# Decentralized Machine Learning Control (of Drone Swarms)

Miloš Stanković

| | |
|---|---|
| University Singidunum<br>Belgrade, Serbia | Copelabs, Universidade Lusófona<br>Lisboa, Portugal |

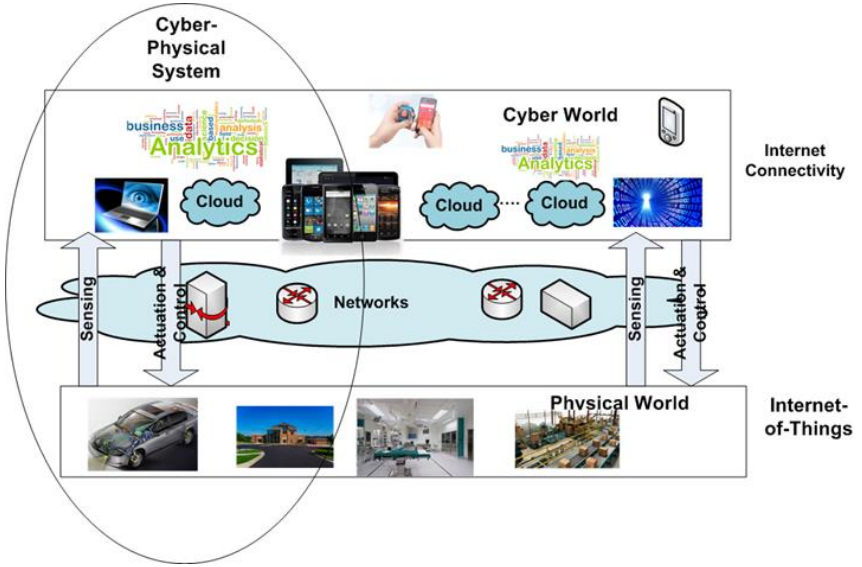
# Talk Outline

1. Our research on Decentralized Machine Learning Control
2. Multi Drone Reinforcement Learning Examples
3. Why we need cloud resources
4. About the specific OCRE project

# Background

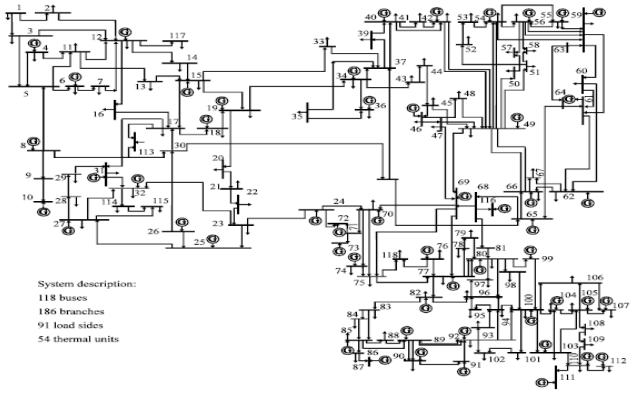
- Interdisciplinary subject: Control and Communication Engineering, Data Science (Machine Learning), Computer Science, Applied Mathematics
- Foundation of **Intelligent Networked Cyber-Physical Systems (INCPS)**

- Complex distributed autonomous multi-agent systems

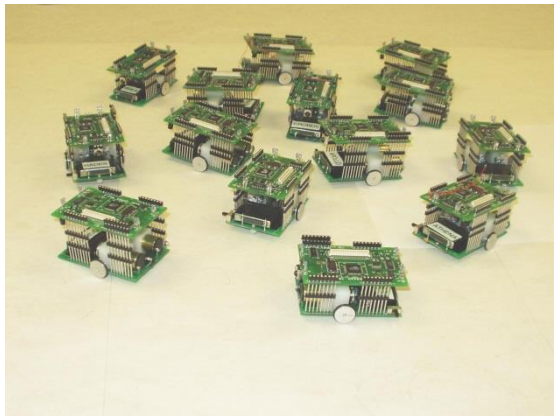



- Numerous revolutionary and pervasive applications: **swarm robotics, autonomous vehicles, smart buildings, cities and power grids, intelligent agriculture, transportation** and **manufacturing systems**, etc.

# Examples

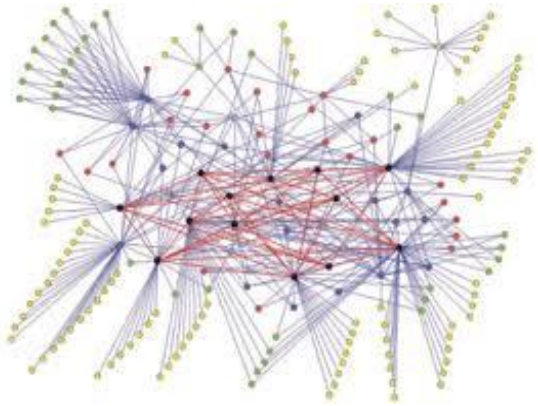



Smart Power Grid



Robotic Wireless Networks

Platoons and Formations of Vehicles



Biomolecular Networks

Smart Agriculture

IoT


4

# Project Summary

- The general objective:

Development of advanced methods and algorithms for **decentralized Machine Learning Control (MLC)** for **Networked Cyber-Physical Systems**

- Complex, spatially distributed and networked autonomous multi-agent dynamical systems
- The methodological solutions cross the traditional boundaries between **(deep) machine learning, control systems (reinforcement learning), and decentralization of functions**.


- Acknowledgements: **EC RIA**, **OCRE project**, **Science Fund of the Republic of Serbia**, and **FCT**

# Concept and Methodology

- **Dimensionality, uncertainty and information structure constraints ⇒ decentralized decision making theory**
- **Uncertainty and vulnerability** (subsystems, interconnections, environment, communication channels and computing devices) - intrinsic to large-scale multi-agent systems

  ⇒ Decentralization provides superior **robustness** to **structural uncertainty**
- Drawbacks of decentralization:

  **1) Lack of awareness of the global mission**
  - Proposed approaches:
    - Distributed inter-agent agreement based on **consensus techniques**
    - **Game theoretic** approach
    - **Learning and adaptation** techniques:
    - **Reinforcement Learning (RL)** – recently very popular methodology for dealing with decision making problems in uncertain environments. Could represent a basis for development of decentralized learning control for INCPS

  **2) Increased vulnerability (**to inconvenient faults and/or security attacks)

  - Proposed approaches **increasing resilience:**
    - **Robust statistics and/or game theory**
    - **Learning and adaptation** techniques

# Our Results on Distributed Multi-Agent RL

# RL Problem Setup and Basics

# Basic Ideas

- Learning what to do – how to map situations to actions — so as to maximize a numerical reward
- **Trial-and-error** search
- **Delayed (averaged) reward** (interaction is with **dynamical systems**)
- Direct learning from own experience - **different from supervised learning** or system identification
- Important challenge: trade-off between **exploration** and **exploitation**

# RL Problem setup

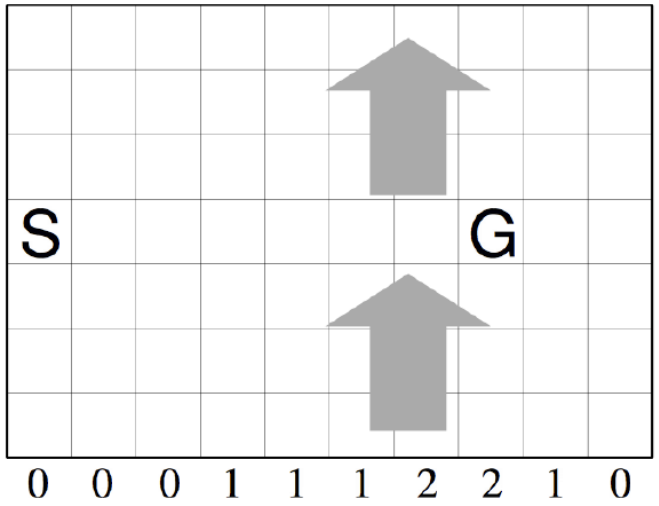
- Markov decision process: $(S, \mathcal{A}, P, r, \gamma)$ – finite set of states, finite actions set, transition matrix $P(s'|s,a)$, $r(s,a,s')$ immediate reward, $\gamma \in (0,1)$ discount factor

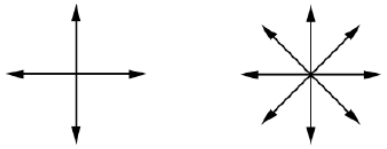
- Discounted infinite horizon payoff (**value function**):

$$V(s) = E\left\{\sum_{n=0}^{\infty} \gamma^n r_{n+1} | s(0) = s\right\}$$

# Example

- (Windy) gridworld







- Reward = -1 per time-step until reaching goal
- Undiscounted

# Example

- Driving on a highway model


- $p(\text{moving}, a^{highway}) = \frac{1}{\text{state}}$ , $p(\text{stuck}, a^{highway}) = 1 - \frac{1}{\text{state}}$
- $p(\text{moving}, a^{exit}) = 0.8$ , $p(\text{stuck}, a^{exit}) = 0.2$
- $r(a^{exit}) = -4$, $r(a^{highway}) = -1$

- Goals:
  - Evaluate a (possibly randomized) policy (state-action map)
  - Find the optimal policy

# Policy Evaluation and Optimal Control of MDPs

- Randomized stationary policy $\pi$: $S \times \mathcal{A} \to [0,1]$
- **1) For given $\boldsymbol{\pi}$ find $\boldsymbol{V^{\pi}(s)}$:**
- Bellman prediction equation:

$$V = R + \gamma PV$$

$$V = \left[V(s_1), \dots, V\left(s_{|S|}\right)\right]^T, R = \left[R(s_1), \dots, R\left(s_{|S|}\right)\right]^T,$$

$$R(s) = \sum_{s' \in S} \sum_{a \in \mathcal{A}} \pi(s', a)\, P(s' \,|s, a) r(s, a, s'),$$

$$P(s, s') = \sum_{a \in \mathcal{A}} \pi(s', a)\, P(s' \,|s, a)$$

- **2) Find $\boldsymbol{\pi}$ maximizing the discounted reward:**
- Bellman optimality equation:

$$V^*(s) = \max_{a \in \mathcal{A}} \{R(s, a) + \gamma \sum_{s' \in S} P(s' \,|s, a)\, V^*(s')\}$$

$$R(s, a) = \sum_{s' \in S} P(s' \,|s, a) r(s, a, s')$$

- Resulting policy is deterministic optimal policy

# Iterative Policy Evaluation

- For given $\pi$ find $V^{\pi}(s)$

$$V_{k+1} = R + \gamma P V_k, \qquad V_0 = 0$$

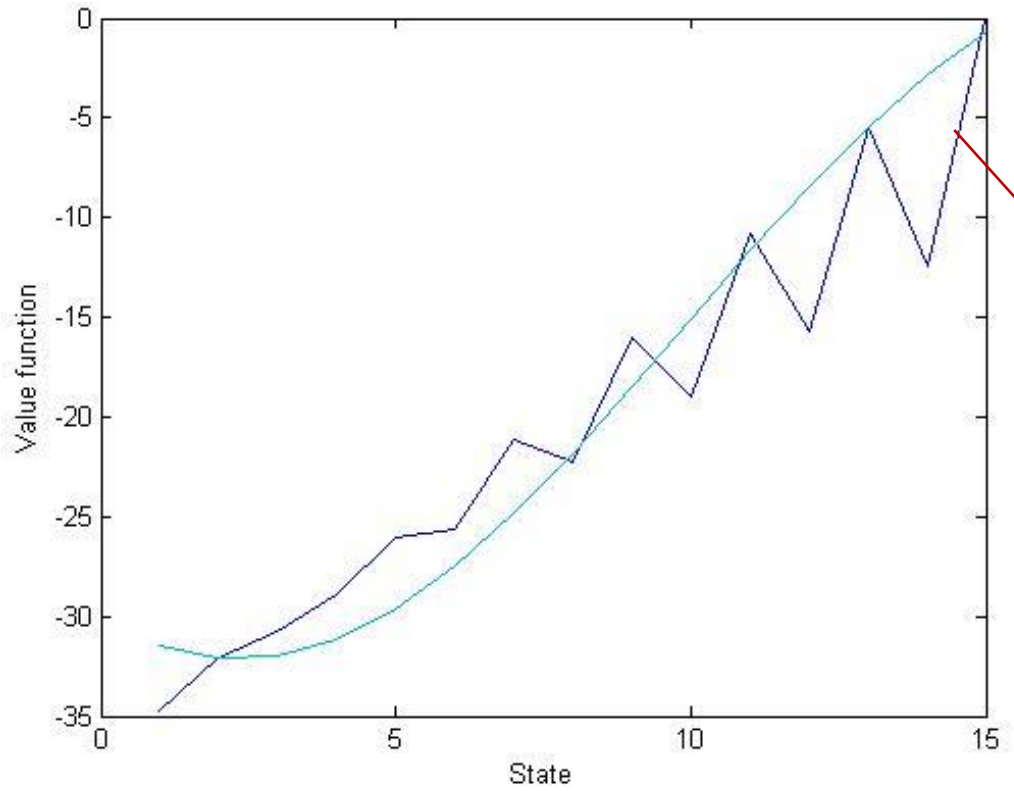
$$R(s) = \sum_{s' \in S} \sum_{a \in \mathcal{A}} \pi(s', a)\, P(s' \,|s, a) r(s, a, s'),$$

$$P(s, s') = \sum_{a \in \mathcal{A}} \pi(s', a)\, P(s' \,|s, a)$$

- For finding optimal policy:
  - **policy iteration**: After finding $V_{\pi}$ improve the policy greedily based on current $V_{\pi}$
  - **value iteration**: iteratively apply the Bellman optimality equation

$$V^*_{k+1}(s) = \max_{a \in \mathcal{A}} \{\mathrm{R(s, a)} + \gamma \sum_{s' \in S} P(s' \,|s, a)\, V^*_k(s')\}$$

# Example


Value function

# Monte Carlo Methods

- To evaluate state $s$:
- The first time-step $t$ that state $s$ is visited in an episode:
  - Increment counter $N(s)$
  - Increment total return $S(s) \leftarrow S(s) + G_t$

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s)$ converges to true value

- Monte-Carlo policy evaluation uses **empirical mean return** instead of expected return

# Temporal Difference Methods

- We want to update in each iteration
- Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma\, V(S_{t+1})$:

TD target

$$V(S_t) \leftarrow V(S_t) + (R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t))$$

TD error

# On-policy vs. Off-policy Learning

- Behavior policy $\pi^{\mathrm{b}}$is different than target policy $\pi$
- Importance sampling:

$$V(s) = E_{\pi}^{b}\left\{\sum_{n=0}^{\infty} \rho_n \gamma^n \, r_{n+1} | s(0) = s\right\}$$

$$\rho_n = \frac{\pi(a_n|s_n)}{\pi^b(a_n|s_n)}$$

# MC vs TD

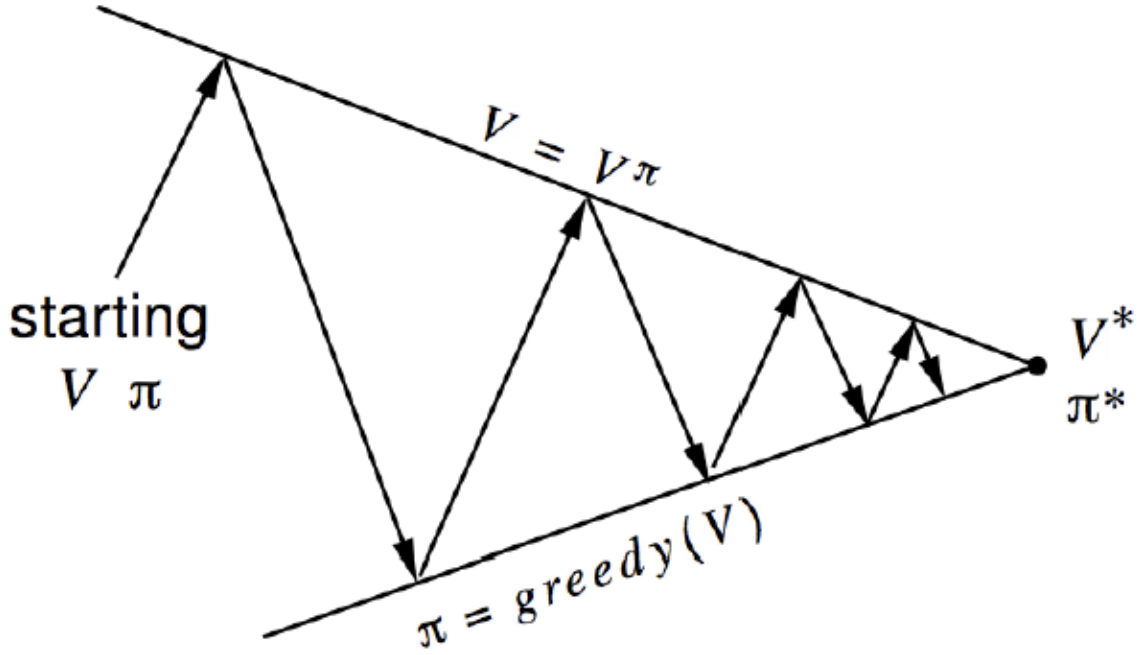
- MC has **high variance, zero bias (**not very sensitive to initial value)
- **Good convergence properties** (even with function approximation)
- Very simple to understand and use

- TD has **low variance, some bias**
- Usually **more efficient** than MC
- TD **not always convergent** with function approx.

