ECE7121 Learning-based control – 2025 Fall

Environment and Simulation

INHA UNIVERSITY

Overview

- IL -> RL / control problem
- Multi-arm bandit
- MDP / POMDP
- Continuity
- Dynamic system model
- Simulation
- Simulators

Limitations of imitation learning

- We need experts (e.g., human)
  - Could be expensive
- Can not go beyond the expert level
  - We want super intelligence
- Expert demo could be multi-modal and not optimal
  - Learning might go wrong

RL Success

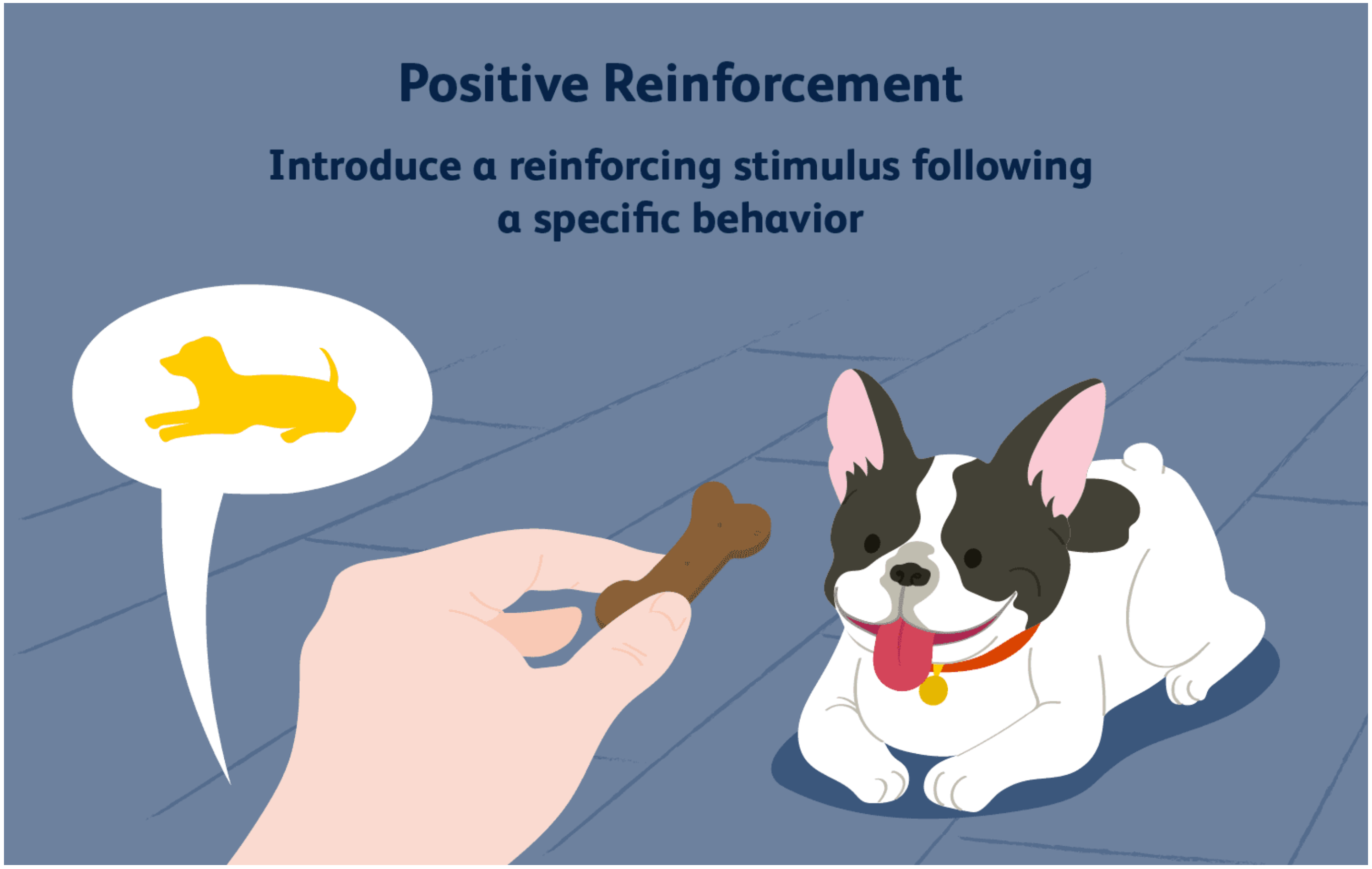
> Game

AlphaGo: Go World champion

AlphaStar: Grandmaster (99.8%)

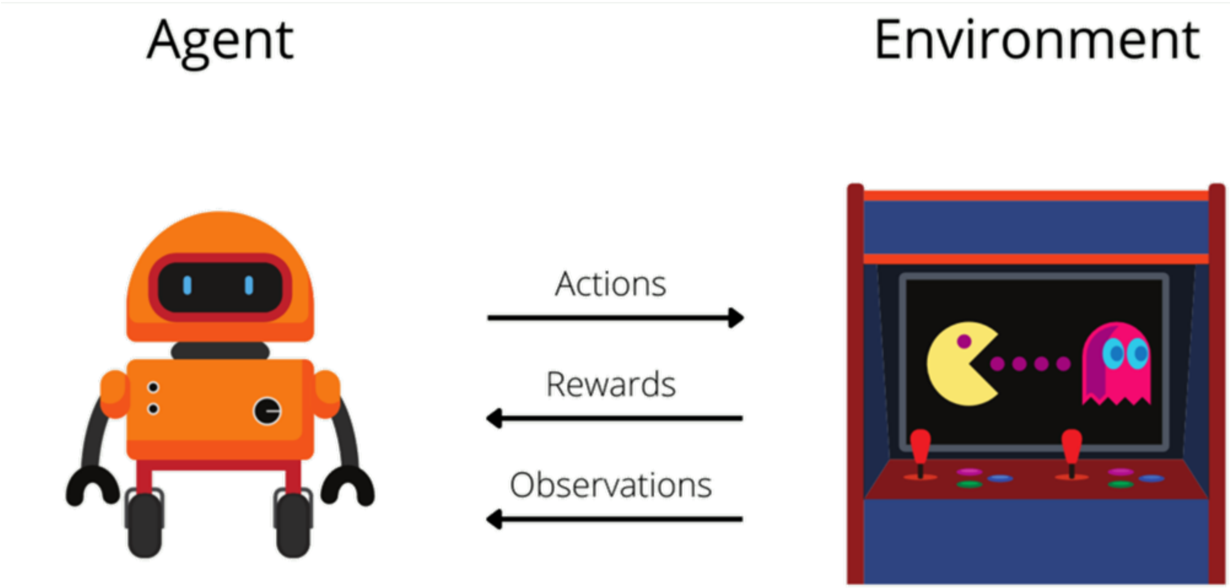
RL = trial-and-error learning

- Reinforcement in educational psychology
  - Big advantage: AI can learn autonomously



RL = trial-and-error learning

> Learning a policy that maximizes rewards by interacting with the environment



- Each action results in an immediate reward.
- We want to choose actions that maximize our immediate reward in expectation.

Multi-armed bandits

> bandit = 

One-armed bandit: slot machine

Multi-armed bandit: multiple slot machines

Multi-armed bandits

- The state does not change!
  - We don’t move. We use the same slot machines.
- We have N slot machines and select one to pull.
  - We have N possible actions.
- We will get an immediate reward from the pulled slot machine.
  - Each slot machine gives a random reward.
  - The reward probability of each machine is fixed but unknown.

- Objective: maximize cumulative rewards

How to earn money?

- Strategy 1: pull each once, exploit the best




Reward: 100₩ 0₩ 200₩ 30₩

- Next, we only pull the third machine

How to earn money?

