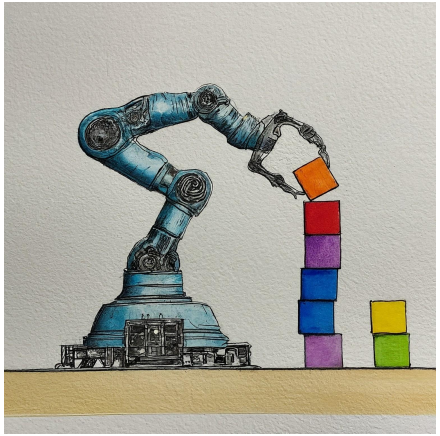
Lab 1 - Introduction to Task Planning

AIR 24-25



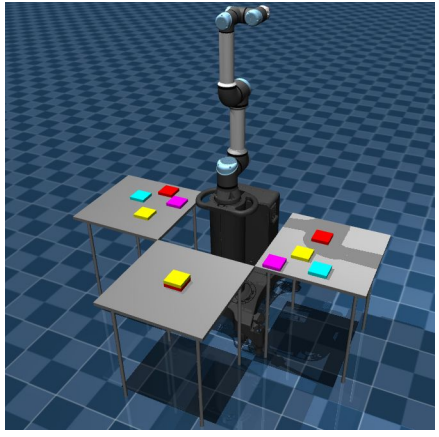
Ido Jacobi & Shiran Peeran & Sarah Keren

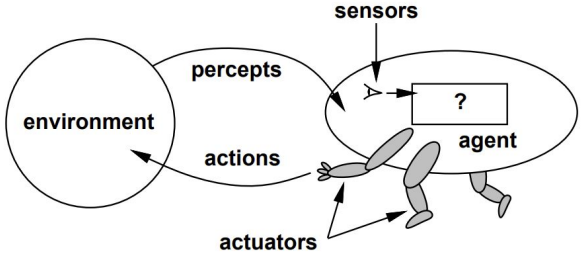
Collaborative AI and Robotics

TECHNION | The Henry and Marilyn Taub Faculty of Computer Science

Course and Lab Objectives

- The focus of this course: robotic agents that need to perform complex tasks that require
  - high-level reasoning and planning
  - understanding of how to move and operate in the physical environment
- We will start by exploring task planning and explore the limitations of task planning for robotic agents.
- The labs are meant to complement the theoretical material learned in class with in-depth exploration of some of the ideas we learned and with some hands-on experience on getting robots to do what we want them to do





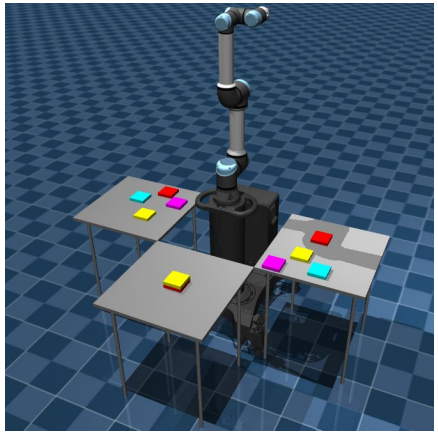
Technical Details

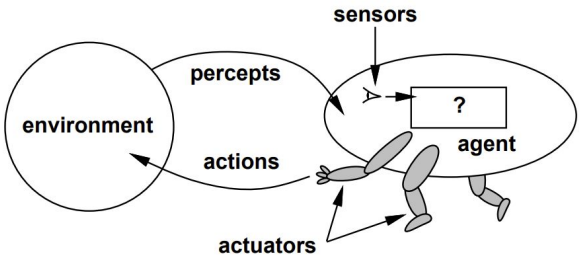
- The tutorials are provided in the form of Jupyter notebooks, suitable for running online via Google Colab.
- Basic knowledge of Python and PDDL is required.
- At the beginning of each notebook, there is a link to run it on Colab.
- If you would like to run the notebooks locally on your machine, you can download them, but some installations may be required (e.g., numpy).
- Each lab will include a presentation (like this one) and a complementary Google notebook with which you can practice the learned ideas.