# Control

Policy Optimization

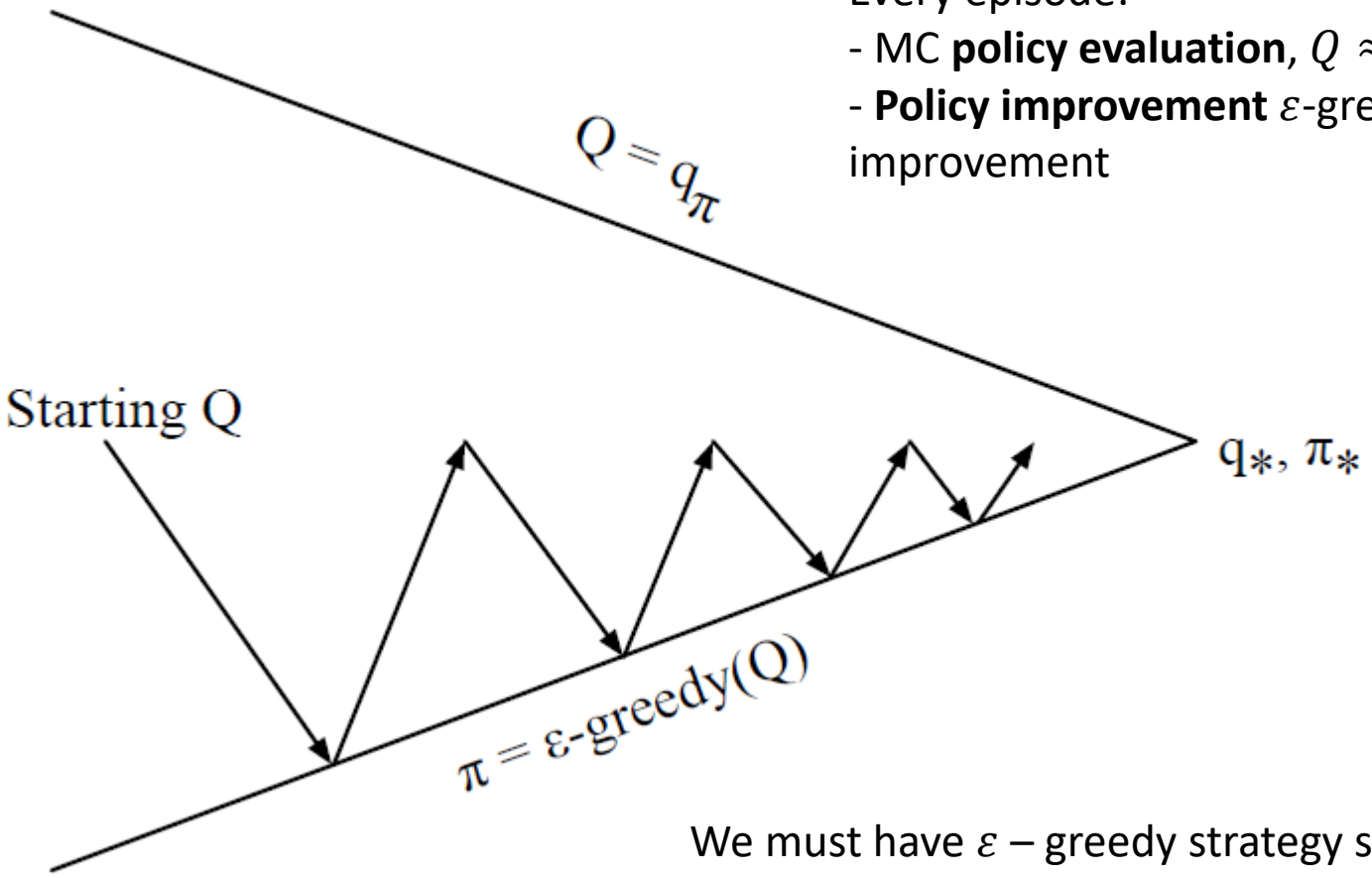
# Policy Iteration




- Recall – we use the Belman equation

$$V = R + \gamma PV$$

# Monte-Carlo Policy Iteration



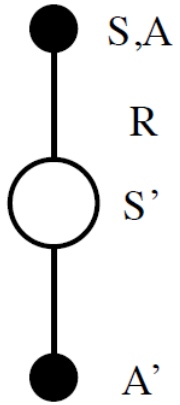

Every episode:
- MC **policy evaluation**, $Q \approx q_\pi$
- **Policy improvement** $\varepsilon$-greedy policy improvement

We must have $\varepsilon$ – greedy strategy since we need sufficient **exploration**!

# TD – based Control

- Due to the advantages of TD: use TD instead of MC in our control loop
- Apply TD to $Q(S, A)$
- Use $\varepsilon$-greedy policy improvement
- Update **every time-step**

# SARSA Learning

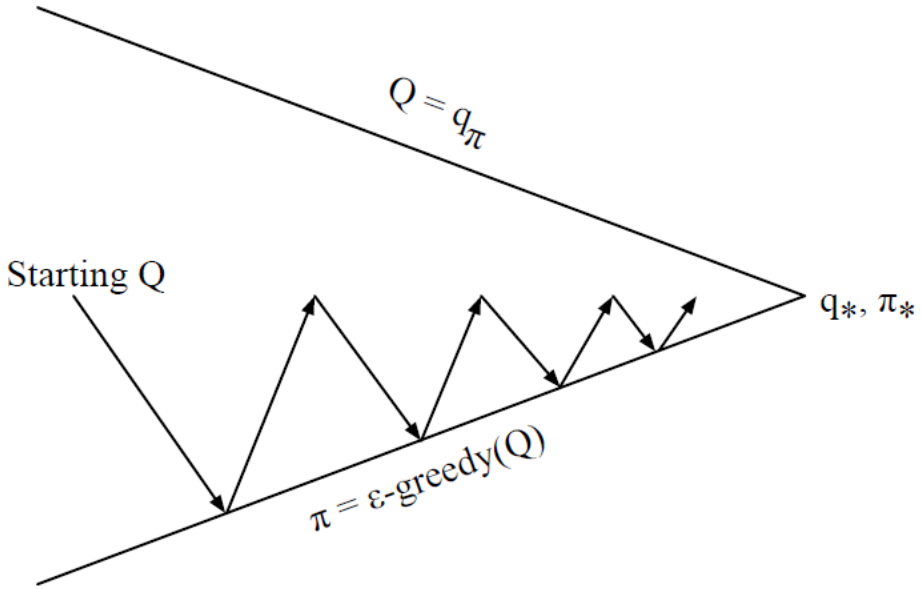



- SARSA learning:

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha(\mathrm{R_{k+1}} + \gamma Q_k(s_{k+1}, a_{k+1}) - Q_k(s_k, a_k)$$

- On-policy

# SARSA Learning




- **Every time-step**:
- Policy evaluation SARSA, $Q \approx q_\pi$
- Policy improvement $\varepsilon$-greedy policy improvement

# Q-Learning

- The **target policy $\boldsymbol{\pi}$** is greedy w.r.t. $Q(s,a)$

$$\pi(S_{t+1}) = \text{argmax}_{a'} Q(S_{t+1}, a')$$

- The **behavior policy $\boldsymbol{\mu}$** is $\varepsilon$-greedy w.r.t. $Q(s,a)$
- Q-learning:

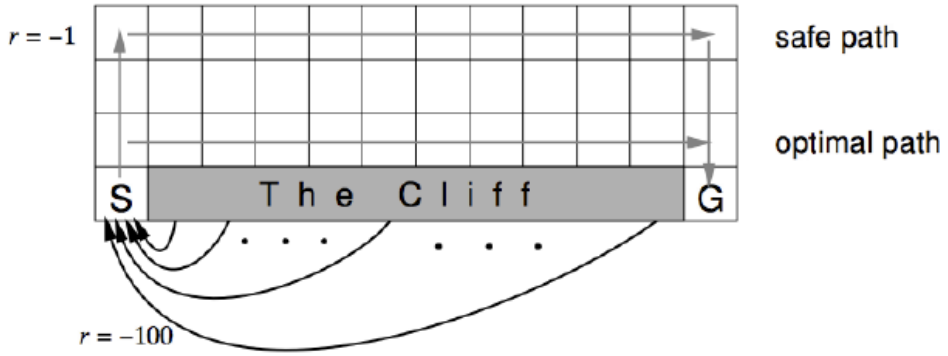
$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha(\mathrm{R_{k+1}} + \gamma \max_a Q_k(s_{k+1}, a) - Q_k(s_k, a_k))$$

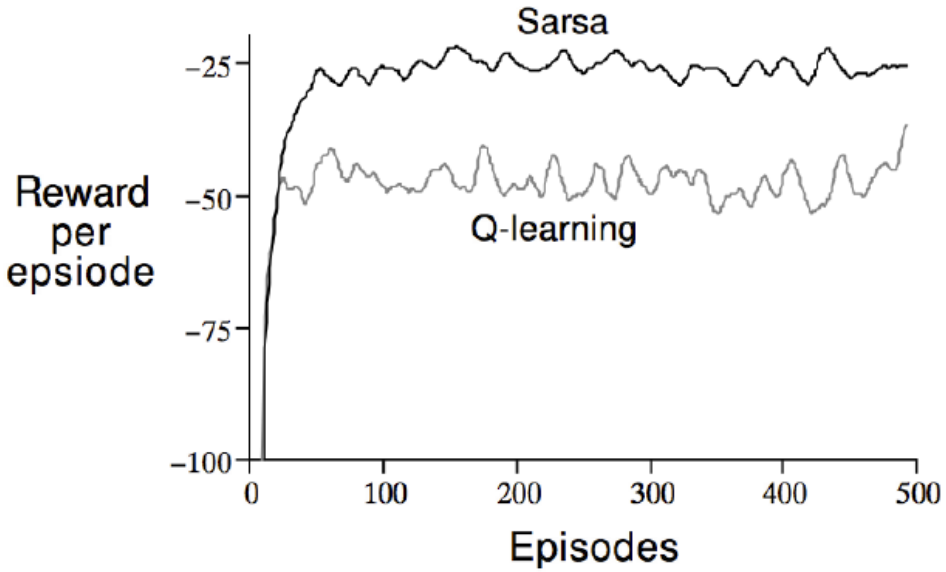
# SARSA vs Q-learning

- Q-learning control converges to the **optimal action-value function**

- SARSA converges to the **action-value function corresponding to the applied policy** (usually $\varepsilon$- greedy)

# Cliff Walking Example



# Large-Scale Reinforcement Learning

- Examples:
  - Backgammon: $10^{20}$ states
  - Computer Go: $10^{170}$ states
  - Robots/Drones: continuous state space

# Value Function Approximation

- Problem with large MDPs:
- **Too many states and/or actions to store in memory**
- It is **too slow to learn** the value of each state individually

- Solution for large MDPs:
- **Function approximation**

$$\hat{v}(s, \boldsymbol{w}) \approx v_\pi(s) \quad \text{or} \quad \hat{q}(s, a, \boldsymbol{w}) \approx q_\pi(s, a)$$

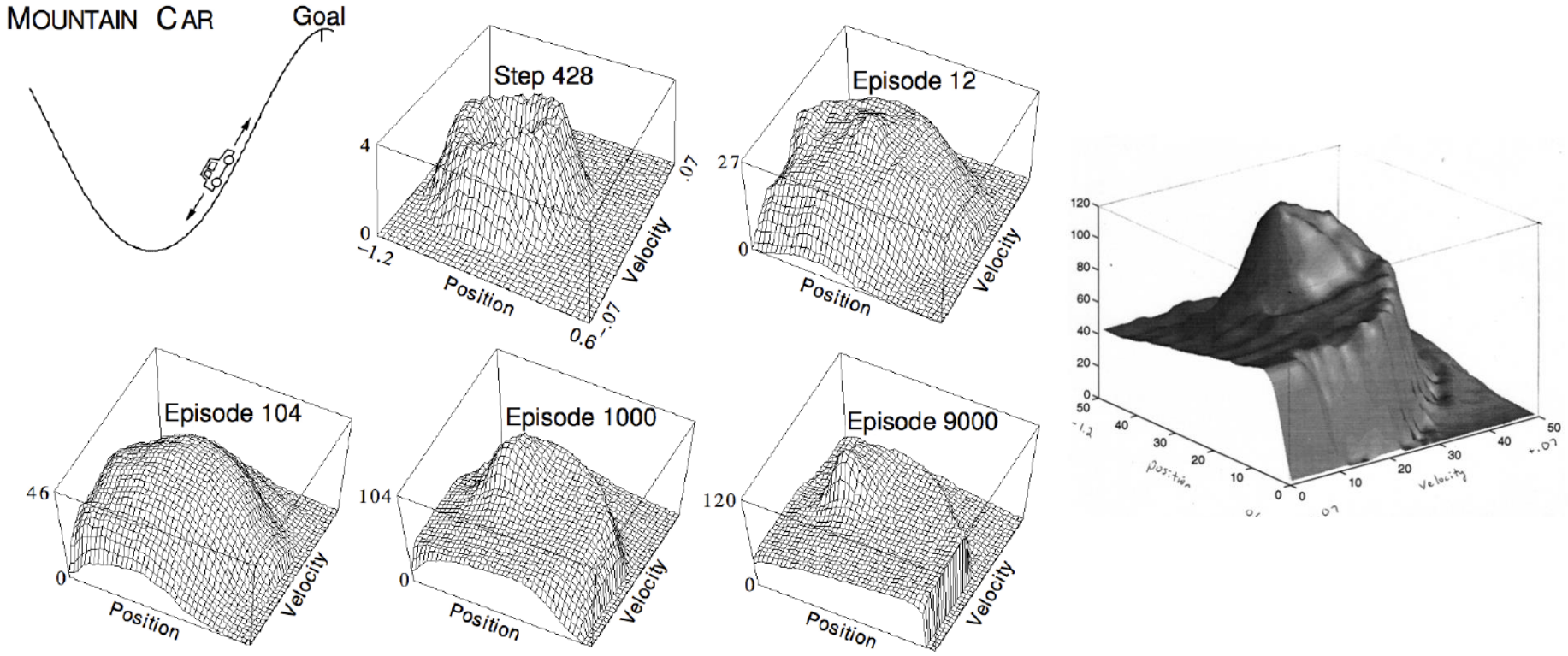
- Possibility to generalize from seen states to unseen states
- Update parameters $\boldsymbol{w}$ using MC or TD learning

# Which Function Approximator?

- There are many function approximators, e.g.
  - **Linear** combinations of features
  - **Neural network**
  - Decision tree
  - Nearest neighbour
  - Fourier / wavelet bases
  - Etc.

