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...

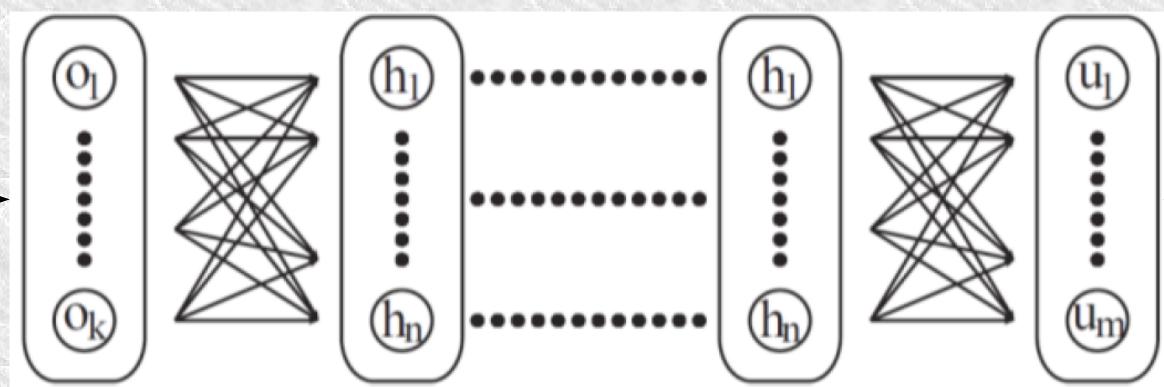
Why deep learning?

Standard Robotic Control

Observations → **State Estimation** → **Motion Planning** → **Low-level Controller** → **Motor torques**

