Motion Planning: Overview and Foundations

CS 6301 Special Topics: Introduction to Robot Manipulation and Navigation

Professor Yu Xiang

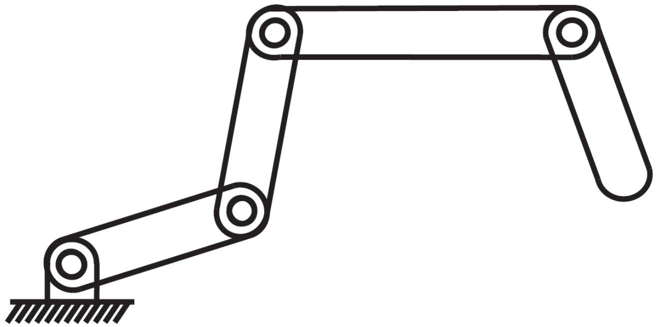
The University of Texas at Dallas

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
- Recall configuration space (C-space)

Configuration Space

- The configuration of a robot is a complete specification of the position of every point of the robot.
- The minimum number n of real-valued coordinates needed to represent the configuration is the number of degrees of freedom (DOF) of the robot.
- The n-dimensional space containing all possible configurations of the robot is called the configuration space (C-space).
- The configuration of a robot is represented by a point in its C-space.



- 4 revolute joints
- 4 DOFs

Configuration Space

- The configuration of a robot arm with n joints
  - n joint positions $q = (\theta_1, \ldots, \theta_n)$
- Free C-space $\mathcal{C}_{\text{free}}$
  - Configurations where the robot neither penetrates an obstacle nor violated a joint limit
- Robot state $x = (q, v) \in \mathcal{X}$
  - For second order dynamics, state is configuration and velocity $v = \dot{q}$
  - When using velocities as control input, state is the configuration $q(x)$

$$\mathcal{X}_{\text{free}} = \{x \mid q(x) \in \mathcal{C}_{\text{free}}\}$$

Equations of Motion

- The equations of motion of a robot

$$\dot{x} = f(x, u)$$

Forward dynamics

Robot state

Control inputs $u \in \mathcal{U} \subset \mathbb{R}^m$

- Integral form

$$x(T) = x(0) + \int_0^T f(x(t), u(t))dt$$

Motion Planning

- Given an initial state $x(0) = x_{\text{start}}$ and a desired final state x_{goal} find a time T and a set of control $u : [0, T] \rightarrow \mathcal{U}$ such that the motion

$$x(T) = x(0) + \int_0^T f(x(t), u(t))dt$$

satisfies

$$x(T) = x_{\text{goal}}$$

$$q(x(t)) \in \mathcal{C}_{\text{free}} \; for \; all \; t \in [0, T]$$

Types of Motion Planning Algorithms

- Path planning vs. motion planning
  - Path planning is a purely geometric problem of finding a collision-free path $q(s), s \in [0,1] \quad q(0) = q_{\text{start}} \quad q(1) = q_{\text{goal}}$
  - No concern about dynamics/control inputs
- Control inputs: $m = n$ **versus** $m < n$
  - When m < n, the robot cannot follow many paths
  - E.g., a car, n = 3, m = 2
- Online vs. Offline
  - Online is needed when the environment is dynamic

Types of Motion Planning Algorithms

- Optimal vs. satisficing
  - In addition to reaching the goal state, we might want the motion planner to minimize a cost

$$J = \int_0^T L(x(t), u(t))dt$$

Time-optimal L=1

Minimum-effort $L = u^{\mathrm{T}}(t)u(t)$

- Exact vs. approximate
  - Approximate $\|x(T) - x_{\text{goal}}\| < \epsilon$

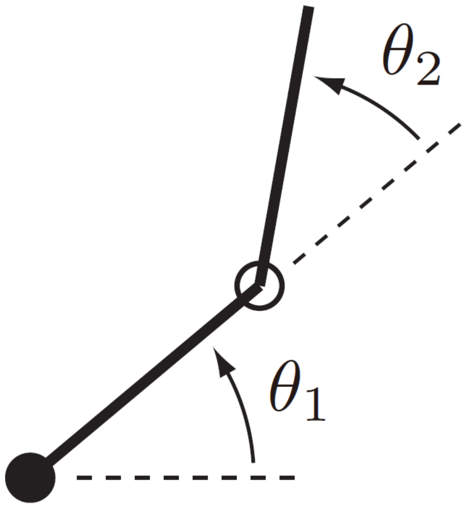
- With or without obstacles
  - Some motion planning problems are challenging even without obstacles
  - When m< n or optimality is desired

