Motion Planning

Zackory Erickson

Carnegie Mellon University
The Robotics Institute

16-762: Mobile Manipulation
Spring 2026



Teams for Labs 2+ and Course project

- Teams of 3 students
- Use **Canvas Discussions** to find team members, or look for team members in-person!
- Discussions Link: https://canvas.cmu.edu/courses/52596/discussion_topics/796930
- Once you find a team, please have one member submit the team member names: https://forms.gle/nNzqL7iCRX6jCxUj9

Let’s now look at IK on Stretch

Continuing from last lecture:

```
source env/bin/activate
pip3 install --upgrade ikpy graphviz networkx urchin
```

https://github.com/Zackory/mm2026/blob/main/stretch_python/stretch_ik.py

Overview

Pick and place

Configuration space

Path planning

MoveIt 2

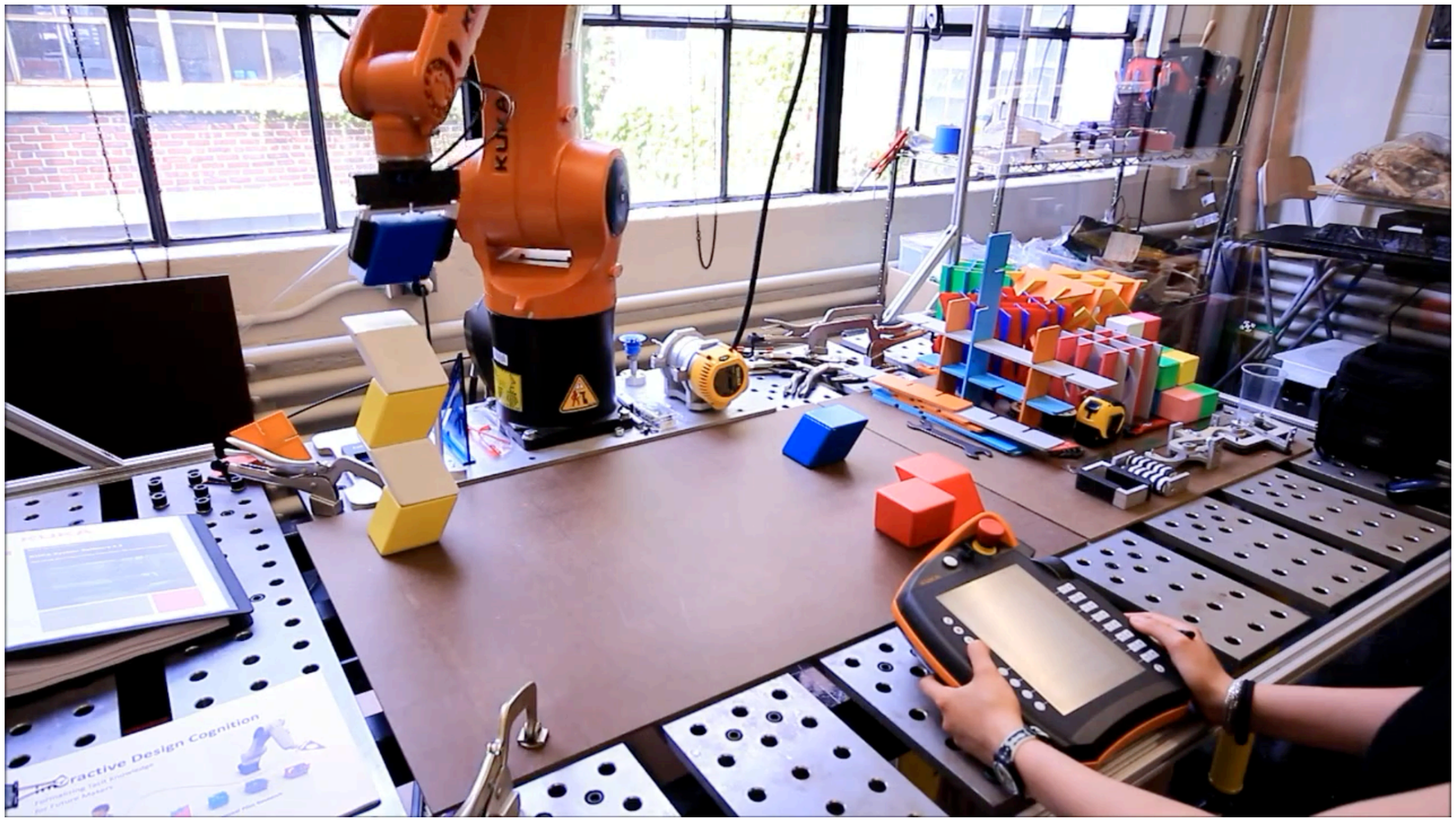
Pick and place

- Let `nextobj()` describe next object to be moved;
- Let `start(obj)` give the initial location of `obj`;
- Let `goal(obj)` give the goal location of `obj`.
- Pick and place follows the pattern:

```
FOR obj = nextobj()
   MOVETO start(obj)   // pick
   CLOSE
   MOVETO goal(obj)    // place
   OPEN
```

