The Basic Frameworks for Embodied AI

Hao Su

Based on CVPR 2022 Tutorial on “Building and Working in Environments for Embodied AI”

Outline

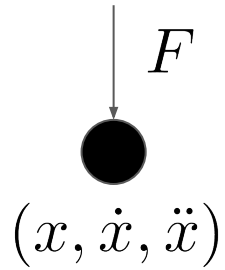
- Modeling and approaches for Embodied AI
  - World model
  - Learning-based methods to solve tasks
  - Classic robotics

How do People Model the World?

x Position

$\dot{x} := dx/dt$ Velocity

$\ddot{x} := d\dot{x}/dt$ Acceleration



Newton's second law of motion

$$F = m\ddot{x}$$

How do People Model the World?

If we call $(x, \dot{x})$ the **state** of the world,

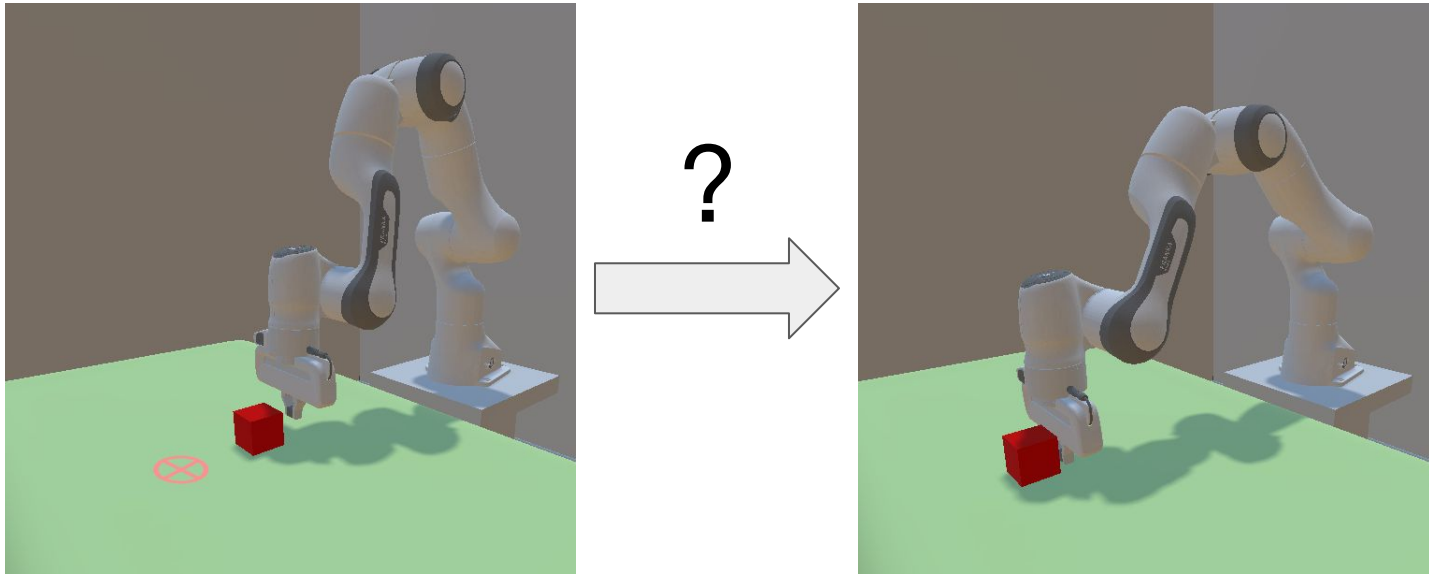
And F an **action** on the world.

Newton's second law models the **transition** of state under action over time.

$$\frac{d}{dt}(x, \dot{x}) = (\dot{x}, \frac{F}{m})$$

An Embodied AI Example

Task: push block to target location.

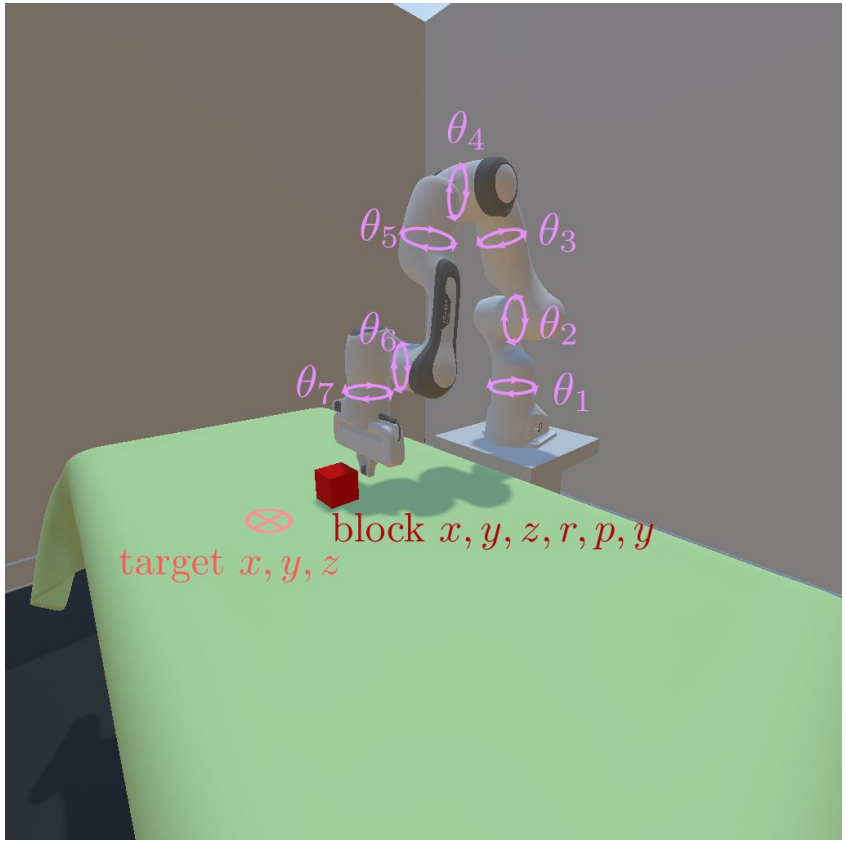


States

A **state** is a configuration of the world.