Our repo: https://github.com/CLAIR-LAB-TECHNION/AIR

Lab 1 - Objectives

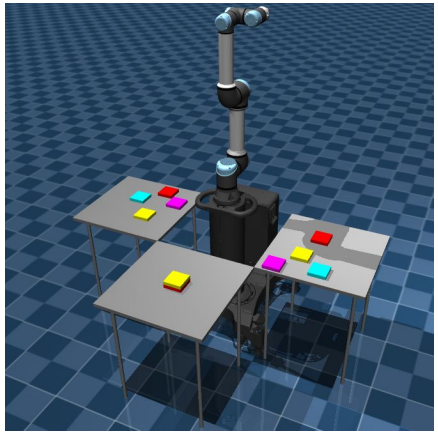
- Getting to know the complexities of task planning for robotic settings
- Understanding PDDL and its limitations of in representing complex settings
- Understanding the need to integrate task and motion planning

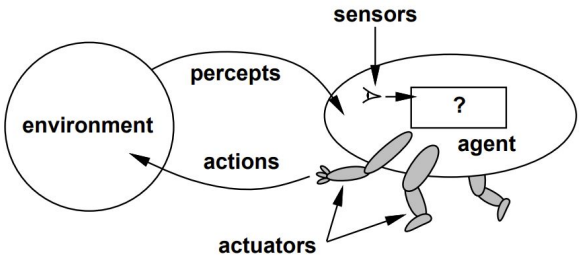




Lab 1 - Tasks

1. Getting familiar with PDDL
2. Solving PDDL problems using Best First Search and formulating a heuristic fundtion
3. Understanding the expressive limitations of PDDL





Task Planning with PDDL

Task Planning - Classical Formulation

$$\langle S, s_0, S_G, A, f, c \rangle$$

- S is a finite and discrete set of states, — State space induced by feature set V
- $s_0 \in S$ is the *known initial state*,
- $S_G \subseteq S$ is the non-empty set of goal states,
- $A(s) \subseteq A$ represents the set of actions in A that are applicable in each state $s \in S$,
- $f(a, s)$ is the *deterministic transition function* where $s' = f(a, s)$ is the state that follows s after doing action $a \in A(s)$, and
- $c(a, s)$ is a *positive cost* for doing action a in the state s.

A solution or *plan* in this model is a sequence of applicable actions $a_0, \ldots, a_n$ that generates a state sequence $s_0, s_1, \ldots, s_{n+1}$ where s_{n+1} is a goal state. More precisely, the action a_i is applicable

Geffner, Hector, and Blai Bonet. *A concise introduction to models and methods for automated planning*. Springer Nature, 2022.

A blocks world example

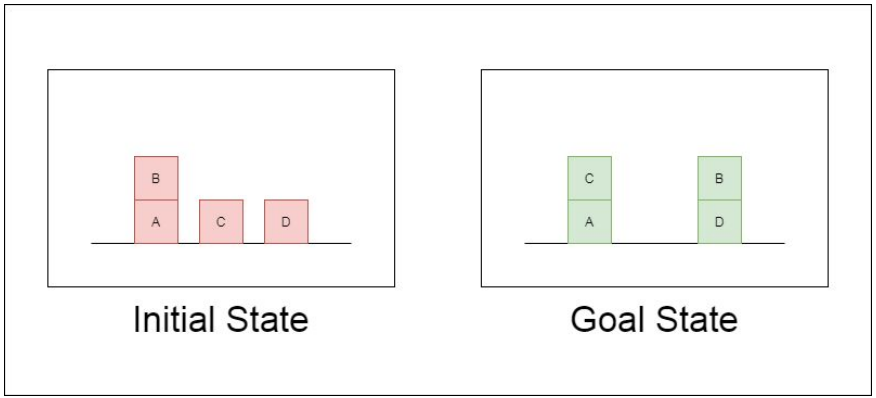
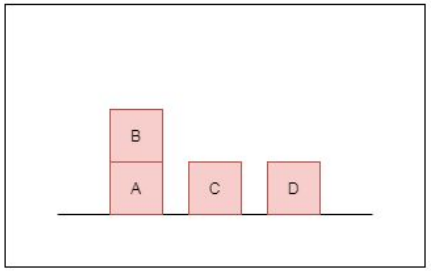
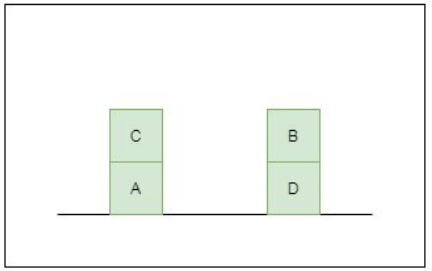