Pick and place applications

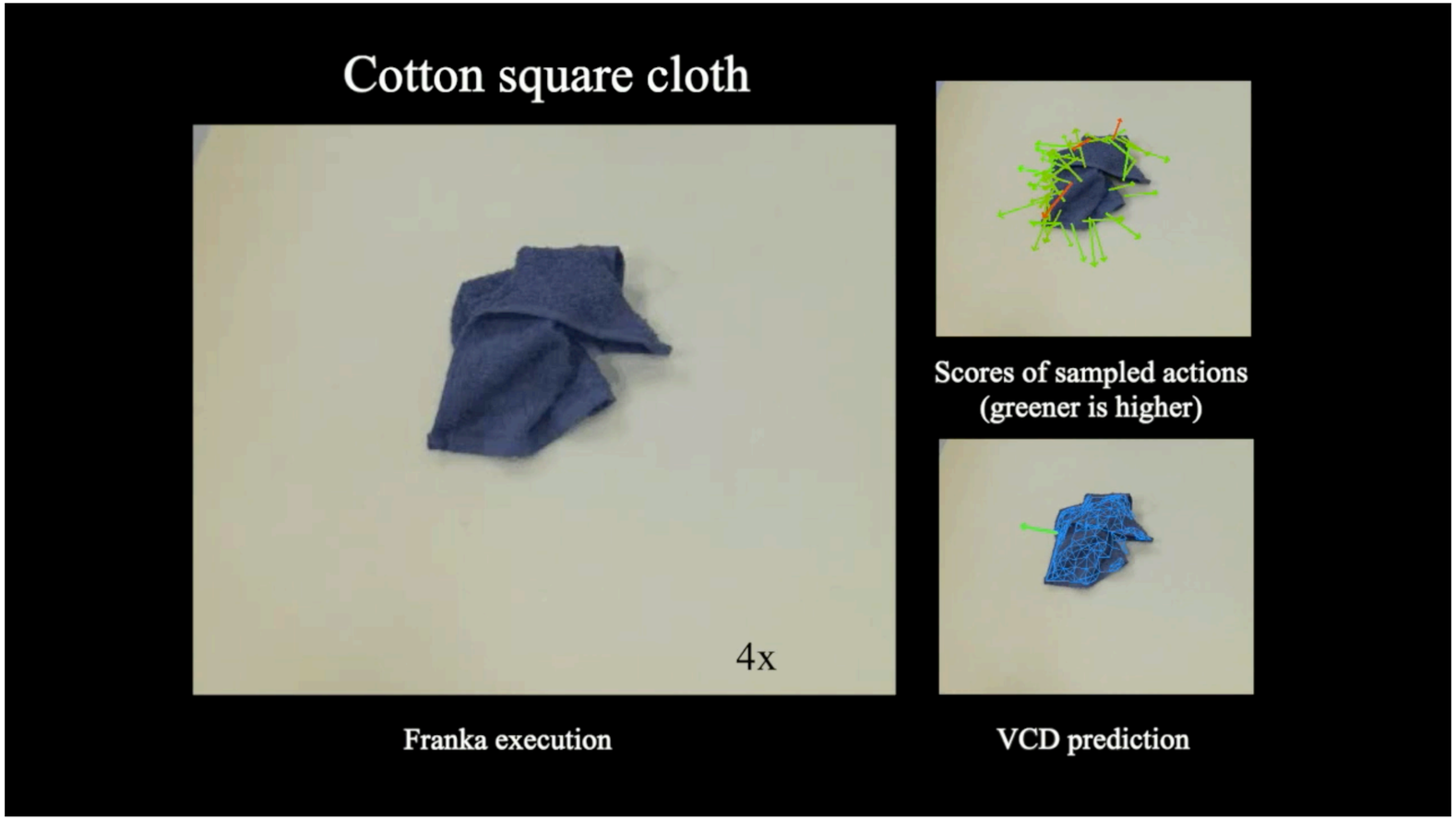
- Clearly, pick and place applies to ‘rigid’ objects, such as blocks:



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

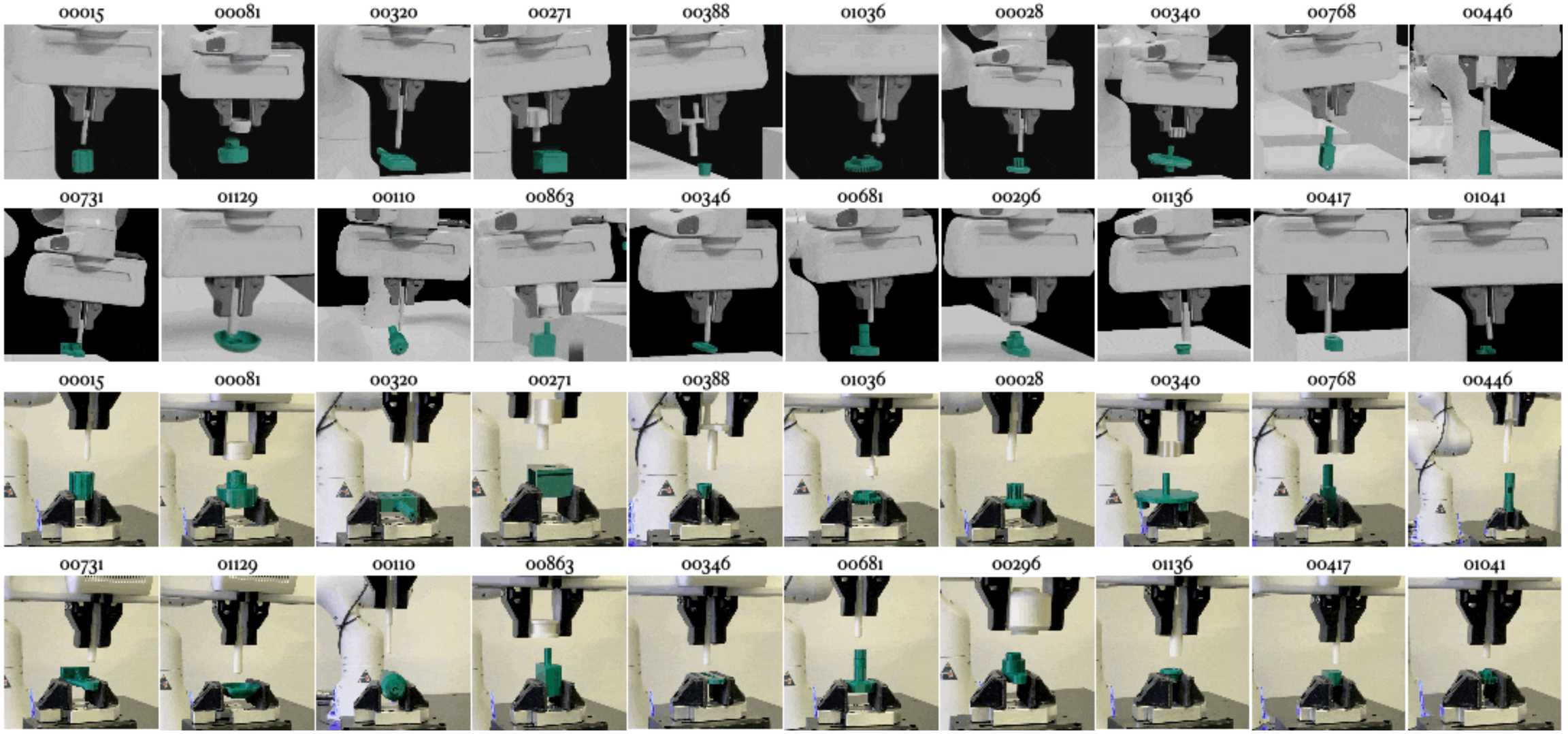
Pick and place applications

- Does pick and place apply to deformable objects?



Xingyu Lin*, Yufei Wang*, Zixuan Huang, David Held, Learning Visible Connectivity Dynamics for Cloth Smoothing, CoRL 2021.

Can pick and place be challenging?



AutoMate: Specialist and Generalist Assembly Policies over Diverse Geometries

Can pick and place be challenging?

Deep Reactive Policy: Learning Reactive Manipulator Motion Planning for Dynamic Environments

Pick and place assumptions

- Each object is attached either to the fixed world frame or the end effector frame.
- Pick and place manipulation assumes:
  - `CLOSE` attaches the object rigidly to the effector
  - `OPEN` attaches the object to the fixed world frame
  - The robot follows the path exactly.

- There can be challenges in each of the assumptions:
  - Grasping: producing a stable grasp.
  - Placing: gracefully placing an object in a stable pose
  - Control error: sensor monitoring, robust actions, compliant motions are required.

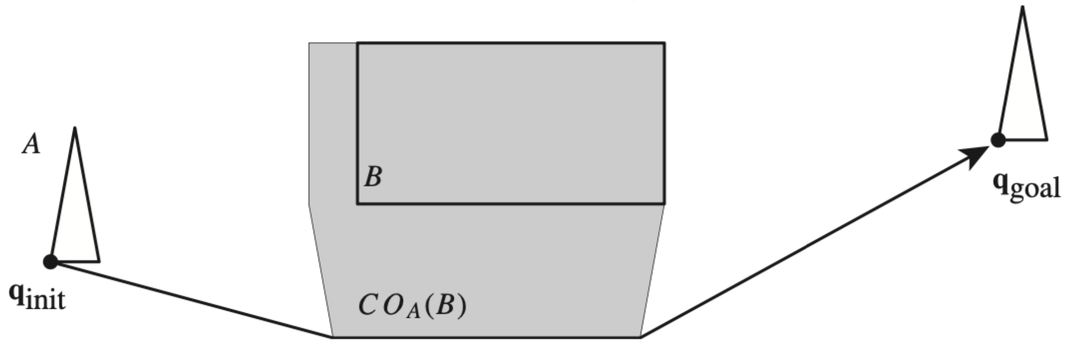
Pick and place assumptions

- Pick and place is very prominent in table-top manipulation
- Many (not all) pick and place implementations assume no obstacles and linear motion between pick and place locations.

- How can we plan paths around obstacles?

