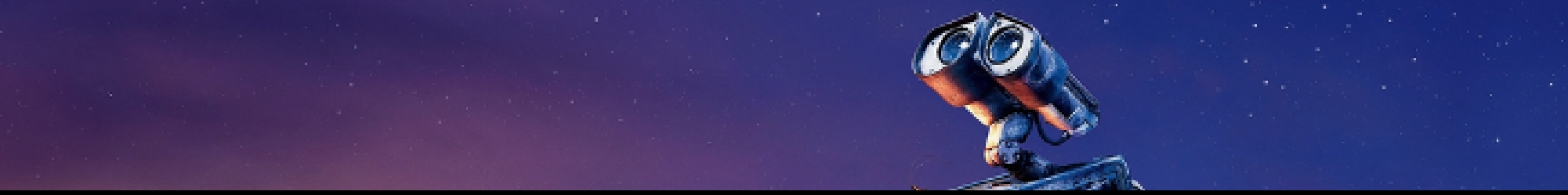


UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

Planning

RO51 - Introduction to Mobile Robotics

Zhi Yan

May 29, 2024

https://yzrobot.github.io/

www.utbm.fr

utbm
université de technologie
Belfort-Montbéliard

What?

- The task of coming up with a sequence of actions that will achieve a goal is called **planning**.
  - **Motion planning**: a process to find the movement steps from a starting state to a goal state.
  - **Task planning**: a process to find a series of actions taken to achieve a goal that cannot be achieved by a single action.
- Planning is a formalized procedure for decision-making.

Why?

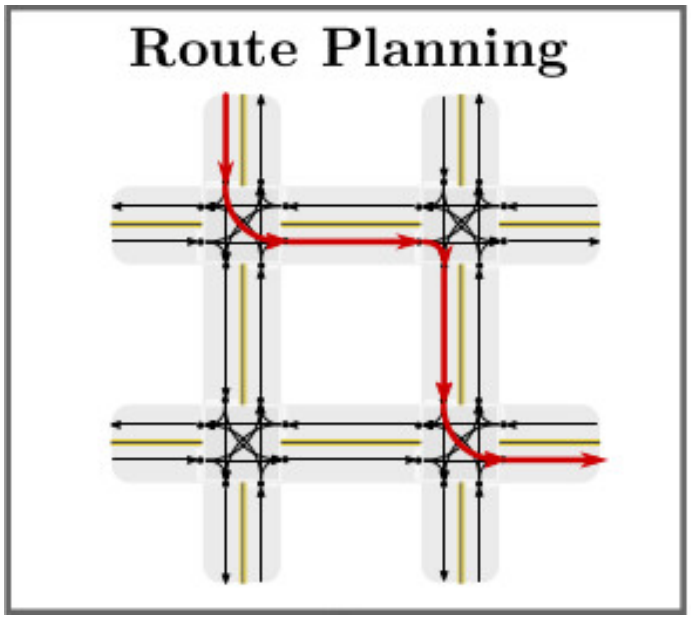
- Planning, whether for human society or the field of robotics, is an important means to achieve goals effectively.
- The essence of planning is an optimization problem.

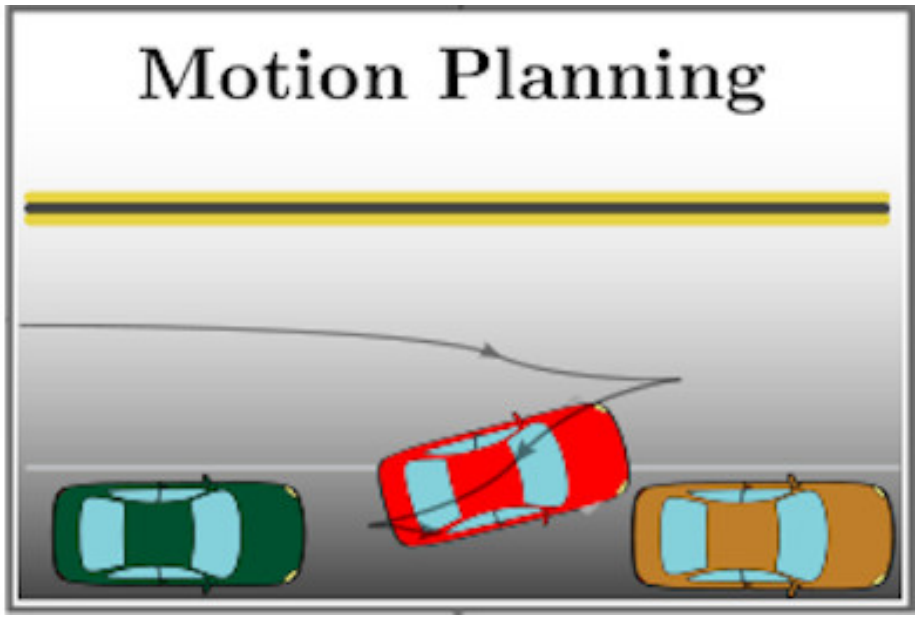
Motion planning

- In robotics, the motion planning problem involves producing a continuous robot motion from one configuration to another in a configuration space while avoiding collision with obstacles.
- Motion planning is eminently necessary for mobile robots since, by definition, a robot accomplishes tasks by moving in the real world.

Motion planning

- Motion planning vs. Path planning:
  - **Motion Planning**: concrete, e.g. take into account the real size of the robot, kinematics and dynamics models, etc.
  - **Path planning**: abstract, e.g. treat the robot as a point, ignore its kinematics and dynamics models, etc.





Motion planning

Configuration space

- In classical mechanics, the parameters that define the configuration of a system are called generalized coordinates, and the space defined by these coordinates is called the configuration space (or C-space) of the physical system.
- In robotics, a robot configuration is a specification of the positions of all robot points relative to a fixed coordinate system, and the space of all configurations of the robot is called the configuration space (or C-space) of the robot.
- Simply speaking, the C-space is the space related to the structure and shape of the robot.