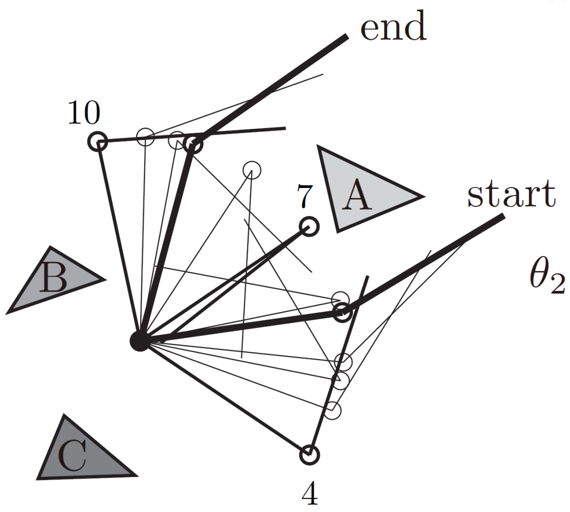
Properties of Motion Planners

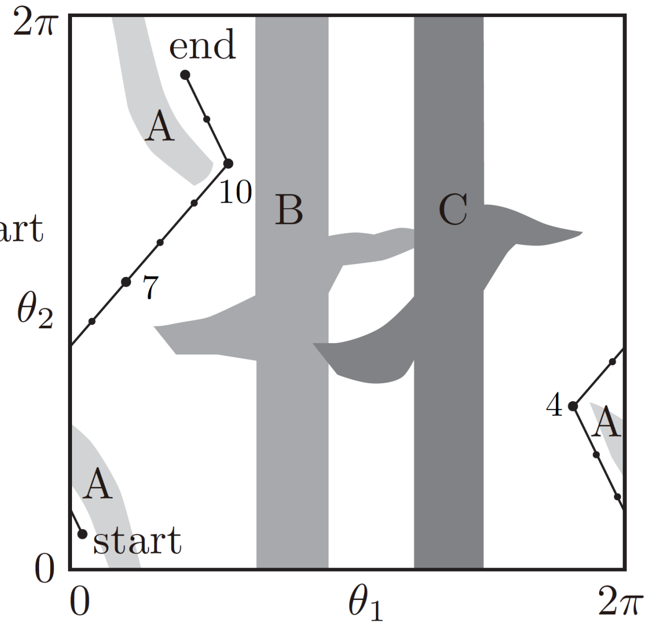
- Multiple-query vs. single-query planning
  - Multiple-query can build a data structure for $\mathcal{C}_{\text{free}}$
- "Anytime" planning
  - Continues to look for a better solution after a first solution is found
  - The planner can be stopped at anytime
- Completeness
  - A motion planner is said to be complete if it is guaranteed to find a solution in finite time if one exists, and to report failure if there is no feasible motion plan
- Computational complexity
  - The amount of time the planner takes to run or the amount of memory it requires

Configuration Space Obstacles

- Workspace obstacles partition the configuration space into two sets
  - Free space and obstacle space $\mathcal{C} = \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{obs}}$
  - Joint limits are treated as obstacle in the configuration space
- A 2R planar arm



