EE565:Mobile Robotics
Lecture 12

Welcome

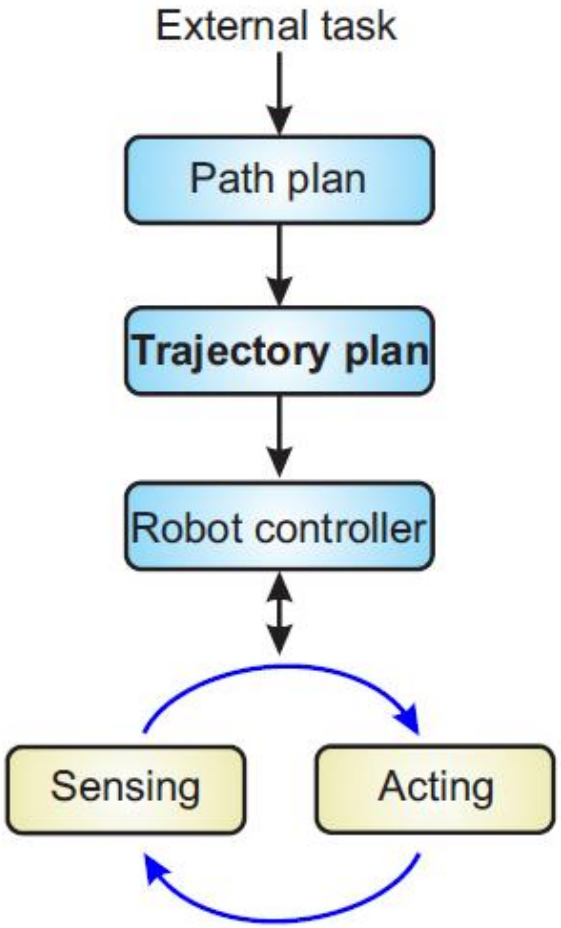
Dr. –Ing. Ahmad Kamal Nasir

Today's Objectives

Where am I going? How do I get there?

- Configuration/work spaces,
- Path Planning algorithms:
  - Dijkstra,
  - Greedy First,
  - A*
- Obstacle avoidance: Bug Algorithms

Trajectory Vs. Path



Challenges

- Calculate the optimal path taking potential uncertainties in the actions into account
- Quickly generate actions in the case of unforeseen objects

Classic Two-layered Architecture

map

sensor data

Planning

low frequency

sub-goal

Collision Avoidance

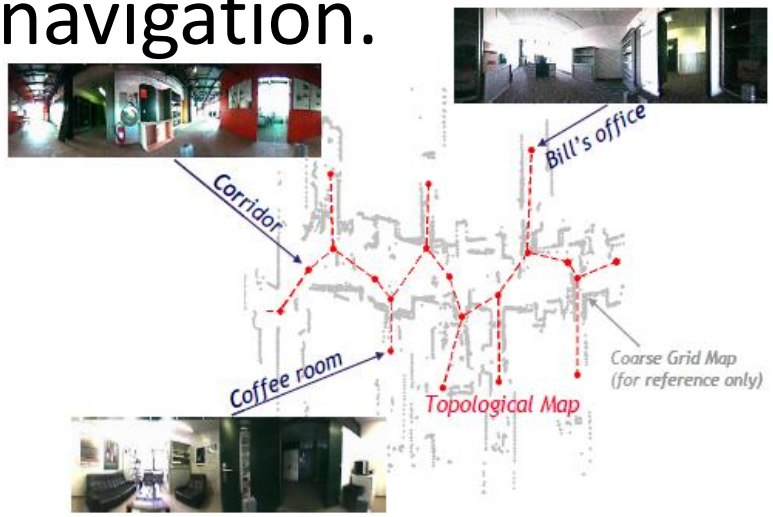
high frequency

motion command

robot

Planning Problem

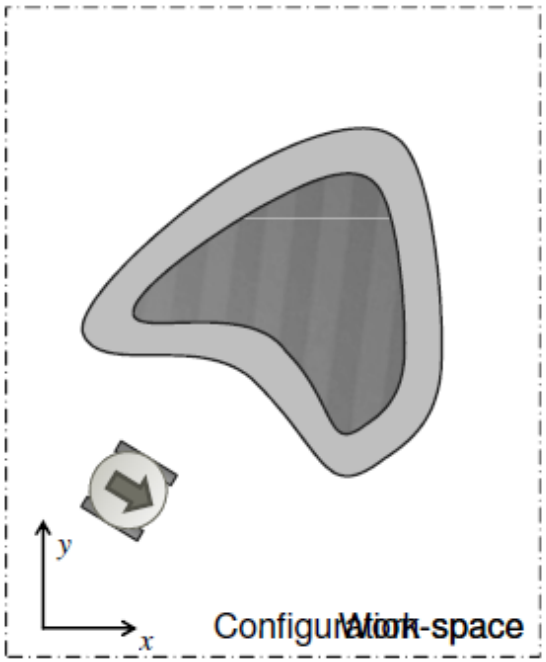
- Find a path in the physical space from an initial position to a goal position avoiding all collisions with obstacles
- Assumption: there exists a good enough map of the environment for navigation.
  - Topological
  - Metric
  - Hybrid methods

