Motion Planning: Algorithms

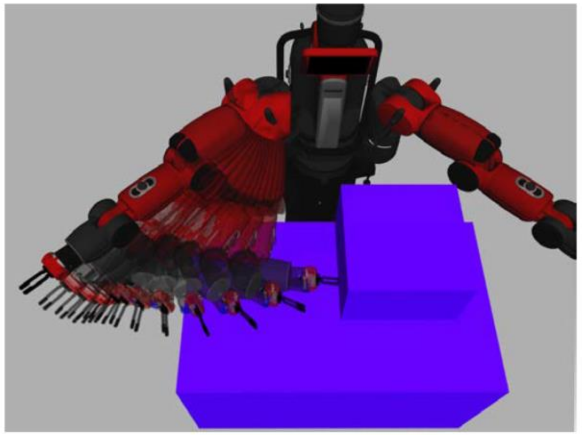
CS 6301 Special Topics: Introduction to Robot Manipulation and Navigation

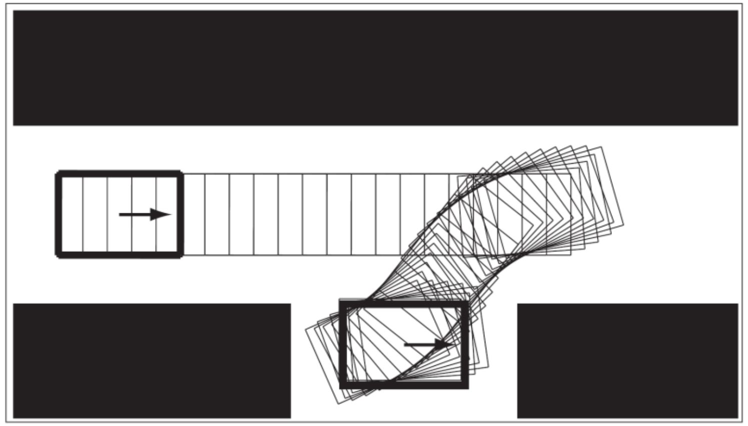
Professor Yu Xiang

The University of Texas at Dallas

Motion Planning

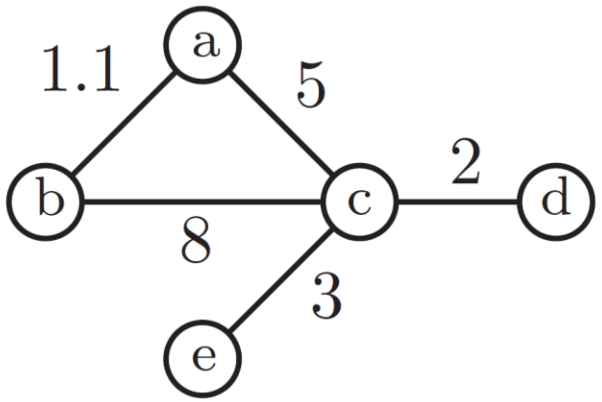
- Motion planning: finding a robot motion from a start state to a goal state (A to B)
  - Avoids obstacles
  - Satisfies other constraints such as joint limits or torque limits
- Path planning is a purely geometric problem of finding a collision-free path





A* Search Algorithm

- Finds a minimum-cost path on a graph
- Cost: sum of the positive edge costs along the path
- Data structures used
  - OPEN: a list of nodes not explored yet
  - CLOSE: a list of nodes explored already
  - cost[node1, node2]: positive, edge cost, negative, no edge
  - past_cost[node]: minimum cost found so far to reach node from the start node
  - parent[node]: a link to the node preceding it in the shortest path found so far



A* Search Algorithm

- Initialization
  - The matrix cost is constructed to encode the edges
  - OPEN is the start node 1
  - past_cost[1] = 0, past_cost[node] = infinity
- At each step
  - Remove the first node from OPEN and call it current
  - The node current is added to CLOSE
  - If current in the goal set, finished
  - Otherwise, for each neighbor of current that is not in CLOSE, compute

```
  tentative_past_cost
= past_cost[current] + cost[current,nbr]
```

A* Search Algorithm

- At each step (continued)
  - If `tentative_past_cost < past_cost[nbr]` — Found a shorter path

    `past_cost[nbr] = tentative_past_cost`

    `parent[nbr]` is set to `current`

    Compute estimated total cost for nbr

    `est_total_cost[nbr]` $\leftarrow$ `past_cost[nbr] +`
    `heuristic_cost_to_go(nbr)`

    Add nbr to the correct position in OPEN (a sorted list)

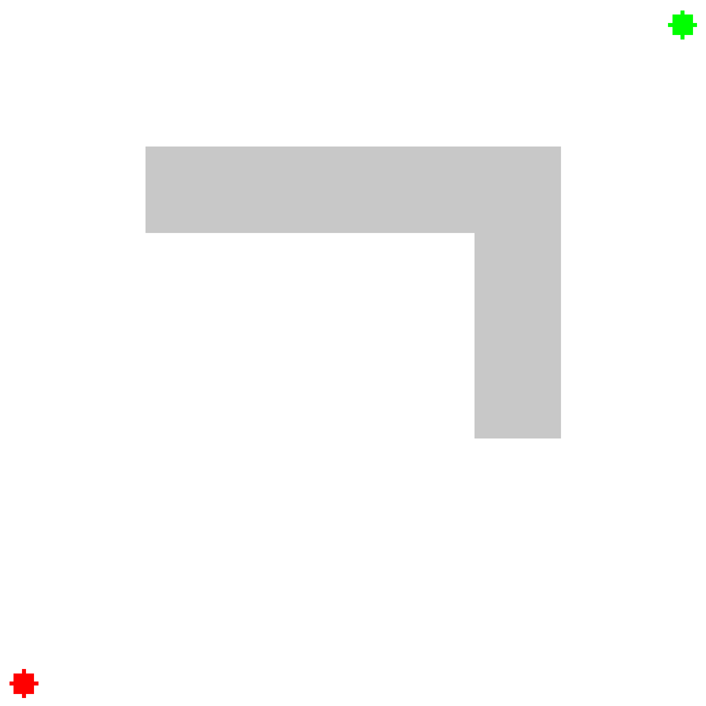
A* Search Algorithm

Algorithm 10.1 A^* search.

```
OPEN ← {1}
past_cost[1] ← 0, past_cost[node] ← infinity for node ∈ {2, . . . , N}
while OPEN is not empty do
  current ← first node in OPEN, remove from OPEN
  add current to CLOSED
  if current is in the goal set then
    return SUCCESS and the path to current
  end if
  for each nbr of current not in CLOSED do
    tentative_past_cost ← past_cost[current]+cost[current,nbr]
    if tentative_past_cost < past_cost[nbr] then
      past_cost[nbr] ← tentative_past_cost
      parent[nbr] ← current
      put (or move) nbr in sorted list OPEN according to
          est_total_cost[nbr] ← past_cost[nbr] +
                  heuristic_cost_to_go(nbr)
    end if
  end for
end while
return FAILURE
```

