Adversary Robust A3C

Zhaoyuan Gu

Sept, 2017

Contents

Buzz Words

- Maxon - an actuator manufacturer
- ROS - Robot Operating System, a mechanism for process communication and a set of robotics software tools

Buzz Words

- RL - Reinforcement Learning
- Policy - the strategy a RL algorithm adopt to generate an action given the environment observation
- A3C - a RL algorithm for robot control
- RARL - a method to enhance robustness for RL
- TensorFlow - a python framework widely used in learning area

The Story

- RL is not robust, it fails with very small disturbance
- we try to increase the robustness of RL using a method called RARL
- we use A3C as the baseline, A3C works perfectly on robot control tasks, it is easy to implement and extend, we try strengthening the algorithm by applying RARL
- RARL + A3C = AR-A3C (Adversary Robust A3C)

The Story

- A3C = Asynchronous Advantage Actor-Critic**[2]**
  - Asynchronous, multi-thread
  - Actor-Critic (AC), a popular RL algorithm, a combination of policy iteration and value iteration
- RARL = Robust Adversary Reinforcement Learning**[3]**
  - dual agent, Protagonist and Adversary
  - enhance robustness by adding adversarial force to the environment during training

[2] Asynchronous methods for deep reinforcement learning
[3] Robust Adversarial Reinforcement Learning

How does A3C work

1. Controller calculate Action using last State
2. System gives new Reward and State
3. Controller update using Reward

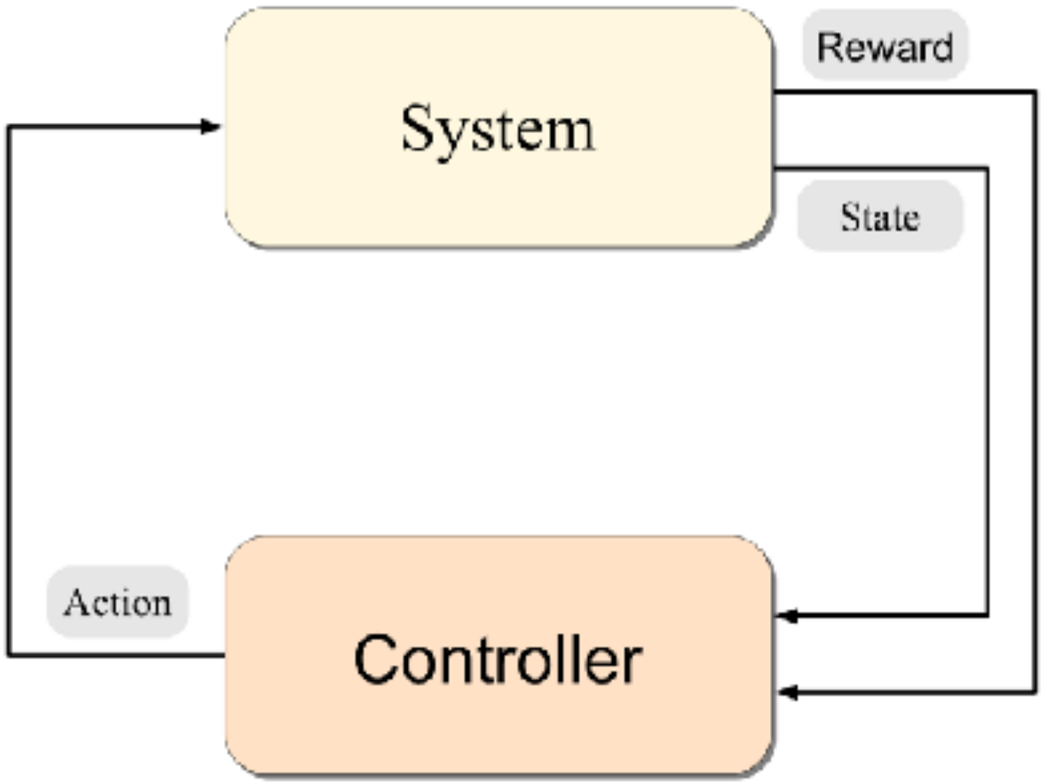


Fig. RL Structure[1]