Deep Robotic Control

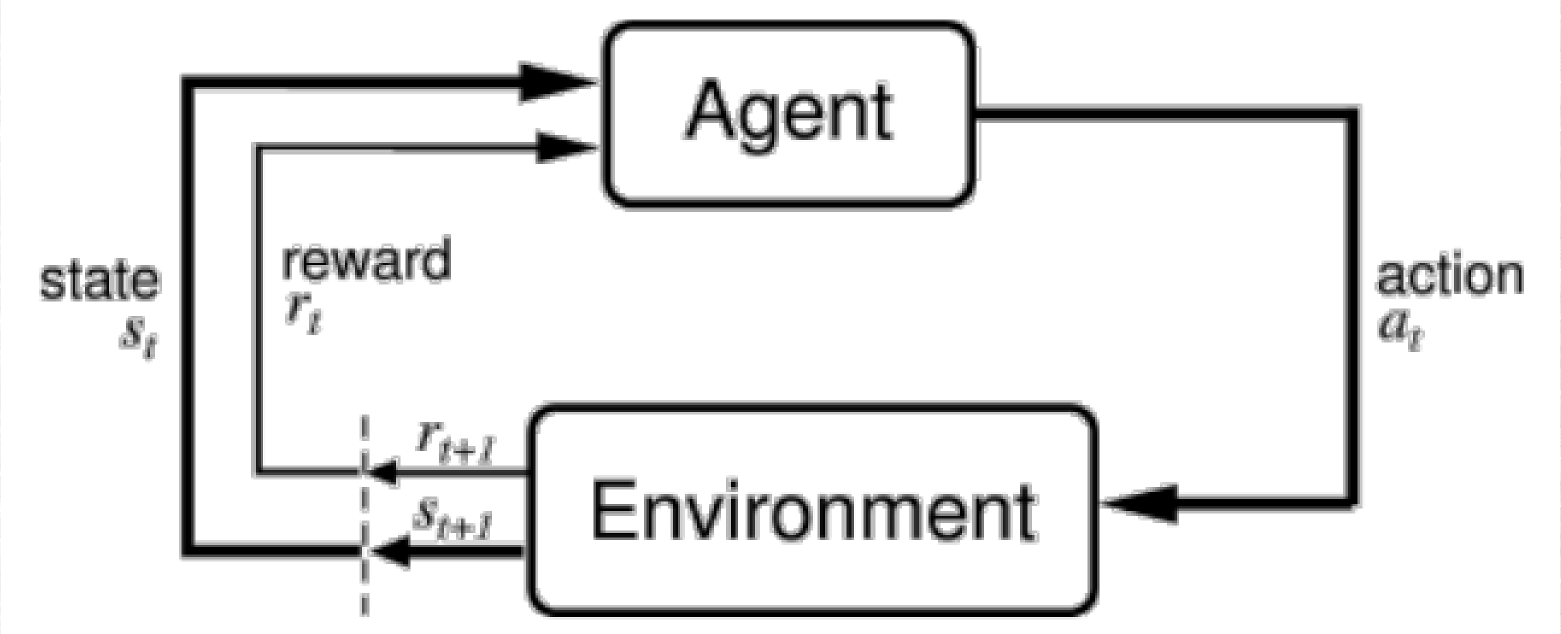
Observations →



→ **Motor torques**

Reinforcement learning

A branch of machine learning



The objective is to maximize the expected reward

Reinforcement learning

Markov Decision Process

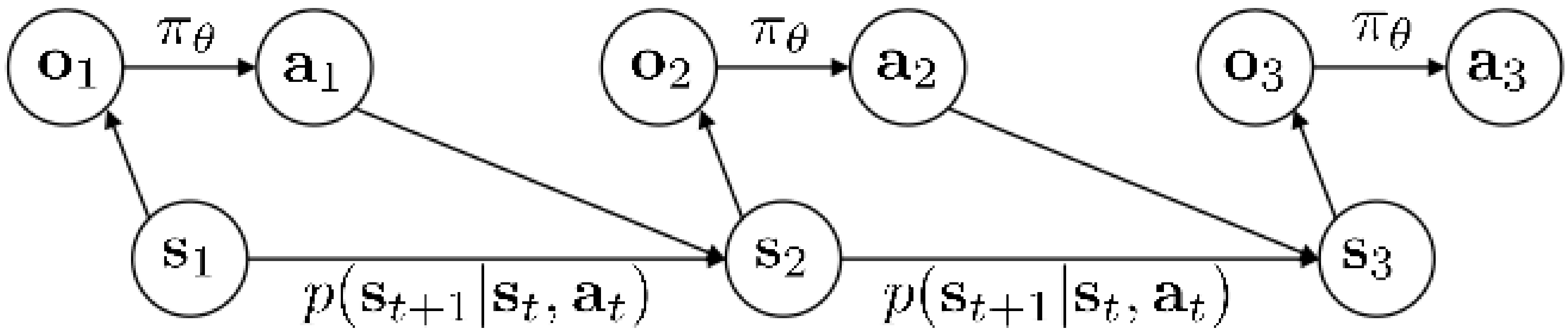
$\mathbf{s}_t$ – state

$\mathbf{o}_t$ – 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

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

Supervised learning

x – images
y – labels

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

Reinforcement learning

s – state
a – actions
(from actor network)