- Guaranteed to return a minimum-cost path
- Best-first searches

A* Search Algorithm

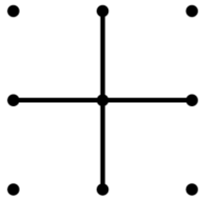


The empty circles represent the nodes in the *open set*, i.e., those that remain to be explored, and the filled ones are in the closed set. Color on each closed node indicates the distance from the goal: the greener, the closer. One can first see the A* moving in a straight line in the direction of the goal, then when hitting the obstacle, it explores alternative routes through the nodes from the open set.

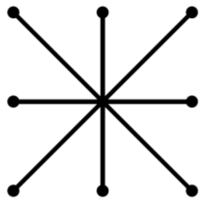
https://en.wikipedia.org/wiki/A*_search_algorithm

Grid Methods

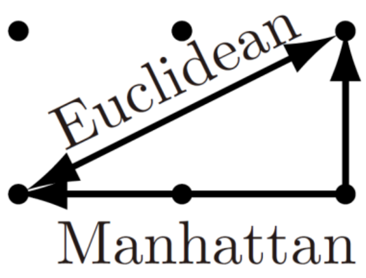
- Discretize the configuration space into a grid
  - If the C-space is n dimension, we use k grid points along each dimension
  - The C-space is represented by k^n grid points
- We can apply the A* search algorithm for path planning with a C-space grid
  - Define the neighbors of a grid point
  - If only axis-aligned motions are used, the heuristic cost-to-go should be based on Manhattan distance
  - A node nbr is added to OPEN only if the step from current to nbr is collision-free