[1] Reinforcement Learning Algorithms for MDPs

How does A3C work

1. Actor-Critic Pair works as a Controller
2. Both Actor and Critic are Neural Network
3. Actor calculate Action using last State
4. System calculate Reward and State given action
5. Critic calculate Evaluation using State
6. Actor update using Evaluation
7. Critic update using Reward

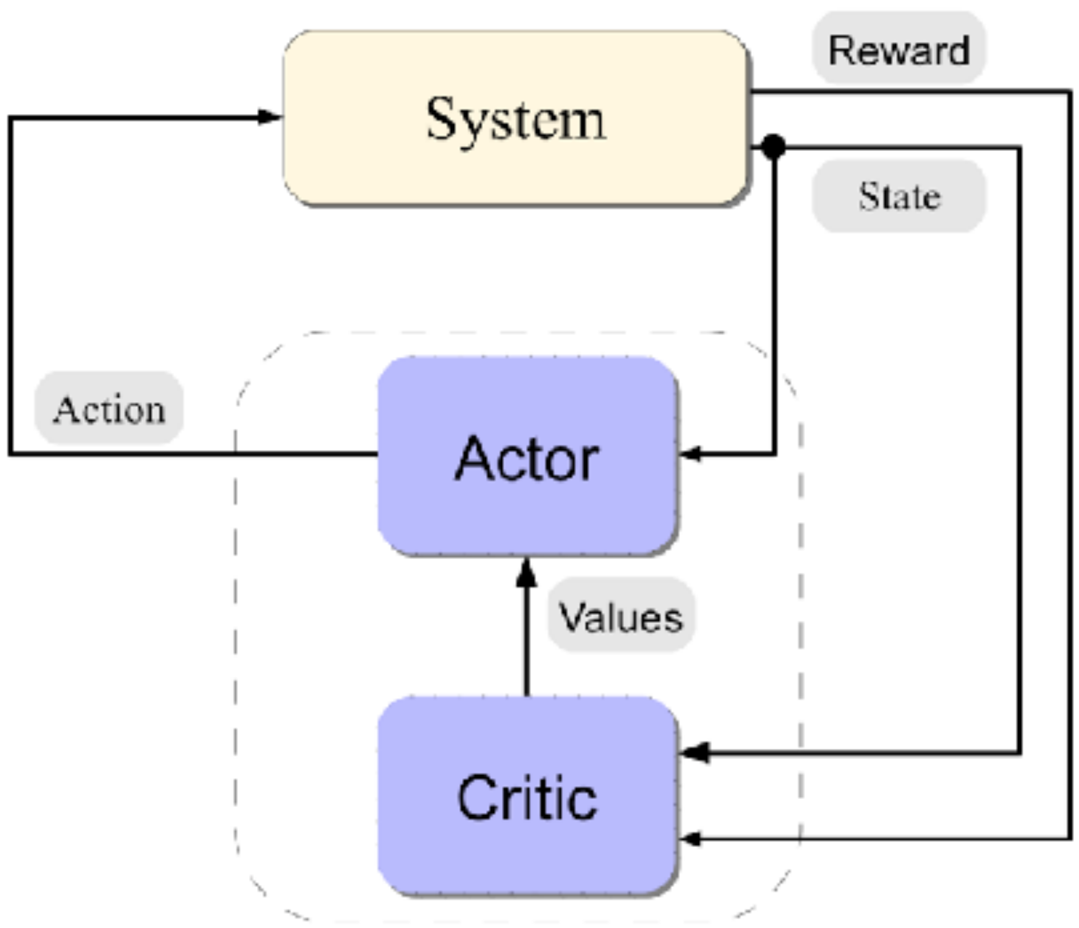


Fig. AC Structure[1]

[1] Reinforcement Learning Algorithms for MDPs

How does A3C work

- The system includes a **global AC** and several **worker ACs**
- **worker AC** works simultaneously, each interacting with different copies of environment
- **worker AC** store their experience and upload to **Global AC** to update the Policy
- **worker AC** pull the updated Policy and go on exploring the environment
- the **Global AC** absorb all the experience and finally converge to optimum