Motion planning

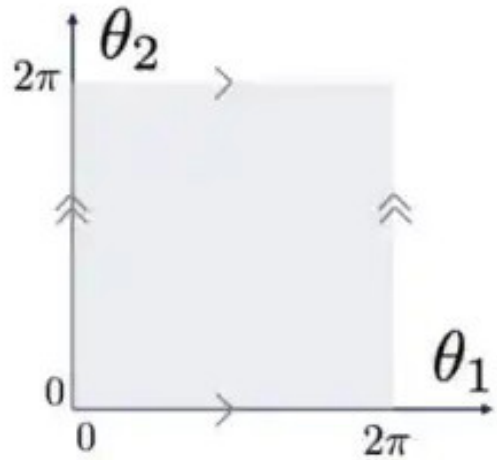
Configuration space

- The topology of C-space is usually not that of a Cartesian space.
- The C-space is described as a **topological manifold**.

2DOF robot arm

Torus

$C=S^1 \times S^1$



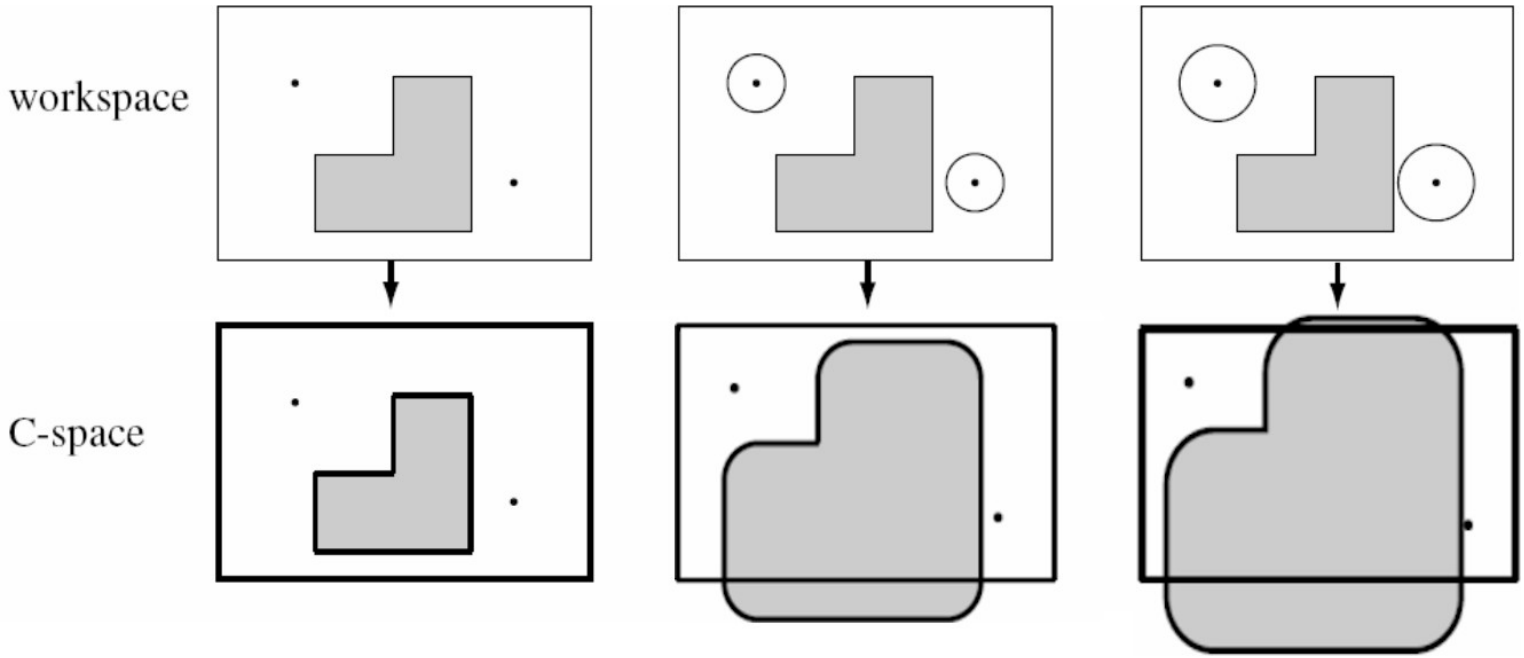
$[0, 2\pi) \times [0, 2\pi)$

Note that the two edges with arrows are glued together.

Motion planning

Configuration space

- The robot and obstacle geometry is described in a 2D or 3D workspace, while the motion is represented as a path in C-space.
- Example: circular robot

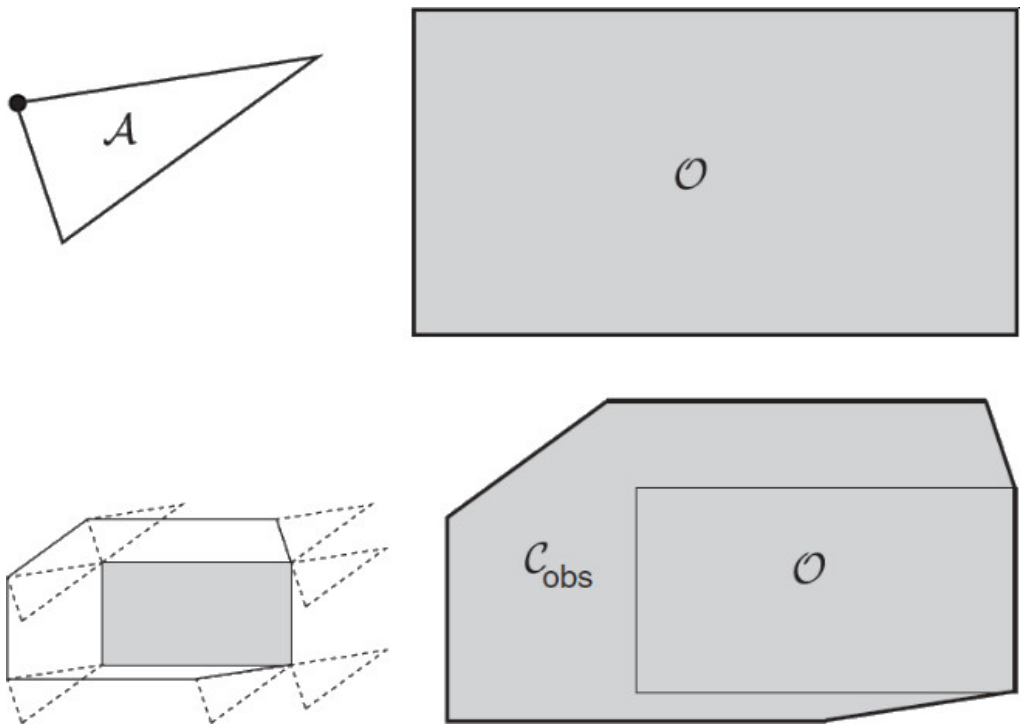


C-space is obtained by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius.

