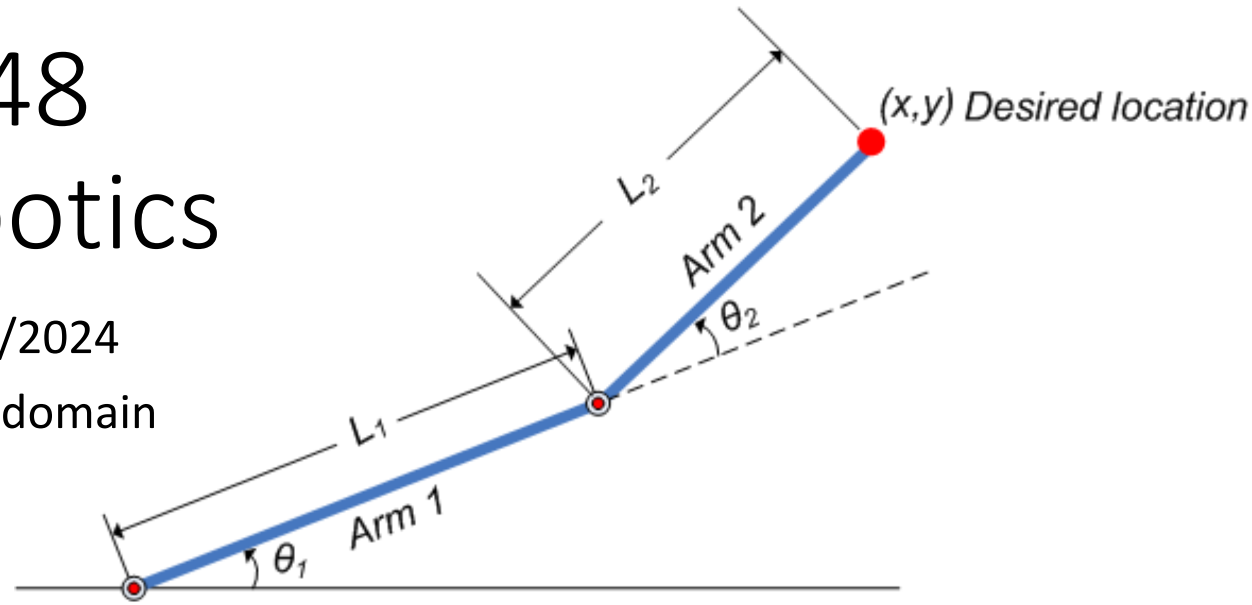
CS440/ECE448
Lecture 30: Robotics

Mark Hasegawa-Johnson, 4/2024



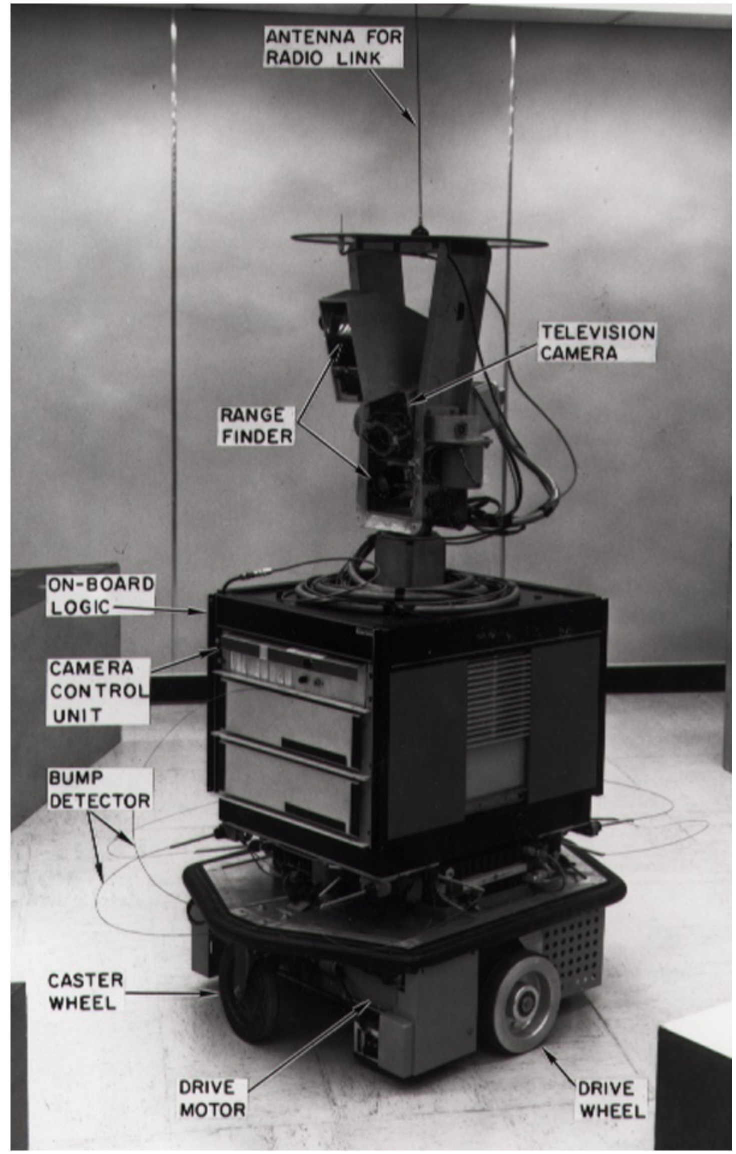
Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning
  - Visibility graph
  - Rapid Random Trees (RRT)
- Trajectory control
  - Proportion-Integral-Derivative (PID) controller
  - Model predictive control

What is a “Robot”?

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

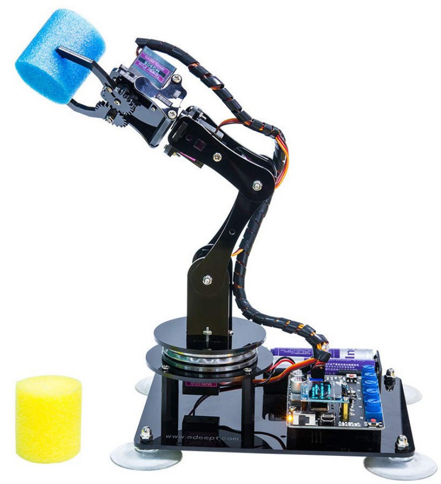
- Planning
  - Antenna for radio link
  - On-board logic
  - Camera control unit
- Perceiving
  - Range finder
  - Television camera
  - Bump detector
- Acting
  - Caster wheel
  - Drive motor
  - Drive wheel



Example: Robot Arm

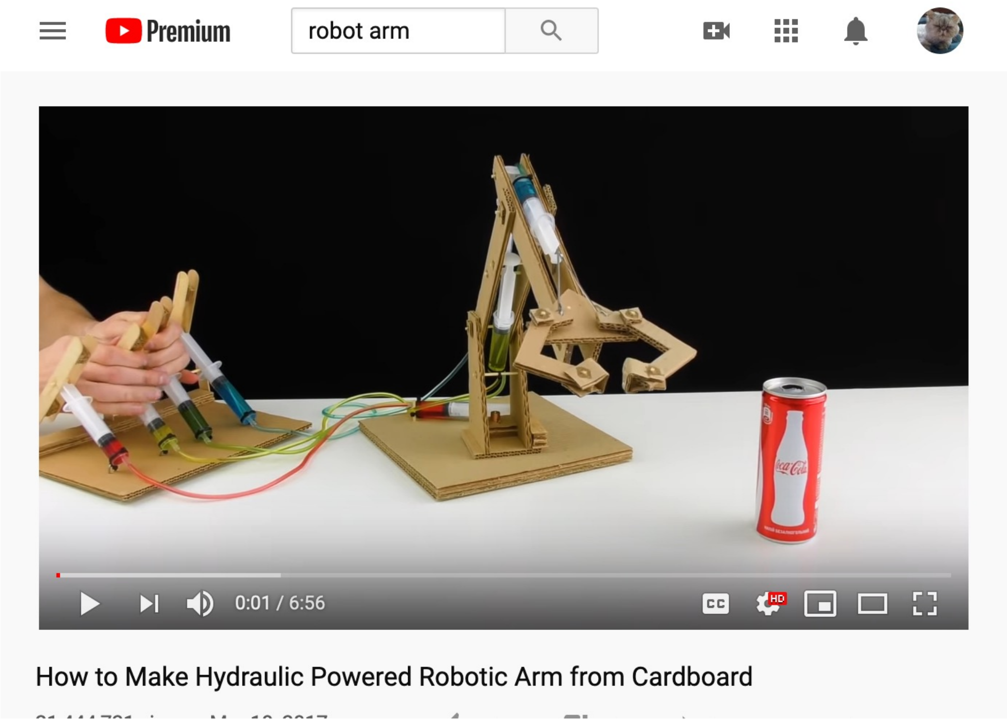
Adeept robot arm for Arduino (from Amazon)

- 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



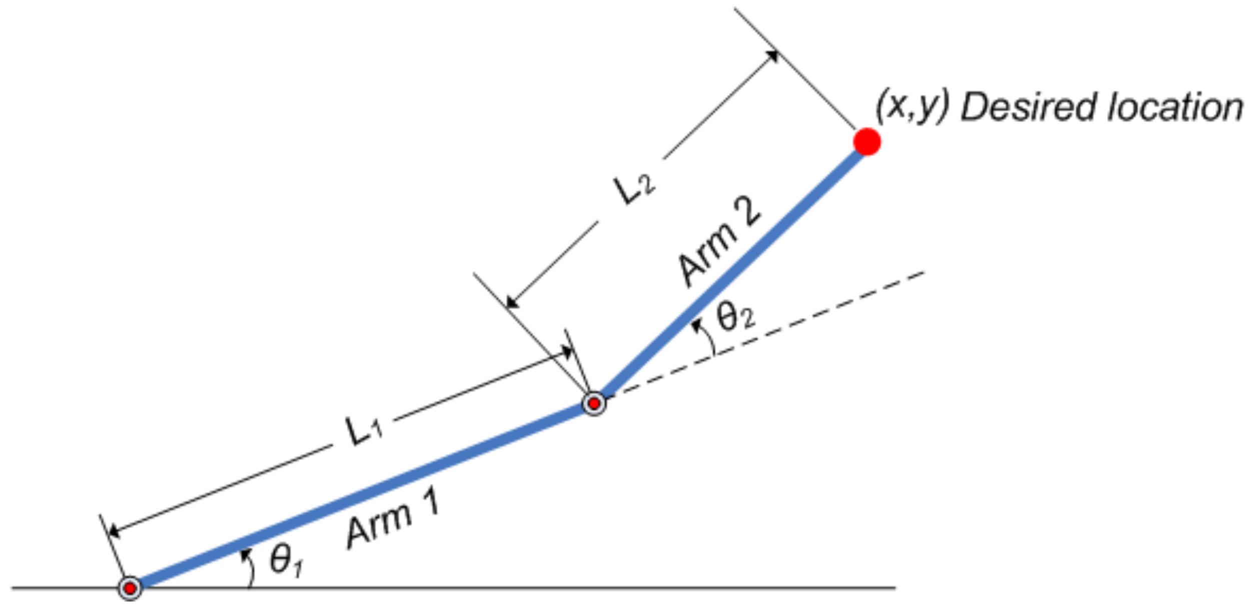
Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning
  - Visibility graph
  - Rapid Random Trees (RRT)
- Trajectory control
  - Time scaling
  - Proportion-Integral-Derivative (PID) controller
  - Model predictive control

