Deep Reinforcement Learning

Cyber-Physical Systems Programming M
December 2024
-
Sebastiano Mengozzi
sebastiano.mengozzi@unibo.it

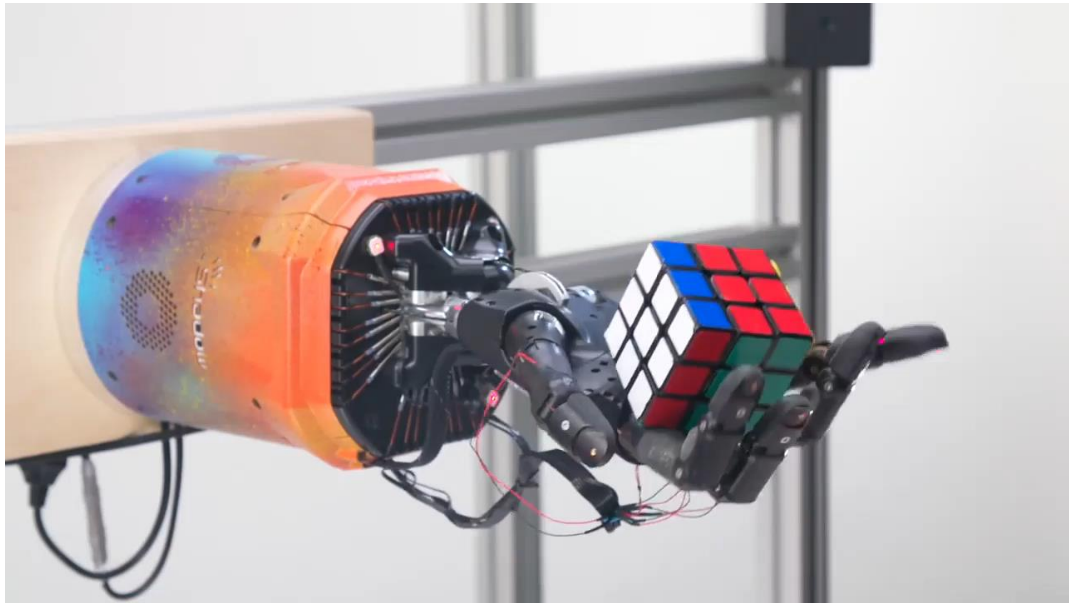
Outline

Background Material and Credits – pt.1

- Sutton & Barto, 2018, Reinforcement Learning: An Introduction
- Deepmind x UCL, 2021, Reinforcement Learning Course
- UC Berkeley, 2023, Deep Reinforcement Learning CS285
- Laura Graesser, 2019, Foundations of Deep Reinforcement Learning

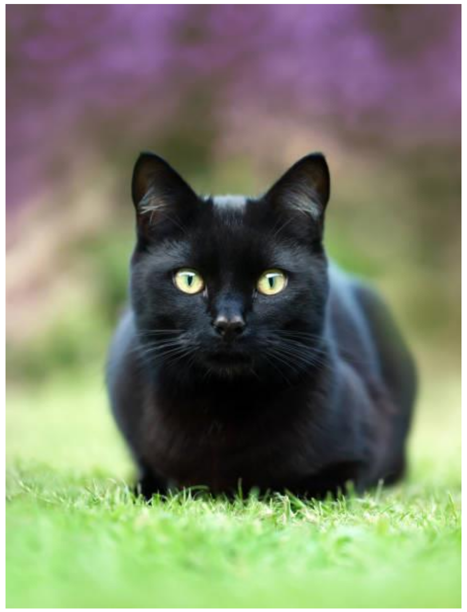
Goal

- Demystifying Deep Reinforcement Learning: it is not magic
- Understand core mechanisms that lead to the solution of dynamical problems



Introduction to Reinforcement Learning

- Success of Artificial Intelligence
  - Allow the machine to find solutions themselves
  - Specify high-level goal, show examples
    - e.g., classification



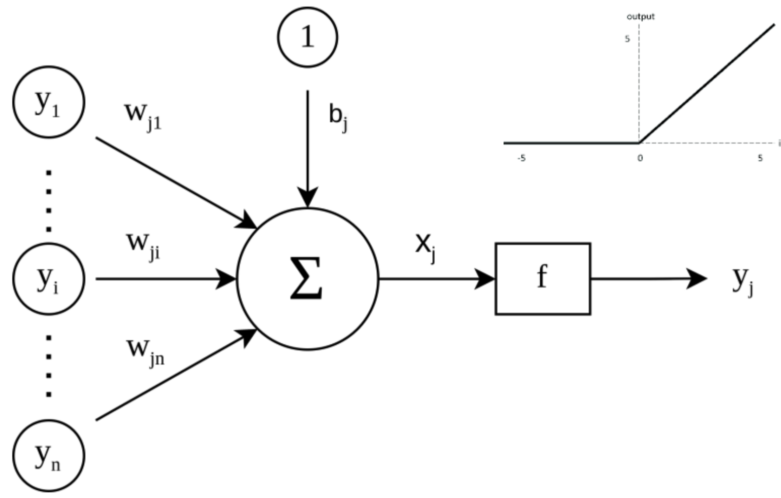
Introduction to Reinforcement Learning

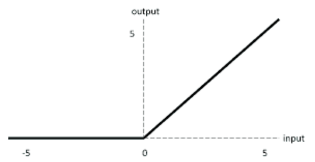
- Success of Artificial Intelligence
  - Allow the machine to find solutions themselves
  - Specify high-level goal, show examples
    - e.g., classification

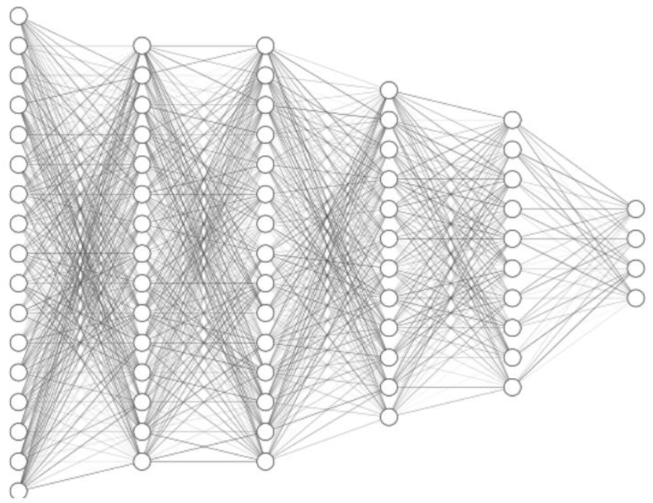
  - Instead of trying to produce a program to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain.
    - Alan Turing (mid-1900s)

Neural Networks

- Neuron model
  - Weights and bias, NN trained parameters
  - Activation function: nonlinear
  - Neural Networks: layers of neurons (input, hidden, output)
  - Nonlinearity + at least one hidden layer = General Function Approximation

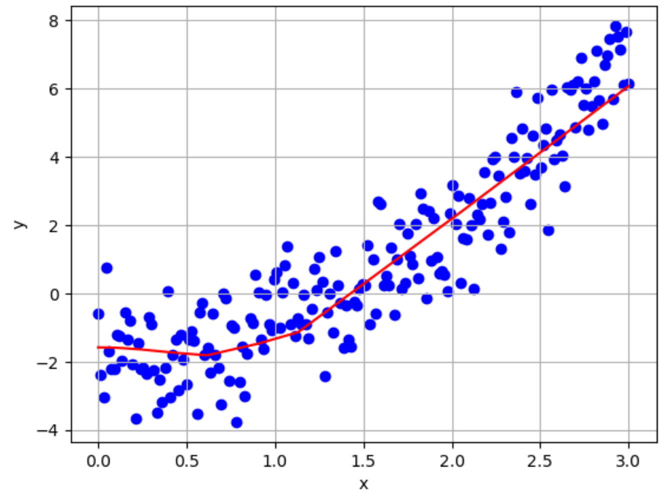






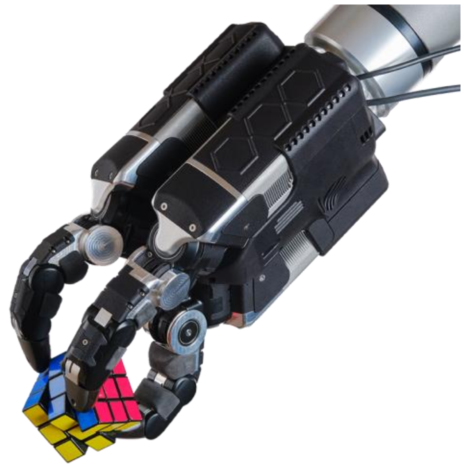
Neural Networks

- Neuron model
  - Weights and bias, NN trained parameters
  - Activation function: nonlinear
  - Neural Networks: layers of neurons (input, hidden, output)
  - Nonlinearity + at least one hidden layer = General Function Approximation



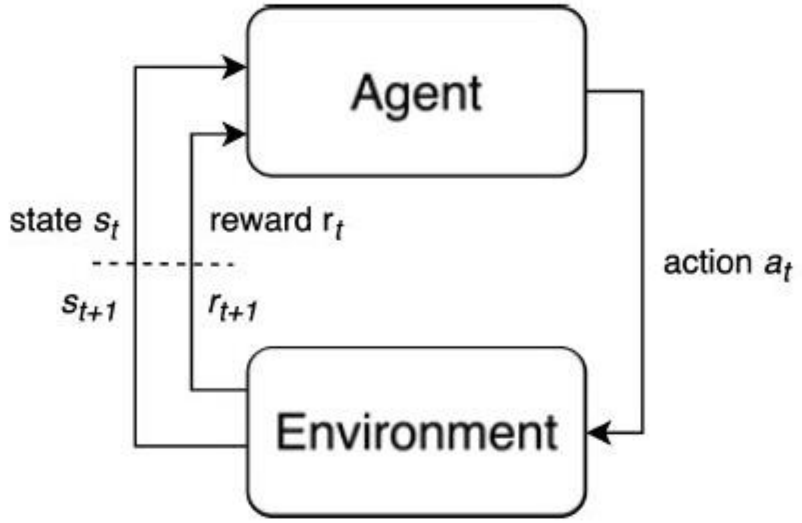
RL vs Supervised and Unsupervised Learning

- Supervised learning
  - Labeled dataset
  - Oracle
- Unsupervised learning
  - Not labeled dataset
  - Lack of supervision
- Reinforcement learning
  - Dataset generated online (e.g. robotics)
  - No complete lack of supervision (online feedback, *reward*)



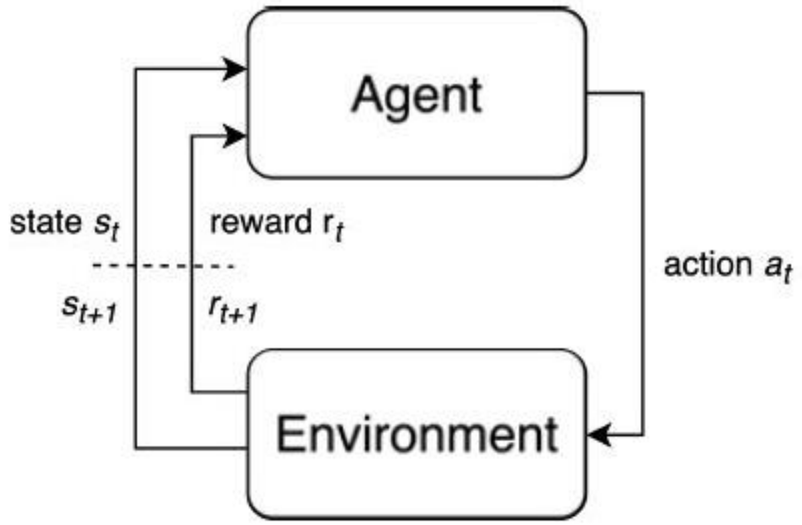
The Reinforcement Learning Loop

- Feedback loop system of Agent and Environment
  - Action (a)
  - State or Observation (s)
  - Reward (r)
- Goal: optimize the sum of rewards through repeated interactions



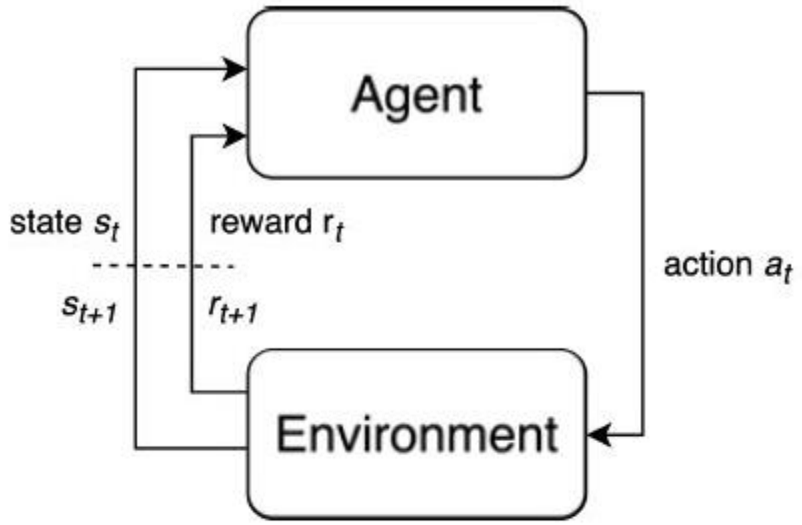
The Reinforcement Learning Loop

- The actions are selected by the Agent according to a policy (π)
- The policy (π) is a function that maps observations into actions
- Actions can be discrete (e.g., turn left/right, move forward/backward) or continuous (e.g., steering angle)



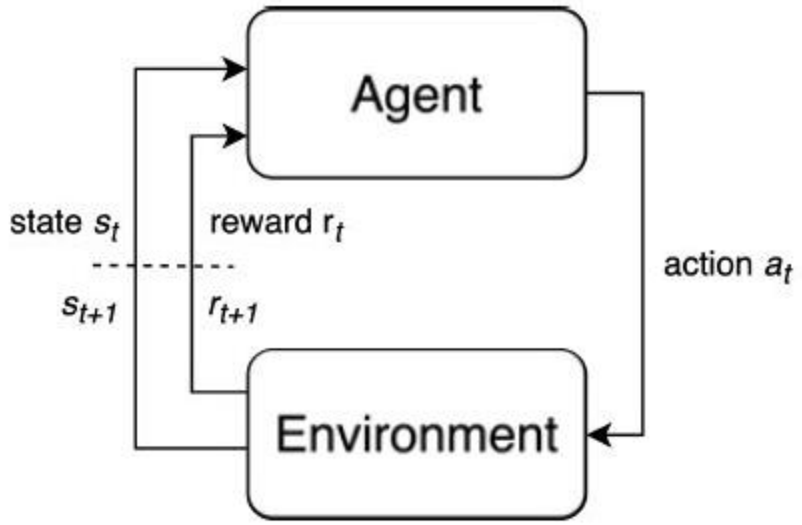
The Reinforcement Learning Loop

- The observation is what the Agent sees of the Environment state before choosing an action
- E.g., pixels, state variables of a system, sensor readings



