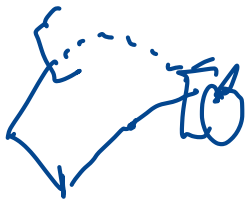
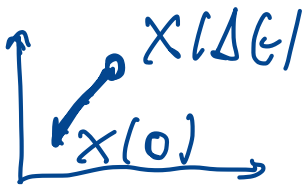
PRM RRT

Overview of Motion Planning

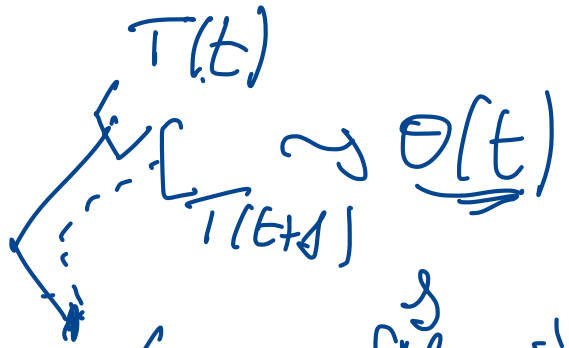
- **Motion planning** is the problem of finding a robot motion from start state to a goal state that avoids obstacles in the environment
- Recall the **configuration space or C-space:** every point in the C-space $\mathcal{C} \subset \mathbb{R}^n$ corresponds to a unique configuration q of the robot
  - E.g., configuration of a robot arm is $q = (\theta_1, \theta_2, \ldots, \theta_n)$
- The **free C-space** $\mathcal{C}_{\text{free}}$ consists of the configurations where the robot neither collides with obstacles nor violates constraints

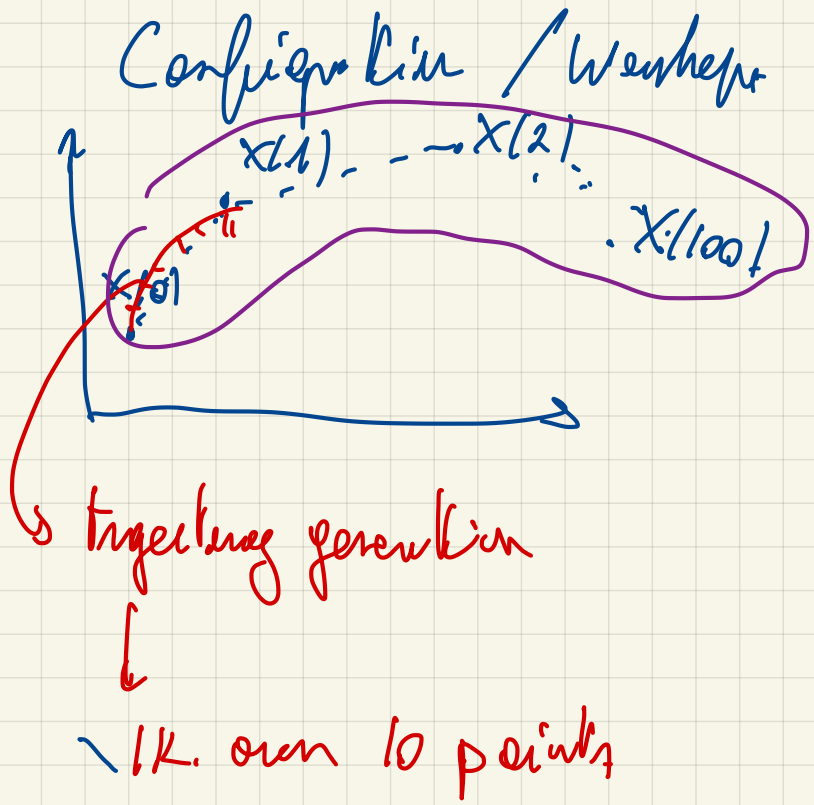


· Trajectory generation.
· IK.



$\rightsquigarrow X(1) = X_0 \exp\left[\log X_0^{-1} X_1\right]$





Motion planning
- obstacles
- geometry of robot
- (dynamics)

Trajectory generation
[
↳ IK. even 10 points

Motion Planning

not dealing with control here.

Given an initial state $x(0) = x_{start}$ and a desired final state x_{goal}, find a time T and a set of controls $u: [0, T] \rightarrow \mathcal{U}$ such that the motion satisfies $x(T) = x_{goal}$ and $q\big(x(t)\big) \in \mathcal{C}_{\text{free}}$ for all $t \in [0, T]$

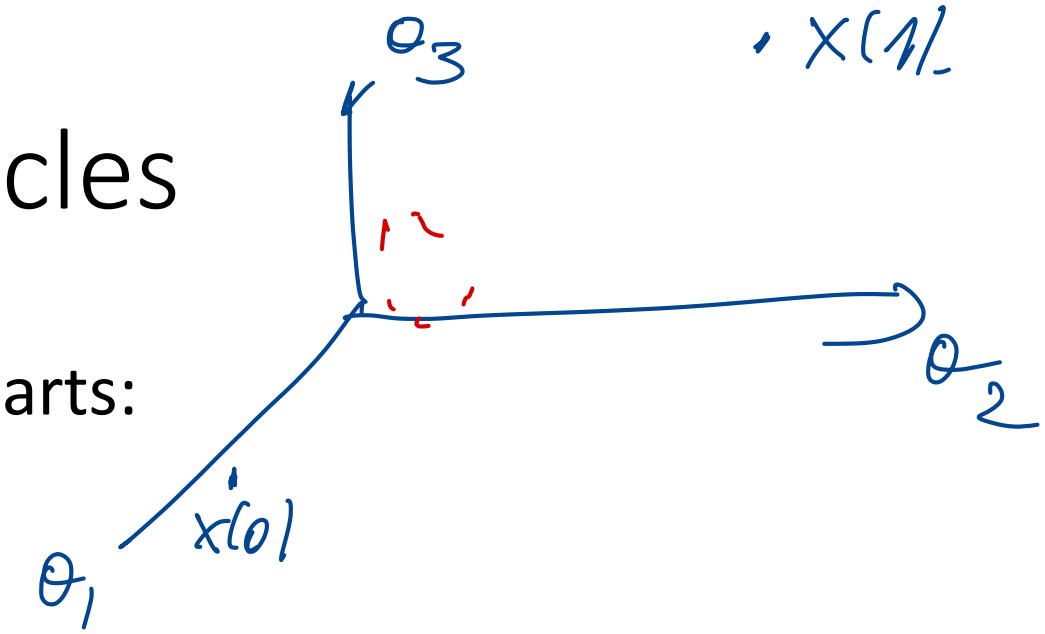
Assumptions:

1. A feedback controller can ensure that the planned motion is followed closely
2. An accurate model of the robot and environment will evaluate $\mathcal{C}_{\text{free}}$ during motion planning

Properties of Motion Planners

- **Multiple-query versus single-query planning**
- **"Anytime" planning**
  - Continues to look for better solutions after first solution is found
- **Computational complexity**
  - Characterization of the amount of time a planner takes to run or the amount of memory it requires

Configuration Space Obstacles



- We want to partition our C-space into parts:

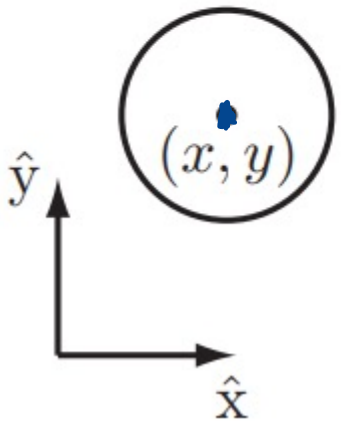
obstacles are in workspace

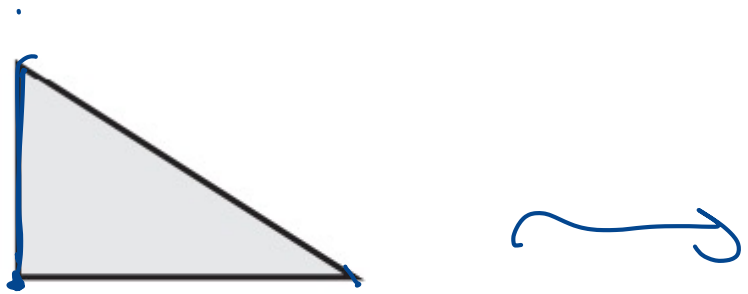
- If obstacles break C_{free} into separate components and q_0 and q_1 are not in the same connected components, then there is no collision free path

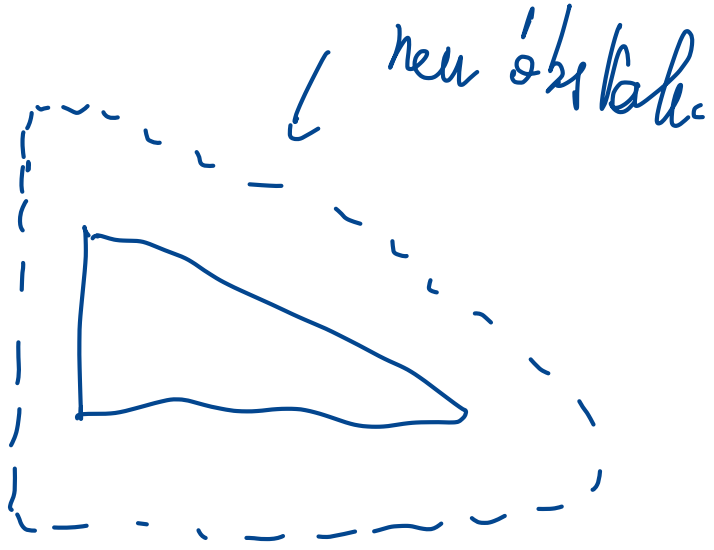
part I: Transfer 1. geometry of robot
2. obstacles in workspace
to configuration space.