4-connected

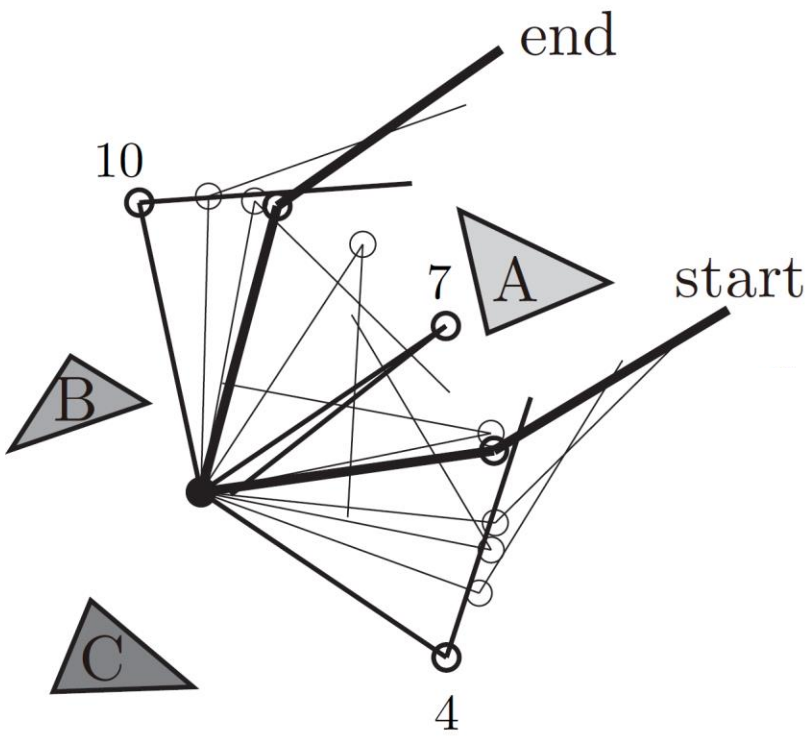


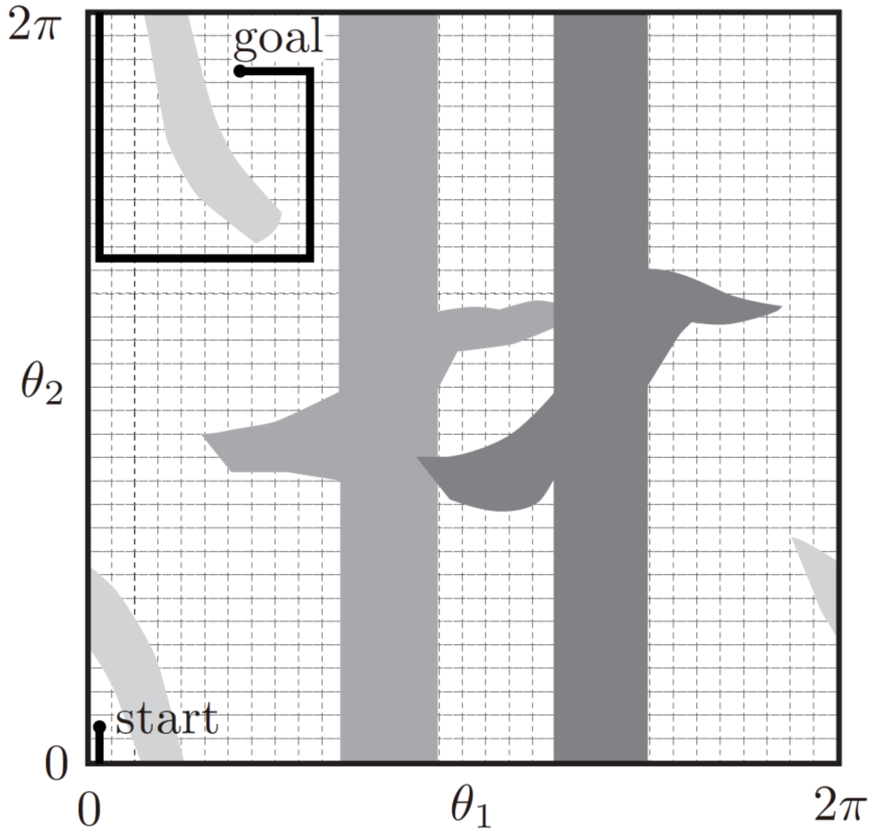
8-connected



Grid Methods

- A* grid-based path planner





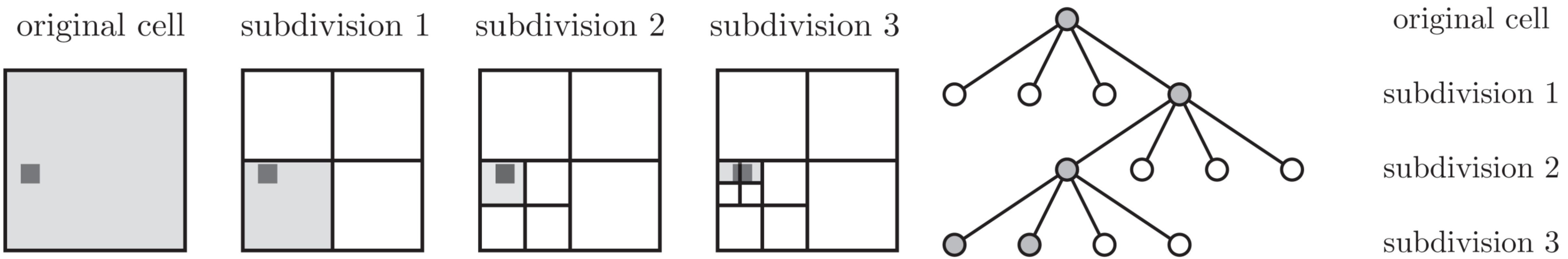
$k = 32$

Grid Methods

- Grid-based path planning is only suitable for low-dimensional C-space
  - Number of grid points k^n

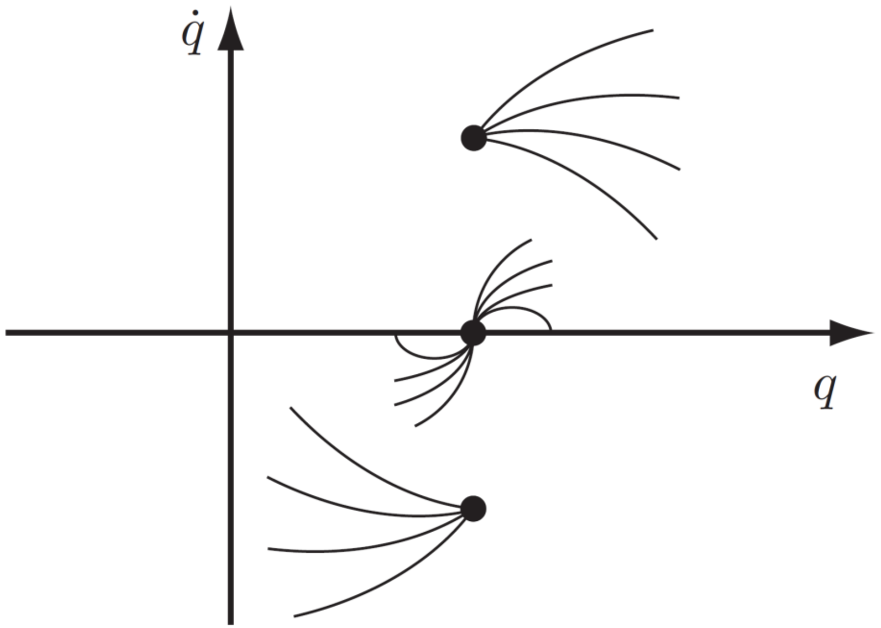
```
>>> np.power(32, 7.0)
34359738368.0
```

- Multi-resolution grid representation



Grid Methods with Motion Constraints

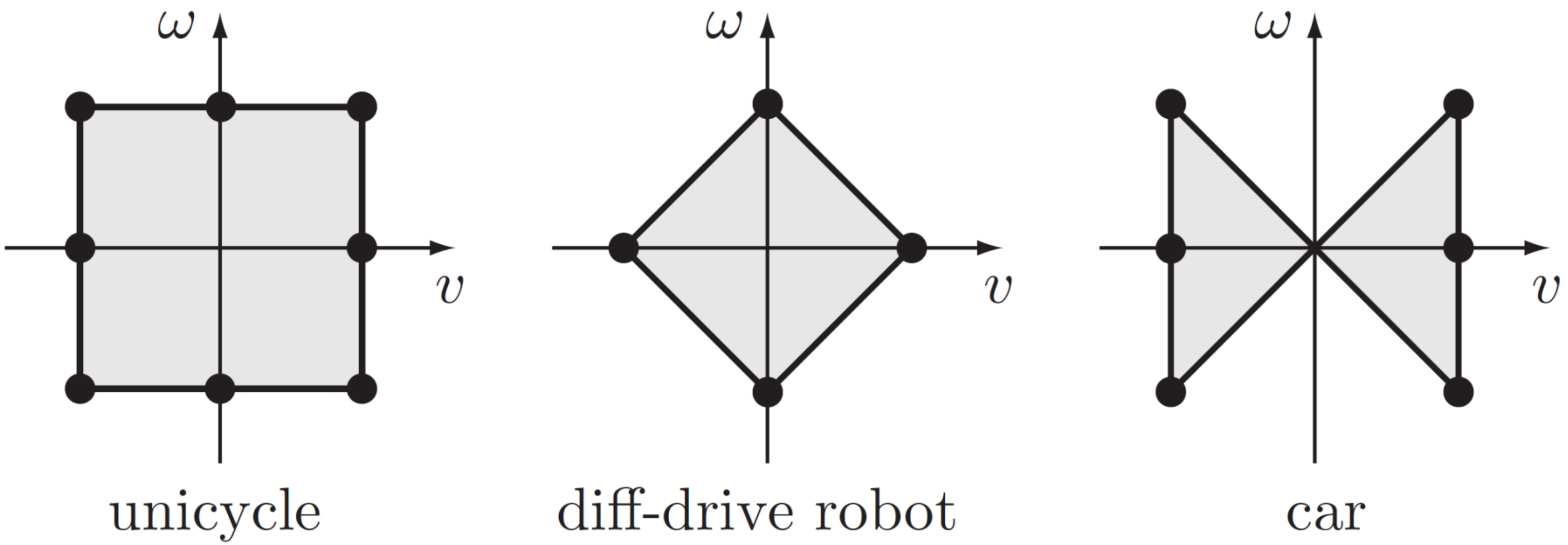
- A robot may not be able to reach all the neighbors in a grid
  - A car cannot move to the side
  - motions for a fast-moving robot arm should be planned in the state space, not just in the C-space