# Linearly Approximated Q-learning in Mountain Car Example

- Python code



# Deep Q Learning

- DQN: Q-Learning with a **Deep Neural Network as a function approximator**
- Training can be unstable
- Experience Replay and Target Network (*T* steps frozen parameters) helps stabilization

- CNNs – applicability to purely vision based learning

# Experience Replay in Deep Q-Networks (DQN)

- DQN uses experience replay and fixed Q-targets:

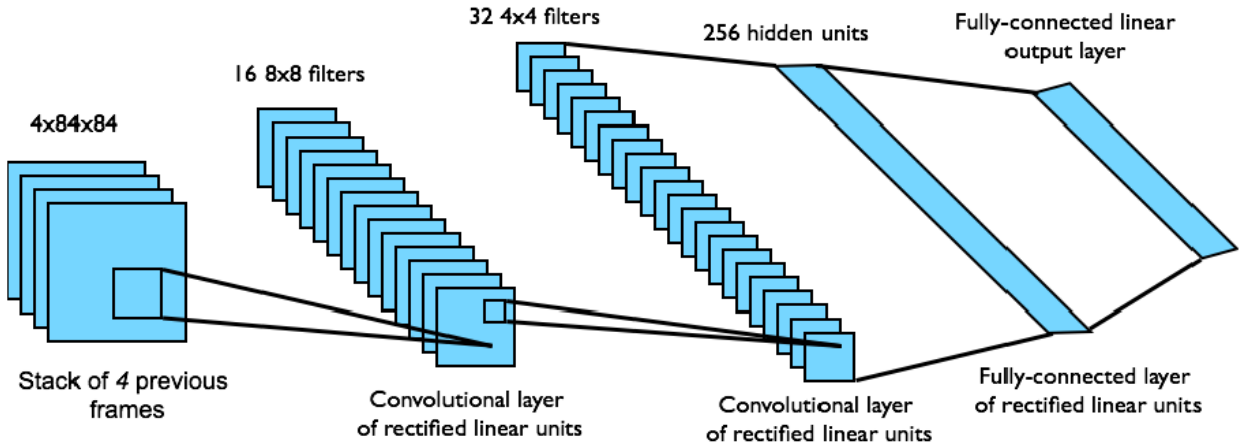
- Take action at according to $\varepsilon$-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $D$
- Sample random mini-batch of transitions $(s, a, r, s')$ from $D$
- Compute Q-learning targets w.r.t. old, fixed parameters $w^-$

- Optimize (using SGD) MSE between Q-network output and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s'}\{\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w_i)\right)^2$$

# Example - DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels
- Input state $s$ is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



# Policy gradient methods

- Parameterize the policy function
- Find gradient of the objective function with respect to the policy parameters
- Typically results in the so called Actor-Critic methods

# Summary of Typical Single-Agent Algorithms

- Value function learning/approximation:
  - Monte-Carlo based
  - Temporal-difference based (TD, TD($\lambda$), GTD($\lambda$), ETD($\lambda$))
  - Least-Square methods
- Q-function learning:
  - Q-learning
  - SARSA
  - …
- Policy gradient/ Actor-Critic methods

# Distributed Multi-Agent RL

# Multi-Agent Policy Evaluation -Problem Setup

- $N+1$ MDPs characterized by $\{S, A, p(s'|s,a), R(s,a,s')\}$

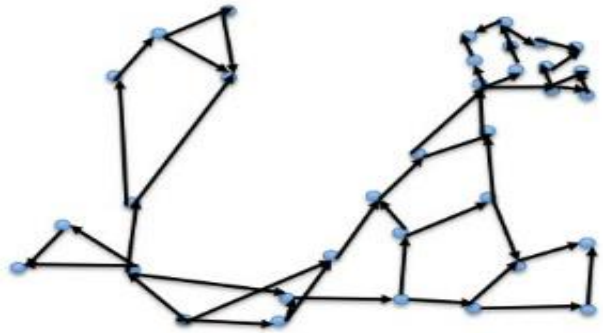
Finite set of states

Finite set of actions

Transition probabilities

Random rewards

- Each $\mathrm{MDP}^{(i)}$ has an associated stationary policy $\pi^{(i)}(a|s)$
- The goal of each agent: find the **state-value function** for the reference $\mathrm{MDP}^{(0)}$ with **(target) policy $\boldsymbol{\pi^{(0)}}$**
- Each agent $i$ applies (**behavior) policy $\boldsymbol{\pi^{(i)}}$**, and can only observe **local state transitions and rewards**
- **Inter-agent communication** is allowed according to the given network topology
- **Cooperative off-policy reinforcement learning**



# Off-Policy Value Function Approximation

- State space is typically large ⇒ value function parameterization:

$$v_{\pi(0)} \approx v_\theta = \Phi\theta, \qquad \theta \in \mathcal{R}^p, v_\theta \in \mathcal{R}^M$$

Feature vectors $\phi(s), s \in S$

- Typically $M \gg p$
- Goal: find optimal set of parameters $\Theta = [\theta_1, \dots, \theta_N]$ minimizing

$$J(\Theta) = \sum_{i=1}^{N} q_i J_i(\theta_i) \text{ subject to } \theta_1 = \theta_2 = \cdots = \theta_N$$

$$J_i(\theta_i) = \left\| \Pi_{\xi_i} \left\{ T^{(\lambda_i)} v_{\theta_i} - v_{\theta_i} \right\} \right\|_{\xi_i}$$

- Projected generalized Bellman errors
- (Weighted) projection operator
- Generalized Bellman operator (with $\lambda$-parameters)
- Steady state distribution parameters

# Off-Policy Value Function Approximation

- By calculating local gradients the following two **local gradient descent algorithms** (Sutton et al. 2009) can be derived:
  - GTD2($\lambda$):

Importance sampling weights

Discount factors

$$\theta_i(n+1) = \theta_i(n) + \alpha(n)\rho_i(n)\left[\Phi\left(S_i(n)\right) - \gamma_i(n+1)\phi\left(S_i(n+1)\right)\right]e_i(n)^T w_i(n)$$

$$w_i(n+1) = w_i(n) + \beta(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \phi\left(S_i(n)\right)\phi\left(S_i(n)\right)^T w_i(n)\right]$$

Step sizes

Temporal difference term

Feature vector

- Eligibility traces:

$$e_i(n) = \lambda_i(n)\gamma_i(n)\rho_i(n-1)e_i(n-1) + \phi(S_i(n))$$

# Off-Policy Value Function Approximation

- TDC($\lambda$):

$$\theta_i(n+1) = \theta_i(n) + \alpha(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \rho_i(n)\left(1-\lambda_i(n+1)\right)\gamma_i(n+1)\phi\left(S_i(n+1)\right)\right]e_i(n)^T w_i(n)$$

$$w_i(n+1) = w_i(n) + \beta(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \Phi\left(S_i(n)\right)\Phi\left(S_i(n)\right)^T w_i(n)\right]$$

# Non-gradient based algorithms (faster)

- Instead of calculating gradients of the objective, we consider algorithms:
- **1) Standard TD($\lambda$) (not stable in general!)**:

$$\theta_i(n+1) = \theta_i(n) + \alpha_i(n)e_i(n)\rho_i(n)[R_i(n) - \gamma_i(n+1)\phi_i(n+1)^T\theta_i(n) - \phi_i(n)^T\theta_i(n)]$$

Discount factors

Step size

Importance sampling weights

Feature vector

Temporal difference term

  - Eligibility trace vector: $e_i(n) = \lambda_i(n)\gamma_i(n)\rho_i(n-1)e_i(n-1) + \phi_i(n)$
- **2) Emphatic TD($\lambda$) (ETD($\lambda$), stable!,** Sutton et al. 2016):

$$e_i(n) = \lambda_i(n)\gamma_i(n)\rho_i(n-1)e_i(n-1) + \mu_i(n)\phi_i(n)$$
$$\mu_i(n) = \lambda_i(n)w_i(n) + (1-\lambda_i(n))f_i(n)$$
$$f_i(n) = \gamma_i(n)\rho_i(n-1)f_i(n-1) + w_i(n)$$

# Proposed Algorithms for Distributed VF Approximation

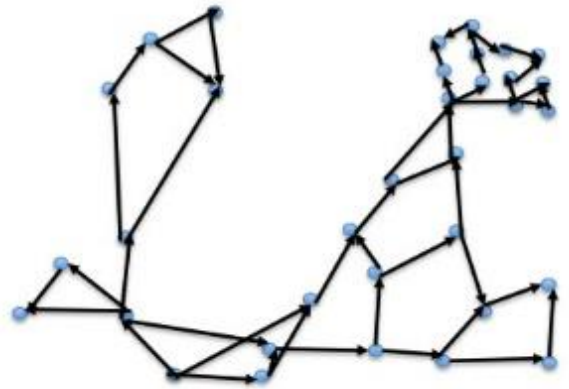
- D1-GTD2($\lambda$):

$$\theta_i'(n) = \theta_i(n) + \alpha(n) q_i \rho_i(n) \left[\Phi\left(S_i(n)\right) - \gamma_i(n+1)\phi\left(S_i(n+1)\right)\right] e_i(n)^T w_i(n)$$
$$w_i'(n) = w_i(n) + \beta(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \phi\left(S_i(n)\right)\phi\left(S_i(n)\right)^T w_i(n)\right]$$

- Convexification:

Random network weights

$$\theta_i(n+1) = \sum_{j=1}^{N} a_{ij}(n)\theta_j'(n)$$
$$w_i(n+1) = w_i'(n)$$



# Proposed Algorithms for Distributed VF Approximation

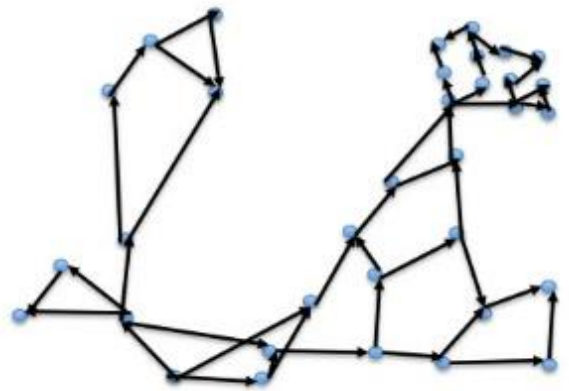
- D2-GTD2($\lambda$):

$$\theta_i'(n) = \theta_i(n) + \alpha(n) q_i \rho_i(n) \left[\Phi\left(S_i(n)\right) - \gamma_i(n+1)\phi\left(S_i(n+1)\right)\right] e_i(n)^T w_i(n)$$
$$w_i'(n) = w_i(n) + \beta(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \phi\left(S_i(n)\right)\phi\left(S_i(n)\right)^T w_i(n)\right]$$

- Convexification:

Random network weights

$$\theta_i(n+1) = \sum_{j=1}^{N} a_{ij}(n)\theta_j'(n)$$
$$w_i(n+1) = \sum_{j=1}^{N} a_{ij}(n) w_j'(n)$$



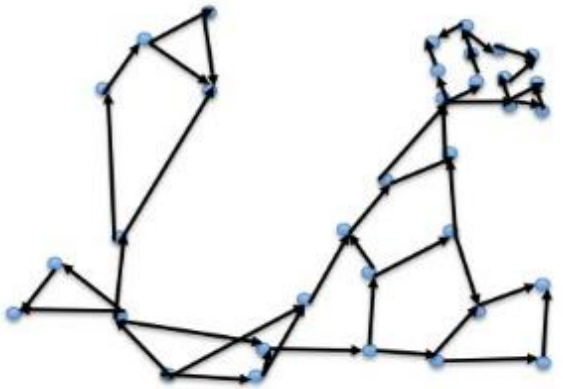
# Proposed Algorithms for Distributed VF Approximation

- D1-TDC($\lambda$):

$$\theta_i'(n) = \theta_i(n) + \alpha(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \rho_i(n)\left(1 - \lambda_i(n+1)\right)\gamma_i(n+1)\phi\left(S_i(n+1)\right)\right]e_i(n)^T w_i(n)$$
$$w_i'(n) = w_i(n) + \beta(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \phi\left(S_i(n)\right)\phi\left(S_i(n)\right)^T w_i(n)\right]$$

- Convexification:

$$\theta_i(n+1) = \sum_{j=1}^{N} a_{ij}(n)\theta_j'(n)$$
$$w_i(n+1) = w_i'(n)$$



# Proposed Algorithms for Distributed VF Approximation

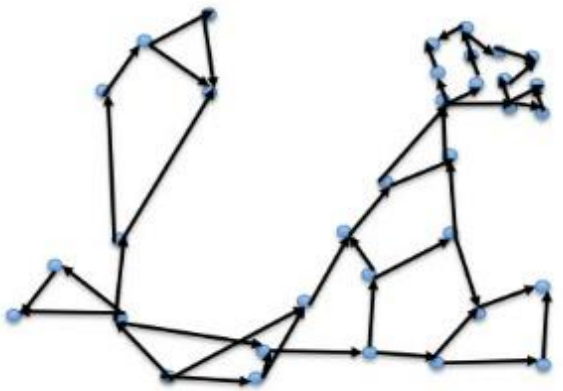
- D2-TDC($\lambda$):

$$\theta_i'(n) = \theta_i(n) + \alpha(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \rho_i(n)\left(1 - \lambda_i(n+1)\right)\gamma_i(n+1)\phi\left(S_i(n+1)\right)\right]e_i(n)^T w_i(n)$$
$$w_i'(n) = w_i(n) + \beta(n)\left[e_i(n)\delta_i\left(v_{\theta_i}, n\right) - \phi\left(S_i(n)\right)\phi\left(S_i(n)\right)^T w_i(n)\right]$$

- Convexification:

Random network weights

$$\theta_i(n+1) = \sum_{j=1}^{N} a_{ij}(n)\theta_j'(n)$$
$$w_i(n+1) = \sum_{j=1}^{N} a_{ij}(n)w_j'(n)$$



# Proposed Algorithms for Distributed VF Approximation

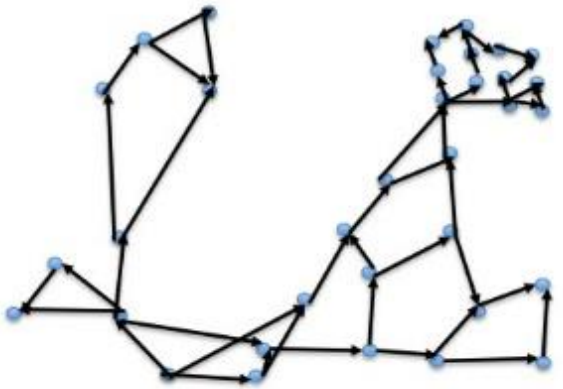
- D-TD($\lambda$) and D-ETD($\lambda$) :

$$\theta_i'(n) = \theta_i(n) + \alpha_i(n)e_i(n)\rho_i(n)[R_i(n) - \gamma_i(n+1)\phi_i(n+1)^T\theta_i(n) - \phi_i(n)^T\theta_i(n)]$$

- Convexification:

Random network weights

$$\theta_i(n+1) = \sum_{j=1}^{N} a_{ij}(n)\theta_j'(n)$$



# Convergence

- Rigorous proofs of stochastic convergence:
  - M.S. Stankovic, M. Beko and S.S. Stankovic. **Distributed Consensus-Based Multi-Agent Temporal-Difference Learning**, *Automatica*, Vol. 151, 2023.
  - M.S. Stanković, M. Beko and S.S. Stanković, **Distributed Value Function Approximation for Collaborative Multi-Agent Reinforcement Learning**, *IEEE Transactions on Control of Network Systems*, 8(3), pp. 1270 – 1280, 2021.
  - M. S. Stankovic, M. Beko and S. S. Stankovic, **Distributed Consensus-Based Multi-Agent Temporal-Difference Learning**, *60th IEEE Conference on Decision and Control (CDC),* 2021..
  - …

# Illustrative simulation results

- Highway model

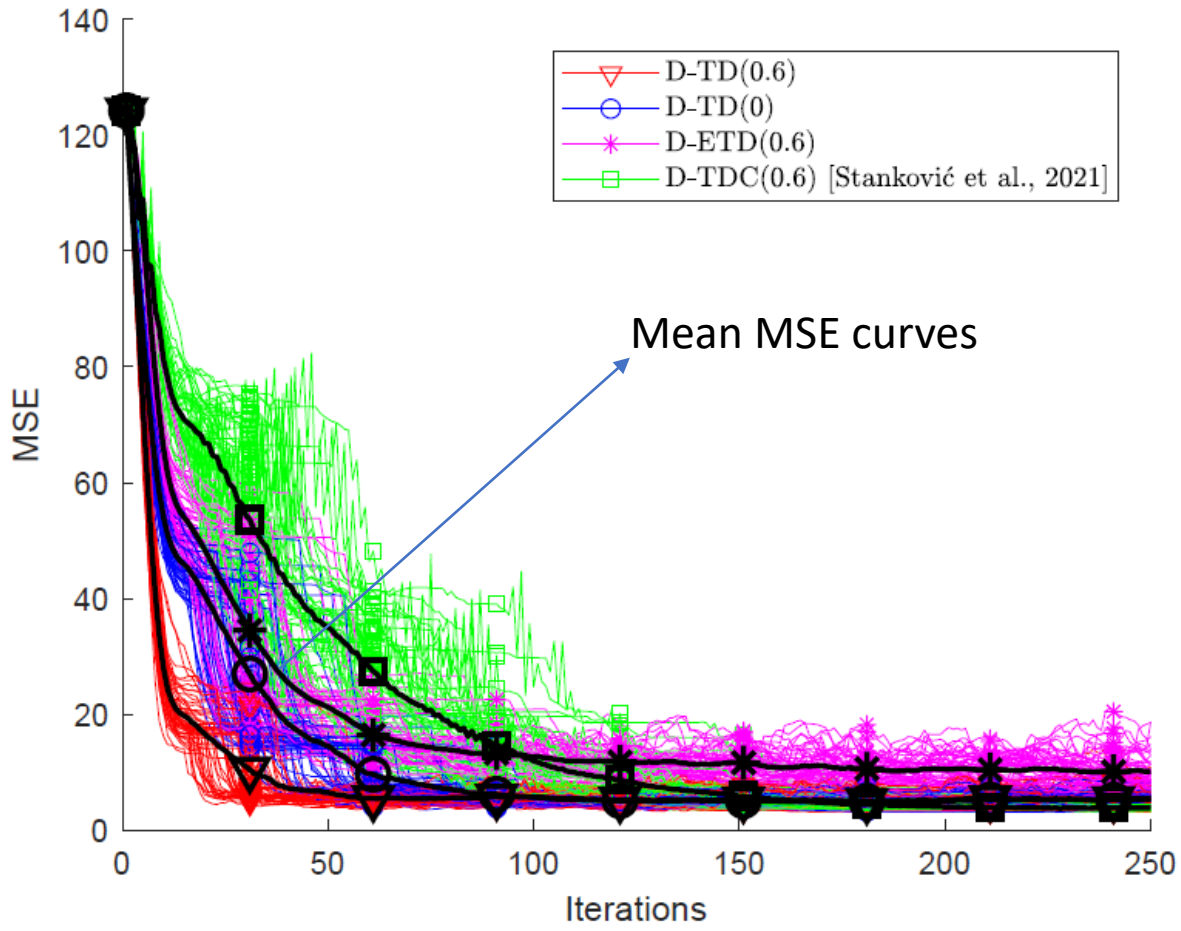

- $p(\text{moving}, a^{highway}) = \frac{1}{\text{state}}$ , $p(\text{stuck}, a^{highway}) = 1 - \frac{1}{\text{state}}$
- $p(\text{moving}, a^{exit}) = 0.8$ , $p(\text{stuck}, a^{exit}) = 0.2$
- $R(a^{exit}) = -4$, $R(a^{exit}) = -1$