The Reinforcement Learning Loop

- The reward guides the system in reaching the goal
- Positive or negative number (good or bad interaction)
- Given after every interaction of the Agent with the Environment



The Reinforcement Learning Loop

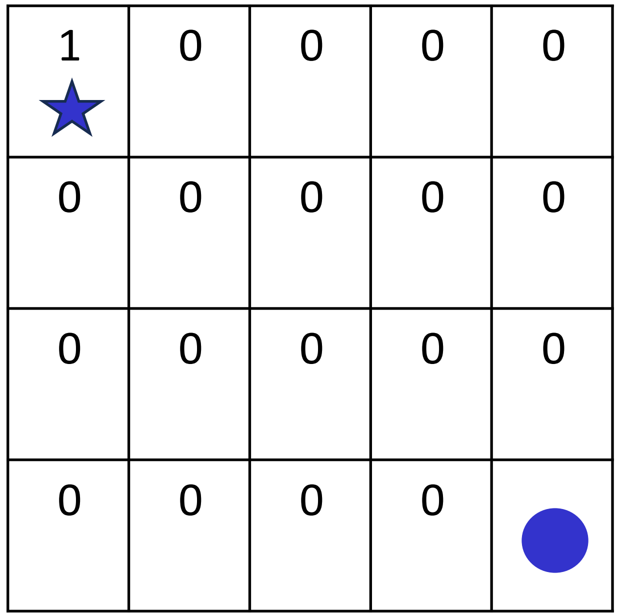
- Goal: maximize the cumulative reward (return):

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ \ldots$$

- Based on the **reward hypothesis**: *any goal can be formalized as the outcome of maximizing a cumulative reward*
- Reward examples:
  - Flying a drone: distance from a target point, airtime, stability, …
  - Play ATARI games: maximize score, survival time, …
  - Autonomous driving: avoid obstacles, maximize fuel efficiency, …

Learning Values Functions

- Goal: optimize the sum of rewards through repeated interactions
- Value functions more informative than rewards
- Learn to maximize the reward through value functions
  - More informative than rewards, less sparse
- How to learn them? More on that later



1	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	

Value Function

- Expected cumulative reward, from a state s

$$V^{\pi}(s) = \mathbb{E}[G_t | s_0 = s, \pi] = \mathbb{E}[\sum_{t=0}^{T} \gamma^t R_t | s_0 = s, \pi]$$

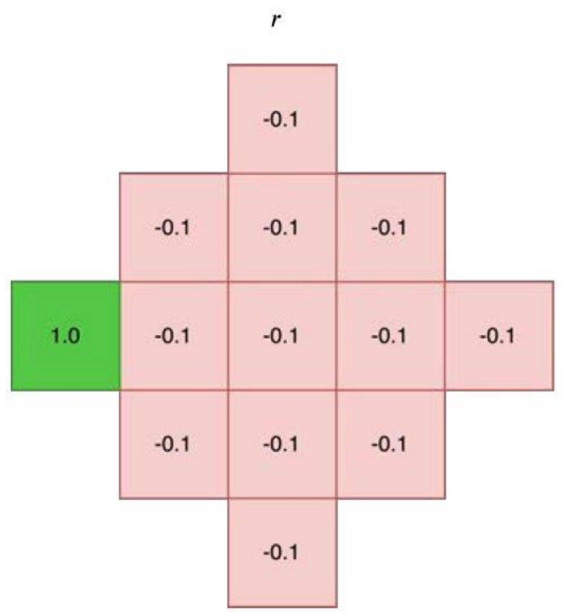
- Value of being in a certain state
- Discount factor $\gamma \in [0,1]$
  - Trades off importance of immediate vs long-term rewards
- Depends on a policy π

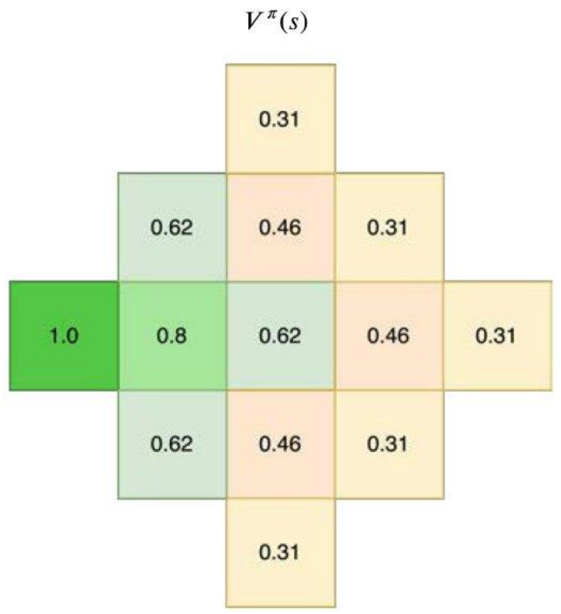
Value Function

- Expected cumulative reward, from a state s

$$V^{\pi}(s) = \mathbb{E}[G_t | s_0 = s, \pi] = \mathbb{E}[\textstyle\sum_{t=0}^{T} \gamma^t R_t | s_0 = s, \pi]$$

- Example: π shortest path, $\gamma = 0.9$





Action Value Function

- Expected cumulative reward, from a state s and action a

$$Q^{\pi}(s,a) = \mathbb{E}[G_t |\, s_0 = s\,; a_0 = a, \pi] = \mathbb{E}[\textstyle\sum_{t=0}^{T} \gamma^t R_t |\, s_0 = s\,; a_0 = a, \pi]$$

- Value of taking an action in a specific state
- Value function and Action value function are linked:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)]$$

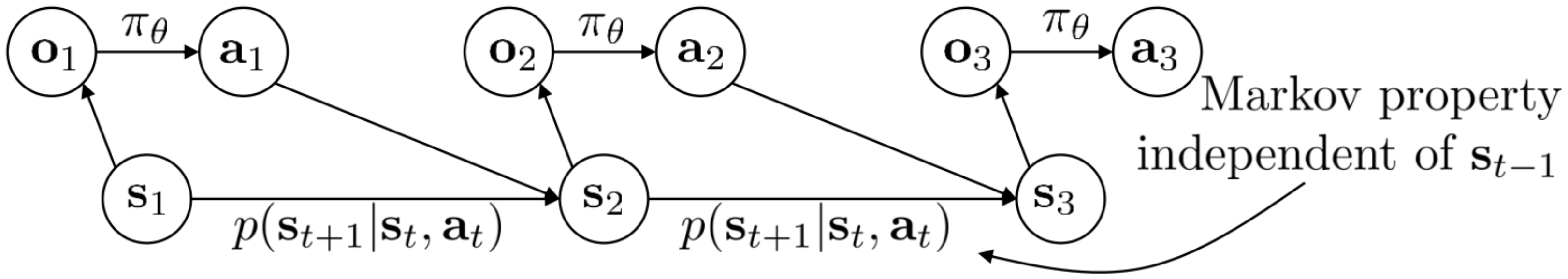
Advantage Function

- Measures if an action is better or worse than the policy’s average action in a particular state

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

- Removing the bias of the current state
- Considering expected values of Q^{π} and V^{π}
- $A^{\pi} > 0$, positive action outcome
- $A^{\pi} < 0$, negative action outcome

Markov Decision Process

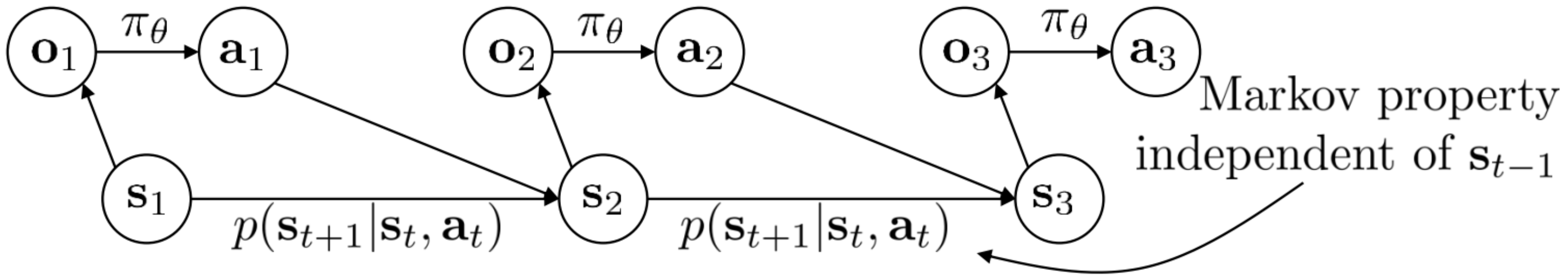


- Markov Decision Process (MDP): mathematical formulation of agent-environment interaction

$$M = (A, S, p, r, \gamma)$$

- Transition model not known: dealing with probabilities and expected values
- Discount avoids infinite returns in cyclic Markov processes

Markov Decision Process



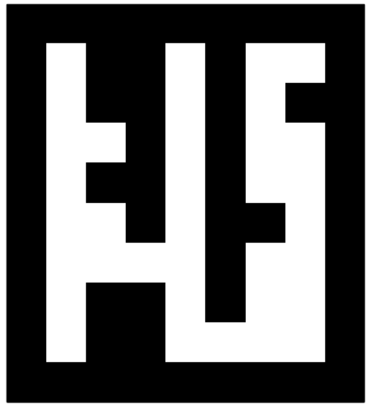
- Markov Decision Process (MDP): mathematical formulation of agent-environment interaction

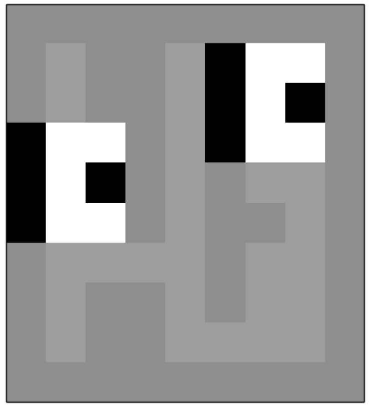
$$M = (A, S, p, r, \gamma)$$

- Markov property: the future is independent of the past given the present
- The state captures all relevant information from the history
  - The state is a sufficient statistic of the past

Markov Decision Process

- Full observability
  - The agent sees the full environment state
  - State = Observation
- Partial observability
  - Examples: robot with camera vision, poker playing agent only observes public cards
  - Called partially observable Markov decision process (POMDP)





Learning Values Functions

- Goal: optimize the sum of rewards through repeated interactions
- Value functions more informative than rewards
- Learn to maximize the reward through value functions
- Techniques:
  - Dynamic programming
  - Monte Carlo sampling
  - Temporal Difference learning

Dynamic programming

- Learning through updates derived from the Bellman Optimality Equation

$$\forall s : V_{k+1}(s) \leftarrow \max_a \mathbb{E}[R_{t+1} + \gamma V_k(S_{t+1}) \mid S_t = s, A \sim \pi(S_t)]$$

- By maximizing over a, the optimal Value function is found
- Computational complexity is $O(|S^3|)$ – only possible for small problems

Monte Carlo Learning (MC)

- Update Value function towards *sampled* returns

$$V_{k+1}(s_t) \leftarrow V_k(s_t) + \alpha\big(G_t - V_k(s_t)\big)$$

- Unbiased estimate of the value function
- MC must wait until the end of the episode before the return is known
- Learn from complete trajectories (storage)
- Low bias, high variance

Temporal Difference Learning (TD)

- Update Value function towards *estimated* returns

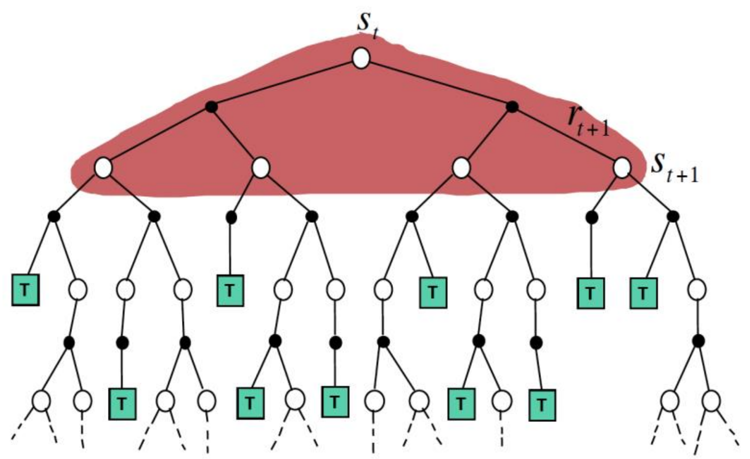
$$V_{k+1}(s_t) \leftarrow V_k(s_t) + \alpha\big(R_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t)\big)$$

- α multiplies the TD-error
- Learn online, from single transitions and incomplete sequences
- High bias, low variance

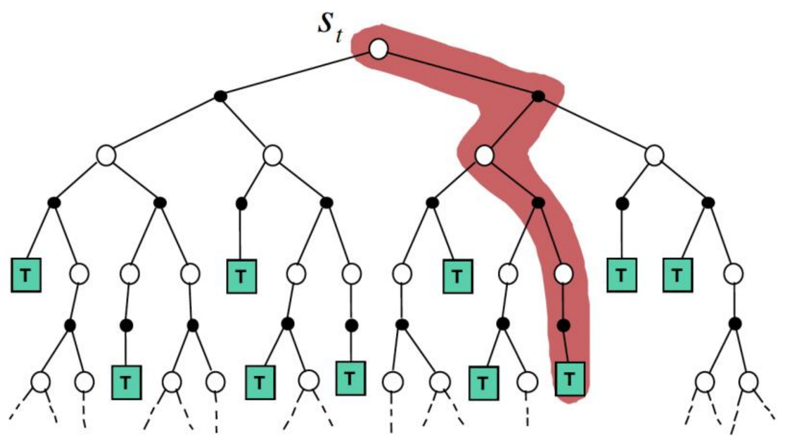
- Same idea for Action Value functions

Dynamic Programming vs MC vs TD

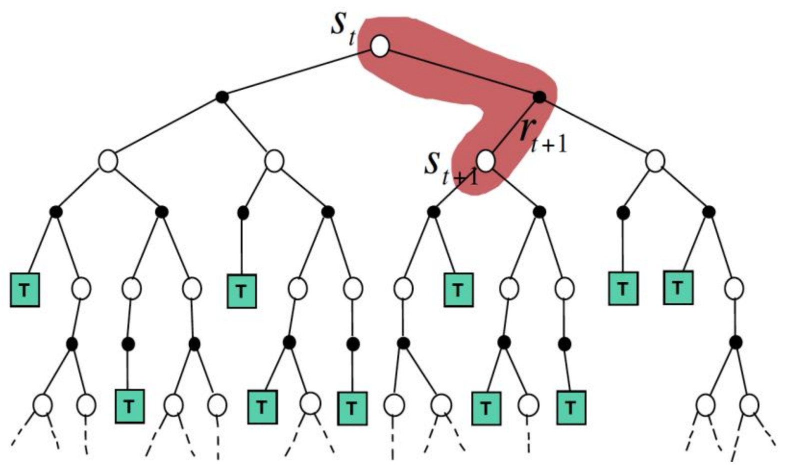
$$v(S_t) \leftarrow \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid A_t \sim \pi(S_t)\right]$$



$$v(S_t) \leftarrow v(S_t) + \alpha\left(G_t - v(S_t)\right)$$



$$v(S_t) \leftarrow v(S_t) + \alpha\left(R_{t+1} + \gamma v(S_{t+1}) - v(S_t)\right)$$