Sample trajectories emanating from three initial states in the phase space of a dynamic system

Grid Methods with Motion Constraints

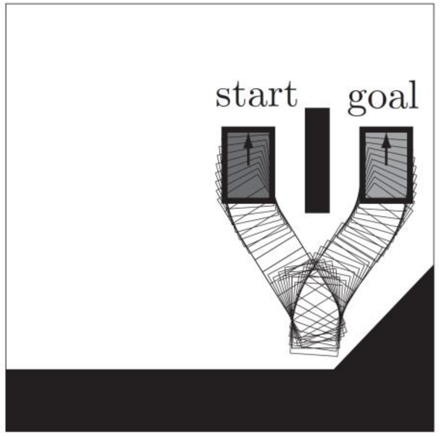
- Control for mobile robot (v, ω)
  - v: forward-backward linear velocity
  - w: angular velocity

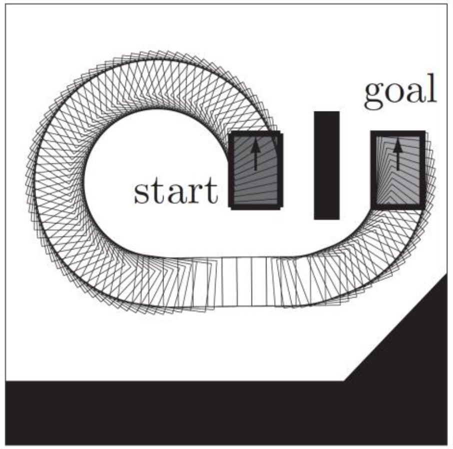


Grid Methods with Motion Constraints

Algorithm 10.2 Grid-based Dijkstra planner for a wheeled mobile robot.

OPEN $\leftarrow \{q_{\text{start}}\}$
past_cost[q_{start}] $\leftarrow 0$
counter $\leftarrow 1$
while OPEN is not empty and counter < MAXCOUNT **do**
  current $\leftarrow$ first node in OPEN, remove from OPEN
  if current is in the goal set **then**
    return SUCCESS and the path to current
  end if
  if current is not in a previously occupied C-space grid cell **then**
    mark grid cell occupied
    counter $\leftarrow$ counter + 1
    for each control in the discrete control set **do**
      integrate control forward a short time Δt from current to q_{new}
      if the path to q_{new} is collision-free **then**
        compute cost of the path to q_{new}
        place q_{new} in OPEN, sorted by cost
        parent[q_{new}] $\leftarrow$ current
      end if
    end for
  end if
end while
return FAILURE

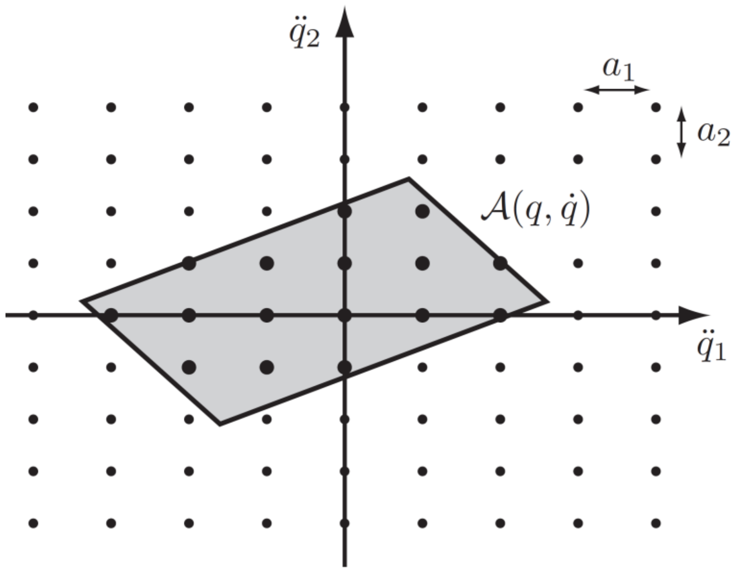




Reversals are penalized

Grid Methods with Motion Constraints

- For a robot arm, we can plan directly in the state space $(q, \dot{q})$
- Let $\mathcal{A}(q, \dot{q})$ represent the set of accelerations that are feasible on the basis of the limited joint torques
- Discretization
- Apply a breath-first search in the state space
  - To find a trajectory from a start state to a goal
  - When exploration is made from $(q, \dot{q})$
  - Use $\mathcal{A}(q, \dot{q})$ to find the control actions
  - Integrate the control actions for Δt



Sampling Methods

- Grid-based methods delivers optimal solutions subject to the chosen discretization, but computationally expensive for high DOFs

- Sampling methods
  - Randomly or deterministically sampling the C-space or state-space to find the motion plan
  - Give up resolution-optimal solutions of a grid search, quickly find solutions in high-dimensional state space
  - Most sampling methods are probabilistically complete: the probability of finding a solution, when one exists, approaches 100% as the number of samples goes to infinity

Rapidly exploring Random Trees (RRTs)

Algorithm 10.3 RRT algorithm.

