ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA
DEPARTMENT OF ELECTRICAL, ELECTRONIC AND INFORMATION ENGINEERING
MASTER'S DEGREE IN AUTOMATION ENGINEERING

A Deep Reinforcement Learning approach based on policy gradient for Mobile Robot Navigation

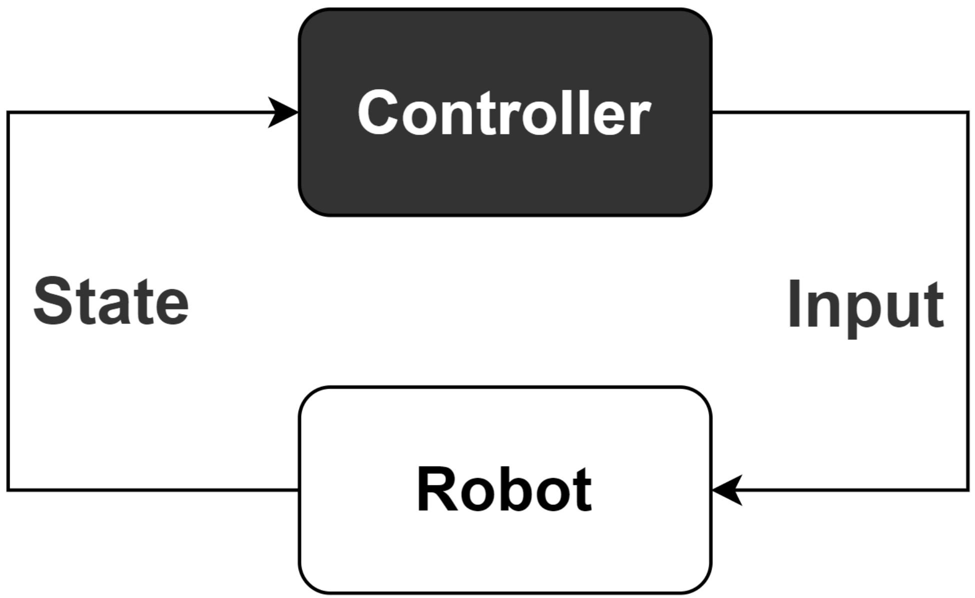
Candidate:
Enrico Pianazzi

Advisor:
Prof. Giuseppe Notarstefano

Co-Advisors:
Dott. Andrea Camisa
Ing. Lorenzo Sforni

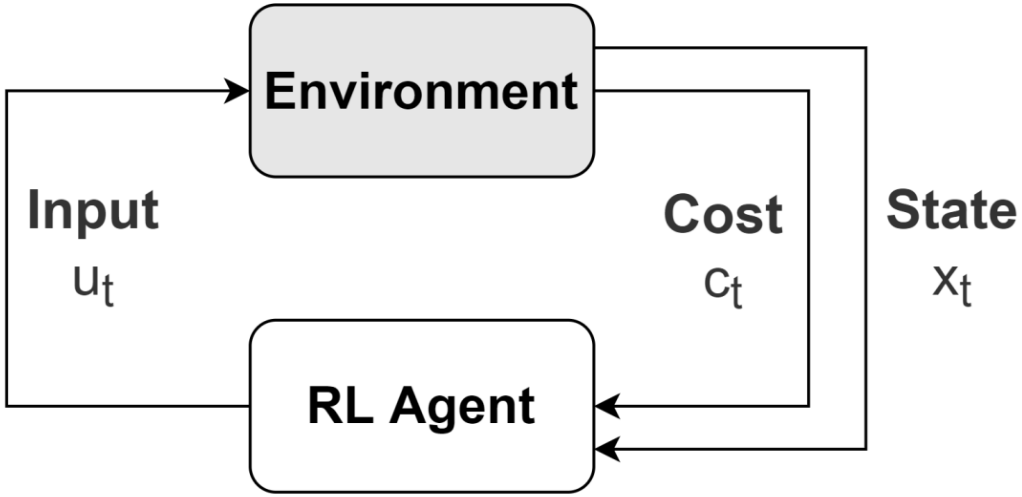
Motivations

- Autonomous navigation features in mobile robots are highly valuable
- Reinforcement Learning can achieve complex data-driven control
- State feedback control without the need of complex sensory data