Learning Policies

- Learn to solve a goal by learning Values functions: solving an indirect problem
- Now learn directly the policy: function that maps states to actions

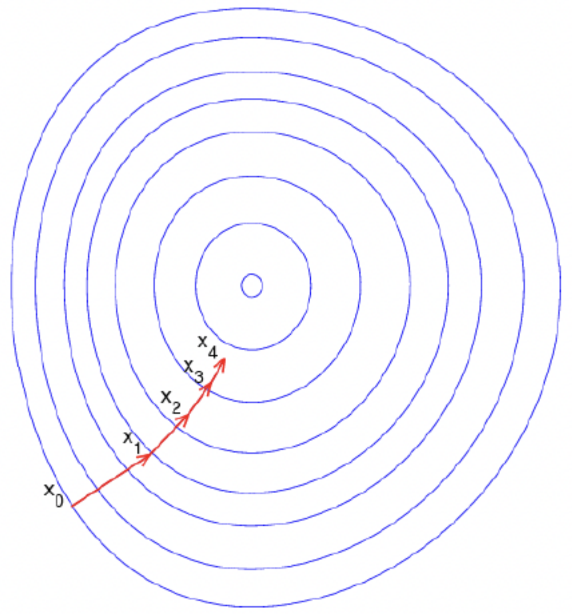
$$\pi_\theta(a|s) = p(a|s,\theta)$$

- Policy based reinforcement learning is an optimization problem
- Find the best parameters θ that maximizes the objective $J(\theta)$
- Idea ascent the gradient: based on the *Policy Gradient Theorem*

$$\Delta\theta = \alpha\nabla_\theta J(\theta)$$

- Update:

$$\theta_{t+1} = \theta_t + \alpha\big(R_{t+1} - b(S_t)\big)\nabla_\theta log\pi(A_t|S_t)$$



Learning Policies

- Idea ascent the gradient: based on the *Policy Gradient Theorem*

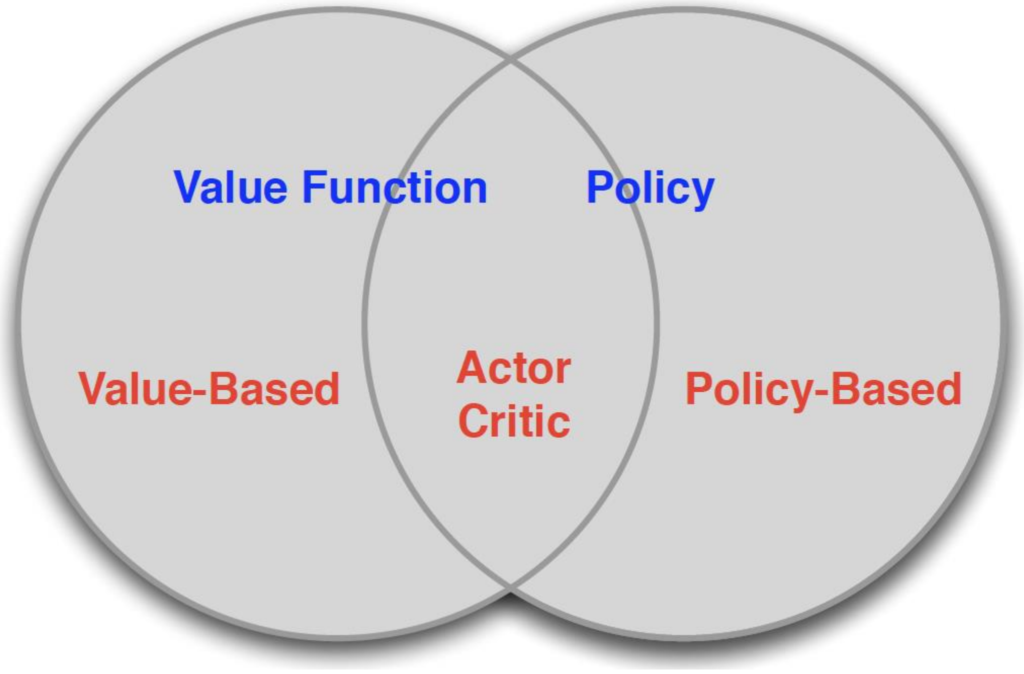
$$\Delta\theta = \alpha \nabla_{\theta} J(\theta)$$

- Update:

$$\theta_{t+1} = \theta_t + \alpha\big(R_{t+1} - b(S_t)\big)\nabla_{\theta} log\pi(A_t|S_t)$$

- *b*: baseline reduce variance, not the dependent on the action (Advantage value function)
- Shaping the probability at training time

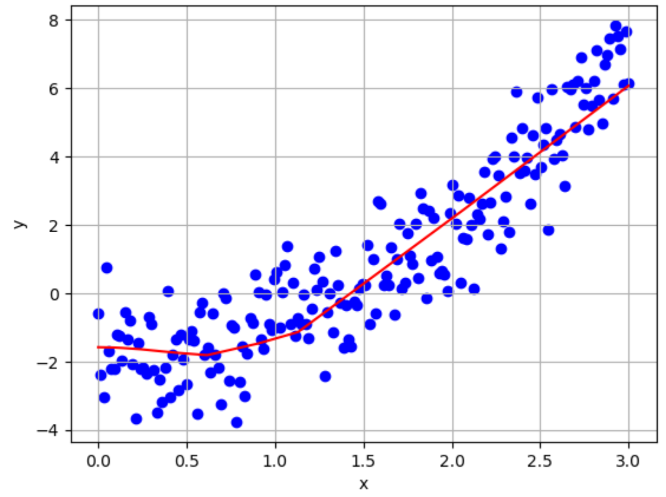
Actor-Critic



- Exploit the best of the both worlds: Actor-Critic
- A policy is reinforced with a reinforcing signal generated using a learned value-function

Approximation with NNs

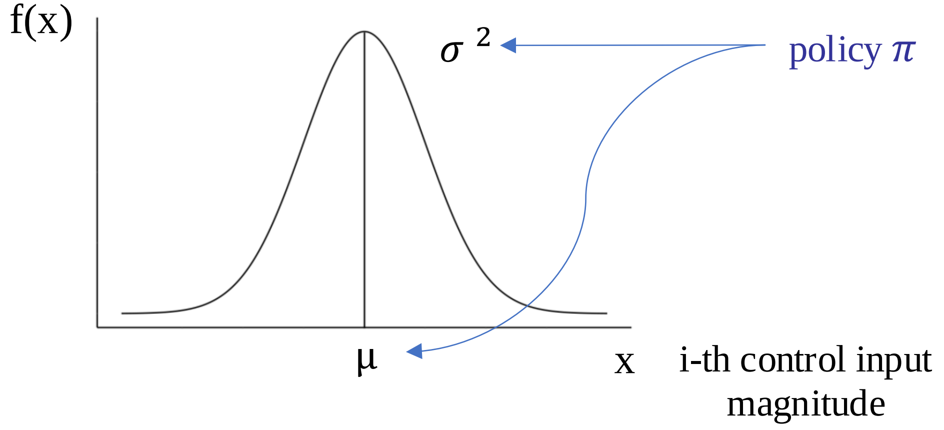
- Value functions and policies can be approximated with *Deep Neural Networks*
- General function approximators
- **Deep Reinforcement Learning** algorithms



Advantage Actor-Critic (A2C)

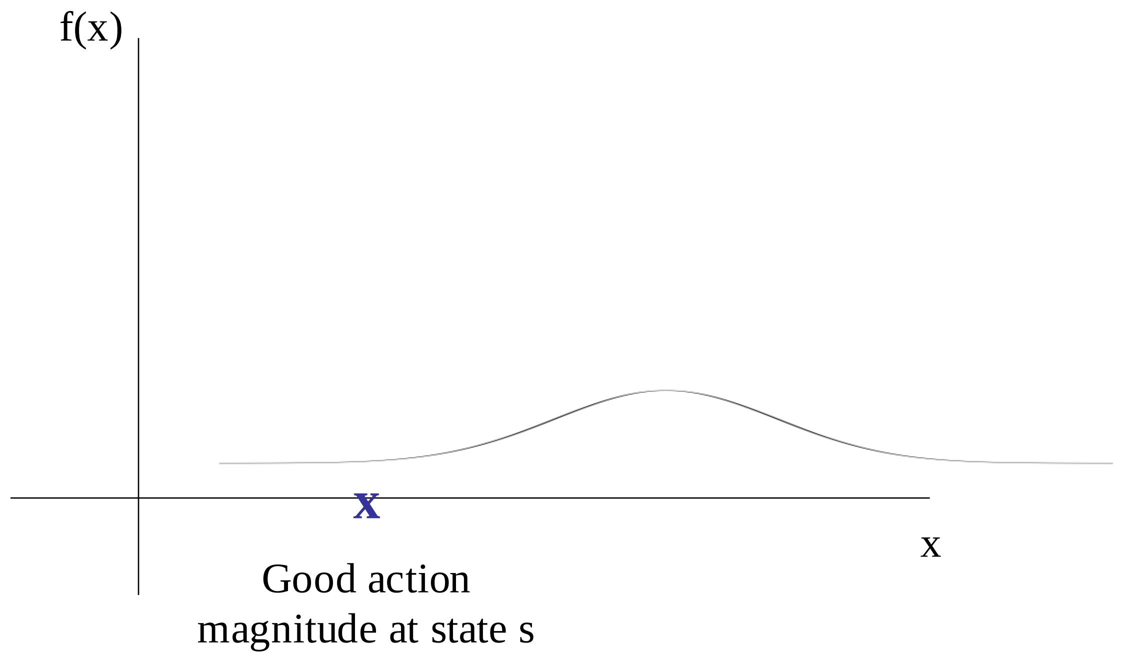
- Two components
  - *Actor*, which learns a parametrized policy
  - The policy π is a neural network which outputs *means* and *variances* of normally distributed random variables, one for each control input (action).
  - Probability of selecting a specific magnitude for the continuous control input given the current state.
  - Explorative behavior

Probability distribution of the i-th control input



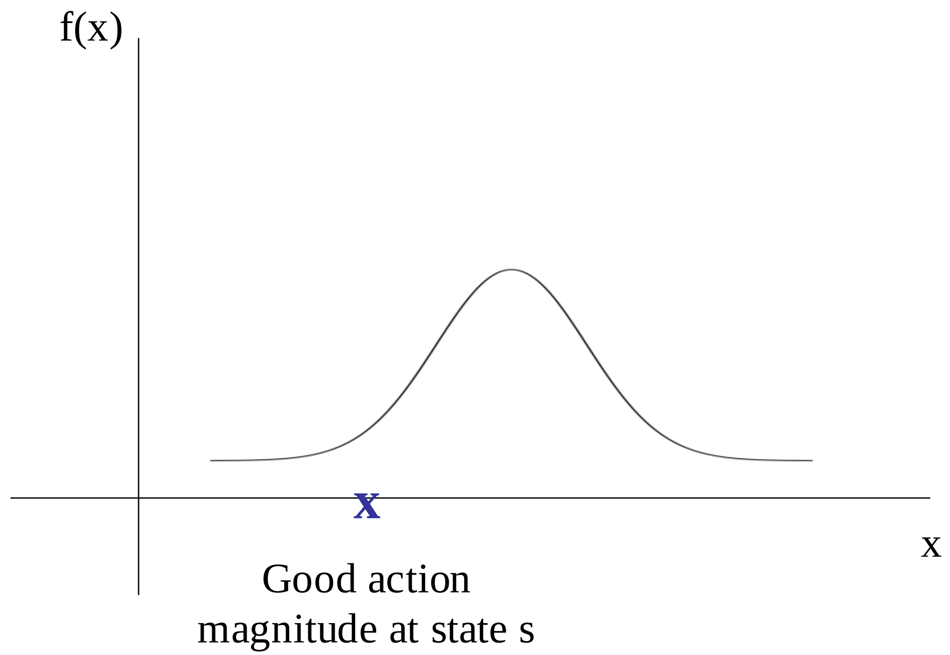
Actor

- During the training, the actor becomes, thanks to the critic, more confident about the action to take at state s (lowers the variance of the distribution)
- The good action magnitude becomes the most probable
- When deployed the mean value is used, no sampling



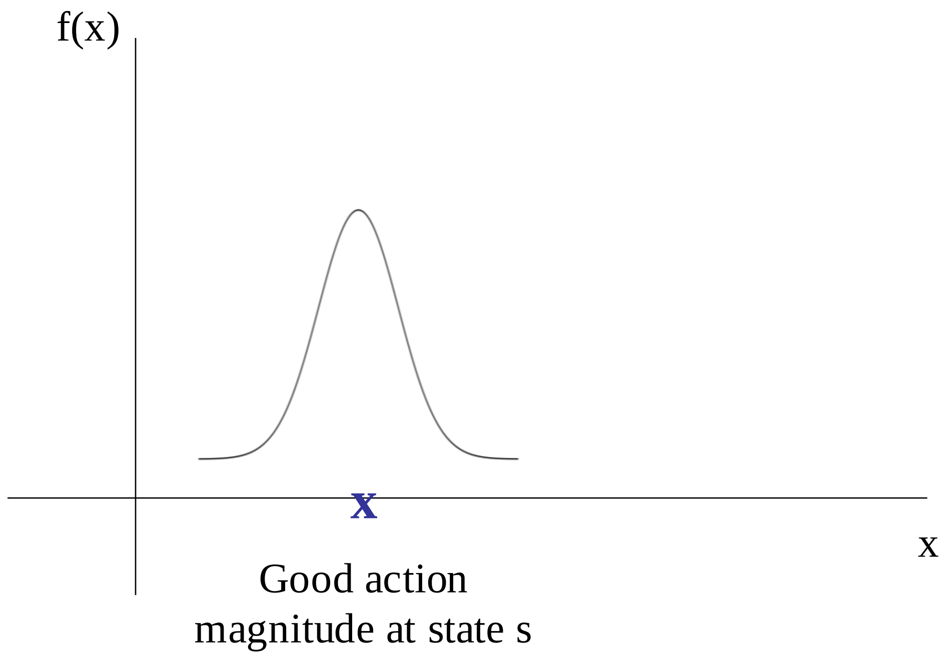
Actor

- During the training, the actor becomes, thanks to the critic, more confident about the action to take at state s (lowers the variance of the distribution)
- The good action magnitude becomes the most probable
- When deployed the mean value is used, no sampling