- Target policy: $p(a^{exit}) = 0.8$
- $N = 10$ agents
- Behavior policies: $p(a^{exit}) = (0.6, 0.5, 0.9, 0.81, 0.4, 0.67, 0.3, 0.55, 0.45, 0.6)$
- 7 features – Gaussian radial basis distances to states 1,3,5,7,9,11 and 13

# Variance and rate of convergence comparison

- 50 Monte Carlo simulations
- $\lambda = 0.4$
- Average variance of D-TD(λ) : 0.93
- Average variance of D-ETD(λ) is 1.46

- Conclusion: D-TD($\lambda$) has **smaller variance and faster rate of convergence**

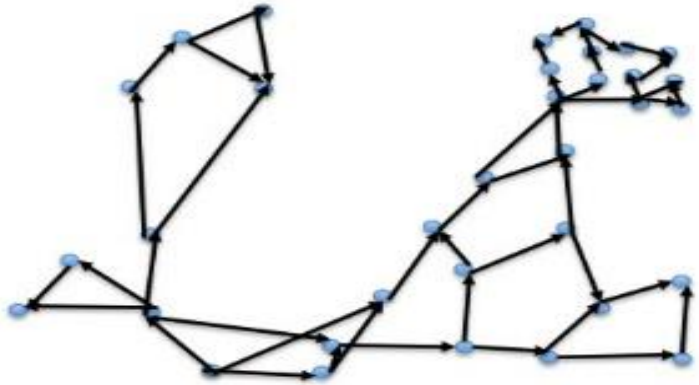


# Multi-Task Policy Optimization

- Q-learning (similar to policy evaluation)

- Actor-Critic (much better convergence properties)

# Problem setup



- $N$ MDPs characterized by $\{S, A, p^i(s'|s,a), R_i(s,a,s')\}$

Finite set of states

Finite set of actions

Transition probabilities

Random rewards

- Each agent $i$ applies a control policy $\pi^{(i)}(a|s)$
- The **local goal** of each agent: find $\pi^{(i)}(a|s)$ optimizing the **state-value function:**

Discount factors

$$V^{\pi^i,i}(s) = E_{\pi^i}\left\{R^i_{t+1} + \sum_{j=1}^{\infty}\prod_{k=1}^{j}\gamma^i\left(s^i_{t+k}\right)R^i_{t+j+1}|s^i_t = s\right\}$$

- **Off-policy** setup – each agent interacts with its environment using a local behavior policy $\pi^i_b$
- Each agent can only observe **local state transitions and rewards**
- **Inter-agent communication** is allowed according to the given network topology
- **Cooperative reinforcement learning**

# Value and Policy Function Approximations

- State space is typically large ⇒ value and policy function parameterizations
- *Critic stage – linear parameterization:*

$$V_{\theta^i}^i(s) = \theta^{iT}\varphi^i(s)$$

Local *features* $\varphi^i(s) \in \mathcal{R}^{L_\theta}$

Local *parameter vector*

- Typically $L_\theta << M$
- *Actor stage*:
- Policy $\pi^i = \pi_{w^i}$ parameterized using the *policy parameter vector* $w^i \in \mathcal{R}^{L_w}$ $L_w << M$
- Off-policy scenario – importance ratio: $\rho_t^i = \pi_{w^i}\left(a_t^i \middle| s_t^i\right) / \pi_b\left(a_t^i \middle| s_t^i\right)$

# Local objectives

- The *expected linear approximation* of the *local value function*

$$J^i\left(\theta^i\left(w^i\right)\right) = \theta^{iT} E_i\{\varphi_t^i\} = \theta^{iT} \sum_s d_b^i(s)\varphi^i(s)$$

Steady state distribution parameters

- Locally optimal values are $w^{i*} = \text{Argmax}_{w^i} J^i\left(\theta^i\left(w^i\right)\right), i = 1, \ldots, N$

# Multi-Agent Objective

- *Multi-objective optimization*
- *Global objective function:*

Utility parameter vector
$\dim(c) = N, 0 \leq c^i \leq 1, \sum_i c^i = 1$

$$J(w^1, \ldots, w^N; c) = \sum_{i=1}^{N} c^i J^i \left(\theta^i\left(w^i\right)\right)$$

- The goal is to learn a *single policy* that performs optimally for the averaged tasks
- The common policy function:

$$\pi^1_{w^*} = \cdots = \pi^N_{w^*} = \pi_{w^*}$$

# Proposed Critic algorithm 1 (ETD($\lambda$))

step size $\alpha_t^i > 0$ (fast time scale)

$$\tilde{\theta}_t^i = \theta_t^i + \alpha_t^i \rho_t^i \delta_t^i \varepsilon_t^i$$

$$\delta_t^i = R_{t+1}^i + \gamma^i \theta_t^{iT} \varphi_{t+1}^i - \theta_t^{iT} \varphi_t^i$$

$$\varepsilon_t^i = m_t^i \varphi_t^i + \gamma^i \lambda^i e_{t-1}^i$$

$$m_t^i = \lambda^i + \left(1 - \lambda^i\right) q_t^i$$

$$q_t^i = 1 + \gamma^i \rho_{t-1}^i q_{t-1}^i$$

# Proposed Critic algorithm 2 (GTD(1))

step size $\alpha_t^i > 0$ (fast time scale)

$$\tilde{\theta}_t^i = \theta_t^i + \alpha_t^i \rho_t^i \delta_t^i e_t^i$$

$$\delta_t^i = R_{t+1}^i + \gamma^i \theta_t^{iT} \varphi_{t+1}^i - \theta_t^{iT} \varphi_t^i$$

$$e_t^i = \varphi_t^i + \gamma^i \rho_{t-1}^i e_{t-1}^i$$

$$\rho_t^i = \pi_{w^i}(a_t^i | s_t^i) / \pi_b(a_t^i | s_t^i)$$

# Proposed Actor 1

- Derived using exact policy gradient, assuming ETD($\boldsymbol{\lambda}$) for the Critic
- The policy gradient can be derived as:

$$\nabla_{w^i} J^i(w^i) = \lim_{t\to\infty} E_i\{\rho_t^i \delta_t^i \tilde{\varepsilon}_t^i)\}$$

$$\tilde{\varepsilon}_t^i = \tilde{f}_t^{\lambda^i,i} \nabla_{w^i} \log \pi_{w^i}(a_t^i|s_t^i) + \tilde{\mu}_t^i + \gamma^i \rho_{t-1}^i \, \tilde{\varepsilon}_{t-1}^i$$

$$\tilde{\mu}_t^i = \tilde{f}_t^{\lambda^i,i} + \gamma^i \rho_{t-1}^i [\, (\tilde{m}_{t-1}^i - \lambda^i) \nabla_{w^i} \log \pi_{w^i}(a_t^i|s_t^i) + \tilde{\mu}_{t-1}^i$$

$$\tilde{f}_t^{\lambda^i,i} = \tilde{m}_t^i + \gamma^i \rho_{t-1}^i \lambda^i \tilde{f}_{t-1}^{\lambda^i,i}$$

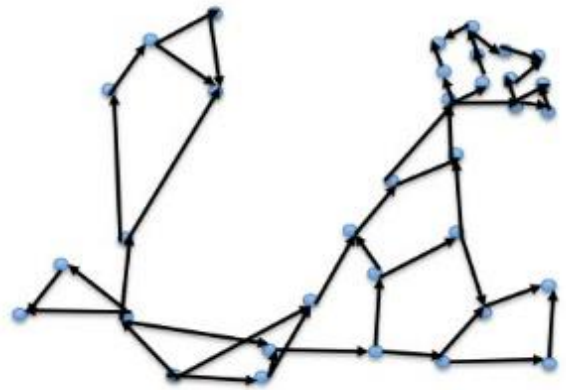
$$\tilde{m}_t^i = 1 + \gamma^i \rho_{t-1}^i (\tilde{m}_{t-1}^i - \lambda^i)$$

# Proposed Actor 1

step size $\beta_t^i \ll \alpha_t^i$ (slow time scale)

- Part 1 (local updates):

$$\widetilde{w}_t^i = w_t^i + \beta_t^i \delta_t^i \tilde{\varepsilon}_t^i$$



- Part 2 (distributed consensus):

$$w_{t+1}^i = \sum_{j \in \mathcal{N}_i} a_t^{ij} \, \widetilde{w}_t^j$$

- $a_t^{ij}$ elements of an $N \times N$ row-stochastic **random** matrix $A_t = \left[a_t^{ij}\right]$
- $a_t^{ij} = 0$ if Agent $j$ does not communicate with Agent $i$ at time step $t$

- The algorithm asymptotically provides consensus

$$w^1 = \cdots = w^N = w^*$$

# Proposed Actor 2

- Derived using exact policy gradient, assuming GTD(1) for the Critic
- The policy gradient can be derived as:

$$\nabla_{w^i} J^i(w^i) = \lim_{t\to\infty} E_i\{\rho_t^i \delta_t^i \tilde{e}_t^i)\}$$

$$\rho_t^i = \pi_{w^i}(a_t^i|s_t^i)/\pi_b(a_t^i|s_t^i)$$

$$\delta_t^i = R_{t+1}^i + \gamma^i \theta_t^{iT} \varphi_{t+1}^i - \theta_t^{iT} \varphi_t^i$$

$$\tilde{e}_t^i = f_t^i \nabla_{w^i} \log \pi_{w^i}(a_t^i|s_t^i) + \gamma^i \rho_{t-1}^i \, \tilde{e}_{t-1}^i$$

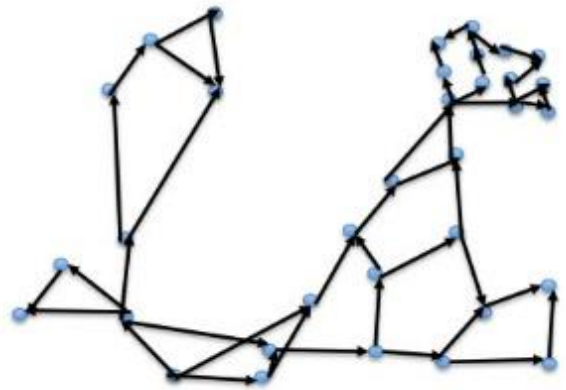
$$f_t^i = 1 + \gamma^i \rho_{t-1}^i \lambda^i f_{t-1}^i$$

# Proposed Actor 2

step size $\beta_t^i \ll \alpha_t^i$ (slow time scale)

- Part 1 (local updates):

$$\widetilde{w}_t^i = w_t^i + \beta_t^i \rho_t^i \delta_t^i \tilde{e}_t^i$$



- Part 2 (distributed consensus):

$$w_{t+1}^i = \sum_{j \in \mathcal{N}_i} a_t^{ij} \, \widetilde{w}_t^j$$

- $a_t^{ij}$ elements of an $N \times N$ row-stochastic **random** matrix $A_t = \left[a_t^{ij}\right]$
- $a_t^{ij} = 0$ if Agent $j$ does not communicate with Agent $i$ at time step $t$

- The algorithm asymptotically provides consensus

$$w^1 = \cdots = w^N = w^*$$

# Convergence

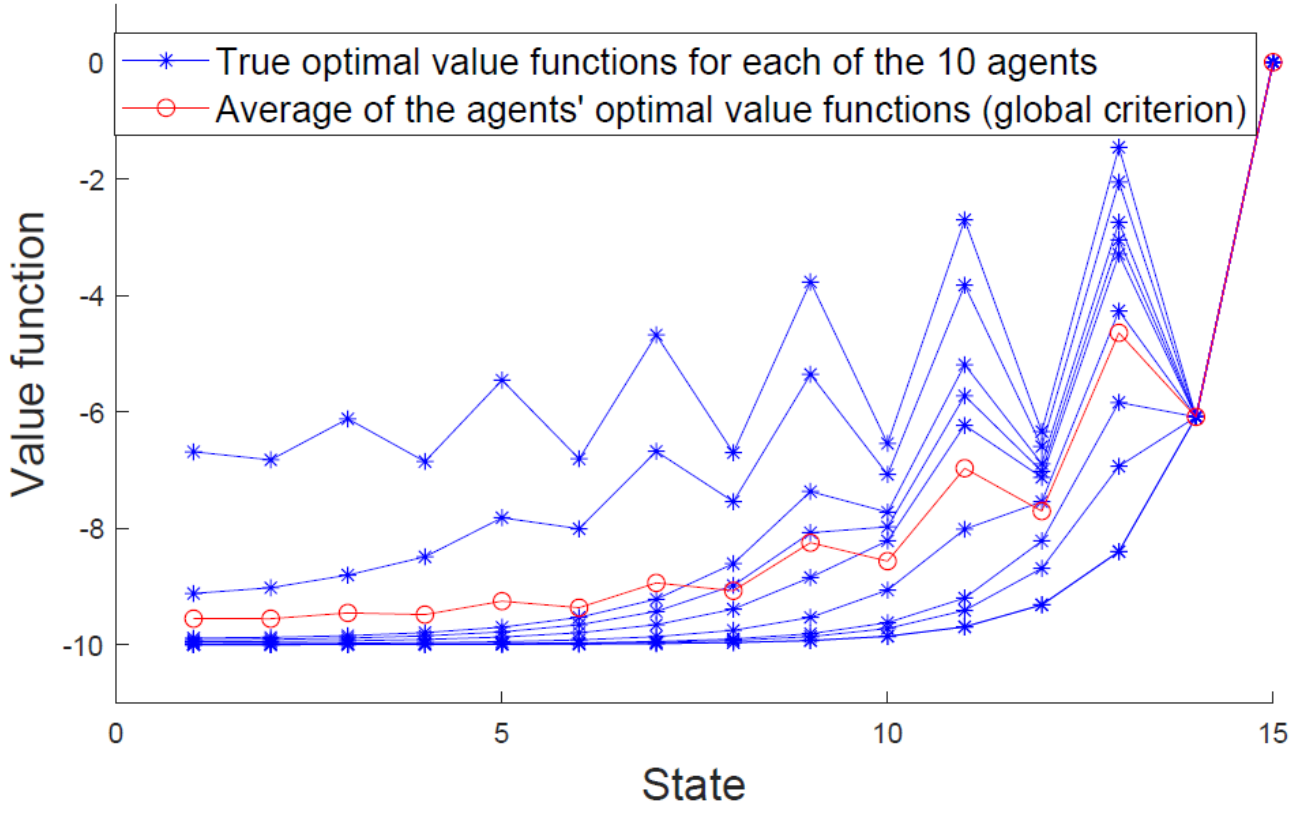
- Rigorous proofs of stochastic convergence:
  - M.S. Stankovic, M. Beko, N Ilic and S.S. Stankovic. **Multi-Agent Off-Policy Actor-Critic Algorithm for Distributed Multi-Task Reinforcement Learning,** *European Control Conference, ECC 2023*, 2023.
  - M.S. Stankovic, M. Beko, N Ilic and S.S. Stankovic. **Multi-Agent Actor-Critic Multitask Reinforcement Learning based on GTD(1) with Consensus**, *61st IEEE Conference on Decision and Control, CDC 2022*, Cancun, Mexico, 2022.
  - M.S. Stankovic, M. Beko and S.S. Stankovic. **Convergent Distributed Actor-Critic Algorithm Based on Gradient Temporal Difference**, *30th European Signal Processing Conference, EUSIPCO 2022*, Belgrade, Serbia, 2022.
  - …

# Main Properties and Benefits

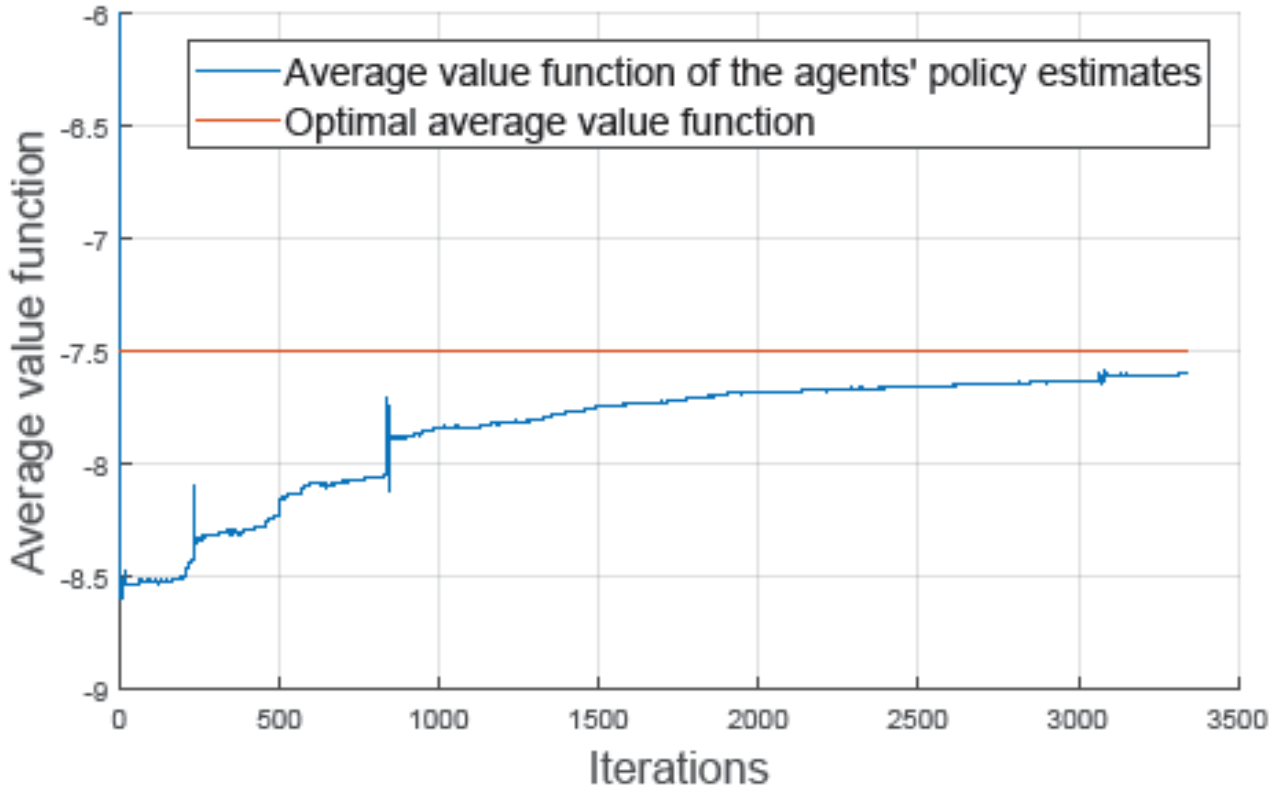
- **Fully decentralized** (only local state transitions and rewards observations)
- Tool **for organizing/fusing coordinated actions** (behavior policies, eligibility traces), speeding-up/enabling convergence by exploiting agents' complementarities
- Tool for **parallelization**, speeding up the convergence by **reducing the overall variance** which has two sources:
  - underlying **MC stochastic dynamics**
  - possible presence of white noise **noise in the one-step rewards**
- Tool for **improvement of the final VF approximation precision** (error with respect to the true VF of the target policy)