Configuration Space: Circular Mobile Bot

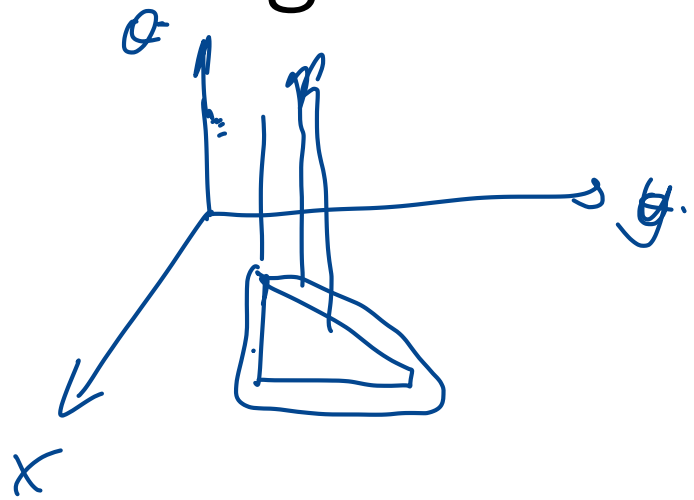
[geometry of robot ~ obstacles in workspace !

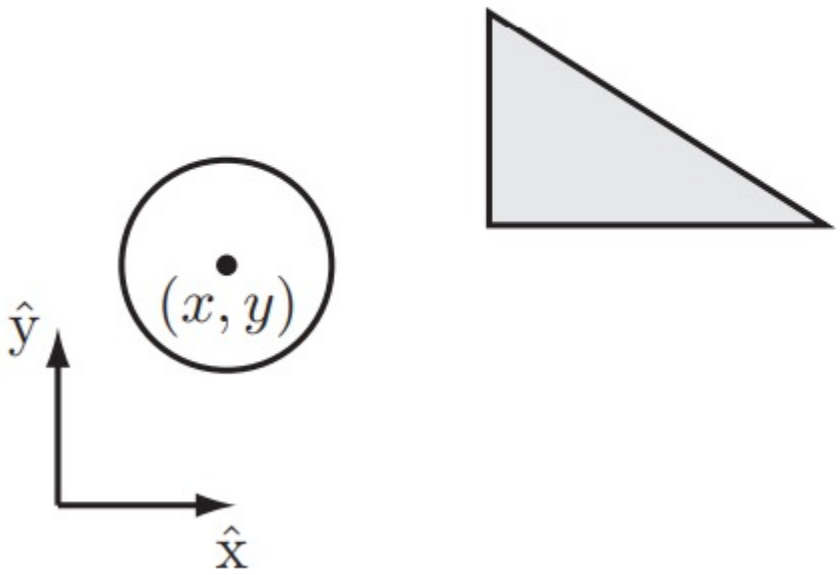


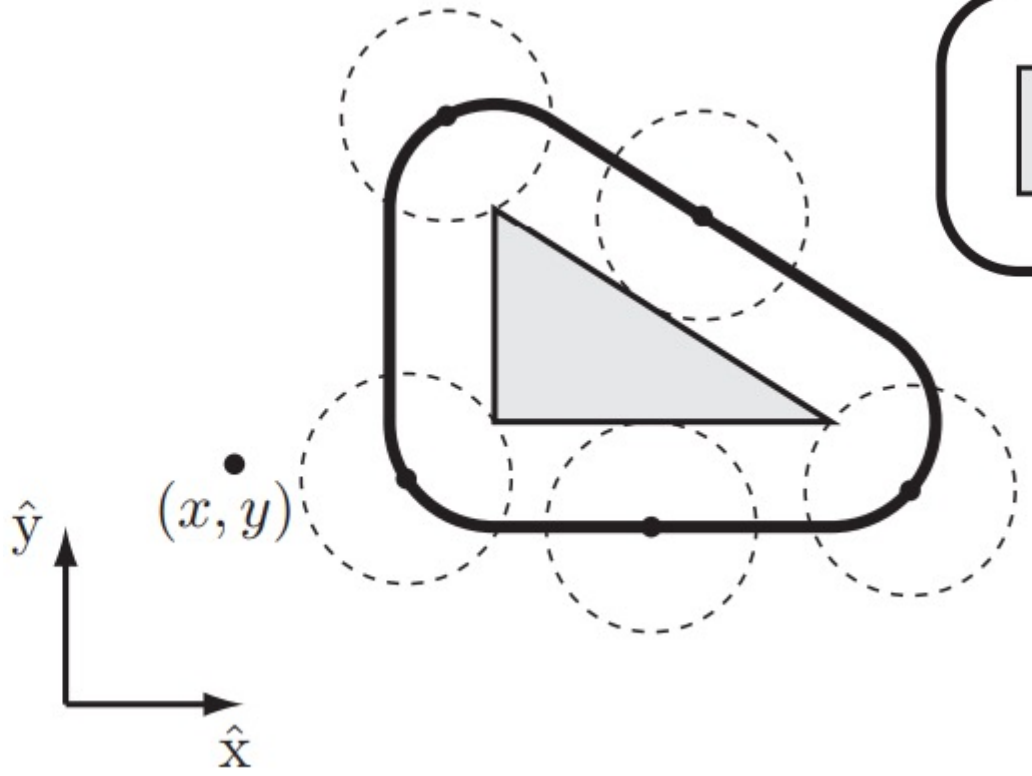


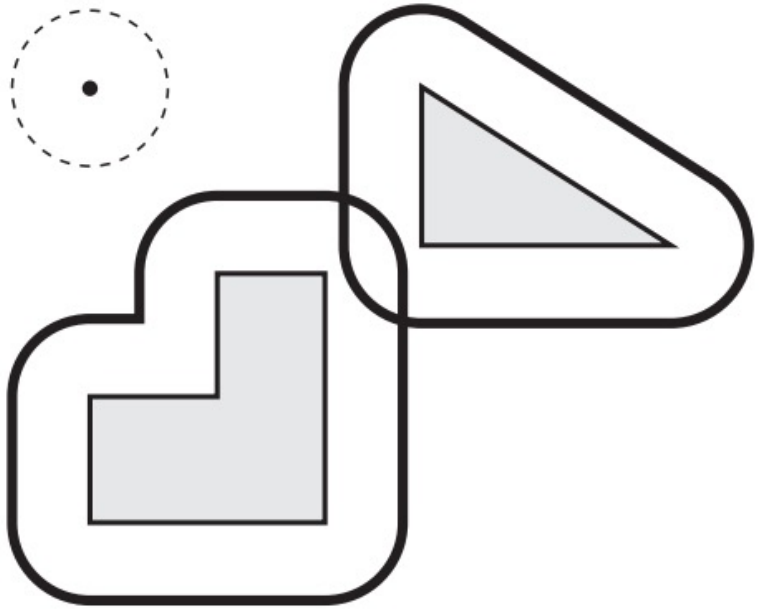


Configuration Space: Circular Mobile Bot

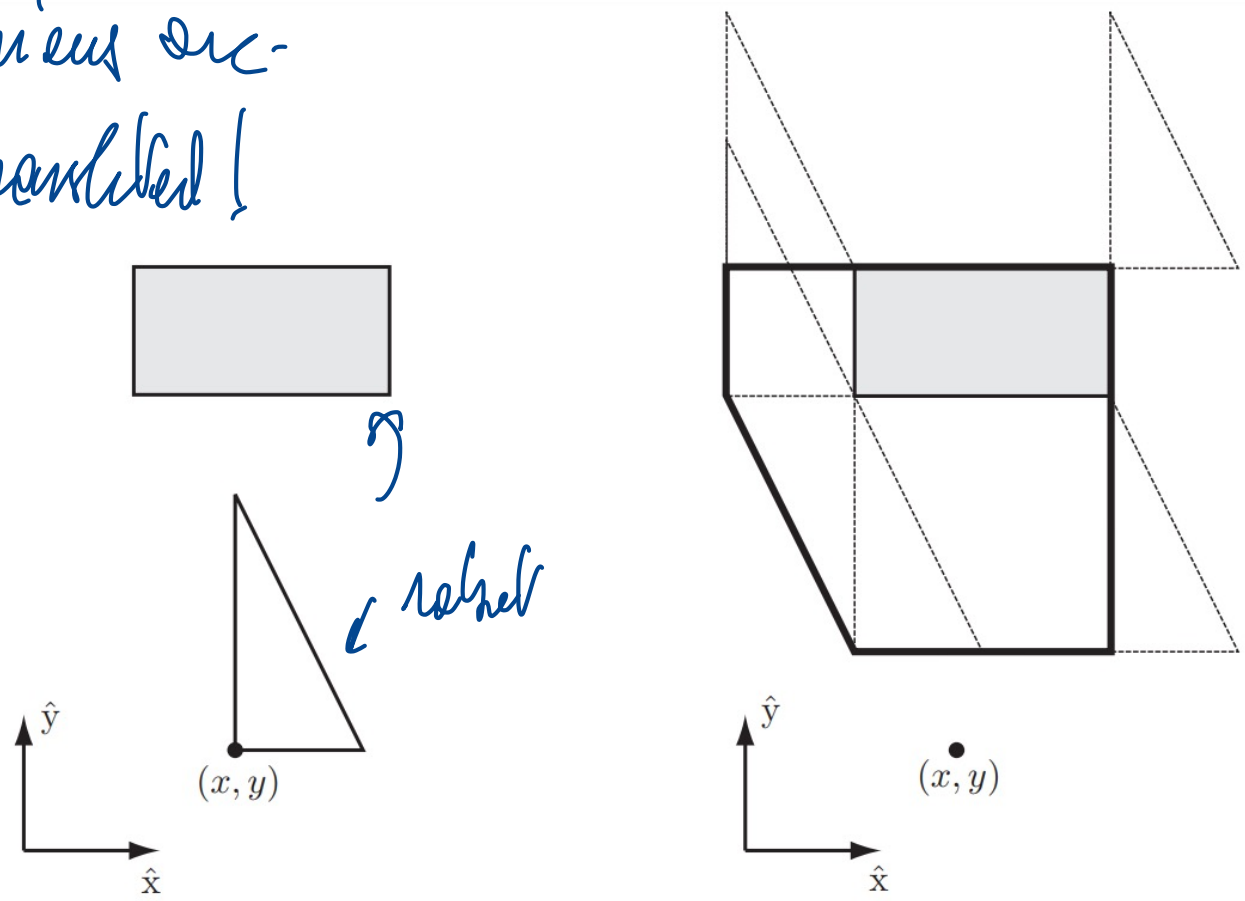








