

Syrian Arab Republic  
Ministry of higher education  
Tishreen University  
Mechanical & Electronical  
Engineering faculty



# DLRA

## Deep Learning for a Robotic Arm

Final project  
Mechatronics Engineering  
July 2017

# Contents

- Project statement.
- Application and motivations.
- Why deep learning?
- Algorithm discussion.
- Algorithm implementation.
- Simulation experiment.
- Real world experiment.
- Results.
- Development.

# Project statement

- Apply and asses algorithm that give a robotic arm the ability to learn how to achieve a mission inside its work space.



# Applications and motivations

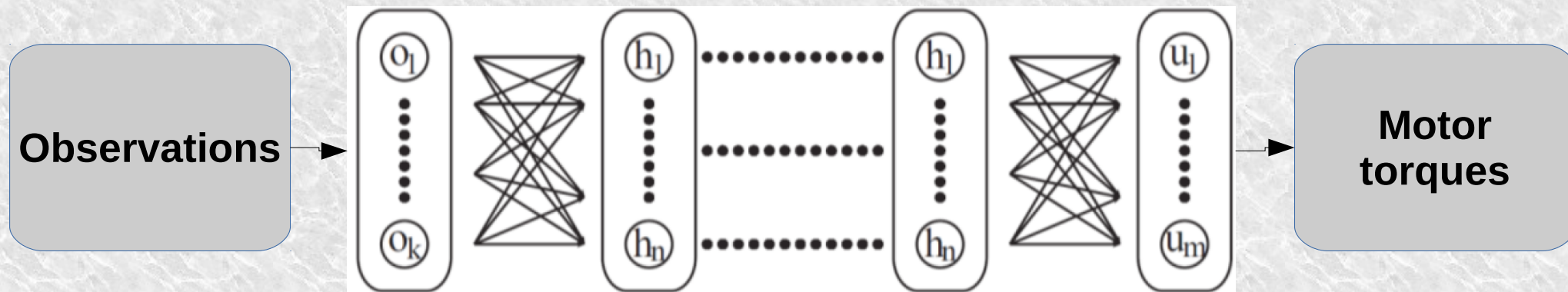
- Robot arms became an essential tool in industrial process.
- Deep robotic learning is an ongoing research topic.
- Robot arms can learn industrial mission like reaching an object, pick and place, welding, etc...