> Strategy 2: pull each 4 times, exploit the best

Reward:

100₩	0₩	200₩	30₩
100₩	300₩	0₩	500₩
100₩	0₩	0₩	20₩
100₩	400₩	0₩	40₩
400₩	700₩	200₩	590₩

> Next, we only pull the second machine

Achieving a balance

- Pulling the same machine several times
  = learning reward probability distribution and its mean
  = **exploration (collecting information)**

- Pulling the best machine
  = exploiting the best to earn money (given current information)
  = **exploitation (collecting reward)**

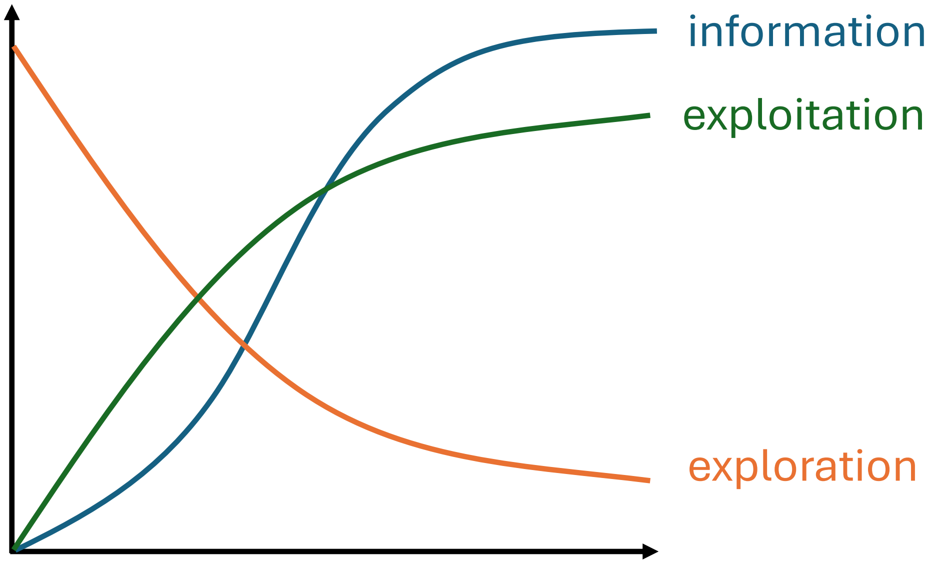
- Action-value for action a is its mean reward
  - $\mathrm{Q}_t^*(a) = \mathbb{E}[R_{t+1}|A_t = a]$, action-value estimate: $Q_t(a) \approx Q_t^*(a)$
  - $A_t^* = \arg max\, Q_t(a)$
  - If $A_t = A_t^*$: exploiting
  - If $A_t \neq A_t^*$: exploring
- We need to do both

Exploration dilemma

- Exploration vs Exploitation dilemma
  - The best long-term strategy may involve short-term sacrifices
  - This is not a problem unique to RL; it is a fundamental issue in the decision making of any intelligent agent.
- Restaurant selection
  - exploitation: go to your favorite restaurant
  - exploration: try a new restaurant
- Studying
  - exploitation: solve example problems
  - exploration: read additional materials

Exploration dilemma

> $\epsilon - greedy$ algorithm



Markov decision process

- Multi-armed bandit
  - $\tau: (A_t, R_{t+1}, A_{t+1}, R_{t+2}, A_{t+2}, R_{t+3}, \dots)$

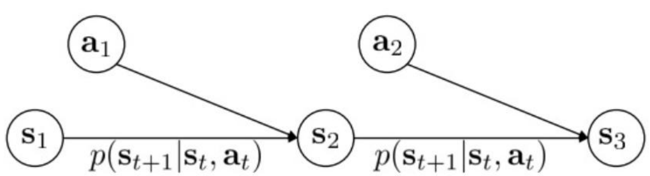
- Markov decision process
  - $\tau: (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, A_{t+2}, R_{t+3}, \dots)$

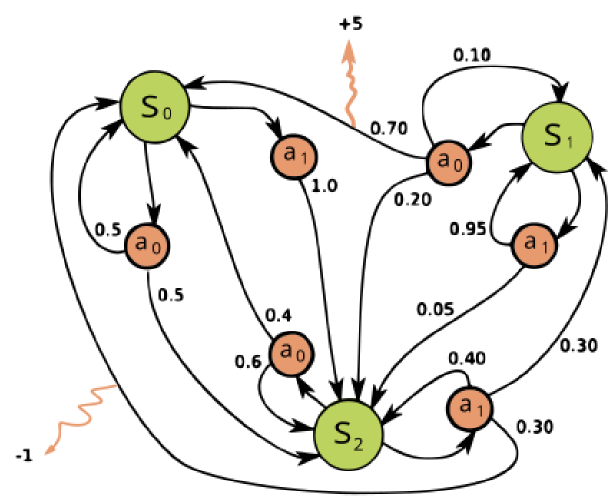
- Markov property
  - $p[R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_0, S_1, A_1, R_1, \dots S_t, A_t]$
    $= p[R_{t+1} = r, S_{t+1} = s' | S_t, A_t]$
  - Only the present determines the future; we can ignore the history.

Markov decision process

> Finite Markov decision process is a tuple (S, A, T, r, γ)
  - S is a finite set of states $s \in S$
  - A is a finite set of actions $a \in A$
  - T is one step transition/dynamics function $p(s'|s,a)$
  - $r(s,a,s')$ is a reward function
  - γ is a discount factor $(0 \leq \gamma \leq 1)$

> $\pi(a|s)$: a policy is a distribution over actions given states



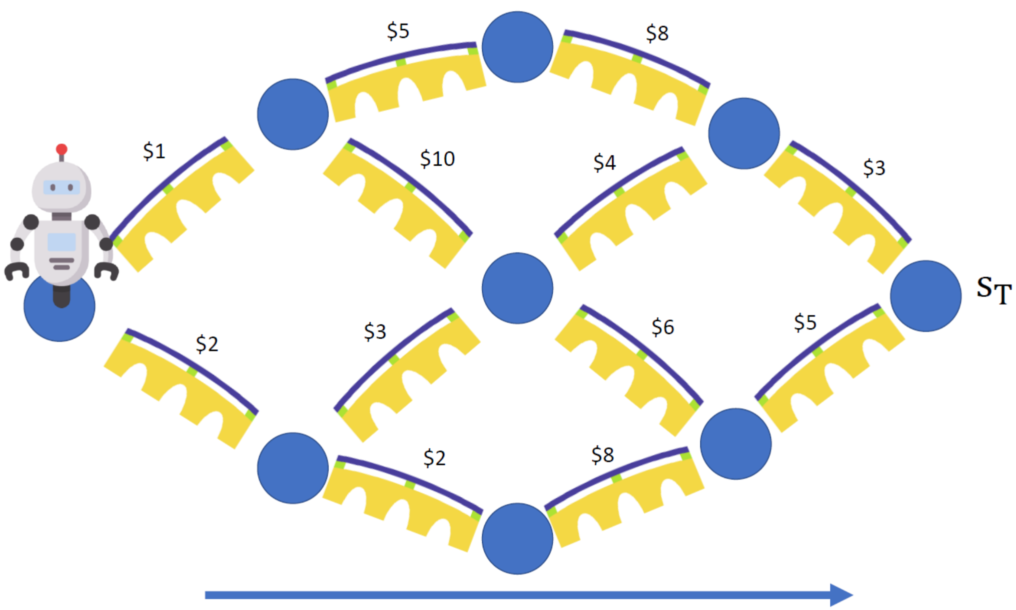


Markov decision process

> Maximize your sum of future rewards (cumulative reward)
  - $G = R(\tau) = R_1 + R_2 + R_3 + R_4 \ldots$

  - Future rewards may less important
  - $G = R(\tau) = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots$

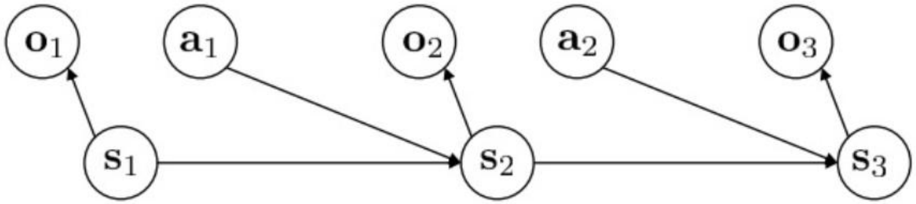
> The robot collects toll on every bridge