Actor

- During the training, the actor becomes, thanks to the critic, more confident about the action to take at state s (lowers the variance of the distribution)
- The good action magnitude becomes the most probable
- When deployed the mean value is used, no sampling



Critic

- A critic, which reinforce the actor with Advantage Value function
- How much better or worse an action is than the average available action in a particular state
- A neural network learns the state-value function $\hat{V}$ to estimate A

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$
$$A^{\pi}_{NSTEP}(s_t, a_t) \approx\ r_t + \gamma r_{t+1} + \cdots + \gamma^{n} r_{t+n} + \gamma^{n+1}\hat{V}^{\pi}(s_{t+n+1}) - \hat{V}^{\pi}(s_t)$$

- How good or bad a state is based on the reward obtained

Example: one step Actor-Critic

Critic Update parameters $\boldsymbol{w}$ of $v_{\boldsymbol{w}}$ by TD (e.g., one-step) or MC

Actor Update $\boldsymbol{\theta}$ by policy gradient

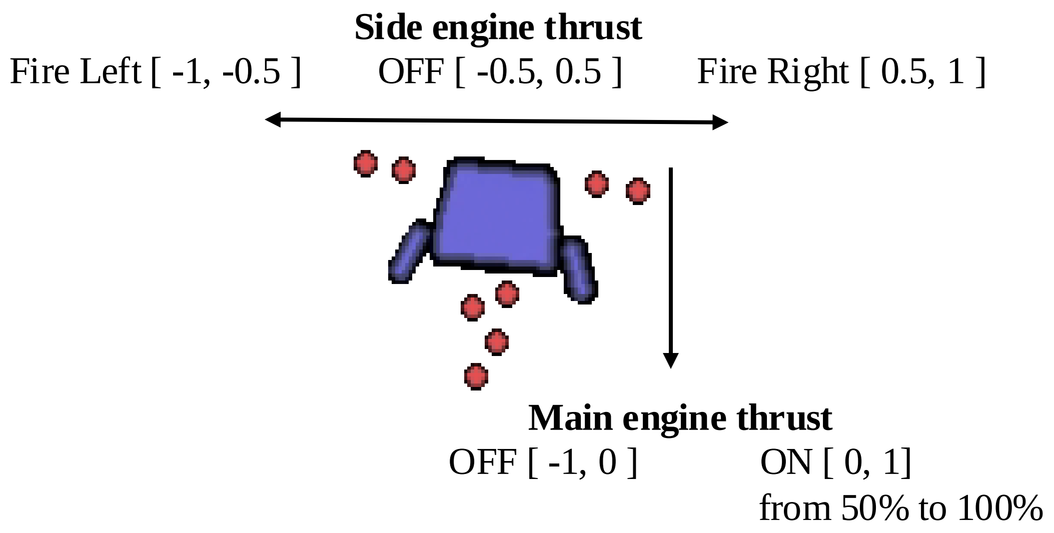
```
**function** ONE-STEP ACTOR CRITIC
    Initialise $s, \boldsymbol{\theta}$
    **for** t = 0, 1, 2, ... **do**
        Sample $A_t \sim \pi_{\boldsymbol{\theta}}(S_t)$
        Sample $R_{t+1}$ and $S_{t+1}$
        $\delta_t = R_{t+1} + \gamma v_{\boldsymbol{w}}(S_{t+1}) - v_{\boldsymbol{w}}(S_t)$        [one-step TD-error, or **advantage**]
        $\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta\, \delta_t\, \nabla_{\boldsymbol{w}} v_{\boldsymbol{w}}(S_t)$        [TD(0)]
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \delta_t\, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t \mid S_t)$        [Policy gradient update (ignoring $\gamma^t$ term)]
```

Proximal Policy Optimization (PPO)

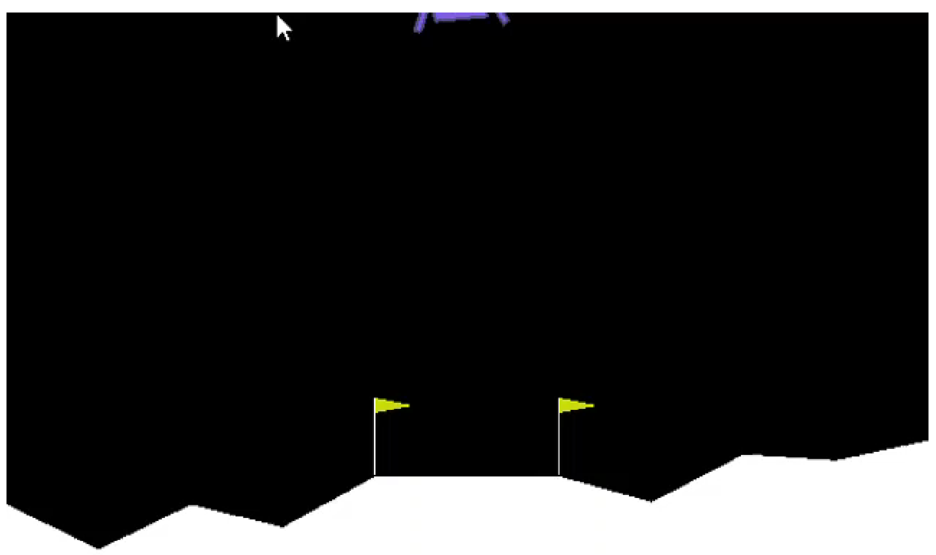
- A2C prone to performance collapse
- Bad policy $\rightarrow$ Bad data …
- Nonisometric mapping: small update in parameter space could result in large update in the policy space
- PPO enforce monotonic policy improvement by defining a surrogate objective

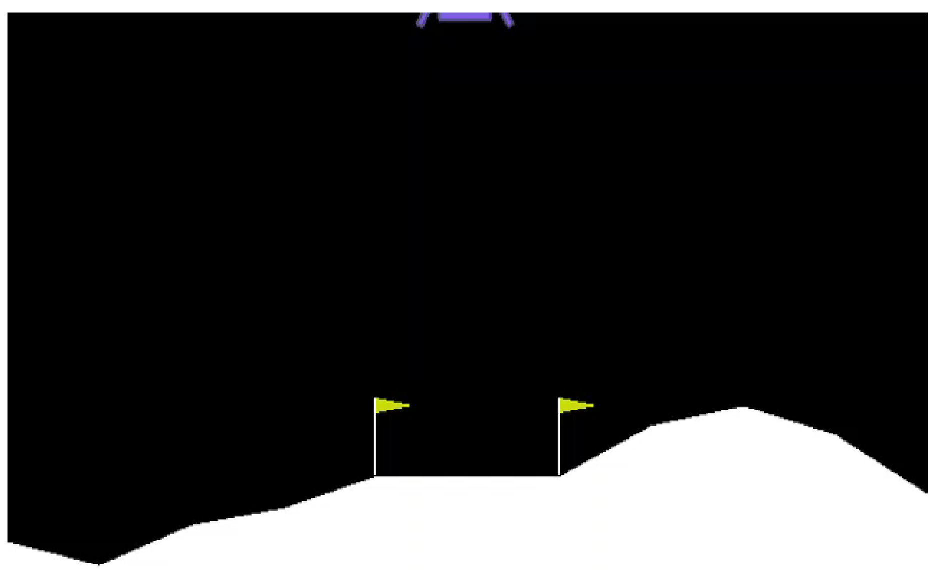
Example: Lunar Lander v2

- OpenAI Gym "LunarLanderContinuous-v2"
- Solved when 200 points are reached
- Observation vector $\left[x, y, \dot{x}, \dot{y}, \theta, \dot{\theta}, c_L, c_R\right]$
- Two possible continuous control inputs



Example: Lunar Lander v2





End of Part 1

Autonomous Drone Flight and Deep Reinforcement Learning

Outline

- Part II

- Background Material and Credits
- Motivation
- Quadrotors
- Learning-based approach
- Isaac Gym
- Hardware: GPUs
- Sim2Real
- Agile Flight

Background Material and Credits – pt. 2

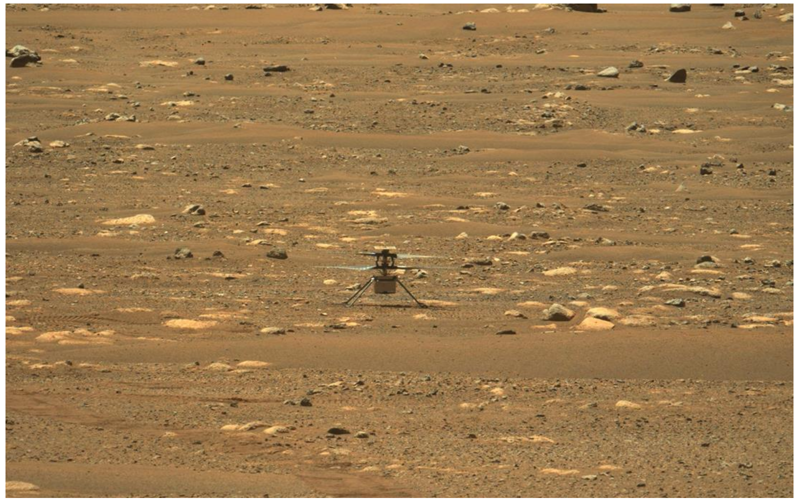
- E. Kaufmann et al., “A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight”
- Fässler M. et al., “Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor”
- Y. Song and D. Scaramuzza, “Policy Search for Model Predictive Control with Application to Agile Drone Flight”
- Nvidia Isaac Gym: Viktor Makoviychuk et al., “Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning”
- Mengozzi S. et al., “Towards Nano-Drones Agile Flight Using Deep Reinforcement Learning”
- Reinforcement Learning library: rl-games

Motivation

- Drones are among the most versatile and agile robots that humans have ever created [Loquercio et al., Science]
- The global commercial market is expected to reach 57.16 billion by 2030 [Commercial Drone Market Size, Share & Trends Report 2030]

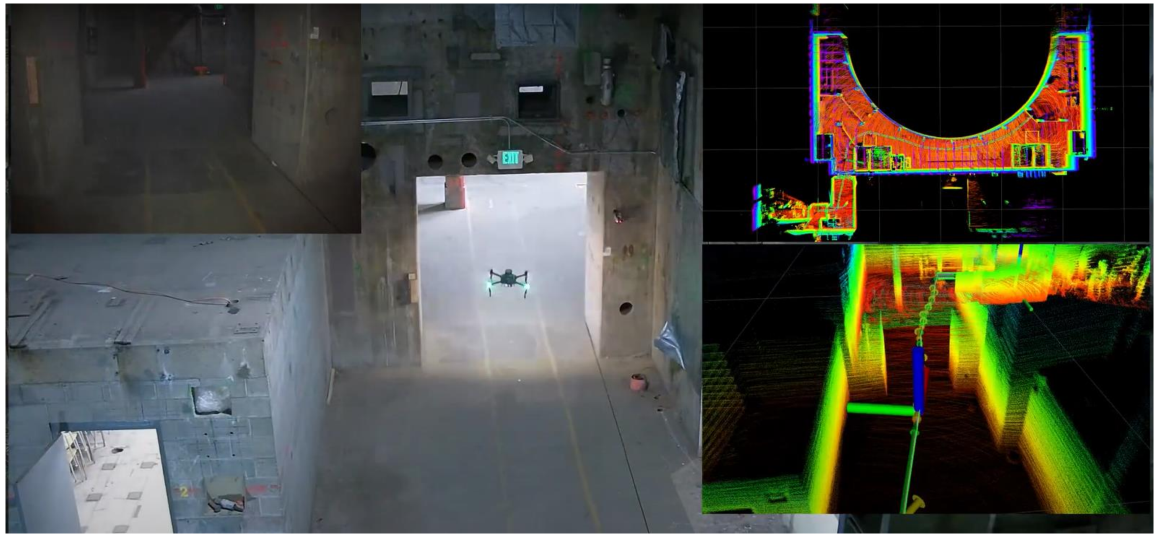
Motivation

- Drones are among the most versatile and agile robots that humans have ever created [Loquercio et al., Science]
- The global commercial market is expected to reach 57.16 billion by 2030 [Commercial Drone Market Size, Share & Trends Report 2030]



Motivation

- Drones are among the most versatile and agile robots that humans have ever created [Loquercio et al., Science]
- The global commercial market is expected to reach 57.16 billion by 2030 [Commercial Drone Market Size, Share & Trends Report 2030]



Motivation

- How is it possible to make an autonomous drone agile in unknown scenarios?
    - Advanced control techniques (e.g., MPC)

- Pros
    - Best performance in nominal case
    - Optimality and safety guarantees
- Cons
    - Not robust for real unknown scenarios
    - Resource hungry
    - The solution could have induced biases and heuristics (e.g., when designing a cost function)

Motivation

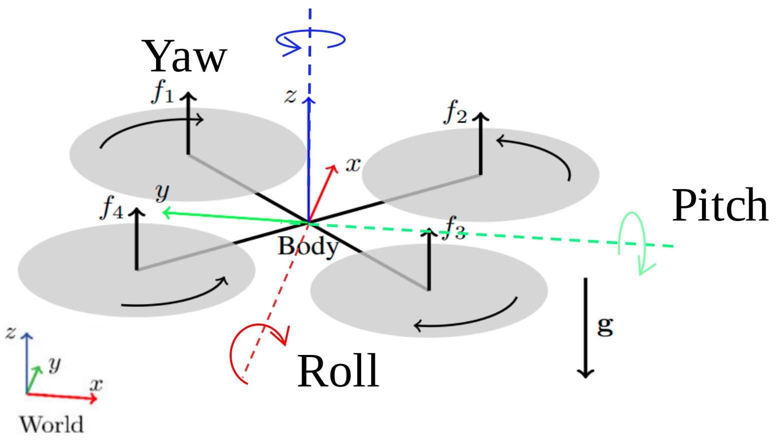
- How is it possible to make an autonomous drone agile in unknown scenarios?
  - Learning-based approach (e.g., DRL)

- Pros
  - Good performances in nominal and unknown scenarios (randomization)
  - Policy can explore and exploit the maximum dynamical capabilities of the platform
  - The policy is an emerging behavior, not specified a priori (reward function)
  - Resource efficiency (NNs inference)
- Cons
  - The policy could find unfeasible solutions in real-world (sim2real)
  - Explainability

