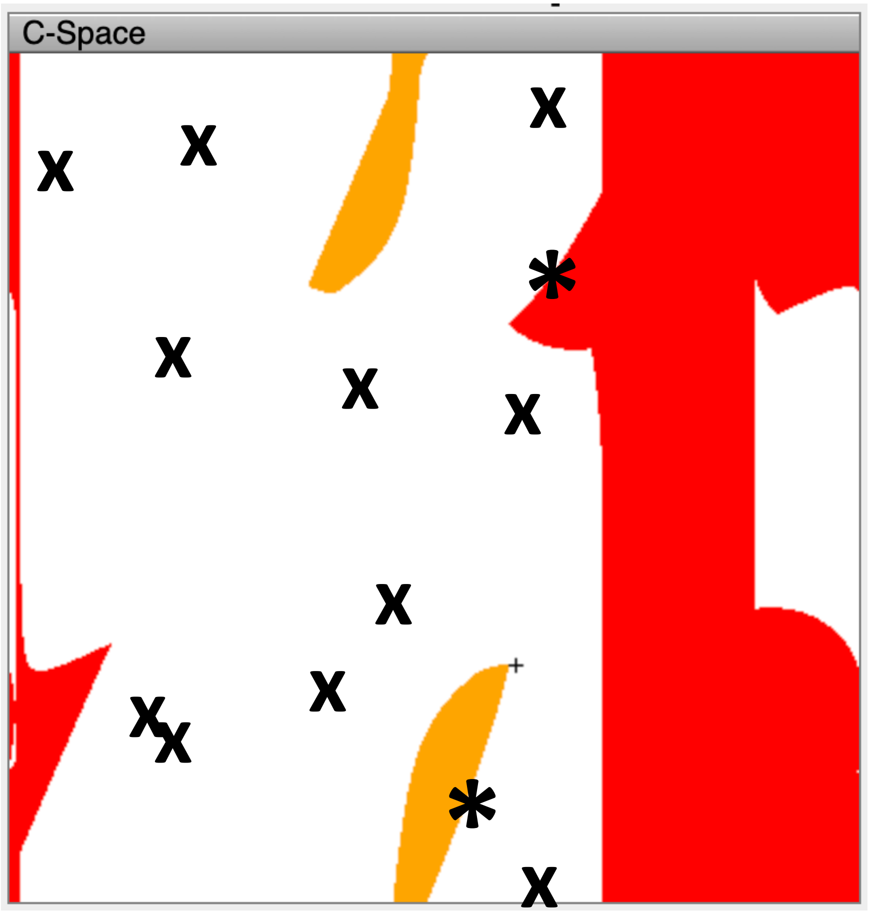
RRT

1. Generate a bunch of randomly sampled points to serve as search nodes
2. Eliminate the points that are inside obstacles
3. Perform A* over the remaining points to find the best path
4. Generate more samples in the vicinity of best points
5. Repeat steps 2 through 4