- In this example
  - Joint angles θ1-θ7
  - block position and orientation
  - target position

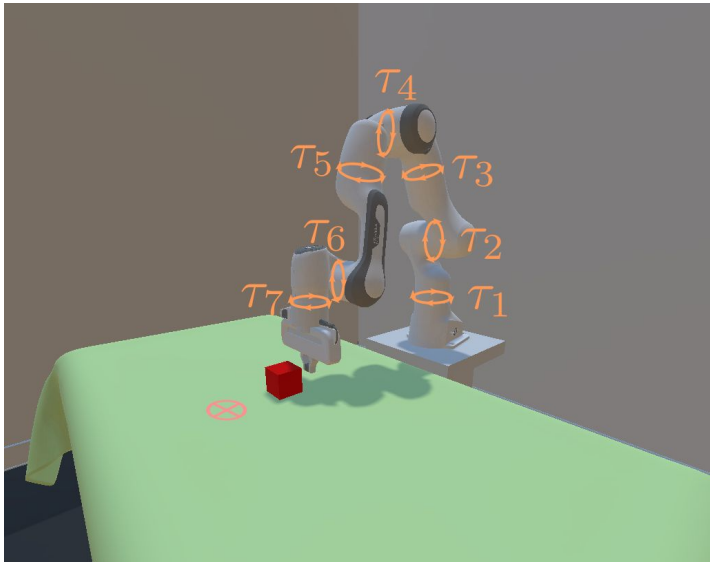
The collection of all states is called the state space $\mathcal{S}$.



Actions

An **action** is a robot command.

- For example
  - Motor torque



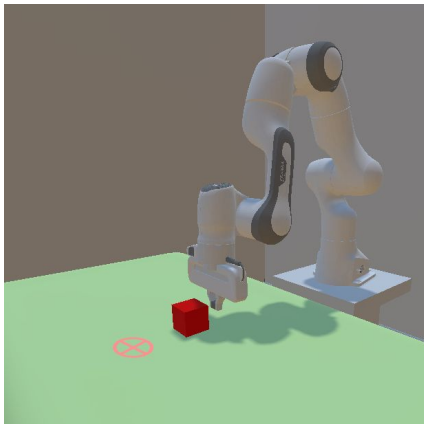
The collection of all actions is called the action space $\mathcal{A}$.

Transition

The **transition function** $\mathcal{T}$ describes how the **state** changes over time according to an **action**.

Formally, $\mathcal{T}$ describes the rate of change of the state given the current state and action.

$$\dot{s} := \frac{ds}{dt} = \mathcal{T}(s, a)$$



Transition function: classical mechanics

The Forward Model

The forward model is a 3-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T})$

$\mathcal{S}$: State Space — all possible world states

$\mathcal{A}$: Action Space — all possible control signals

$\mathcal{T}$: Transition — environment dynamics

Modeling Transition on a Computer

On a computer, things are discrete.

$$\dot{s} = \mathcal{T}(s, a) \xrightarrow{\text{Discretize over time}} s_{t+1} = \hat{\mathcal{T}}(s_t, a_t)$$

We call 1/Δt as the action frequency

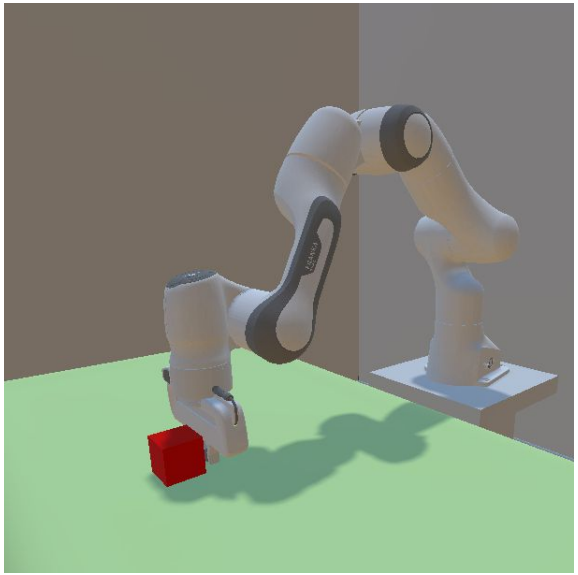
In general, the transition can be stochastic.

$$s_{t+1} \sim \mathcal{T}(\cdot | s_t, a_t)$$

Note: one may model stochasticity in the continuous time case (stochastic differential equations) but it is out of scope in this tutorial.

When is a Task Successful?

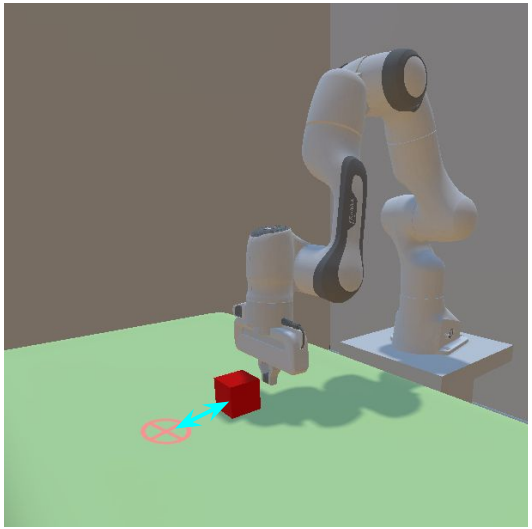
- How do we know if a task is complete?
- Idea: define success on states
  - Box xyz is close to target xyz
  - Box velocity is close to 0
  - Robot velocity is close to 0



When is a Task Successful?

- More generally, we can introduce a **reward** function $\mathcal{R}$ to measure how successful the current state/action is.