The Robot Arm Reaching Problem

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

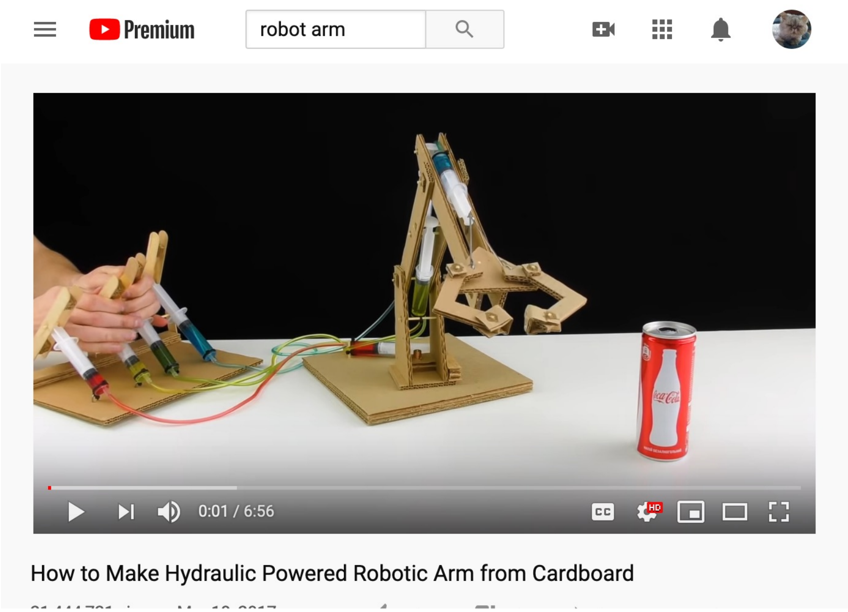
- Our goal is to reach a particular location (x,y)
- But we can't control (x,y) directly! What we actually control is (θ_1, θ_2).



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**, C, is the set of joint angles that govern the robot's shape. For example, if we have four angles to control, then $C \subset \mathbb{R}^4$:

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



Forward kinematics

The **forward kinematics** function, $\varphi_{\boldsymbol{b}}(\boldsymbol{q})$, maps (point on robot × configuration space)→(workspace). This is just geometry. Example:

- $\boldsymbol{b} = [b_1, b_2]^T$ = a particular point on the arm which is b meters from the shoulder, $0 \leq b_1 \leq L_1, 0 \leq b_2 \leq L_2$
- $\boldsymbol{q} = [\theta_1, \theta_2]^T$

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

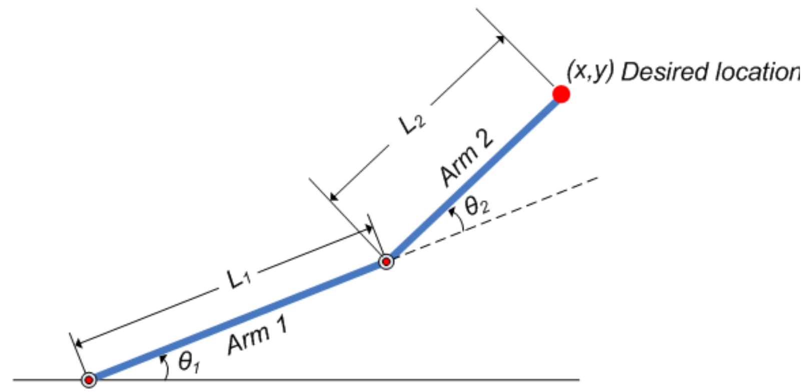
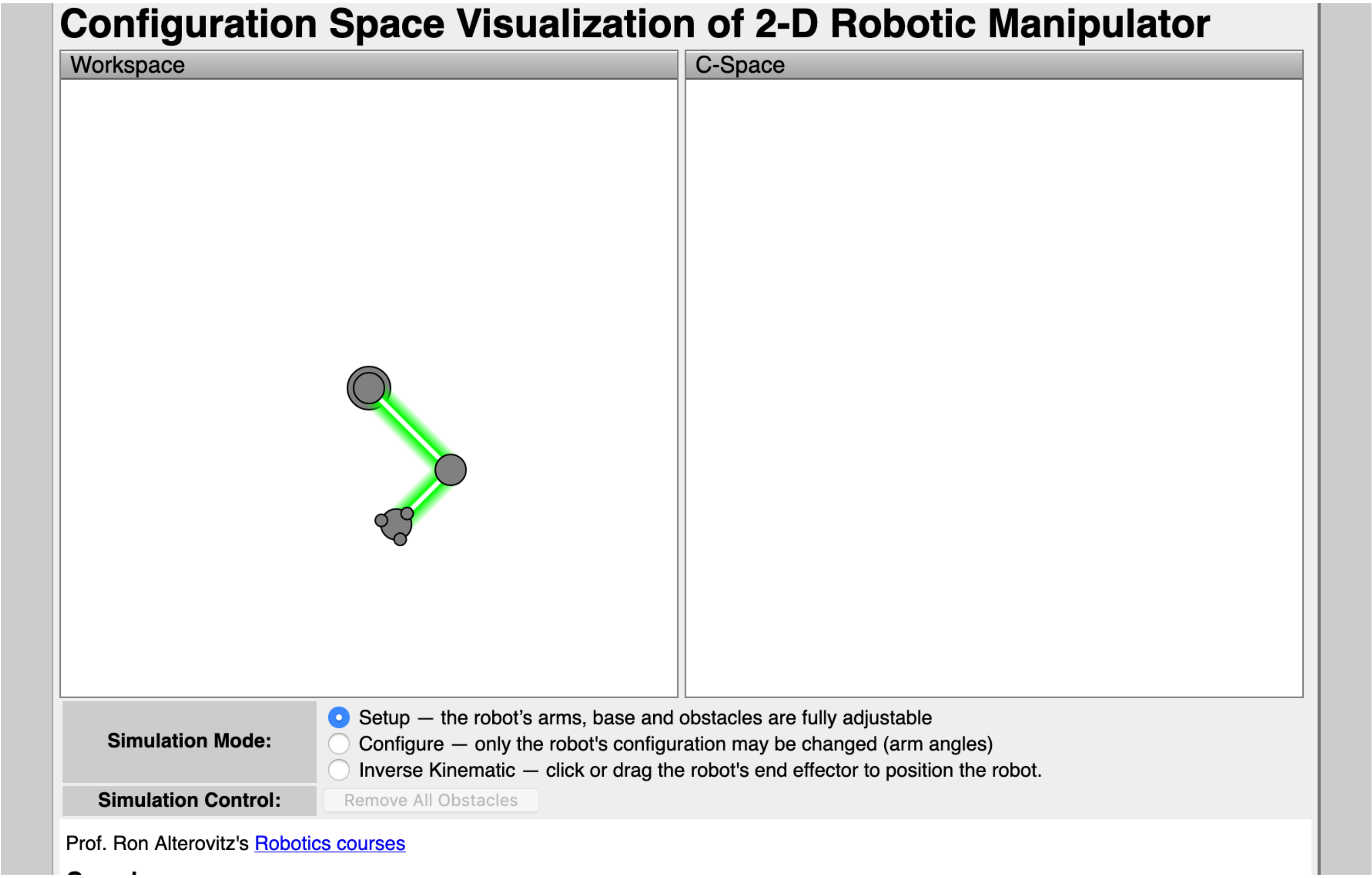


Image © https://www.mathworks.com/help/fuzzy/modeling-inverse-kinematics-in-a-robotic-arm.html

The Robot Arm Reaching Problem

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



Quiz

Try the quiz!

https://us.prairielearn.com/pl/course_instance/147925/assessment/2412878

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.
- In order to do that, we need **inverse kinematics**: a function that converts obstacles in the workspace, $\mathcal{W}_{\text{obs}}$, into equivalent obstacles in configuration space, C_{obs}.

$$C_{\text{obs}} = \{q: \exists b: \varphi_b(\boldsymbol{q}) \in \mathcal{W}_{\text{obs}}\}$$

- For example: we usually do this by just exhaustively testing every point in configuration space, to see if it runs into an obstacle.