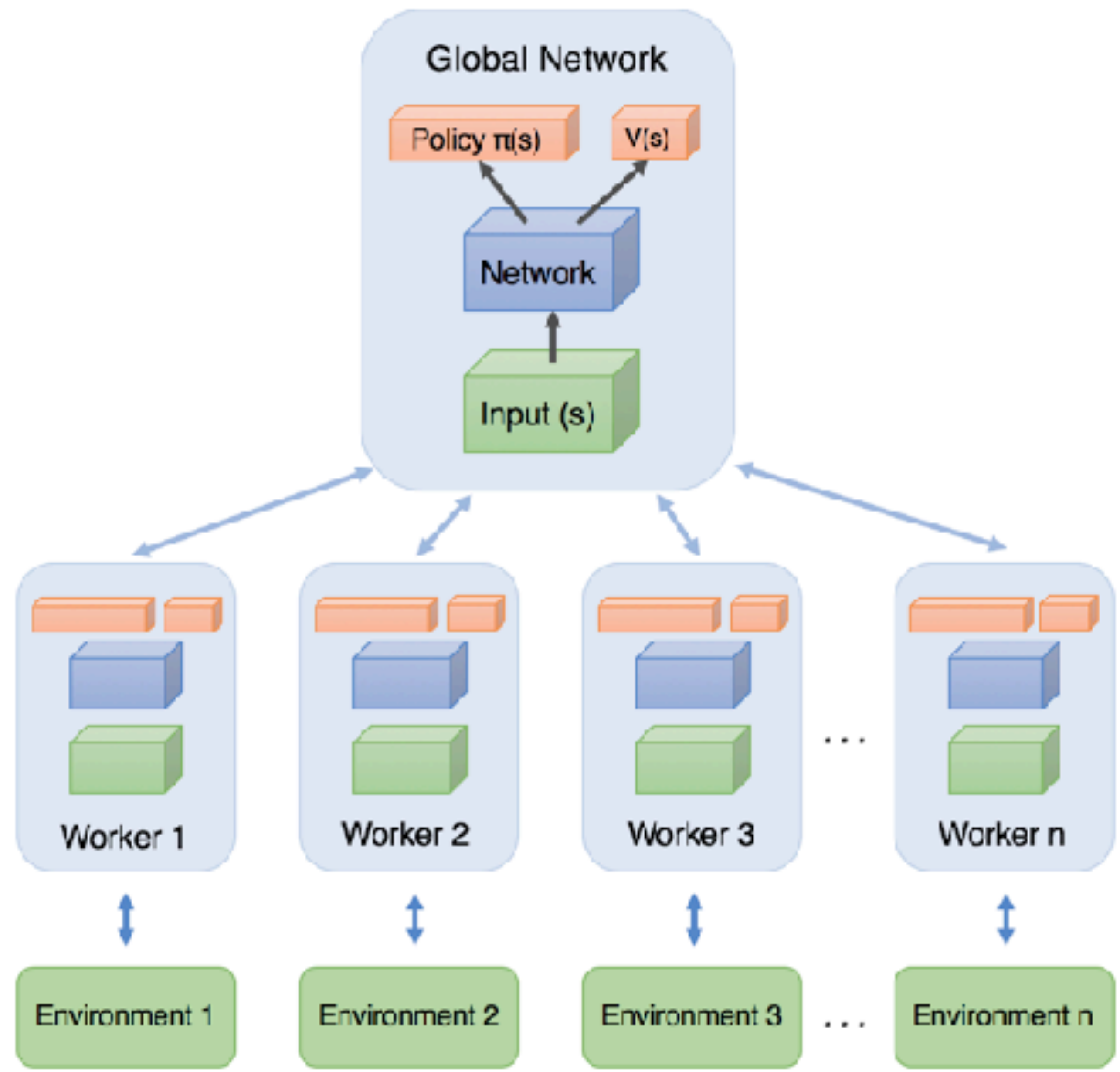


Fig. A3C Structure

How does RARL work?