- For example
  - The environment gives a reward of 1 when the block is close to the target, 0 otherwise.



When is a Task Successful?

- More generally, we can introduce a **reward** function $\mathcal{R}$ to measure how successful the current state/action is.

- The 4 tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ is formally known as a **Markov Decision Process (MDP)**.

Markov Decision Process

Markov Decision Process is a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$

$\mathcal{S}$: State Space — all possible world states

$\mathcal{A}$: Action Space — all possible control signals

$\mathcal{T}$: Transition — environment dynamics

$\mathcal{R}$: Reward — how successful is the state/action

How to Solve Embodied AI Tasks

To solve an embodied AI task, the agent needs to know what action to take given the current state.

This is called a **policy**.

A policy π takes a **state** and outputs an **action** (can be stochastic).

$$a \sim \pi(\cdot|s)$$

A good policy should eventually complete the task (reach a successful state or accumulate a great amount of reward).

How to Solve Embodied AI Tasks

- Imitate an expert.
  - Imitation learning
  - Both $\mathcal{T}$ and $\mathcal{R}$ are not needed
- Learn to accumulate reward in an MDP
  - Reinforcement learning
  - Model-free: $\mathcal{T}$ is not modeled
  - Model-based: $\mathcal{T}$ is learned in the process
- Design rules based on mechanics
  - Classic robotics
  - $\mathcal{T}$ is modeled in advance (including learned models)

Outline

- Modeling and approaches for Embodied AI
  - World model
  - Learning-based methods to solve tasks
    - Imitation learning, reinforcement learning
  - Classic robotics

Optimal Policy

For a given policy $a \sim \pi(\cdot|s)$

We run the policy on the environment for H steps and collect rewards

$a_t \sim \pi(\cdot|s_t) \quad s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t) \quad r_{t+1} \sim \mathcal{R}(\cdot|s_t, a_t, s_{t+1})$

An optimal policy is the one that maximizes the expected cumulative reward

$$\mathbb{E}[\sum_{t=1}^{H} r_t]$$

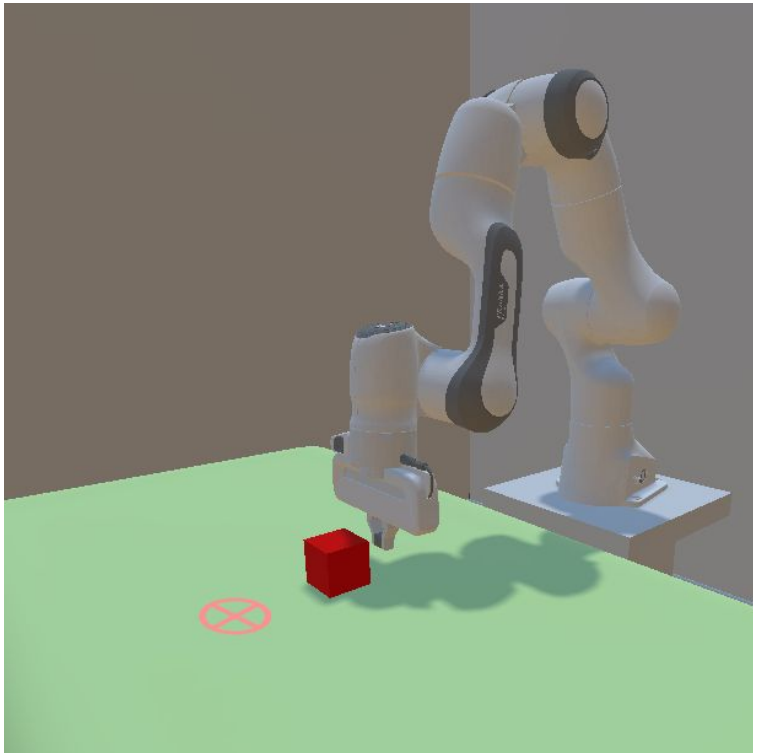
Note: in practice, a discount factor is often used to handle the case H=∞. It is not discussed here for simplicity.

Example of Optimal Policy

The environment gives a reward of 1 when the block is close to the target, 0 otherwise.

Let's also assume the system is terminated when the reward is 1.

An optimal policy is one that moves the block to the target eventually.



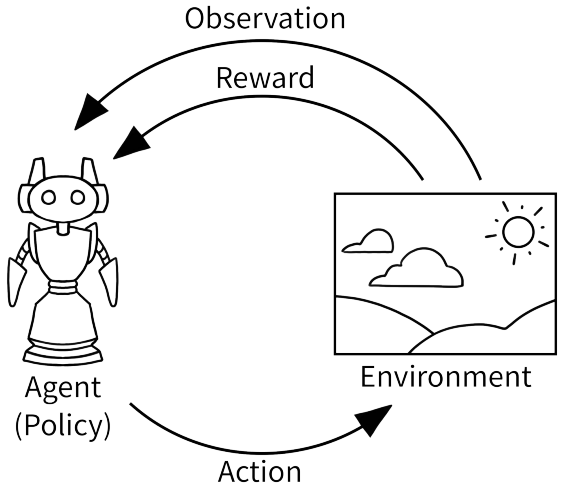
Partially-Observable MDP

In practice, the **state** is not always known.

Instead, we get some **observation**.