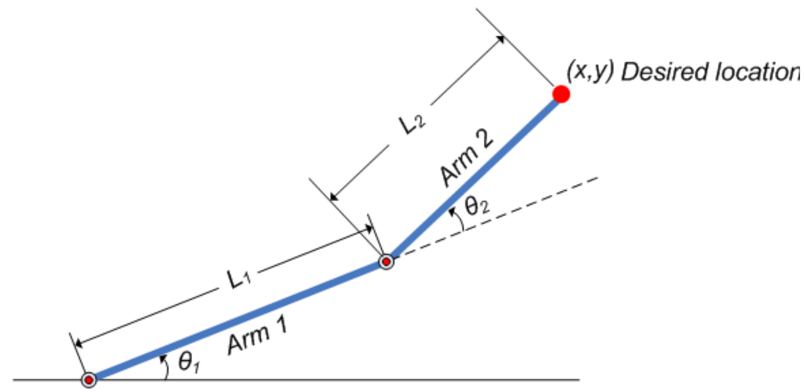
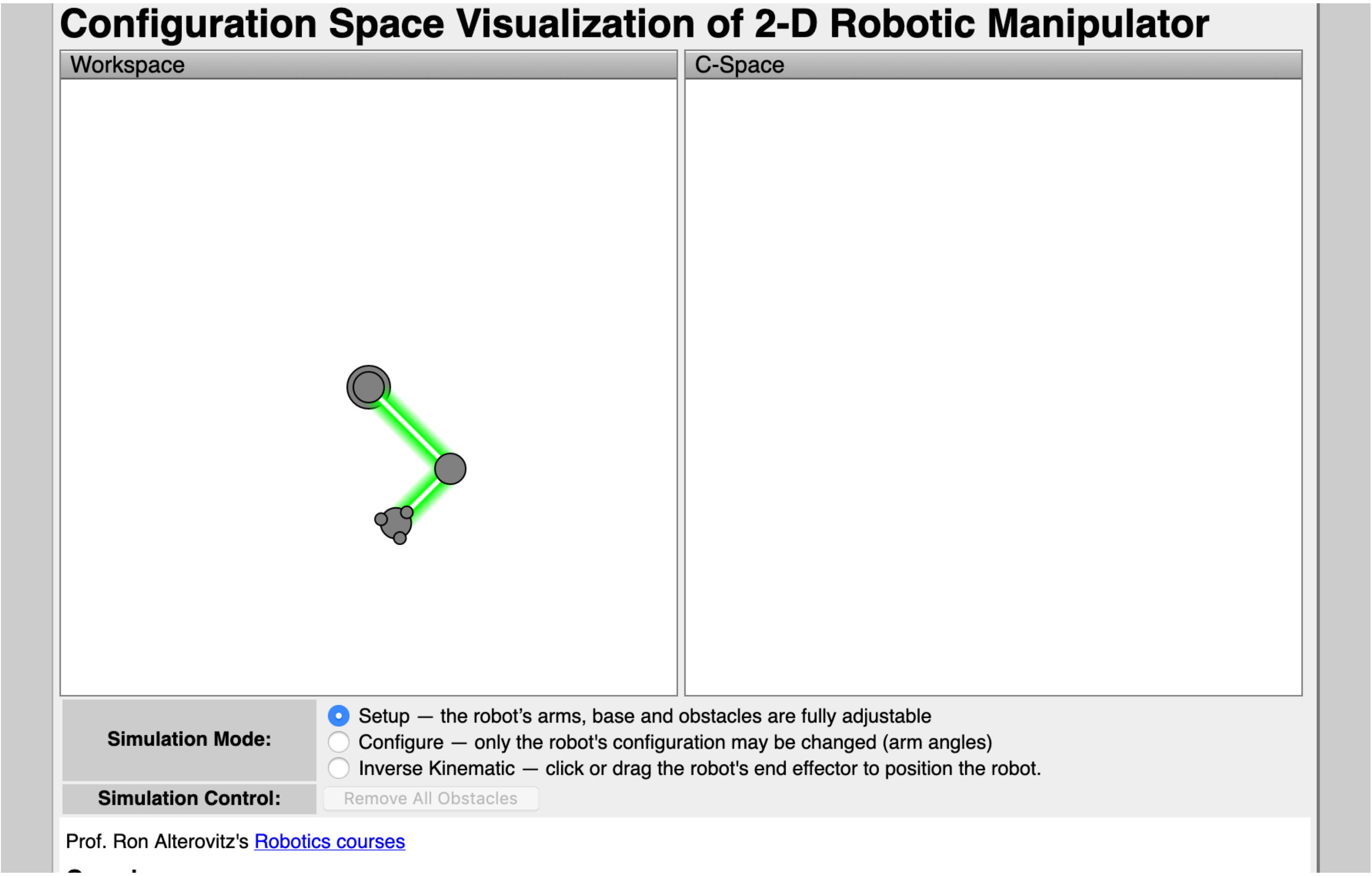


Image © https://www.mathworks.com/help/fuzzy/modeling-inverse-kinematics-in-a-robotic-arm.html

The Robot Arm Reaching Problem

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

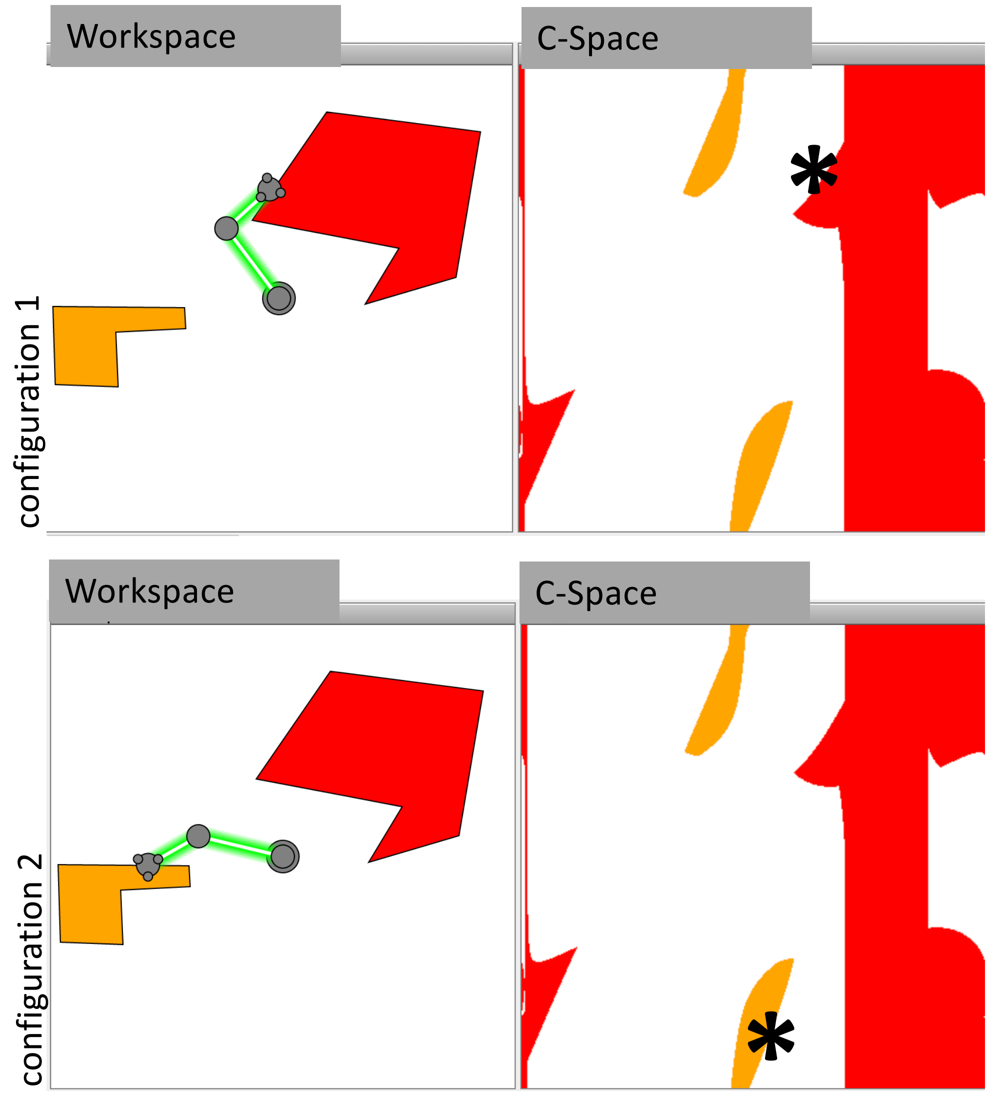


Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning
  - Visibility graph
  - Rapid Random Trees (RRT)
- Trajectory control
  - Time scaling
  - Proportion-Integral-Derivative (PID) controller
  - Model predictive control
- Model-based and model-free RL

The planning problem

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



What is “best”?

We need some way to define the word “best.”

Assumption: The shortest path in C-space is the best way to get from config 1 to config 2.

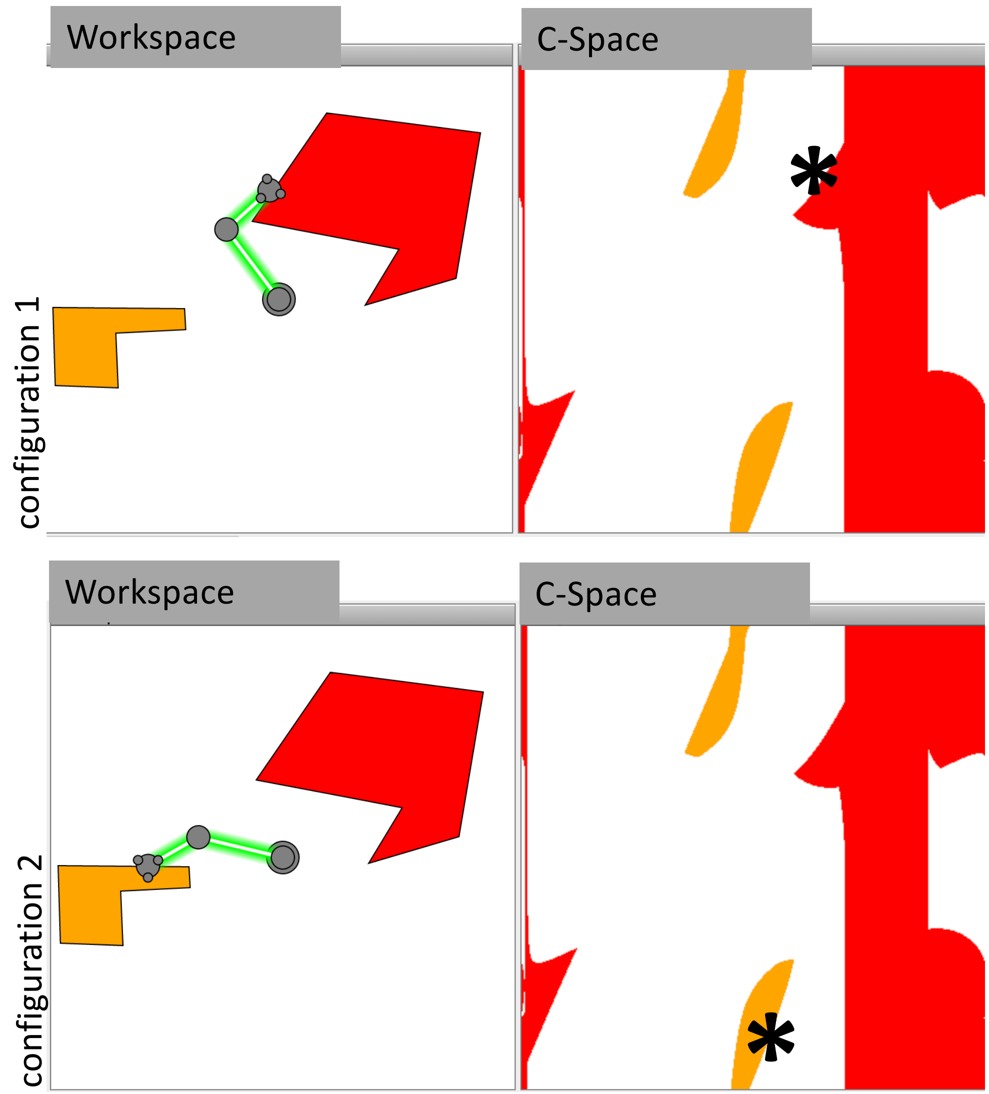
Implied assumption:

Longer path in C-space =

More manipulation of robot motors =

Greater energy expenditure =

Bad.