Image from https://apoorvdixit619.medium.com/goal-stack-planning-for-blocks-world-problem-41779d090f29

Planning





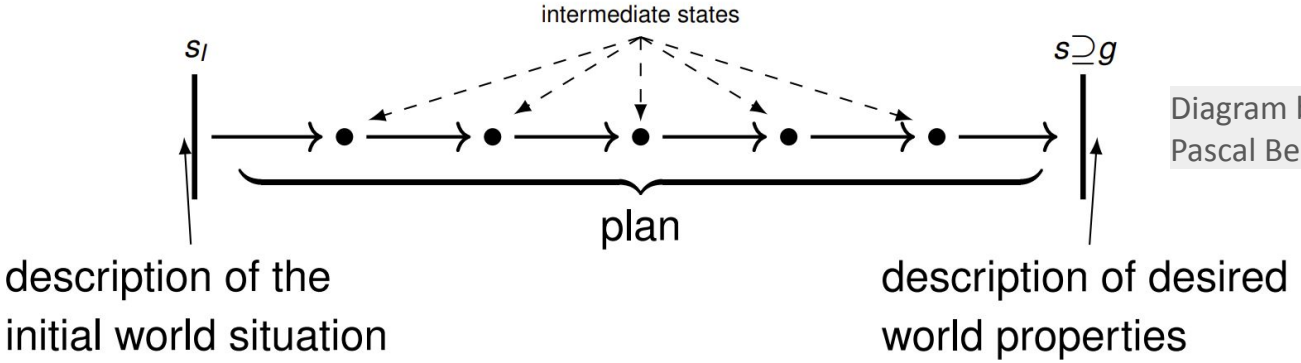
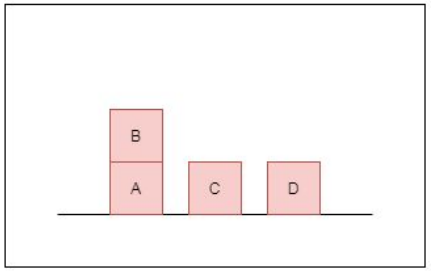


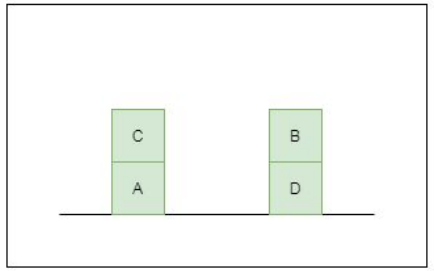
Diagram by Pascal Bercher

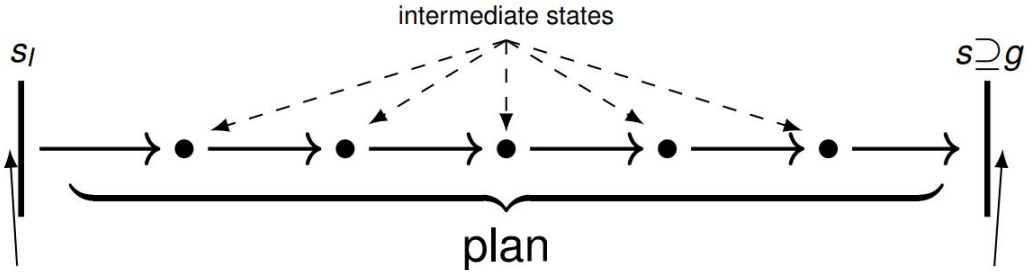
Classical planning: discrete, deterministic, fully observable and single agent.

Problem description induces a state transition system, with nodes as states and actions as transitions.