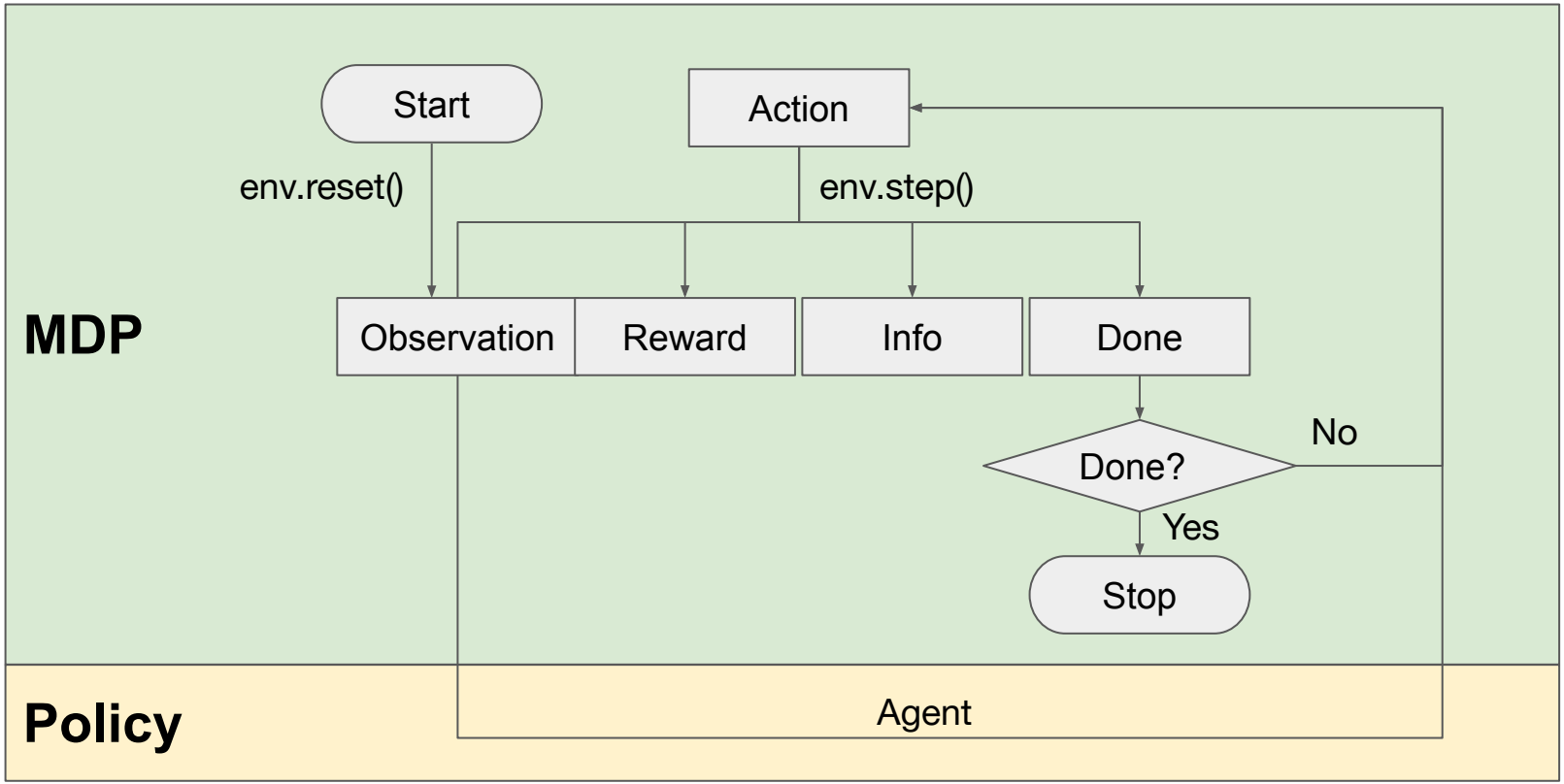
E.g., position of the cube vs an image of the cube

- Common observations
  - RGB-D image
  - Position & velocity of objects and robots
  - Task information (e.g. goal)
  - Other sensory readings



OpenAI Gym https://www.gymlibrary.ml/content/api/

Simulating MDP on a Computer



https://www.gymlibrary.ml/

How to get a Good Policy

Now how do we find a good policy?

- Idea 1: assume an expert (e.g., human) has solved the task; mimic this behavior — **imitation learning**.
- Idea 2: interact with the environment and try to improve the policy with reward —**reinforcement learning**.

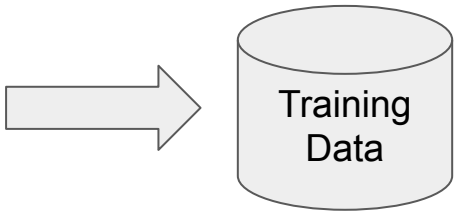
Imitation Learning

- Input: expert demonstrations $\{(s_t, a_t)\}$
- Output: policy $a \sim \pi_\theta(\cdot|s)$

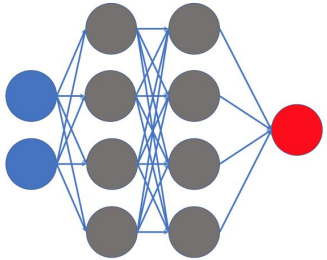
Expert

Observation

Action

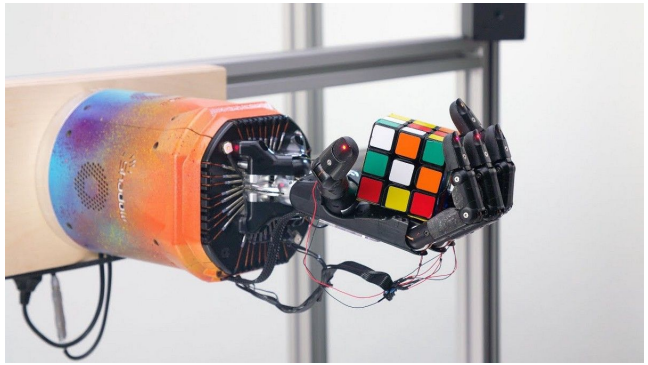


Supervised
Learning



Reinforcement Learning (RL)