# Experiment 1

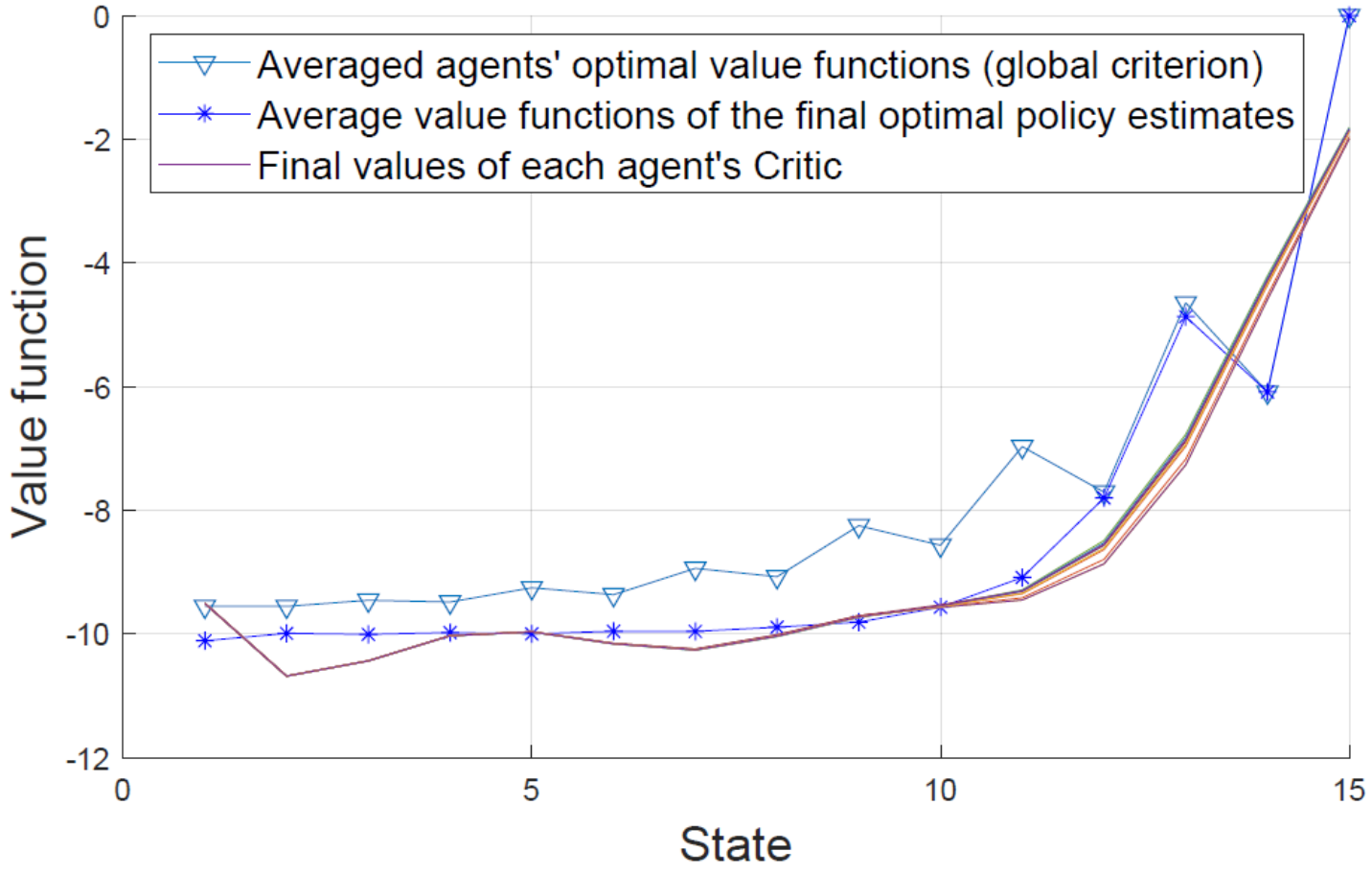
- **Multi task** setup: $p(\text{moving}, a^{exit})$, $p(\text{stuck}, a^{exit})$, $R(a^{exit})$ and $\pi_b(a^{exit}|s)$ differ among the agents
- Actor: Gibbs parameterization with **tabular features**
- Equal local consensus weights
- **Successful convergence** to the optimal policy using our algorithm



# Experiment 1

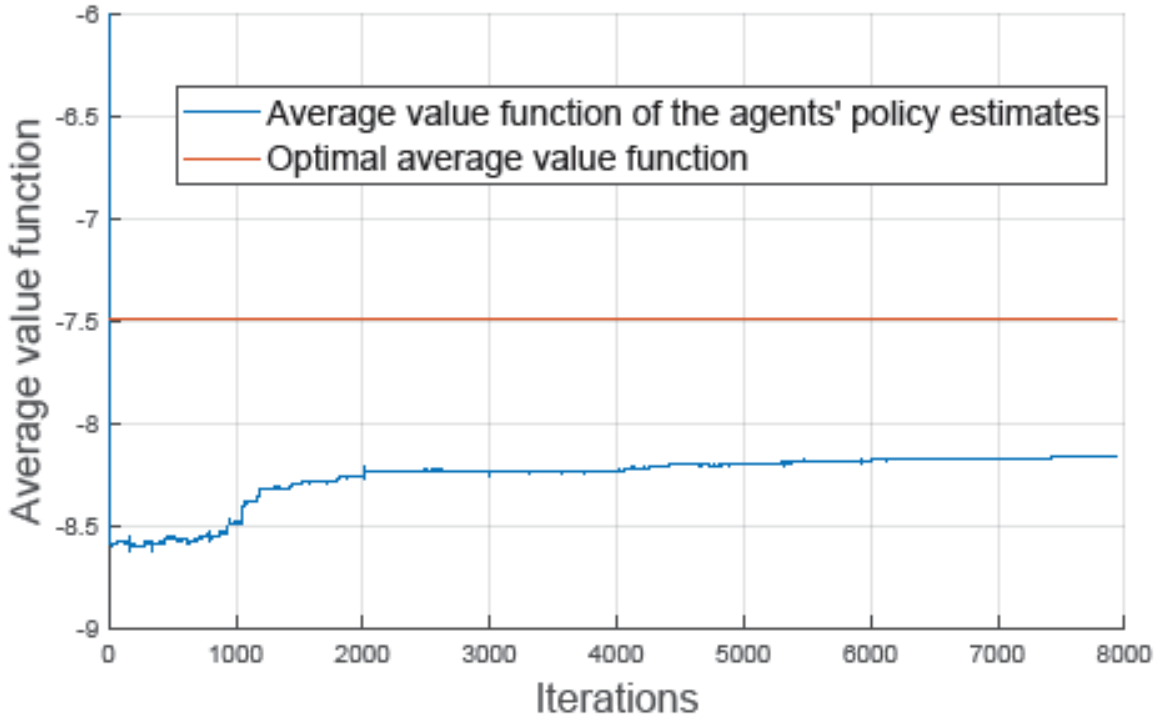


# Experiment 1

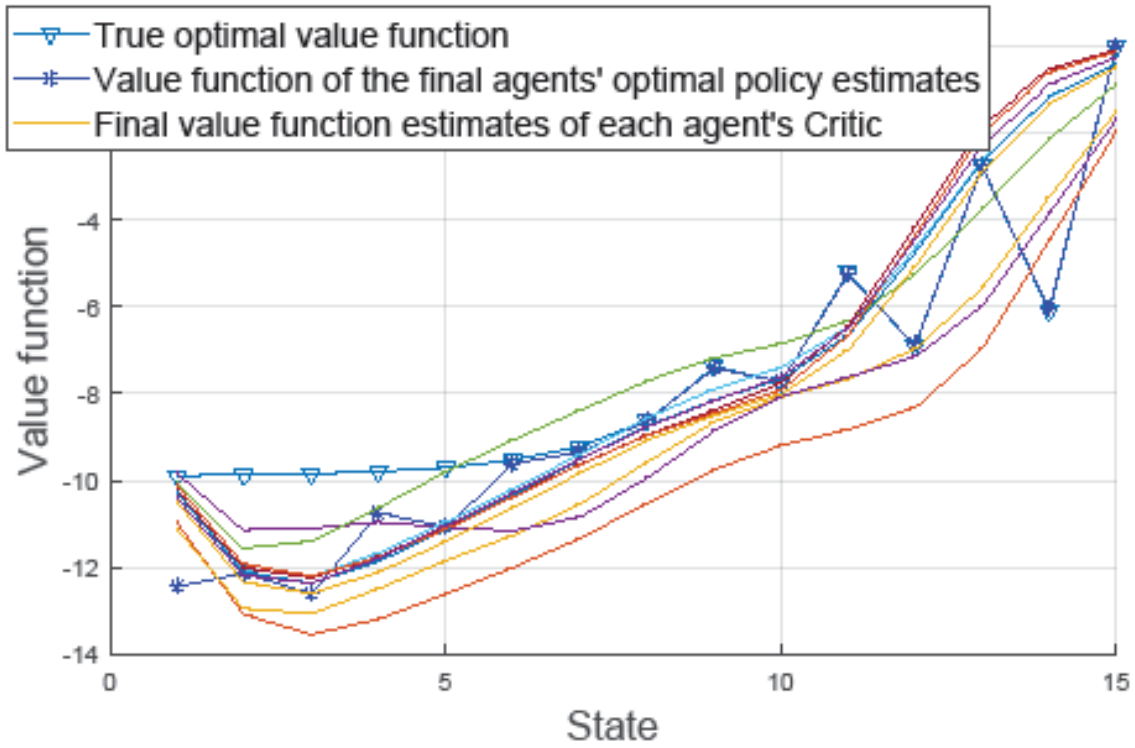


# Experiment 2

- Actor: Gibbs parameterization with **binary features with lower dimensionality**
- The agents are **not restricted** to particular subsets of states
- Convergence to the optimal policy is **not guaranteed**



# Experiment 2



# Drone Swarms Project Summary

- Specific focus: Algorithmic solutions that enable a system of autonomous Micro-Aerial Vehicles (Cooperative Swarm), to **safely perform complex tasks in unknown environments**
- Using: on-board sensors and inter-drone communication
- Automatic accomplishment of environmental mapping, localization, path planning, target detection/tracking and flight control
- Applications:
  - Prevention and assistance with natural disasters (e.g. floods, earthquakes, wildfires etc.)
  - Inspection and maintenance of industrial infrastructure
  - Assistance with search and rescue operations
  - Tracking the spread of disease, etc.

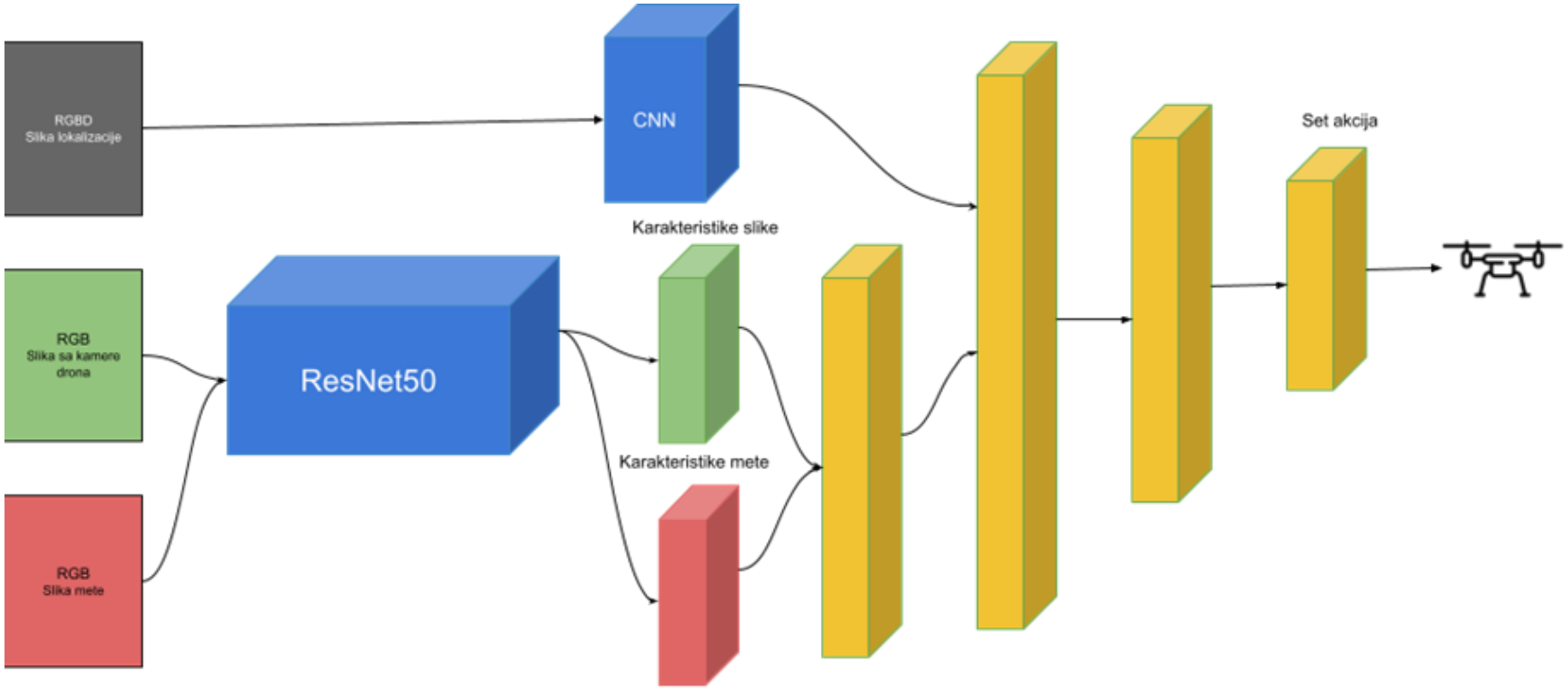
- Advantages:
  - Efficient and accurate mission accomplishment (faster than any human)
  - No risk of physical harm
  - Coordinated actions for mission optimization

# Autonomous drone control for visual search based on deep reinforcement learning

# RL model

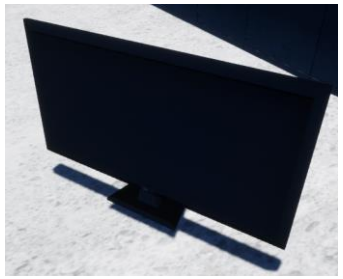
- Based on the DQN algorithm
- Deep NN is used to approximate the state
- Memory of past experience
- Target network and training network
- e-greedy policy
- Input from 3 different sources
- Action space is discrete
- Reward is calculated with object detection algorithm (YOLO) with assistance of feature extraction network (ResNet)