1: initialize search tree T with x_{start}
2: **while** T is less than the maximum tree size **do**
3: $\quad x_{\text{samp}} \leftarrow$ sample from $\mathcal{X}$
4: $\quad x_{\text{nearest}} \leftarrow$ nearest node in T to x_{samp}
5: $\quad$ employ a local planner to find a motion from x_{nearest} to x_{new} in the direction of x_{samp}
6: $\quad$ **if** the motion is collision-free **then**
7: $\qquad$ add x_{new} to T with an edge from x_{nearest} to x_{new}
8: $\qquad$ **if** x_{new} is in $\mathcal{X}_{\text{goal}}$ **then**
9: $\qquad\quad$ **return** SUCCESS and the motion to x_{new}
10: $\qquad$ **end if**
11: $\quad$ **end if**
12: **end while**
13: **return** FAILURE

kinematic problems

$$x = q$$

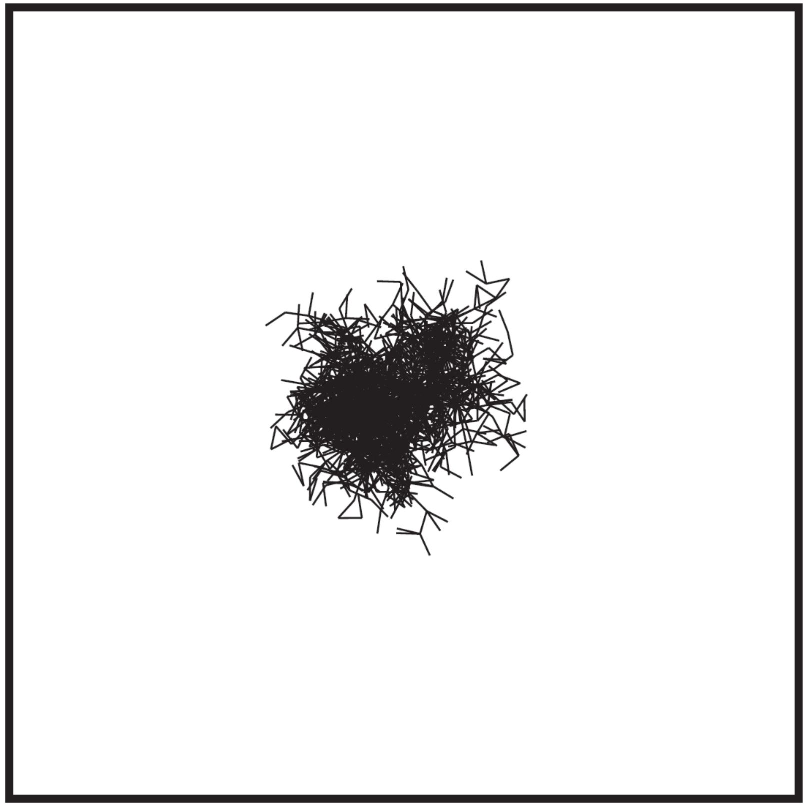
- Line 3, uniform sampling with a bias towards goal
- Line 4, Euclidean distance
- Line 5, use a small distance d from

x_{nearest} on the straight line to x_{samp}

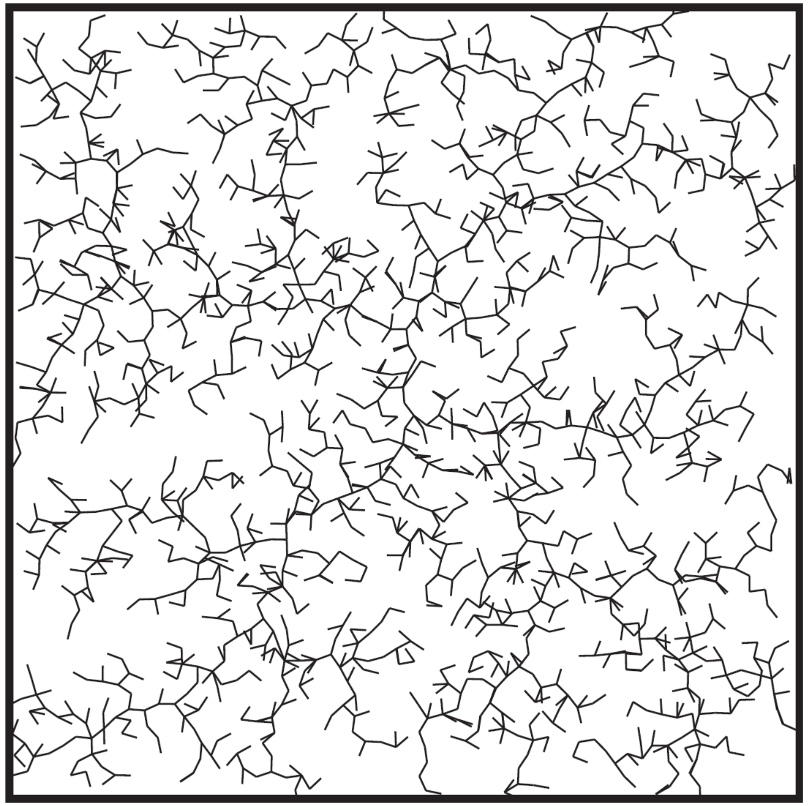
dynamic problems

$$x = (q, \dot{q})$$

Rapidly exploring Random Trees (RRTs)



A tree generated by applying a uniformly-distributed random motion from a randomly chosen tree node does not explore very far.



A tree generated by the RRT algorithm

2000 nodes

Rapidly exploring Random Trees (RRTs)