POMDP

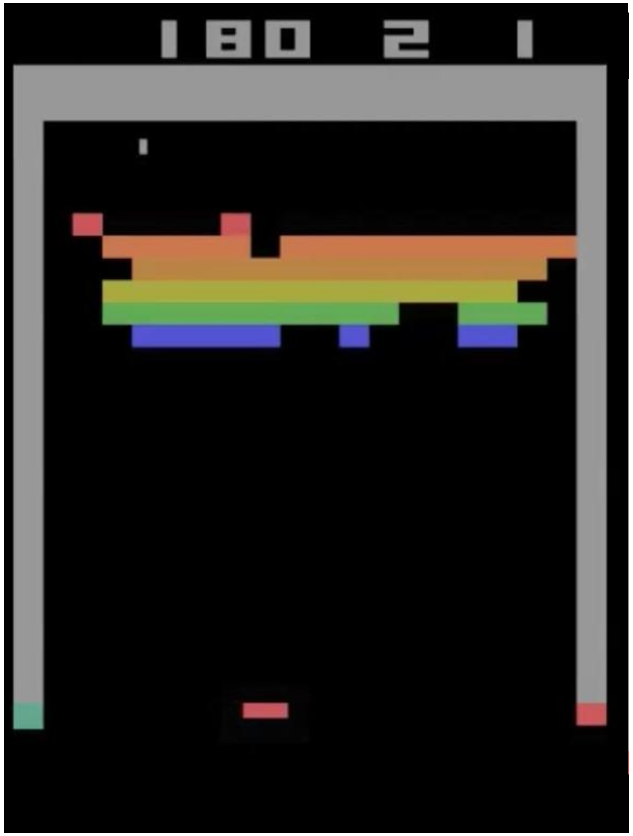
- Partially Observable MDP
  - Finite POMDP is a tuple $(S, A, O, T, h, r, \gamma)$
  - h is the observation model $p(o|s)$

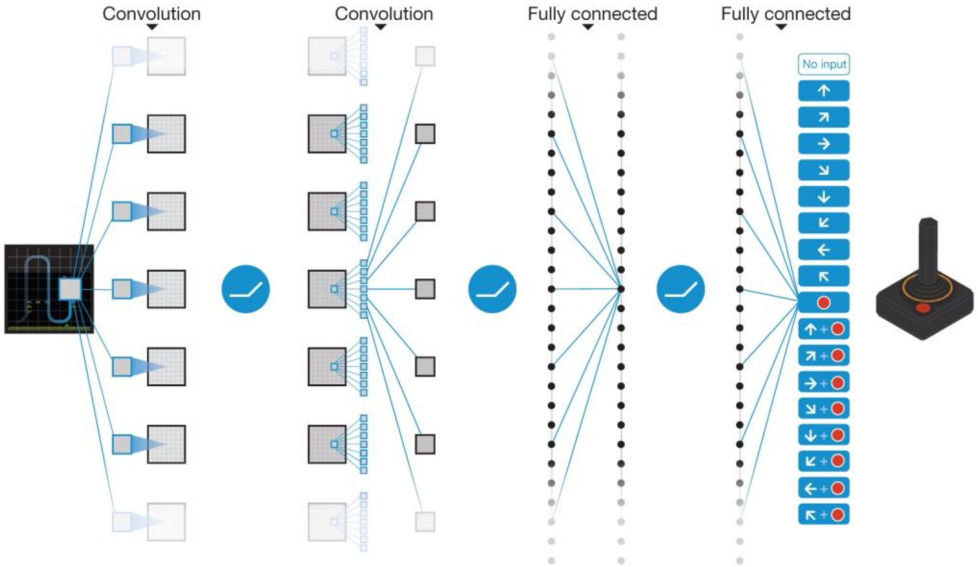
- Learn a policy $\pi(a|o)$



POMDP

> Playing Atari with DQN





RL in robot control

> Now, we want to control diverse robotic platforms
  - Can we formulate a control problem mathematically?

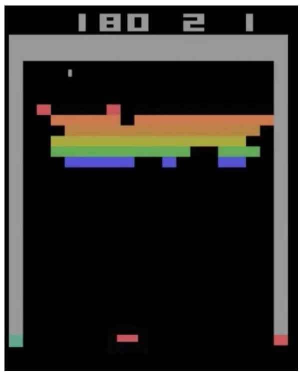
> Optimal control = optimization of dynamic systems

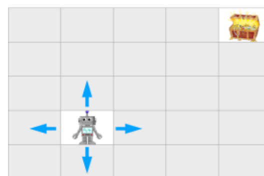
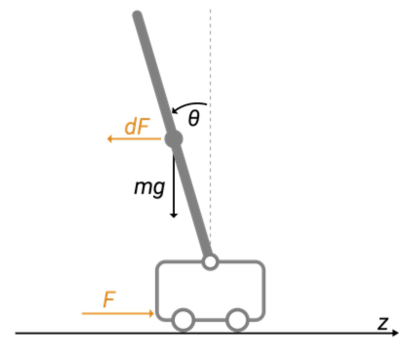
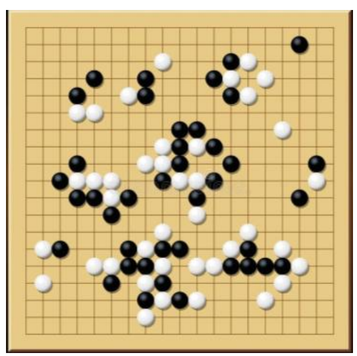
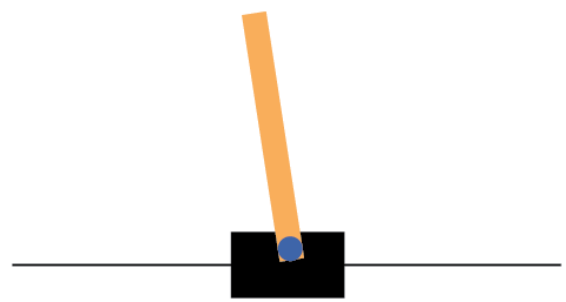
Optimal control

- Optimal control problem (OCP) have three ingredients:
  - dynamic system model
  - constraints
  - objective function

- Typical control methods for complex robotic systems:
  - 1. Model the system
  - 2. Define a desired behavior (cumulative reward)
  - 3. Find the trajectory that maximizes the goal
  - 4. Find the inverse model
  - 5. Find the actuation to follow

Continuity

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">State space</td></tr>
<tr><td>Discrete</td><td>Continuous</td></tr>
<tr><td rowspan="2">Time space</td><td>Continuous</td><td>

</td><td>
</td></tr>
<tr><td>Discrete</td><td></td><td></td></tr>
</table>

Dynamic system models

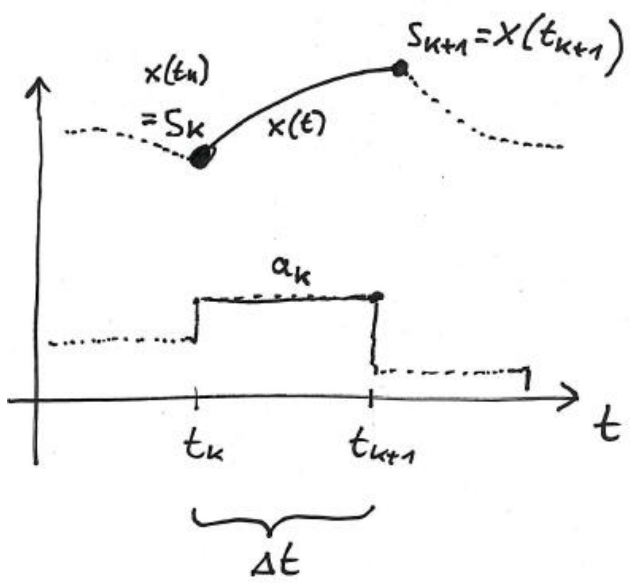
- Discrete time
  - $s_{k+1} = f(s_k, a_k)$