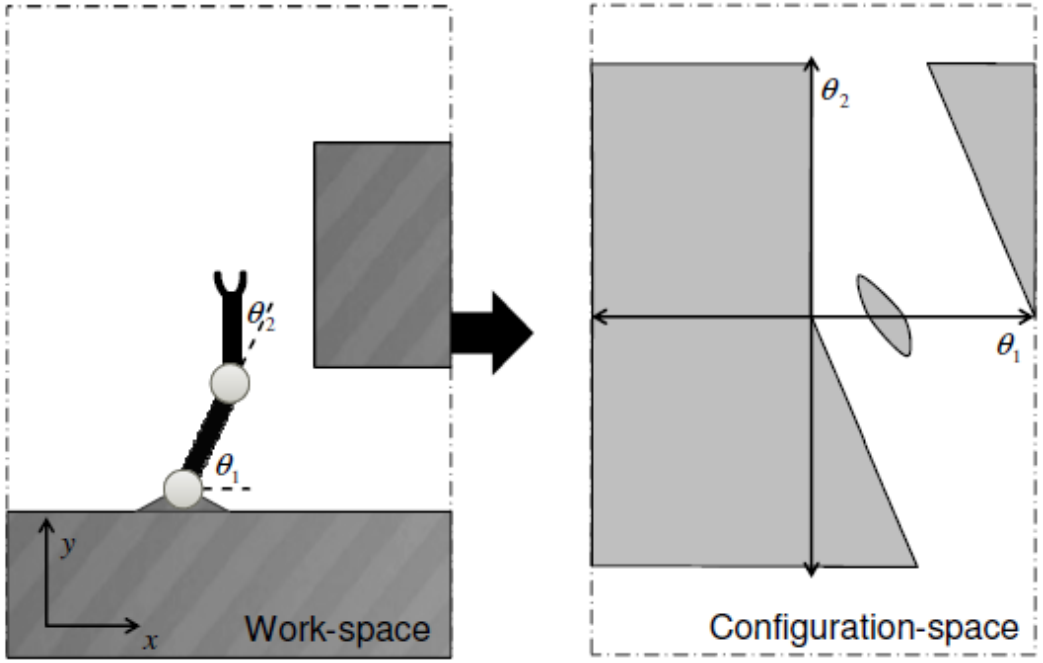


Planning Problem (Cont.)

- We can generally distinguish between
  - (global) path planning and
  - (local) obstacle avoidance.
- First step:
  - Transformation of the map into a representation useful for planning
  - This step is planner-dependent
- Second step:
  - Plan a path on the transformed map
- Third step:
  - Send motion commands to controller
  - This step is planner-dependent (e.g. Model based feed forward, path following)

Work Space Vs Configuration Space



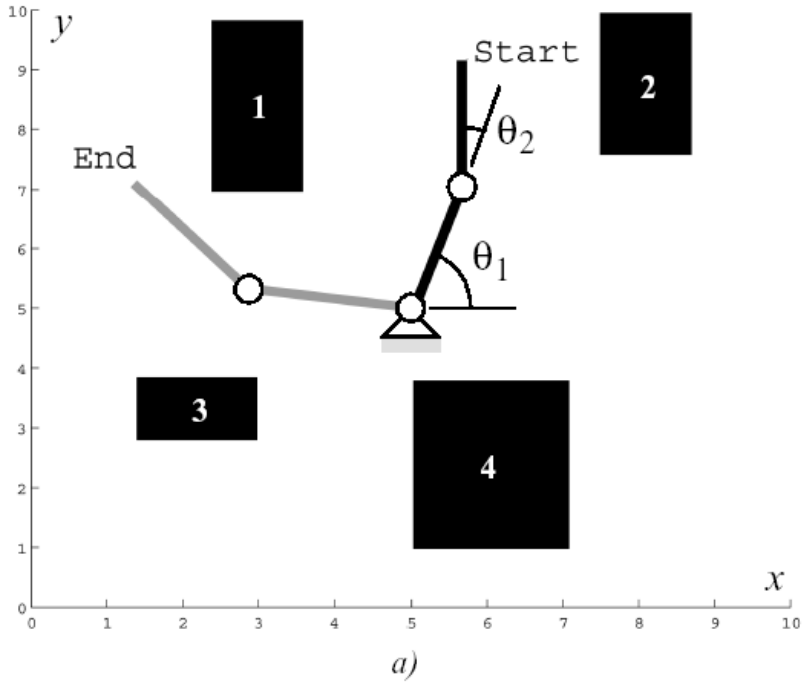


Notation

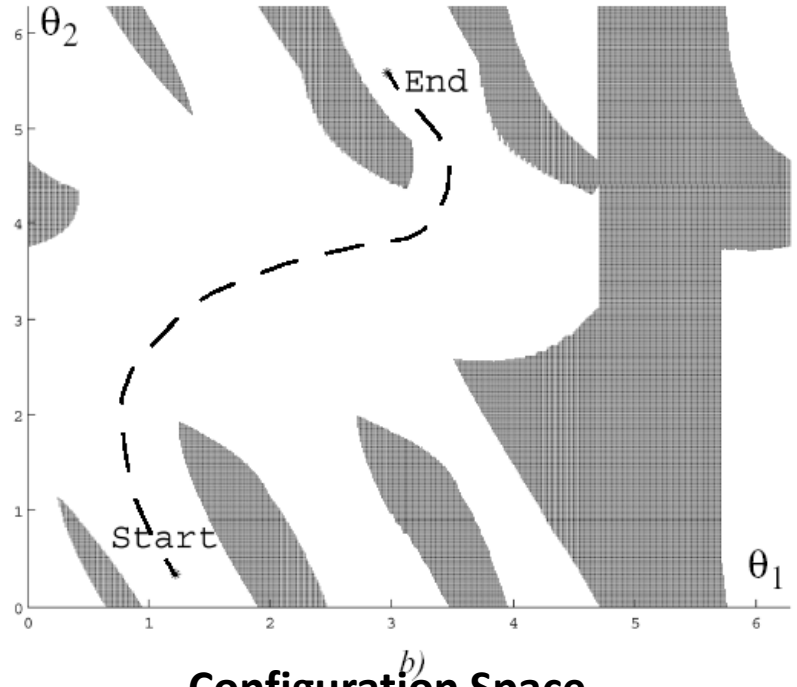
- Configuration space $\boldsymbol{C} \subseteq \mathbb{R}^{\boldsymbol{d}}$
- Configuration $\boldsymbol{q} \in \boldsymbol{C}$
- Free space $\boldsymbol{C_{free}}$
- Obstacle space $\boldsymbol{C_{obs}}$
- Properties
  - $\boldsymbol{C_{free}} \cup \boldsymbol{C_{obs}} = \boldsymbol{C}$
  - $\boldsymbol{C_{free}} \cap \boldsymbol{C_{obs}} = \boldsymbol{O}$