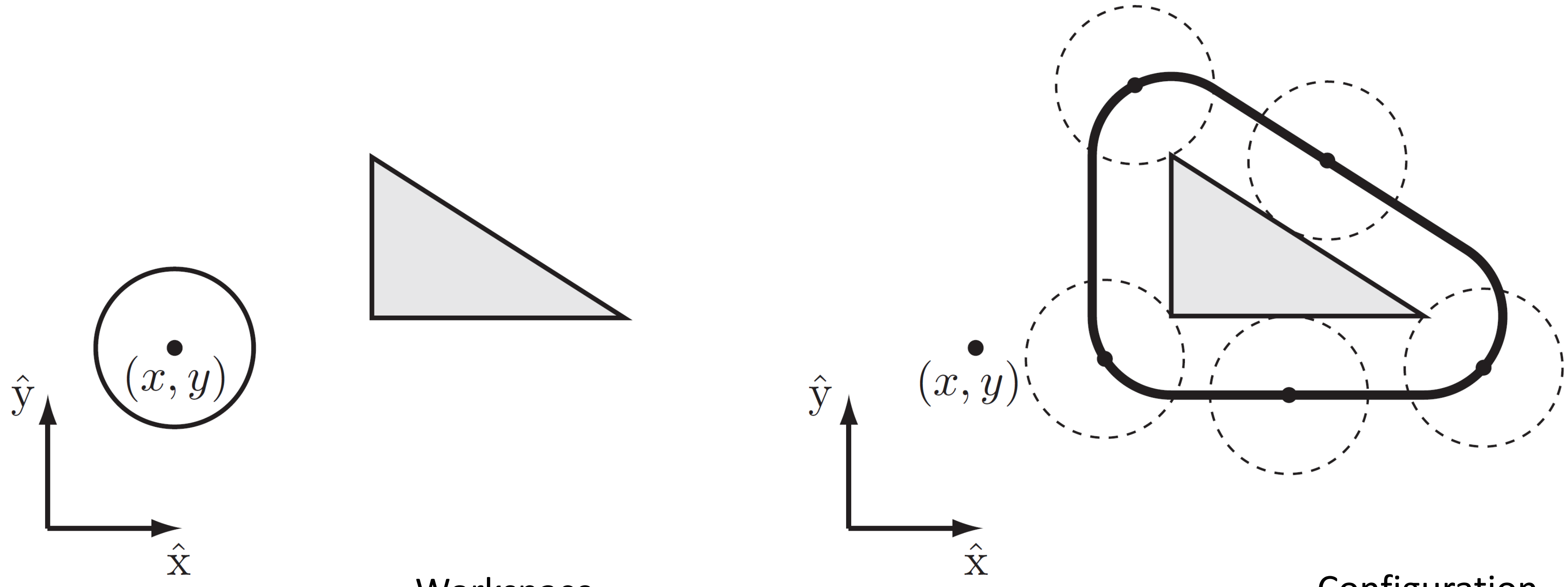




Configuration space

Configuration Space Obstacles

- A circular planar mobile robot

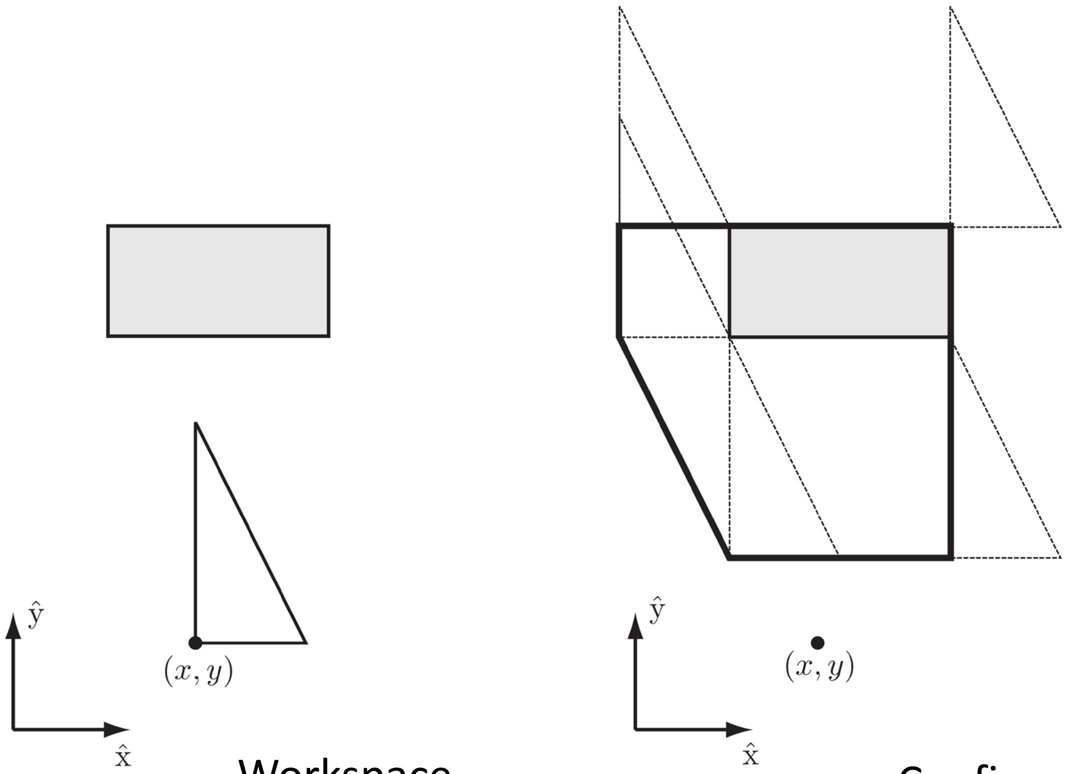


(a) Workspace space

(b) Configuration space

Configuration Space Obstacles

- A Polygonal Planar Mobile Robot That Translates

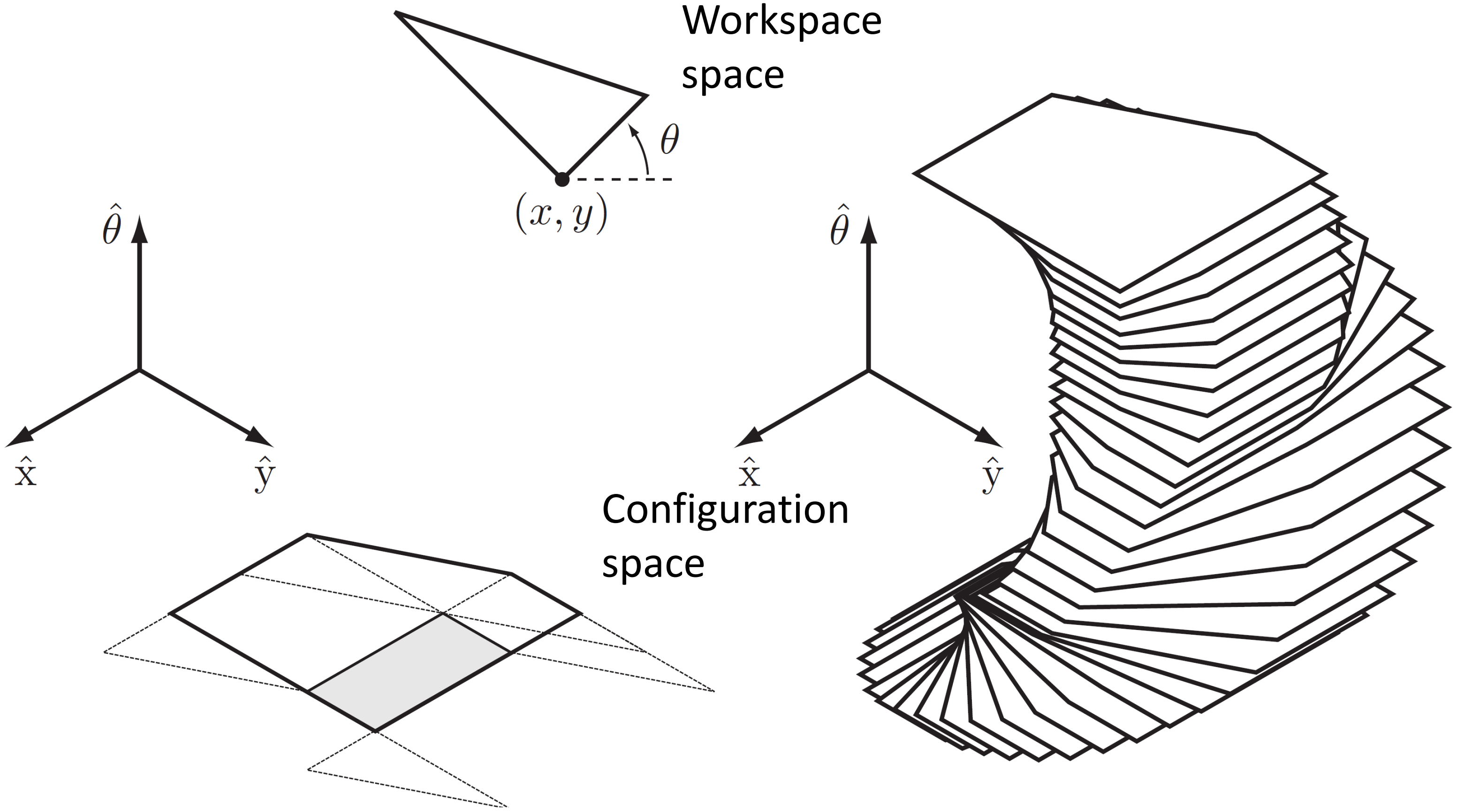


(a) Workspace space

(b) Configuration space

Configuration Space Obstacles

- A Polygonal Planar Mobile Robot That Translates and Rotates



Distance to Obstacles

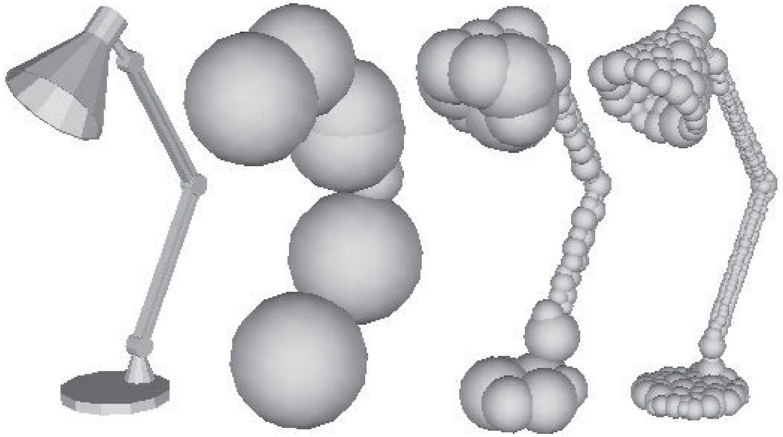
- Given a C-obstacle $\mathcal{B}$ and a configuration q, the distance between a robot and the obstacle