# Deep Q Approximation Neural Network

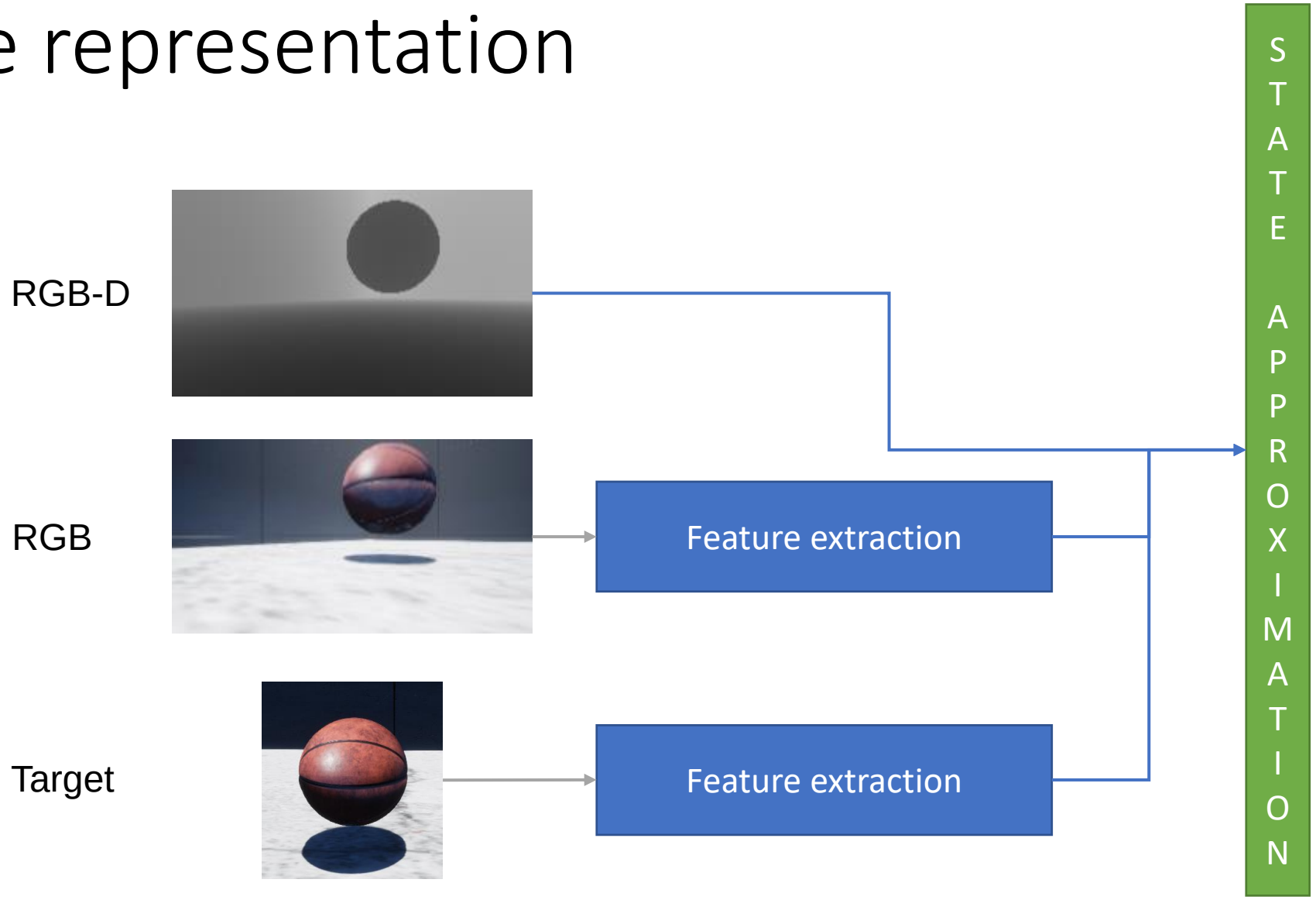


# Environment example

Targets

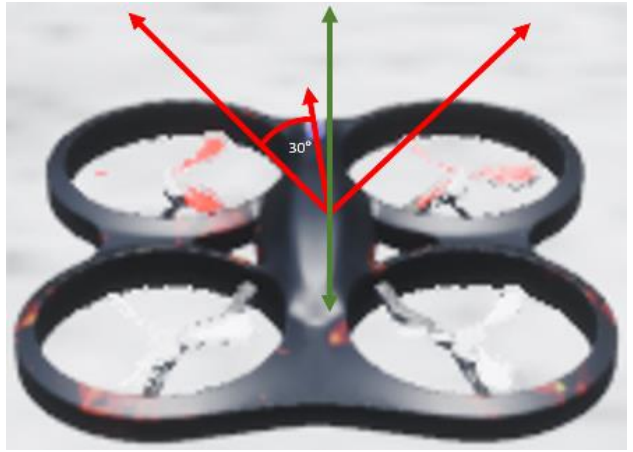


# State representation

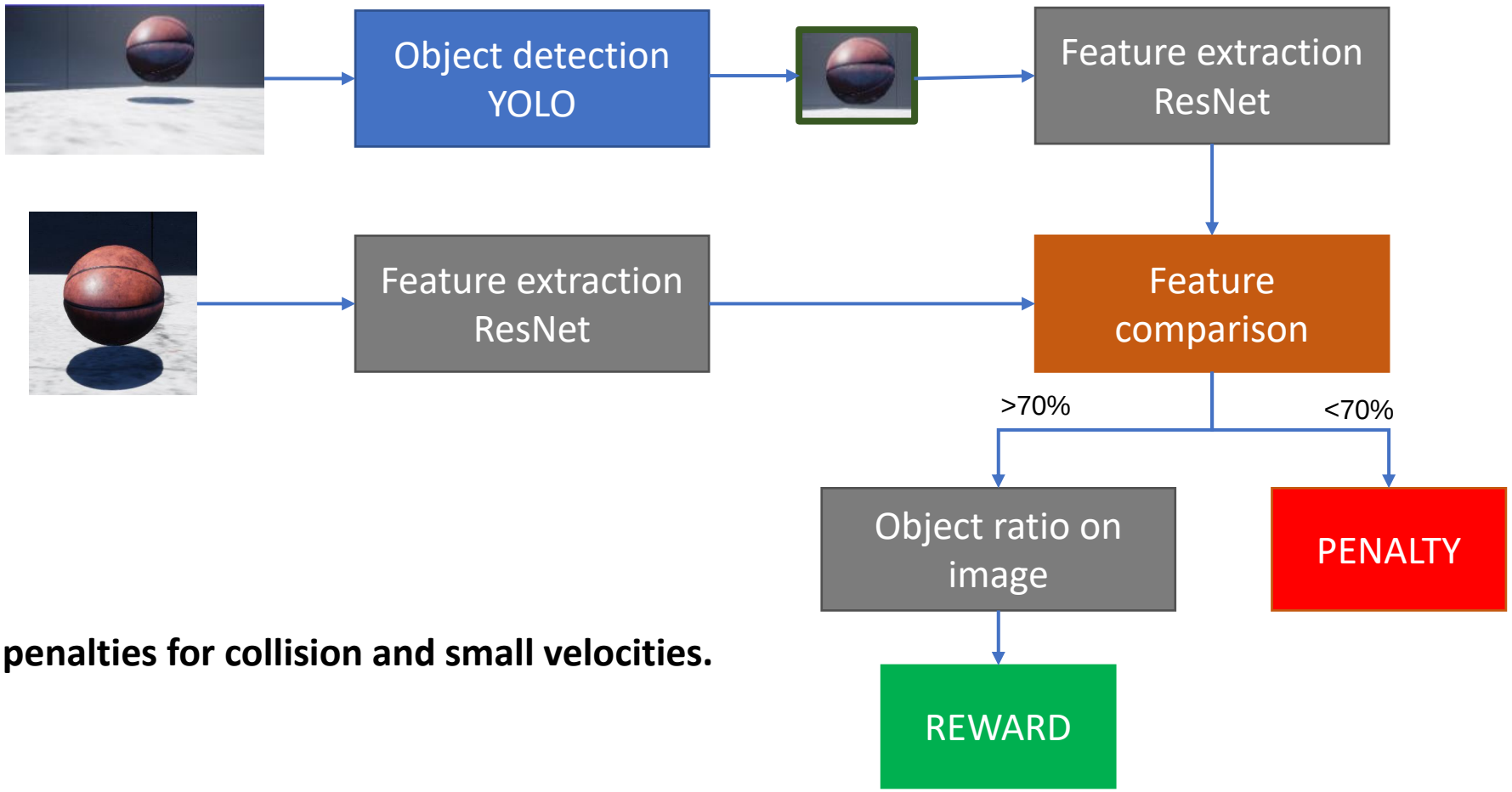


# Actions

- Forward
- Up
- Down
- Left 30°
- Right 30°
- Stop

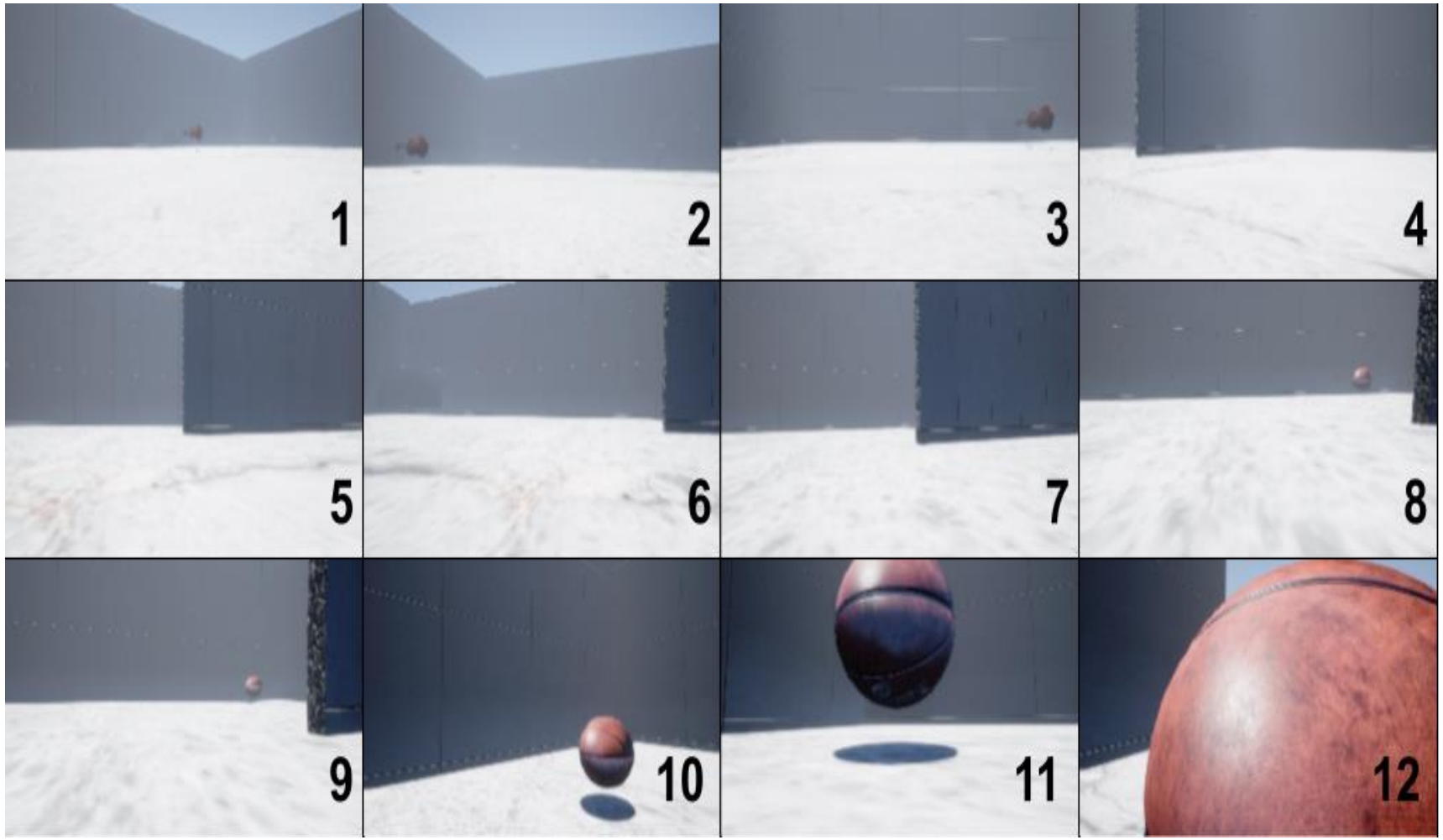


# Rewards



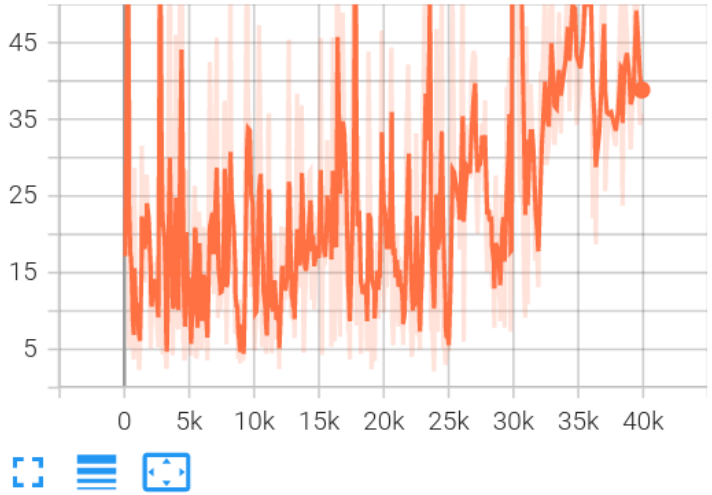

**Large penalties for collision and small velocities.**

# Observation examples

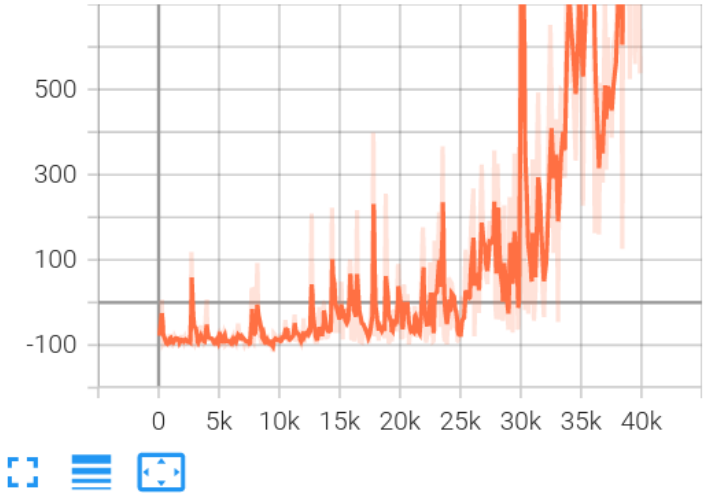


# Reward evolution

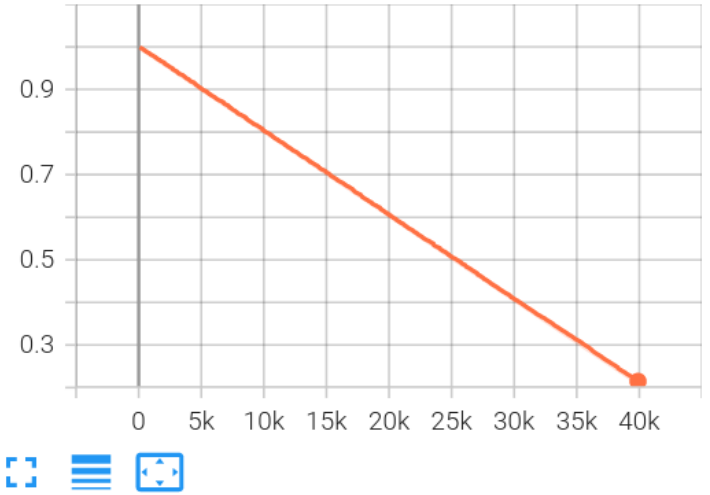
rollout/ep_len_mean
tag: rollout/ep_len_mean