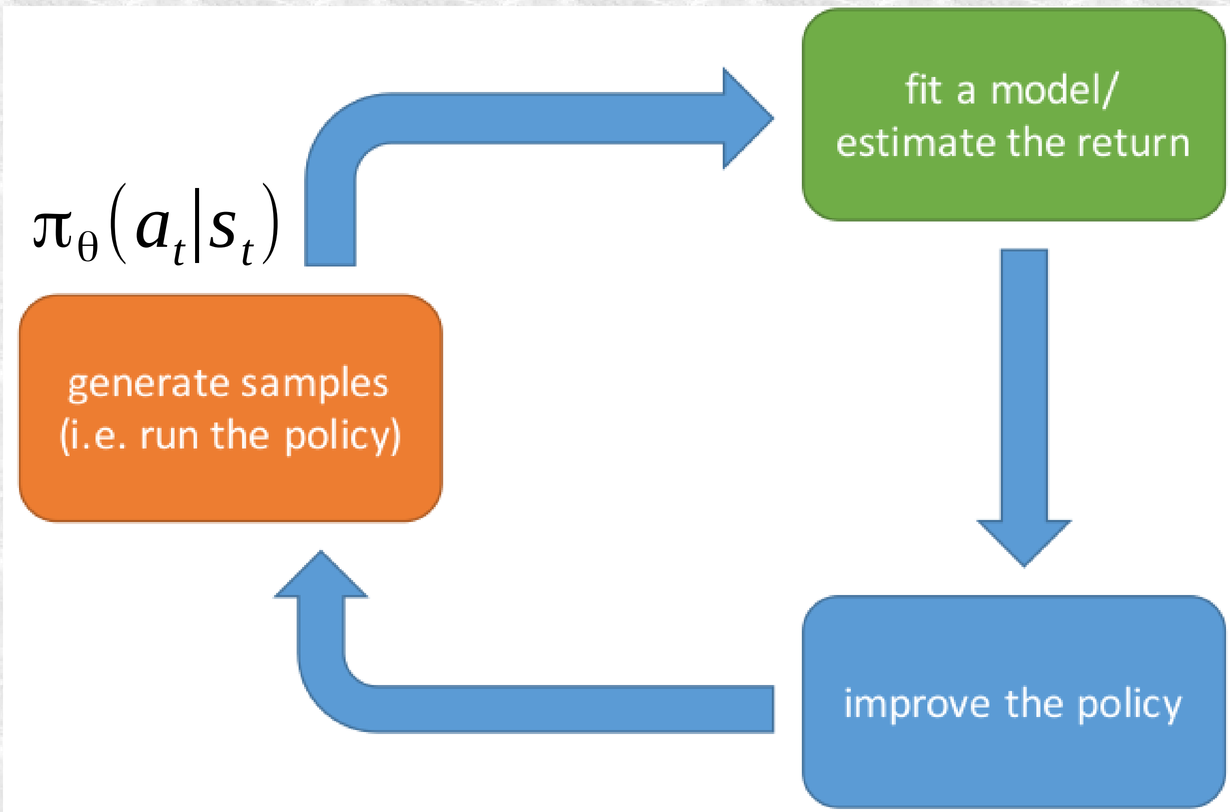
Algorithm discussion

DDPG
Deep Deterministic Policy Gradients

Lillicrap’16

Deep Q learning + Actor-critic = DDPG

Policy gradients



$\pi_\theta(a_t|s_t)$

$r(s_t, a_t)$

$\theta \leftarrow \theta + \nabla_\theta J$

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

Policy gradients

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

Gradient vector of policy:

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

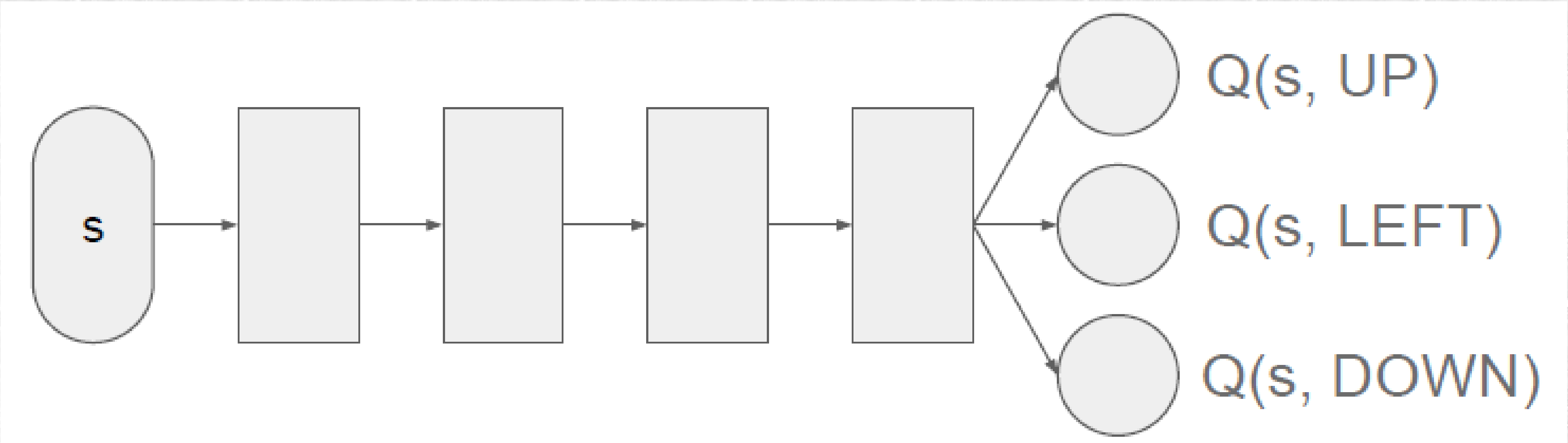
Score (reward) Function:

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

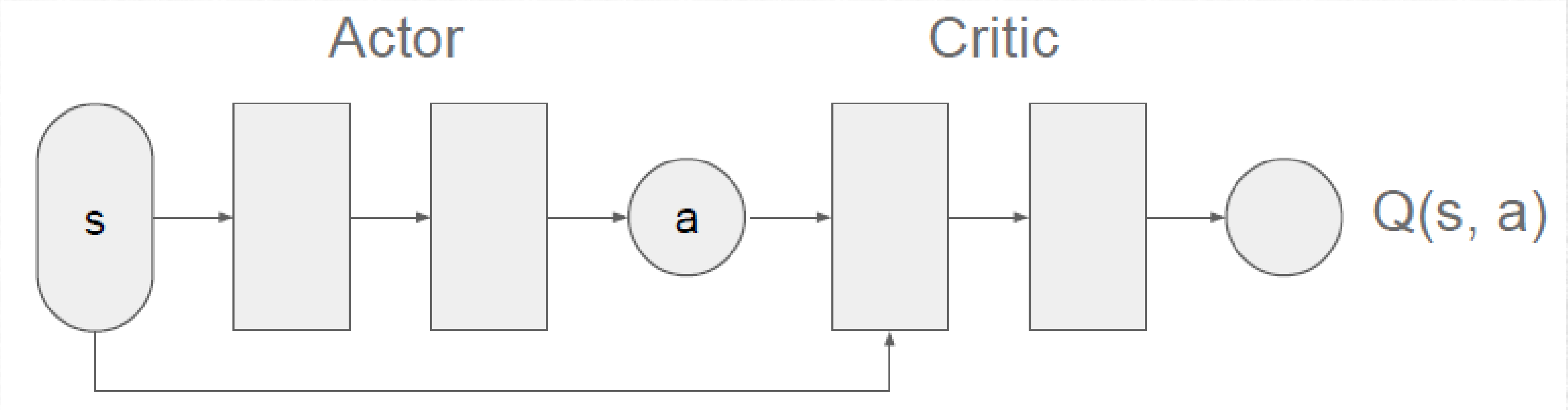


Actor Critic

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 θ^Q and θ^μ.
Initialize target network Q' and μ' 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