Work Space (Map) Vs Configuration Space

- State or configuration *q* can be described with *k* values *qi*



Work Space

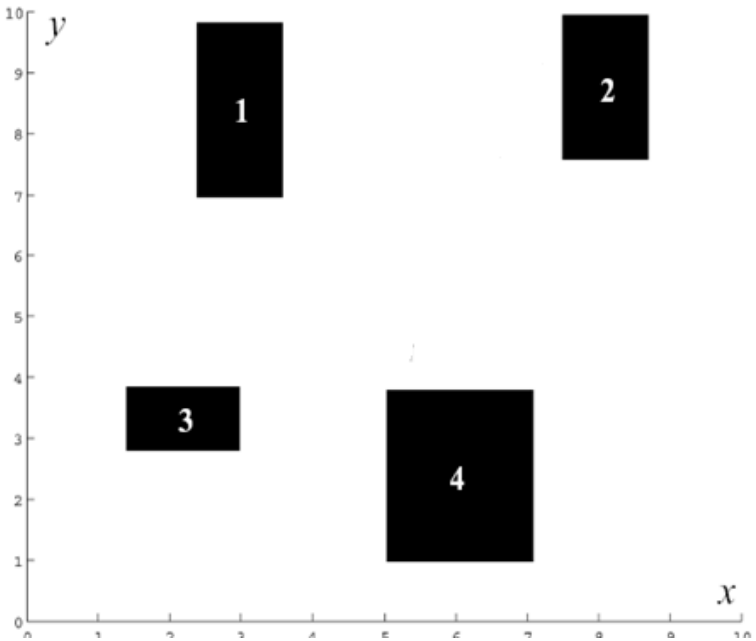


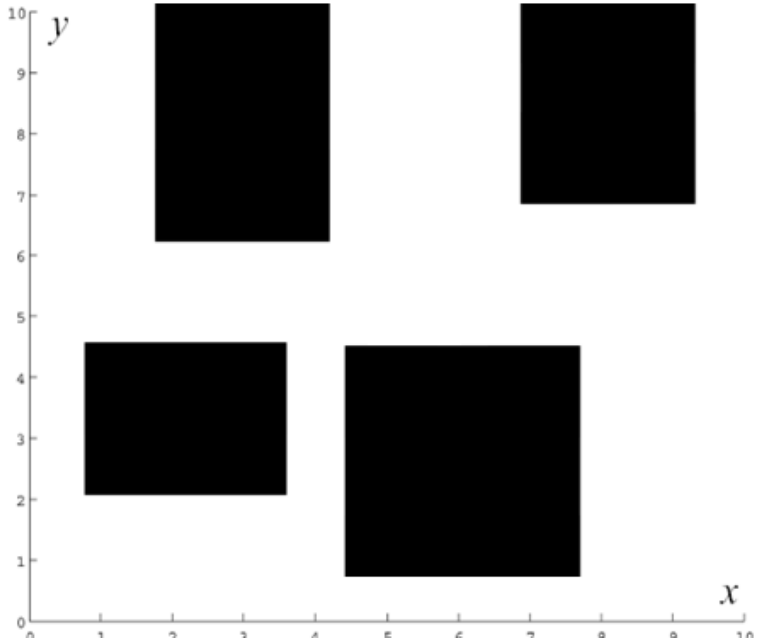
Configuration Space

the dimension of this space is equal to the Degrees of Freedom (DoF) of the robot

Configuration Space for a Mobile Robot

- Mobile robots operating on a flat ground have 3 DoF: (x, y, θ)
- For simplification, in path planning mobile roboticists often assume that the robot is holonomic and that it is a point. In this way the configuration space is reduced to 2D (x,y)
- Because we have reduced each robot to a point, we have to inflate each obstacle by the size of the robot radius to compensate

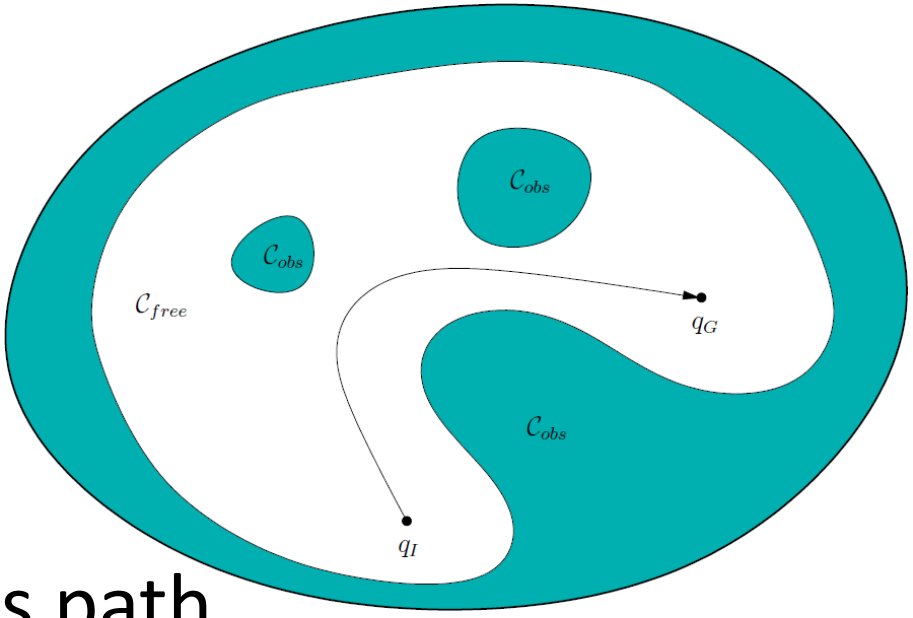




Configuration Space (C-Space)