- Continuous time
  - Ordinary differential equation (ODE)
  - $\dot{s} = \frac{ds}{dt}(t) = f_c(s(t), a(t))$

- Stochastic
  - Adding random disturbance ϵ

Control problem to MDP

> Transform a continuous system into a discrete system
  - define $s_k = s(\mathrm{t_k})$ on equidistant time grid $\mathrm{t_k} = \mathrm{k}\Delta t$ with sampling time
  - use zero order hold control $\mathrm{a(t) = a_k}$ on $t \in [t_k, t_{k+1}]$
  - use numerical simulation to compute ODE solution (Euler integration, Runge-Kutta RK4)



Numerical simulation and integration

> Simplest implementation is a single step of an Euler integrator
  - $s' = f(s, a) = s + \Delta t \, f_c(s, a)$
  - $\Delta t \to 0$, simulation becomes more accurate

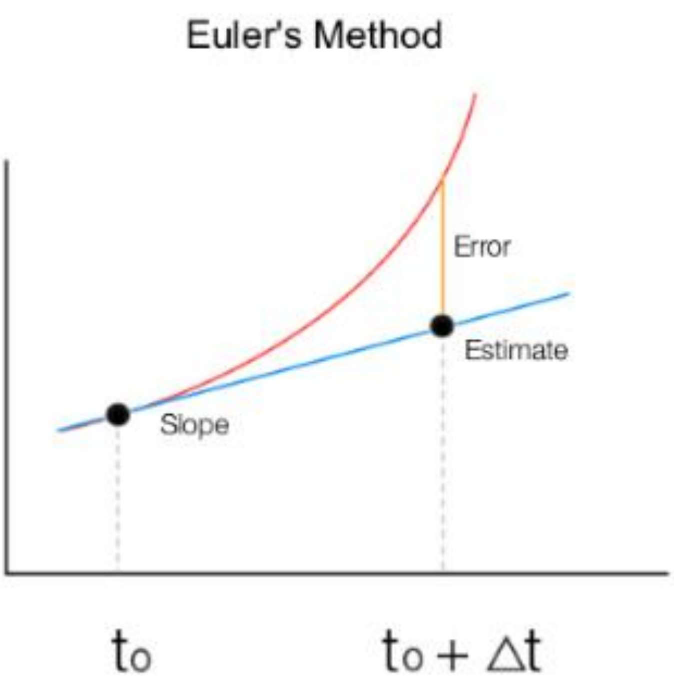
> Even if we have a fixed time step Δt,
  - more accurate simulation can be made by iterating an Euler integrator
  - for i=1:N, $x_{i+1} = \frac{\Delta t}{N} f_c(s, a)$

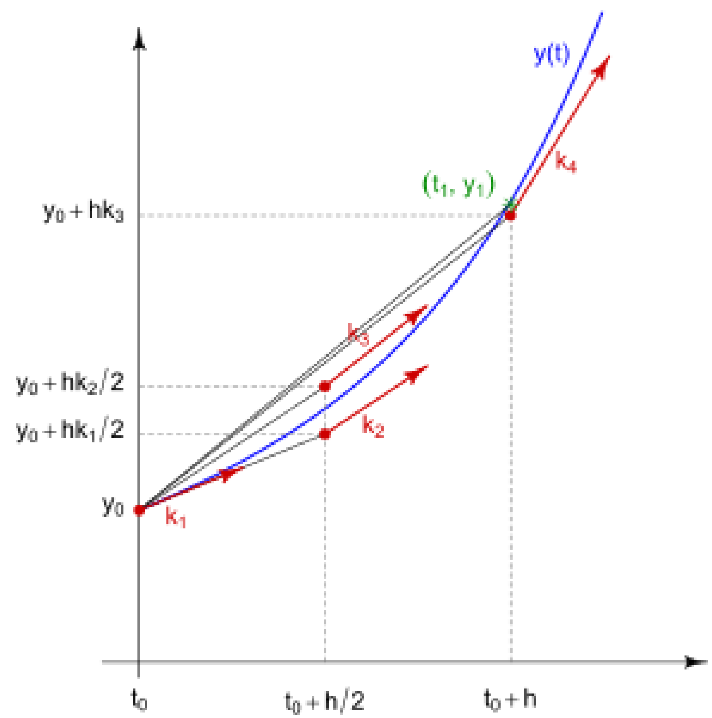
> Runge Kutta method is more accurate and efficient
  - Runge Kutta 4th order
    $f(s, a) = s + \Delta t/6(v_1 + 2v_2 + 2v_3 + v_4)$

$$
\begin{aligned}
v_1 &:= f_c(s, a) \\
v_2 &:= f_c(s + (\Delta t/2)\, v_1, a) \\
v_3 &:= f_c(s + (\Delta t/2)\, v_2, a) \\
v_4 &:= f_c(s + \Delta t \, v_3, a)
\end{aligned}
$$

Numerical simulation and integration

- $\dot{s} = s + a$, for $\Delta t = 1$. Exact solution is $f(s, a) = e = 2.718$
- Four Euler steps give 2.441 (10.2% error), RK4 gives 2.708 (0.36% error)

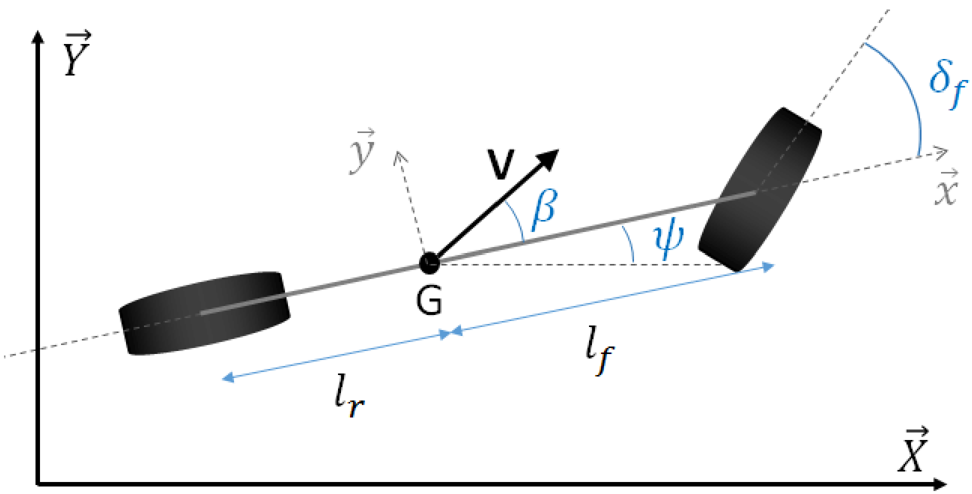




Dynamics model example

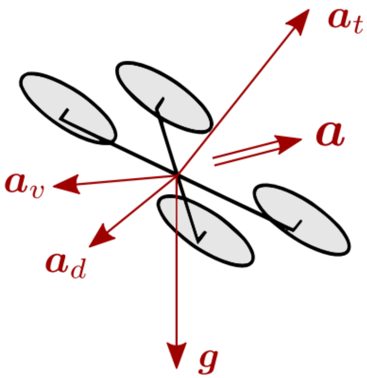
- Bicycle model

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ \psi \\ v \end{bmatrix} = \begin{bmatrix} v\cos(\psi+\beta) \\ v\sin(\psi+\beta) \\ \frac{v}{L_r}\sin\beta \\ a \end{bmatrix}, \text{with } \beta := \arctan(\frac{L_r}{L_r+L_f}\arctan\delta)$$