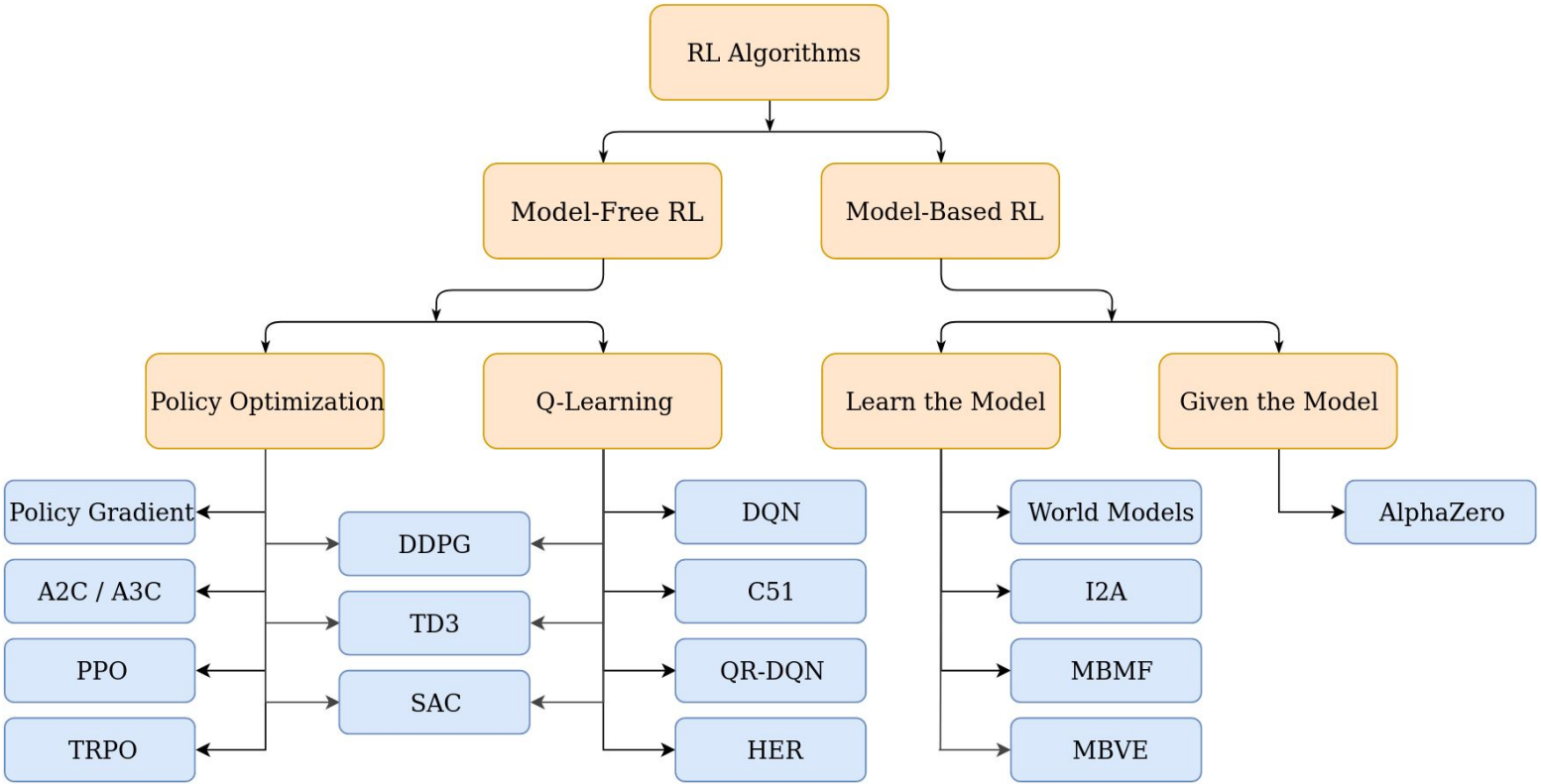
- What if we do not have expert data?
- Learn from interaction experience.
  a. Interact with environment (env.step) to collect experience.
  b. Use collected experience to improve the current policy.
  c. Repeat ab.



https://openai.com/blog/solving-rubiks-cube/

Recommended reading:
https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver

RL Taxonomy



OpenAI Spinup https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Combining RL and Expert Demonstrations

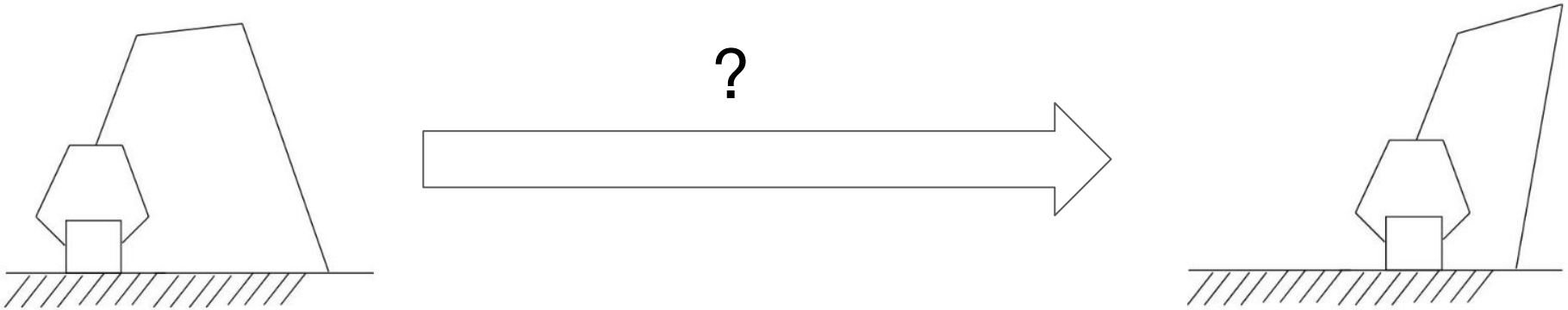
- “Learning from demonstrations”
  - Offline RL: train RL with given experience without further interactions
  - Augmenting online RL training with demonstrations
  - Dynamic movement primitives
  - Learning transition model from demonstrations

Outline

- Modeling and approaches for Embodied AI
  - World model
  - Learning-based methods to solve tasks
  - Classic robotics