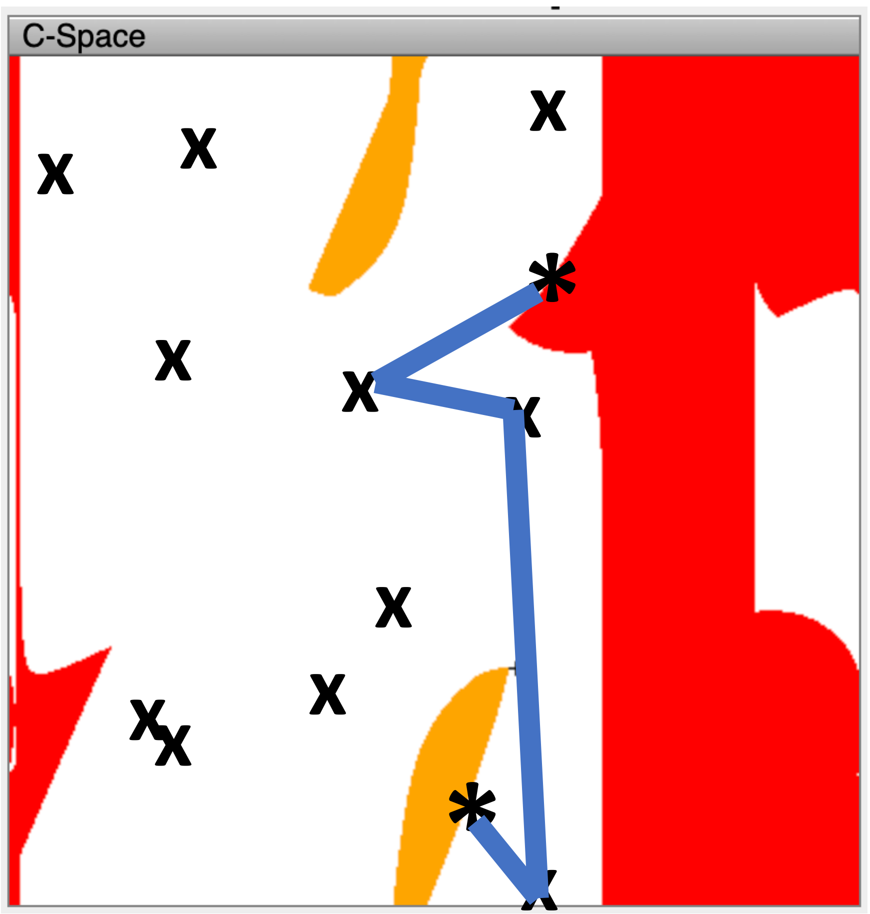
RRT

1. Generate a bunch of randomly sampled points to serve as search nodes
2. Eliminate the points that are inside obstacles
3. Search the remaining points to find the best path
4. Generate more samples in the vicinity of best points
5. Repeat steps 2 through 4



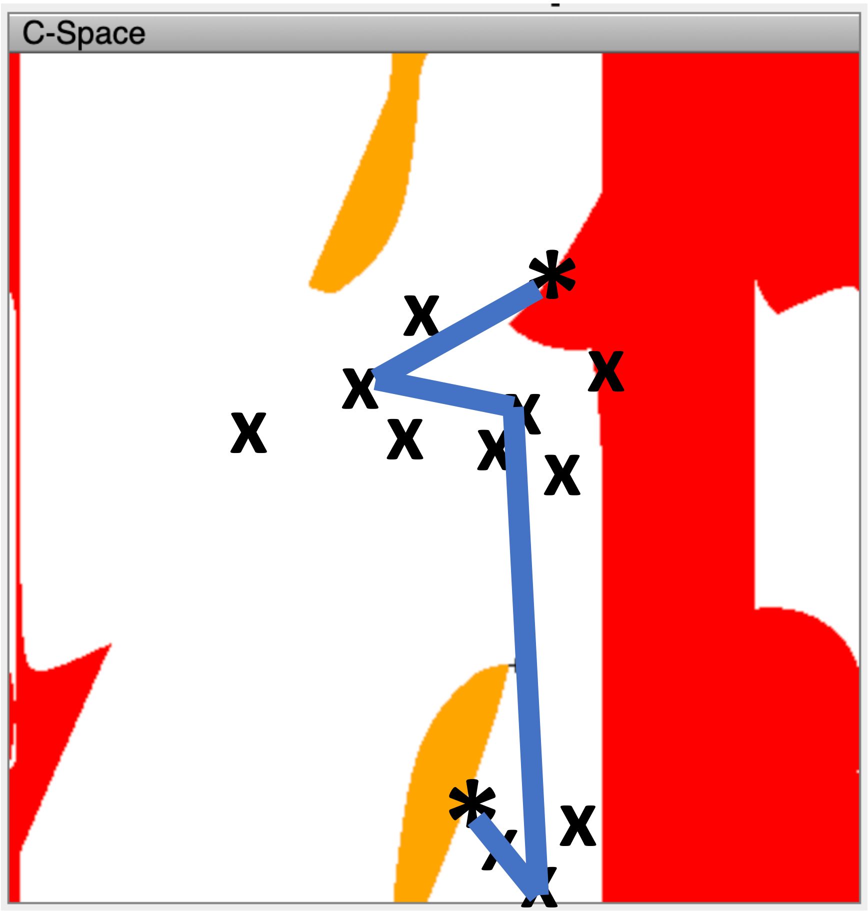
RRT

1. Generate a bunch of randomly sampled points to serve as search nodes
2. Eliminate the points that are inside obstacles
3. Search the remaining points to find the best path
4. Generate more samples in the vicinity of best points
5. Repeat steps 2 through 4