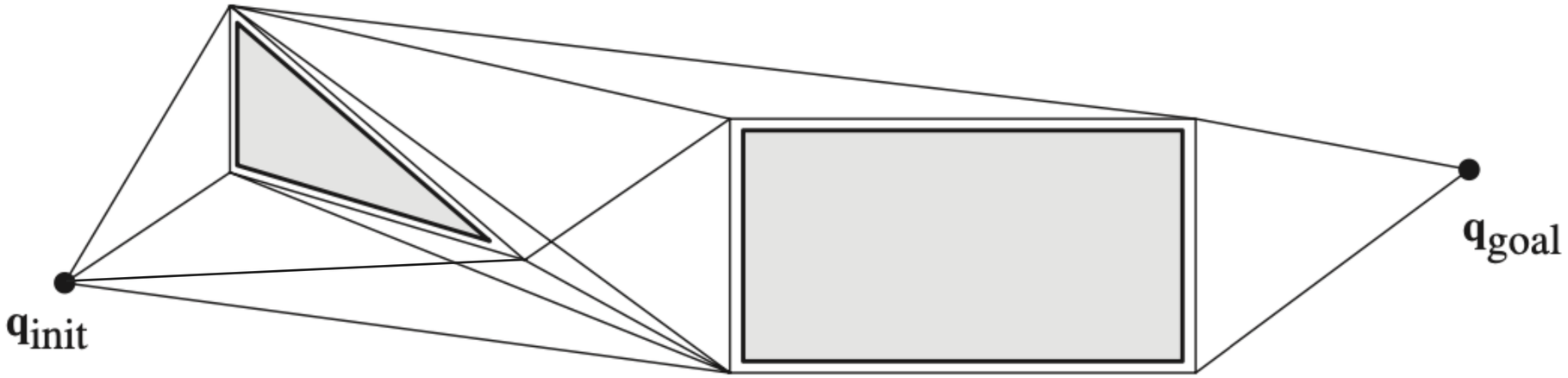
The configuration space (Cspace) transform

- Let $\boldsymbol{q}$ be the configuration of some moving object A;
- Let $\boldsymbol{Q}$ be the set of all $\boldsymbol{q}$, the *configuration space*;
- Let B be some fixed object/obstacle;
- Let $CO_A(B)$ be the set of all $\boldsymbol{q}$ such that A at $\boldsymbol{q}$ intersects B.
- $CO_A(B)$ is the **Cspace obstacle**.
- *Path planning* (collision avoidance) often means finding a path $\boldsymbol{q}(t)$ from $\boldsymbol{q}_{init}$ to $\boldsymbol{q}_{goal}$ without passing through the interior of $CO_A(B)$.



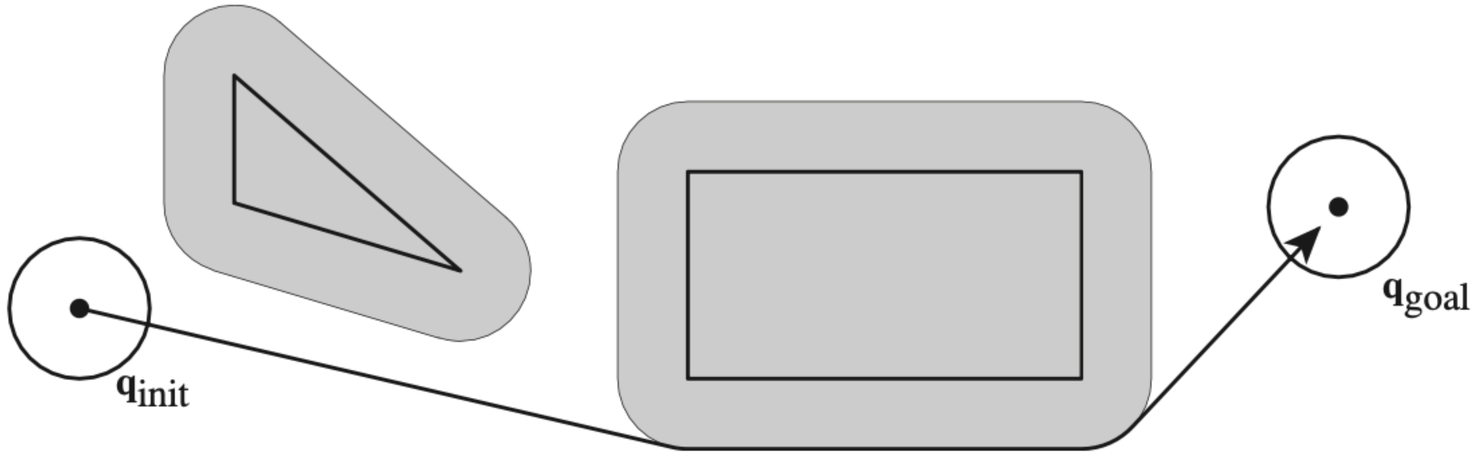
Example 1: point in plane

- The moving object A is a point (no volume) in the plane;
- The fixed objects B_i are polygons;
- Define **visibility graph**: for every vertex pair, include the line segment if it is in free space;
- Search the visibility graph… (how?)
- How would you find any valid path? How about shortest path?
- BFS, DFS, Dijkstra's, A*, bidirectional, etc.