$$
\begin{aligned}
d(q, \mathcal{B}) &> 0 && \text{(no contact with the obstacle)}, \\
d(q, \mathcal{B}) &= 0 && \text{(contact)}, \\
d(q, \mathcal{B}) &< 0 && \text{(penetration)}.
\end{aligned}
$$

- A distance measurement algorithm determines $d(q, \mathcal{B})$
- A collision detection algorithm determines whether $d(q, \mathcal{B}_i) \leq 0$

Distance to Obstacles

- Approximation of 3D shapes using 3D spheres

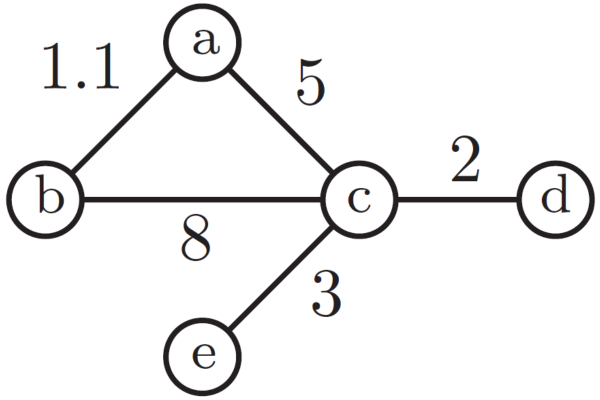


- Robot k spheres of radius R_i centered at $r_i(q)$
- Obstacle l spheres of radius B_j centered at b_j
- The distance between the robot and the obstacle