- RARL is compatible with other RL algorithms
- RARL strengthen the RL by applying adversarial disturbance on robot while training
- the adversary is also an agent with the same AC network, it learns to violate the task
- the agent trained in environment with adversary is resistant to mild disturbance, thus more robust

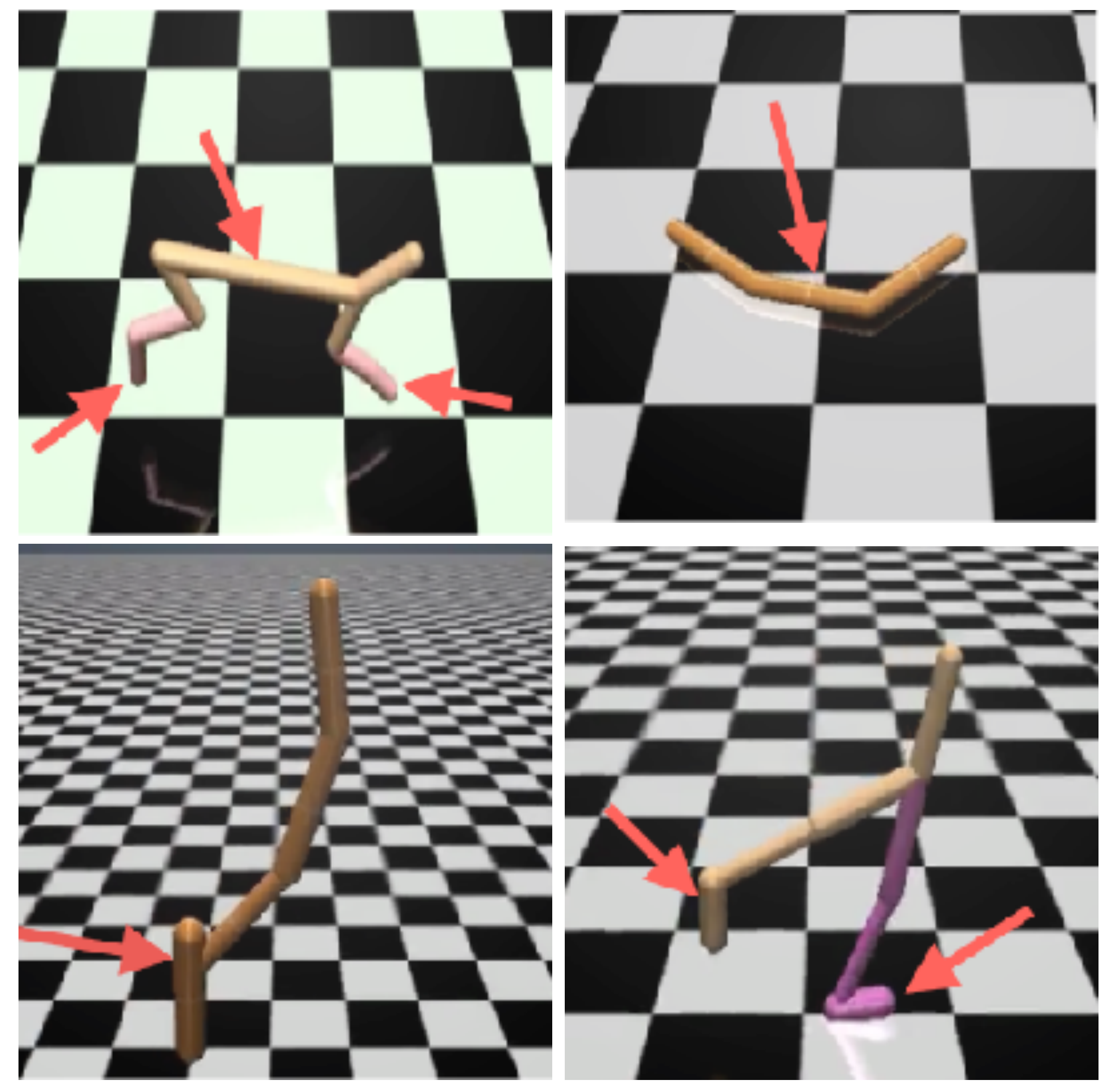


Fig. RARL adversary apply disturbance[3]