- Work space
  - Position in 3D → 3 DOF
- Configuration space
  - Reduced pose (position + yaw) → 4 DOF
  - Full pose → 6 DOF
  - Pose + velocity → 12 DOF
  - Joint angles of manipulation robot
  - …
- Planning takes place in **configuration space**

Basic Motion Planning Problem



- **Given**
  - Free space $\boldsymbol{C_{free}}$
  - Initial configuration $\boldsymbol{q_I}$
  - Goal configuration $\boldsymbol{q_G}$
- **Goal:** Find a continuous path

$$\boldsymbol{\tau : [0\ ,1] \Rightarrow C_{free}}$$

with

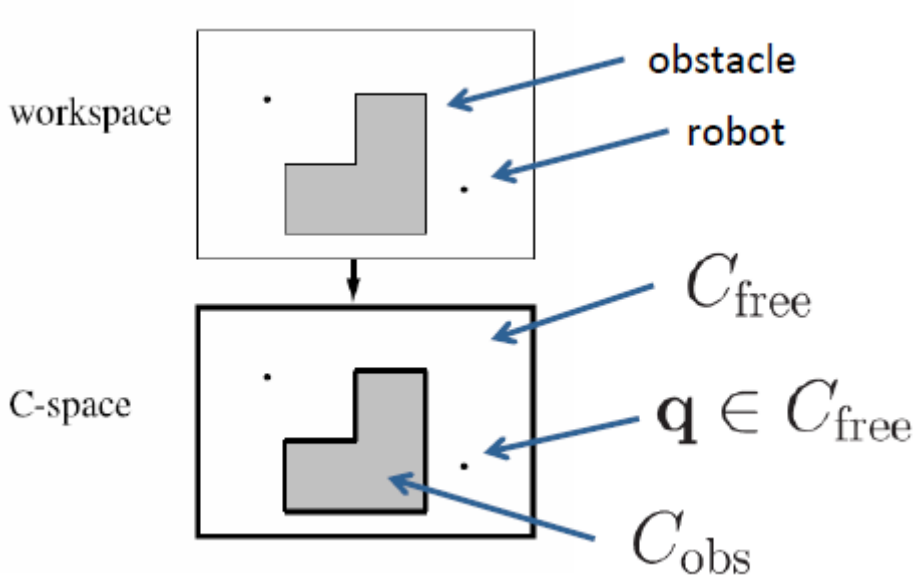
$$\boldsymbol{\tau(0) = q_I, \tau(1) = q_G}$$

Motion Planning Sub-Problems

- **C-Space discretization** (generating a graph / roadmap)
- Search algorithm (Dijkstra's algorithm, A*, …)
- Re-planning (D*, …)
- Path tracking (PID control, potential fields, funnels, …)

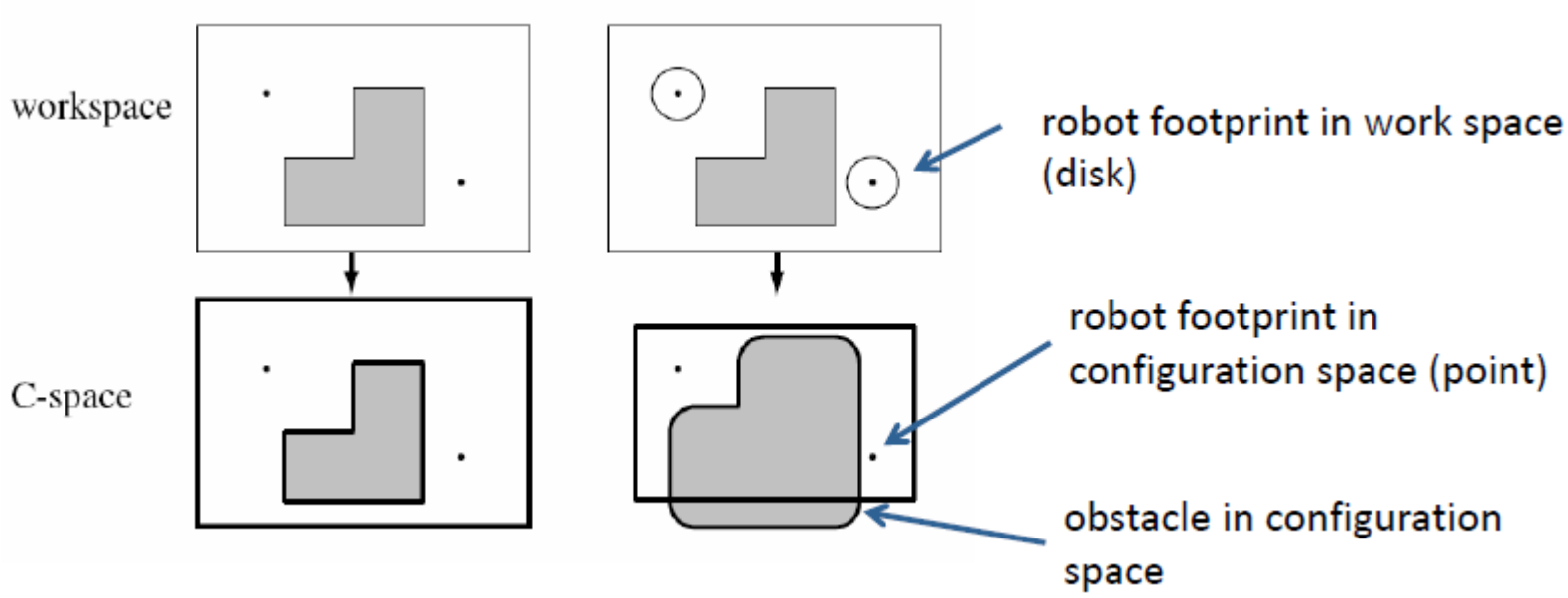
Example – Point Robot

- What are admissible configurations for the robot? Equiv.: What is the free space?
- "Point" robot