Example 2: disk in plane

- Especially good for cylindrical indoor mobile robots (think Roomba)!
- The moving object A is a disk in the plane;
- To avoid collision, the center of the disk must be no closer than a distance r to any obstacle.
- Question: how could we use the approach from Example 1 here?
- We can expand each obstacle by r.
- Visibility graph includes all bi-tangents in free space.



Example 3: translating polygon in plane

- We represent the polygon's translation by position $\boldsymbol{q} = (x, y)$.

$$\text{collision} \iff \exists_{\boldsymbol{a}\in A, \boldsymbol{b}\in B} \boldsymbol{a} + \boldsymbol{q} = \boldsymbol{b}$$

- Interpretation: Collision iff there is a point "a" on object A that when translated by q, will intersect with a point b on object B.
- The Cspace obstacle is

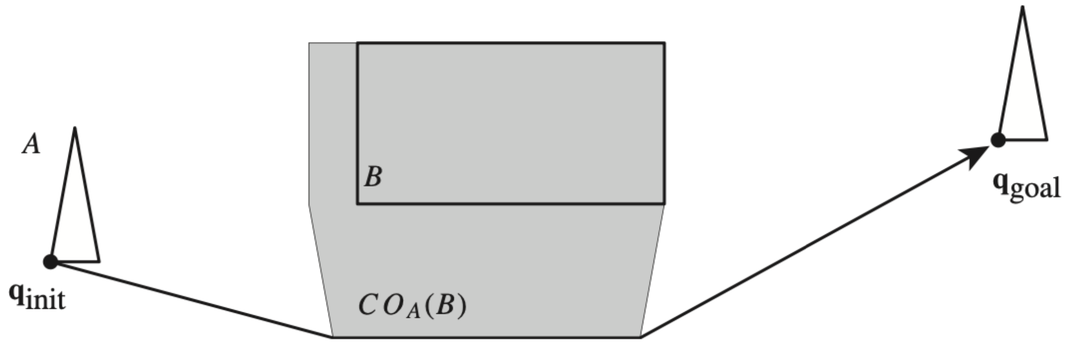
$$\begin{aligned} CO_A(B) &= \{\boldsymbol{q} \mid \exists_{\boldsymbol{a}\in A, \boldsymbol{b}\in B} \boldsymbol{a} + \boldsymbol{q} = \boldsymbol{b}\} \\ &= \{\boldsymbol{b} - \boldsymbol{a} \mid \boldsymbol{a} \in A, \boldsymbol{b} \in B\} \\ &= B \ominus A \end{aligned}$$

Question: How would $CO_A(B)$ change if we allowed rotation?

- ("$\ominus$" is called *Minkowski difference*) For A and B convex:

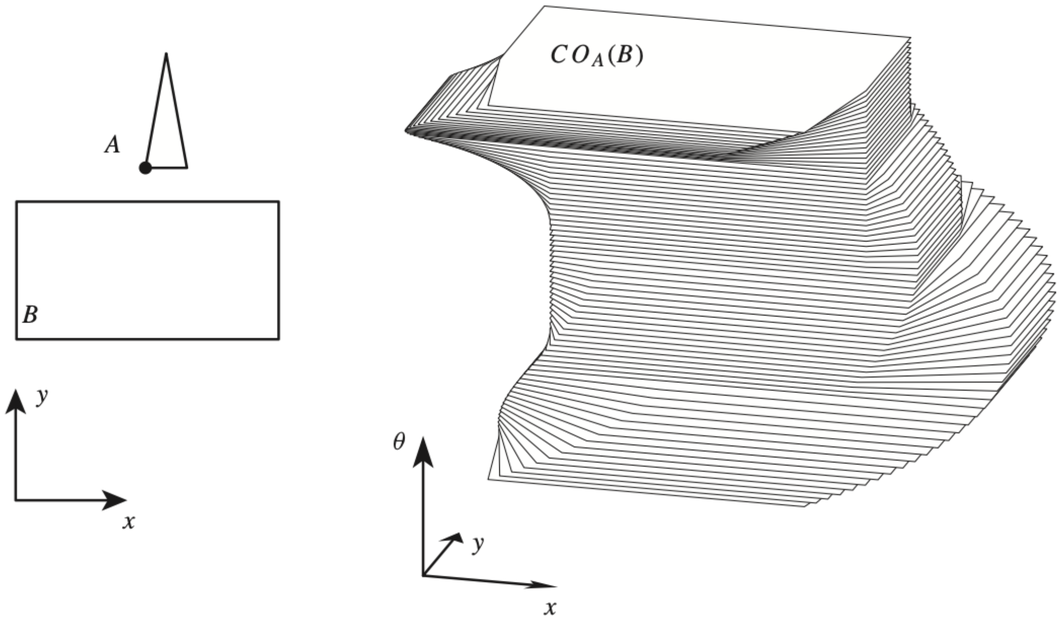
$$CO_A(B) = \text{conv}(\text{vert}(B) \ominus \text{vert}(A))$$

- conv(…) is the convex hull, which can informally be visualized on the 2D plane as a rubber band around an object, or in 3D as a piece of paper crumpled around the object.