AR-A3C

- AR-A3C Combines A3C with RARL method
- for each Protagonist, another same AC pair is added to apply Adversarial Action
- the Adversary AC is updated to minimize the reward given the state and reward

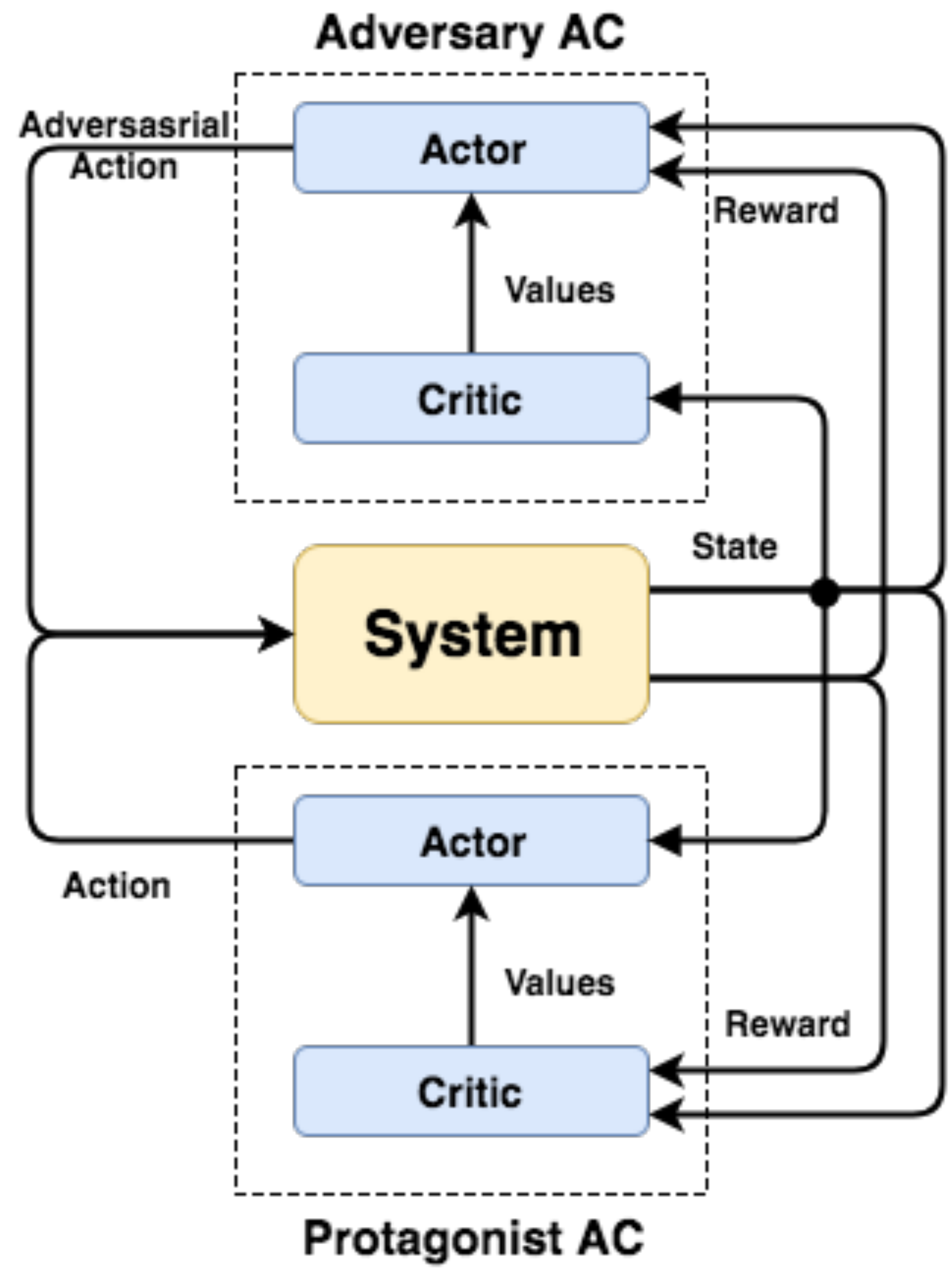


Fig. AR-A3C single thread structure

AR-A3C

- within each thread, a worker contains one Protagonist AC and one Adversary AC
- each worker interact with environment, then upload observed experience to Global