# Why deep learning?

## Standard Robotic Control

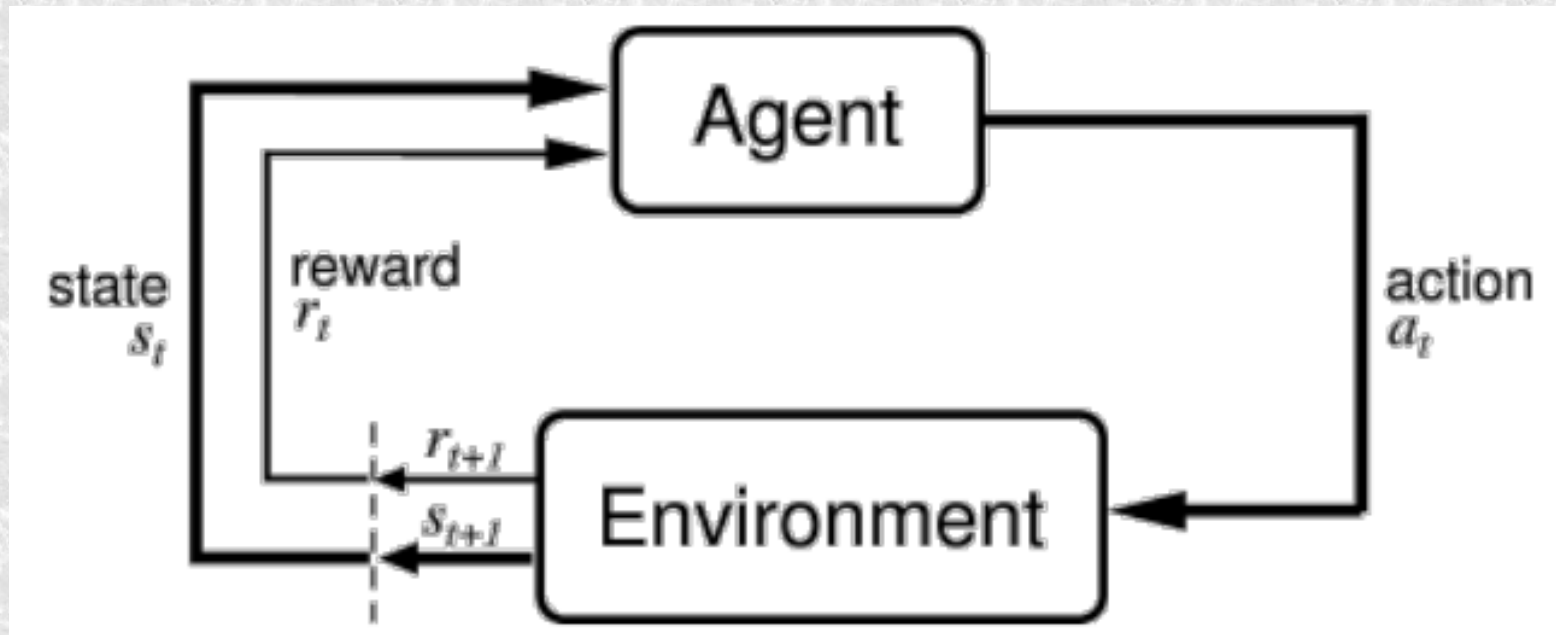


## Deep Robotic Control



# Reinforcement learning

A branch of machine learning



The objective is to maximize the expected reward

# Reinforcement learning

## Markov Decision Process

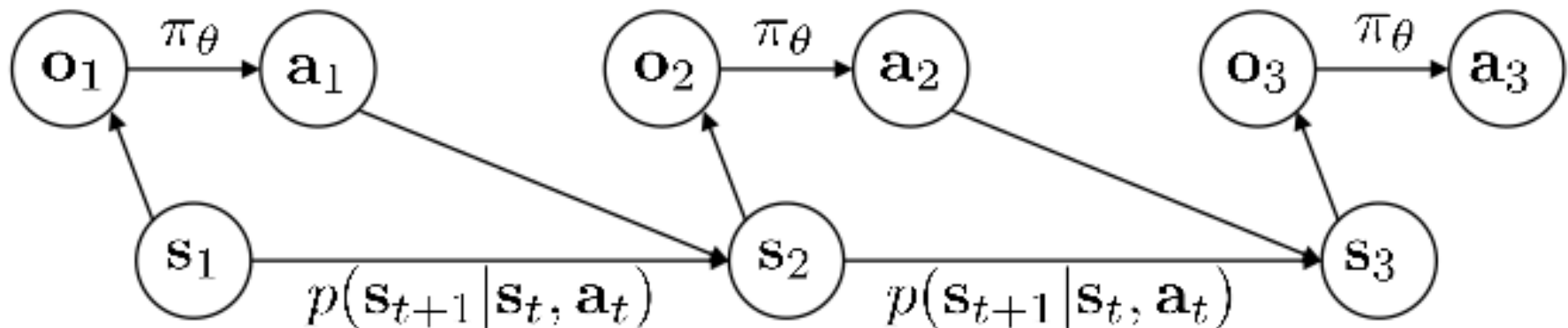
$\mathbf{s}_t$  – state

$\mathbf{o}_t$  – observation

$\mathbf{a}_t$  – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$  – policy

$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$  – policy (fully observed)



# Reinforcement vs Supervised learning

The objective of the learning is to maximize

$$\sum_i \log(p(y_i|x_i))$$

Supervised learning

x – images  
y – labels

$$\sum_i E[r(a_t, s_t)]$$

Reinforcement learning

s – state  
a – actions  
(from actor network)