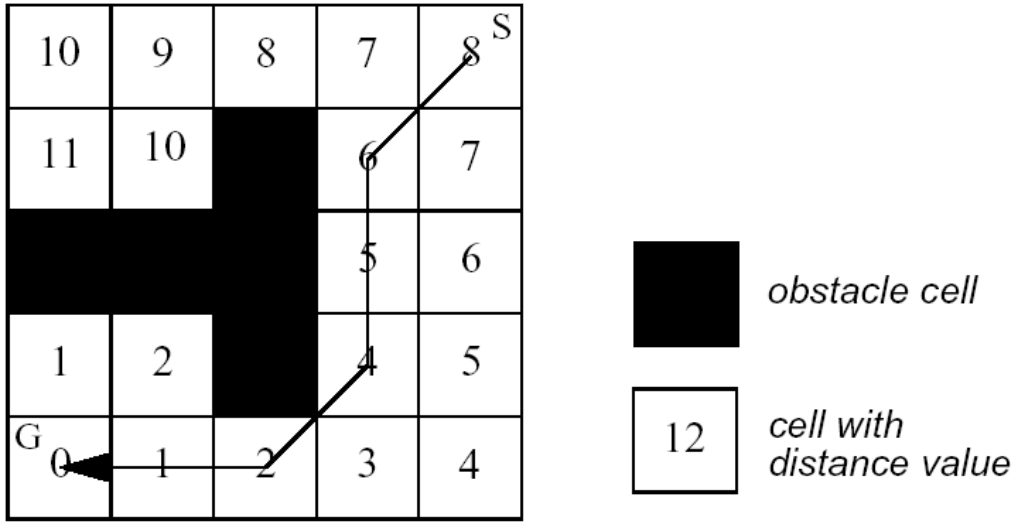
Example – Circular Robot

- What are admissible configurations for the robot? Equiv.: What is the free space?
- Circular robot



Path Planning – Graph Search

- Overview
  - Solves a least cost problem between two states on a (directed) graph
  - Graph structure is a discrete representation
- Limitations
  - State space is discretized → completeness is at stake
  - Feasibility of paths is often not inherently encoded
- Algorithms
  - (Preprocessing steps)
  - Greedy Algorithm
  - Dijkstra
  - A* and variants
  - D* and variants

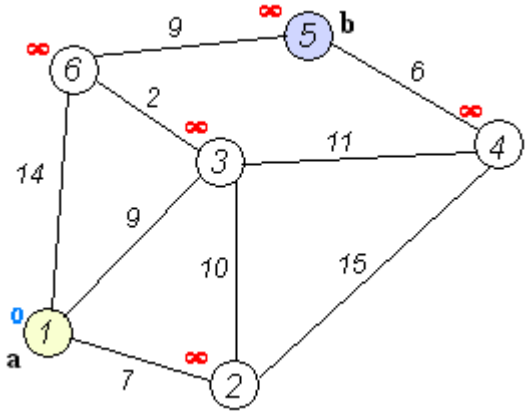


Greedy First Algorithm

- Greedy best-first algorithm takes into account the distance from current position to the target without considering the already travelled path distance, therefore, it reaches to the target as quickly as possible by using a heuristic function to guide its way toward the goal.
- But there is a caveat, the quickest path might be longer if there comes obstacles in the way and the algorithm has to re-plan the path. Therefore, the problem lies in looking only for shortest distance toward the goal and neglecting the already traversed distance from the start.

Dijkistra Algorithm

- It was designed by Edsger Dijkstra's in **1959**
- Dijkstra's algorithm takes into account only the distance travelled from **start position to the current position**, therefore, it try to reach to the target in a **shortest path** without considering the target direction which results it into longer time to find the shortest path.



A* Algorithm

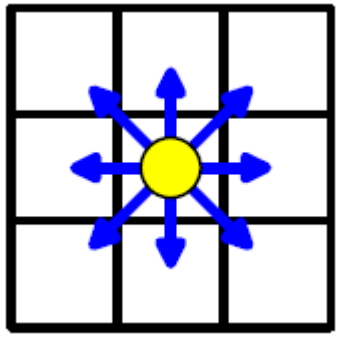
- A* is a path finding algorithm between two points on an OGM. It can work on rectangular, triangular, hexagonal or any other type of grid map. It is a **deterministic** algorithm which will find a path if one exists.
- Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first described the algorithm in **1968**
- A* **combines** the best of greedy best-first search algorithm and Dijkstra's algorithm

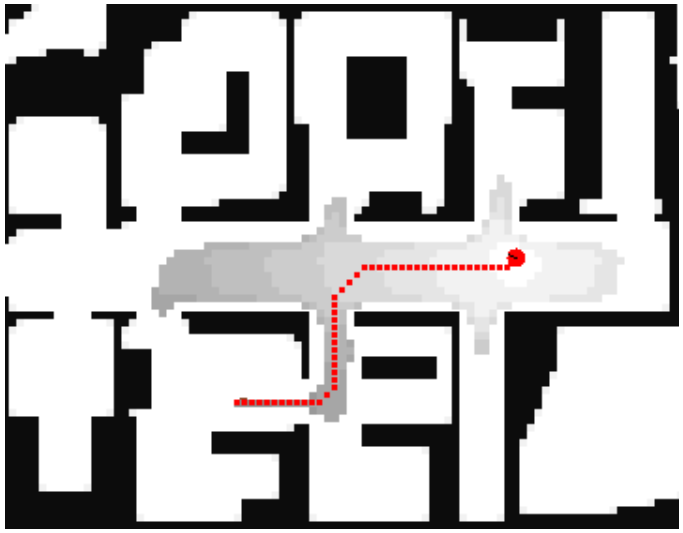
Path Planning with A*

- A* algorithm evaluates at current grid cell location (node) a heuristic function which consists of two parts as follows:

$$f(p) = g(p) + h(p)$$

- $g(p)$ is a function which returns the length of the already traversed path
- $h(p)$returns the estimate of an acceptable distance of the remaining path from current position to the target position





Path Planning with A* (Cont.)