Configuration Space: bot that translates + rotates

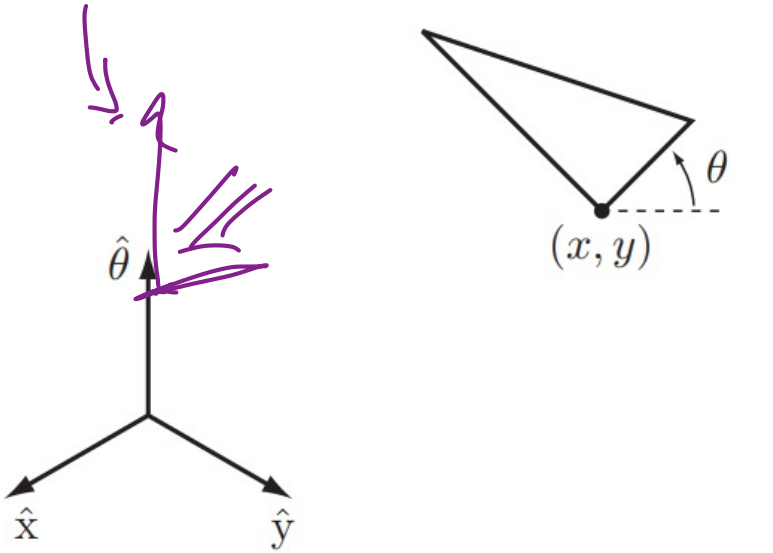


Configuration Space: bot that translates + rotates

(x, y, θ)

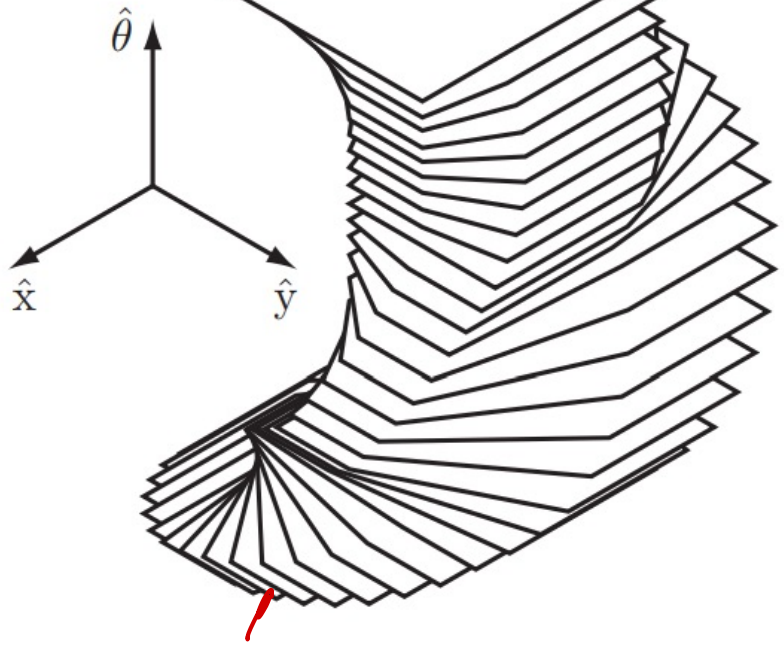
pos. orient.

$\begin{cases} x \in \mathbb{R}^3 \\ R \in SO(3) \end{cases}$

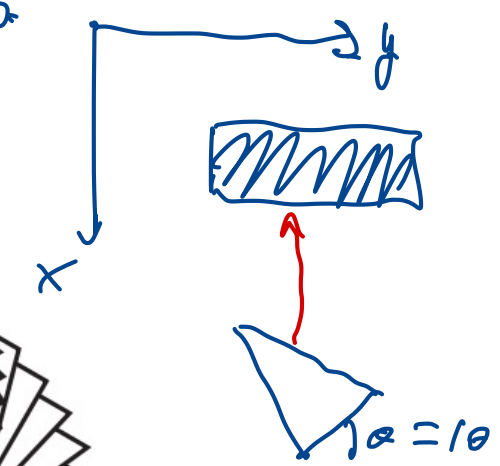


what if $|\theta| < \pi/4$?