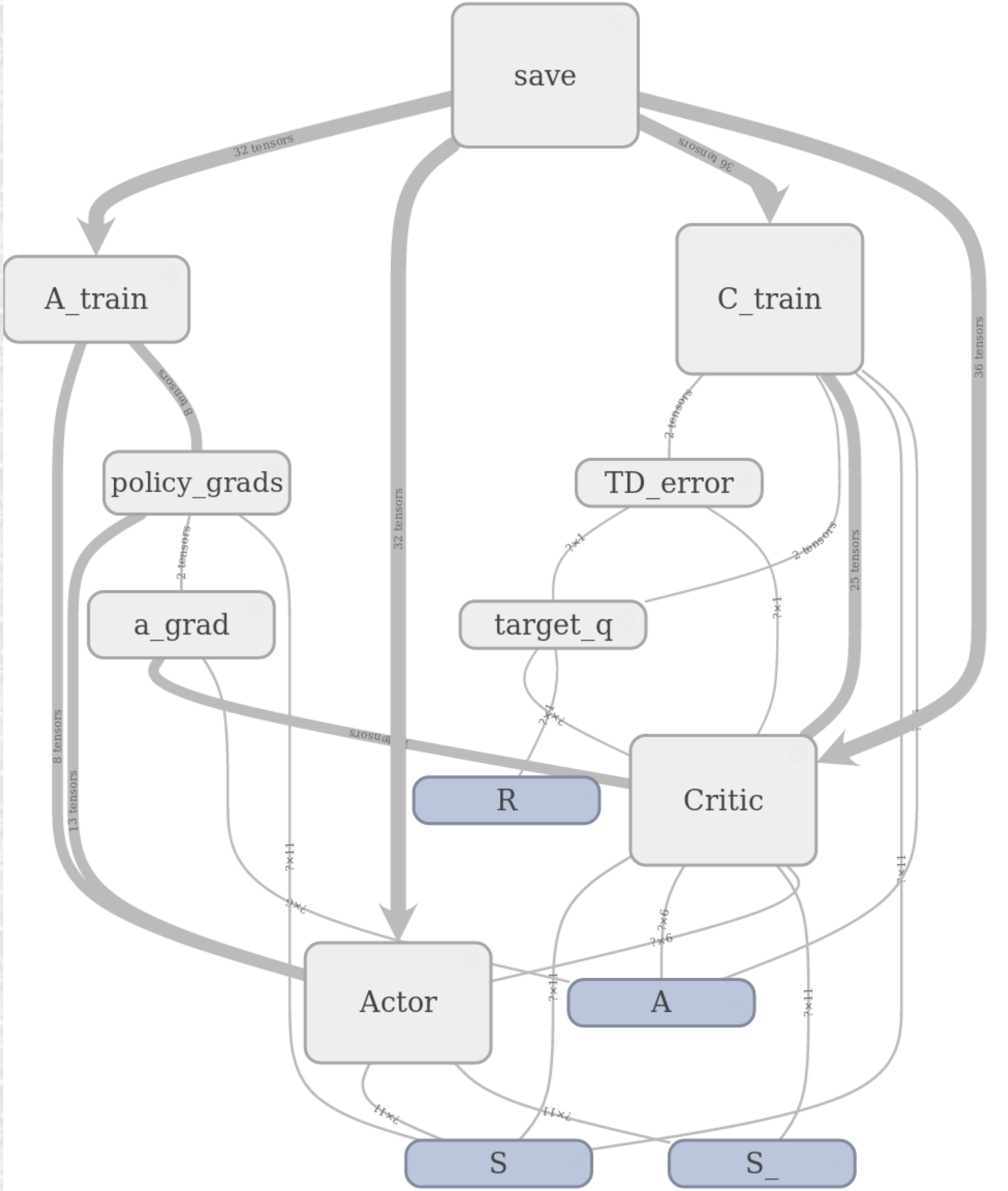
Python

TensorFlow

The code contains 2 parts:

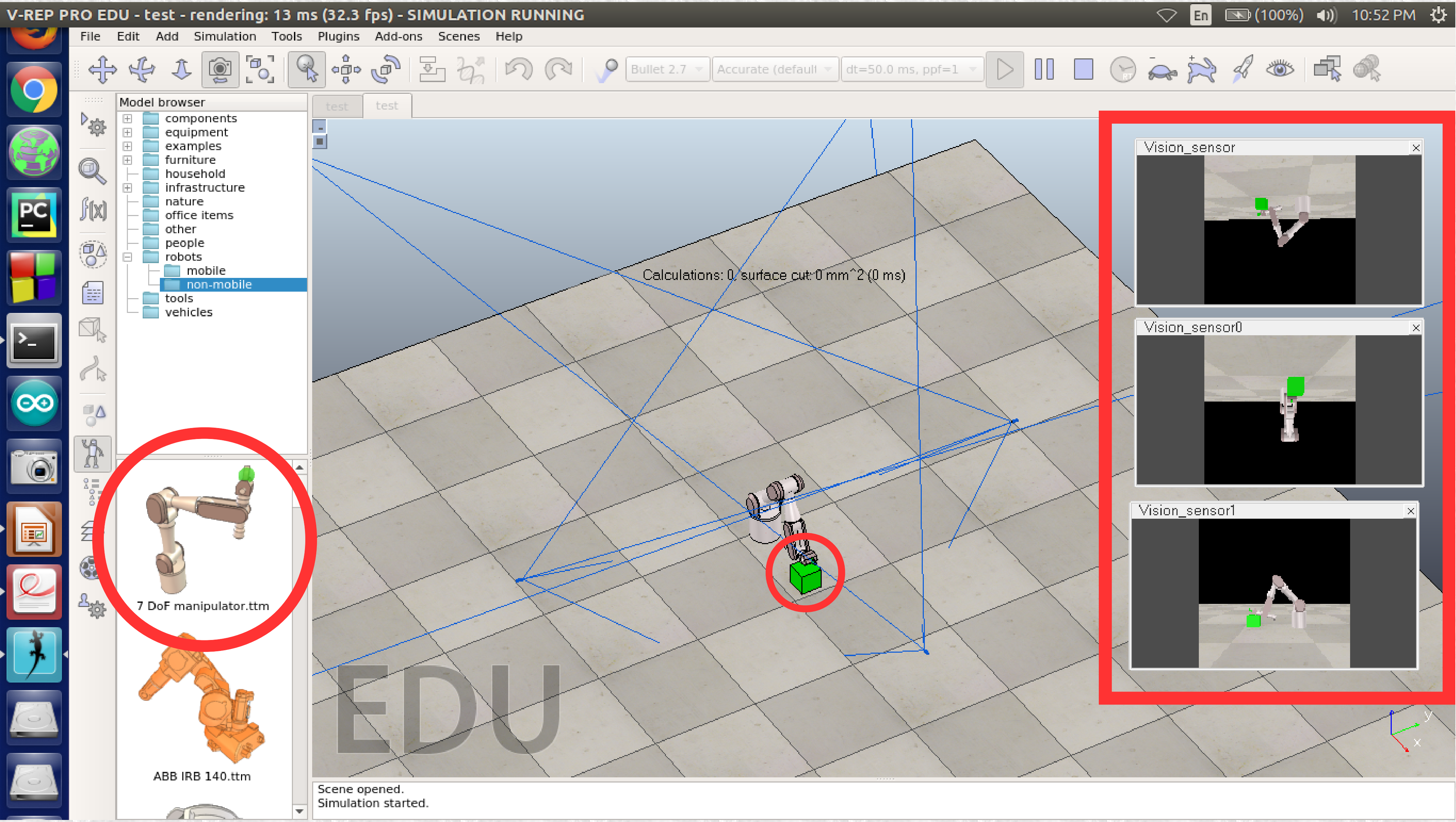
- **Algorithm structure (shared for all missions).**
- **Environment (depends on mission).**