Motion planning

Configuration space

- Example: polygonal robot, translation only

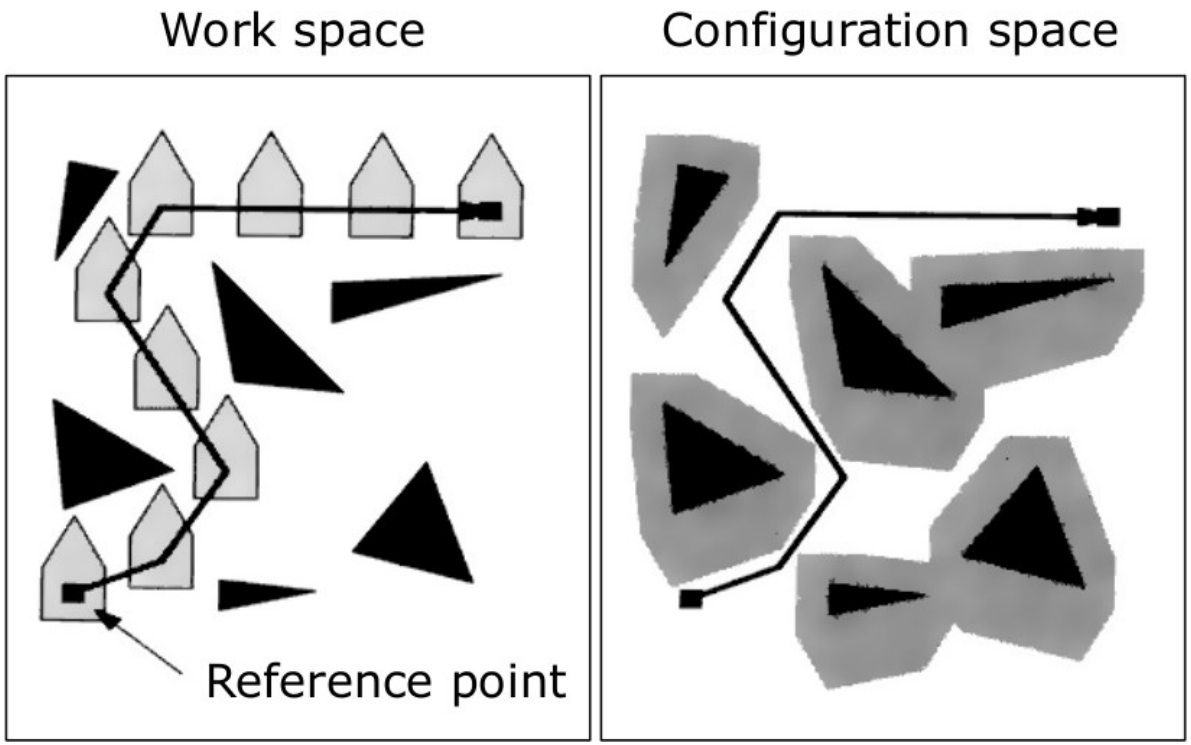


C-space is obtained by sliding the robot along the edge of the obstacle regions.

Motion planning

Configuration space

- Example: polygonal robot, translation only

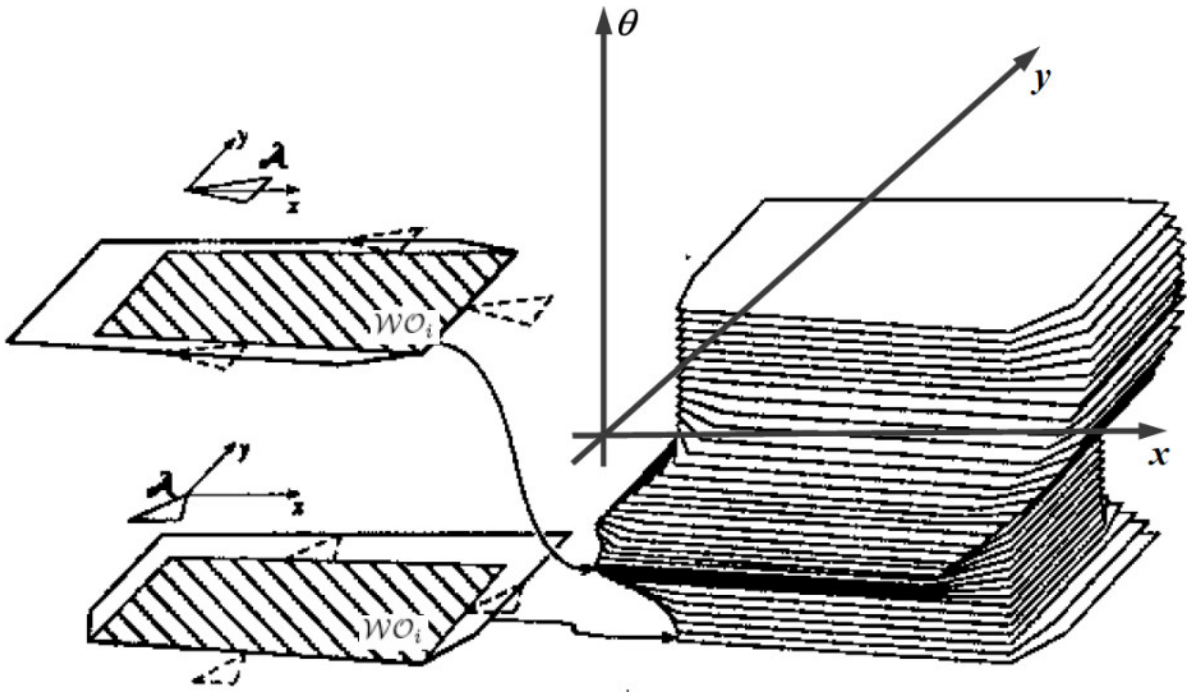


C-space is obtained by sliding the robot along the edge of the obstacle regions, and in C-Space, the robot can be represented as a point (vector).