Motivation

- Autonomy is at the core of these applications
- Deep Reinforcement Learning (DRL): learning to solve problems with no analytic formulation via trial and error
- Quadrotors: demanding benchmark for testing robotic autonomy
    - Unstable and highly nonlinear dynamics
    - Avoid dynamic obstacles in unconstrained environments (e.g., cities)
    - Mitigate low battery duration and constrained computing

Quadrotors



- Model
  - Rigid body in 3D space: 6 DOFs
  - Controlled by four rotor thrusts f_i:

$$mc = f_1 + f_2 + f_3 + f_4$$

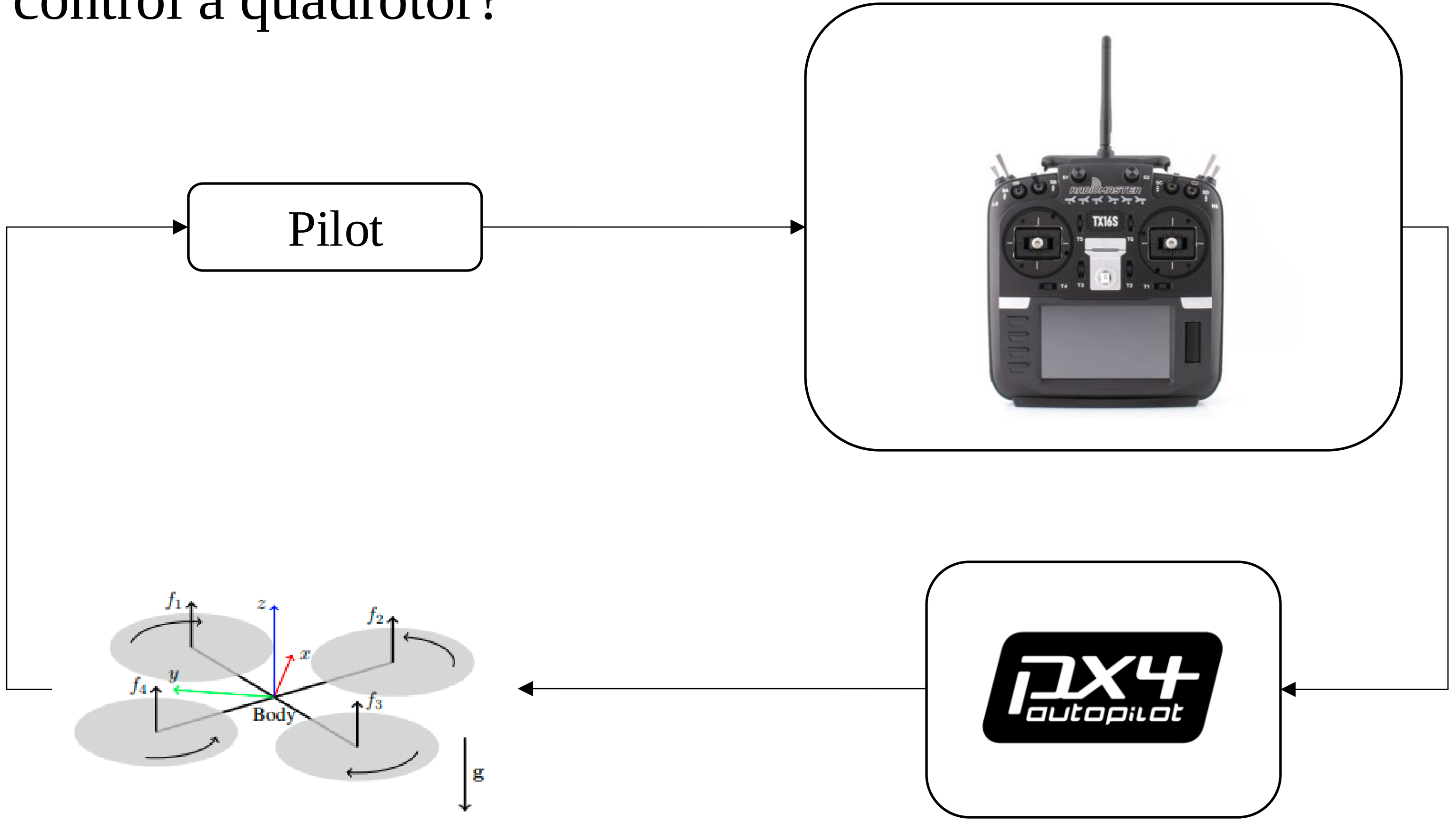
  - c: total mass normalized force, or lift, applied in the center of mass
  - τ: total torque

$$\tau = \begin{bmatrix} \frac{\sqrt{2}}{2} l(f_1 - f_2 - f_3 + f_4) \\ \frac{\sqrt{2}}{2} l(-f_1 - f_2 + f_3 + f_4) \\ k(f_1 - f_2 + f_3 - f_4) \end{bmatrix}$$

  - m: mass, l: arm length, k: rotor-torque coefficient

Quadrotors

- How do we control a quadrotor?

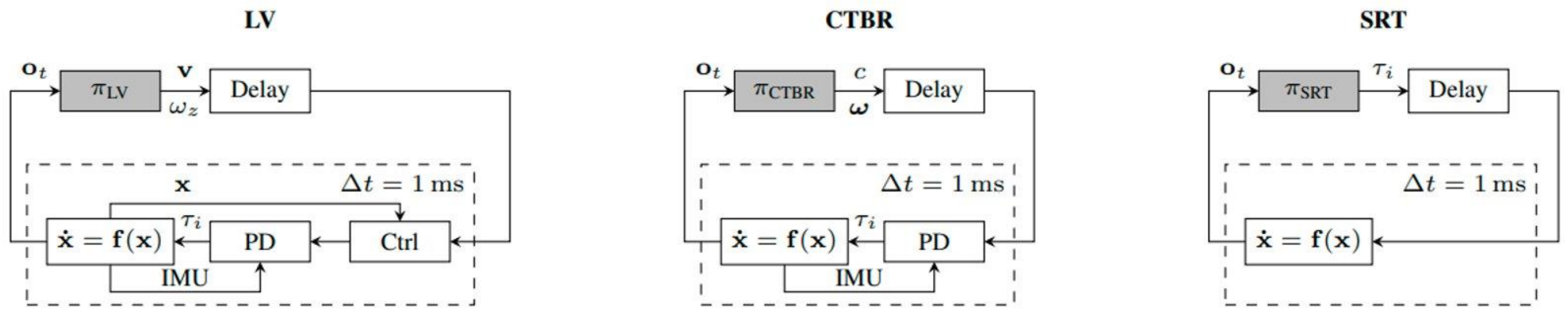


Quadrotors

- How do we control a quadrotor?
- Three possible methods:
  - Linear velocity control (LV)
  - Common thrust and body rates (CTBR): drone racing pilot's mode
  - Single rotor thrust (SRT)

Quadrotors

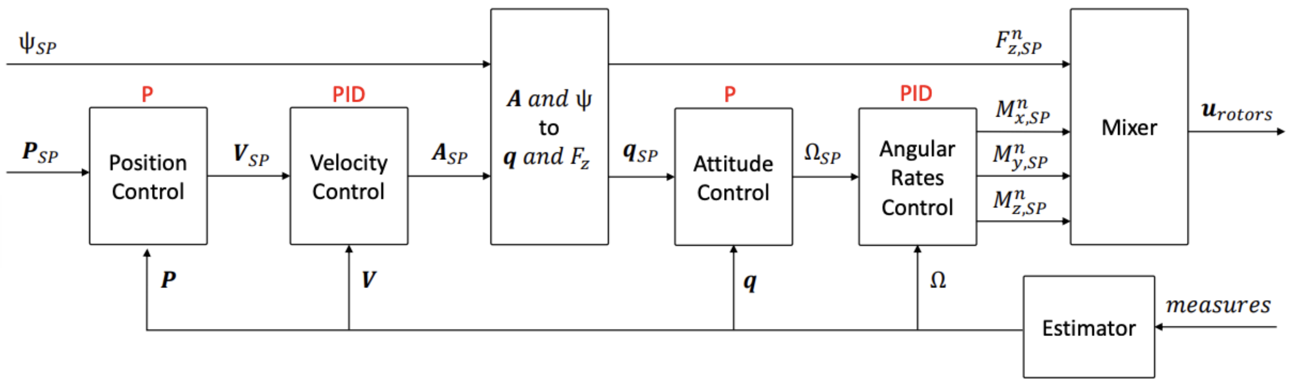
- How do we control a quadrotor?
- Three possible methods:
  - Linear velocity control (LV)
  - Common thrust and body rates (CTBR): drone racing pilot's mode
  - Single rotor thrust (SRT)

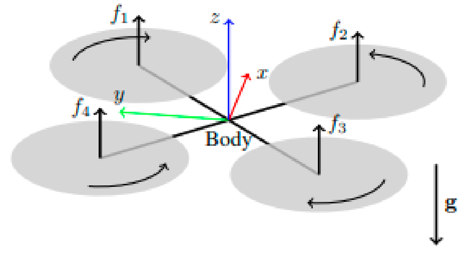


Higher level of abstraction

Quadrotors

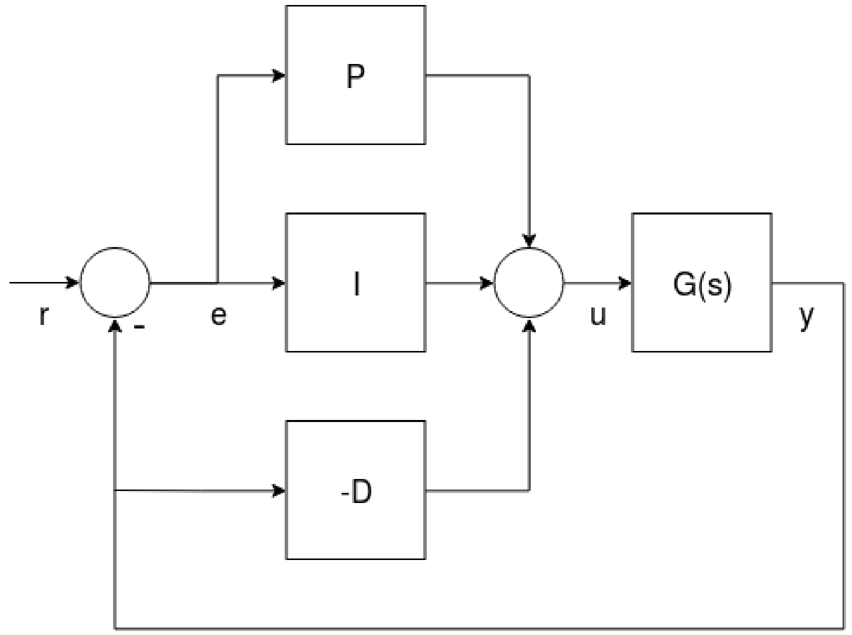
- How do we control a quadrotor?
- Three possible methods:
  - Linear velocity control (LV)
  - Common thrust and body rates (CTBR): drone racing pilot's mode
  - Single rotor thrust (SRT)
- PX4 cascade control loops

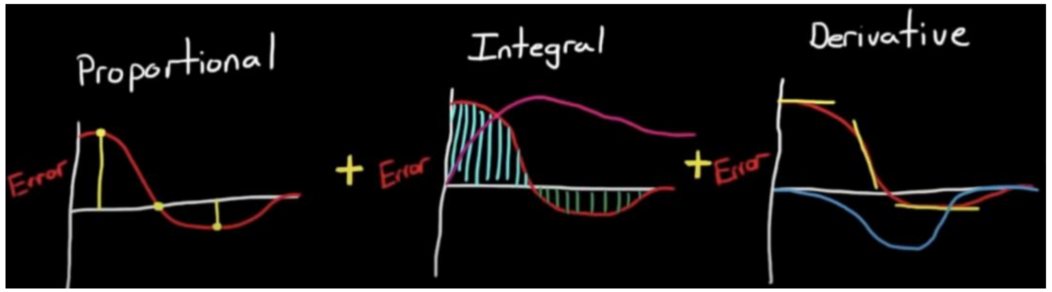




Quadrotors

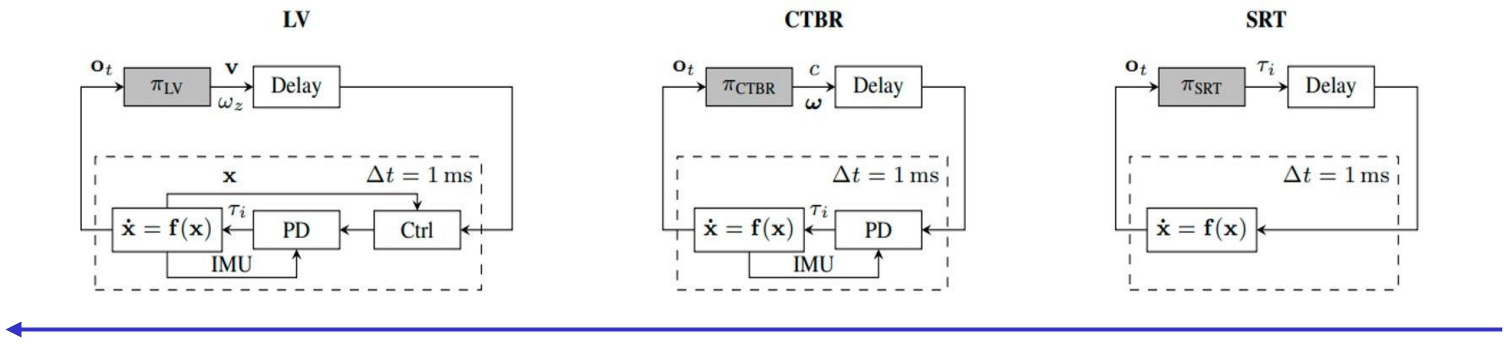
- Proportional Integral Derivative controller
  - Brian Douglass, PID control, a brief introduction