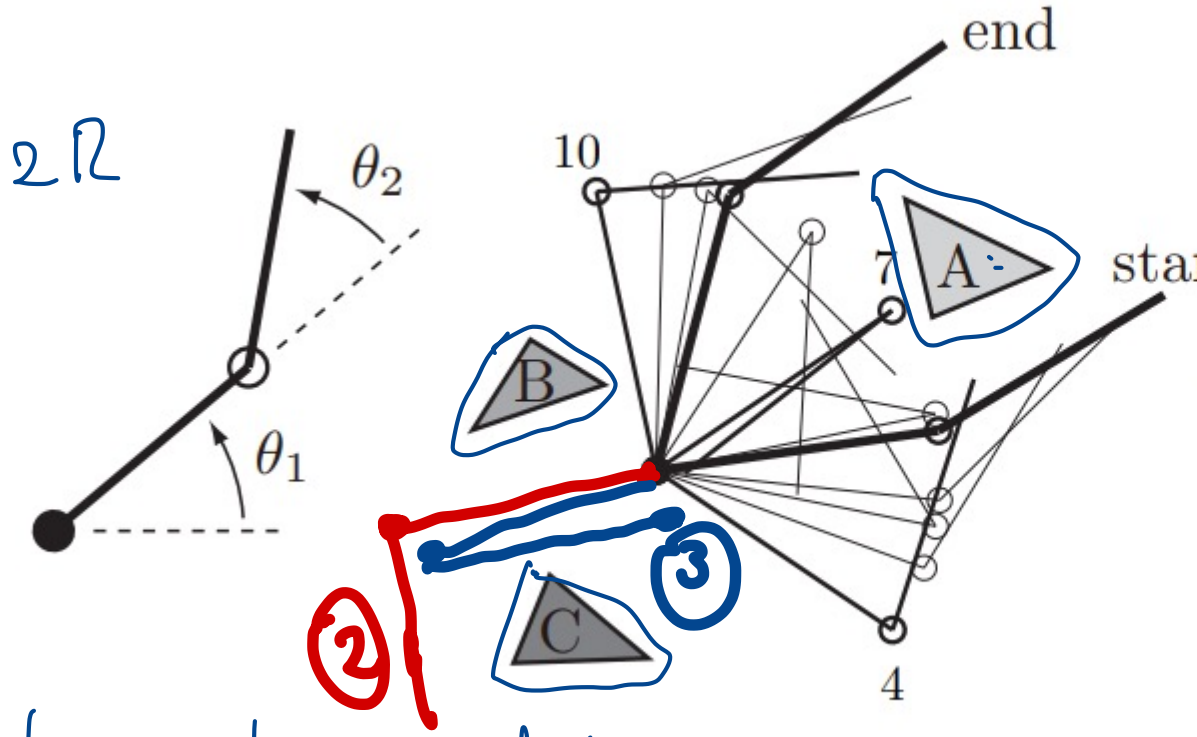
coordinate variable

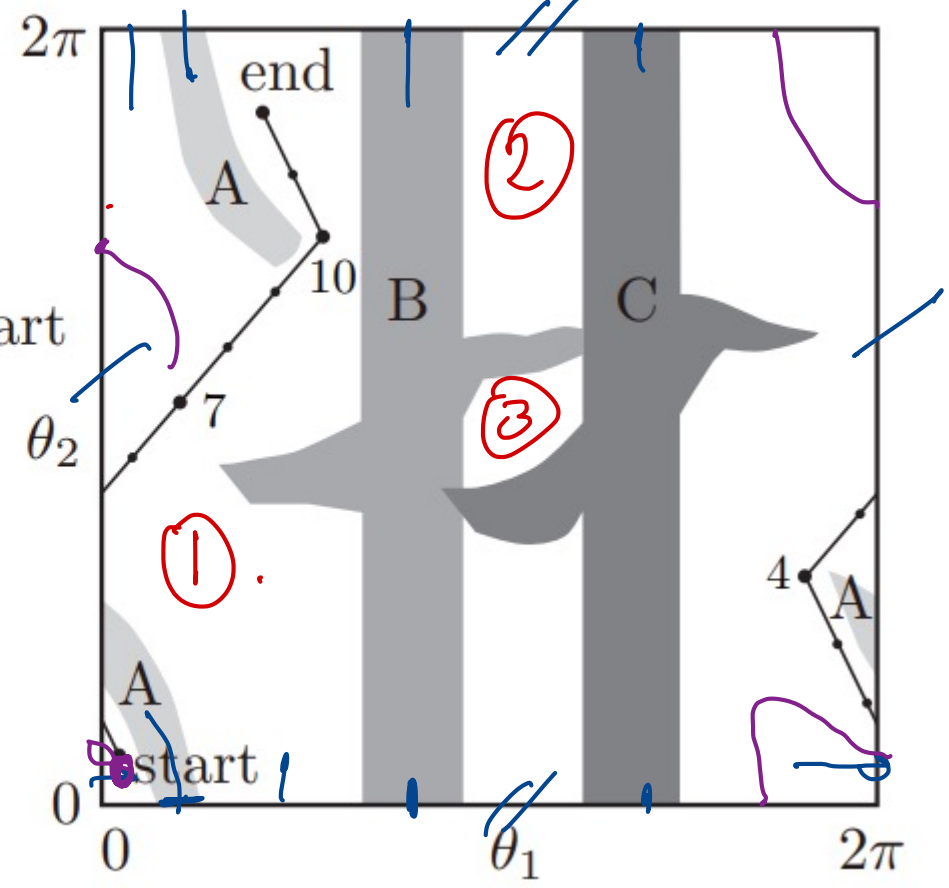


$\theta = 0$

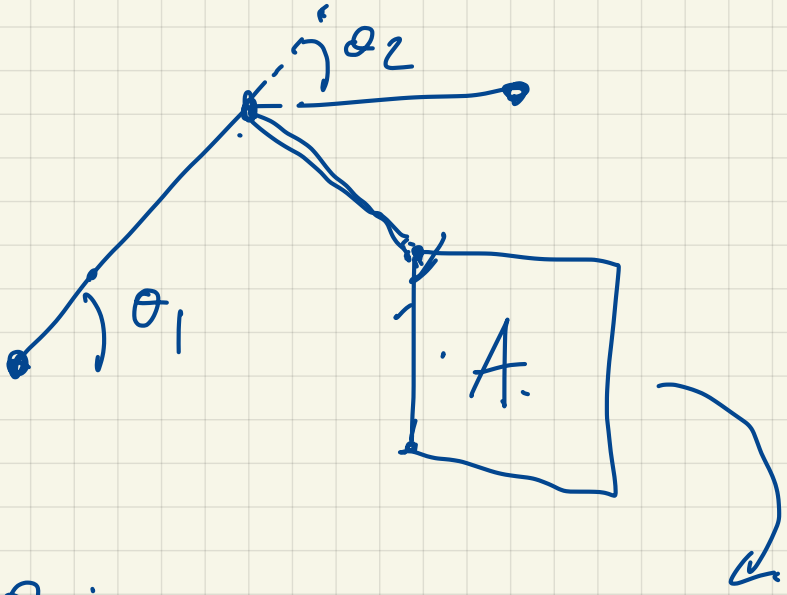


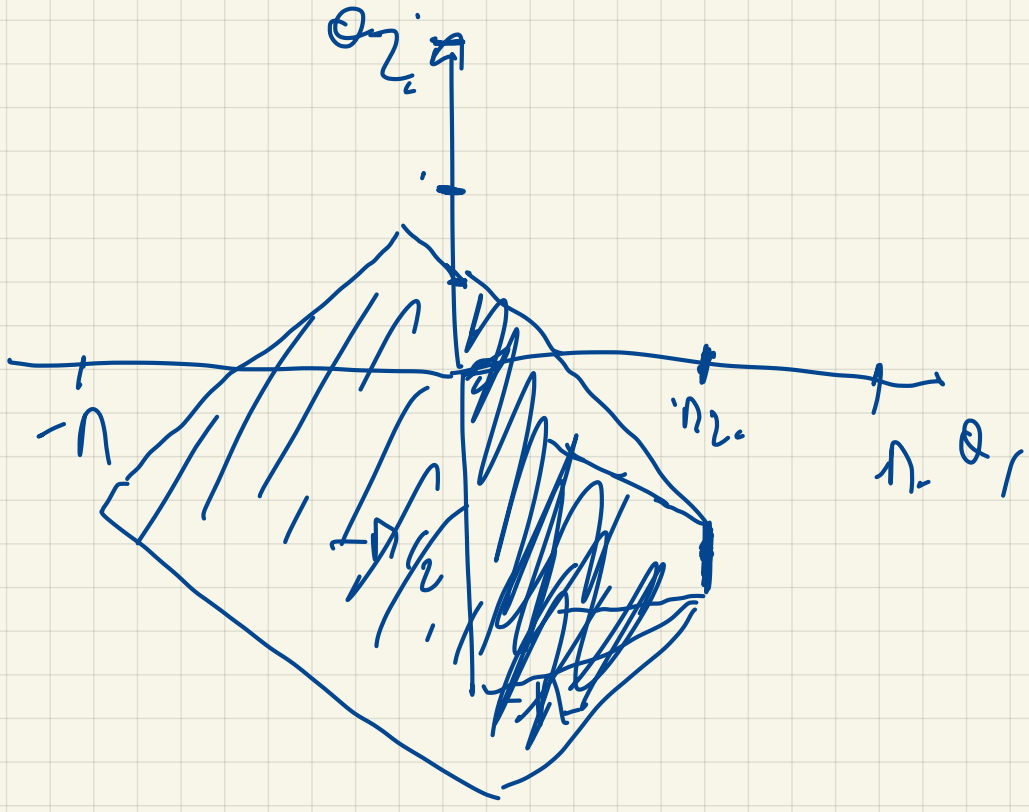
Configuration Space: 2R Planar Arm





this not connected


θ2
θ1
A.