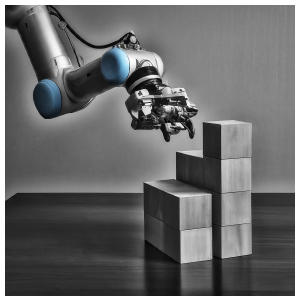
Becomes very complex very quickly: state space is 2^|V|, where V is the number of state features



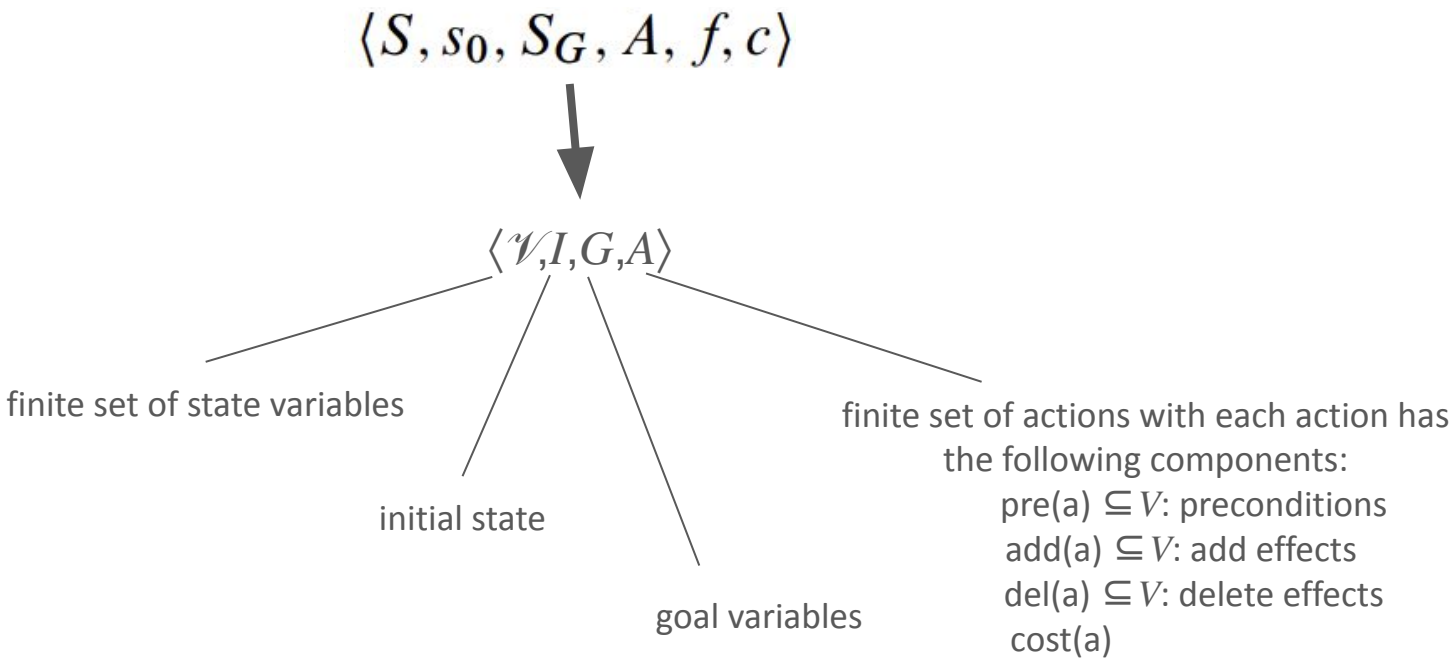




Pickup B -> Stack B on D -> Pickup C -> Stack C on A



STRIPS and PDDL



Geffner, Hector, and Blai Bonet. *A concise introduction to models and methods for automated planning*. Springer Nature, 2022.

STRIPS and PDDL

- STRIPS (Stanford Research Institute Problem Solver) and PDDL (Planning Domain Definition Language) are both used to encoding planning problems.
- PDDL is more expressive:
  - separate domain and problem definitions.
  - domain includes types, predicates, functions, and actions available.
  - The problem description allows describing the objects in the world, the initial state, and the goal.