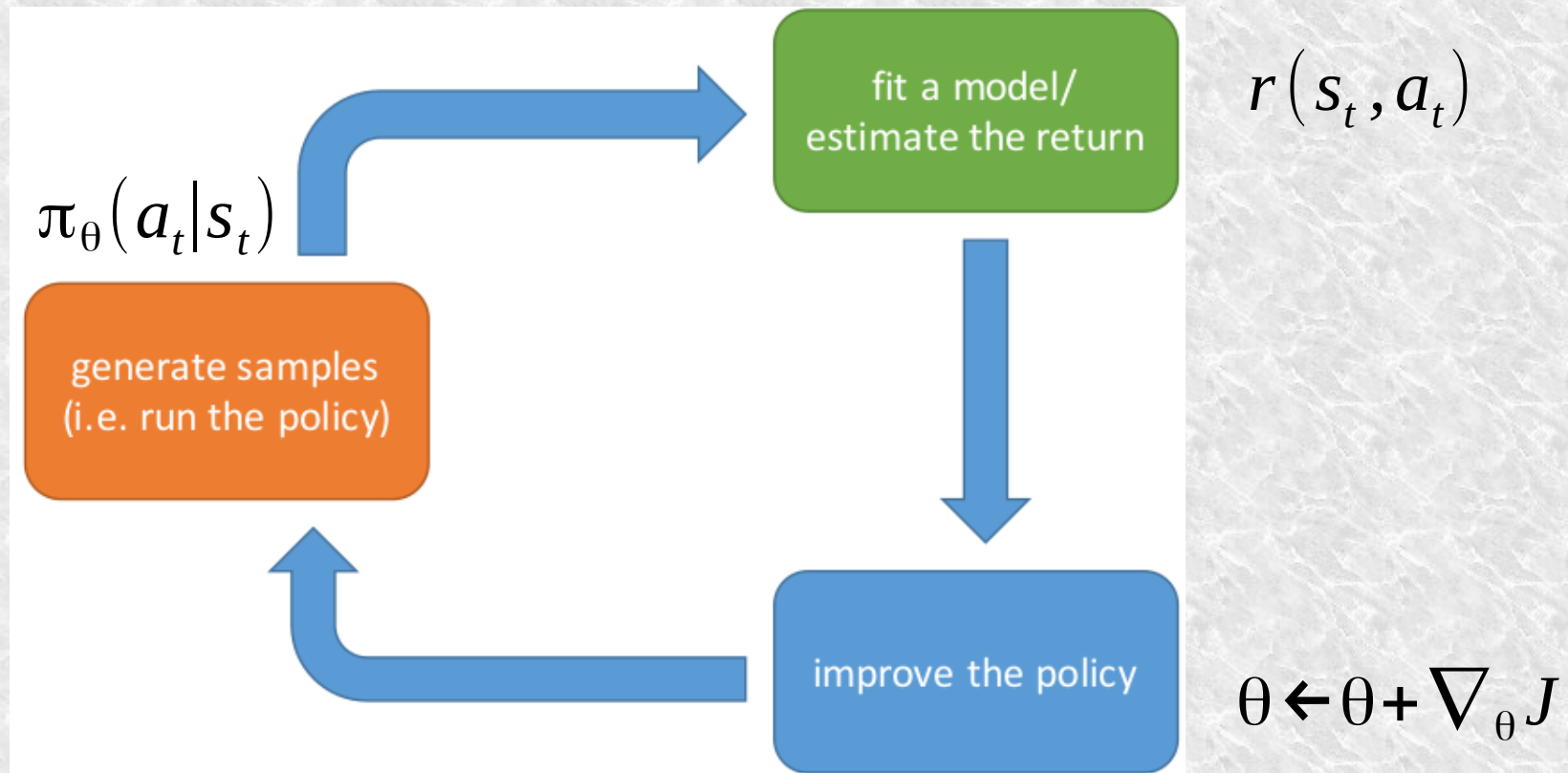
# Algorithm discussion

## **DDPG** **Deep Deterministic Policy Gradients**

Lillicrap'16


**Deep Q learning + Actor-critic = DDPG**

# Policy gradients



$$\nabla_{\theta} J = \sum_i \left( \sum_t \nabla_{\theta} \pi(a_t^i | s_t^i) \right) \left( \sum_t r(s_t^i, a_t^i) \right)$$

# Policy gradients

$$\nabla_{\theta} J = \sum_i \left( \sum_t \nabla_{\theta} \pi(a_t^i | s_t^i) \right) \left( \sum_t r(s_t^i, a_t^i) \right)$$


The diagram illustrates the components of the policy gradient equation. A blue box highlights the two terms in the product:  $\sum_t \nabla_{\theta} \pi(a_t^i | s_t^i)$  and  $\sum_t r(s_t^i, a_t^i)$ . Arrows point from these terms to two separate boxes below: 'Gradient vector of policy' and 'Score (reward) Function'.

## Gradient vector of policy:

gives the direction  
in the parameters space  
to increase the probability of x

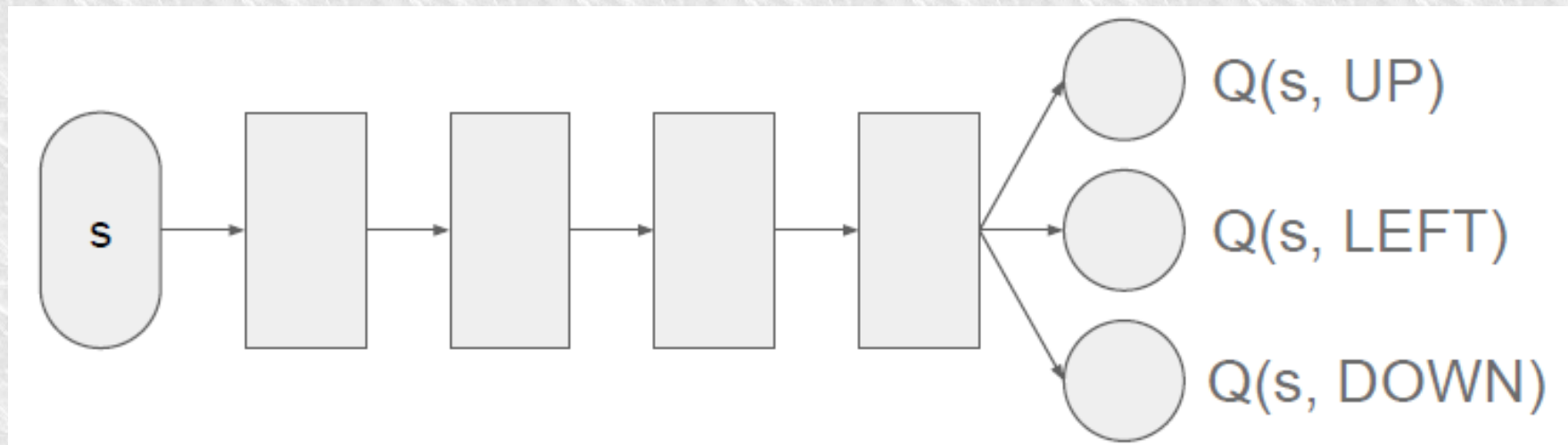
## Score (reward) Function:

by multiplying with score  
samples with higher score  
will have higher expectation

# Deep Q learning

Mnih'13

approximate  $Q(s,a)$  using  
a deep neural network  
 **$Q(s,a)$  is the reward function**