Reinforcement Learning

- Optimize a policy (controller) $\mu(x_t) = u_t$ to minimize future cost
- Only experience cost and state signals
- Division in discrete time steps and episodes



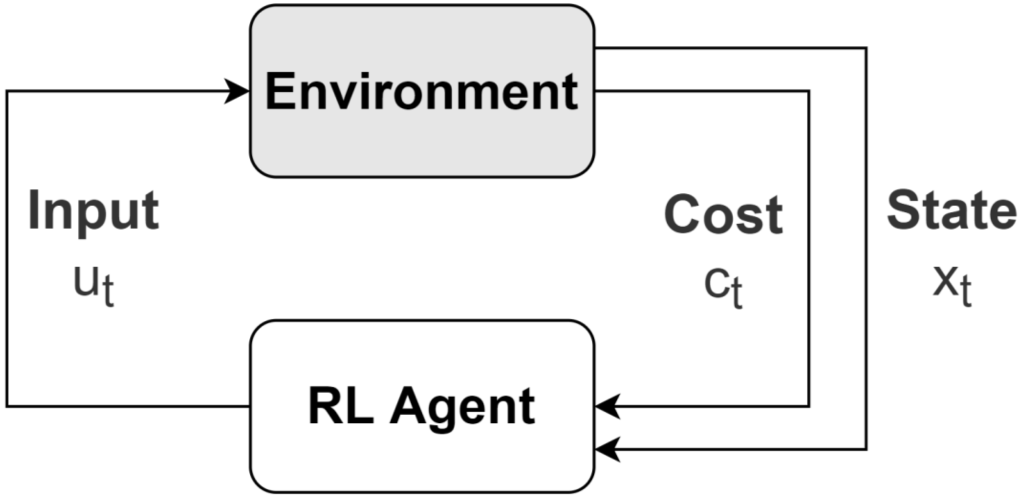
Q-function and Bellman's equation

- Q-function: estimates future cost
- Bellman's equation:

$$Q(x_t, u_t) = c(x_t, u_t) + \gamma Q(x_{t+1}, \mu(x_{t+1}))$$

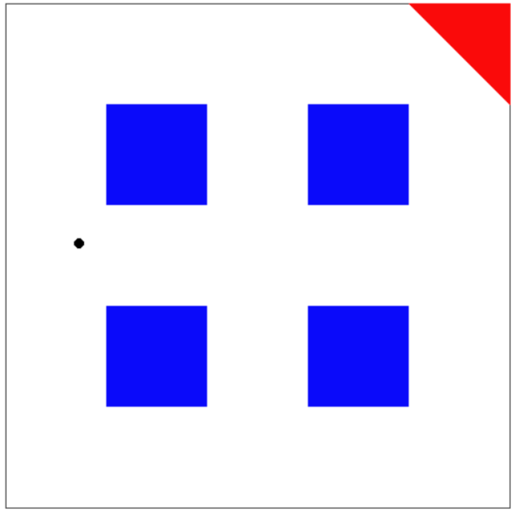
- Temporal Difference error:

$$TD_{er} = c(x_t, u_t) + \gamma Q(x_{t+1}, \mu(x_{t+1})) - Q(x_t, u_t)$$



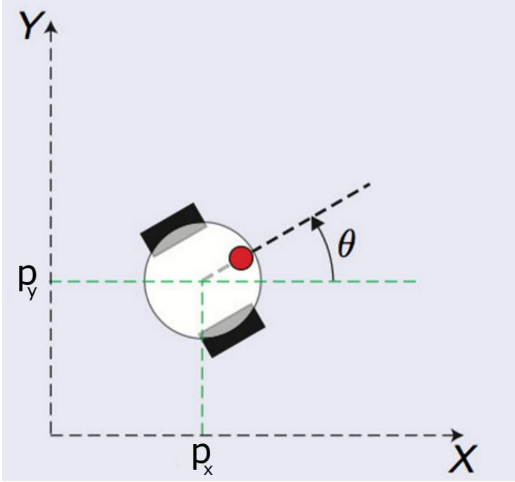
Problem Statement

- Task: reach goal area from any initial configuration in the environment
- Environment limits, obstacles are unknown
- Continuous states and inputs
- Robot simulated as a unicycle, but model is unknown