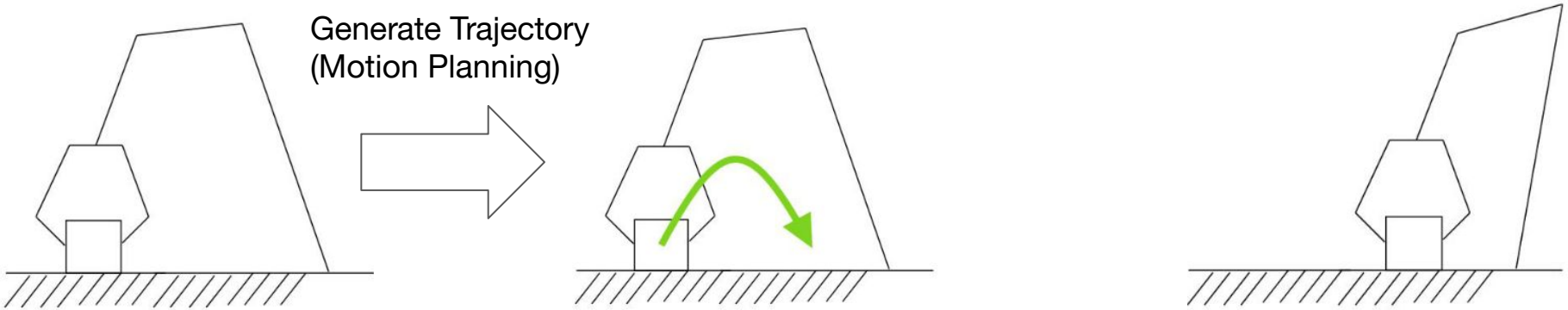
Plan and Control

A popular pipeline in classic robotics is planning and control.



Plan and Control

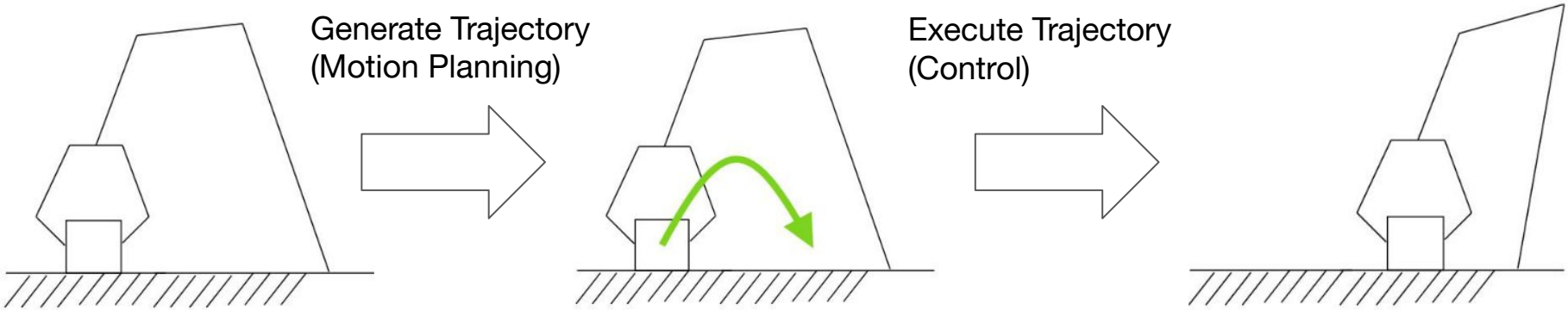
A popular pipeline in classic robotics is planning and control.



Motion planning generates a trajectory (position, velocity, and acceleration) of the robot.

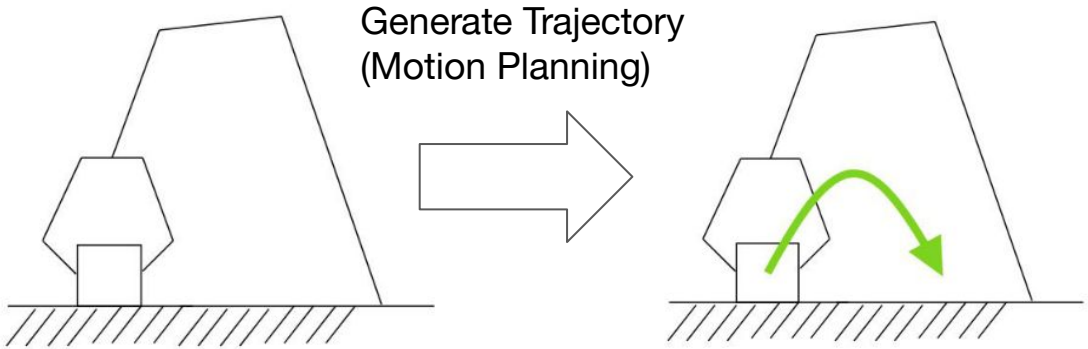
Plan and Control

A popular pipeline in classic robotics is planning and control.



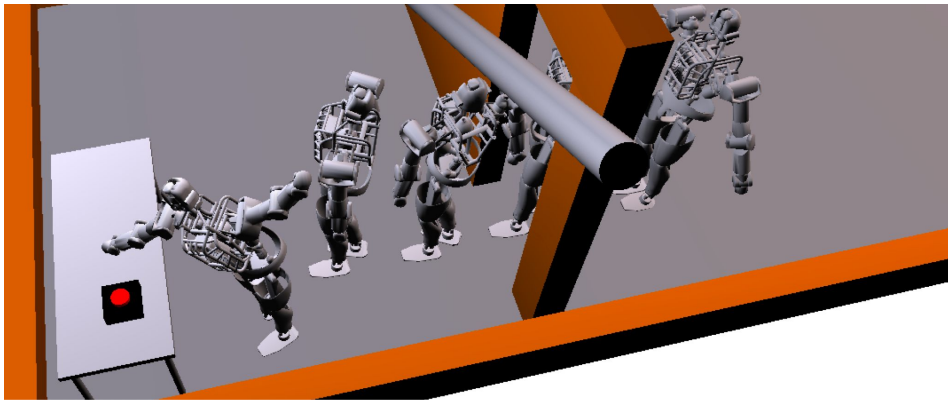
Motion planning generates a trajectory (position, velocity, and acceleration) of the robot.
Control executes the trajectory.