- A* algorithm speed VS shortest path performance is depended on the chosen heuristic functions.
- If $h(p)$ is zero then A* algorithm act like Dijkstra's algorithm which results into the shortest path.
- On the other extreme if $h(p)$ is very high compared to $g(p)$ than it will try to find the path as quickly as possible without considering if it's the shortest, therefore act like Greedy Best First algorithm

Heuristic functions

- On a square grid where robot movements are limited to front, back, left and right use Manhattan distance (L_1-Norm).

$$h(p) = |p.x - e.x| + |p.y - e.y|$$

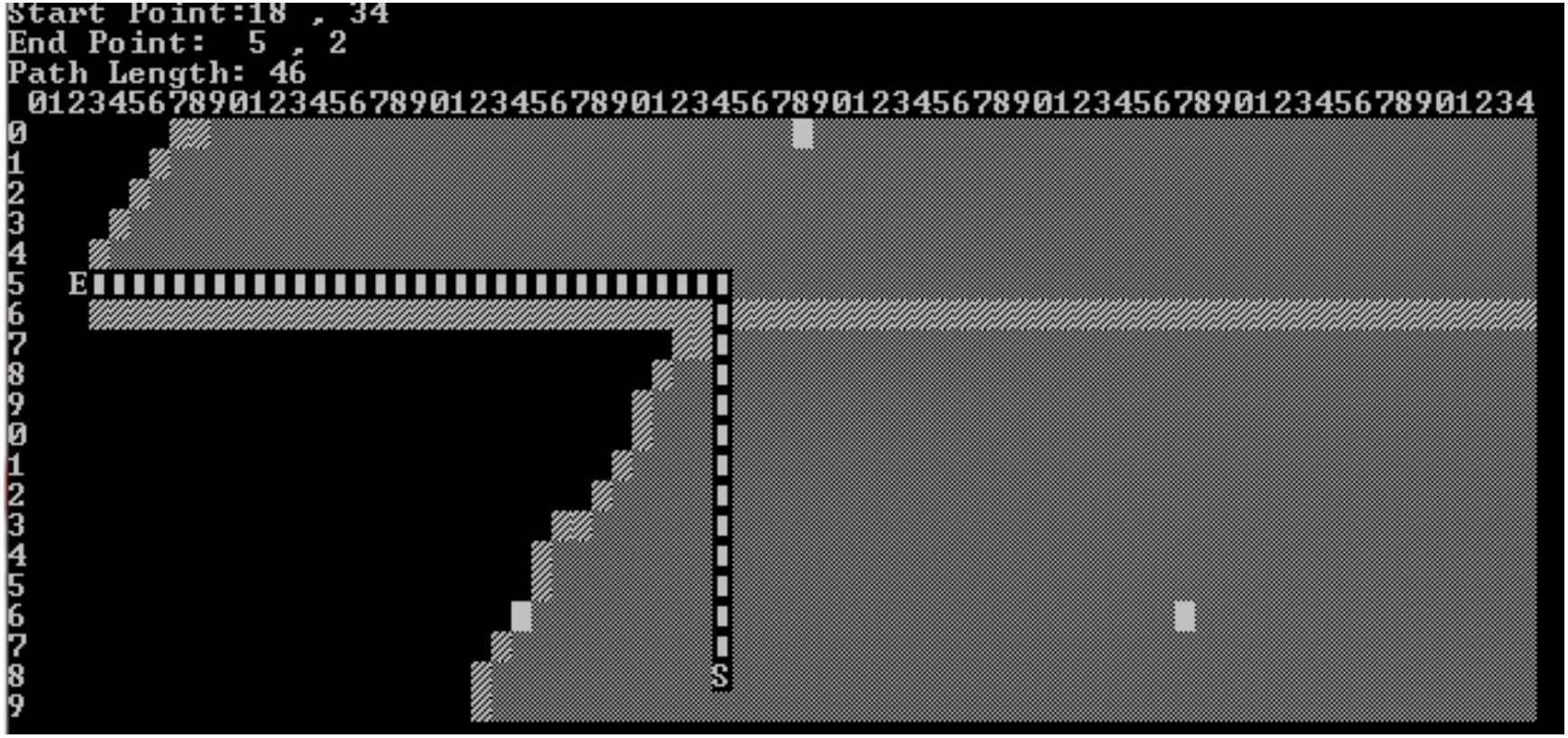
- If the robot can also move diagonal in addition to basic four movements then use the Diagonal or Chebyshev distance (L_∞-Norm)