Finding the shortest path

Here are some algorithms you know that are guaranteed to find the shortest path:

- Dijkstra’s algorithm (BFS)
- A* search

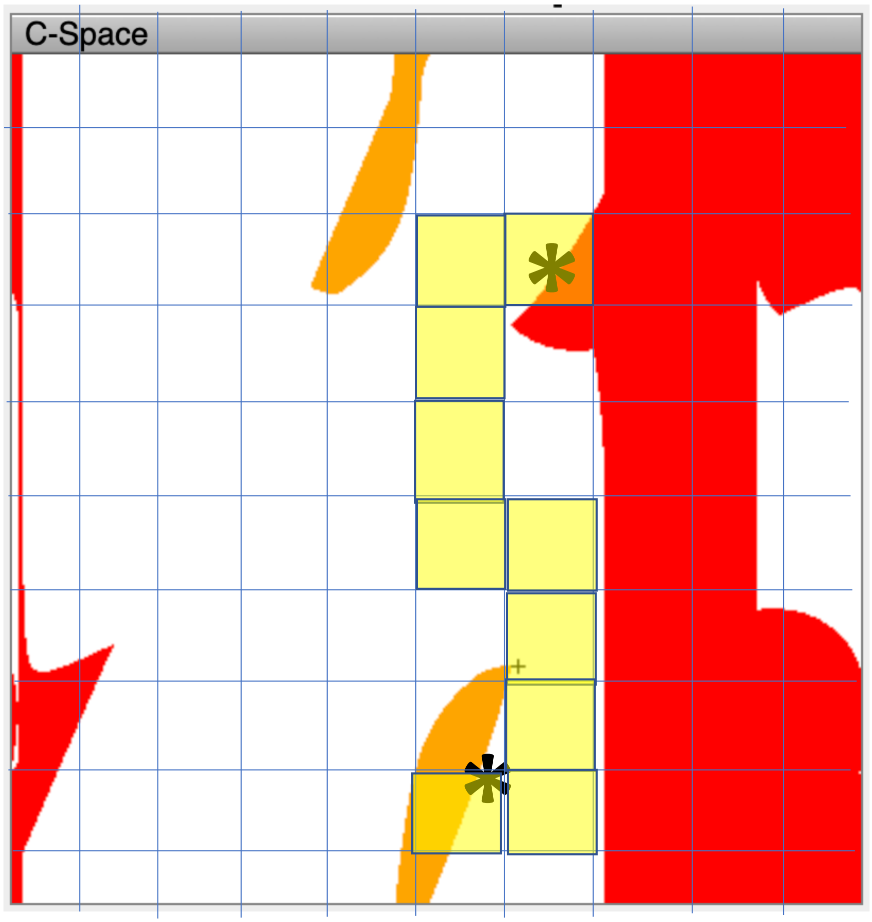
In fact, A* search was invented as a solution to the robot path planning problem. However, A* search is not quite well-suited to this problem, because…

A* requires discretizing the search space

A* assumes a discrete search space.

To apply it to the robot path-planning problem, we first need to discretize C-space.

We can discretize it using a rectangular grid, but doing so reduces the precision of our answer.



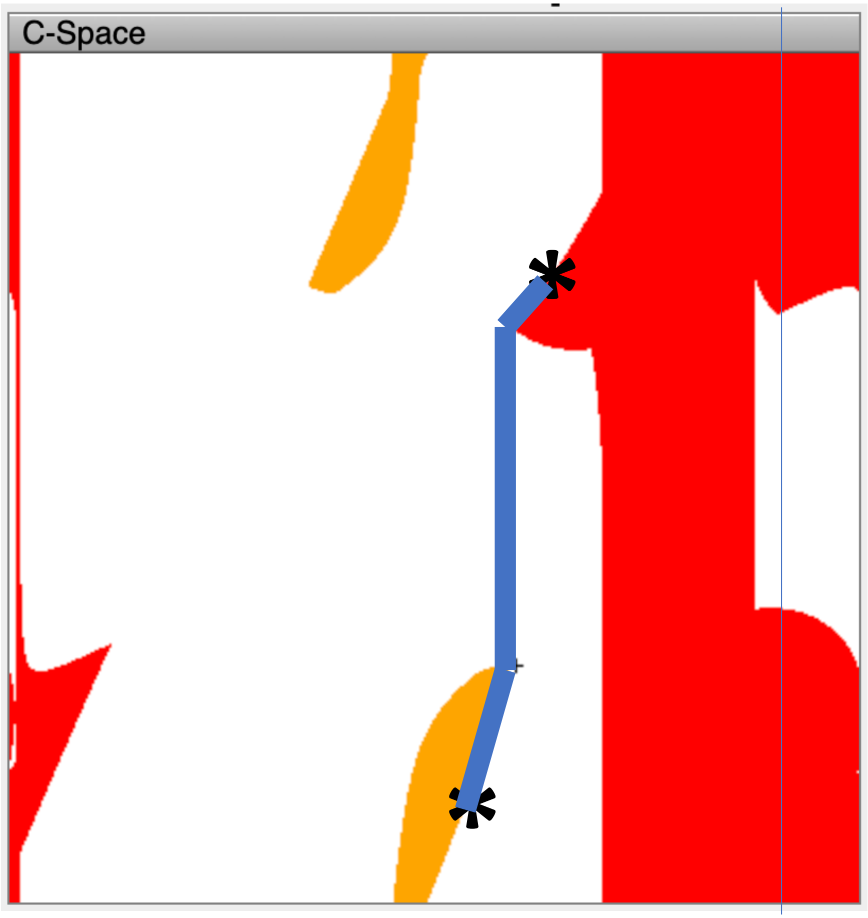
Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning
  - Visibility graph
  - Rapid Random Trees (RRT)
- Trajectory control
  - Time scaling
  - Proportion-Integral-Derivative (PID) controller
  - Model predictive control
- Model-based and model-free RL

Visibility Graph

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

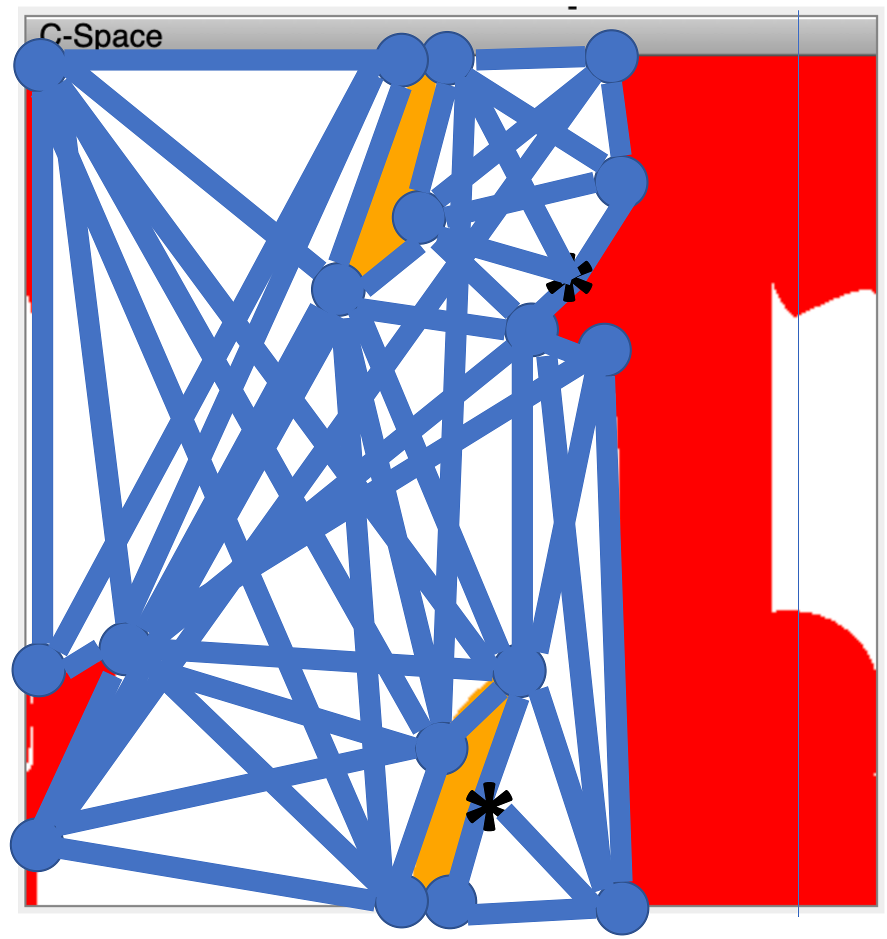
- From starting point to the corner of an obstacle, then…
- …from that corner to another corner, then….
- …from the corner of an obstacle to the goal.