Model-free algorithm



save
A_train
C_train
policy_grads
TD_error
a_grad
target_q
R
Critic
Actor
A
S
S_
32 tensors
36 tensors
32 tensors
36 tensors
8 tensors
2 tensors
2 tensors
2 tensors
25 tensors
8 tensors
13 tensors

Simulation Environment



Simulation Environment

- **V-REP + Python remoteAPI + OpenCV.**
- **State:**

  Joint angles + coordinates
- **Reward:**

  -1*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:

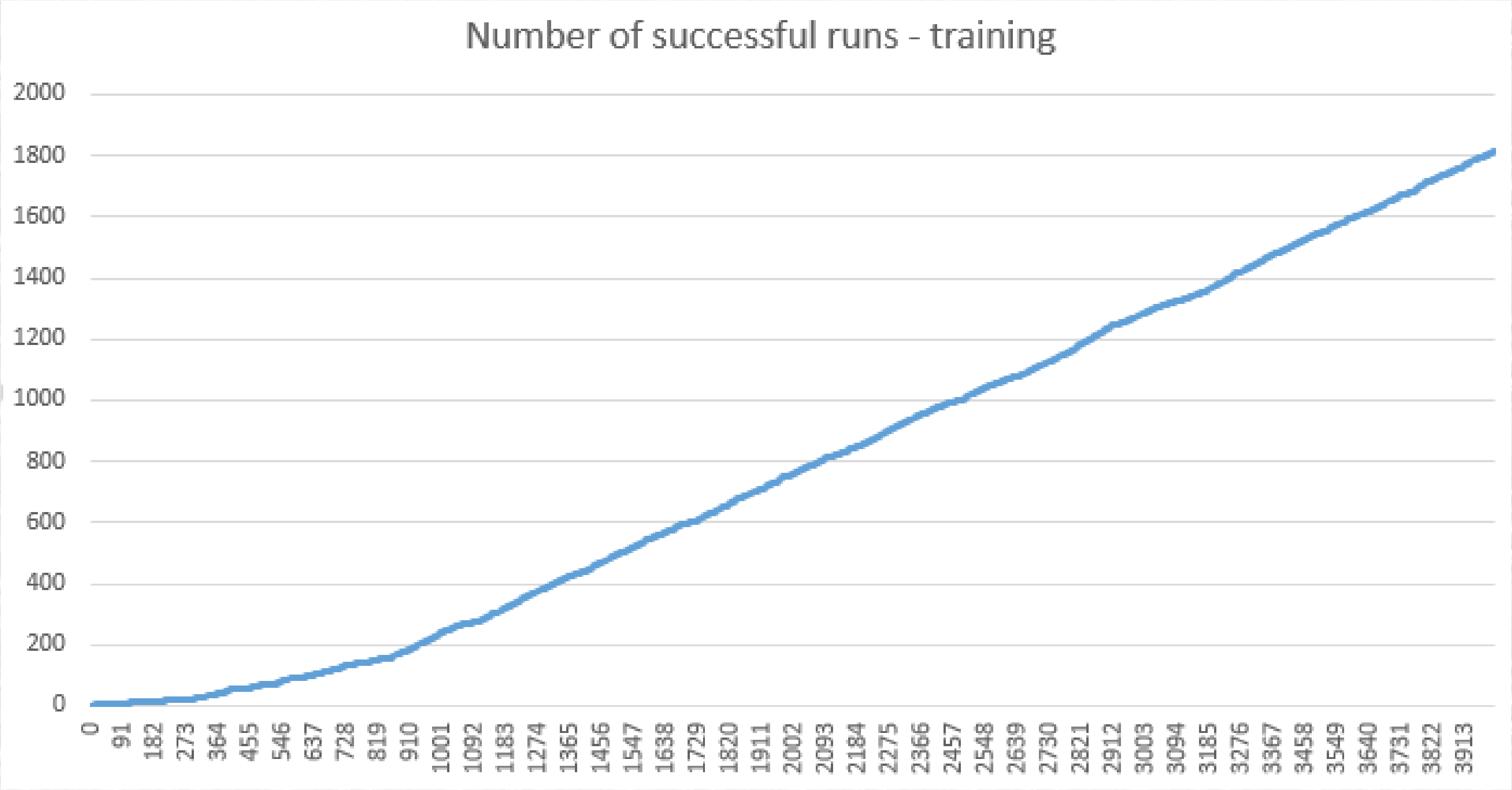
Real world experiment

After 1 hour:

Results

Simulation environment:

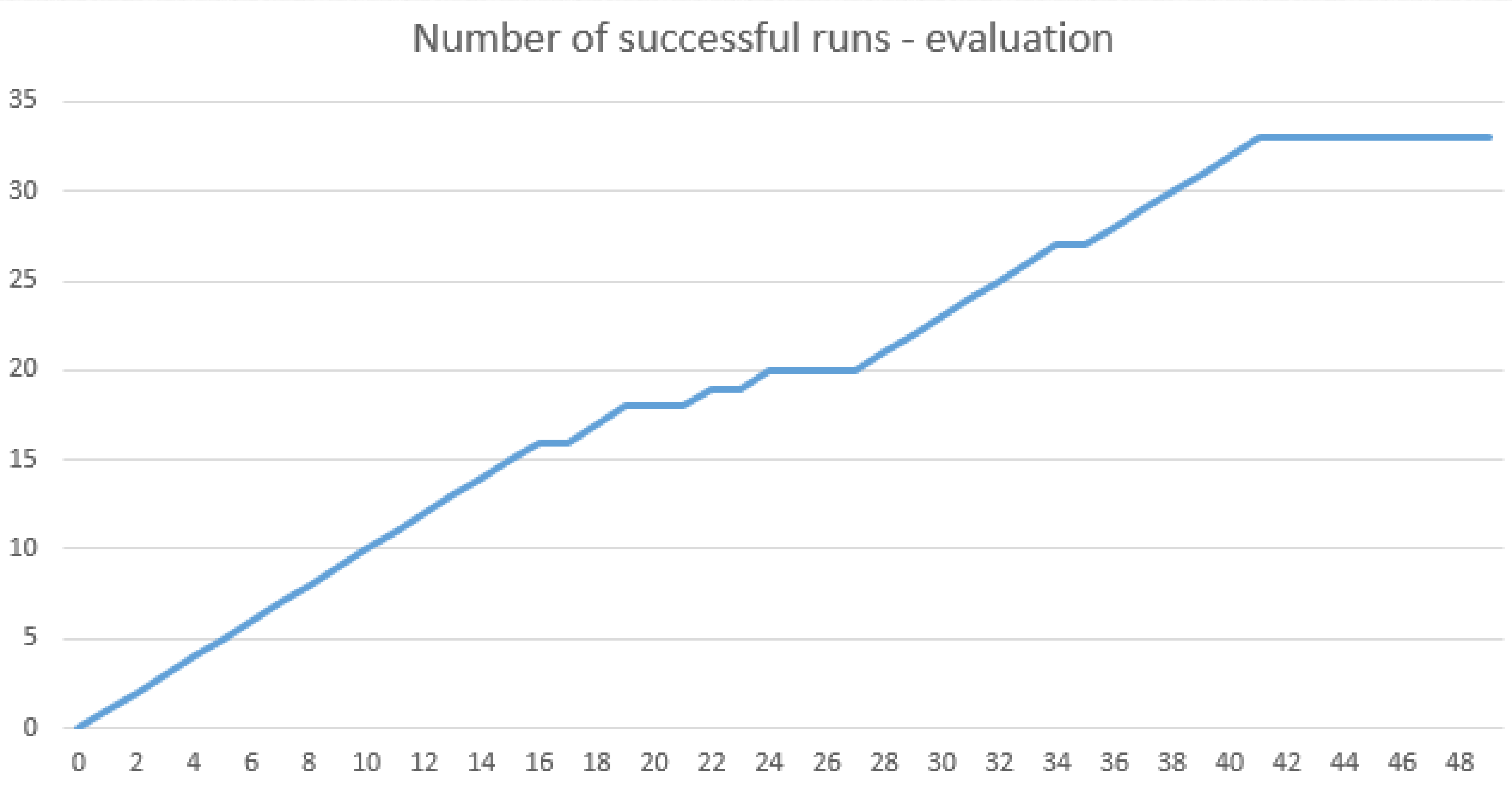
- 5 hours of training -4000 episode
- Success rate 1800/4000 ≈ 45 %