STRIPS

```
Operator: MOVE(x, y, z)
Preconditions: ON(x,y), CLEAR(x), CLEAR(z)
Delete list: ON(x,y), CLEAR(z)
Add list: ON(x,z), CLEAR(y)
```

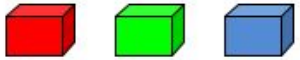
PDDL

```
(:action move
  :parameters (?x ?y ?z - block)
  :precondition (and (on ?x ?y)
                     (clear ?x)
                     (clear ?z))
  :effect (and (not (on ?x ?y))
               (not (clear ?z))
               (on ?x ?z)
               (clear ?y))
  )
```

Formulation of Blocks World

Action/model:

Objects (things):



Predicates (true or false statements about the world, "facts"):

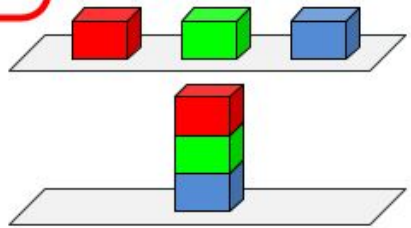
(On A, B)

(Clear A)

Actions (how to change the predicates):

Initial: (Clear) (Clear) (Clear)

Goal: (On ,) (On ,)



slide by Erez Karpas

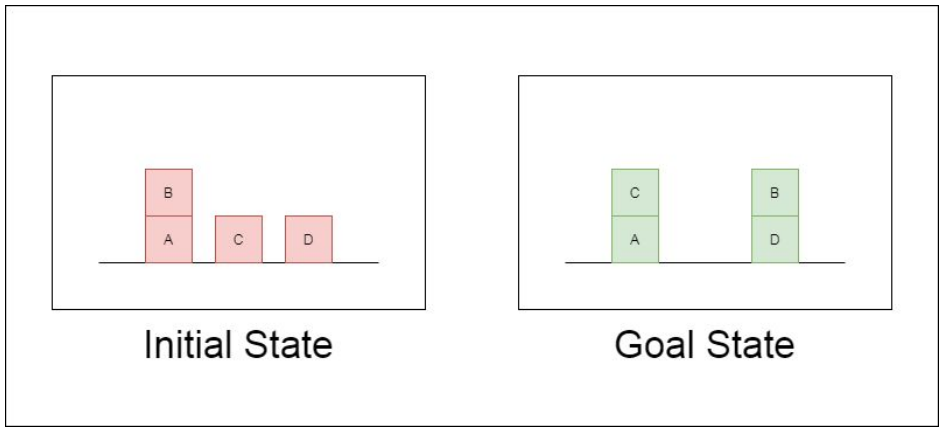
PDDL Formulation of Blocks World

```
(:types block)
(:predicates (on ?b1 ?b2 - block)
             (clear ?b - block)
             (handempty)
             (holding ?b - block))

(:action stack
       :parameters (?b1 ?b2 - block)
       :precondition (and (holding ?b1) (clear ?b2))
       :effect
       (and (not (holding ?b1))
       (not (clear ?b2))
       (clear ?b1)
       (handempty)
       (on ?b1 ?b2)))
```

```
(:init
(handempty)(ontable D)(clear D)(on B A) ...)

(:goal (and
(on B D)(ontable D)
(on C A)(ontable A))))
```



Task 1

Writing a PDDL definition for a blocks world problem

Task Planning with Best First Search

Task Planning: Solution approaches

Common algorithms/techniques:

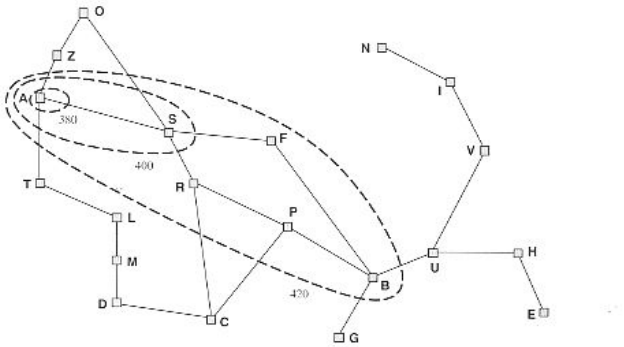
- Search – forward, regression, bidirectional, symbolic
- Compilation – to SAT, to CP
- Partial Order Planning
- Dynamic programming: Bellman Backup – i.e. for problems with uncertainty and rewards, like MDPs and POMDPs.