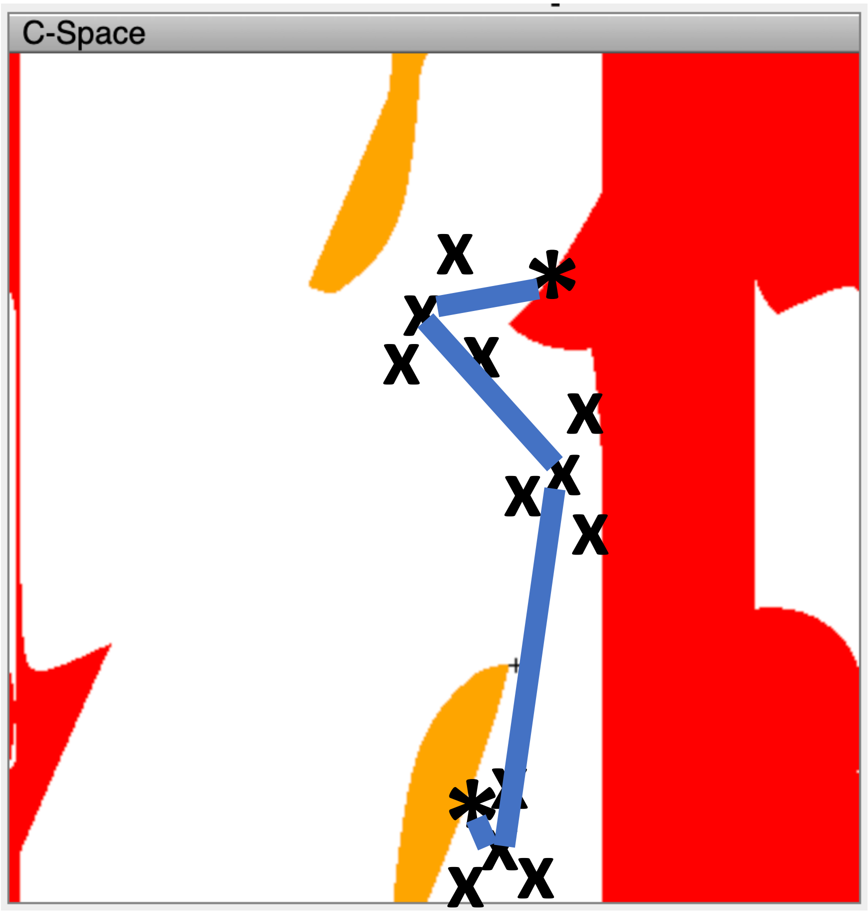
RRT

1. Generate a bunch of randomly sampled points to serve as search nodes
2. Eliminate the points that are inside obstacles
3. Search over the remaining points to find the best path
4. Generate more samples in the vicinity of best points
5. Repeat steps 2 through 4



RRT

1. Generate a bunch of randomly sampled points to serve as search nodes
2. Eliminate the points that are inside obstacles
3. Search over the remaining points to find the best path
4. Generate more samples in the vicinity of best points
5. Repeat steps 2 through 4



Key benefits of RRT

- Even with very limited computation (e.g., you can only afford one iteration), you still get a path that solves the problem
- In the limit of infinite computation (infinite # iterations), you get the best possible continuous-space path

Summary

- The robot path planning problem
- Workspace vs. Configuration space
  - Forward kinematics: $\boldsymbol{w} = \varphi(\boldsymbol{b}, \boldsymbol{c})$
  - Inverse kinematics: $\mathcal{C}_{\text{obs}} = \{\boldsymbol{c}: \exists \boldsymbol{b}: \varphi(\boldsymbol{b}, \boldsymbol{c}) \in \mathcal{W}_{\text{obs}}\}$
- Path planning: What is the best path?
  - Minimize $\mathcal{C}$-space distance while avoiding obstacles
- Path planning: How do you find the best path?
  - Rectangular discretization
  - Visibility graph
  - Rapid Random Trees