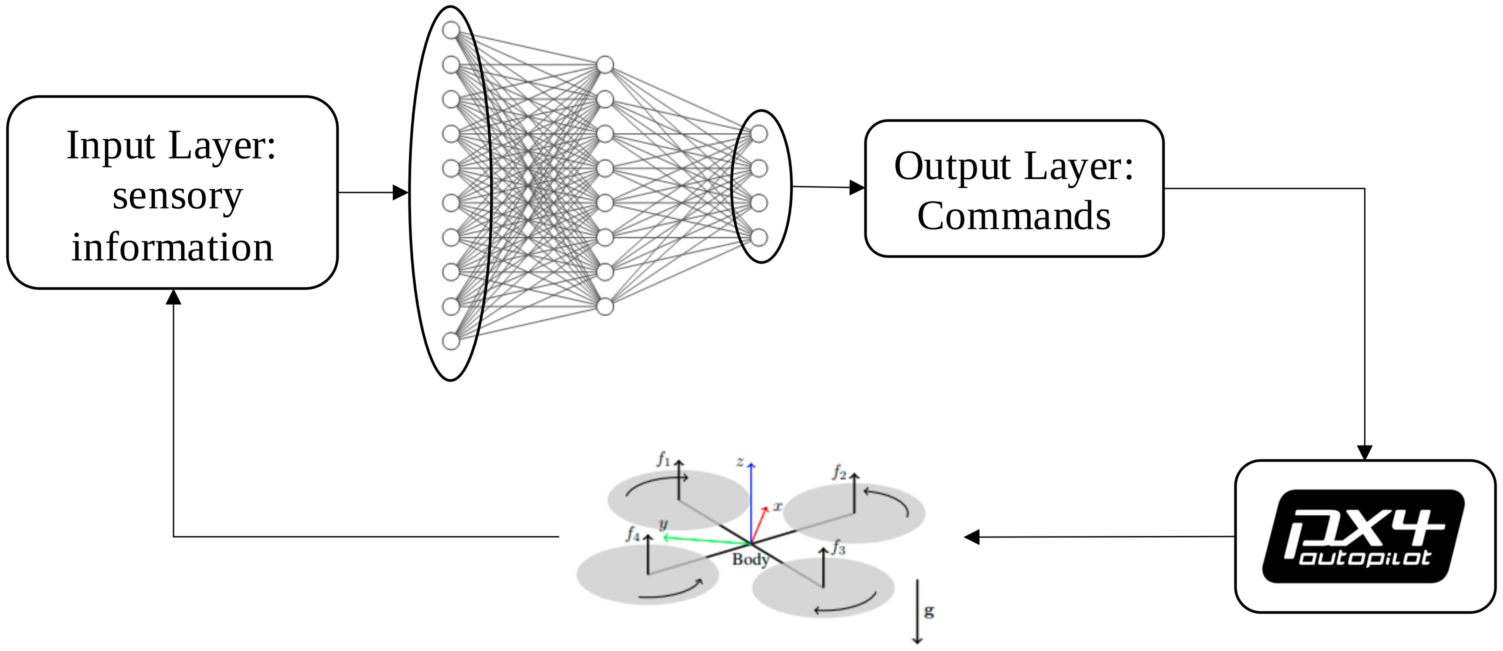
Abstraction level trade-off

- Higher abstraction:
  - The input is «filtered» by the low-level controller
  - Safer command input
  - Cannot exploit the maximum dynamic performances
- Low abstraction:
  - Input directly to the drone motors without «filtering»
  - A wrong input could have irreversible consequences
  - Exploits maximum the dynamic performances



Learning-based Approach

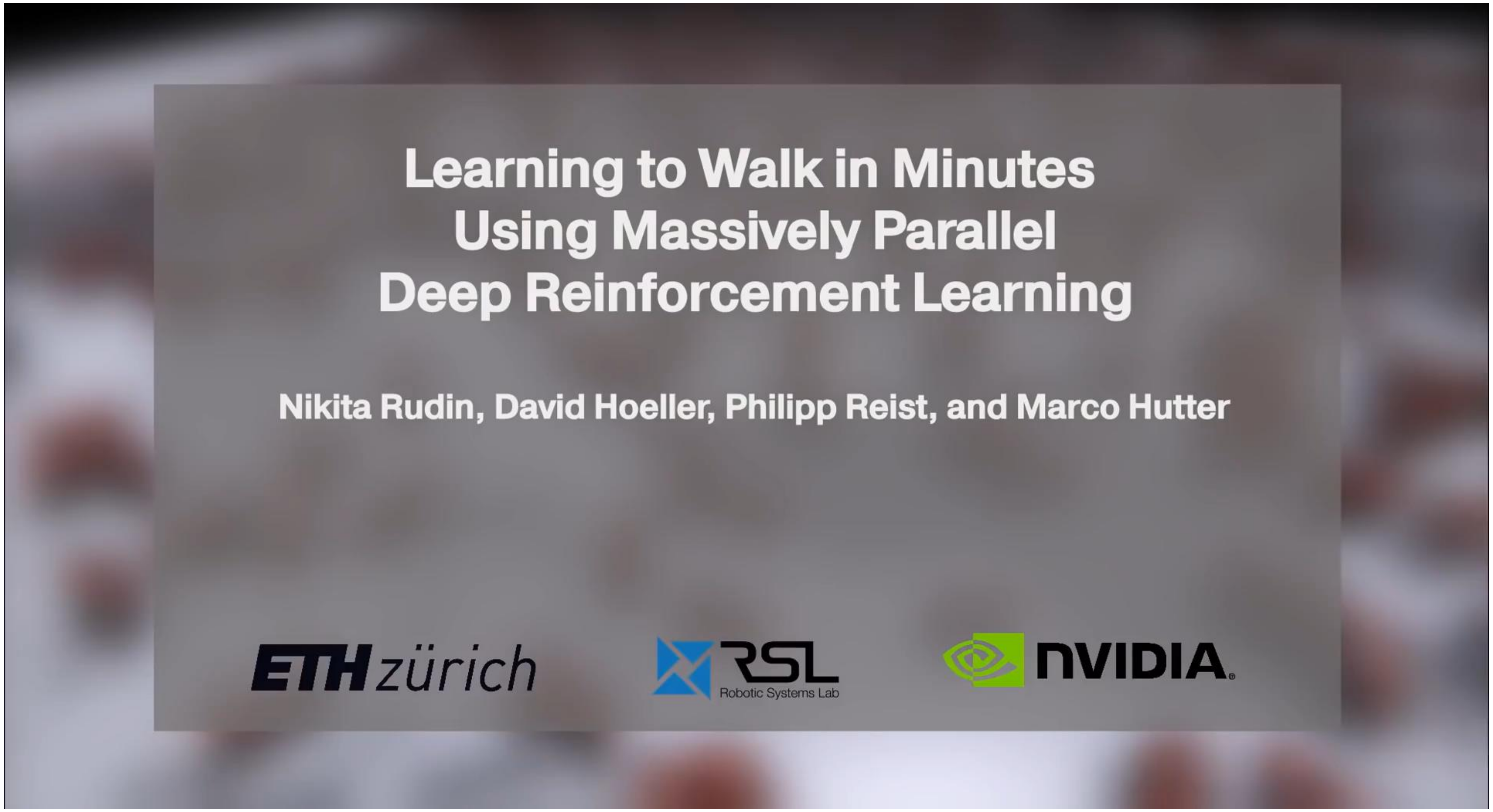
- Substituting the pilot with a Neural Network
- One neuron correspond to one number:
  - position [x, y, z], velocity [vx, vy, vz], acceleration, orientation, …
- Commands: any of the 3 modes (LV, CTBR, SRT)



Learning-based Approach

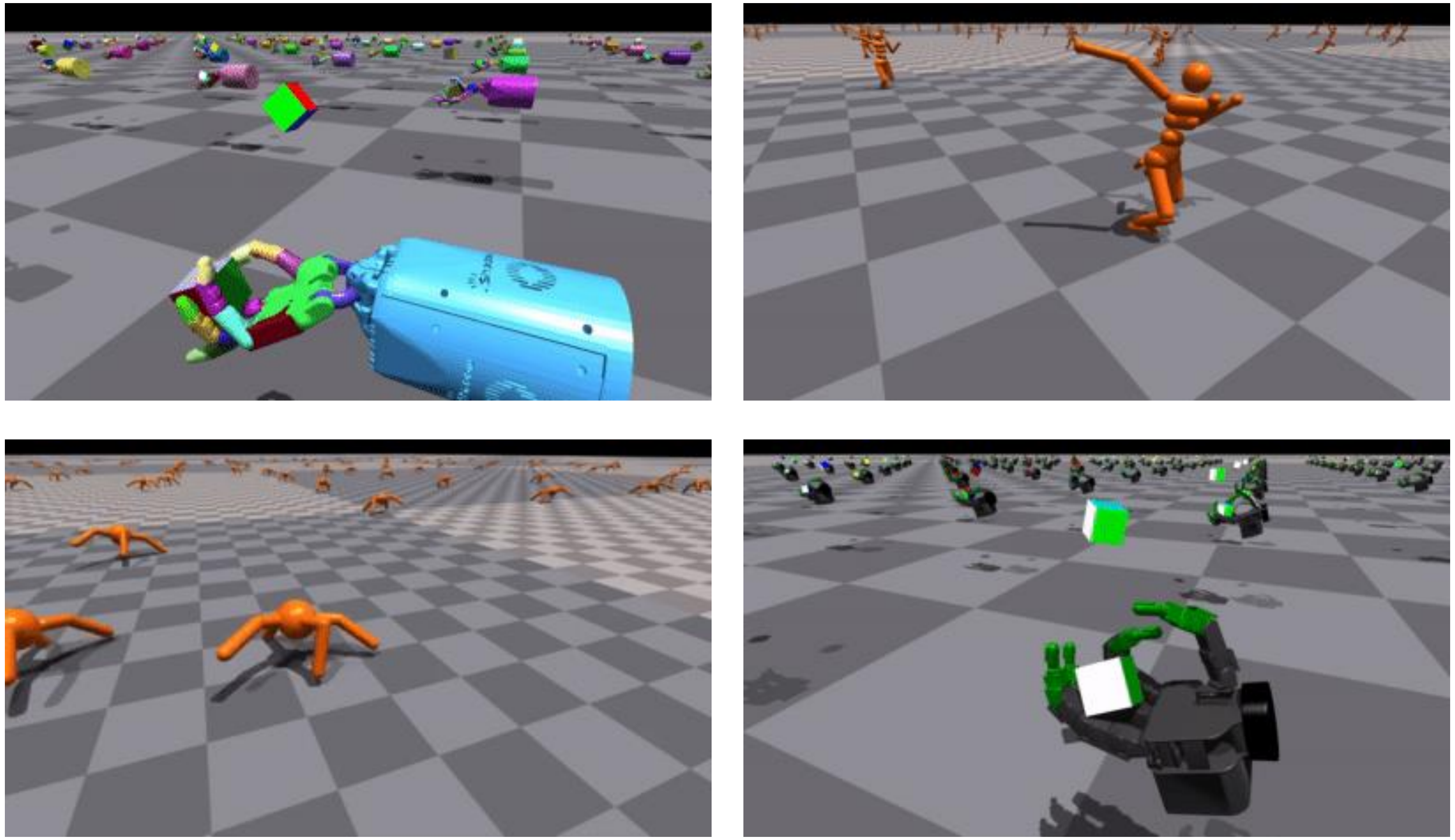
- We need three components:
  - A Deep Reinforcement Learning algorithm to train the NN
  - A high-fidelity simulation optimized for RL
  - Optimized hardware for parallelization (gather the data faster)

Isaac Gym


Learning to Walk in Minutes
Using Massively Parallel
Deep Reinforcement Learning

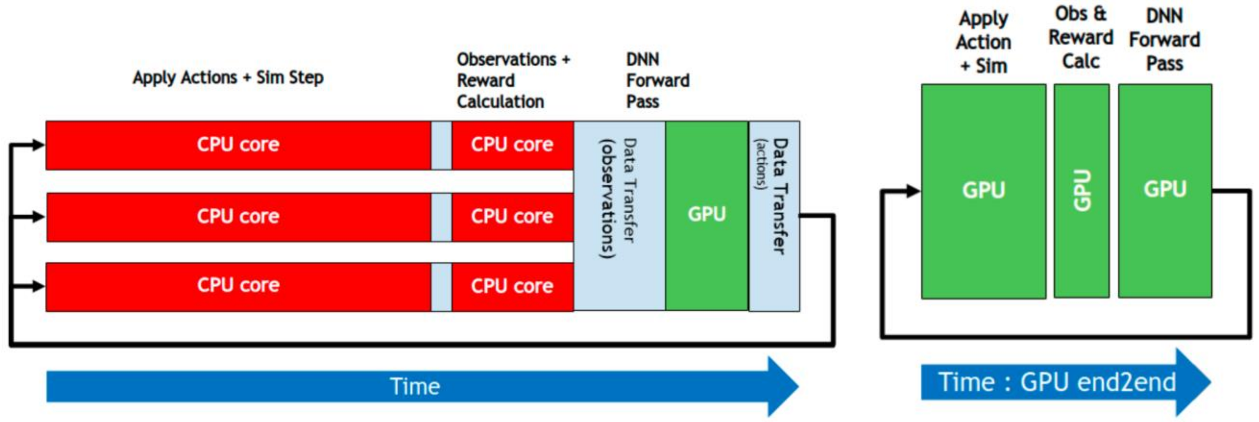
Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter

Isaac Gym



Isaac Gym

- Nvidia simulation environment for Reinforcement Learning
- End-to-end GPU simulation
- Using PyTorch and APIs defines:
  - Assets: agents, obstacles, …
  - Environment: lights, textures, spawning coordinates, interaction forces, contacts
  - Camera sensors, force sensors, …
  - Physical interaction: friction, elasticity, gravity



Isaac Gym

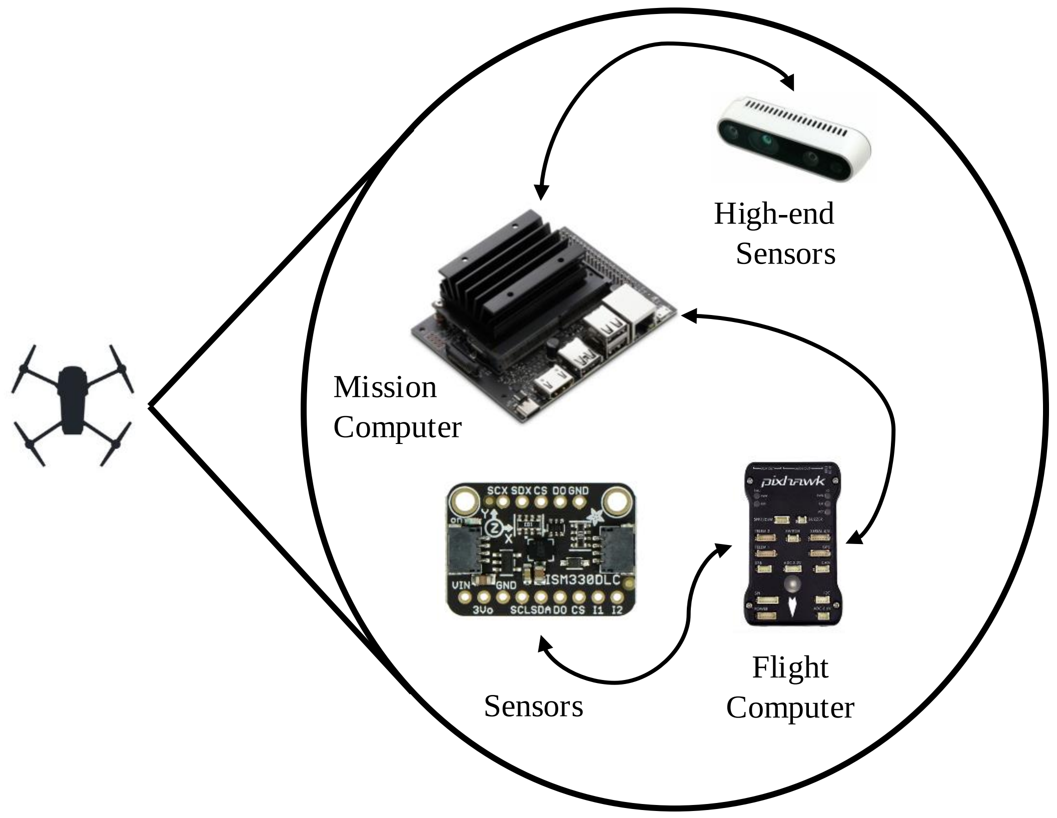
- Isaac Gym simulation loop:
  - Actions (S_t) → Forces and Moments
  - Fed into the simulator: physics update of a time step dt
  - New environment State is returned → new Observations and Reward
  - Fed into the Policy → Actions (S_{t+1})
  - Loop