Dynamics model example

> Drone dynamics



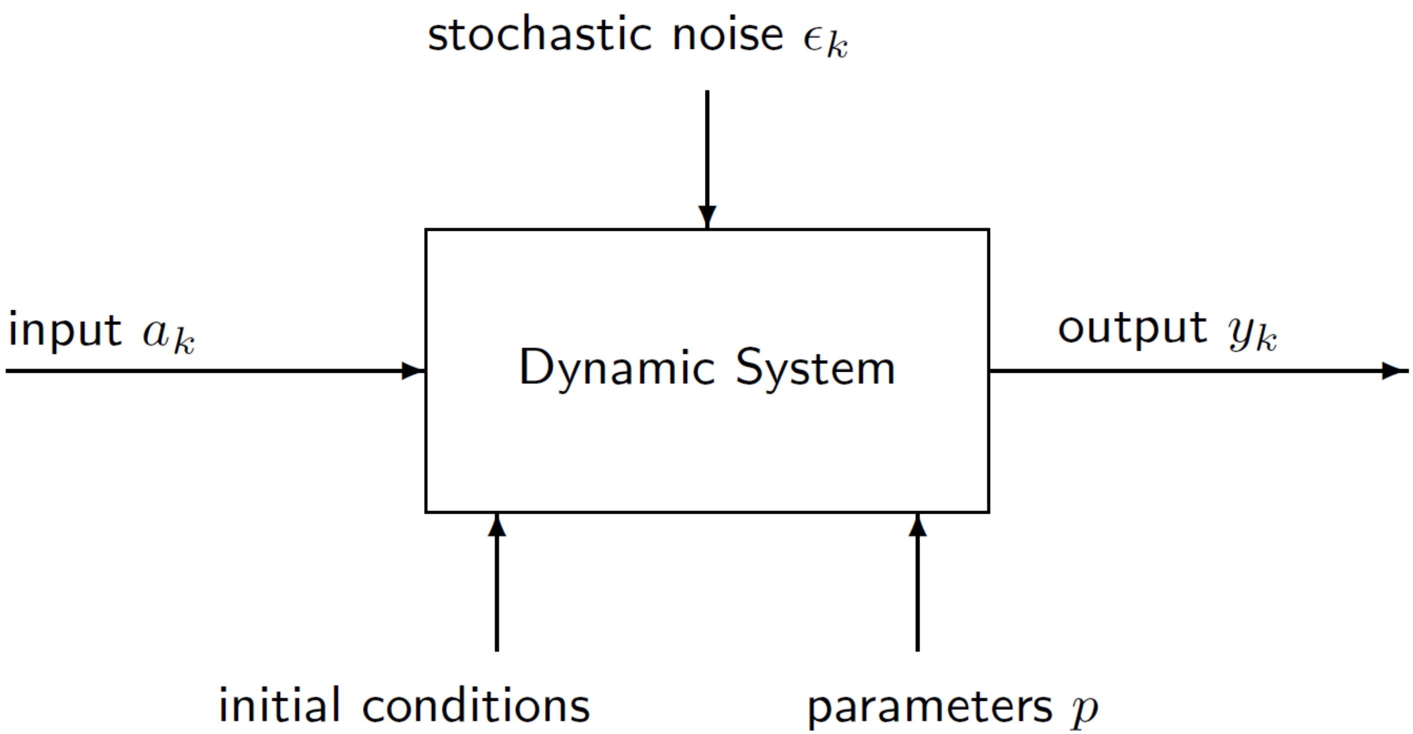
$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \ddot{\phi} \\ \ddot{\theta} \\ \dot{r} \\ \dot{\boldsymbol{a}}_t \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ \dot{v}_x^{(*)} \\ \dot{v}_y^{(*)} \\ \dot{v}_z^{(*)} \\ \dot{\phi} \\ \dot{\theta} \\ \frac{t_\phi}{c_\theta}\dot{\theta} + \frac{1}{c_\phi c_\theta} r \\ -{\omega_{n,\phi}}^2 \phi - 2\zeta_\phi \omega_{n,\phi} \dot{\phi} + {\omega_{n,\phi}}^2 \bar{\phi}_r \\ -{\omega_{n,\theta}}^2 \theta - 2\zeta_\theta \omega_{n,\theta} \dot{\theta} + {\omega_{n,\theta}}^2 \bar{\theta}_r \\ -\sigma_r r + \sigma_r \bar{r}_r \\ -\sigma_t \boldsymbol{a}_t + K \left(\frac{U}{U_n}\right)^\alpha \sigma_t \bar{T} \end{bmatrix}$$

$$\begin{cases} \dot{\boldsymbol{p}} & = \boldsymbol{v} \\ \dot{\boldsymbol{v}} & = \boldsymbol{g} + \frac{1}{m}\mathbf{R}\boldsymbol{f}_t \\ \mathbf{J}\dot{\boldsymbol{\omega}} & = \boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} \end{cases}$$

$$(*) \quad \underbrace{\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix}}_{\dot{\boldsymbol{v}}} = \underbrace{\mathbf{R} \begin{bmatrix} 0 \\ 0 \\ \boldsymbol{a}_t \end{bmatrix}}_{\boldsymbol{a}_t} \underbrace{- \mathbf{R}\mathbf{D}\mathbf{R}^\top \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}}_{\boldsymbol{a}_v} \underbrace{- \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}}_{\boldsymbol{g}}$$

General stochastic models

- We always have some random noise ϵ_k
  - disturbances or measurement errors



State space and MDP

> MDP state space view
  - $s_{k+1} = f(s_k, a_k, \epsilon_k)$
  - $y_k = s_k$

> MDP probabilistic view
  - $p(s_{k+1} | s_k, a_k)$
  - $p(y_k | s_k, a_k) = \delta_d(y_k - s_k)$

> POMDP state space view
  - $s_{k+1} = f(s_k, a_k, \epsilon_k)$
  - $y_k = g(s_k, a_k, \epsilon_k)$

> POMDP probabilistic view
  - $p(s_{k+1} | s_k, a_k)$
  - $p(y_k | s_k, a_k)$

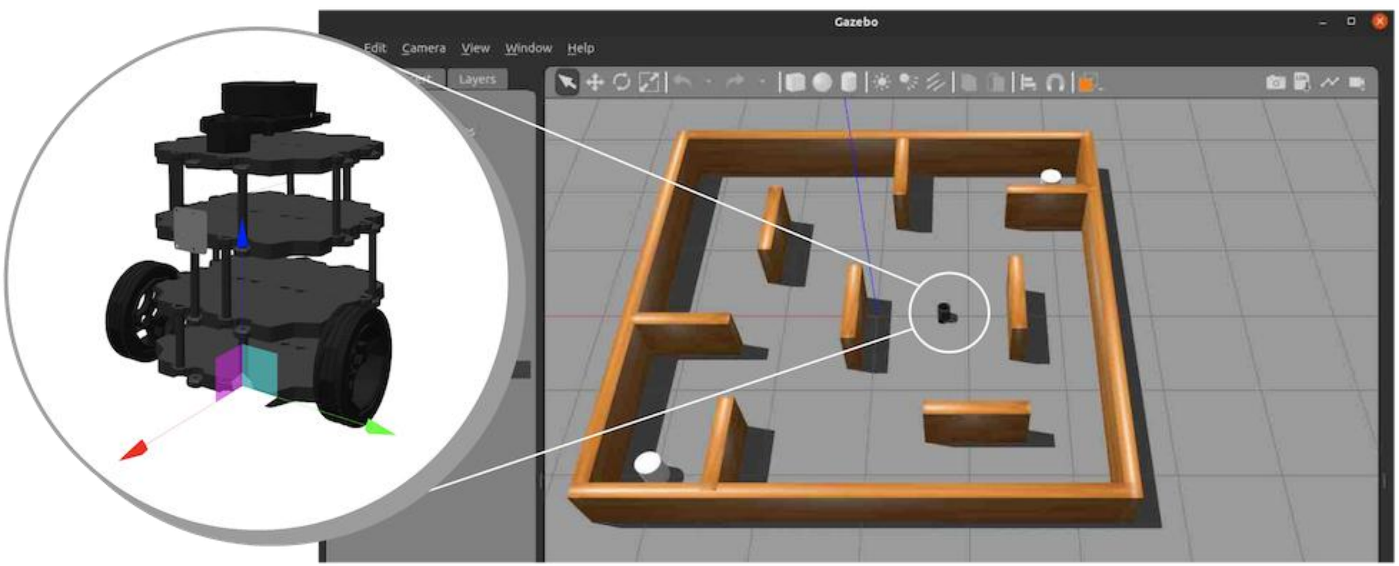
Robot simulators

- Computational physics and computer graphics
- Simulator taxonomy by simulately.wiki