Motion planning

Configuration space

- Example: polygonal robot, translation + rotation



C-space is obtained by sliding the robot along the edge of the obstacle regions in all orientations.

Motion planning

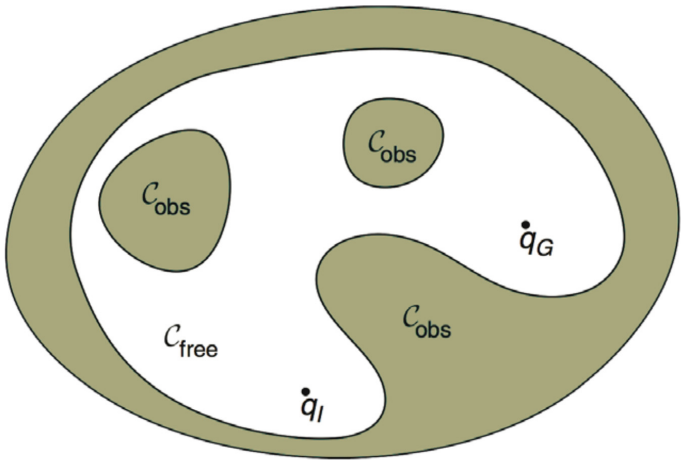
Configuration space

- For robots considered in 2D space, in simple terms, C-Space is the **expansion** of the workspace based on the size of the robot.
- Formally, the C-space is the special Euclidean group $SE(2) = R^2 \times SO(2)$ (where $SO(2)$ is the special orthogonal group of 2D rotations), and a configuration can be represented using three parameters (x, y, θ).
- If considering a robot in 3D space (e.g. drones), the C-space is the special Euclidean group $SE(3) = R^3 \times SO(3)$, and a configuration requires six parameters: (x, y, z) for translation, and Euler angles (α, β, γ).
- For a fixed-base robotic arm, the C-space (i.e. joint space) is N-dimensional, where N corresponds to the number of its joints.

Motion planning

Configuration Space

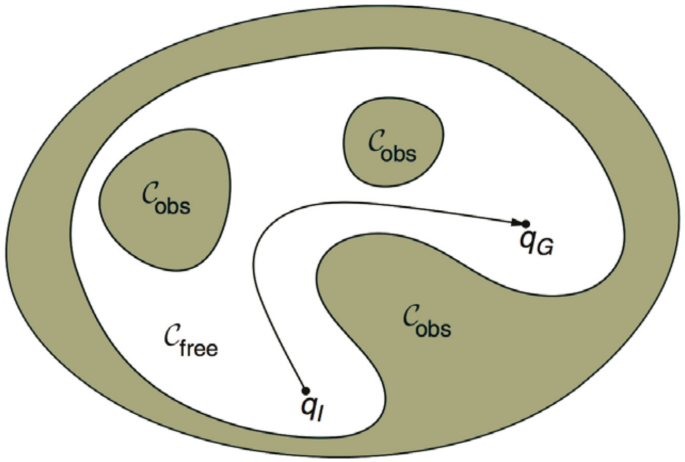
- Two sets: free space and obstacle region
- With $W = \mathbb{R}^m$ being the work space, $O \in W$ the set of obstacles, $A(q)$ the robot in configuration $q \in C$:
  $C_{free} = \{q \in C | A(q) \cap O = \emptyset\}$
  $C_{obs} = C / C_{free}$
- We further define:
  q_I: start configuration
  q_G: goal configuration



Motion planning

Configuration Space

- Then, motion planning amounts to finding a continuous path $\tau : [0, 1] \rightarrow C_{free}$ with $\tau(0) = q_I, \tau(1) = q_G$
- Given this setting, we can do planning with the robot being a point in C-space.



Motion planning

Static vs Dynamic environments

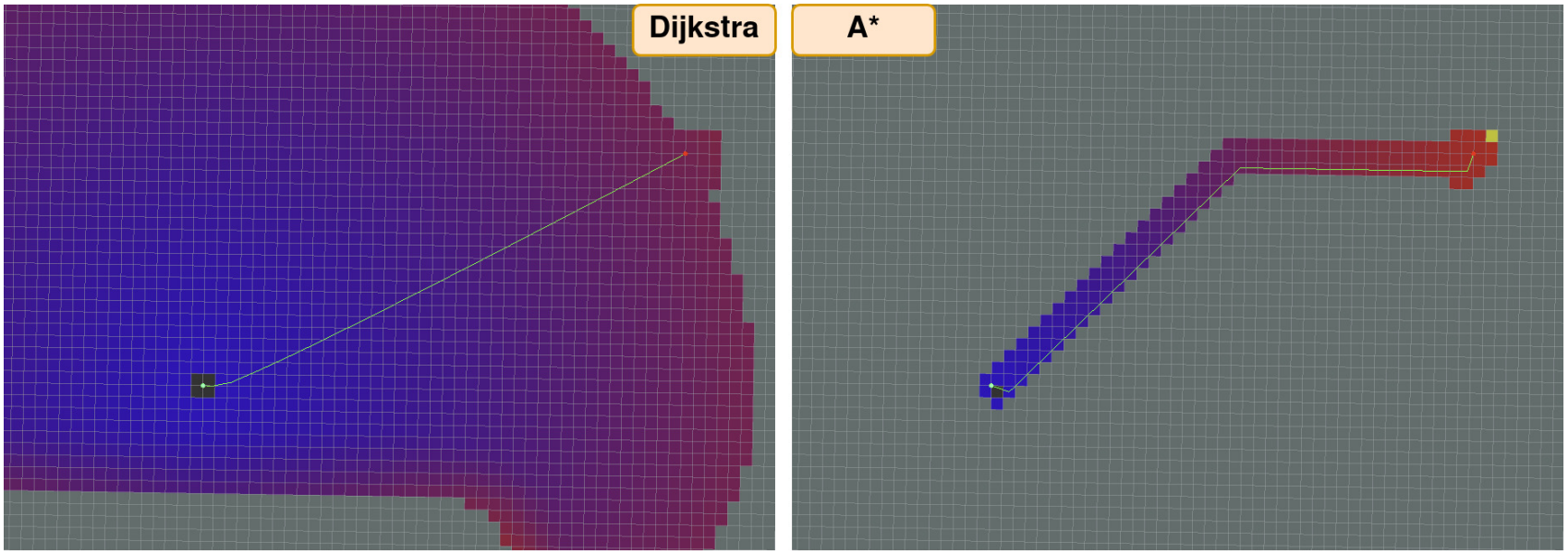
- In a static environment, all objects except the robot are static.
  - One-shot (global) path planning
  - Representative methods: Dijkstra, A*
- In a dynamic environment, besides the robot itself, there are other objects moving, such as humans.
  - Constantly re-planning the path
  - Representative methods: Dynamic A* (D*) <= originally designed for Mars rover, two-layered planning: global planning (for path finding) + local planning (for obstacle avoidance) such as A* + Dynamic Window Approach (DWA)
- Sampling-based methods (e.g. Rapidly-exploring Random Tree (RRT), Probabilistic RoadMap(RPM)) can be used in both static and dynamic environments.