What are some general search algorithms?

Best First Search

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```



Code above from Norvig, P. Russel, and S. Artificial Intelligence. "A modern approach." *Prentice Hall Upper*

Pearl, Judea. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

Task Planning - Extensions

Accounting for stochastic action outcome using a state transition probability function

$$\mathcal{P}^a_{s,s'} = \mathcal{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

Accounting for objectives using a reward function

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

Accounting for reward deprecation using a discount factor

$$\gamma \in [0, 1]$$

Accounting for partial observability using observations Ω and observation function

$$\mathcal{O}^o_{s,a} = \mathcal{P}[O_{t+1} = o | S_t = s, A_t = a]$$

Accounting for action duration, multiple agents, varying costs, etc.

Sequential decision making: what should the agent do to maximize total rewards?

A*

- A* is a special case of Best First Search that uses a heuristic to guide the search from an initial state to a goal state.
- Formally, the priority according to which A* extracts nodes from the frontier is: f(n)=g(n)+h(n) where:
  - n is the node that is considered,
  - g(n) is the cost of the path from the start node to n
  - h(n) is the heuristic estimation of the cost from n to a goal state.

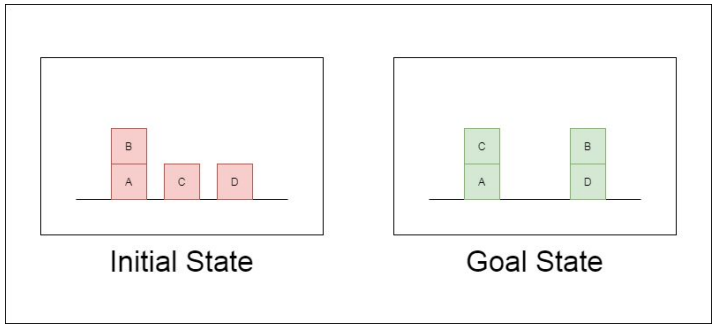
Heuristic Functions

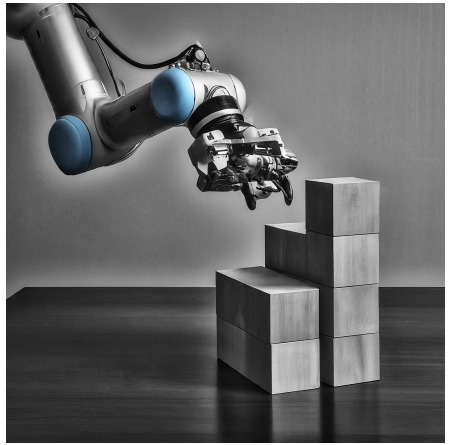
- Heuristic functions are an important element of any heuristic search, and A* in particular.
- Effective heuristics possess certain characteristics that enable the algorithm to efficiently find solutions while minimizing the search effort
- Some key characteristics of heuristic functions:
  - Safeness
  - Goal Awareness
  - Admissibility
  - Efficiency

Task 2

Implementing a heuristic method

Are we done ?

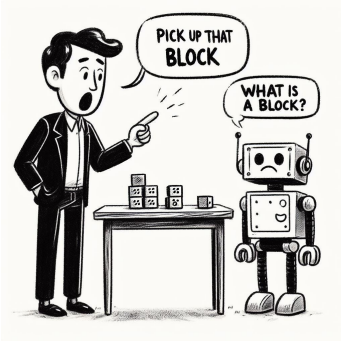




How does it know what a block is ?

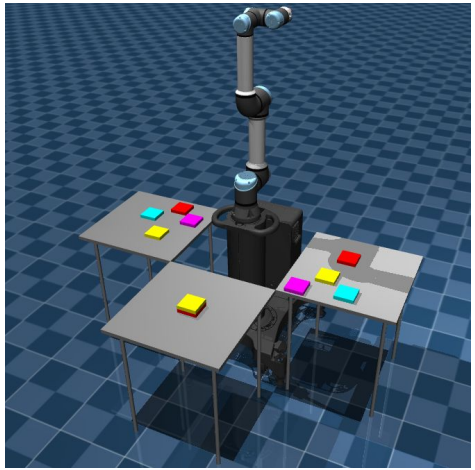
How does it know how to stack, pick up and put down?