$$h(p) = \max(|p.x - e.x|, |p.y - e.y|)$$

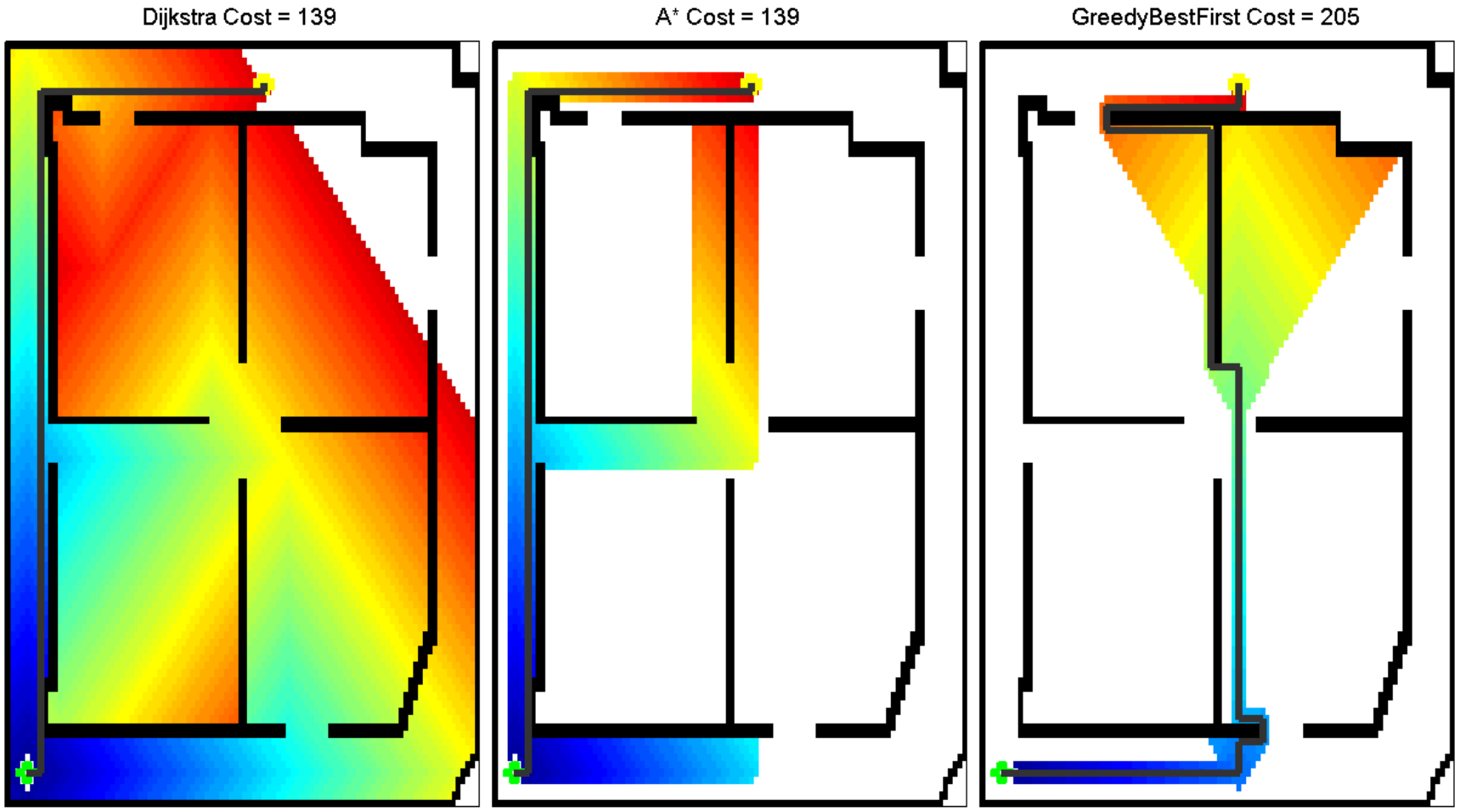
- If the robot can maneuvers in any direction then use the Euclidean distance (L_2-Norm).