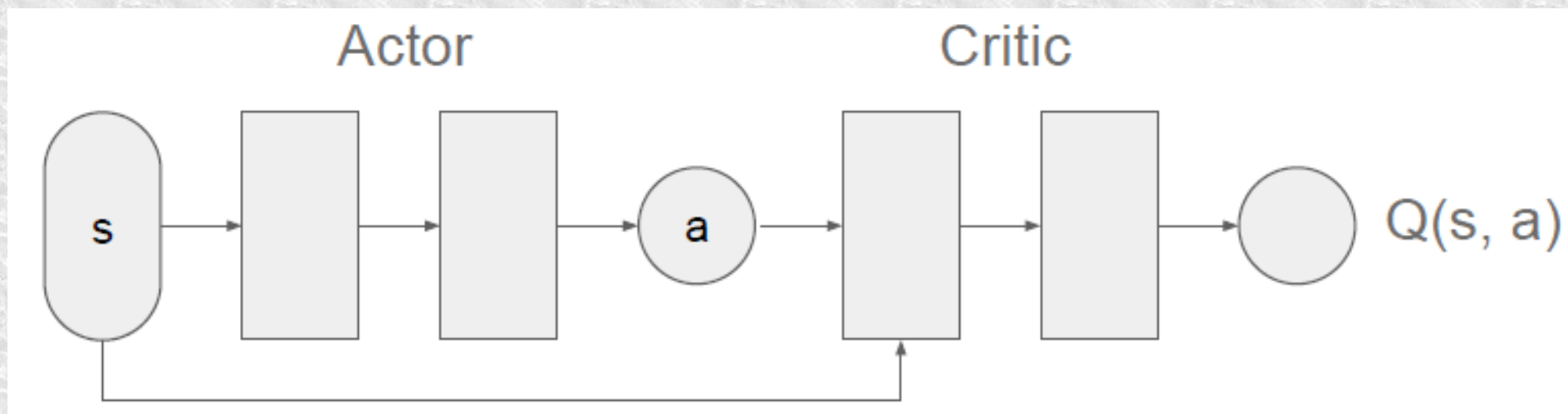


# Actor Critic

Actor implements deterministic policy function

$$a = \mu(s)$$

Evaluate actions using a critic network  $Q(s, a)$



---

**Algorithm 1** DDPG algorithm

---

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .

Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer  $R$

**for** episode = 1, M **do**

    Initialize a random process  $\mathcal{N}$  for action exploration

    Receive initial observation state  $s_1$

**for** t = 1, T **do**

        Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise

        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$

        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$

        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$

        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

**end for**

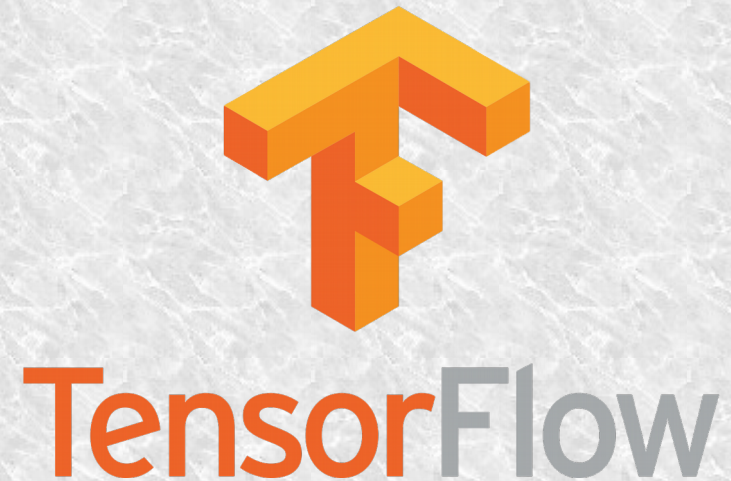
**end for**

---



# Algorithm implementation

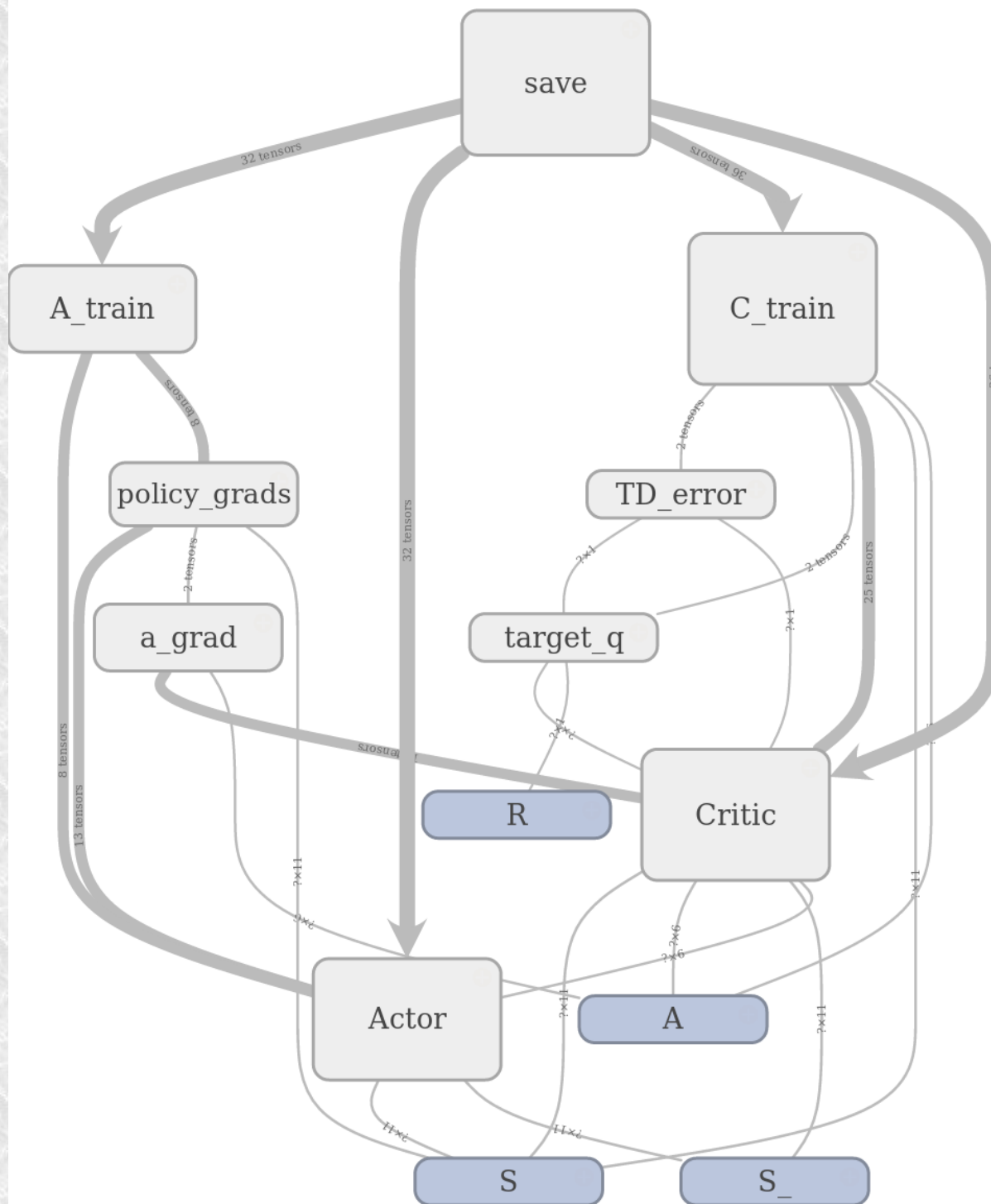
## Python



The code contains 2 parts:

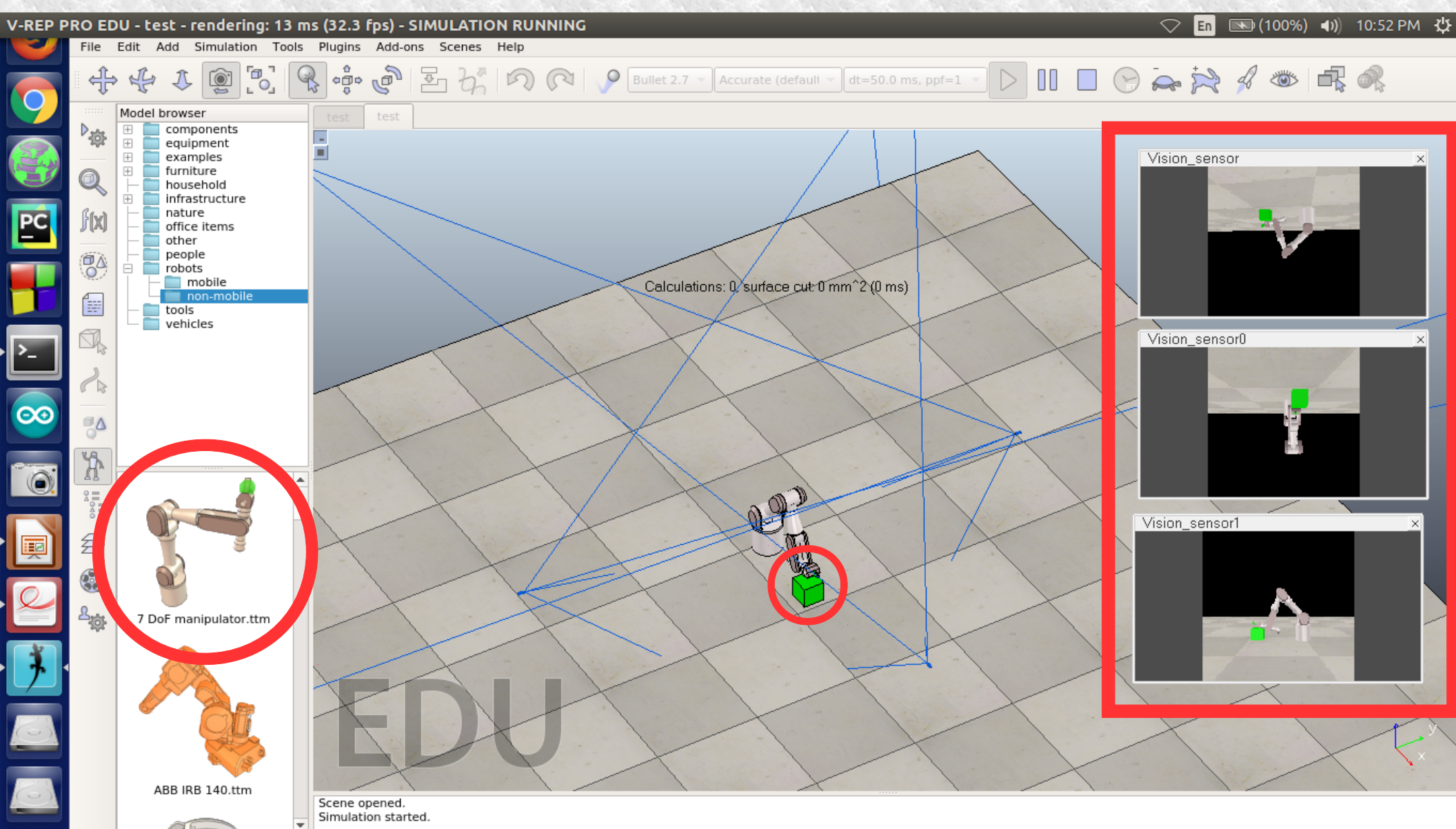
- Algorithm structure (shared for all missions).
- Environment (depends on mission).