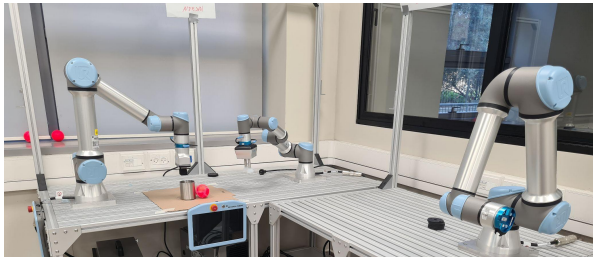
How does it know what’s the best way to pick up?



BW in the real(istic) world

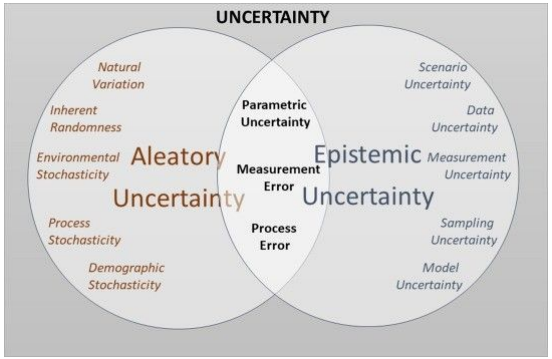
What have we failed to capture with our PDDL formulations ?





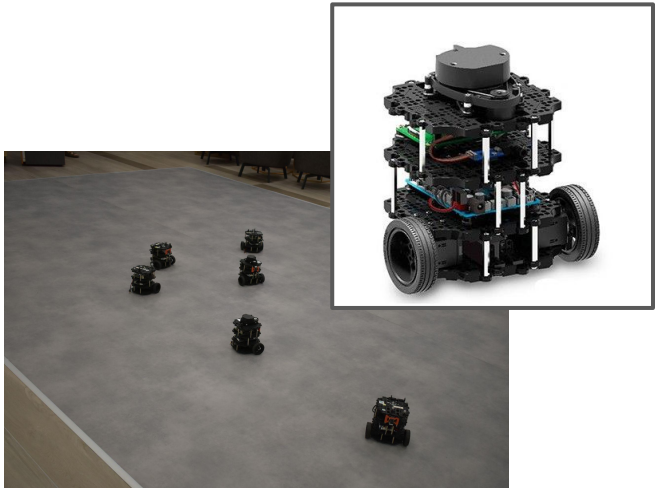
Task Planning vs. the Real World

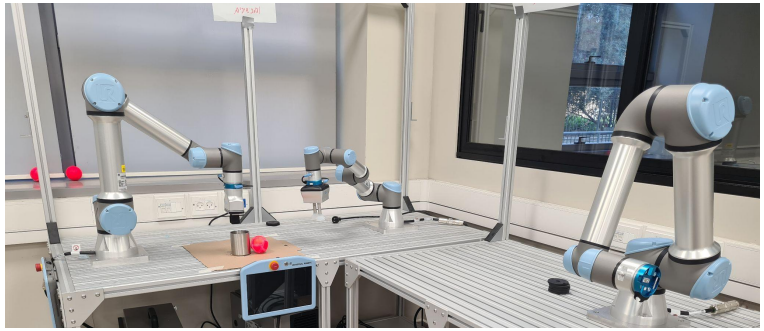
- **Discrete:** plans are finite sequences of actions - while actual actions are continuous
- **Fully modeled actions** (deterministic in classical planning): while actual actions can have unexpected effects that are hard/impossible to model
- **Full knowledge** of world state or of belief (distribution over possible world states) while knowing the actual state /distribution relies on complex sensing and reasoning



https://www.stat.berkeley.edu/users/aldous/157/Papers/Fox_Ulkumen.pdf

Many Types of Robots





Robot Elements

Sensors

Controllers

Actuators

Image from: https://ieeexplore.ieee.org/abstract/document/6386486/

Robotic Agents

state estimation

Maintain a **belief** $\beta \in \boldsymbol{\beta}$ as a probability distribution over world states **S**