$$h(p) = \sqrt{(p.x - e.x)^2 - (p.y - e.y)^2}$$

Example A*

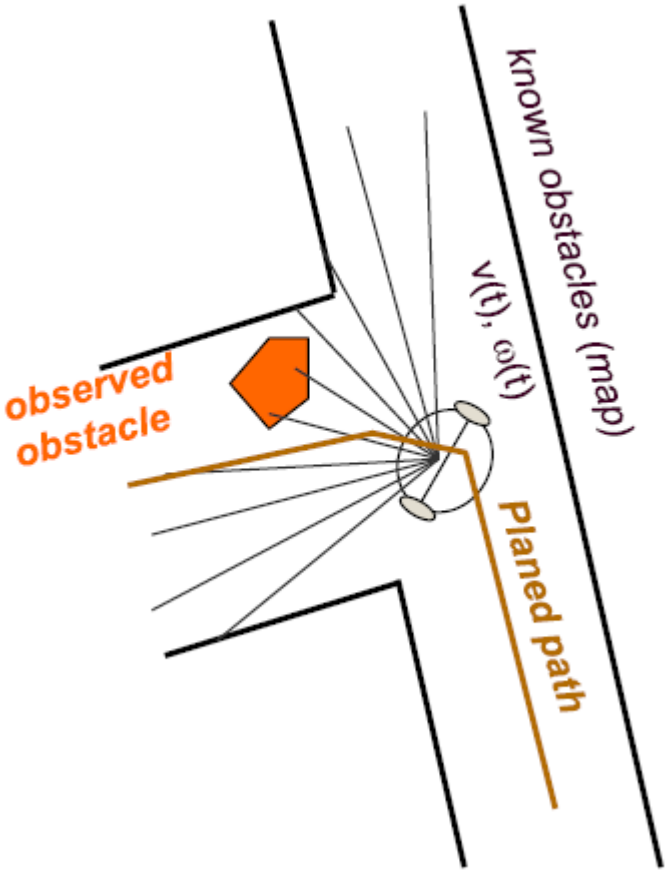


Comparison



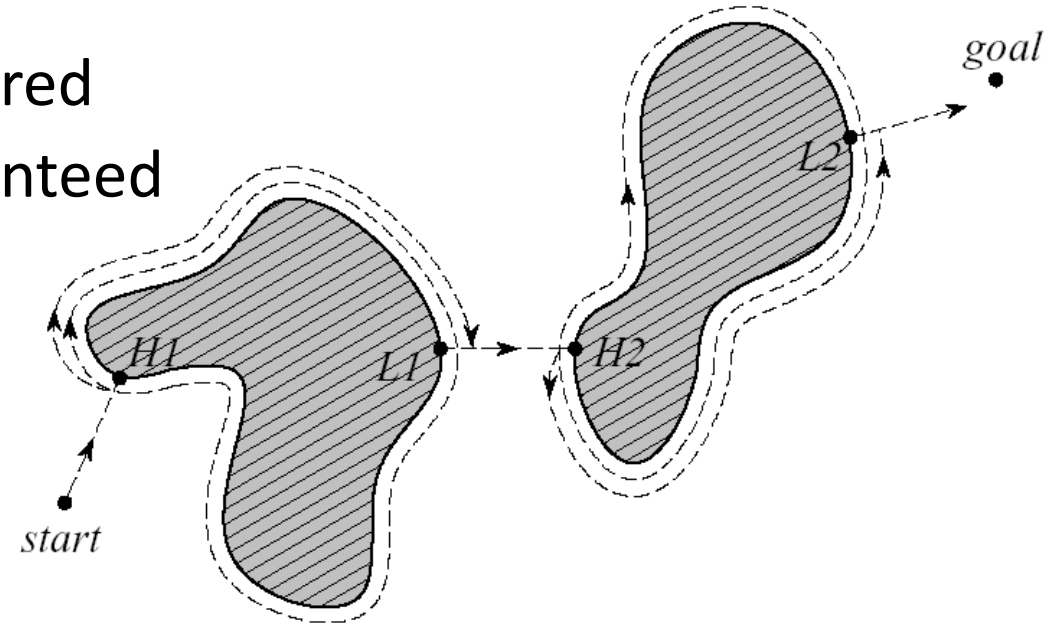
Obstacle Avoidance (Local Path Planning)

- The goal of obstacle avoidance algorithms is to avoid collisions with obstacles
- They are usually based on a local map
- Often implemented as a more or less independent task



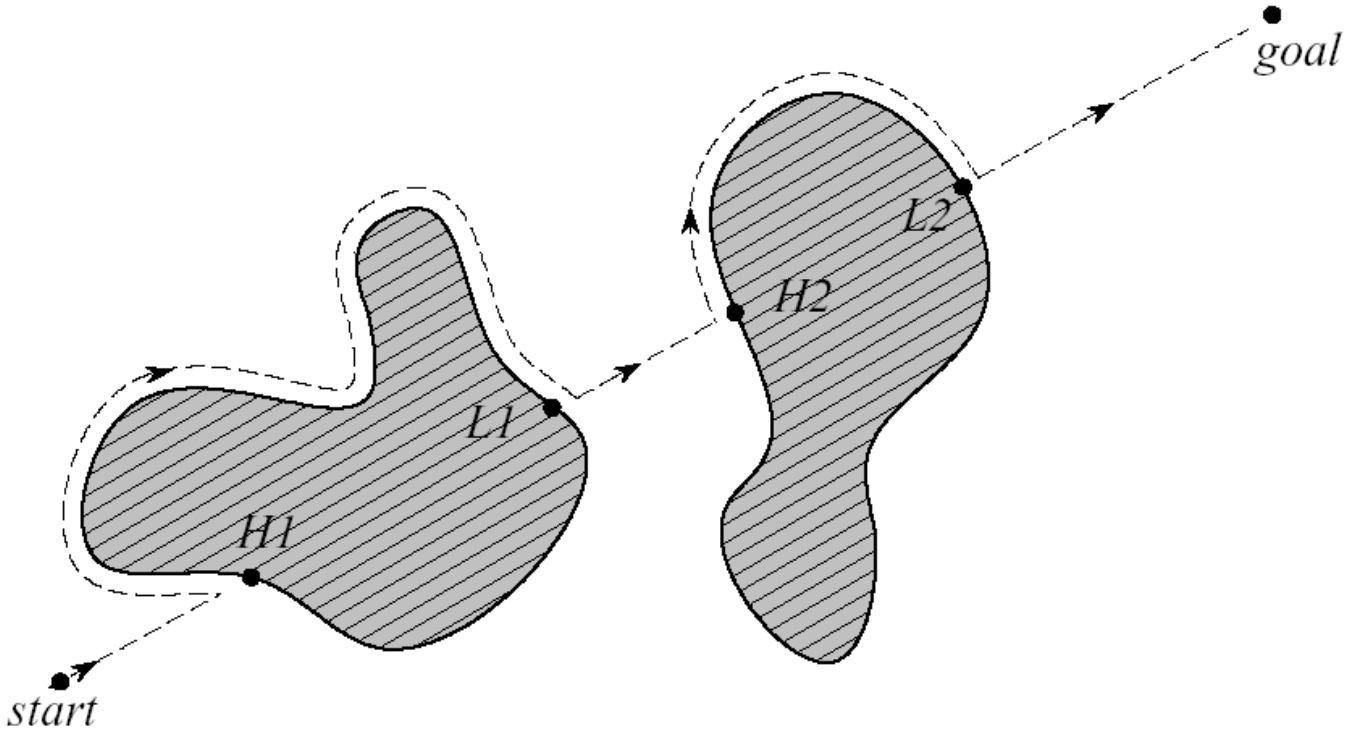
Obstacle Avoidance: Bug1

- Following along the obstacle to avoid it
- Each encountered obstacle is once fully circled before it is left at the point closest to the goal
- Advantages
  - No global map required
  - Completeness guaranteed
- Disadvantages
  - Solutions are often
  - highly suboptimal



Obstacle Avoidance: Bug2

- Following the obstacle always on the left or right side
- Leaving the obstacle if the direct connection between start and goal is crossed



Summary

- Configuration/work spaces,
- Path Planning algorithms:
  - Dijkstra,
  - Greedy First,
  - A*
- Obstacle avoidance: Bug Algorithms

Questions