### Model-free algorithm





# Simulation Environment



# Simulation Environment

- **V-REP + Python remoteAPI + OpenCV.**
- **State:**  
Joint angles + coordinates
- **Reward:**  
-1\* $\text{square of distances}$   
+ when **done**



# Real world experiment

- 2 Cameras + 3 DOF robot arm only
- Because of USB bandwidth limitation



# Real world experiment

At the beginning of training:





# Real world experiment

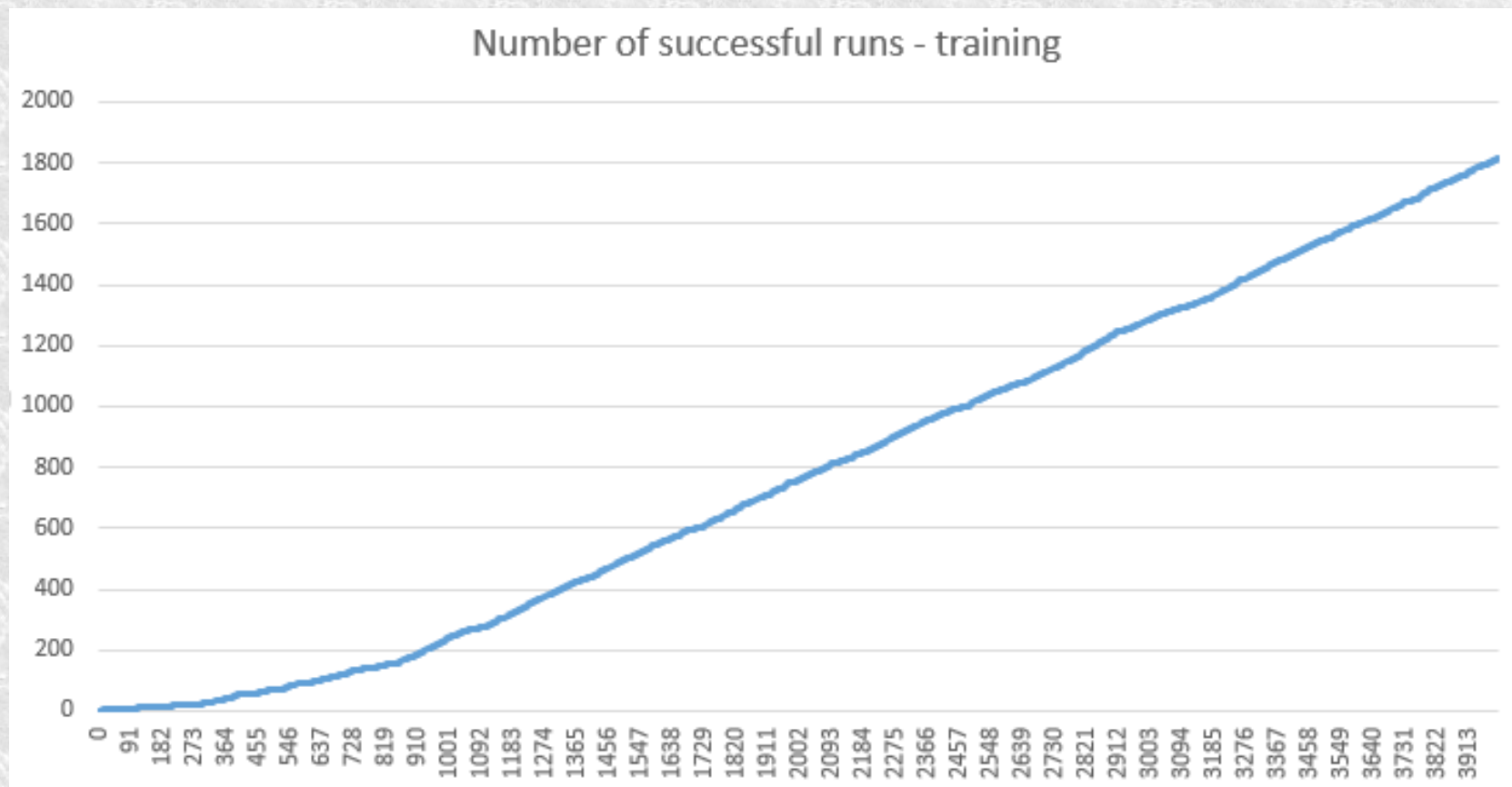
**After 1 hour:**



# Results

## Simulation environment:

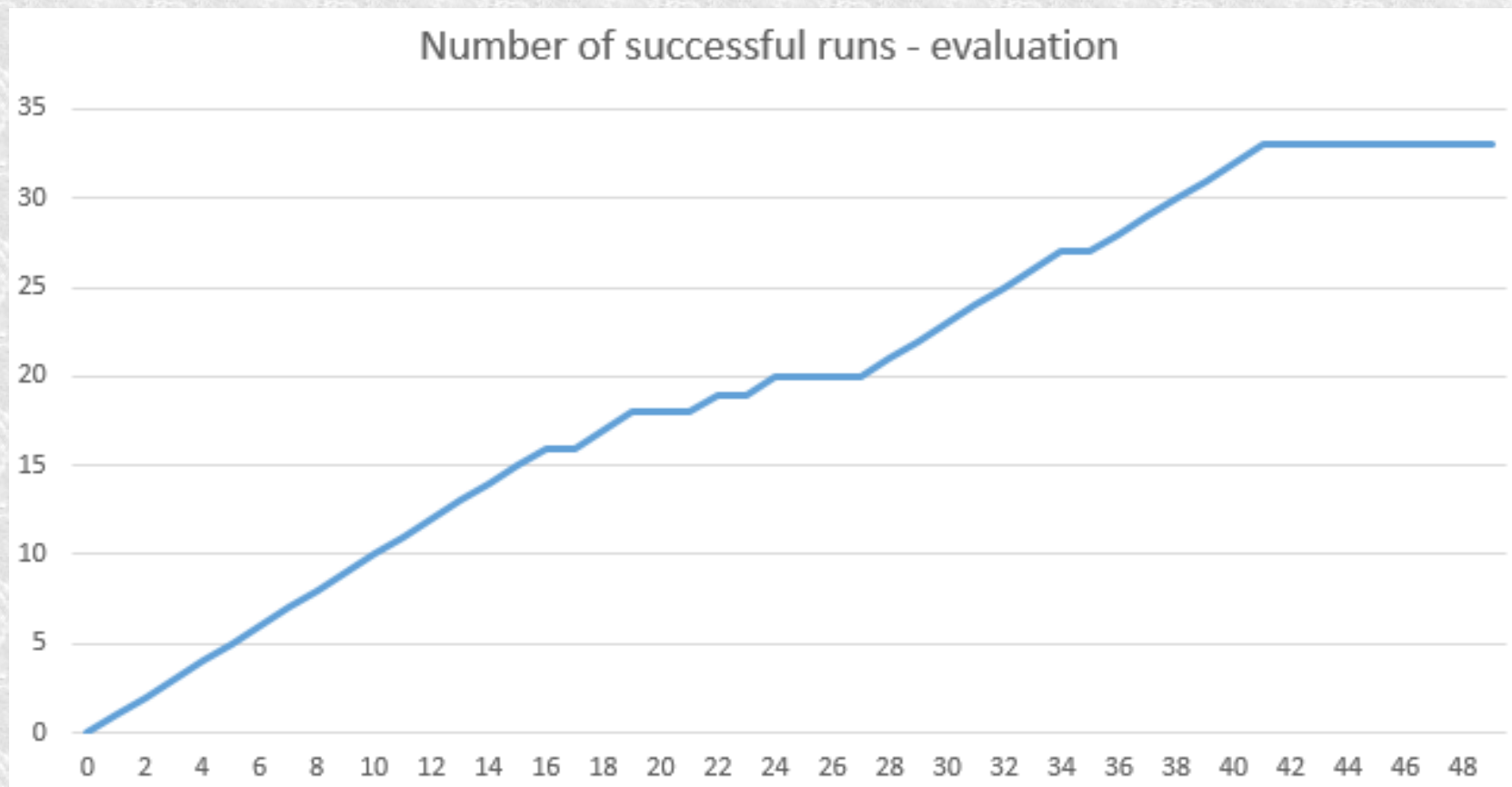
- 5 hours of training -4000 episode
- Success rate  $1800/4000 \approx 45 \%$



# Results

## Simulation environment:

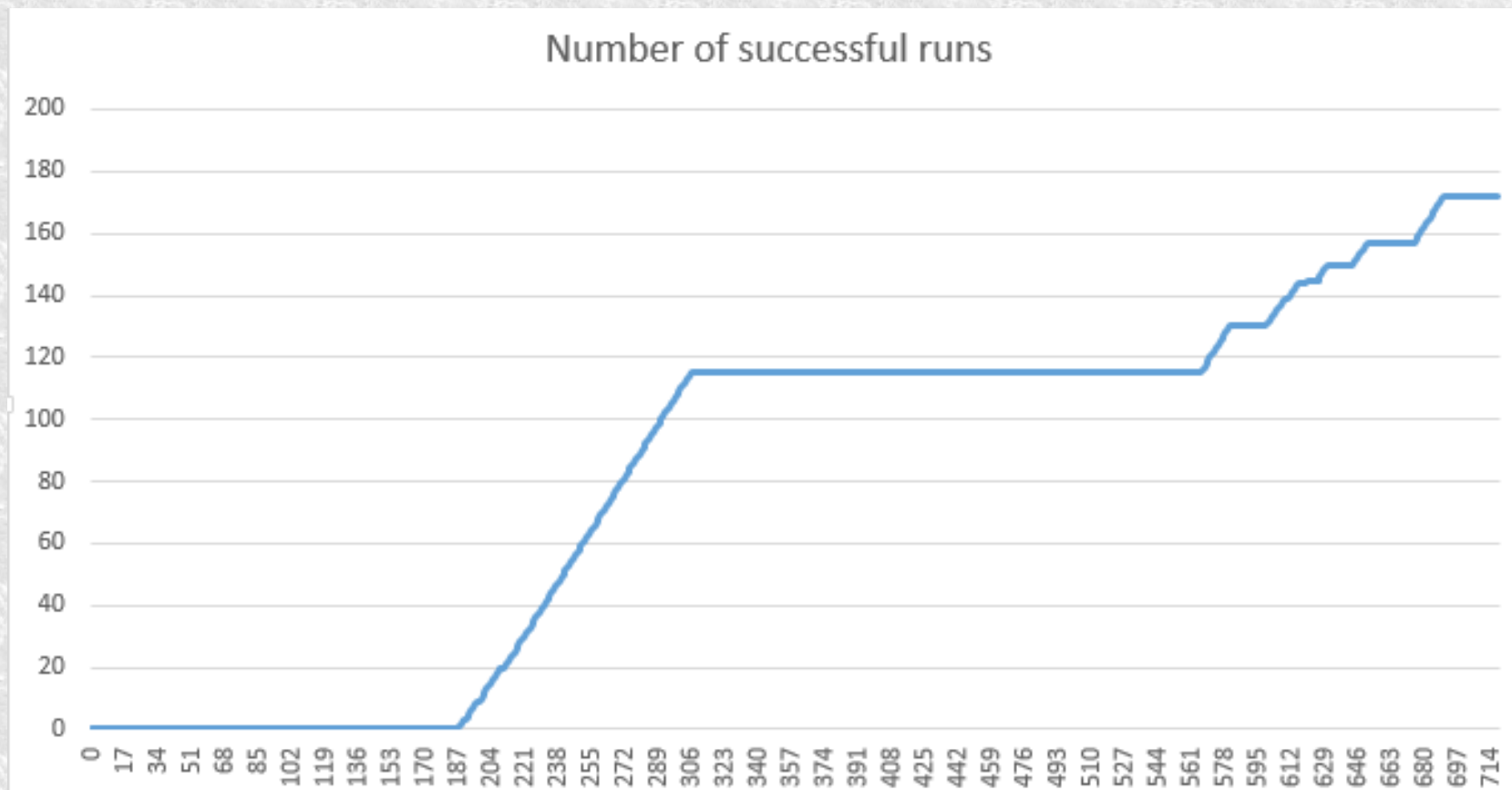
- Evaluation for 50 episodes Max steps 50.
- Success rate  $33/50 \approx 66\%$