A specific discussion of each algorithm is beyond the scope of this lecture (c.f. https://yzrobot.github.io/#Research, http://wiki.ros.org/navigation).

Motion planning

Global planning

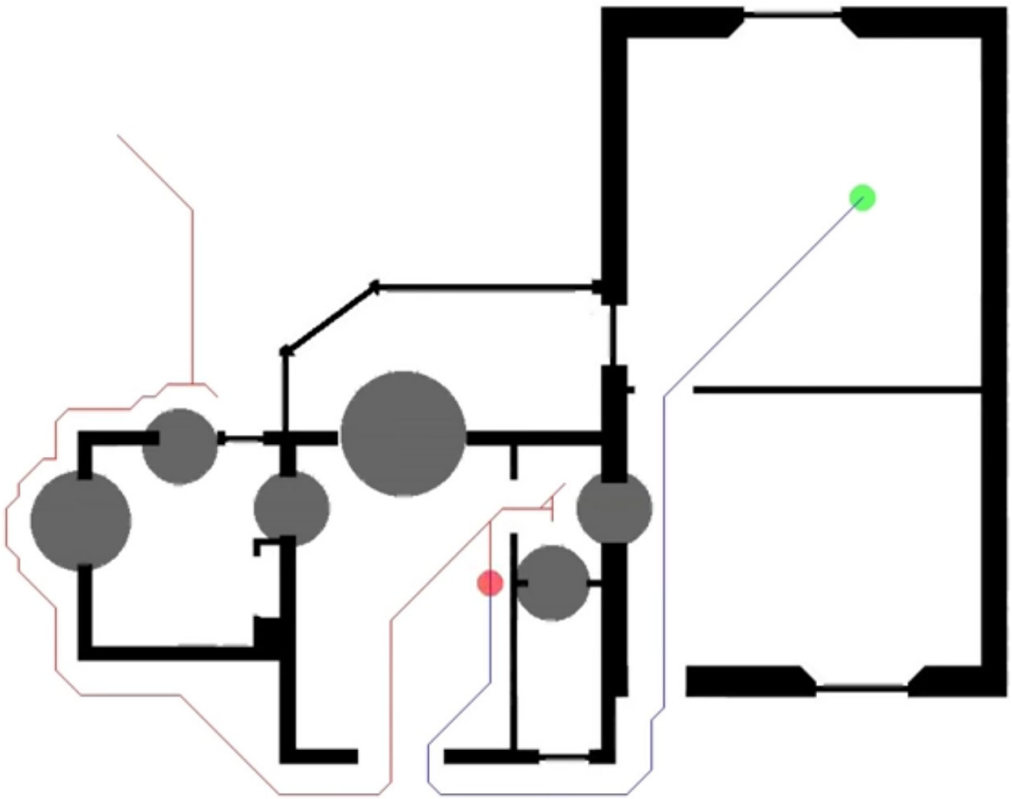
- **Dijkstra**: breadth-first search-like approach, solves the single-source shortest path problem in a weighted directed graph.
- **A***: heuristic algorithm, an upgraded version of Dijkstra, $f(n) = g(n) + h(n)$.



Motion planning

In dynamic environments

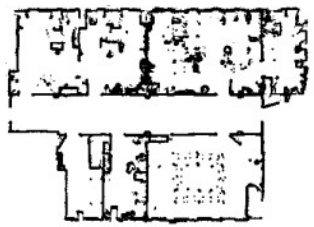
- **D***: search the path from the target position to the starting position (opposite to the A* algorithm).



Motion planning

In dynamic environments

Two-layered robot motion planning architecture:



map

sensor data

Planning

low frequency

sub-goal

Collision Avoidance

high frequency

motion command

robot

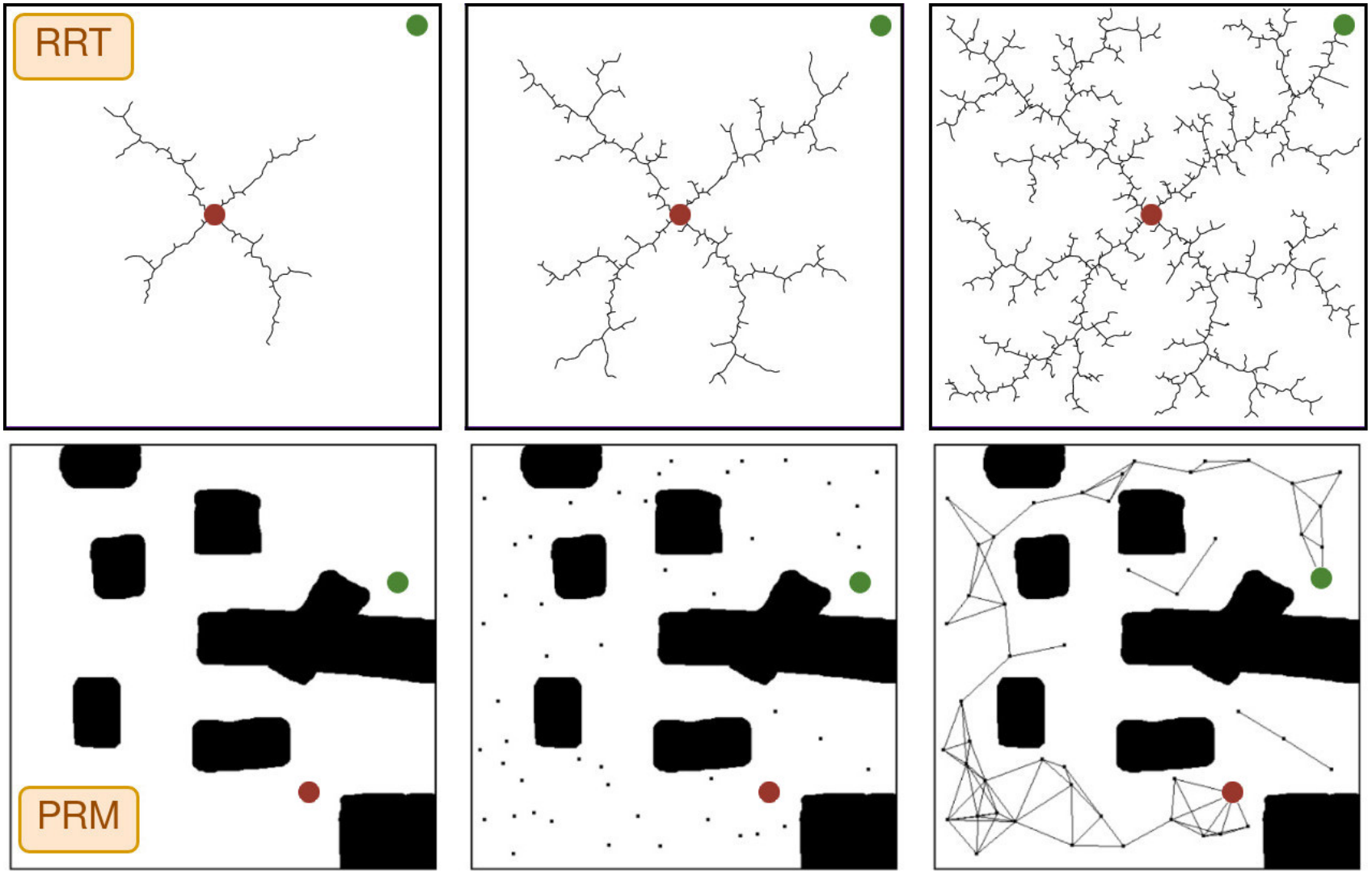
Motion planning

Sampling-based methods

- Stochastic method, effectively dealing with path search in high-dimensional space.
- Probabilistically complete and does not guarantee optimal paths.
- Suffer from narrow passages: the sampling point has only a small probability of falling inside.

Motion planning

Sampling-based methods



Task planning

- Recall: robots need task planning to sequence actions to achieve a goal that is not possible with a single action.
  - Finite State Machine (FSM)[1]
  - Planning Domain Definition Language (PDDL)[2]
  - Answer Set Programming (ASP)

[1] FSM in ROS: `http://wiki.ros.org/executive_smach`
[2] PDDL in ROS: `https://kcl-planning.github.io/ROSPlan/`

Task planning

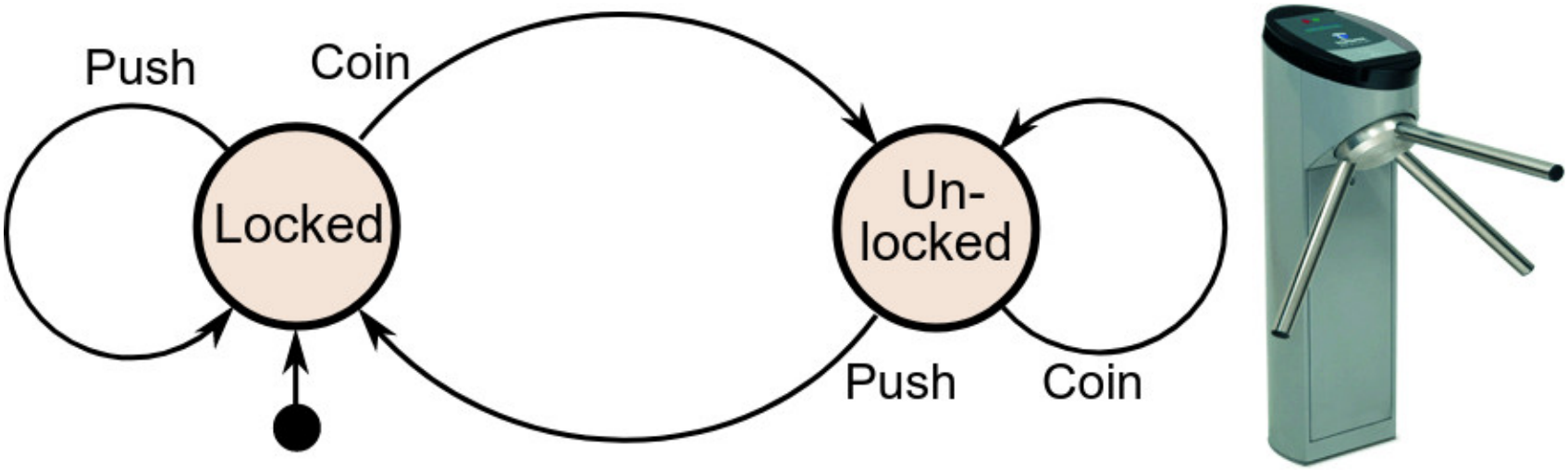
An instance

A typical robot tidy-up task flow:

1. The robot enters the workspace and explores it to get where the objects are scattered.
   - Note: Rolling over objects is not allowed. <= Avoid all obstacles at all times!
2. The robot selects an object and grabs it. <= Which one?
3. The robot brings the object to the designated area.
4. The robot observes and chooses a suitable location to deliver the object.
5. The robot continues the grab and delivery process until all objects are sorted.