Simulator	Physics Engine	Rendering	Sensor 😳	Dynamics	GPU-accelerated Simulation	Open-Source
IsaacSim	PhysX 5	Rasterization; RayTracing	RGBD; Lidar; Force; Effort; IMU; Contact; Proximity	Rigid;Soft;Cloth;Fluid	✔	✗
IsaacGym	PhysX 5, Flex	Rasterization;	RGBD; Force; Contact;	Rigid;Soft;Cloth	✔	✗
SAPIEN	PhysX 5, Warp	Rasterization; RayTracing ⭐;	RGBD; Force; Contact;	Rigid;Soft;Fluid	✔	✔
Pybullet	Bullet	Rasterization;	RGBD; Force; IMU; Tactile;	Rigid;Soft;Cloth	✗	✔
MuJoCo	MuJoCo	Rasterization;	RGBD; Force; IMU; Tactile;	Rigid;Soft;Cloth	✔ 💡	✔
CoppeliaSim	MuJoCo; Bullet; ODE; Newton; Vortex	Rasterization; RayTracing 🔶;	RGBD; Force; Contact;	Rigid;Soft;Cloth	✗	✔
Gazebo	Bullet; ODE; DART; Simbody	Rasterization;	RGBD; Lidar; Force; IMU;	Rigid;Soft;Cloth	✗	✔
Genesis	Genesis	Rasterization; Raytracing	RGBD; Tactile; (update soon)	Rigid;Soft;Cloth;Fluid	✔	✔

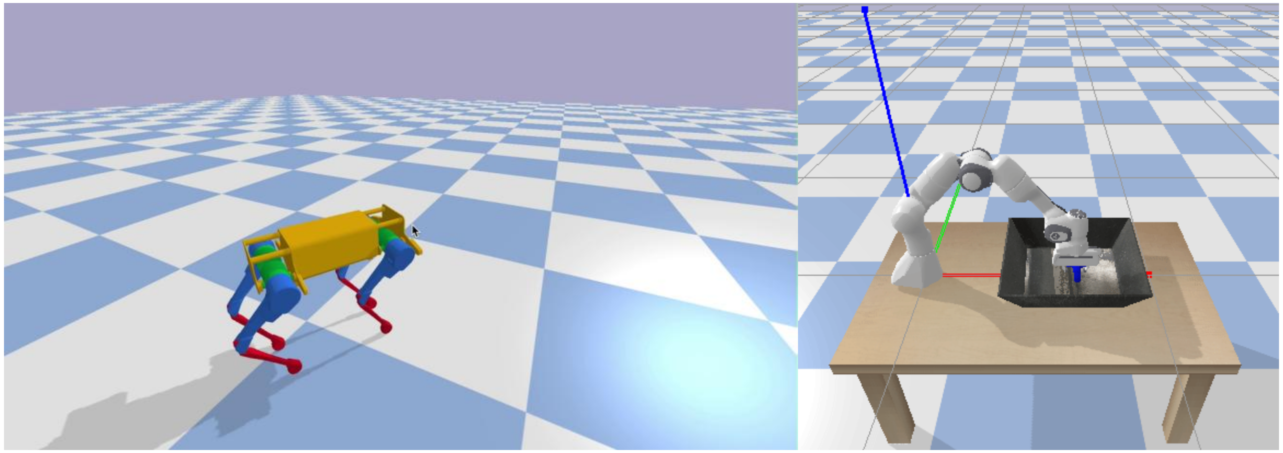
Gazebo

> Developed from 2002
> Open-source robotic simulator
  - Working well with ROS
  - GPU acceleration is not supported
  - Physics engine: Bullet, ODE, DART, Simbody



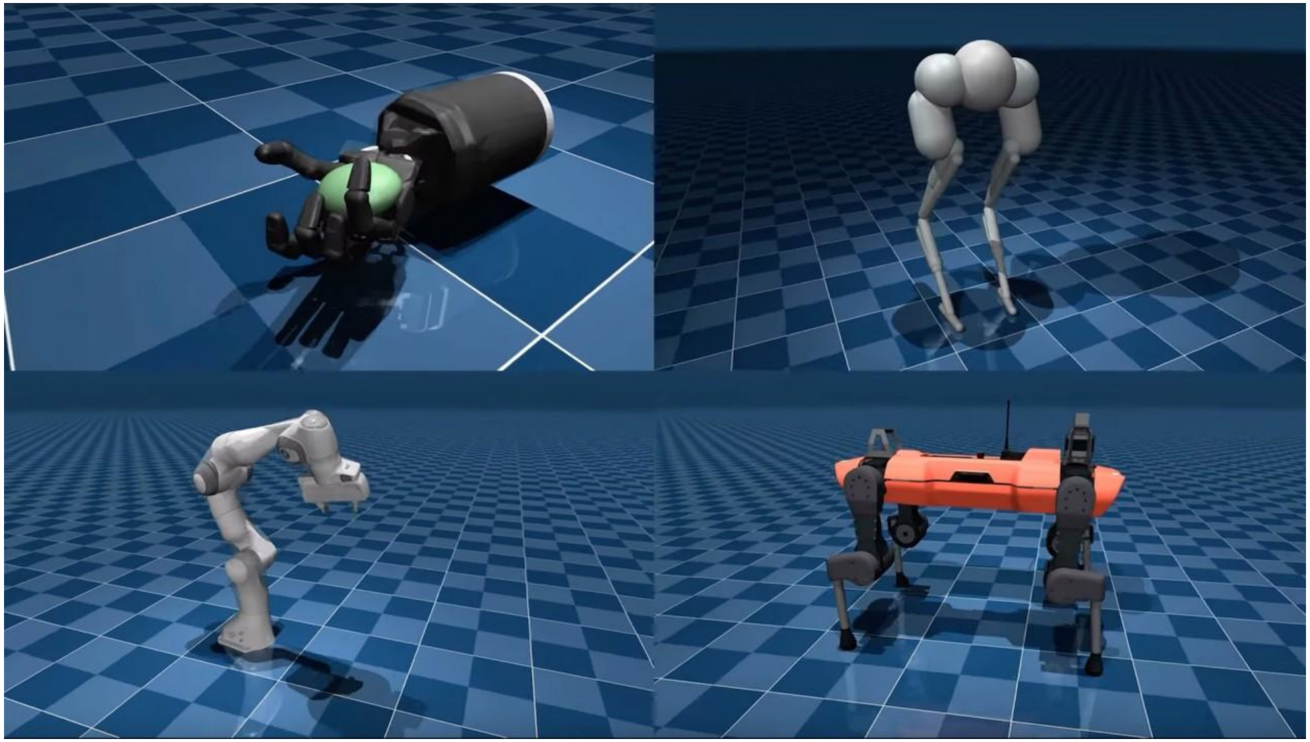
PyBullet

- Python + Bullet physics engine
- Free and open source, easy to use
  - GPU acceleration is not supported