rollout/ep_rew_mean
tag: rollout/ep_rew_mean



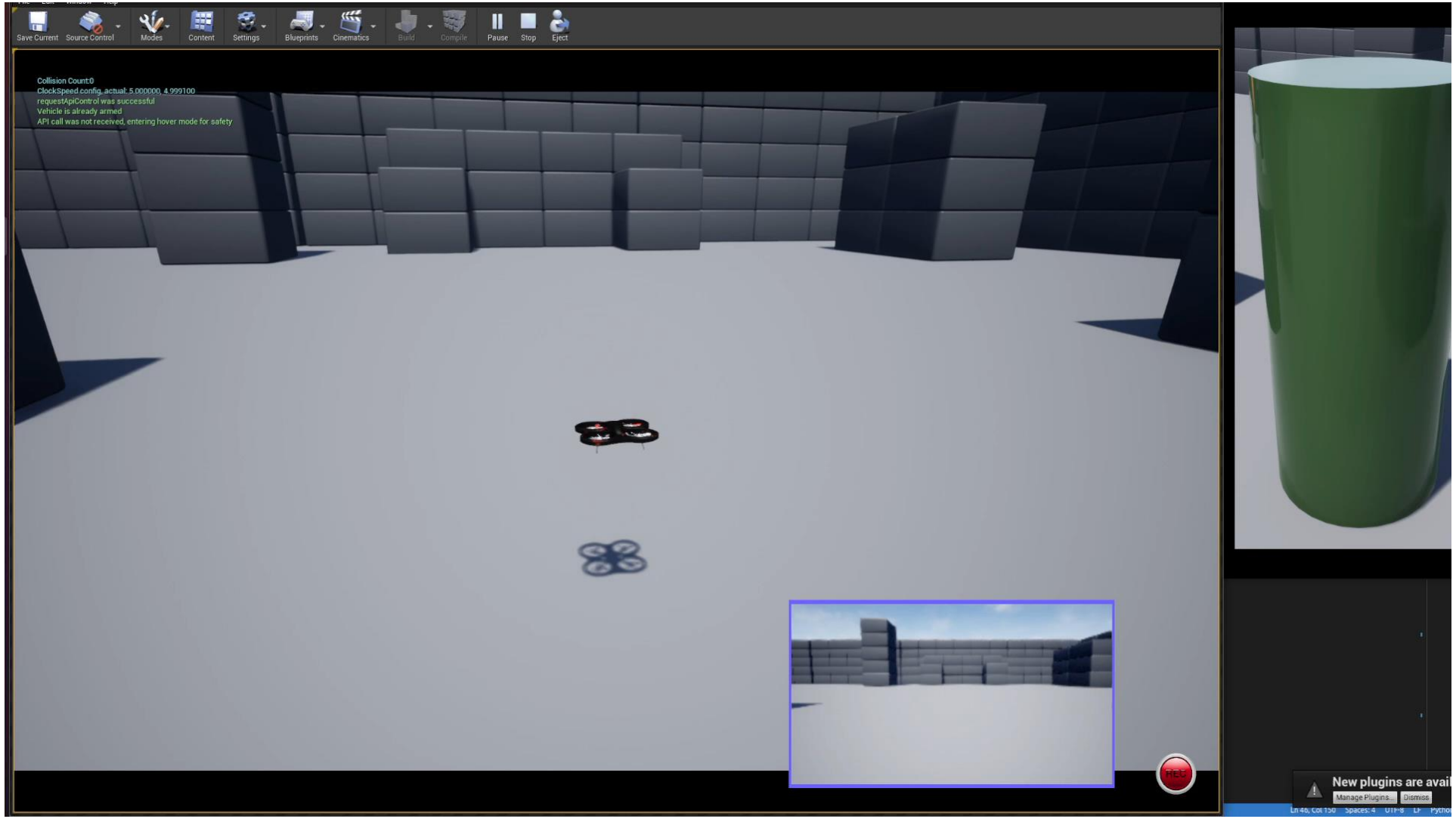

rollout/exploration_rate
tag: rollout/exploration_rate



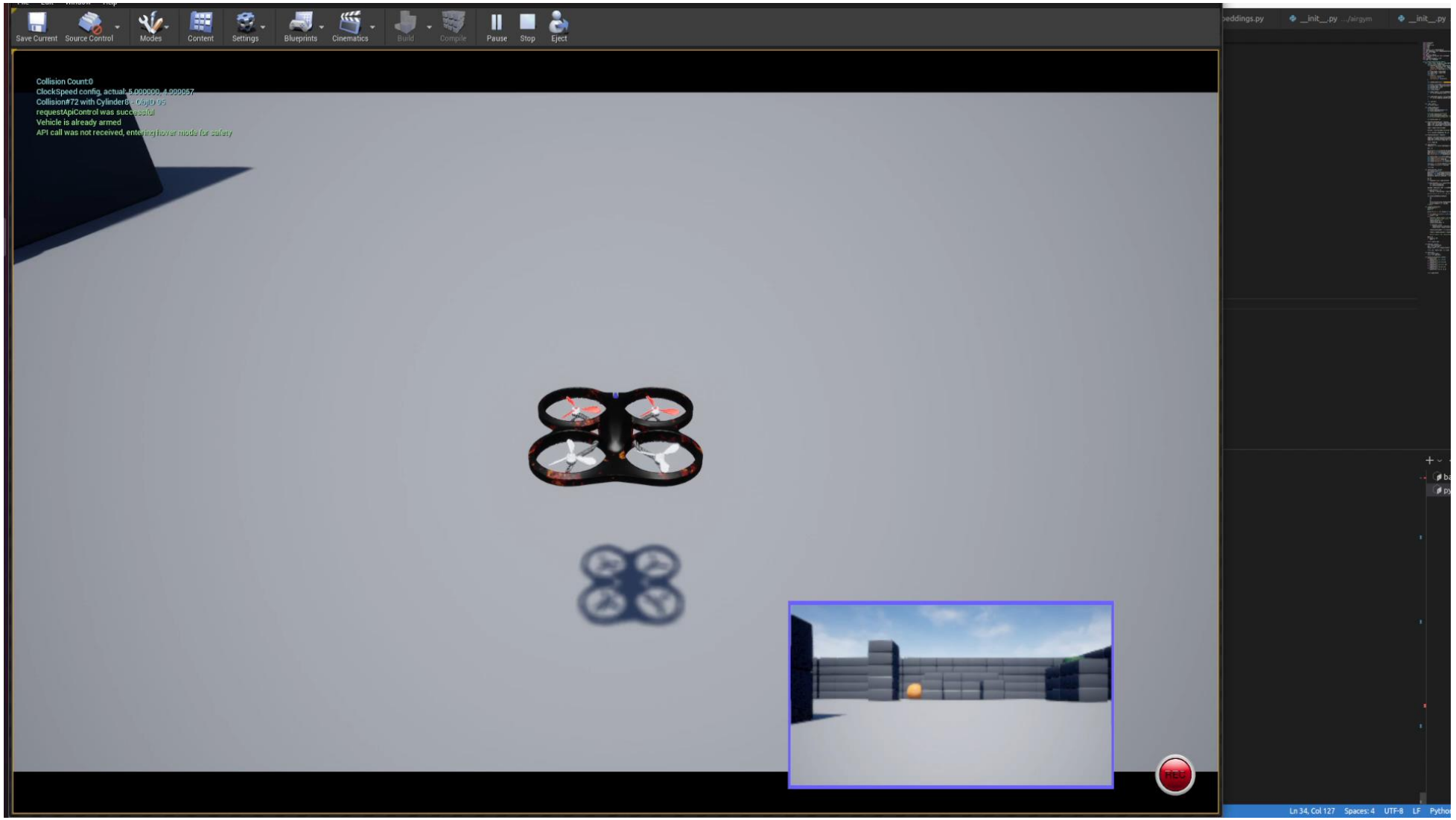

Time steps

Long-time training needed – distribution and decentralization is a necessity

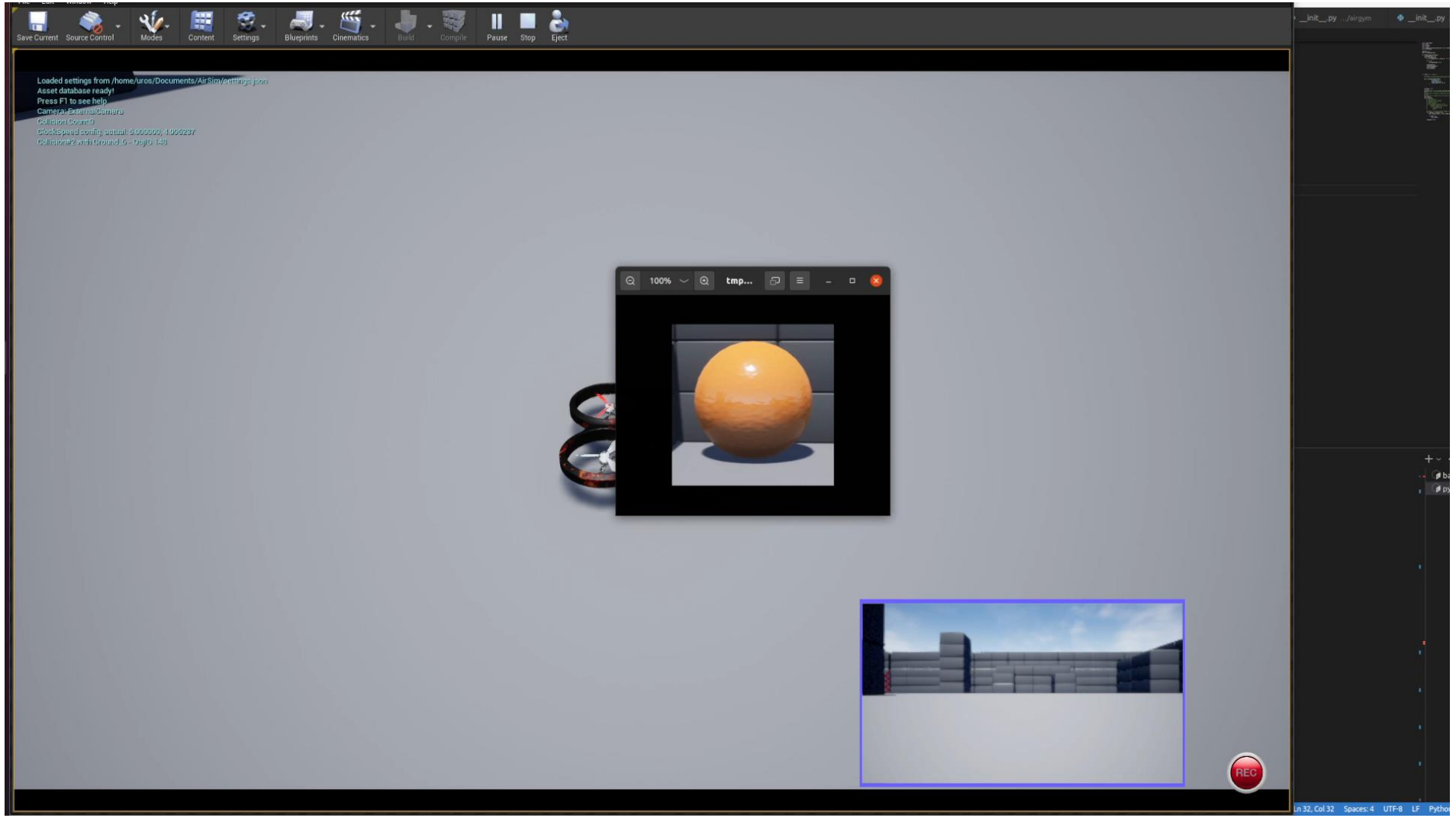
# Example Evaluation Video



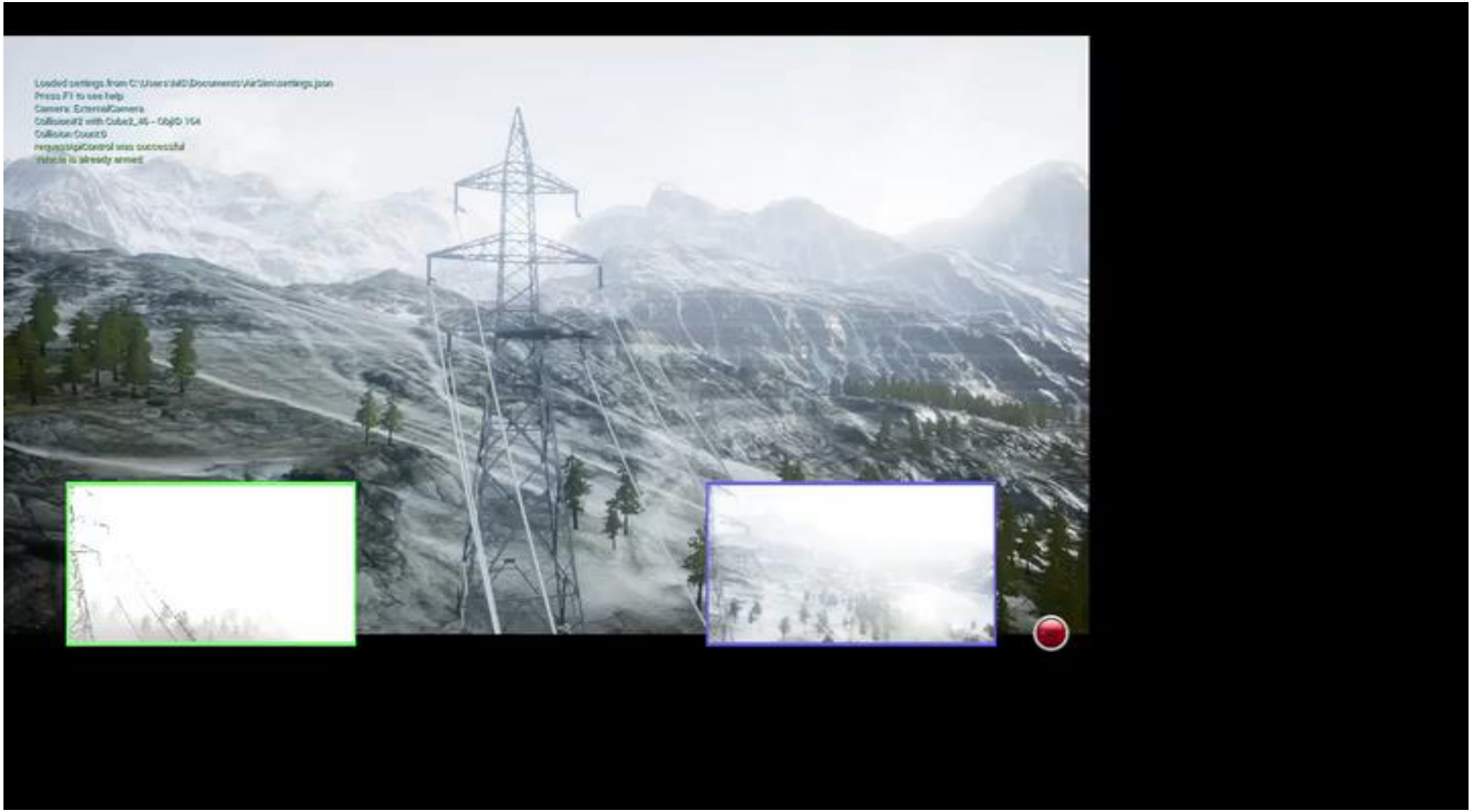
# Example Evaluation Video



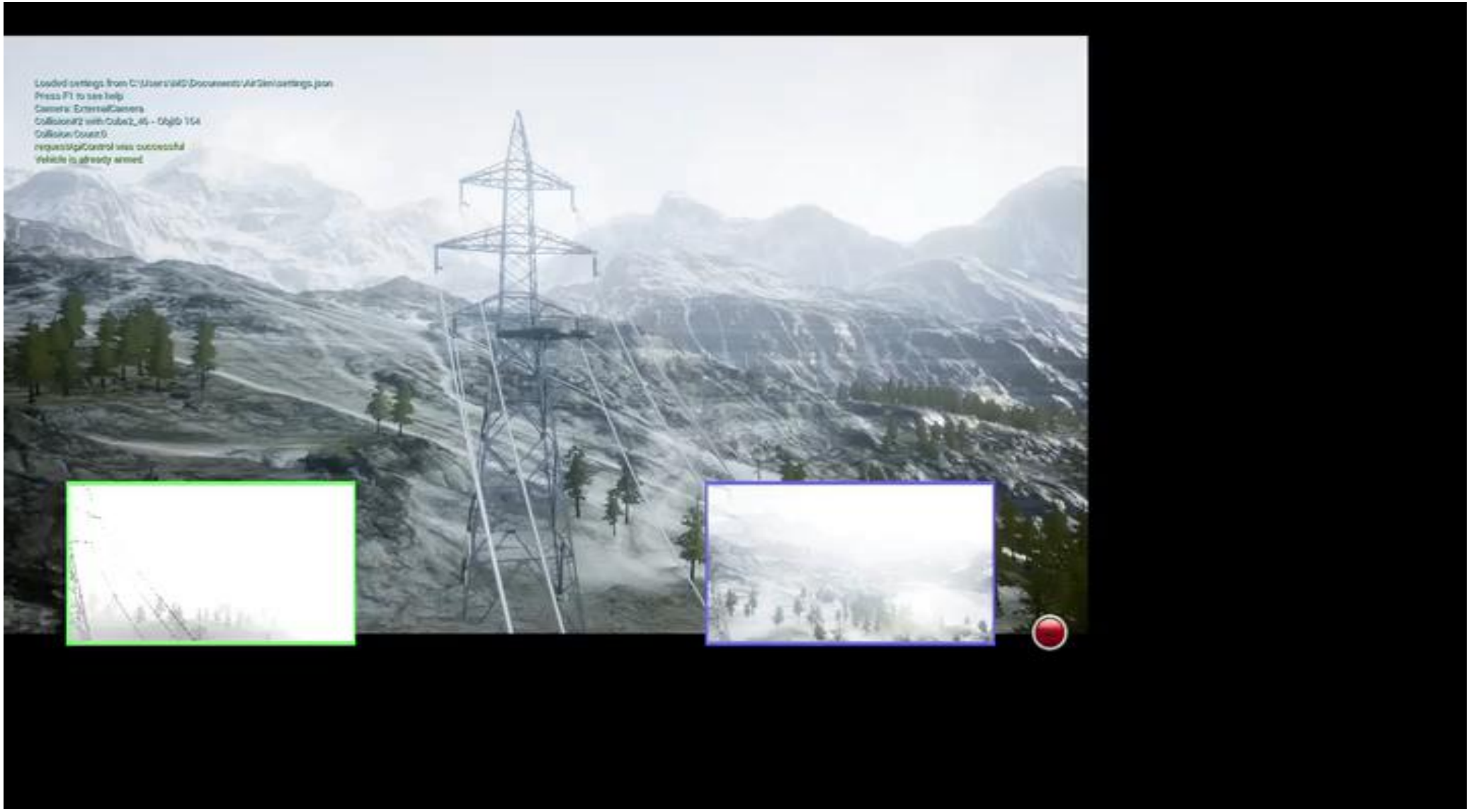
# Example Evaluation Video



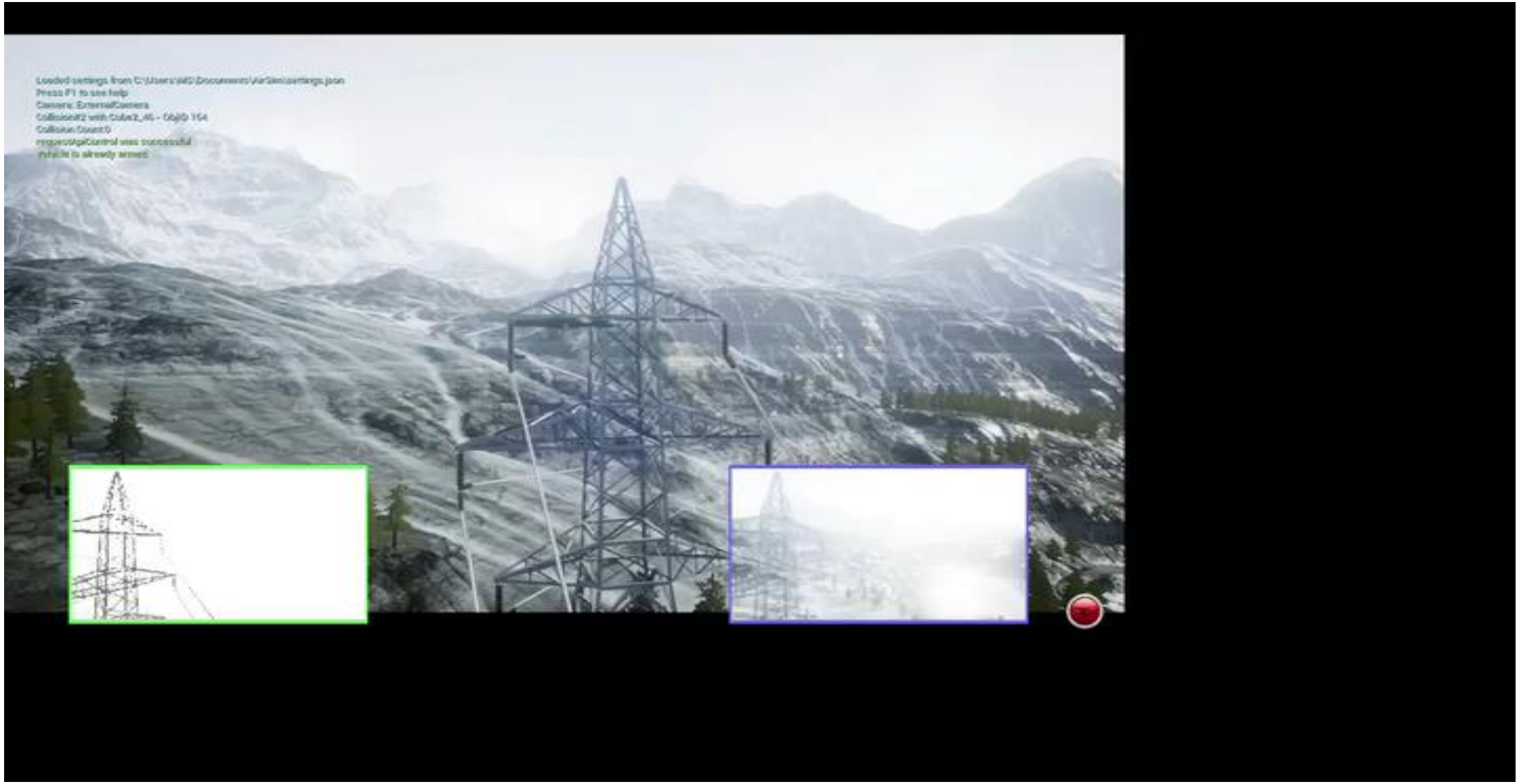
# Power line inspection example



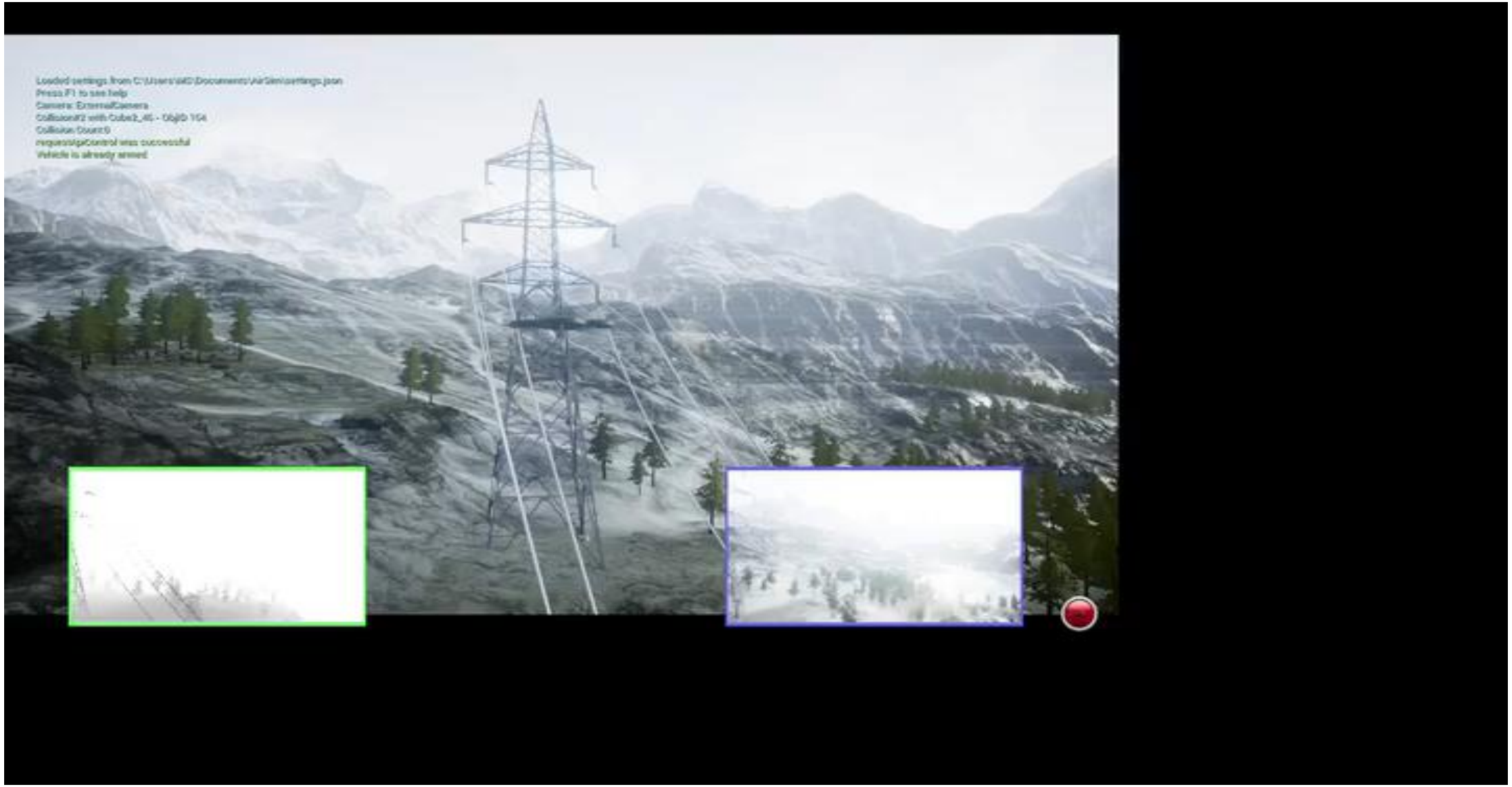
# Power line inspection example



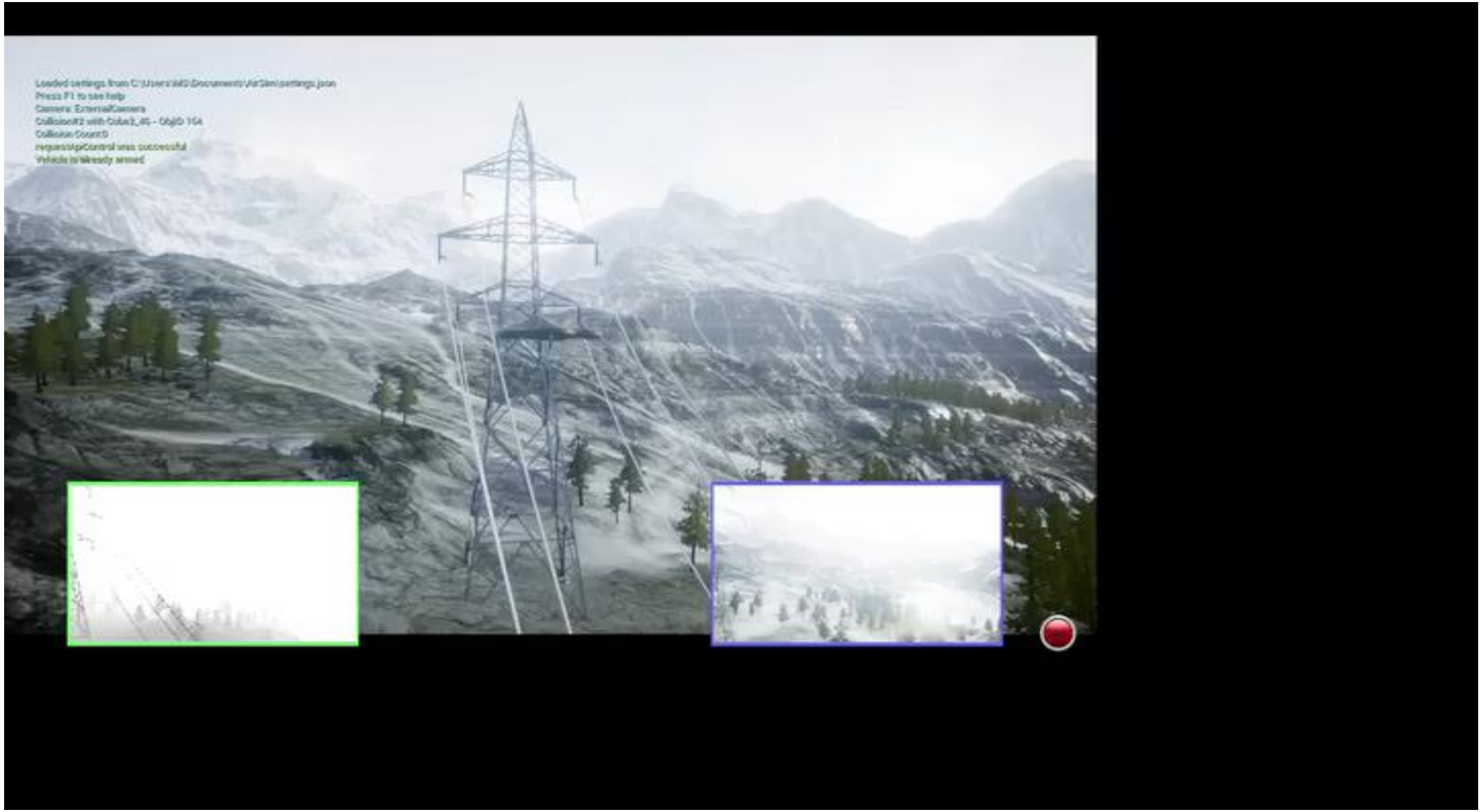
# Power line inspection example



# Power line inspection example



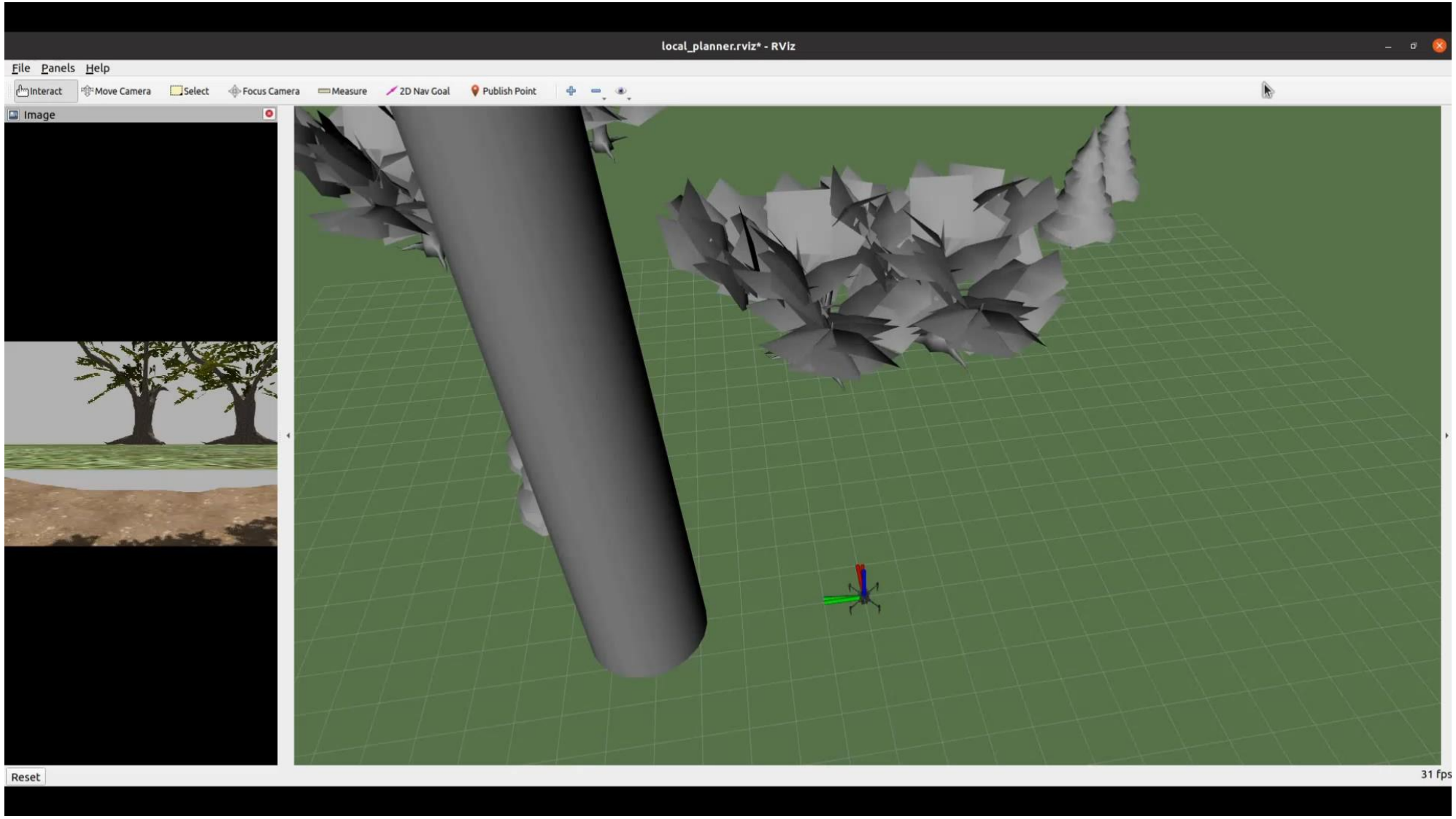
# Power line inspection example



# RL Drawbacks

- Stability with function approximation (especially using DNNs)
- Security (exploration might be risky)
  - Learning from other agents/previous experience
- Sample (in)efficiency –
  - In many cases too much exploration is needed for convergence compared to some other adaptive model based approaches (e.g. adaptive Model Predictive Control)

# Drone wind turbine inspection



# Drones formation control

# Object tracking

# Cloud resources and the OCRE project

# Collaboration

- Institutions involved:
  1. Copelabs, Universidade Lusófona, Lisboa, Portugal
     - The largest private university in Portugal
     - 10 Higher Education Institutes
     - Copelabs: interdisciplinary approach to cyber-physical interconnected systems: **telecommunications and networking; management information systems; data science and artificial intelligence**

  2. Singidunum University
     - The largest private university in Serbia