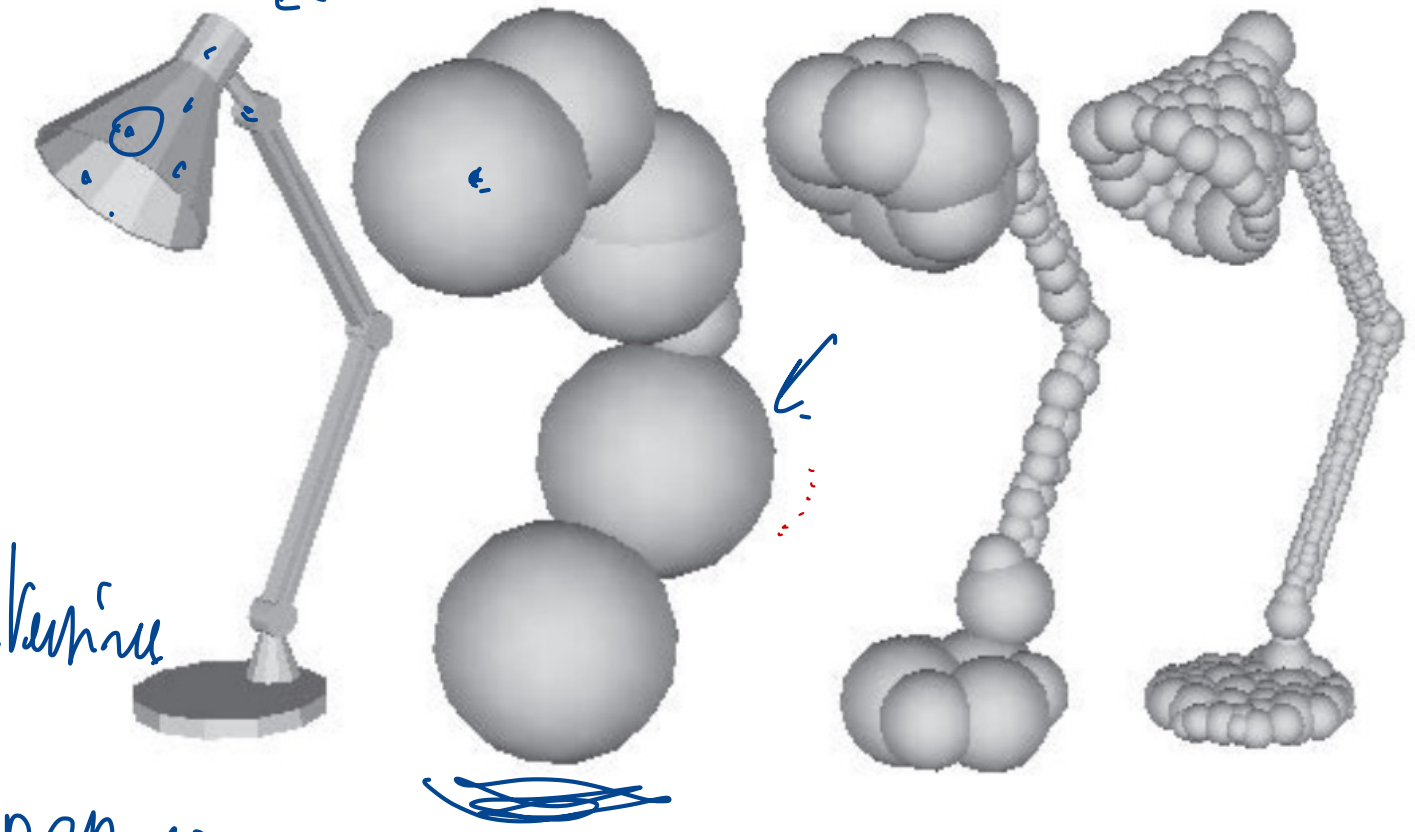

θ2
n2
n1 θ1

Spherical Approximations

Approximate geometry of obstacle.

point inters → modify

One simple method is to approximate the robot and obstacles as unions of overlapping spheres

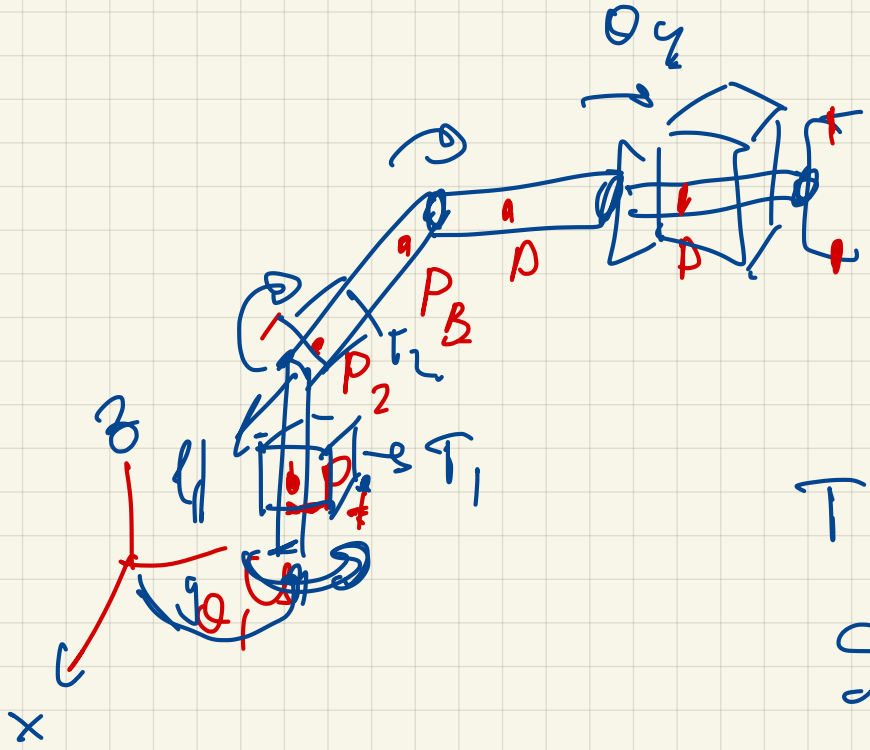


why: high def models → computationally intensive

but, better if narrow passages

spheres allow for fast intersection computations

coarse.



$$T_1 = ?$$

$$S_1 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$e^{[S_1]\theta_1} = \text{pen}$$

$$S_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -l_1 \\ 0 \end{bmatrix}$$

Distance Measures

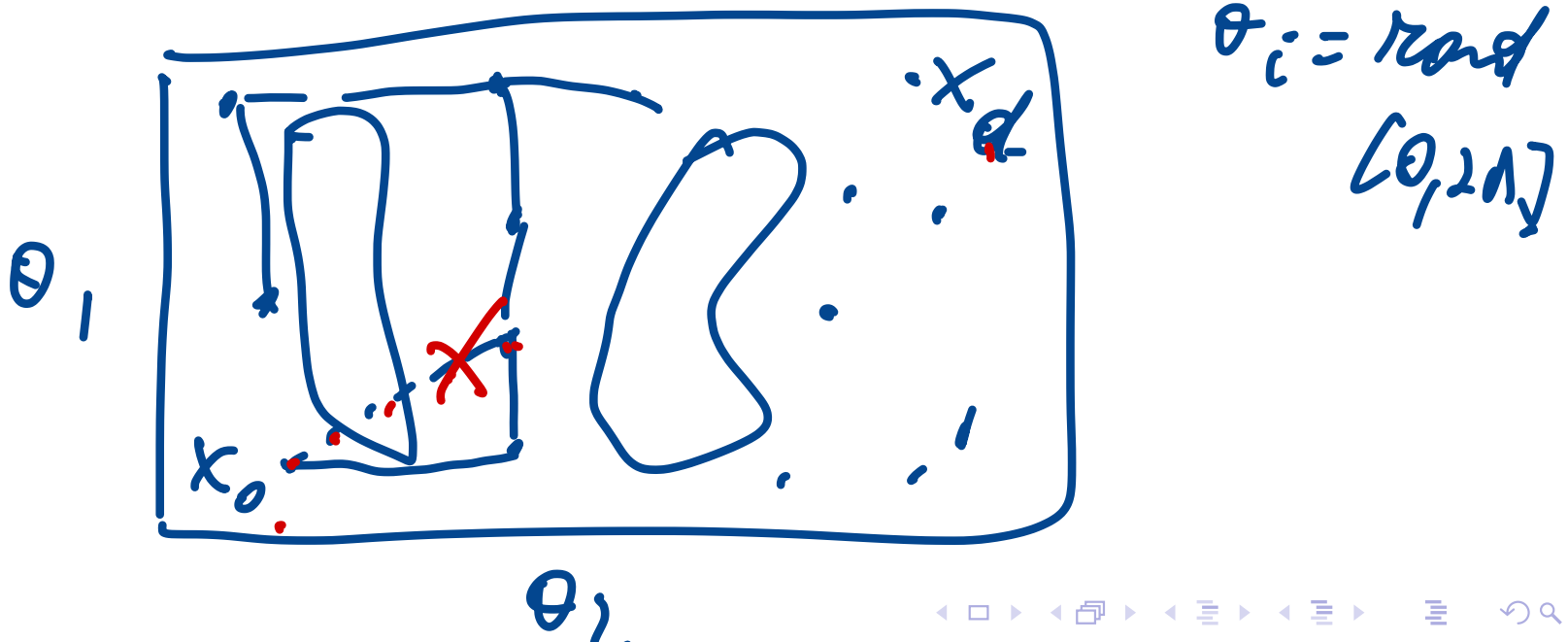
Given a robot at q represented by k spheres of radius R_i centered at $r_i(q)$, and an obstacle $\mathcal{B}$ represented by l spheres of radius B_j centered at b_j, the distance can be calculated as:

Summary

- Defined and discussed concepts relating to **motion planning**
- Discussed configuration space components $\boldsymbol{\mathcal{C}_{\mathbf{free}}}$ and $\boldsymbol{\mathcal{C}_{obs}}$
- Gave example distance measurement approaches to check **collision detection**
- **Next time:** Learn about graphs, searching, and RRT

Motion Planning II: Probabilistic Road Maps and A* search

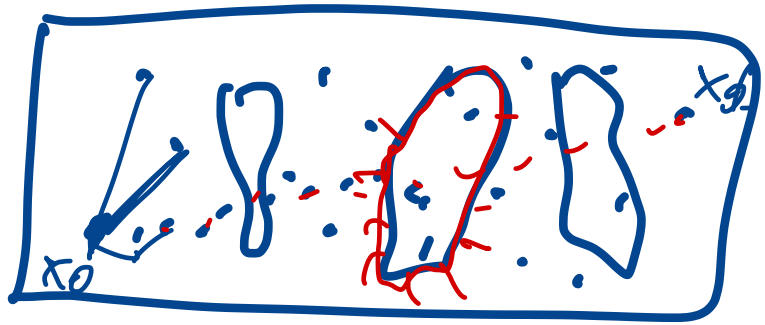
Introduction to Robotics, SP25



Introduction

- Motion planning involves finding a path from a start to a goal configuration.
- Challenges include high-dimensional spaces and obstacles.
- Probabilistic Roadmaps (PRM) are a sampling-based method to solve motion planning problems.