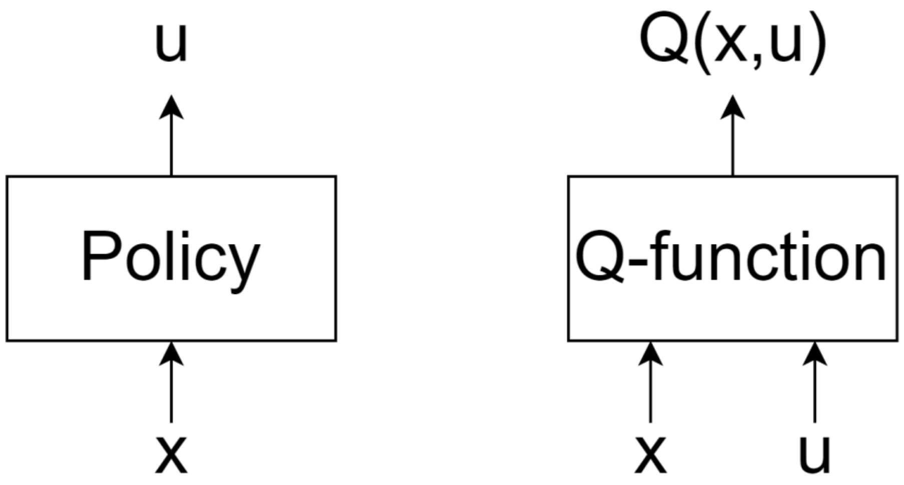
State: $x = [p_x \; p_y \; \theta]$

Model: $\dot{x} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$



Deep Deterministic Policy Gradient (DDPG)

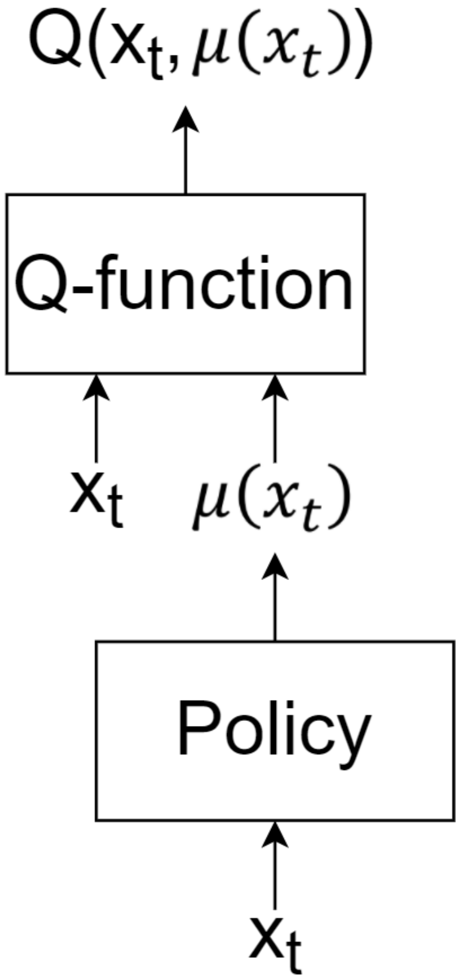
- Policy: controller
- Q-function: measures the cost of a state-input pair
- Implemented as deep neural networks



- Experience Replay: keeps a buffer of past transitions {state, input, cost, next state}

Replay Buffer
$\{x_1, u_1, c(x_1, u_1), x_2\}$
$\{x_2, u_2, c(x_2, u_2), x_3\}$
…
$\{x_\tau, u_\tau, c(x_\tau, u_\tau), x_{\tau+1}\}$