     - Research in cutting-edge engineering areas of **Artificial Intelligence, Cyber-Physical Systems, Internet of Things, and other modern Computer, Communication and Cryptographic Systems**

# Cloud resources

- The project requires **specialized computing resources** to perform **complex and computationally heavy** processing when dealing with complex CPS including Swarms of Drones:
  - Training of the algorithms
  - Demanding simulators of real-life environments
  - Real-time control and learning (when possible)

# Benefits of cloud resources

- **Access to state-of-the-art technology** (e.g. latest GPU technology and high scale compute technology)
- **Agility** (flexible experiments without investing significant funds and pivot to the types of resources optimal for the specific AI applications)
- **Speed of implementation** (the time savings from not having to acquire, provision, implement and configure the necessary resources)
- **IT know-how** (the research team can focus more on scientific applications, the project does not require IT experts or training)
- **Cost** (no investment in the compute resources, which will soon become obsolete)

# The OCRE call and the mini competition

- The grant was awarded in December 2020

- Mini competition was well organized by OCRE
- The whole process lasted about 2 months
- Our main requirements:
  - **Virtual machines with strong both CPUs and GPUs** (including both visualization and machine learning)

  - Machine learning toolsets (deep learning, reinforcement learning)

- 4 offers by leading global cloud service and infrastructure providers
- Google Cloud–Sparkle has been awarded

# Future plans

- Use efficiently the obtained cloud resources
- The project and future research and exploitation plans are ambitious
- As the project evolves we are expecting a need for more cloud resources
- Consider other calls providing cloud resources for research