Probabilistic Roadmaps (PRM)



- PRM consists of two phases:
  1. **Construction Phase**: Build a roadmap by randomly sampling the configuration space and connecting nearby nodes.
  2. **Query Phase**: Find a path from start to goal using the roadmap.

Construction Phase: Milestones and Graph Construction

- **Milestones**: The randomly sampled points in the configuration space are called milestones.
- **Graph Construction**:
  - Each milestone is a node in the graph.
  - Edges are added between milestones if:
    - They are close to each other (e.g., within a certain distance).
    - The straight-line path between them is collision-free.
- The result is a graph (roadmap) that captures the connectivity of the free space.

Construction Phase: Example

nn : nbr in roadmap.

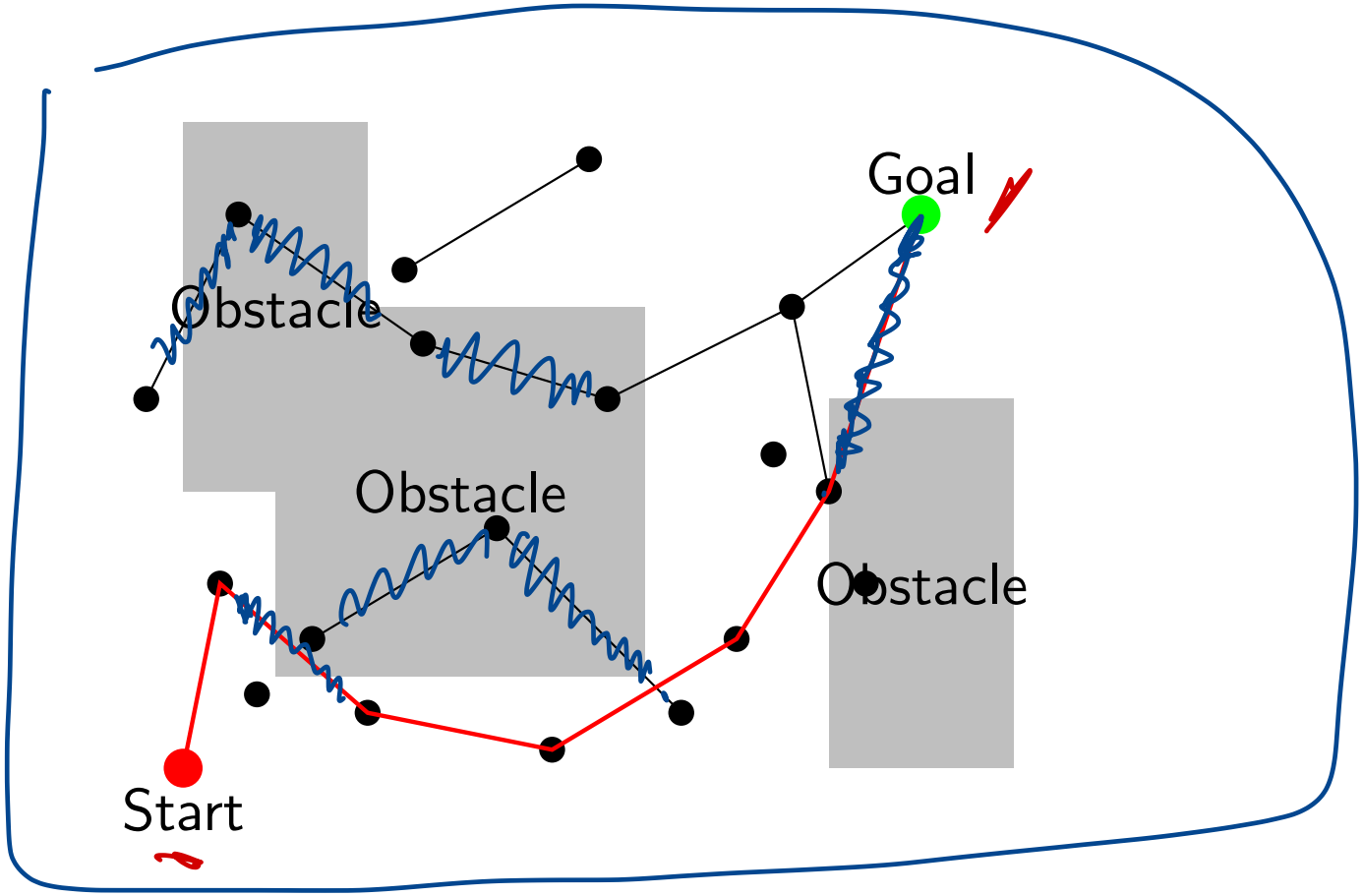
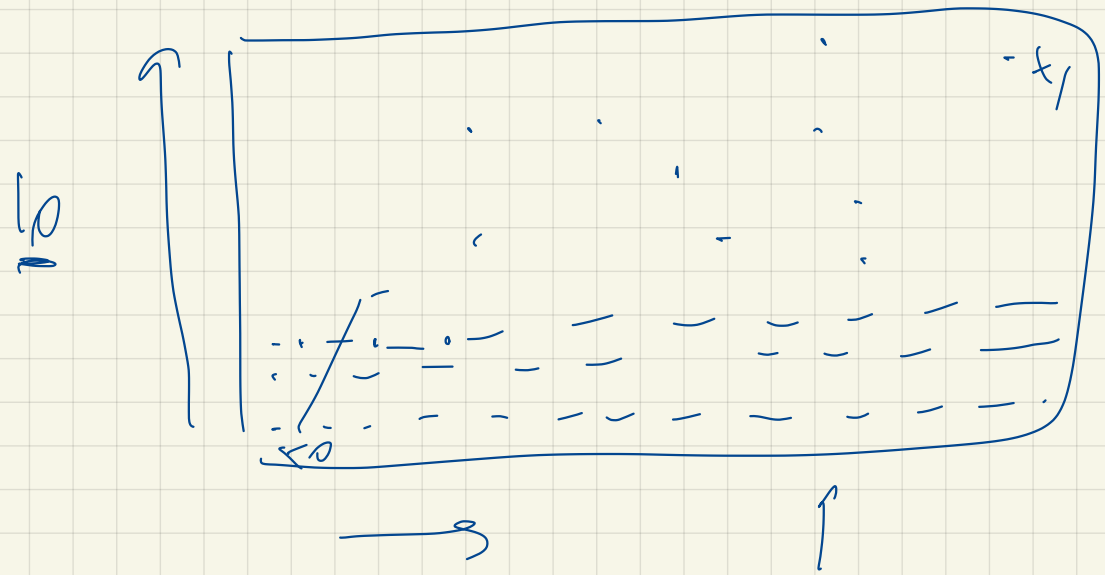


Figure: Example of a roadmap with milestones, obstacles, and a single path (in red) from start to goal.



multiresolution anc.

Curse of dimensionality :

points scale as b^n ; n is # dofs

$n = 32$ dofs.

5^{32}

$\gg 2^{32}$

$5 \to 10.$

Query Phase

- Add the start and goal configurations to the roadmap.
- Find a path from start to goal using graph search algorithms (e.g., Dijkstra's or A*).

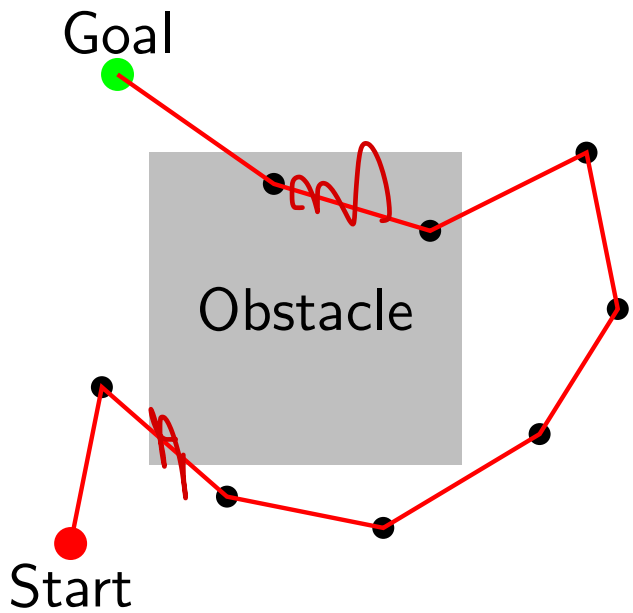
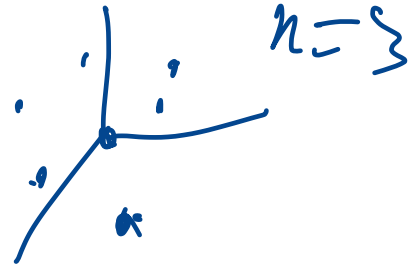


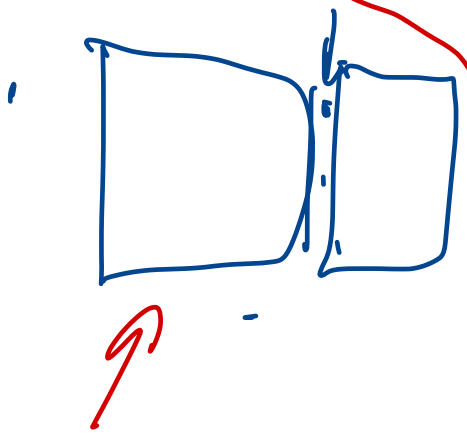
Figure: Path found from start to goal in the query phase.