Policy Training



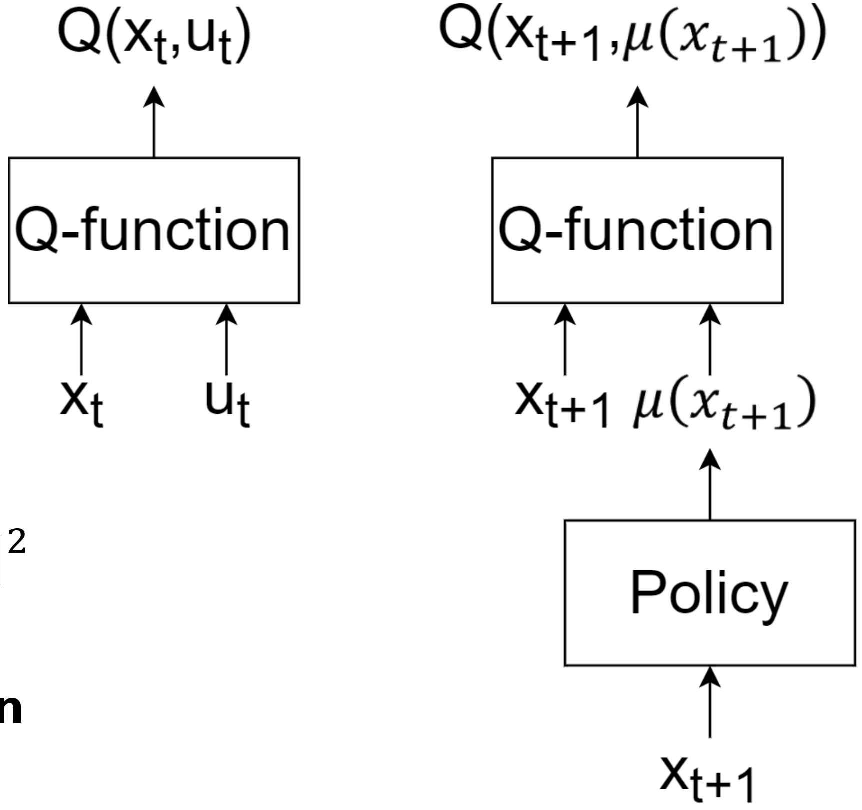
- Trained to choose inputs that minimize cost
- Policy network parameters: θ_k^μ
- Policy gradient update:

$$\theta_{k+1}^\mu = \theta_k^\mu - \alpha_\mu \nabla_{\theta^\mu} Q(x_t, \mu(x_t))$$

Q-function Training

- Sampled transition: $\{x_t, u_t, \mathrm{c}(x_t, u_t), x_{t+1}\}$
- Q-function network parameters: θ_k^Q
- Gradient descent update minimizing the Temporal Difference error:

$$\theta_{k+1}^Q = \theta_k^Q - \alpha_Q \nabla_{\theta^Q} [\underbrace{c(x_t, u_t) + \gamma Q(x_{t+1}, \mu(x_{t+1}))}_{\textbf{Target of the update}} - \underbrace{Q(x_t, u_t)}_{\textbf{Prediction}}]^2$$