Visibility Graph

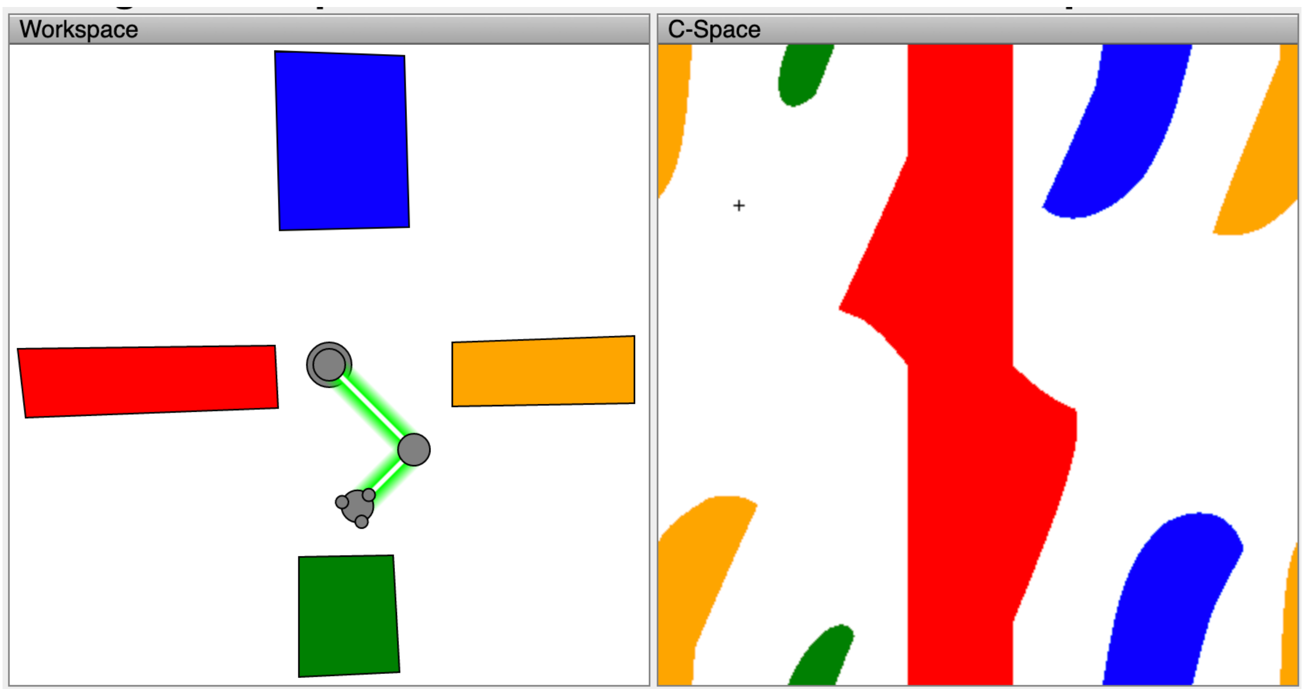
The algorithm, then, is:

1. Find all the corners.
2. Find the distances between every pair of corners.
3. Search that graph, using A*, to find the best path.



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.



Outline

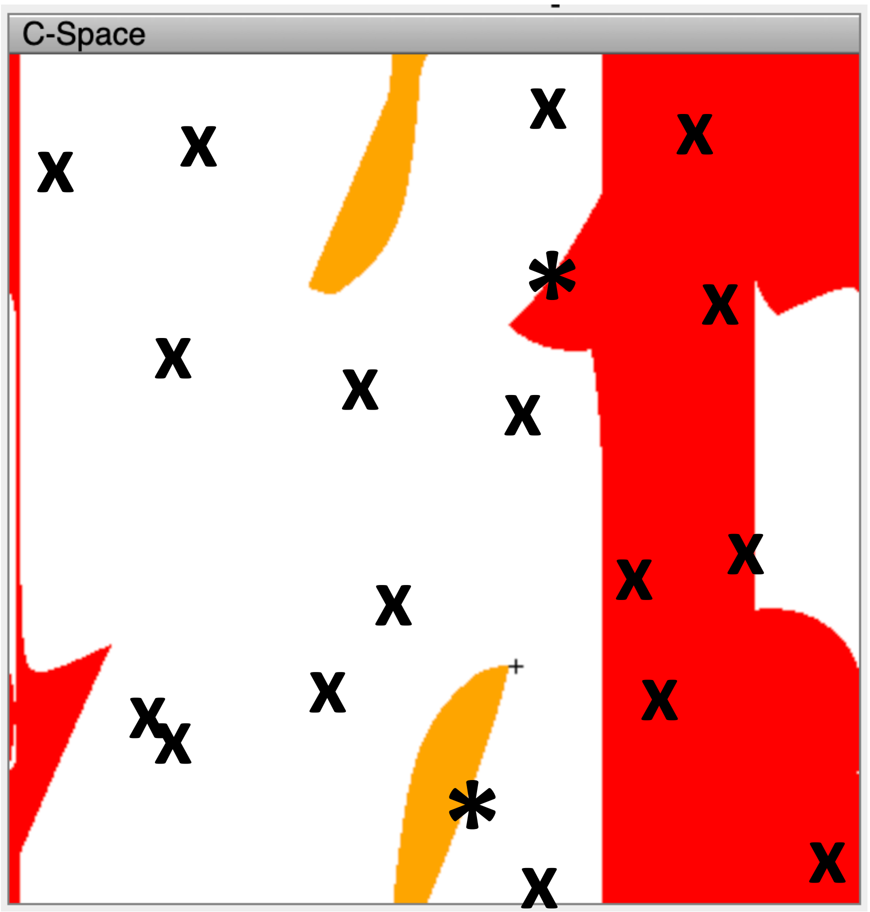
- The robot path planning problem
- Workspace vs. Configuration space
- Path planning
  - Visibility graph
  - Rapid Random Trees (RRT)
- Trajectory control
  - Time scaling
  - Proportion-Integral-Derivative (PID) controller
  - Model predictive control
- Model-based and model-free RL

C-Space Best-path algorithms

- A* on a rectangular grid
  - Search nodes: squares on the grid
- A* on a visibility graph
  - Search nodes: obstacle corners
- A* on a graph of rapid random trees (RRT)
  - Search nodes: randomly sampled points

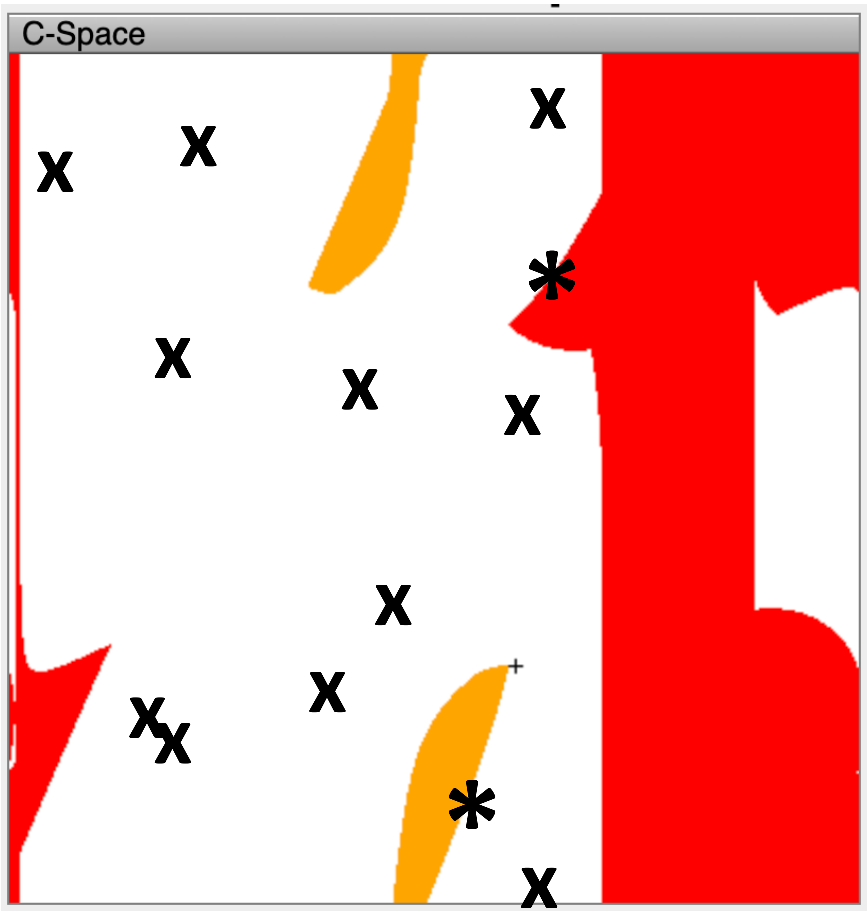
RRT

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



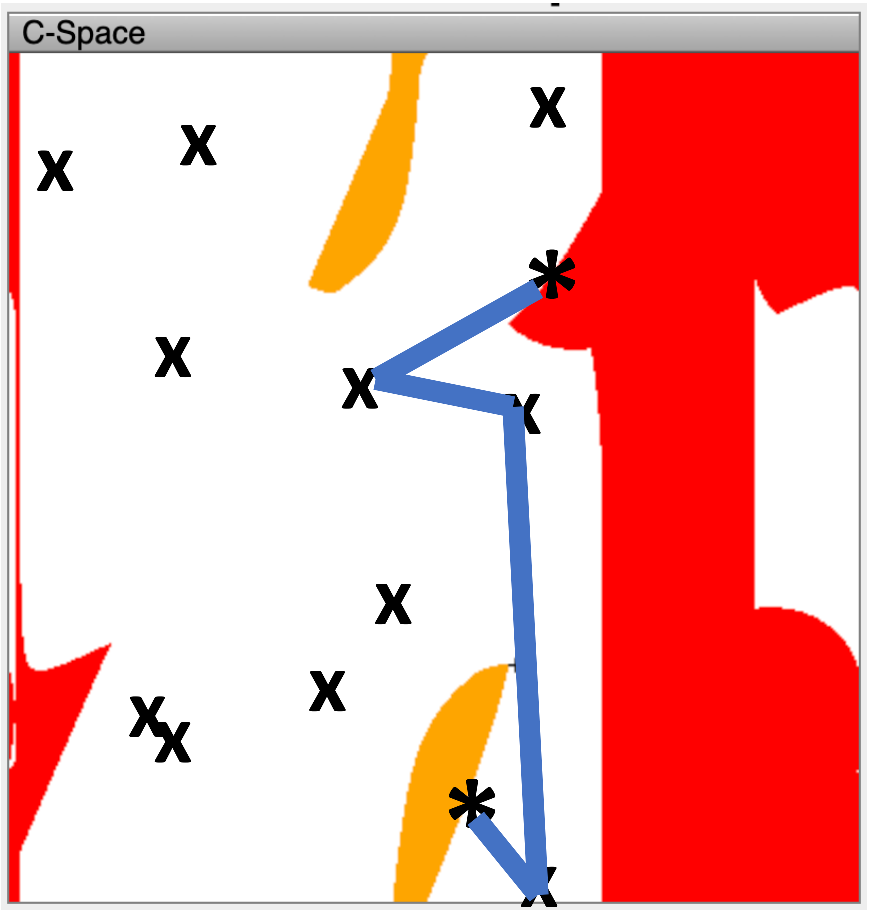
RRT

1. Generate a bunch of randomly sampled points to serve as search nodes
2. Eliminate the points that are inside obstacles
3. 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