Advantages of PRM

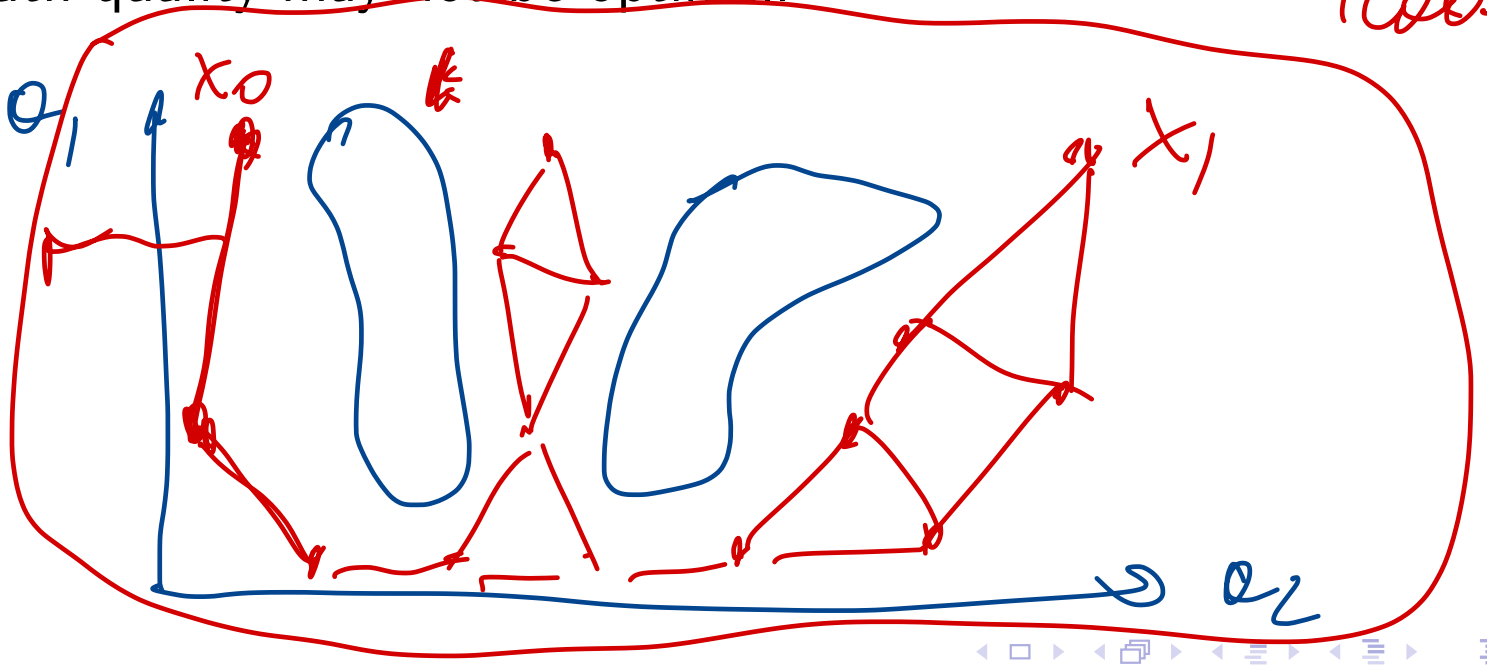


- Efficient in high-dimensional spaces.
- Can handle complex obstacle geometries.
- Probabilistically complete: probability of finding a path approaches 1 as the number of samples increases.

Limitations of PRM



- May not perform well in narrow passages.
- Requires a good sampling strategy to ensure coverage.
- Path quality may not be optimal.



Introduction to A* Search

- A* is a widely used algorithm for finding the shortest path between two nodes in a graph.
- It combines the advantages of Dijkstra's algorithm (guaranteed shortest path) and Greedy Best-First Search (efficiency).
- A* uses a heuristic function to estimate the cost from the current node to the goal, guiding the search.

A* Algorithm: Key Concepts

- **Heuristic Function (h(n))**: Estimates the cost from node n to the goal.
- **Path Cost (g(n))**: The cost from the start node to node n.
- **Total Cost (f(n))**: $f(n) = g(n) + h(n)$.
- A* prioritizes nodes with the lowest $f(n)$ value.

A* Algorithm: Steps

1. Initialize:
   - Open set: Contains nodes to be evaluated (start with the start node).
   - Closed set: Contains nodes already evaluated.
   - $g(n)$ and $f(n)$ values for each node.
2. While the open set is not empty:
   - Select the node with the lowest $f(n)$ from the open set.
   - If this node is the goal, reconstruct the path and return it.
   - Otherwise, move the node to the closed set and expand its neighbors.
   - For each neighbor, update $g(n)$ and $f(n)$ if a better path is found.

A* Example: Graph Setup

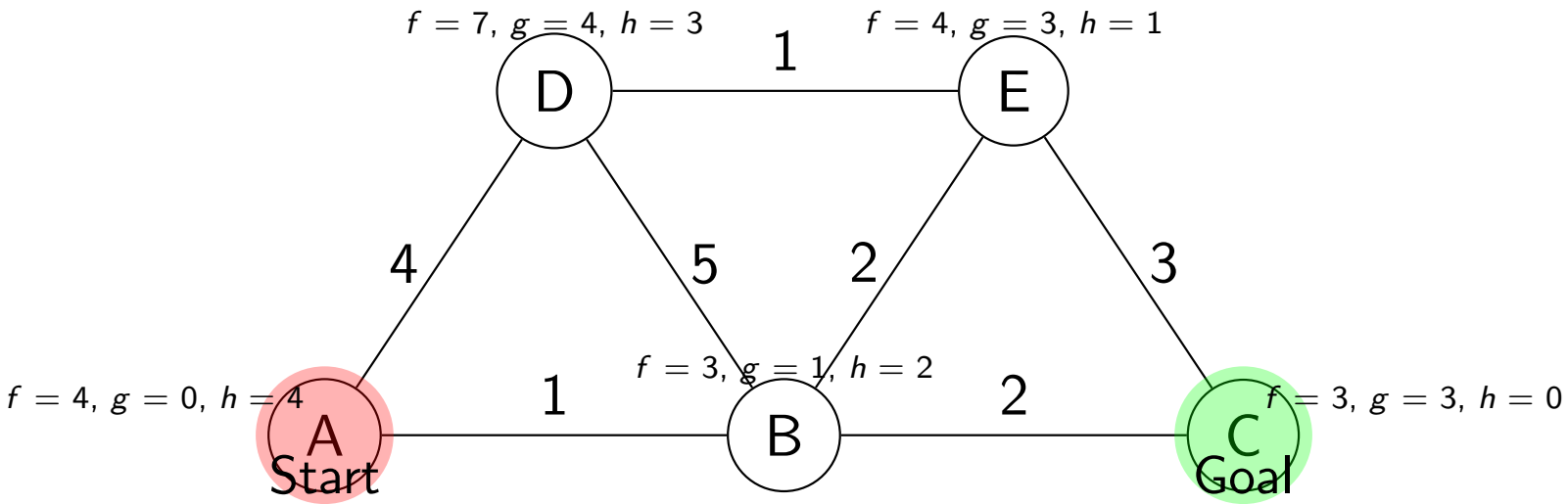


Figure: Example graph with edge weights. Start at A, goal at C.

A* Example: Step-by-Step

- **Heuristic Values**: Assume $h(n)$ is the straight-line distance to C (for simplicity).
- $h(A) = 4$, $h(B) = 2$, $h(C) = 0$, $h(D) = 3$, $h(E) = 1$.