Target Networks

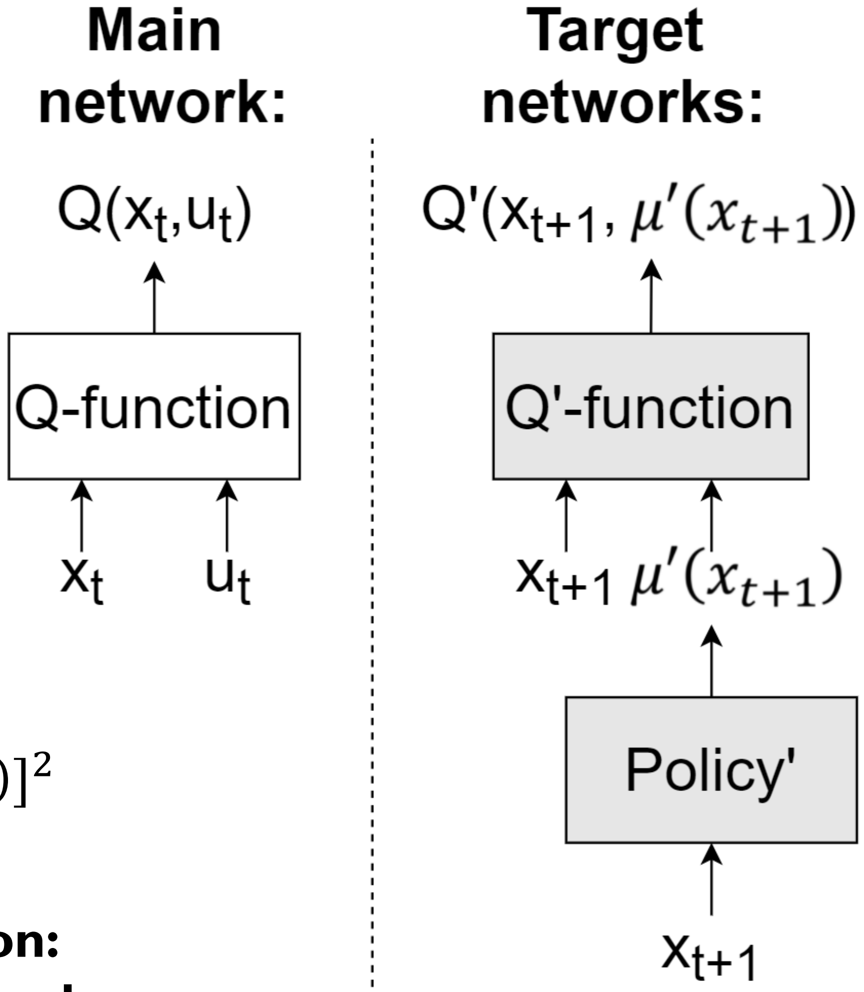
- Copies of the main networks: $Q'(x,u)$ and $\mu'(x)$, with parameters: $\theta^{Q'}$ and $\theta^{\mu'}$
- Slowly track the main networks parameters:

$$\begin{cases} \theta^{Q'} = \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} = \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{cases}$$

With $\tau \ll 1$

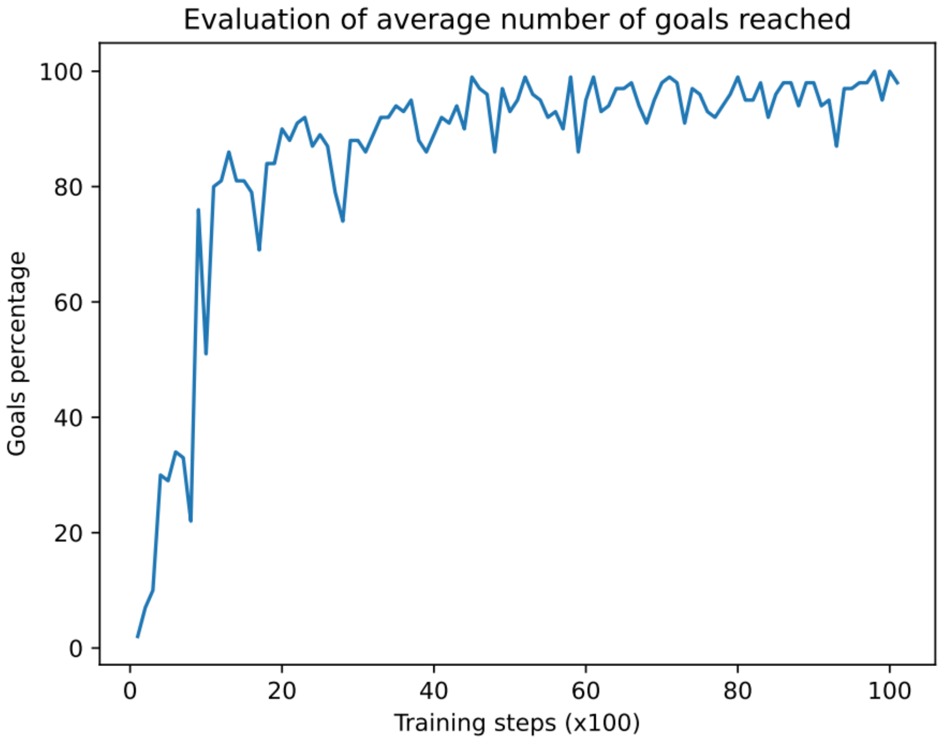
- Only used for target computation of the TD error:

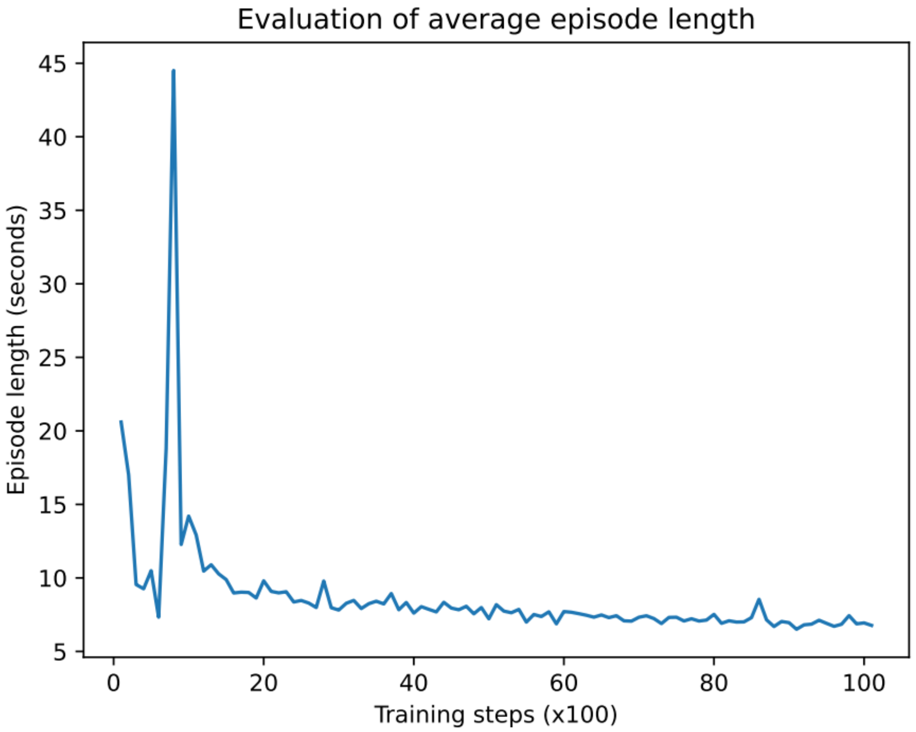
$$\theta_{k+1}^{Q} = \theta_{k}^{Q} - \alpha_Q \nabla_{\theta^Q}[\underbrace{c(x_t,u_t) + \gamma \boldsymbol{Q}'(x_{t+1}, \boldsymbol{\mu}'(x_{t+1}))}_{\text{Target of the update: Target networks}} - \underbrace{Q(x_t,u_t)}_{\text{Prediction: main network}}]^2$$