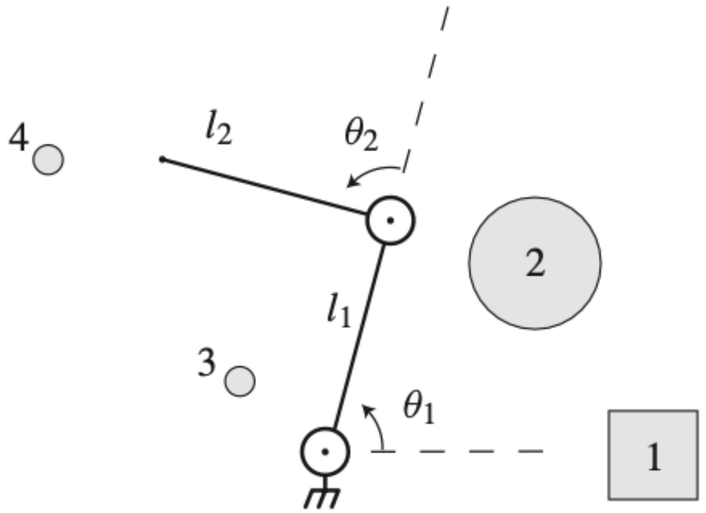
Example 4: planar polygon with rotation

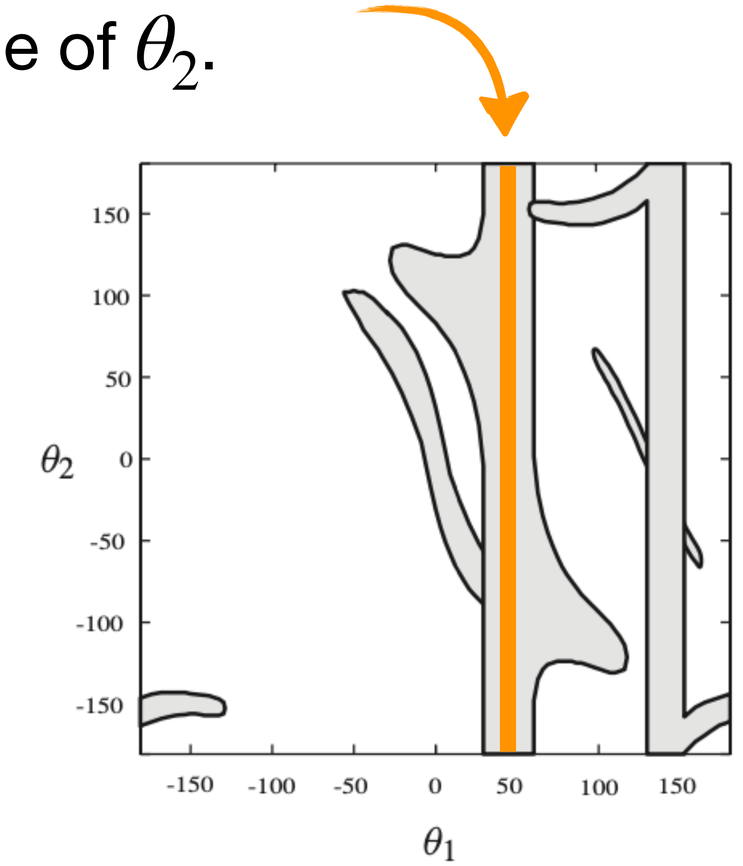
- If we allow the triangle to rotate, the configuration space is 3D.
- Now $\boldsymbol{q} = (x, y, \theta)$.
- Figure on the right shows $CO_A(B)$ at every 5-degree rotation of A.
- This shows all the values of $\boldsymbol{q}$ in which A would intersect with B.



Example 5: two link planar arm

- Two joint angles $\boldsymbol{q} = (\theta_1, \theta_2)$.
- Take note of areas such as $\theta_1 = 50°$, which would cause the robot to intersect object 2, no matter the value of θ_2.





What do we do with the Cspace transform?

- Options:
    1. Transform every path planning problem, and search for collision-free paths in Cspace;
    2. Use it for insight and analysis, but use a variety of techniques depending on the problem.
- Note, the Cspace transform is challenging to implement and use except in the simplest cases.