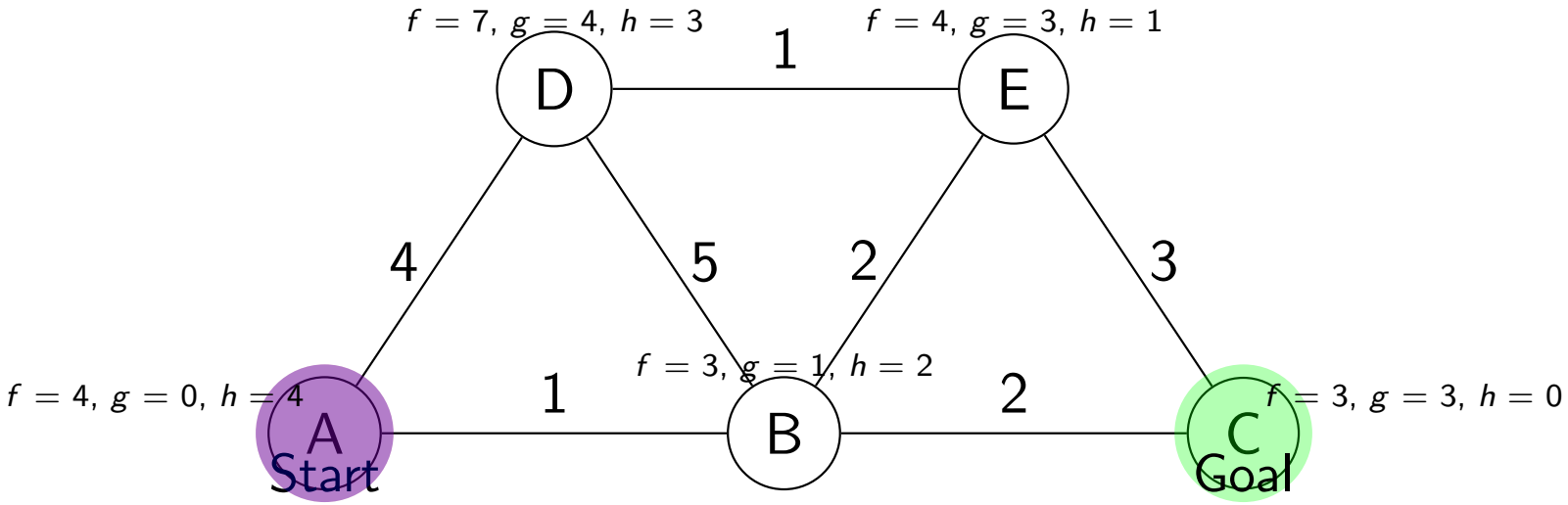


Figure: Step 1: Start at A. $f(A) = g(A) + h(A) = 0 + 4 = 4$.

A* Example: Step-by-Step (Continued)

- Expand A: Neighbors are B ($g(B) = 1$, $f(B) = 1 + 2 = 3$) and D ($g(D) = 4$, $f(D) = 4 + 3 = 7$).
- Select B (lowest $f(n)$).

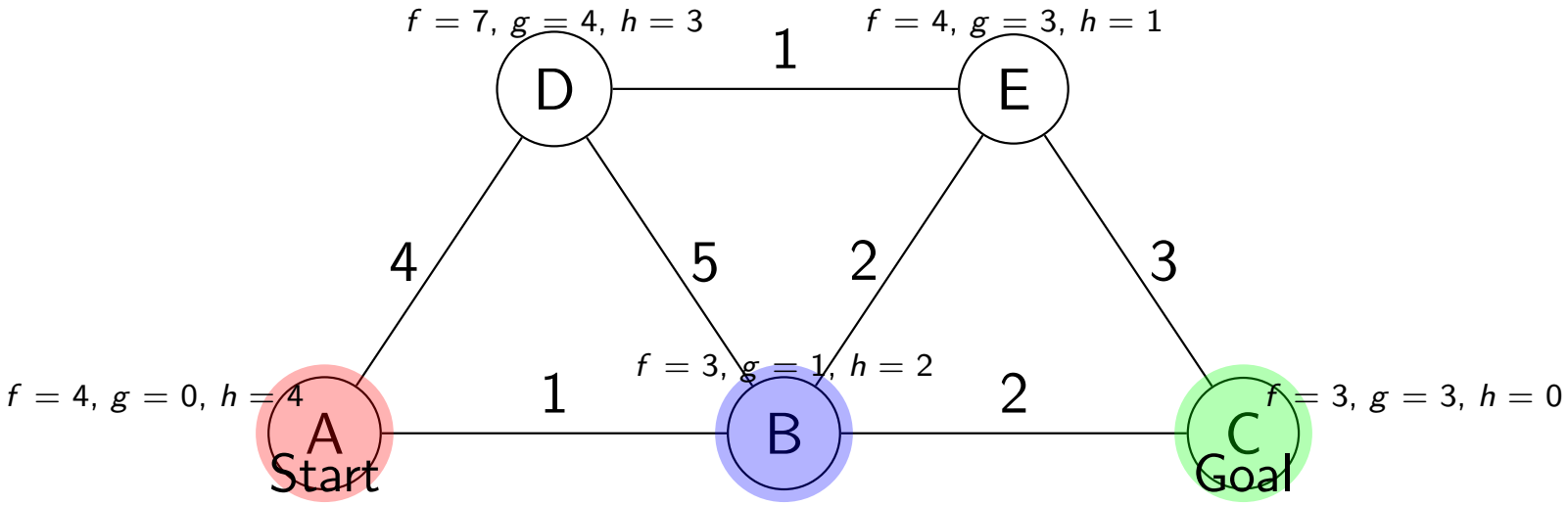


Figure: Step 2: Expand B. $f(B) = 3$.

A* Example: Step-by-Step (Continued)

- Expand B: Neighbors are C ($g(C) = 1 + 2 = 3$, $f(C) = 3 + 0 = 3$), D ($g(D) = 1 + 5 = 6$, $f(D) = 6 + 3 = 9$), and E ($g(E) = 1 + 2 = 3$, $f(E) = 3 + 1 = 4$).
- Select C (lowest $f(n)$).
- C is the goal. Path found: A $\rightarrow$ B $\rightarrow$ C.

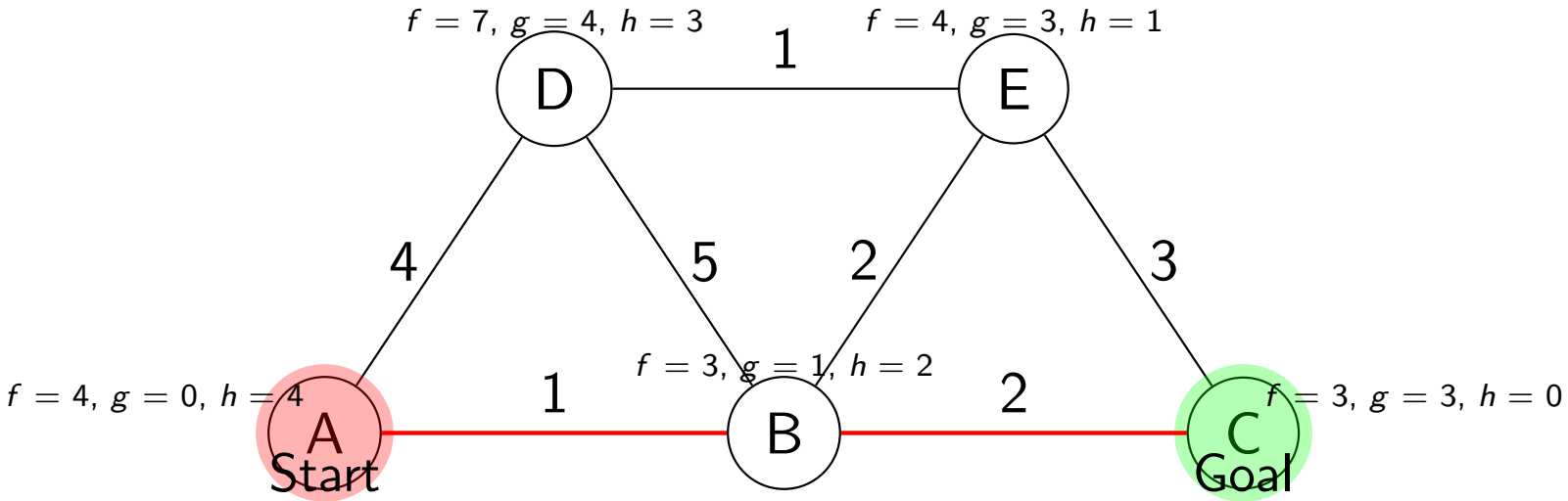


Figure: Step 3: Path found: A $\rightarrow$ B $\rightarrow$ C.

Advantages of A*

- Guaranteed to find the shortest path if the heuristic is admissible (never overestimates the cost).
- More efficient than Dijkstra's algorithm due to the heuristic.
- Widely used in pathfinding applications (e.g., robotics, games).

Limitations of A*

- Requires a good heuristic function for optimal performance.
- Memory usage can be high for large graphs.
- Not suitable for dynamic environments where the graph changes frequently.

Conclusion

- A* is a powerful and efficient algorithm for finding shortest paths in graphs.
- It balances optimality and efficiency using a heuristic function.
- Future work includes improving heuristics and adapting A* for dynamic environments.