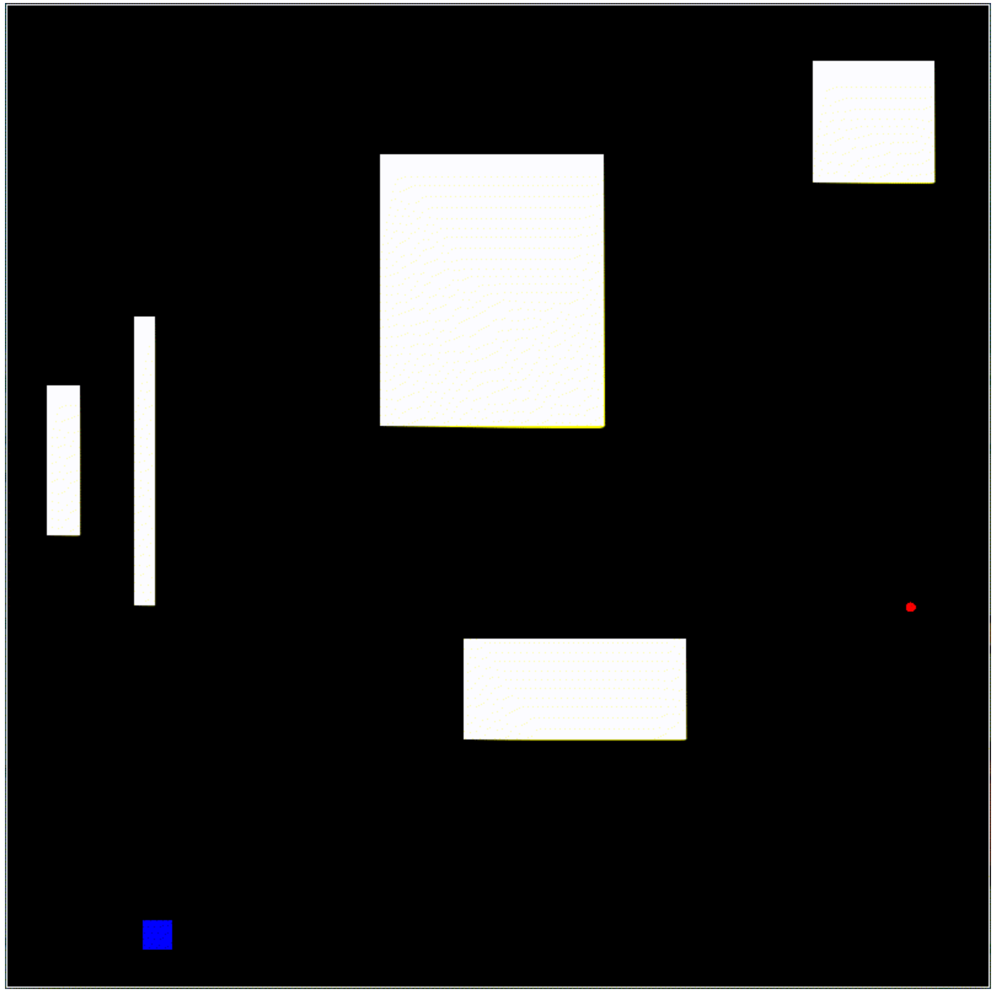
An animation of an RRT starting from iteration 0 to 10000

https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree

Rapidly exploring Random Trees (RRTs)

- Bidirectional RRT
  - Grows two trees, one forward from x_{start} , one backward from x_{goal}
  - Alternating between growing the two trees x_{samp}
  - Trying to connect the two trees by choosing x_{goal} from the other tree
  - Con: faster, can reach the exact goal
  - Pro: the local planer might not be able to connect the two trees