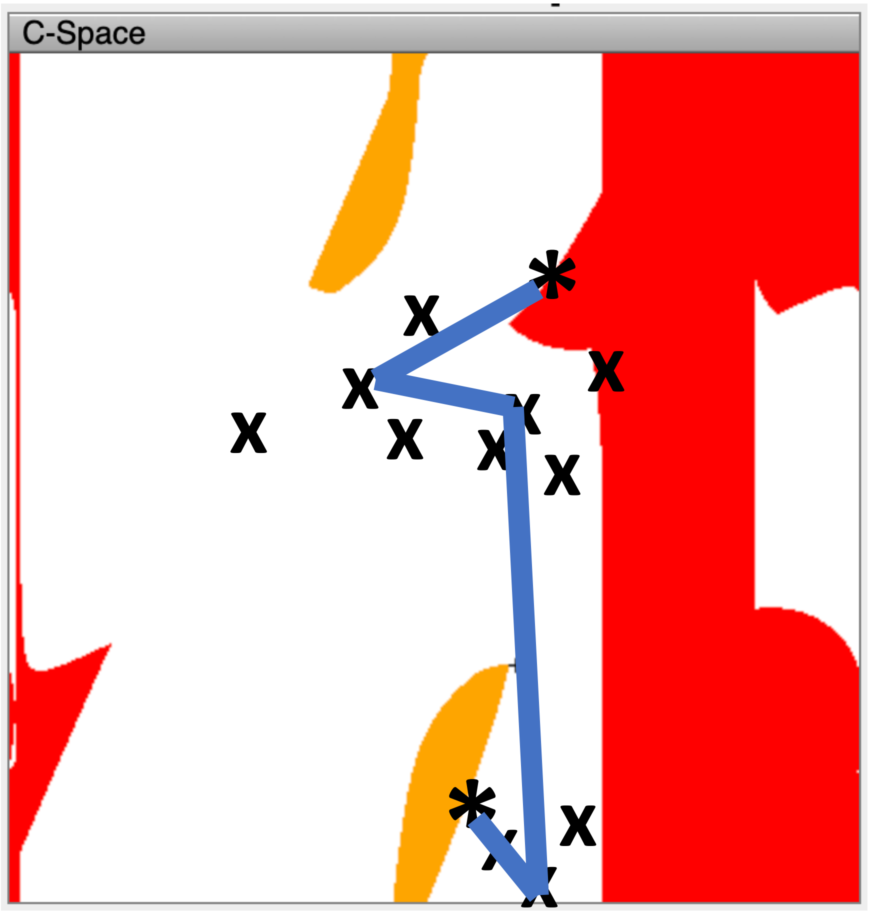
RRT

1. Generate a bunch of randomly sampled points to serve as search nodes
2. Eliminate the points that are inside obstacles
3. 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



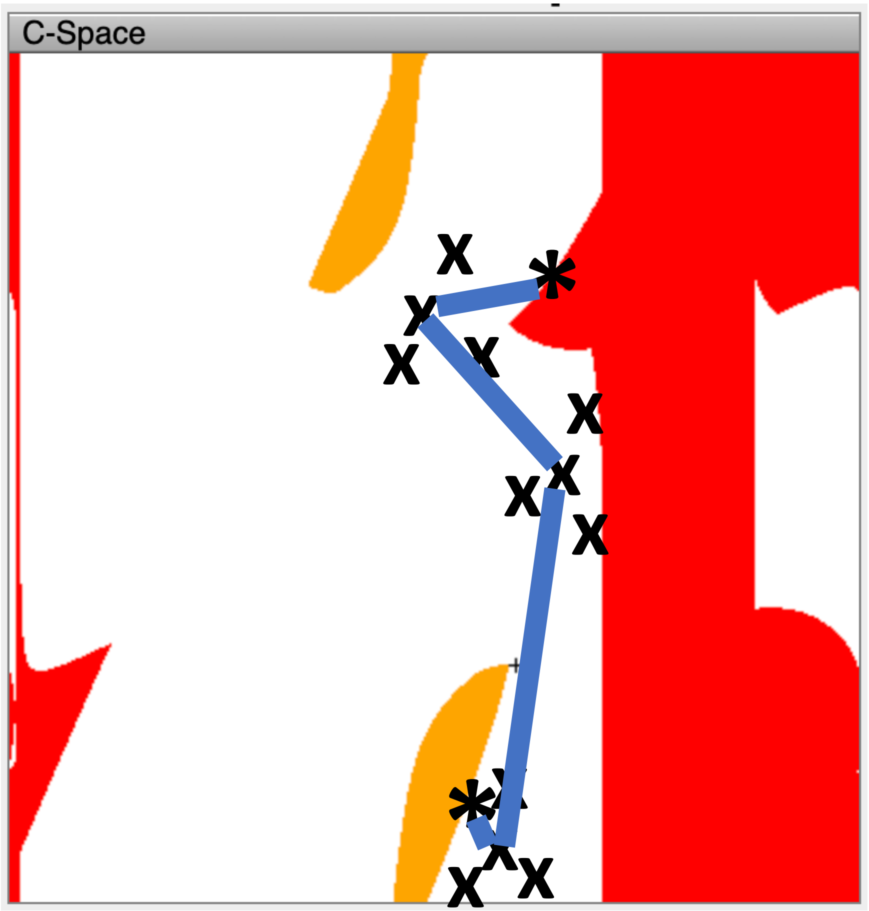
RRT

1. Generate a bunch of randomly sampled points to serve as search nodes
2. Eliminate the points that are inside obstacles
3. 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. 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



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

Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning
  - Visibility graph
  - Rapid Random Trees (RRT)
- Trajectory control
  - Time scaling
  - Proportion-Integral-Derivative (PID) controller
  - Model predictive control

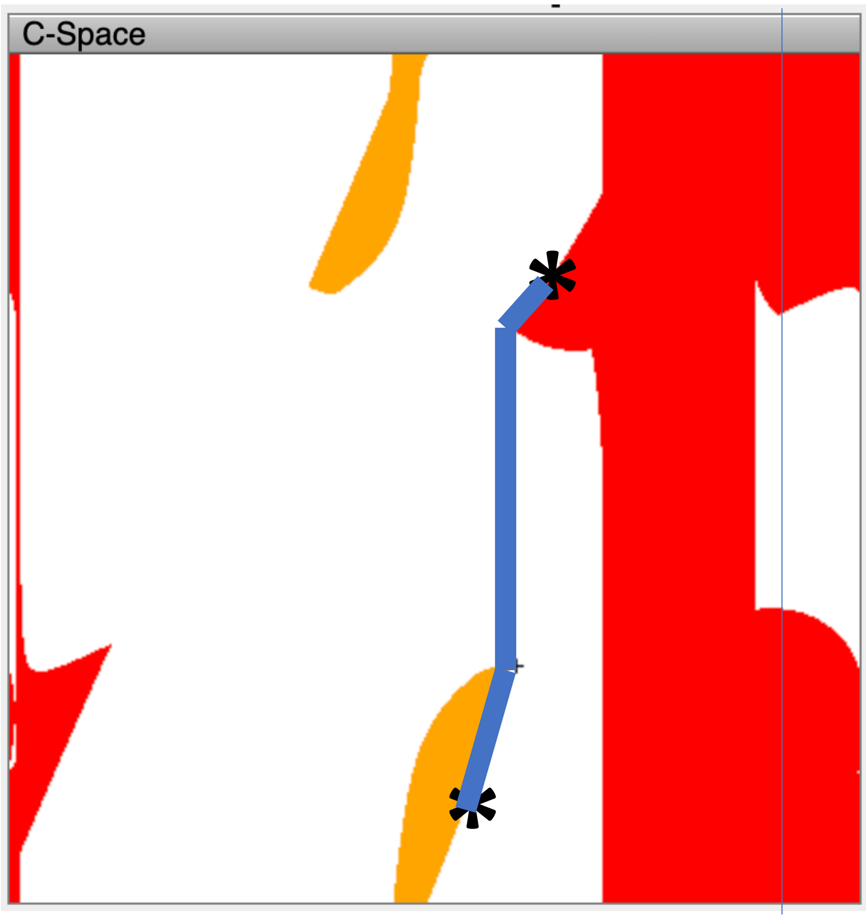
Trajectory control: maximum torque

Now that you have an optimum path, how fast should the robot travel along that path?

Consideration #1: maximum torque.

Find $\boldsymbol{q}(t) = \begin{bmatrix}\theta_1(t)\\ \theta_2(t)\end{bmatrix}$ so that

$$\left|\frac{d^2\theta_1}{dt^2}\right| \leq max_1, \left|\frac{d^2\theta_2}{dt^2}\right| \leq max_2$$



Trajectory control: maximum safe velocity

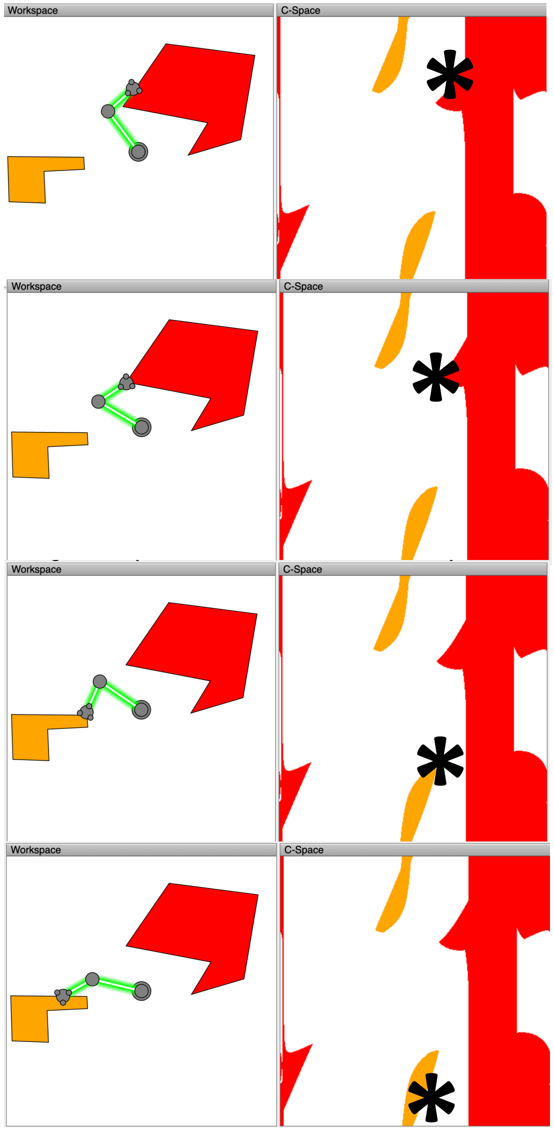
Consideration #2: maximum safe velocity.