Hardware: GPUs

- GPU end-to-end pipeline
  - GPUs accelerators for parallel operations (e.g., NN inference)
  - Simulate thousands of parallel environments
  - Gather huge amount of data: sample inefficiency

Sim2Real

- After training, deploy the policy on a real-world platform

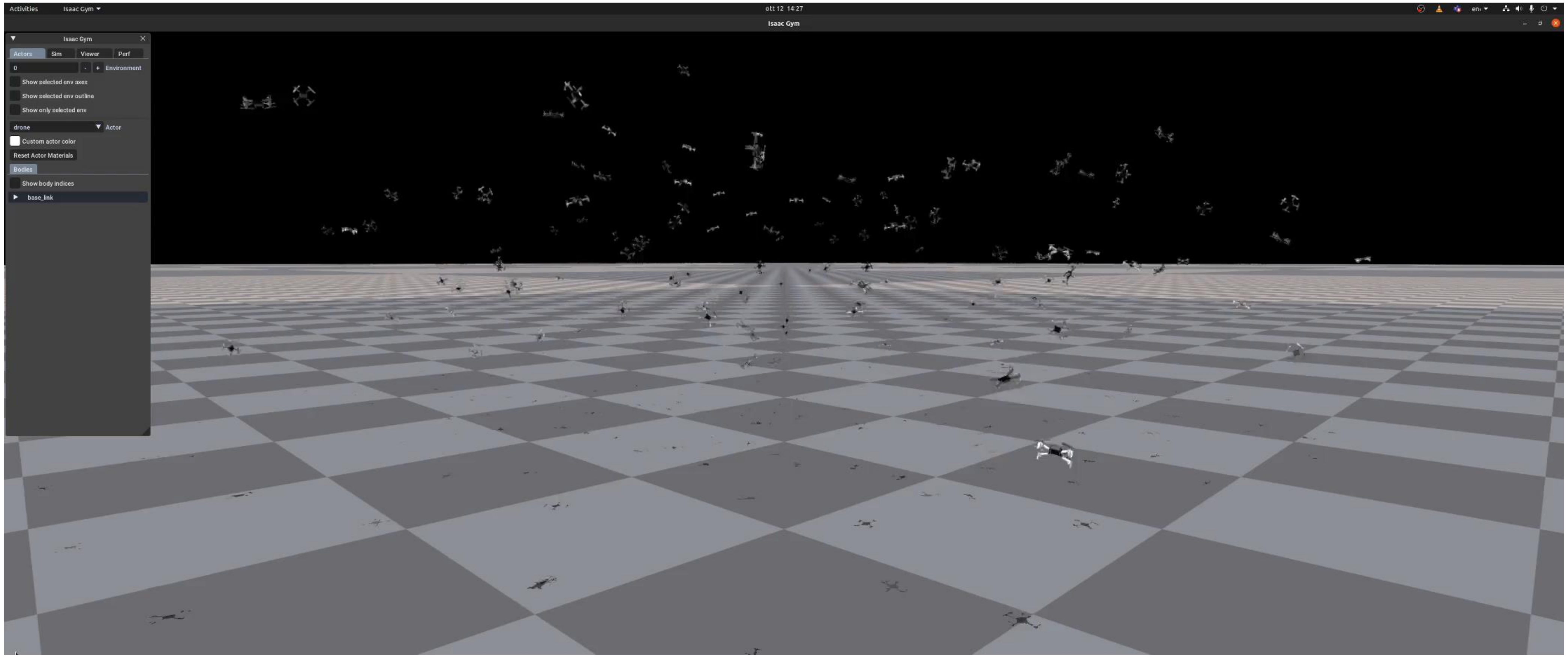


Sim2Real

- After training, deploy the policy on a real-world platform
- Simulation is just an approximation of the real world: *simulation-reality gap*
  - Noise
  - Reflection in images (photorealistic simulators)
  - Different actuation dynamics
  - Air drag
- Make the policy *robust* to uncertainties

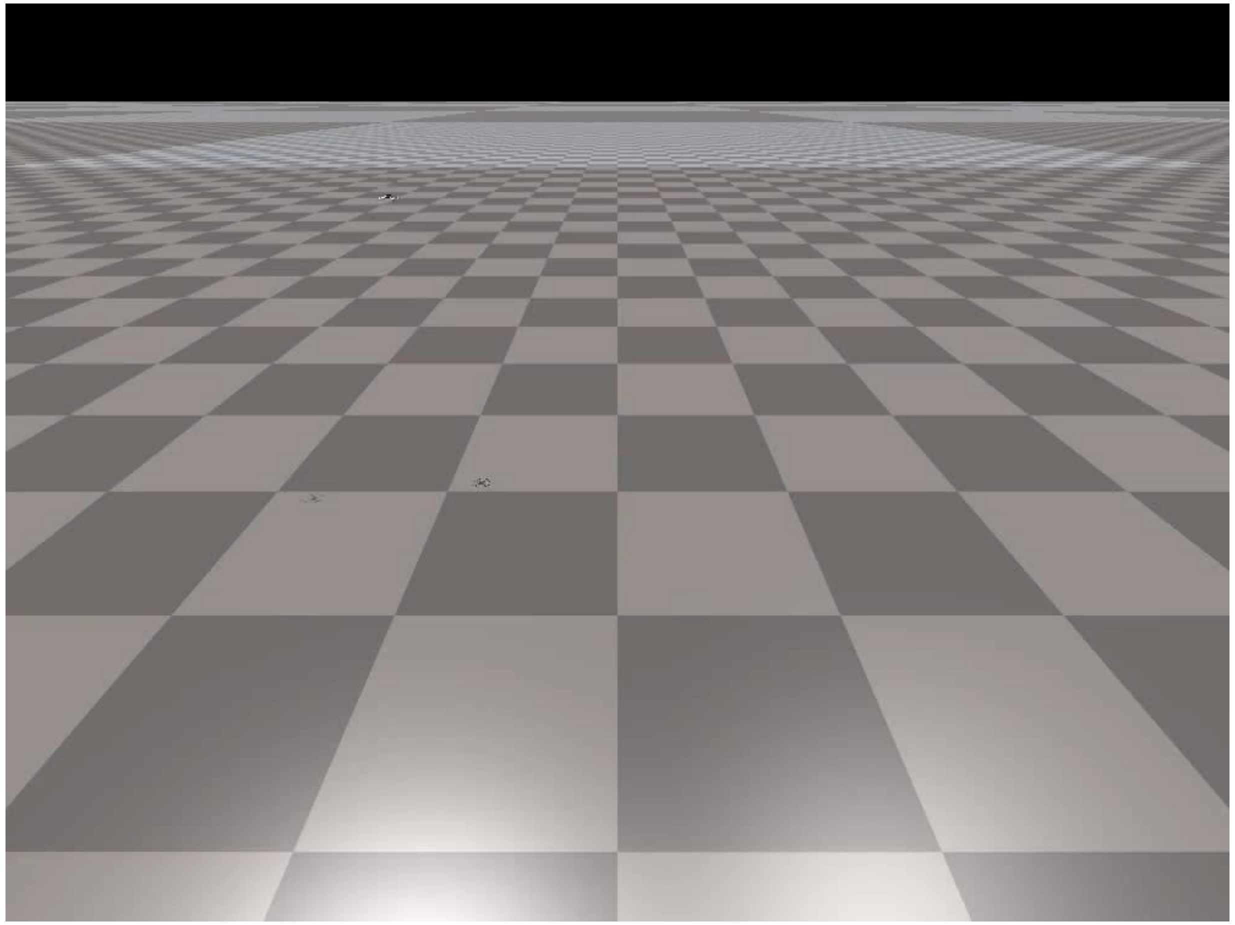
Sim2Real

- Parallel training for hovering



Sim2Real

- Testing the learned policy

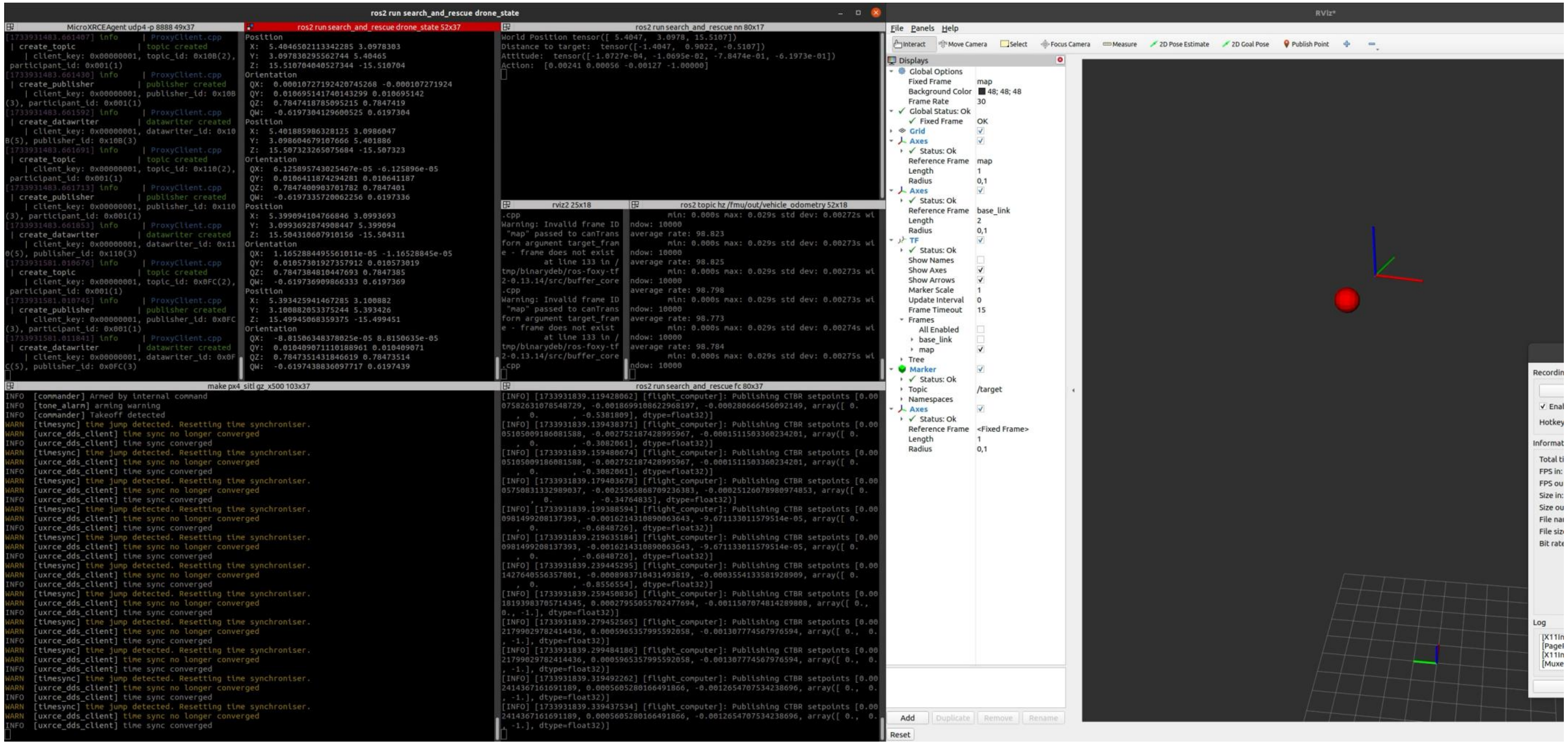


Sim2Real

- Software in the loop (SITL)
  - The actual flight controller software runs together with the simulation
  - Closer step to reality, smaller simulation-reality gap
  - Avoid fatal crashes

Sim2Real

- Software in the loop (SITL)