Task planning

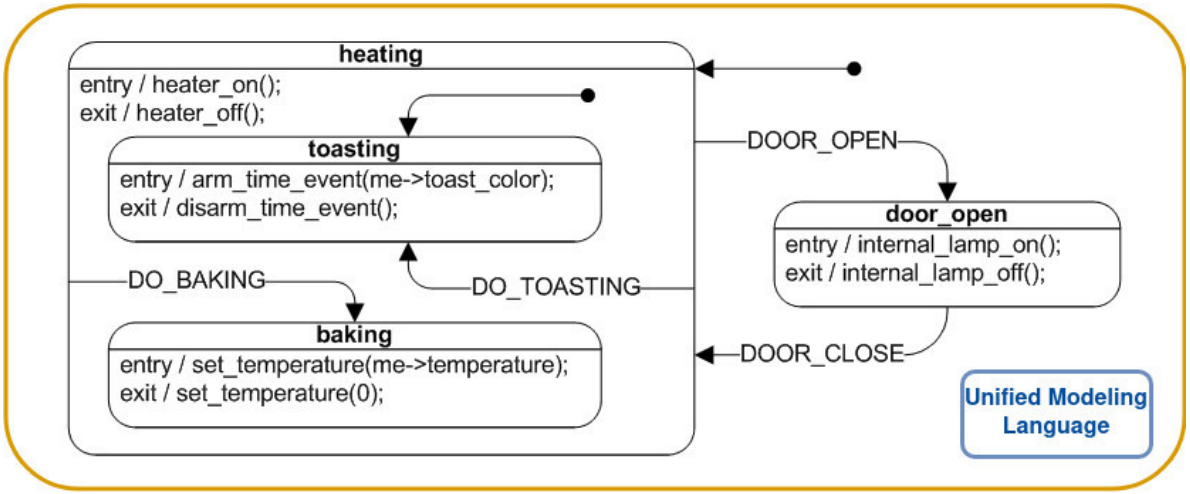
Finite State Machine (FSM)

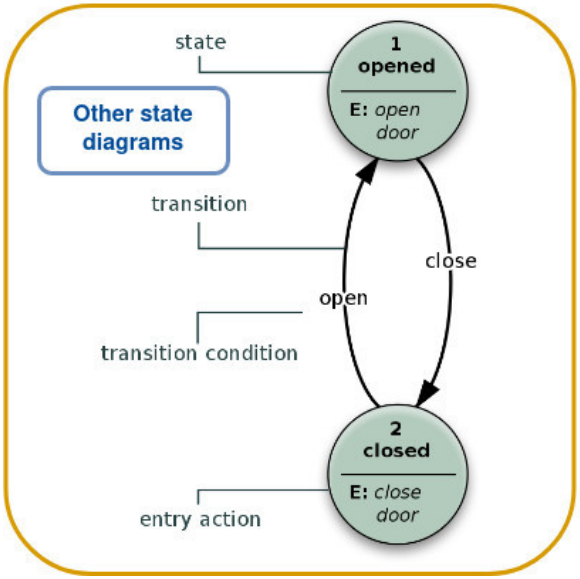


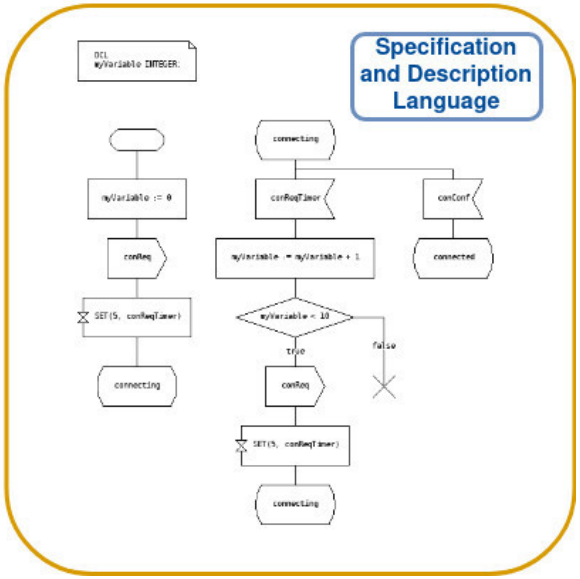
- Two **states**: Locked & Unlocked
- **Initial state**: Locked
- Two **inputs** that trigger each transition: Coin & Push <= The change from one state to another is called a transition.