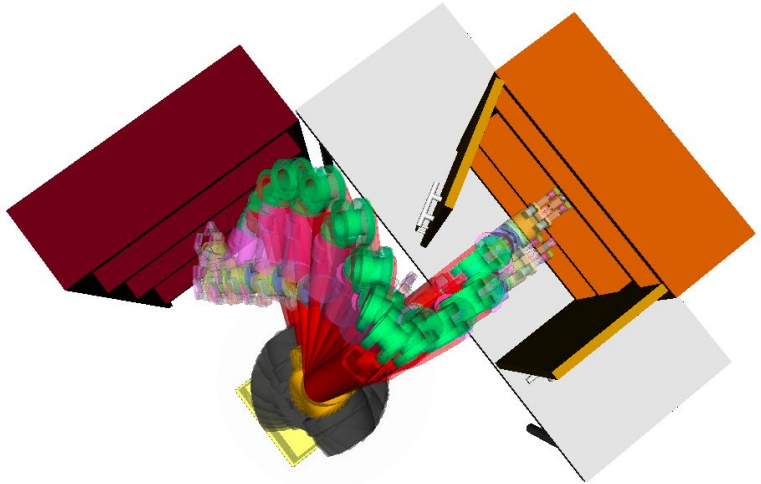
Motion Planning



Motion Planning

- Task: move a robot from one pose to another





Ratliff N, Zucker M, Bagnell J A, et al. CHOMP: Gradient optimization techniques for efficient motion planning, ICRA 2009
Schulman, John, et al. Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization, RSS 2013

Motion Planning

- Task: move a robot from one pose to another
- Assumptions
  - We know the start and goal pose
  - We can verify if a given pose is valid (usually means collision-free)
  - We can verify whether a pose is reachable from another pose using some simple control strategy

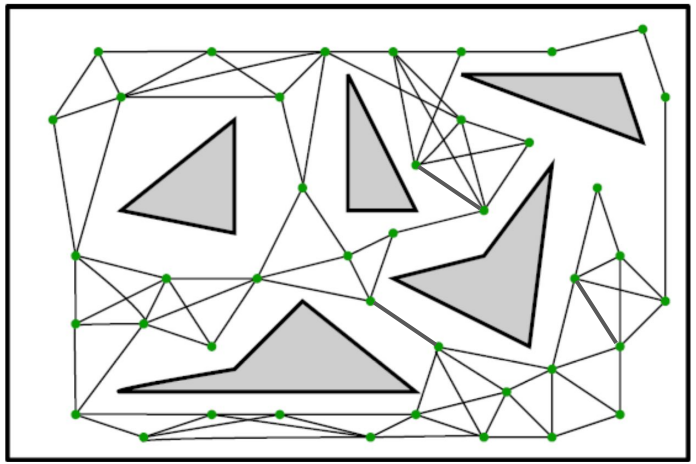
Motion Planning

- Task: move a robot from one pose to another
- Assumptions
  - We know the start and goal pose
  - We can verify if a random pose is valid (usually means collision-free)
  - We can verify whether a pose is reachable from another pose using some simple control strategy
- Algorithms
  - Rapidly-exploring random tree (RRT)
  - Probabilistic roadmap method (PRM)

Motion Planning Example: PRM

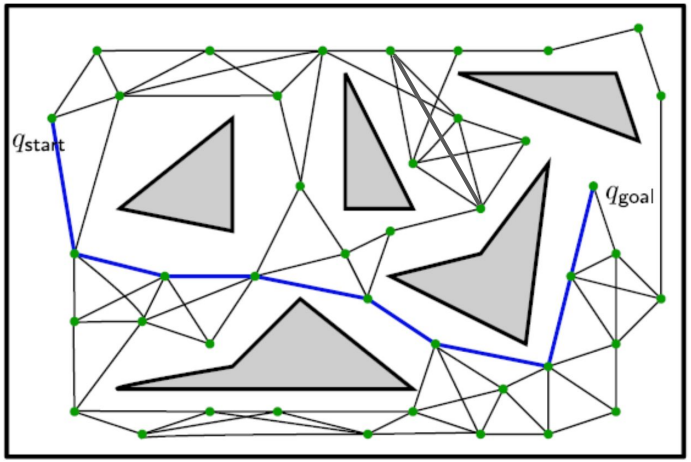
Motion Planning Example: PRM

- Phase 1: Map construction
  - Randomly sample collision-free configurations
  - Connect every sampled state to its neighbors
  - Connect the start and goal states to the graph