$$\beta: S \rightarrow [0,1]$$

decision making

Find a **policy**, mapping **belief** β **and objective into** actions (probabilities)

$$\pi: \boldsymbol{\beta} \square A \rightarrow [0,1]$$

motion control

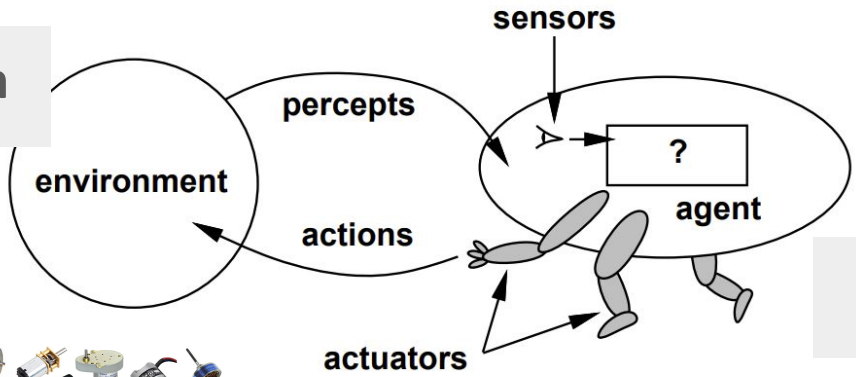
Translate actions into low-level commands (and monitor their execution)

$$u = k_p \cdot e + k_i \cdot \int e dt + k_d \cdot \frac{d}{dt} e$$

https://eng-resources.charlotte.edu/unccengkit/coding/pid-control/

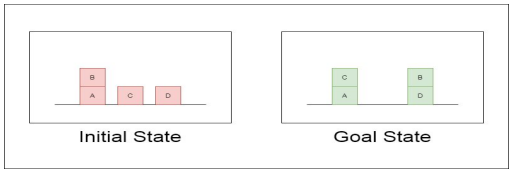
Robotic Agents

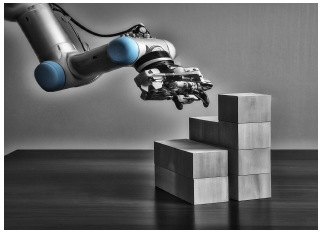
state estimation



decision making

motion control





decision making

```
(:action stack
        :parameters (?b1 ?b2 - block)
        :precondition (and (holding ?b1) (clear ?b2))
        :effect
        (and (not (holding ?b1))
        (not (clear ?b2))
        (clear ?b1)
        (handempty)
        (on ?b1 ?b2)))
```

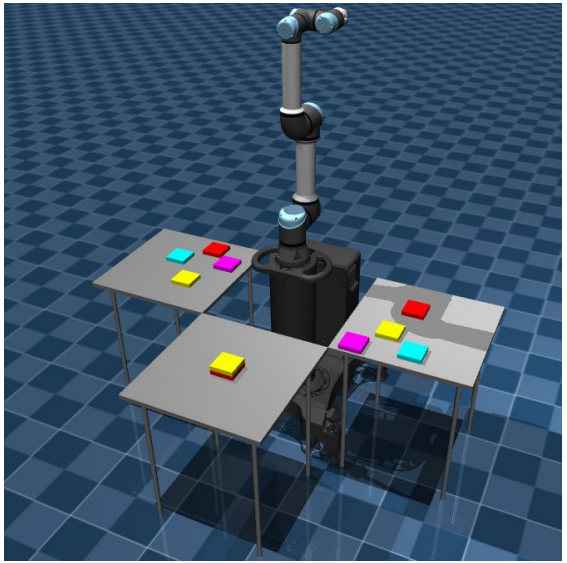
state estimation

motion control

Task planning is not enough

What next ?

Things get messier: N-table Blocks World



Accounting for varying costs, collision avoidance, motion feasibility, multiple (infinite) positions from which an action can be performed, multiple (infinite) ways to reach a position

MuJoCo

Advanced physics simulation