$$d(q, \mathcal{B}) = \min_{i,j} \|r_i(q) - b_j\| - R_i - B_j$$

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]`

    `past_cost[nbr] = tentative_past_cost`

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

    Compute estimated toal 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)

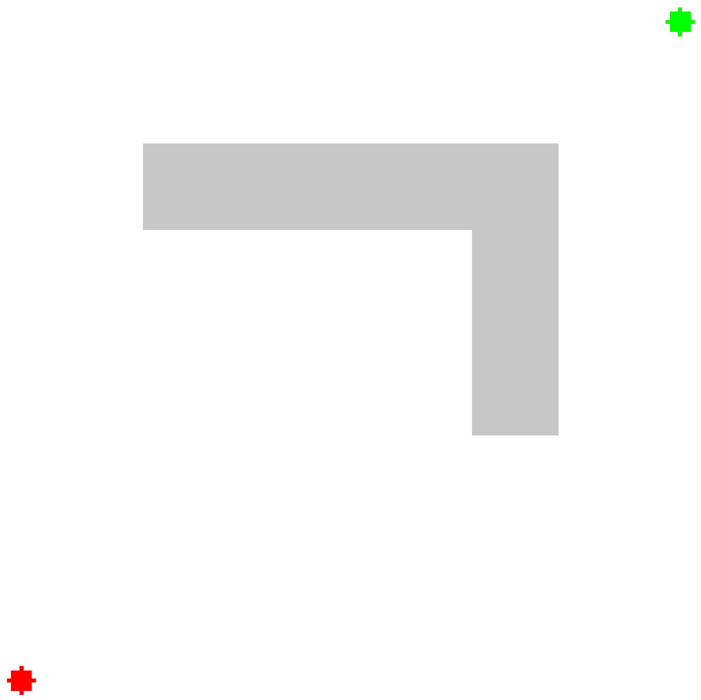
A* Search Algorithm

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



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

Summary

- Overview of motion planning
- Configuration space obstacle
- Distance to obstacles
- A* search algorithm

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.