MicroXRCEAgent udp4 -p 8888 49x37

```
[1733931483.661407] info     | ProxyClient.cpp
 | create_topic             | topic created
   | client_key: 0x00000001, topic_id: 0x10B(2),
 participant_id: 0x001(1)
[1733931483.661430] info     | ProxyClient.cpp
 | create_publisher         | publisher created
   | client_key: 0x00000001, publisher_id: 0x10B
(3), participant_id: 0x001(1)
[1733931483.661592] info     | ProxyClient.cpp
 | create_datawriter        | datawriter created
   | client_key: 0x00000001, datawriter_id: 0x10
B(5), publisher_id: 0x10B(3)
[1733931483.661691] info     | ProxyClient.cpp
 | create_topic             | topic created
   | client_key: 0x00000001, topic_id: 0x110(2),
 participant_id: 0x001(1)
[1733931483.661713] info     | ProxyClient.cpp
 | create_publisher         | publisher created
   | client_key: 0x00000001, publisher_id: 0x110
(3), participant_id: 0x001(1)
[1733931483.661853] info     | ProxyClient.cpp
 | create_datawriter        | datawriter created
   | client_key: 0x00000001, datawriter_id: 0x11
0(5), publisher_id: 0x110(3)
[1733931581.010676] info     | ProxyClient.cpp
 | create_topic             | topic created
   | client_key: 0x00000001, topic_id: 0x0FC(2),
 participant_id: 0x001(1)
[1733931581.010745] info     | ProxyClient.cpp
 | create_publisher         | publisher created
   | client_key: 0x00000001, publisher_id: 0x0FC
(3), participant_id: 0x001(1)
[1733931581.011841] info     | ProxyClient.cpp
 | create_datawriter        | datawriter created
   | client_key: 0x00000001, datawriter_id: 0x0F
C(5), publisher_id: 0x0FC(3)
```

ros2 run search_and_rescue drone_state 52x37

```
Position
 X:  5.4046502113342285 3.0978303
 Y:  3.097830295562744 5.40465
 Z:  15.510704040527344 -15.510704
Orientation
 QX:  0.00010727192420745268 -0.000107271924
 QY:  0.010695141740143299 0.010695142
 QZ:  0.7847418785095215 0.7847419
 QW:  -0.6197304129600525 0.6197304
Position
 X:  5.401885986328125 3.0986047
 Y:  3.098604679107666 5.401886
 Z:  15.507323265075684 -15.507323
Orientation
 QX:  6.125895743025467e-05 -6.125896e-05
 QY:  0.0106411874294281 0.010641187
 QZ:  0.7847400903701782 0.7847401
 QW:  -0.6197335720062256 0.6197336
Position
 X:  5.399094104766846 3.0993693
 Y:  3.0993692874908447 5.399094
 Z:  15.504310607910156 -15.504311
Orientation
 QX:  1.1652884495561011e-05 -1.16528845e-05
 QY:  0.01057301927357912 0.010573019
 QZ:  0.7847384810447693 0.7847385
 QW:  -0.619736909866333 0.6197369
Position
 X:  5.393425941467285 3.100882
 Y:  3.100882053375244 5.393426
 Z:  15.49945068359375 -15.499451
Orientation
 QX:  -8.81506348378025e-05 8.8150635e-05
 QY:  0.010409071110188961 0.010409071
 QZ:  0.7847351431846619 0.78473514
 QW:  -0.6197438836097717 0.6197439
```

ros2 run search_and_rescue nn 80x17

```
World Position tensor([ 5.4047,  3.0978, 15.5107])
Distance to target:  tensor([-1.4047,  0.9022, -0.5107])
Attitude:  tensor([-1.0727e-04, -1.0695e-02, -7.8474e-01, -6.1973e-01])
Action:  [0.00241 0.00056 -0.00127 -1.00000]
```

rviz2 25x18

```
.cpp
Warning: Invalid frame ID
 "map" passed to canTrans
form argument target_fram
e - frame does not exist
         at line 133 in /
tmp/binarydeb/ros-foxy-tf
2-0.13.14/src/buffer_core
.cpp
Warning: Invalid frame ID
 "map" passed to canTrans
form argument target_fram
e - frame does not exist
         at line 133 in /
tmp/binarydeb/ros-foxy-tf
2-0.13.14/src/buffer_core
.cpp
```

ros2 topic hz /fmu/out/vehicle_odometry 52x18

```
        min: 0.000s max: 0.029s std dev: 0.00272s wi
ndow: 10000
average rate: 98.823
        min: 0.000s max: 0.029s std dev: 0.00273s wi
ndow: 10000
average rate: 98.825
        min: 0.000s max: 0.029s std dev: 0.00273s wi
ndow: 10000
average rate: 98.798
        min: 0.000s max: 0.029s std dev: 0.00273s wi
ndow: 10000
average rate: 98.773
        min: 0.000s max: 0.029s std dev: 0.00274s wi
ndow: 10000
average rate: 98.784
        min: 0.000s max: 0.029s std dev: 0.00275s wi
ndow: 10000
```

make px4_sitl gz_x500 103x37

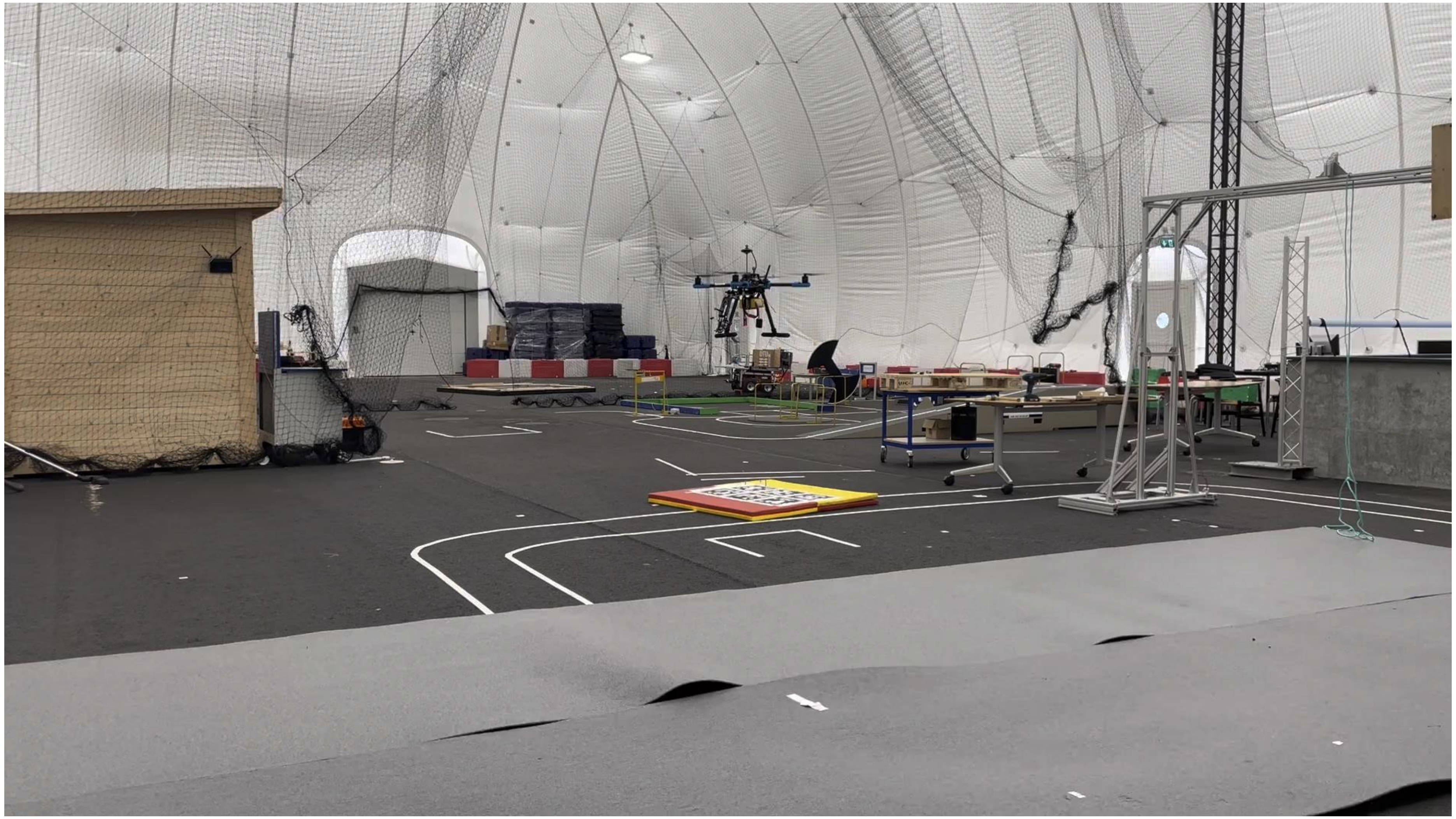
```
INFO  [commander] Armed by internal command
INFO  [tone_alarm] arming warning
INFO  [commander] Takeoff detected
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
WARN  [timesync] time jump detected. Resetting time synchroniser.
WARN  [uxrce_dds_client] time sync no longer converged
INFO  [uxrce_dds_client] time sync converged
```

ros2 run search_and_rescue fc 80x37

```
[INFO] [1733931839.119428062] [flight_computer]: Publishing CTBR setpoints [0.00
07582631078548729, -0.0018699108622968197, -0.000280666456092149, array([ 0.
  ,  0.        , -0.5381809], dtype=float32)]
[INFO] [1733931839.139438371] [flight_computer]: Publishing CTBR setpoints [0.00
05105009186081588, -0.002752187428995967, -0.0001511503360234201, array([ 0.
  ,  0.        , -0.3082061], dtype=float32)]
[INFO] [1733931839.159480674] [flight_computer]: Publishing CTBR setpoints [0.00
05105009186081588, -0.002752187428995967, -0.0001511503360234201, array([ 0.
  ,  0.        , -0.3082061], dtype=float32)]
[INFO] [1733931839.179403678] [flight_computer]: Publishing CTBR setpoints [0.00
05750831332989037, -0.0025565868709236383, -0.00025126078980974853, array([ 0.
      ,  0.        , -0.34764835], dtype=float32)]
[INFO] [1733931839.199388594] [flight_computer]: Publishing CTBR setpoints [0.00
0981499208137393, -0.0016214310890063643, -9.671133011579514e-05, array([ 0.
  ,  0.        , -0.6848726], dtype=float32)]
[INFO] [1733931839.219635184] [flight_computer]: Publishing CTBR setpoints [0.00
0981499208137393, -0.0016214310890063643, -9.671133011579514e-05, array([ 0.
  ,  0.        , -0.6848726], dtype=float32)]
[INFO] [1733931839.239445295] [flight_computer]: Publishing CTBR setpoints [0.00
1427640556357801, -0.0008983710431493819, -0.0003554133581928909, array([ 0.
  ,  0.        , -0.8556554], dtype=float32)]
[INFO] [1733931839.259450836] [flight_computer]: Publishing CTBR setpoints [0.00
18193983705714345, 0.00027955055702477694, -0.0011507074814289808, array([ 0.,
0., -1.], dtype=float32)]
[INFO] [1733931839.279452565] [flight_computer]: Publishing CTBR setpoints [0.00
21799029782414436, 0.0005965357995592058, -0.001307774567976594, array([ 0.,  0.
, -1.], dtype=float32)]
[INFO] [1733931839.299484186] [flight_computer]: Publishing CTBR setpoints [0.00
21799029782414436, 0.0005965357995592058, -0.001307774567976594, array([ 0.,  0.
, -1.], dtype=float32)]
[INFO] [1733931839.319492262] [flight_computer]: Publishing CTBR setpoints [0.00
2414367161691189, 0.0005605280166491866, -0.0012654707534238696, array([ 0.,  0.
, -1.], dtype=float32)]
[INFO] [1733931839.339437534] [flight_computer]: Publishing CTBR setpoints [0.00
2414367161691189, 0.0005605280166491866, -0.0012654707534238696, array([ 0.,  0.
, -1.], dtype=float32)]
```

Global Options	
Fixed Frame	map
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
Axes	✓
✓ Status: Ok	
Reference Frame	map
Length	1
Radius	0,1
Axes	✓
✓ Status: Ok	
Reference Frame	base_link
Length	2
Radius	0,1
TF	✓
✓ Status: Ok	
Show Names	☐
Show Axes	✓
Show Arrows	✓
Marker Scale	1
Update Interval	0
Frame Timeout	15
Frames	
All Enabled	☐
base_link	☐
map	✓
Tree	
Marker	✓
✓ Status: Ok	
Topic	/target
Namespaces	
Axes	✓
✓ Status: Ok	
Reference Frame	<Fixed Frame>
Length	1
Radius	0,1

Sim2Real

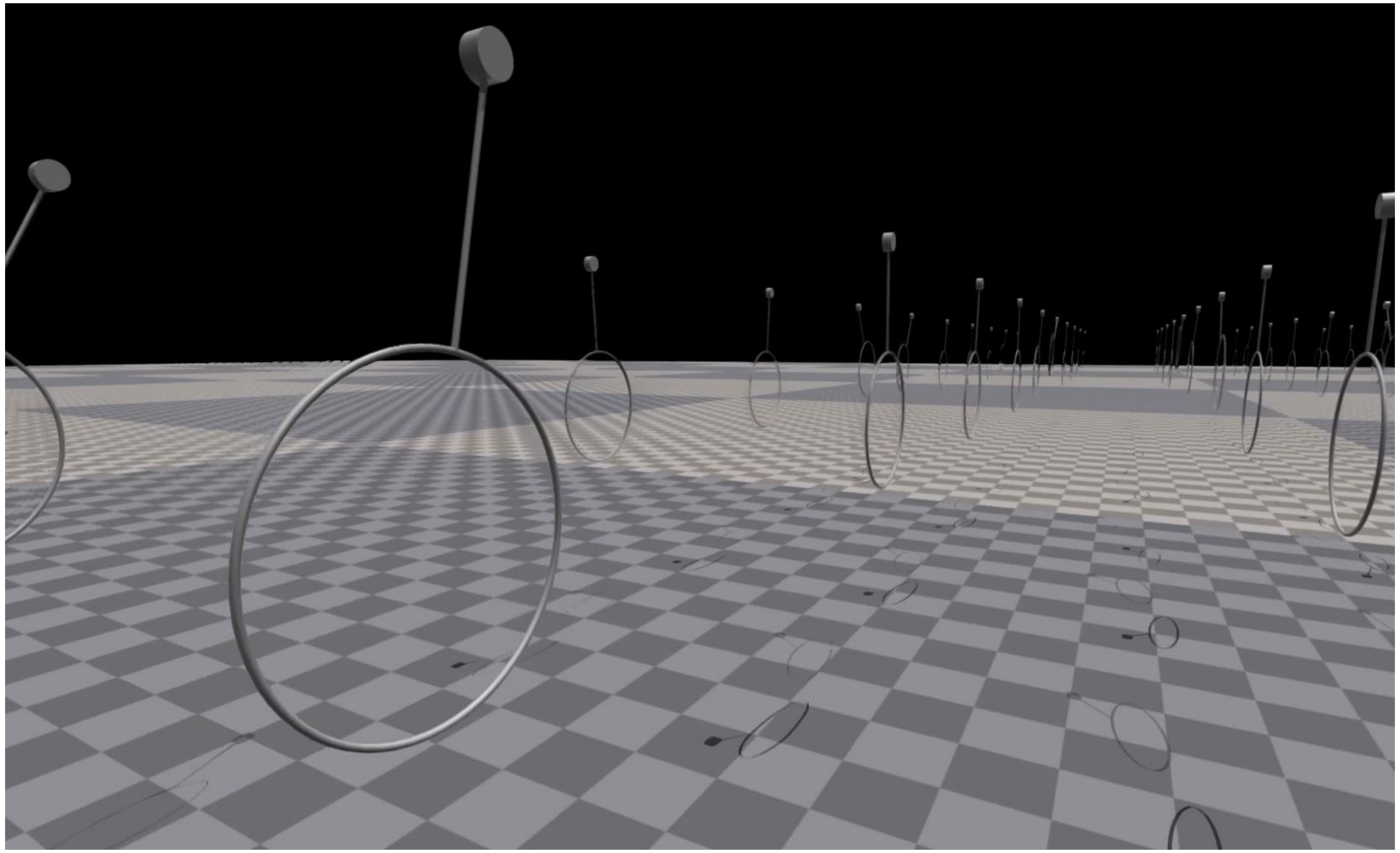


Agile Flight

- Agility: makes the drones fly faster
  - Faster mission execution (e.g., search and rescue)
  - Avoid dynamic obstacles
  - Mitigate limited battery life
  - More complex algorithms

Agile Flight

- Testing the learned agile policy



End of Part 2

Thank you.

Questions?

-

Contacts:
sebastiano.mengozzi@unibo.it