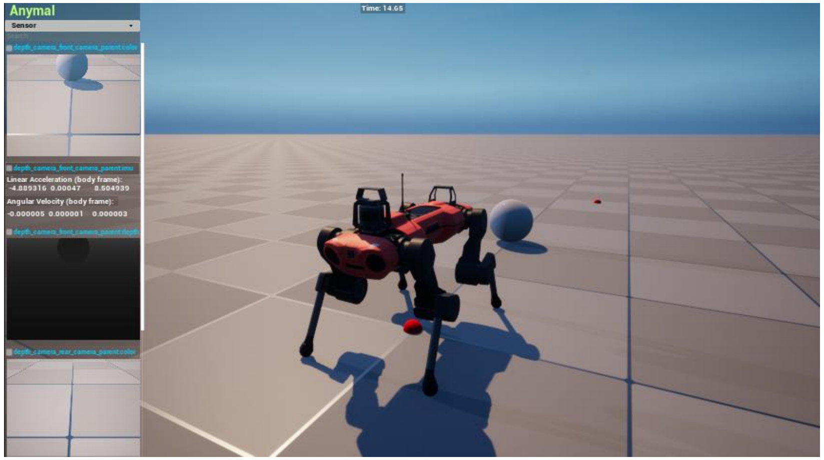
MuJoCo

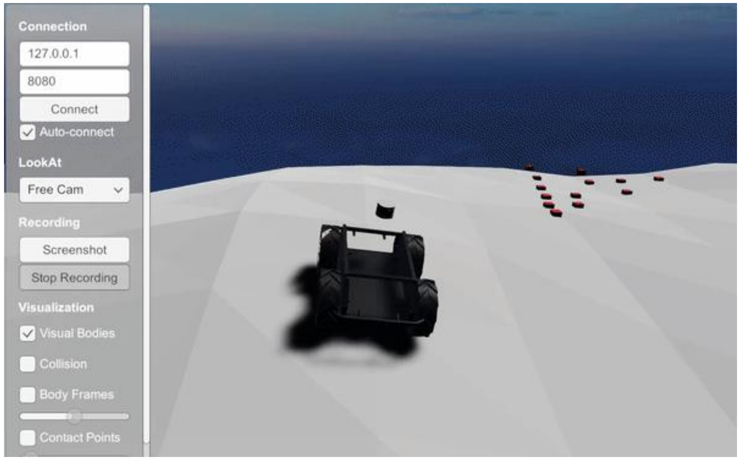
- Developed by Roboti LLC, acquired and made freely available by DeepMind
  - Multi-Joint Dynamics-with-Contact
  - Accurate and efficient contact dynamics (compliant contact model)
  - Differentiable



RaiSim

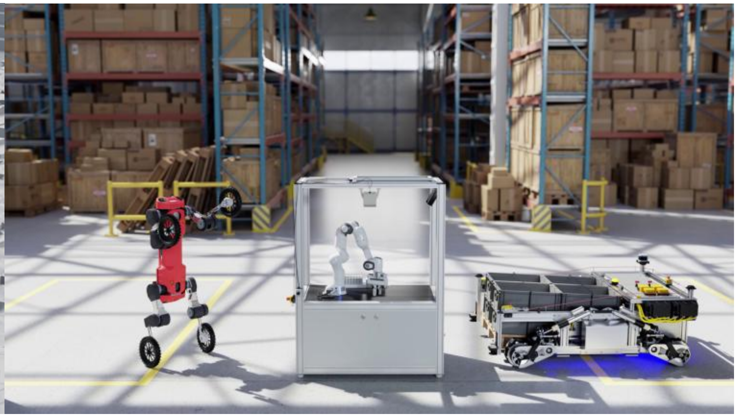
- Developed by Jemin Hwangbo (KAIST)
  - Four major computational components in rigid-body simulators
  - Collision detection, forward dynamics computation, contact properties computation, contact solver
  - Visualization options – Unity, Unreal, Ogre





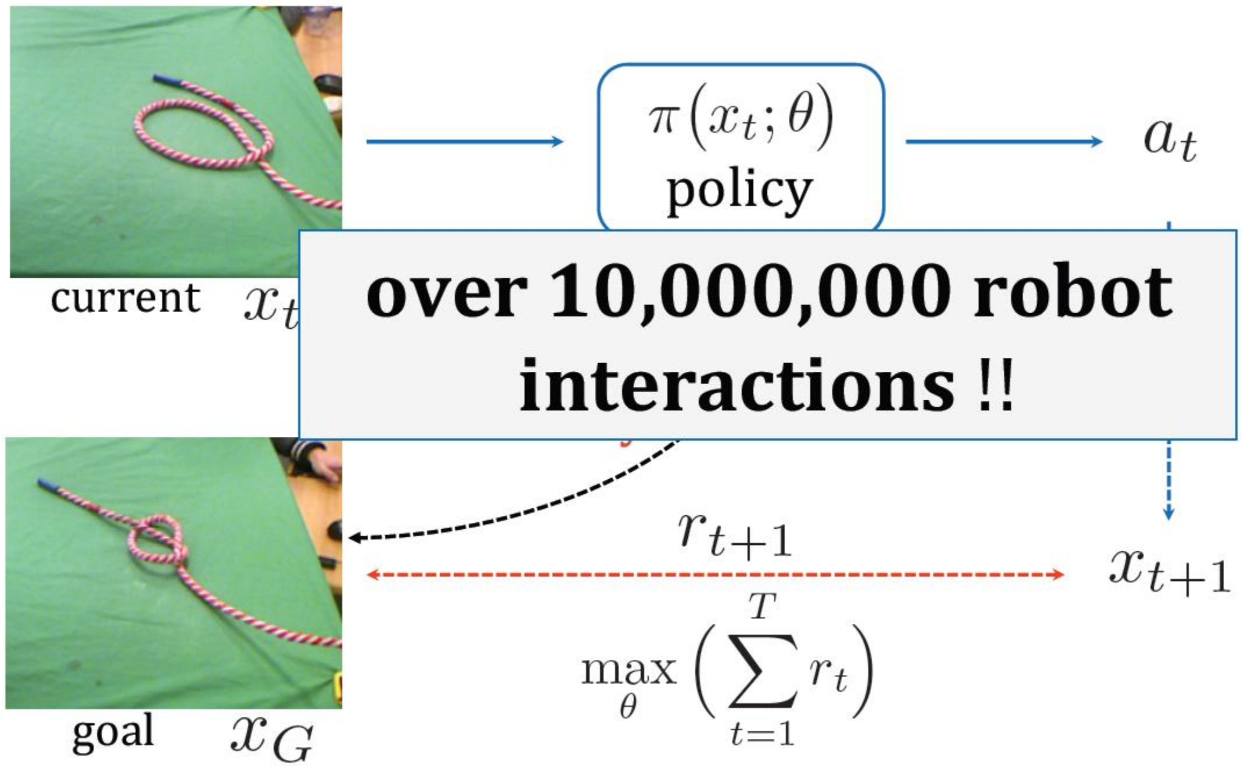
IssacSim

- Developed by NVIDIA
  - End-to-end GPU accelerated
  - Massive parallelization of thousands of environments
  - PhysX 5 physics engine, including fluid dynamics