Path planning

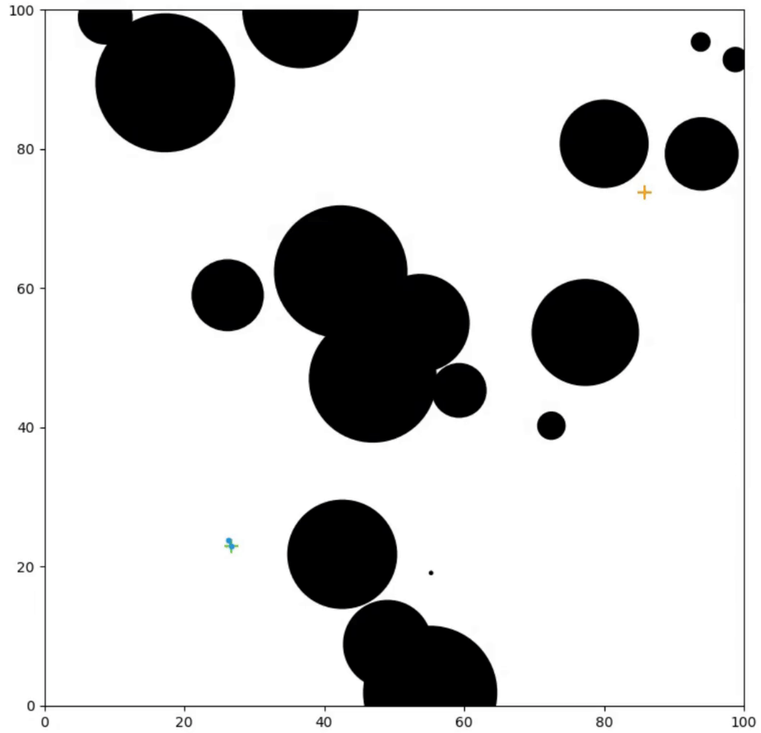
- Visibility graph works well in two dimensions, but not three or more. Shortest paths don’t pass through vertices (simple counterexamples).
- Potential fields have many uses. (See early work by Oussama Khatib)
- Voronoi graphs have been generalized to higher dimensions. (See for example work by Howie Choset)
  - Visuals: https://www.usna.edu/Users/cs/crabbe/SI475/current/simple-model/path_planning/voronoi.pdf
- Cellular decomposition of the free space. (See for example the original Cspace work by Lozano-Perez and Wesley)
- **Best first planner**. Search on a grid in Cspace and use a potential field as a heuristic to guide the search.
- Many other options: probabilistic roadmap, **rapidly exploring random trees**, and sampling-based planning.

Rapidly exploring random trees

- Randomly build a space-filling tree.
  Goal: get as close as possible to mapping every configuration.
- Build a roadmap: a graph $G(V, E)$, where every vertex in V is a configuration and every edge in E is a path connecting two configurations.

```
G.INIT(q0)
LOOP
    a = SAMPLE
    qn = NEAREST(G, a)
    G.ADD.VERTEX(a)
    G.ADD.EDGE(qn, a)
```

- Probabilistically complete as $n \to \infty$.



https://riley-knox.github.io/rrt/

What about obstacles?

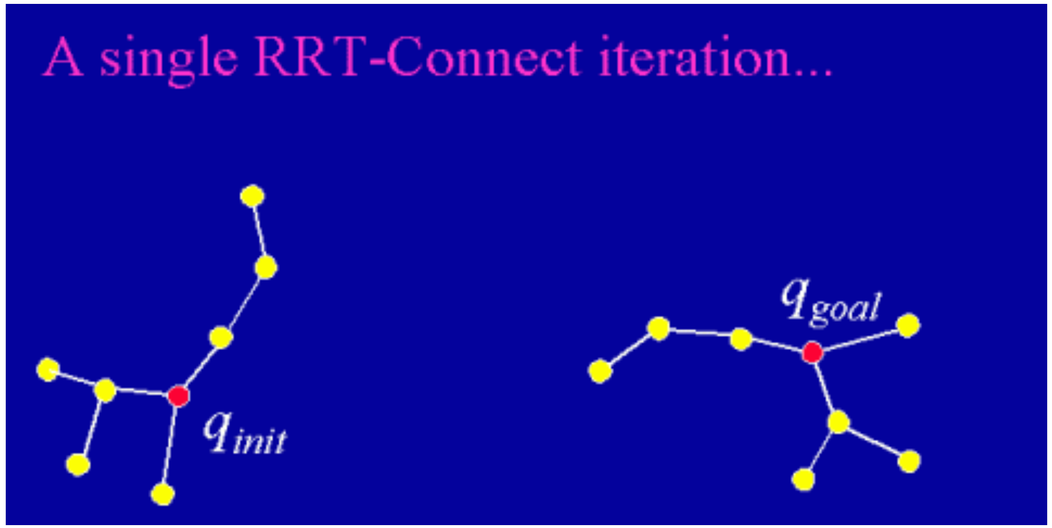
```
G.INIT(q0)
LOOP
   a = SAMPLE
   qn = NEAREST(G, a)
   qs = STOPPING(G, a)
   G.ADD.VERTEX(qs)
   G.ADD.EDGE(qn, qs)
```

What about goals?

- Option 1: sample the goal with some probability p.
  But, unidirectional trees are slow.
- Option 2: bidirectional trees: start one tree from init and one tree from goal!