Bidirectional RRT

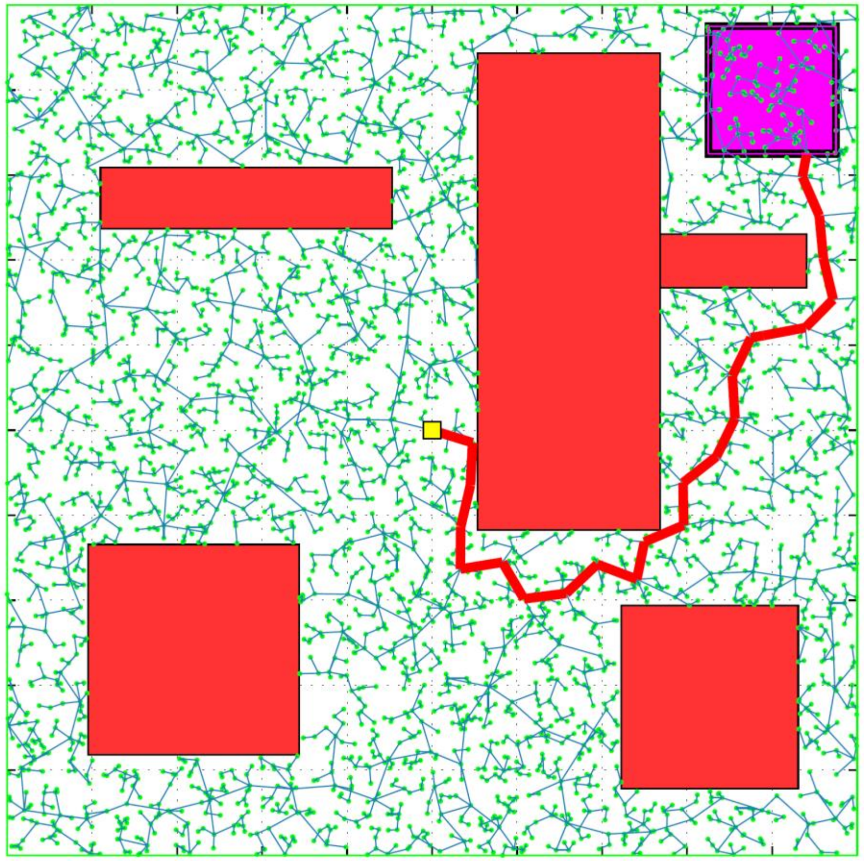


https://github.com/JakeInit/RRT

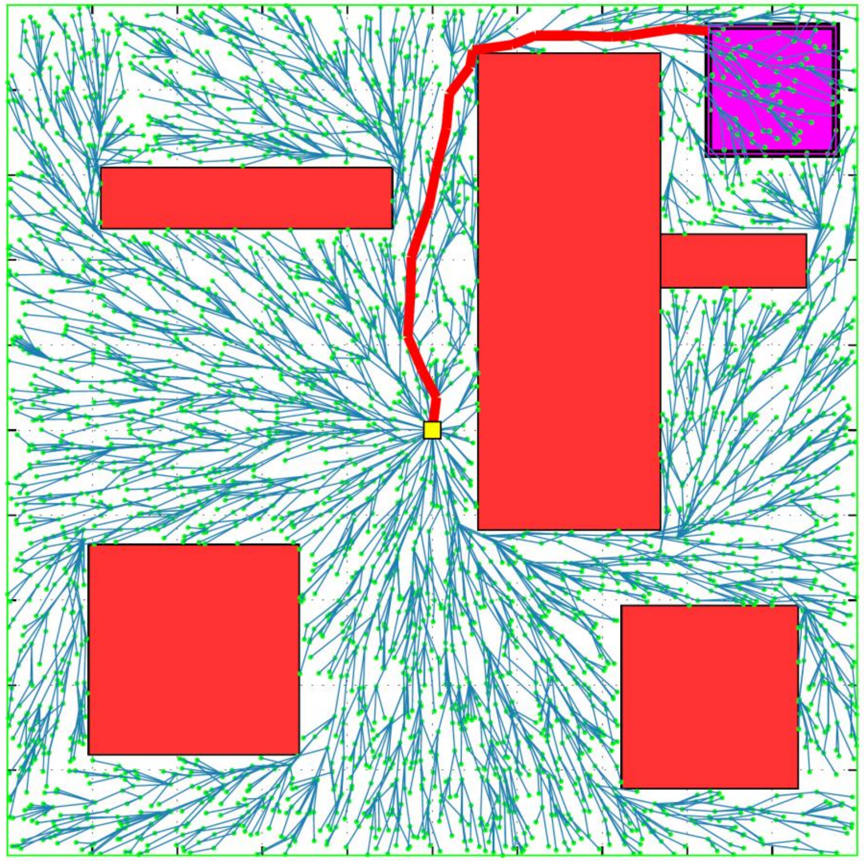
Rapidly exploring Random Trees (RRTs)

- RRT*
  - Continually rewires the search tree to ensure that it always encodes the shortest path from x_{start} to each node in the tree
  - To insert x_{new} to the tree, consider $x \in \mathcal{X}_{\text{near}}$ $\text{sufficiently near to } x_{\text{new}}$
    - Collision free
    - Minimizes the total cost from x_{start} to x_{new}
  - Consider each $x \in \mathcal{X}_{\text{near}}$ to see whether it could be reached at lower cost by a motion through x_{new} , change the parent of x to x_{new} (rewiring)

RRT vs. RRT*



RRT



RRT*

Probabilistic Roadmaps (PRMs)

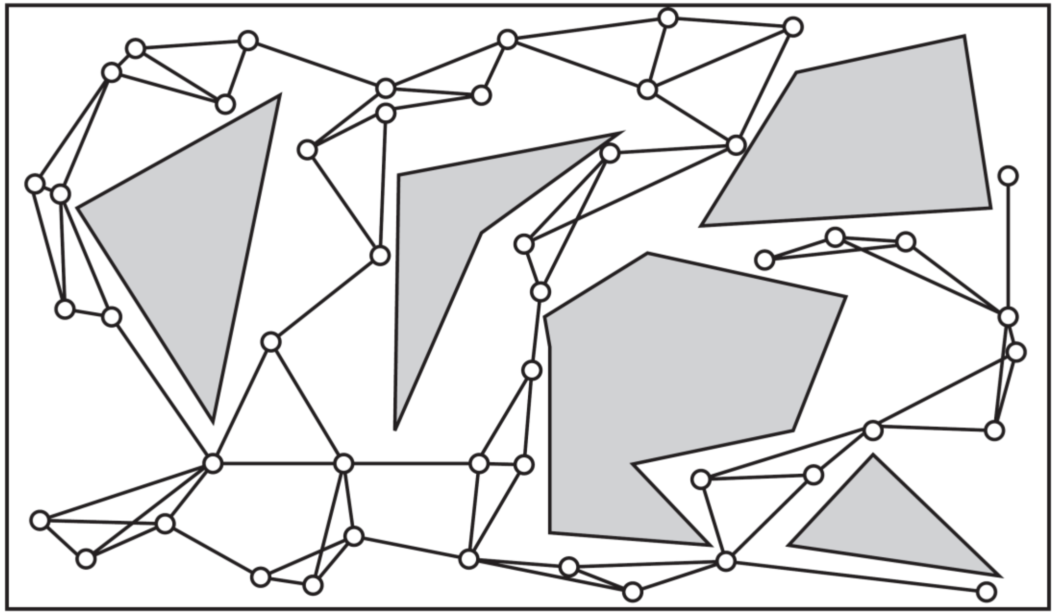
- PRM uses sampling to build a roadmap representation of $\mathcal{C}_{\mathrm{free}}$
- Connect a start node q_{start} and a goal node q_{goal} to the roadmap
- Search for a path, e.g., using A*

Probabilistic Roadmaps (PRMs)

- PRM uses sampling to build a roadmap representation of $\mathcal{C}_{\text{free}}$

Algorithm 10.4 PRM roadmap construction algorithm (undirected graph).

for $i = 1, \ldots, N$ **do**
  $q_i \leftarrow$ sample from $\mathcal{C}_{\text{free}}$
  add q_i to R
end for
for $i = 1, \ldots, N$ **do**
  $\mathcal{N}(q_i) \leftarrow k$ closest neighbors of q_i
  for each $q \in \mathcal{N}(q_i)$ **do**
    if there is a collision-free local path from q to q_i and there is not already an edge from q to q_i **then**
      add an edge from q to q_i to the roadmap R
    end if
  end for
end for
return R



Nonlinear Optimization

- The general motion planning problem

$$
\begin{aligned}
\text{find} \quad & u(t), q(t), T \\
\text{minimizing} \quad & J(u(t), q(t), T) \\
\text{subject to} \quad & \dot{x}(t) = f(x(t), u(t)), && \forall t \in [0, T], \\
& u(t) \in \mathcal{U}, && \forall t \in [0, T], \\
& q(t) \in \mathcal{C}_{\text{free}}, && \forall t \in [0, T], \\
& x(0) = x_{\text{start}}, \\
& x(T) = x_{\text{goal}}.
\end{aligned}
$$