Find $\boldsymbol{q}(t) = \begin{bmatrix} \theta_1(t) \\ \theta_2(t) \end{bmatrix}$ so that

$$\sqrt{\left(\frac{dw_1}{dt}\right)^2 + \left(\frac{dw_2}{dt}\right)^2} \leq v_{max}$$

…where $\boldsymbol{w}(t)$ is any solution to the inverse kinematics:

$$\boldsymbol{w}(t) \in \{\boldsymbol{w}: \exists \boldsymbol{b}: \varphi_{\boldsymbol{b}}(\boldsymbol{q}(t)) = \boldsymbol{w}(t)\}$$

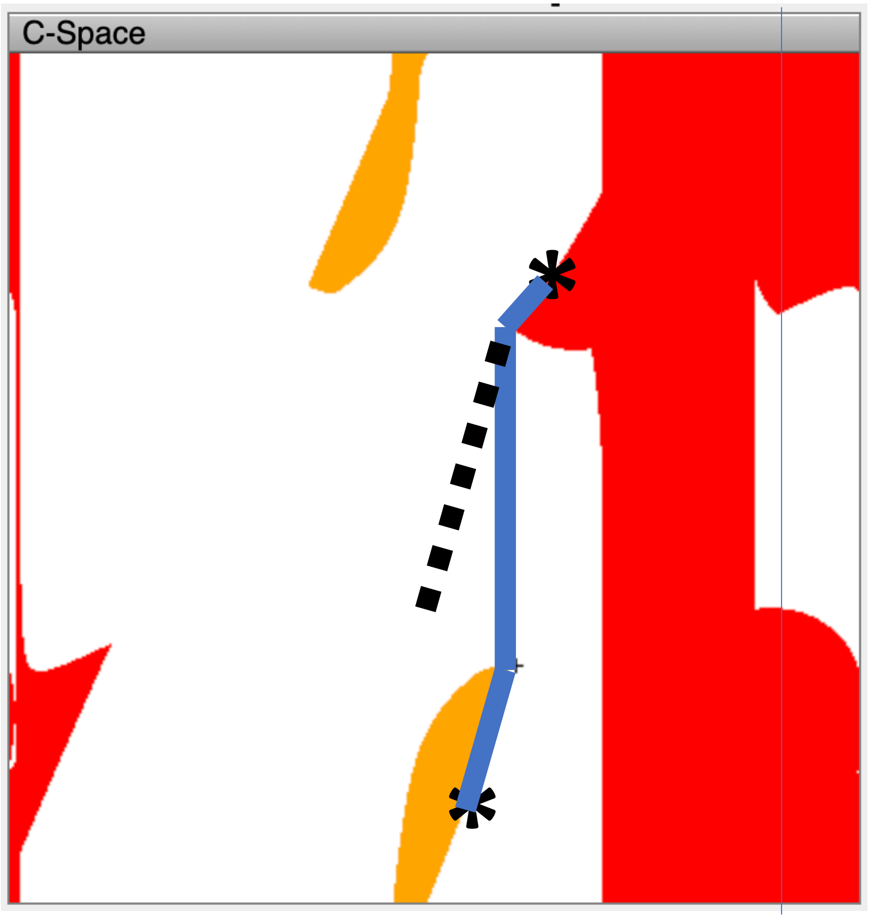


Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning
  - Visibility graph
  - Rapid Random Trees (RRT)
- Trajectory control
  - Time scaling
  - Proportion-Integral-Derivative (PID) controller
  - Model predictive control

Trajectory control: error management!!!

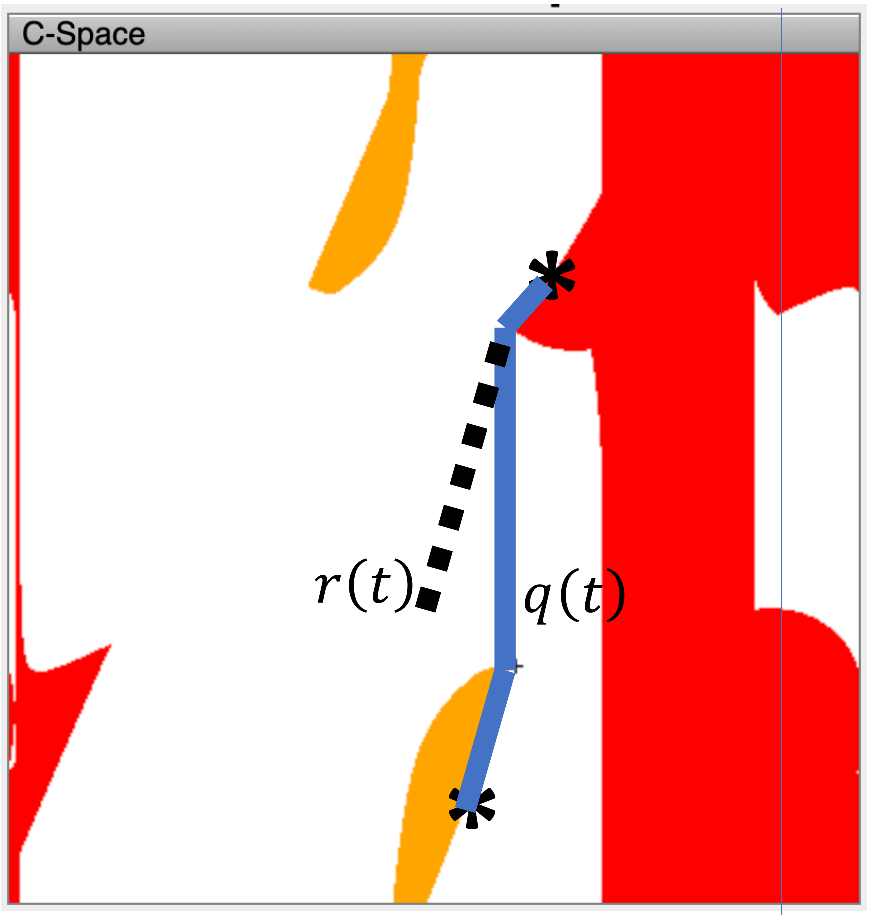
Consideration #3: what do you do if you start on a path but discover that your motor is miscalibrated and you're going the wrong direction?



P-controller

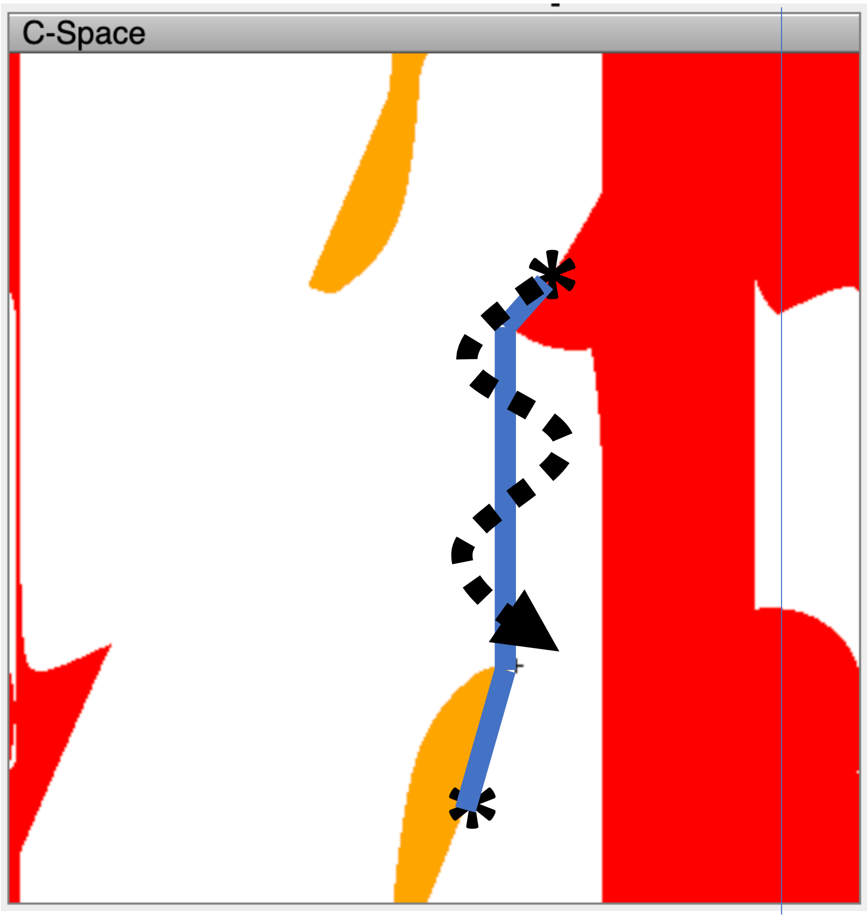
A proportional controller (P-controller) adds some extra torque in proportion to the error:

$$\frac{d^2}{dt^2}\begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix} = K(\boldsymbol{q}(t) - \boldsymbol{r}(t))$$



P-controller Problems

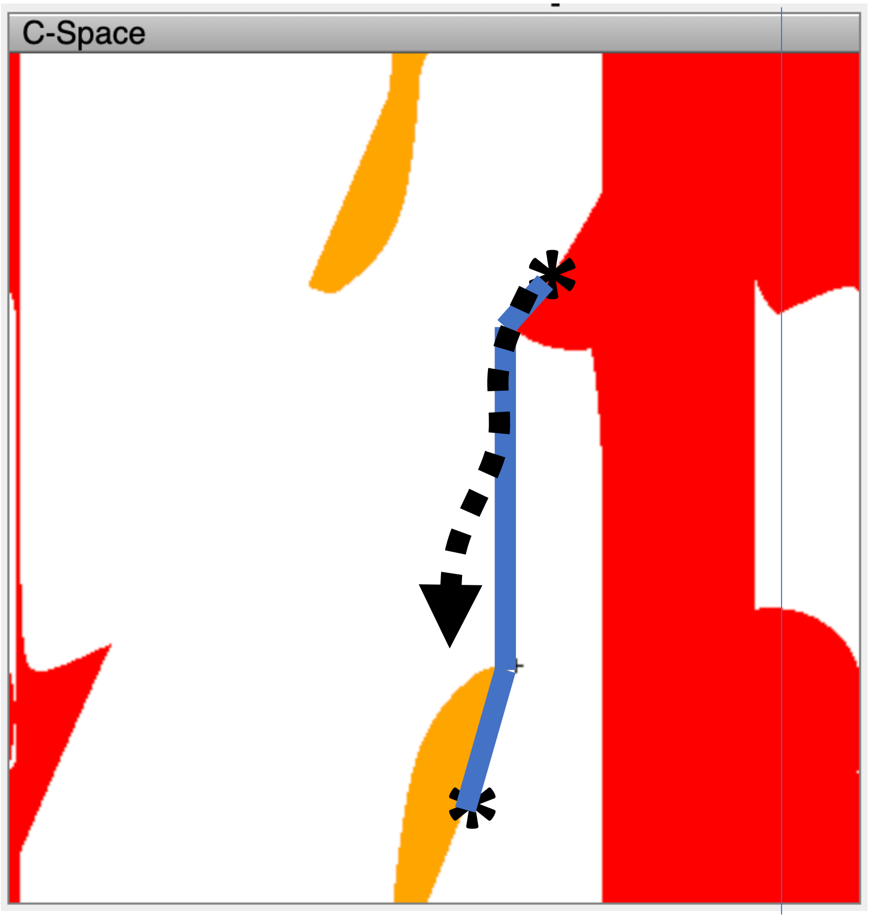
A P-controller tends to result in oscillating overshoot.