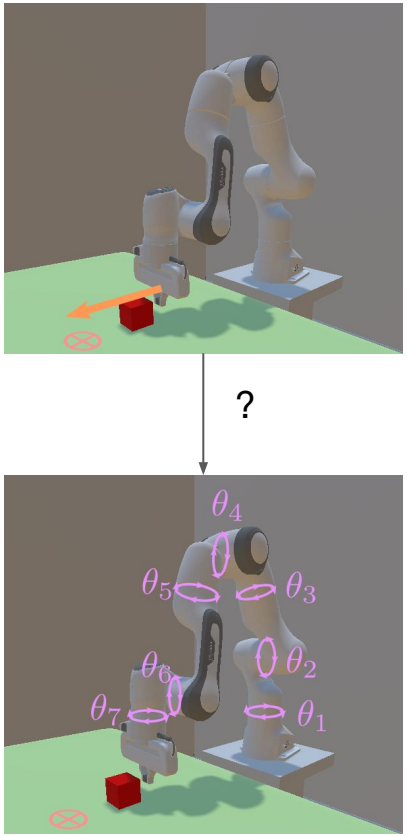
Motion Planning Example: PRM

- Phase 2: Query
  - Run path finding algorithms like Dijkstra



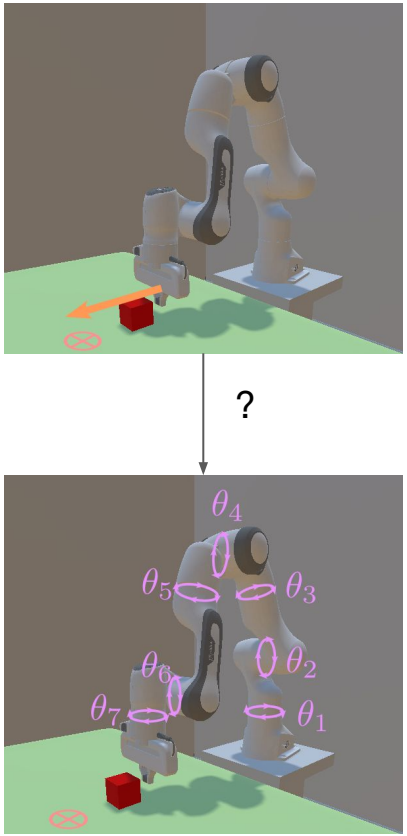
How to Find a Robot Pose For Grasping?

- Some tasks (such as grasping) require moving the gripper to a position.
- How do we find the robot pose of a given gripper pose?



How to Find a Robot Pose For Grasping?

- Some tasks (such as grasping) require moving the gripper to a position.
- How do we find the robot pose of a given gripper pose?
  - **Inverse Kinematics (IK)**



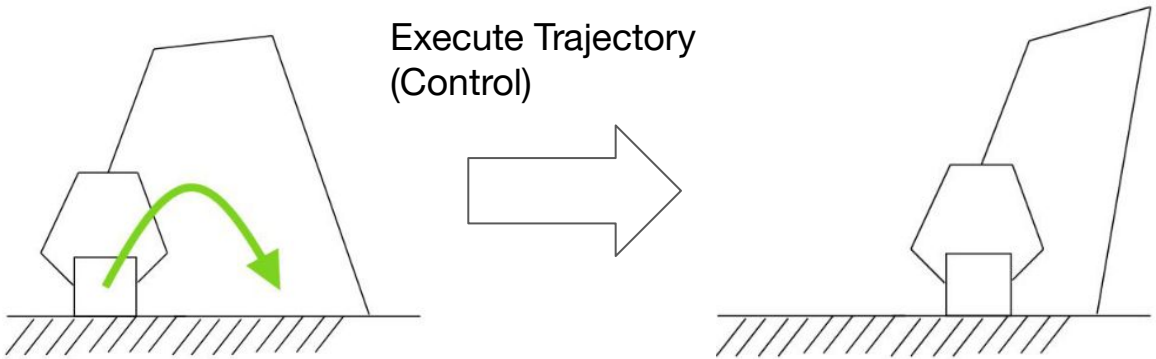

```
robot_model = robot.create_pinocchio_model()

joint_positions, success, error = robot_model.compute_inverse_kinematics(
    link_idx,
    target_pose,
    active_qmask = joint_mask  # joints with mask value 1 are allowed to move
    max_iterations = 100
)
```

Code in SAPIEN

Time Parameterization

- PRM/RRT gives a path with discrete joint positions q_d
- A time parameterization algorithm converts the path q_d to a joint **trajectory** $(q_d, \dot{q}_d, \ddot{q}_d)$ with time.

Control



Control

- Robotic control executes a given trajectory $(q_d, \dot{q}_d, \ddot{q}_d)$ by controlling the joint torques τ

Control

- Robotic control executes a given trajectory $(q_d, \dot{q}_d, \ddot{q}_d)$ by controlling the joint torques τ
  - q represents the joint positions of a robot
- Similar to $F = ma$, the dynamic model of a robot is known.
  - Forward dynamics: $\ddot{q} = \mathrm{FD}(\tau; q, \dot{q})$
  - Inverse dynamics: $\tau = \mathrm{ID}(\ddot{q}; q, \dot{q}) = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q)$

Inertia matrix Coriolis matrix Gravity & other forces

Control

- What we have
  - Trajectory $(q_d, \dot{q}_d, \ddot{q}_d)$
  - Inverse dynamics: $\tau = \mathrm{ID}(\ddot{q}; q, \dot{q})$
- Ideally, using τ computed from $\ddot{q}_d$ gives a perfect trajectory.
- However, the real world is not perfect. What if there is some error?

$$e = q - q_d$$

Use Control in MDP Modeling

- When an RL work says: *we use “velocity control” or “position control” as action*. What does that mean?

Use Control in MDP Modeling

- The action in an MDP can be “target joint velocity” or “target joint position” for a controller.

Use Control in MDP Modeling

- The action in an MDP can be “target joint velocity” or “target joint position” for a controller.
- A controller (such as PD) is used to convert this velocity or position signal to joint torques, which are then used to drive the robot.

Use Control in MDP Modeling

- The action in an MDP can be “target joint velocity” or “target joint position” for a controller.
- A controller (such as PD) is used to convert this velocity or position signal to joint torques, which are then used to drive the robot.
- Joint velocity/position may be a better choice for MDP action (than torque) due to learnability and sim-to-real transferability.

More About Control

- Control focuses on stability and robustness
- There is a huge literature
  - Optimal control
  - Feedforward/feedback control (including PD)
  - Robust control
  - Self-organized control
  - Stochastic control
  - …
- Optimal control has a strong connection with RL

Summary

- Embodied AI Approaches
  - Learning-based methods
    - Imitation learning
    - Reinforcement learning
    - …
  - Classic robotics
    - Planning
    - Control
    - …
- In-depth discussion of these topics

How do we Study Embodied AI Algorithms?

- An environment is required to develop approaches
- Real robot?
  - High costs
  - Safety concerns
- Simulation environment?
  - Physical simulation
  - Camera simulation
  - Assets loading
  - Sim-to-real gaps