Controller Performance Evolution

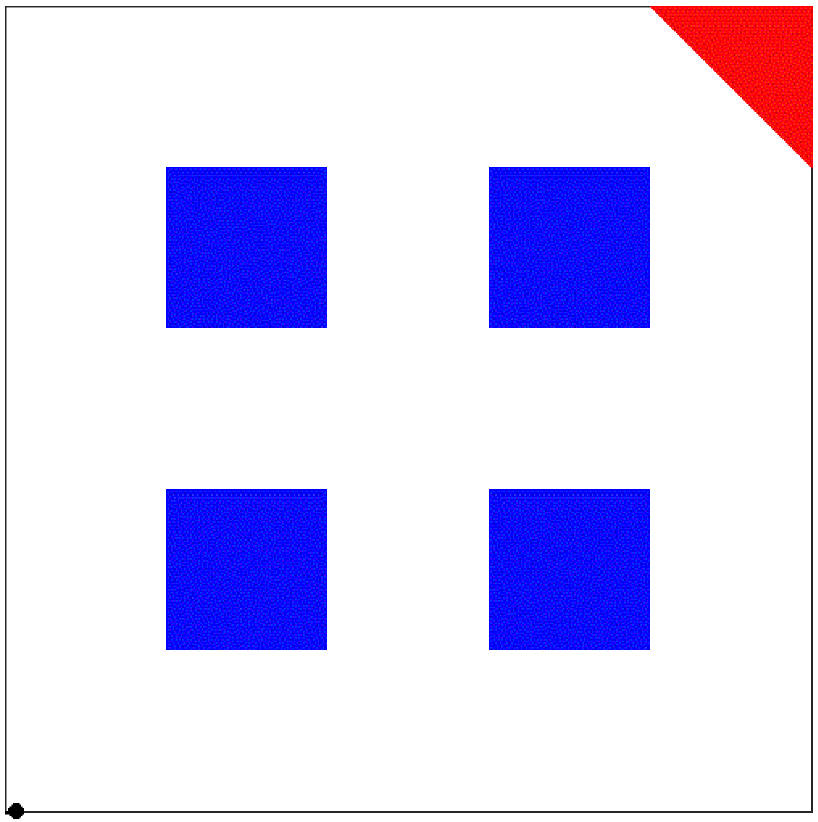
- Experiments from a set of fixed initial conditions
- 10000 episodes training



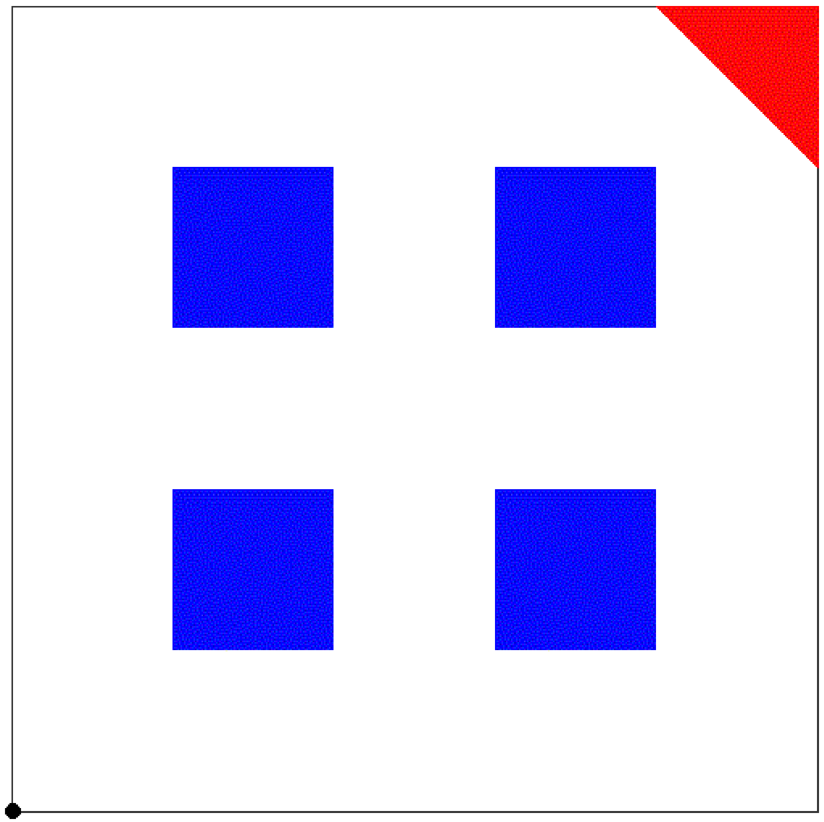


Resulting Robot Trajectories

10Hz simulation:



100Hz simulation:



Conclusions

- Achieved above 95% success rate in the autonomous goal-reaching task
- Provided a custom Python implementation of DDPG and the unicycle simulation
- Drawback: length of training
- Future work: algorithm tuning; distributed setting; integration with model-based techniques