RRT-Connect

RRT_CONNECT_PLANNER(q_{init}, q_{goal})
1 $\mathcal{T}_a$.init(q_{init}); $\mathcal{T}_b$.init(q_{goal});
2 **for** $k = 1$ **to** K **do**
3 $q_{rand} \leftarrow$ RANDOM_CONFIG();
4 **if not** (EXTEND($\mathcal{T}_a, q_{rand}$) = *Trapped*) **then**
5 **if** (CONNECT($\mathcal{T}_b, q_{new}$) = *Reached*) **then**
6 Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
7 SWAP($\mathcal{T}_a, \mathcal{T}_b$);
8 Return *Failure*



Algo from: http://www.kuffner.org/james/papers/kuffner_icra2000.pdf

Gif animation from: http://www.kuffner.org/james/plan/algorithm.php

Issues?

- Smoothing: the dark secret of RRTs.
  - Shortcut path shortening: pick two points. Try to connect them.
- Local planning: how to go from one configuration to another. E.g. steered car, physical constraints, etc. might produce issues.
  (We may not be able to connect two nearest neighbors)
- Nearest neighbor: we can punish bad nodes with weighted nearest neighbor
- Sampling: Should we sample towards obstacle, away from obstacles, under/over obstacles?

OMPL

Open Motion Planning Library

Already integrated into MoveIt 2 and ROS 2 (see next slide)

MoveIt 2 uses RRT-Connect by default

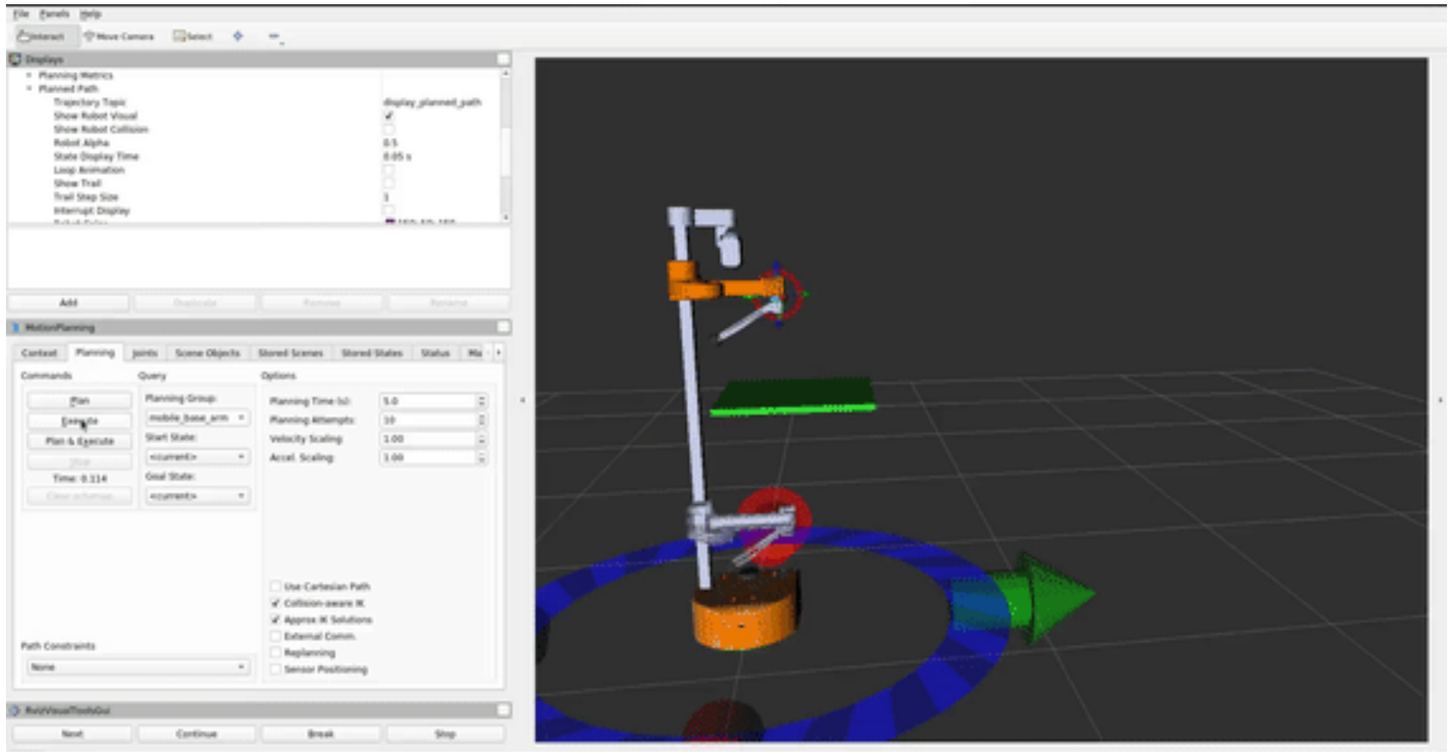
See the gallery:

https://ompl.kavrakilab.org/gallery.html

MoveIt 2 Interface

```
stretch_free_robot_process.py
stretch_robot_home.py
ros2 launch stretch_moveit2 movegroup_moveit2_stretch3.launch.py
```

Using this interface: https://docs.hello-robot.com/0.2/stretch-tutorials/ros2/moveit_rviz_demo/



MoveIt 2 Installation (on a new robot)

I have already installed MoveIt 2 on our Stretch robots for you.

For your reference, here is how to do it: https://github.com/Zackory/stretch3_moveit2