Task planning

Graphical representation of FSM







Task planning

FSM in table form

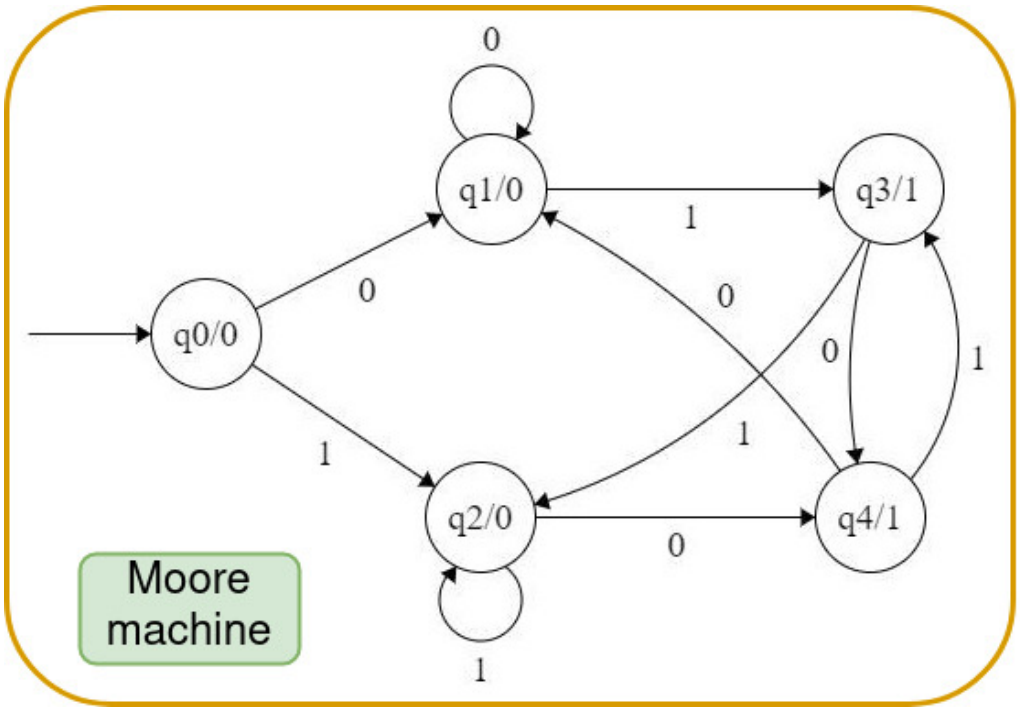
State-transition table

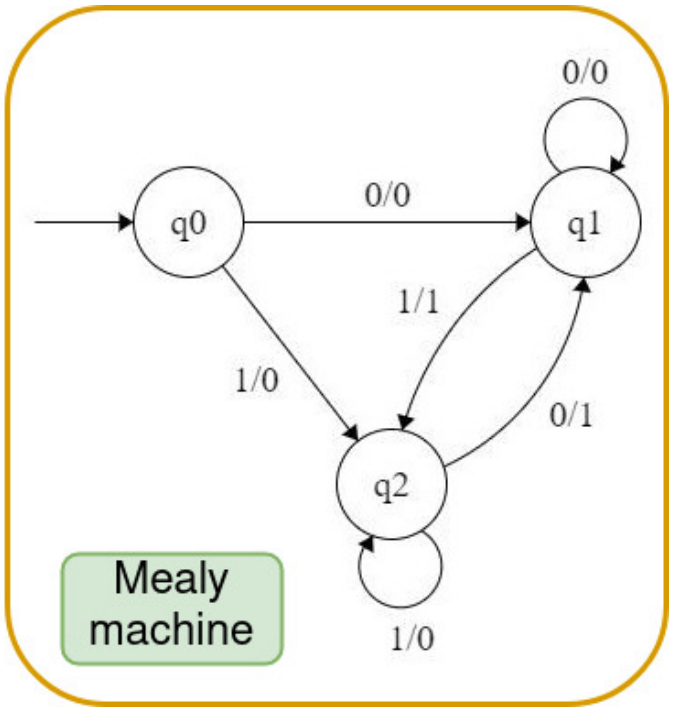
Current state / Input	State A	State B	State C
Input X	...	...	...
Input Y	...	State C	...
Input Z	...	...	...

Task planning

Two types of FSM

- **Moore type**: output depends only on state.
  - Simplified behavior but may require more state.
- **Mealy type**: output depends on both input and state.
  - Less state but possibly more complex behavior.





Task planning

Planning Domain Description Language (PDDL)

- PDDL is a standard **encoding language** for "classical" planning tasks.
- Components of a PDDL planning task:
  - **Objects**: Things in the world that interest us.
  - **Predicates**: Facts that we are interested in (e.g. properties of objects), which can be true or false.
  - **Initial state**: The state of the world that we start in (i.e. things that are true at the start).
  - **Goal specification**: The state of the world we want to end at (i.e. things that we want to be true at the end).
  - **Actions/Operators**: Ways of changing the state of the world (i.e. things that happen that change the facts).

Task planning

Form of PDDL

- PDDL files are text documents and usually have the extension *.pddl*.
- Planning tasks specified in PDDL are separated into two files:
  - A **domain file** for predicates and actions.
  - A **problem file** for objects, initial state and goal specification.

```
(define (domain <domain name>)
  (:predicates
    <predicate-list>
  )

  (:action
    <action-details>
  )
)
```

```
(define (problem <title>)
        (:domain <domain-name>)
        (:objects
        <object-list>
        )

        (:init
                <predicates>
        )
        (:goal
                <predicates>
        )
)
```

Task planning

PDDL for tidy-up

- **Objects**: A corridor, a room, a delivery area, 10 objects, a robot.
- **Predicates**: Is the robot inside the room? Is x the delivery area? Is y an object? Is the robot arm empty? etc.
- **Initial state**: The robot is in the corridor, the robot arm is empty, all objects are in the room, etc.
- **Goal specification**: All objects must be sorted out.
- **Actions/Operators**: The robot can move, pick up an object or drop an object.

Task planning

Answer Set Programming (ASP)

- ASP is a form of **declarative programming** (e.g. SQL) oriented towards difficult (primarily $NP - hard$) search problems.
- ASP tells the computer (robot) "what you (human) want to do", not "how to do it".
- A simple ASP program consists of three parts:
  - **Facts** and **rules**: describe the problem.
  - **Outputs**: check the results.

Task planning

Form of ASP

- ASP is syntactically similar to traditional logic programming and semantically close to non-monotonic logic.
- ASP applies important rules in non-monotonic reasoning: **closed world assumption** (CWA) and **negation as failure** (NAF).
- All propositions that are not stated to be true are assumed to be false by default.
- if "A" is judged not to be inferred, then "not A" is inferred. <= non-monotonic logic

Task planning

Form of ASP

- Consider a knowledge base consisting of information about service robots, cooking robots, and some individuals:
  - Most service robots move.
  - Cooking robots do not move.
  - Cooking robots are service robots.
  - HSR is a service robot.
  - Moley is a cooking robot.
- By ASP:
  - $move(X) \leftarrow service\_robot(X), not \neg move(X)$
  - $\neg move(X) \leftarrow cooking\_robot(X)$
  - $service\_robot(X) \leftarrow cooking\_robot(X)$
  - $service\_robot(HSR) \leftarrow$
  - $cooking\_robot(Moley) \leftarrow$

Summary

- The planning of mobile robots mainly includes motion planning and task planning.
- The knowledge of configuration space is the basis for implementing robot motion planning.
- Global planning is to find a path from the robot's current position to the target position, while local planning is to avoid dynamic obstacles in the environment during movement.
- Task planning and other areas of AI have more in common.
- Extended reading: complete traversal path planning (e.g. for sweeping robot)

The end

Thank you for your attention!

Any questions?