Results

Simulation environment:

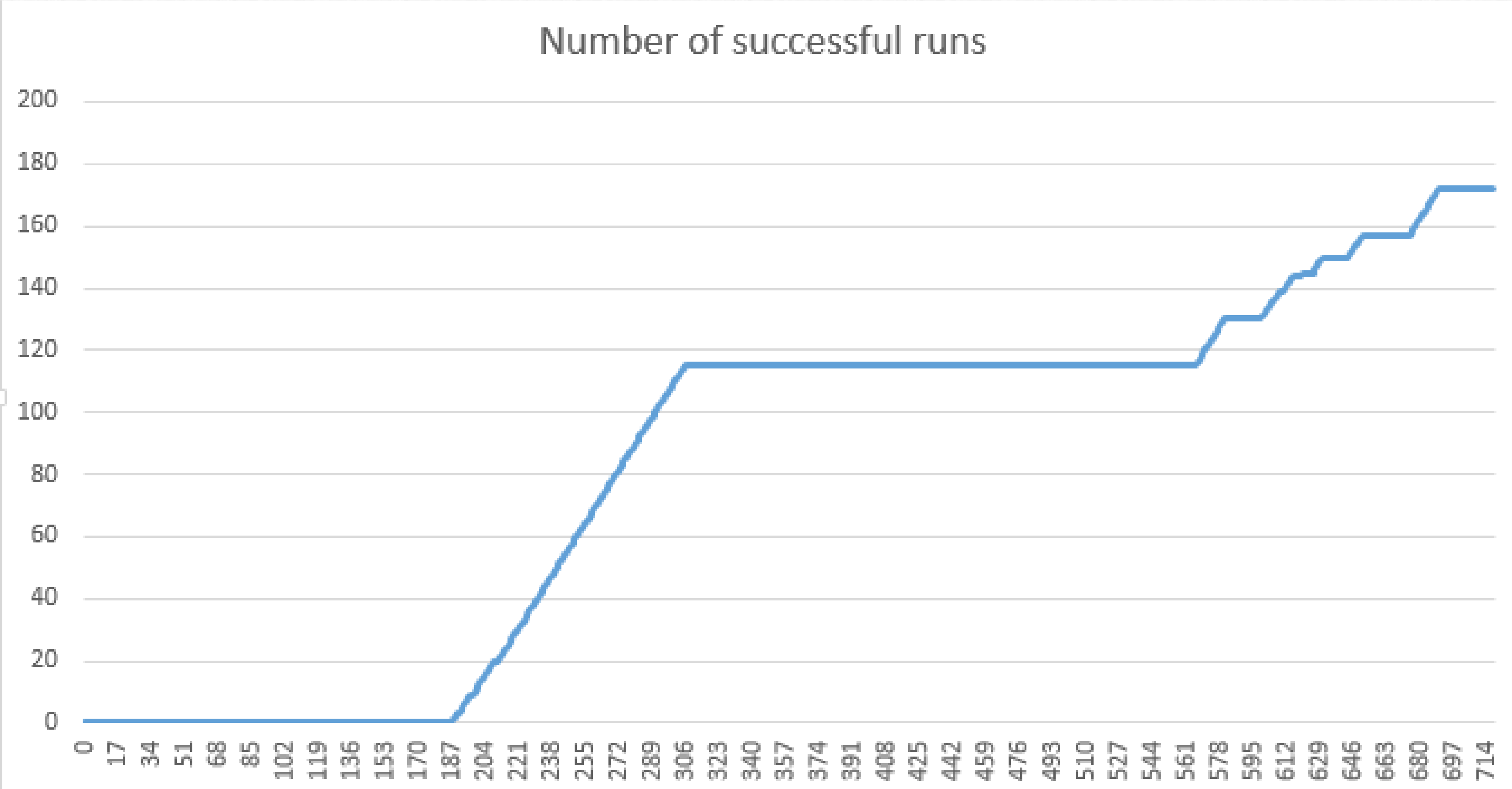
- Evaluation for 50 episodes Max steps 50.
- Success rate 33/50 ≈ 66 %



Results

Real world environment:

- 1 hour in simulation ≈ 2.5 in real world
- Success rate 175/715 ≈ 25 %



Development

New mission

(Flexibility of DDPG algorithm):

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