# Results

## Real world environment:

- 1 hour in simulation  $\approx$  2.5 in real world
- Success rate  $175/715 \approx 25 \%$



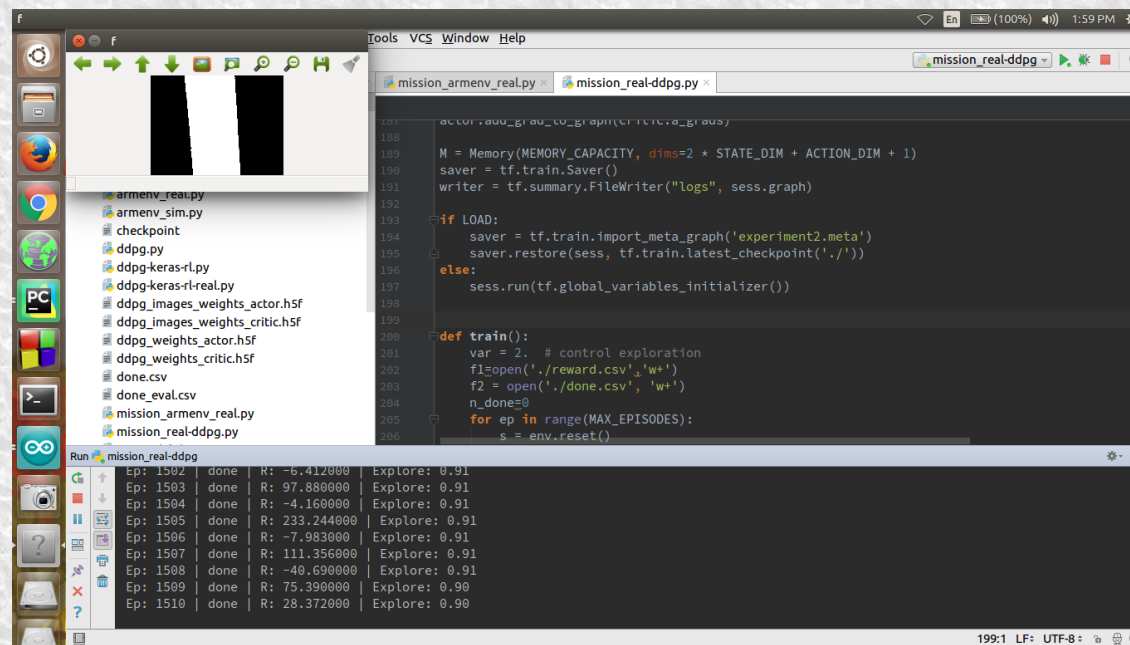


# Development

New mission

(Flexibility of DDPG algorithm):

- Follow a black line with zoom constraints
- Simulate welding process



The screenshot shows a Jupyter Notebook environment. The file explorer on the left lists files such as `armenv_real.py`, `armenv_sim.py`, `checkpoint`, `ddpg.py`, `ddpg-keras-rl.py`, `ddpg-keras-rl-real.py`, `ddpg_images_weights_actor.h5f`, `ddpg_images_weights_critic.h5f`, `ddpg_weights_actor.h5f`, `ddpg_weights_critic.h5f`, `done.csv`, `done_eval.csv`, `mission_armenv_real.py`, and `mission_real-ddpg.py`. The code editor displays the following Python code:

```
287 actor.add_graph(tf.nn.ctc_loss)
288
289 M = Memory(MEMORY_CAPACITY, dims=2 * STATE_DIM + ACTION_DIM + 1)
290 saver = tf.train.Saver()
291 writer = tf.summary.FileWriter("logs", sess.graph)
292
293 if LOAD:
294     saver = tf.train.import_meta_graph('experiment2.meta')
295     saver.restore(sess, tf.train.latest_checkpoint('.'))
296 else:
297     sess.run(tf.global_variables_initializer())
298
299
300 def train():
301     var = 2. # control exploration
302     f1=open('./reward.csv','w+')
303     f2 = open('./done.csv', 'w+')
304     n_done=0
305     for ep in range(MAX_EPISODES):
306         s = env.reset()
```

The terminal window at the bottom shows the following training logs:

Ep	done	R	Explore
1502	done	-6.412000	0.91
1503	done	97.880000	0.91
1504	done	-4.160000	0.91
1505	done	233.244000	0.91
1506	done	-7.983000	0.91
1507	done	111.356000	0.91
1508	done	-40.690000	0.91
1509	done	75.390000	0.90
1510	done	28.372000	0.90