PD-controller

A proportional-derivative controller (PD-controller) adds some extra torque in proportion to the error of the derivative:

$$\frac{d^2}{dt^2}\begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix} = K_P(\boldsymbol{q}(t) - r(t)) + K_D(\dot{\boldsymbol{q}}(t) - \dot{\boldsymbol{r}}(t))$$



Doing this can smooth out the trajectory, but can leave some long-term error

PID-controller

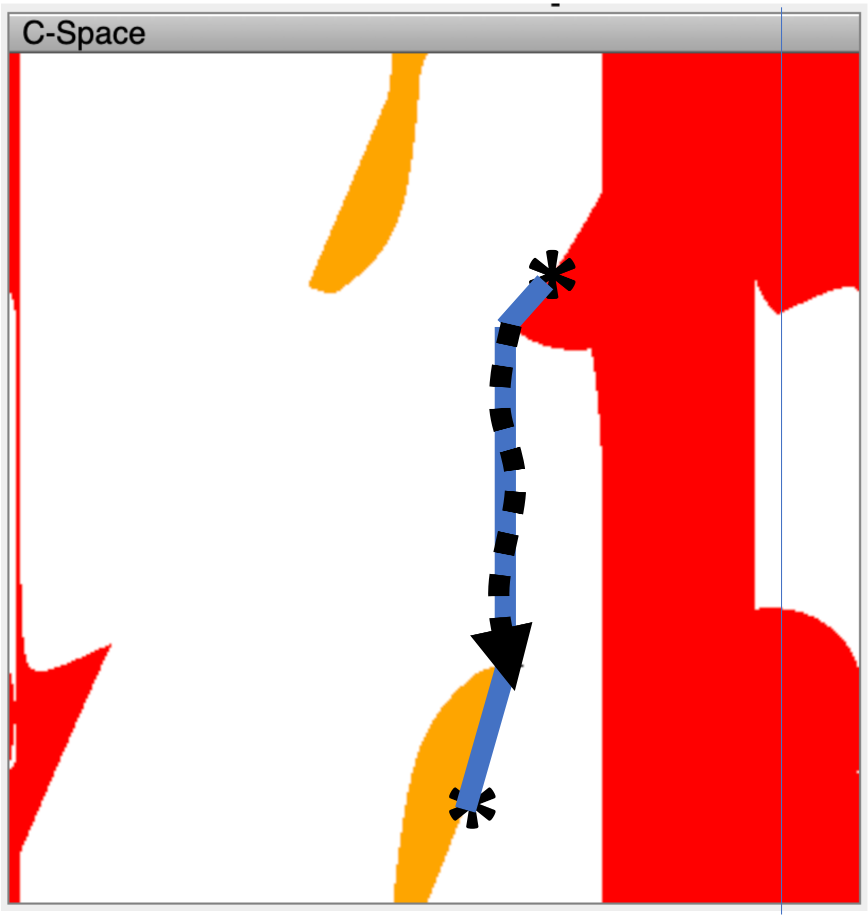
A proportional-integral-derivative controller (PID-controller) adds some extra torque in proportion to the error of the integral:

$$\frac{d^2}{dt^2}\begin{bmatrix}\theta_1\\\theta_2\end{bmatrix} = K_P(\boldsymbol{q}(t) - \boldsymbol{r}(t)) + K_I\int_0^t(\boldsymbol{q}(\tau) - \boldsymbol{r}(\tau))d\tau + K_D(\dot{\boldsymbol{q}}(t) - \dot{\boldsymbol{r}}(t))$$

The P term fixes short-term errors.

The I term fixes long-term errors.

The D term smooths out oscillations.

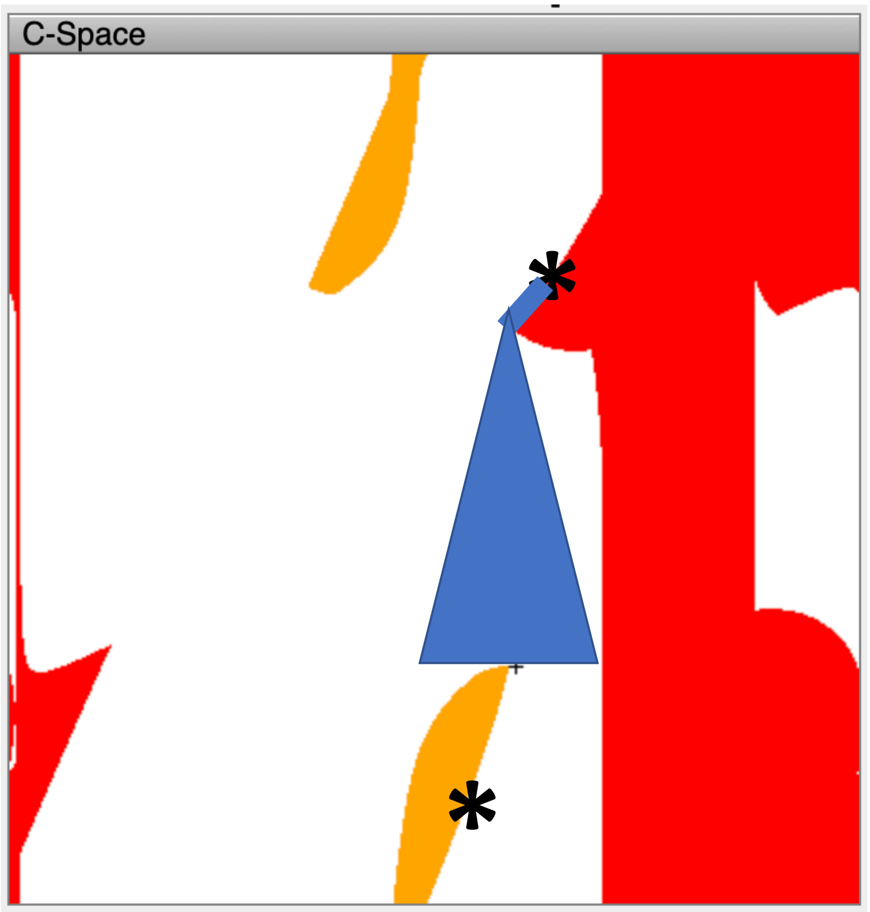


Outline

- The robot path planning problem
- Workspace vs. Configuration space
- Path planning
  - Visibility graph
  - Rapid Random Trees (RRT)
- Trajectory control
  - Time scaling
  - Proportion-Integral-Derivative (PID) controller
  - Model predictive control

What if your motors behave randomly?

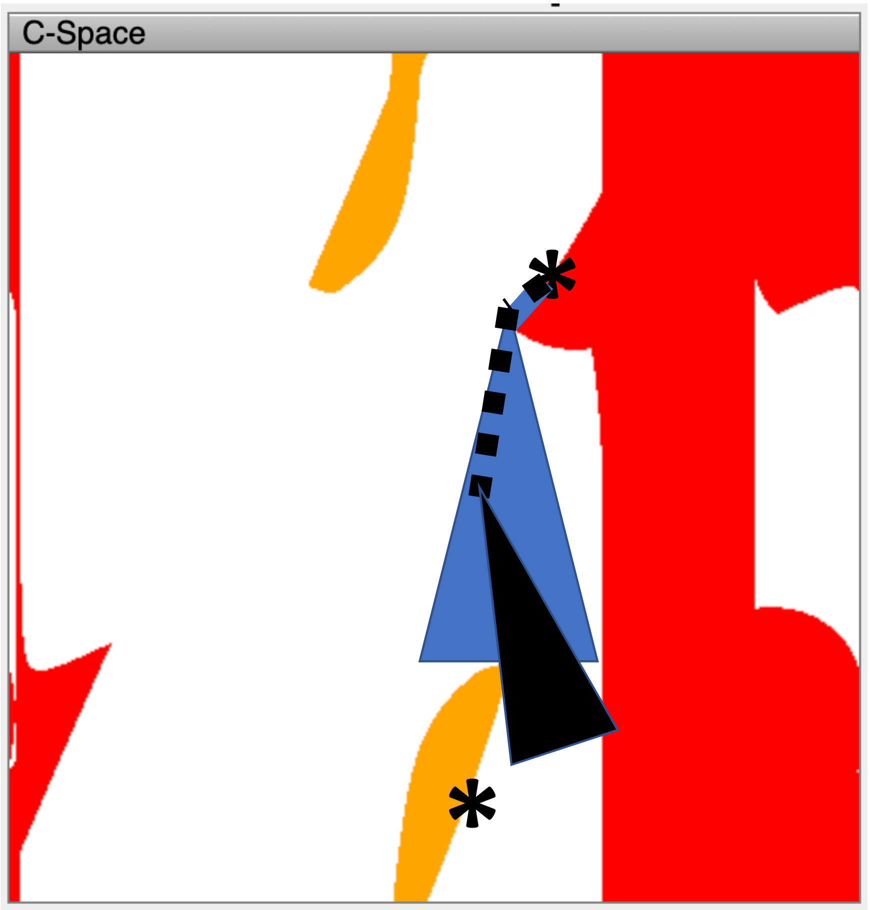
- What if your motors have some randomness?
- Then you might not be able to plan an exact trajectory.
- The best you can do is plan a trajectory that goes in the right general direction.



Model predictive control

… means the following strategy.

1. Plan an optimum trajectory
2. Go partway
3. Observe where you are
4. Recalculate the optimal trajectory
5. Repeat



Summary

- The robot path planning problem
- Workspace (e.g., $\boldsymbol{w} = [x, y]^T$) vs. Configuration space (e.g., $\boldsymbol{q} = [\theta_1, \theta_2]^T$)
- Path planning
  - Visibility graph: states=vertices in configuration space
  - Rapid Random Trees (RRT): states=random, resampled near the best path after every iteration
- Trajectory control
  - Time scaling: Constraints on motor torque, workspace velocity
  - Proportion-Integral-Derivative (PID) controller: Smooth out oscillations
  - Model predictive control: Plan for the possibility of error