Smoothing cost function

$$J = \frac{1}{2} \int_0^T \dot{u}^{\mathrm{T}}(t) \dot{u}(t) dt$$

Trajectory Optimization: CHOMP

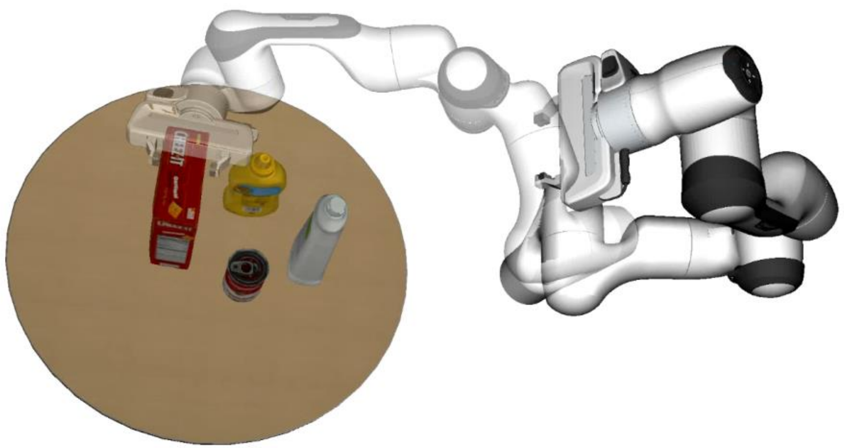
$$f_{\text{motion}}(\xi) = f_{\text{obstacle}}(\xi) + \lambda f_{\text{smooth}}(\xi)$$

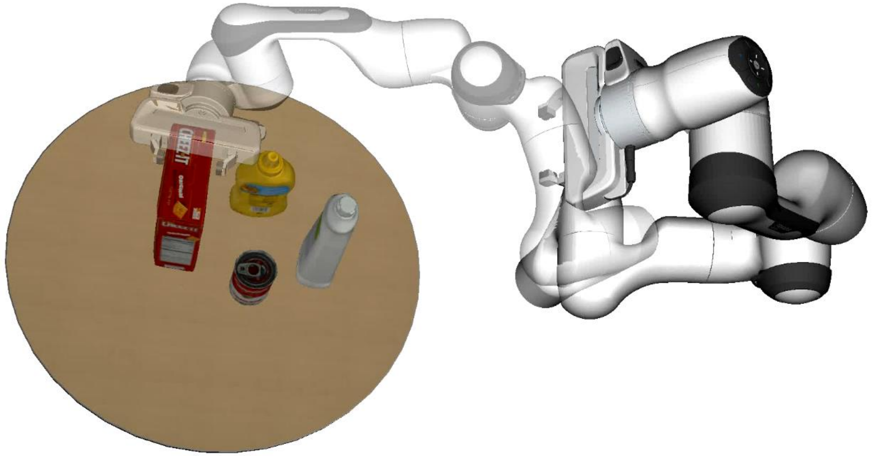
$$\xi = (q_1, \ldots, q_T)$$ A trajectory of robot joint configurations

N steps gradient descent

Initial trajectory with collision

Final trajectory



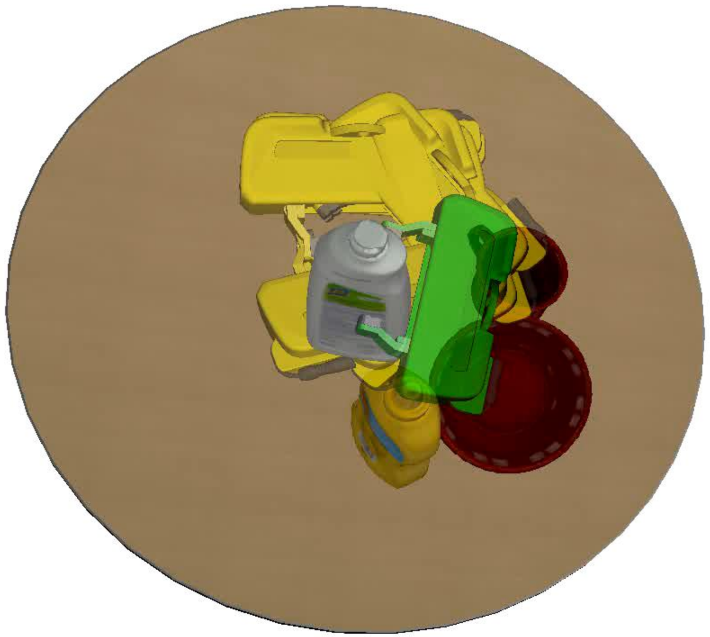


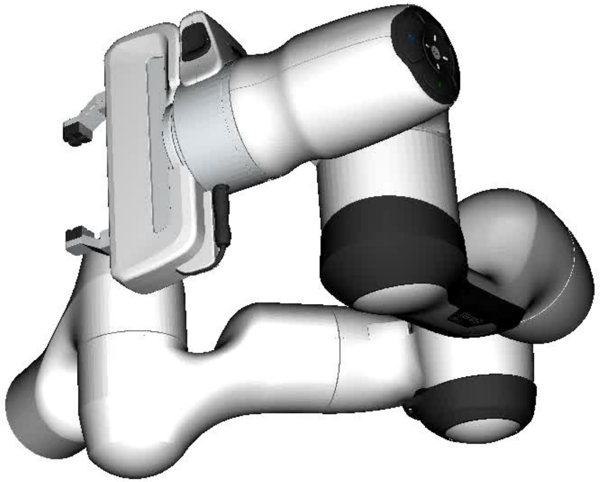
Covariant Hamiltonian Optimization for Motion Planning (CHOMP): Ratliff-Zucker-Bagnell-Srinivasa, ICRA'09

OMG Planner: Trajectory Optimization and Grasp Selection

OMG Iter: 50

100 grasps





Modeling the goal set distribution

Wang-Xiang-Fox, RSS'20

Summary

- Grid methods
  - A*

- Sampling methods
  - RRTs
  - PRMs

- Nonlinear optimization

Further Reading

- Chapter 10 in Kevin M. Lynch and Frank C. Park. Modern Robotics: Mechanics, Planning, and Control. 1st Edition, 2017.

- A* search: P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics, 4(2):100-107, July 1968.

- PRMs. L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for fast path planning in high dimensional conguration spaces. IEEE Transactions on Robotics and Automation, 12:566-580, 1996.

- RRT. S. M. LaValle and J. J. Kuner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, Algorithmic and Computational Robotics: New Directions. A. K. Peters, Natick, MA, 2001.