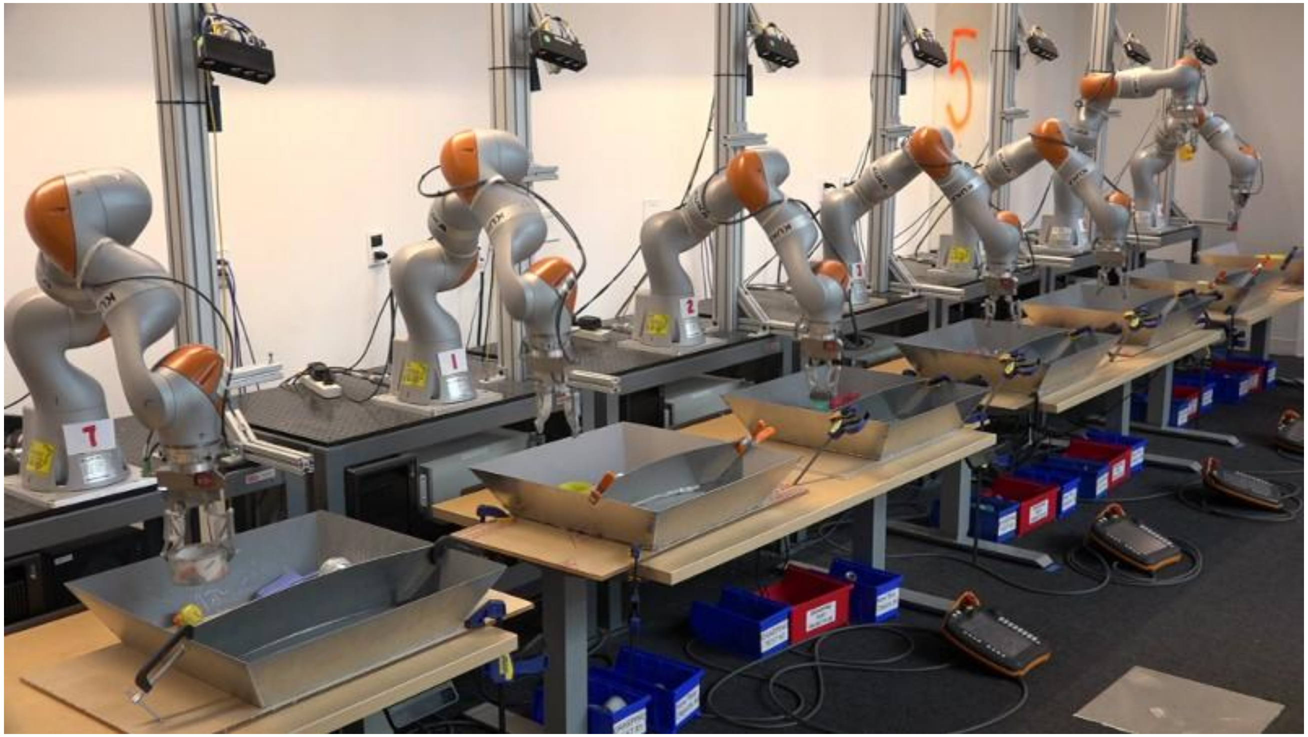
Why simulators for robot learning

> RL is very sample inefficient



Why simulators for robot learning

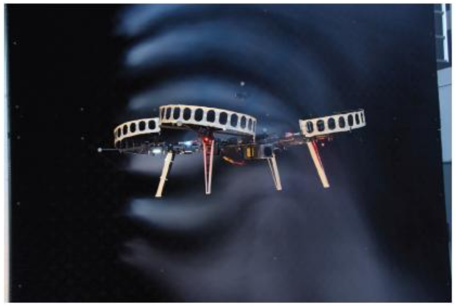
- Google QT-Opt
  - 4 months, 800 robot hours, 7 robots, 580,000 attempts



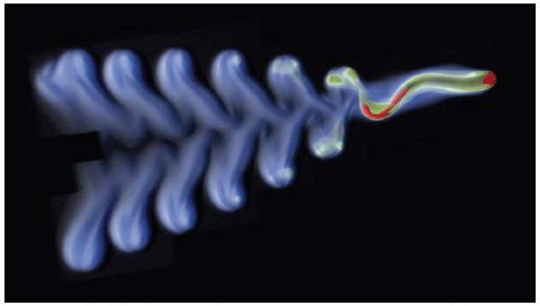
Why simulators for robot learning

- Advantages of using simulated data
  - Cheap, fast, and scalable
  - Safe
  - Labeled (we have access to ground truth
  - No physical harm or deformation (wear and tear)

- Disadvantages
  - Not exactly same with real env. (sim2real)



Aerodynamics in wind, *Neural-Fly*

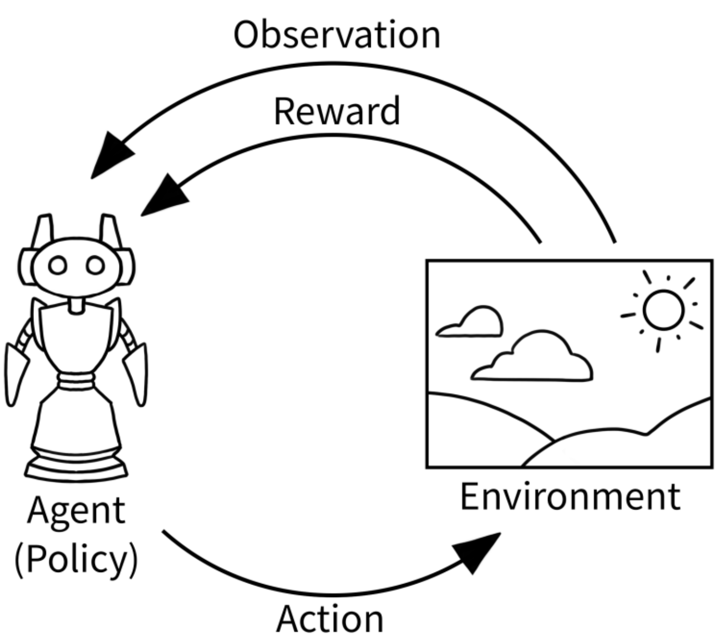


Fluid dynamics, MIT van Rees Lab

Offroad vehicle dynamics, UW Racer team

Environment for RL

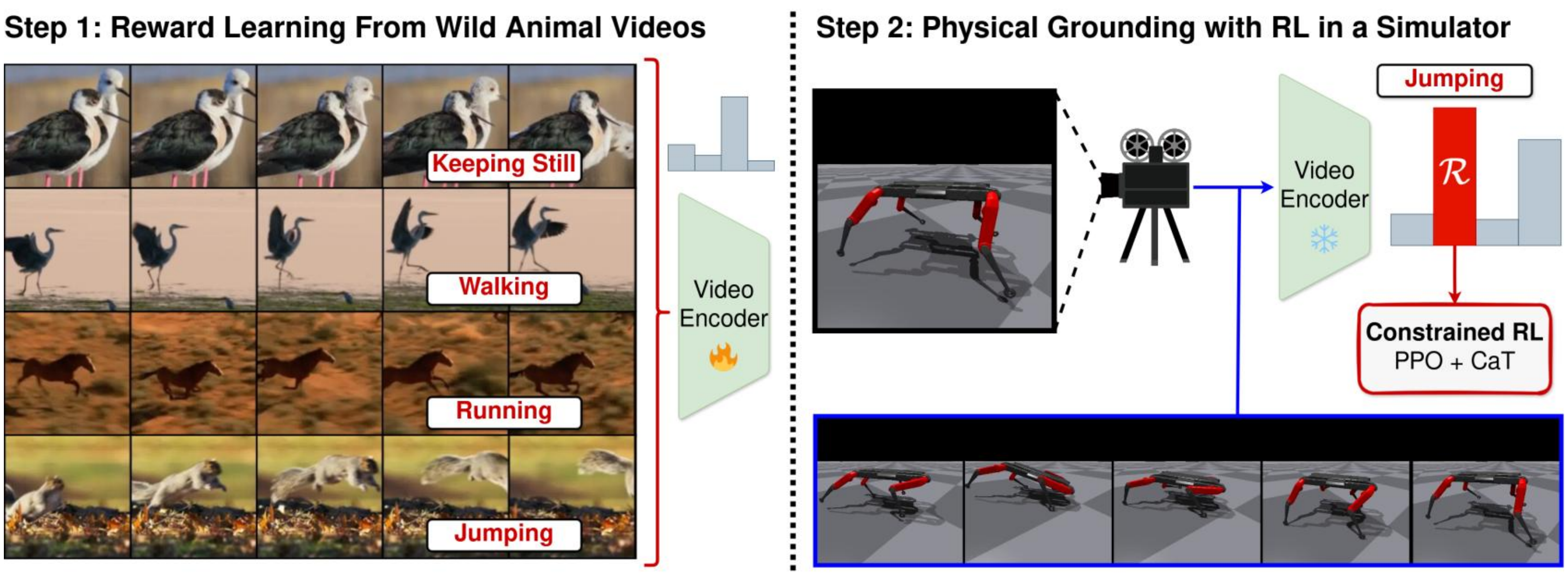
- Gymnasium
  - Provide an API for all single agent RL envs.
  - Maintained by OpenAI




```
env = gym.make
observation, info = env.reset
action = env.action_space.sample()
observation, reward, terminated,
truncated, info = env.step(action)
```

Exploiting abundant data

> RL from wild animal videos



More on simulators

> https://github.com/knmcguire/best-of-robot-simulators

> Gymnasium: https://gymnasium.farama.org/

> Gymnasium-robotics: https://robotics.farama.org/

> 2D sim: https://ir-sim.readthedocs.io/en/stable/

> Drone: https://github.com/utiasDSL/gym-pybullet-drones

> Autonomous driving: https://carla.org/

> Manipulator / Legged robot / Humanoid (contact-dynamics): MuJoCo, Issac Sim, RaiSim, Genesis ...