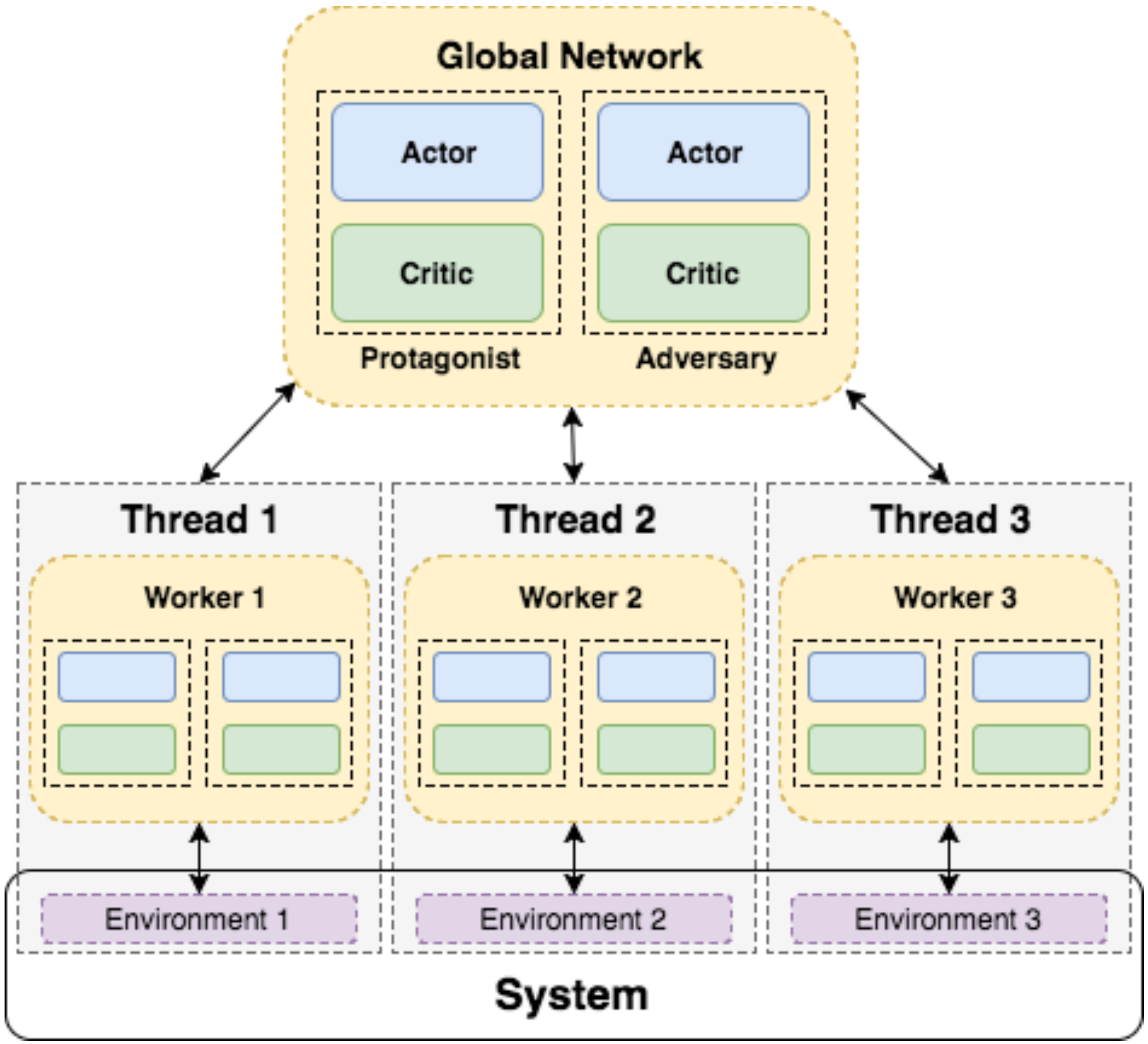
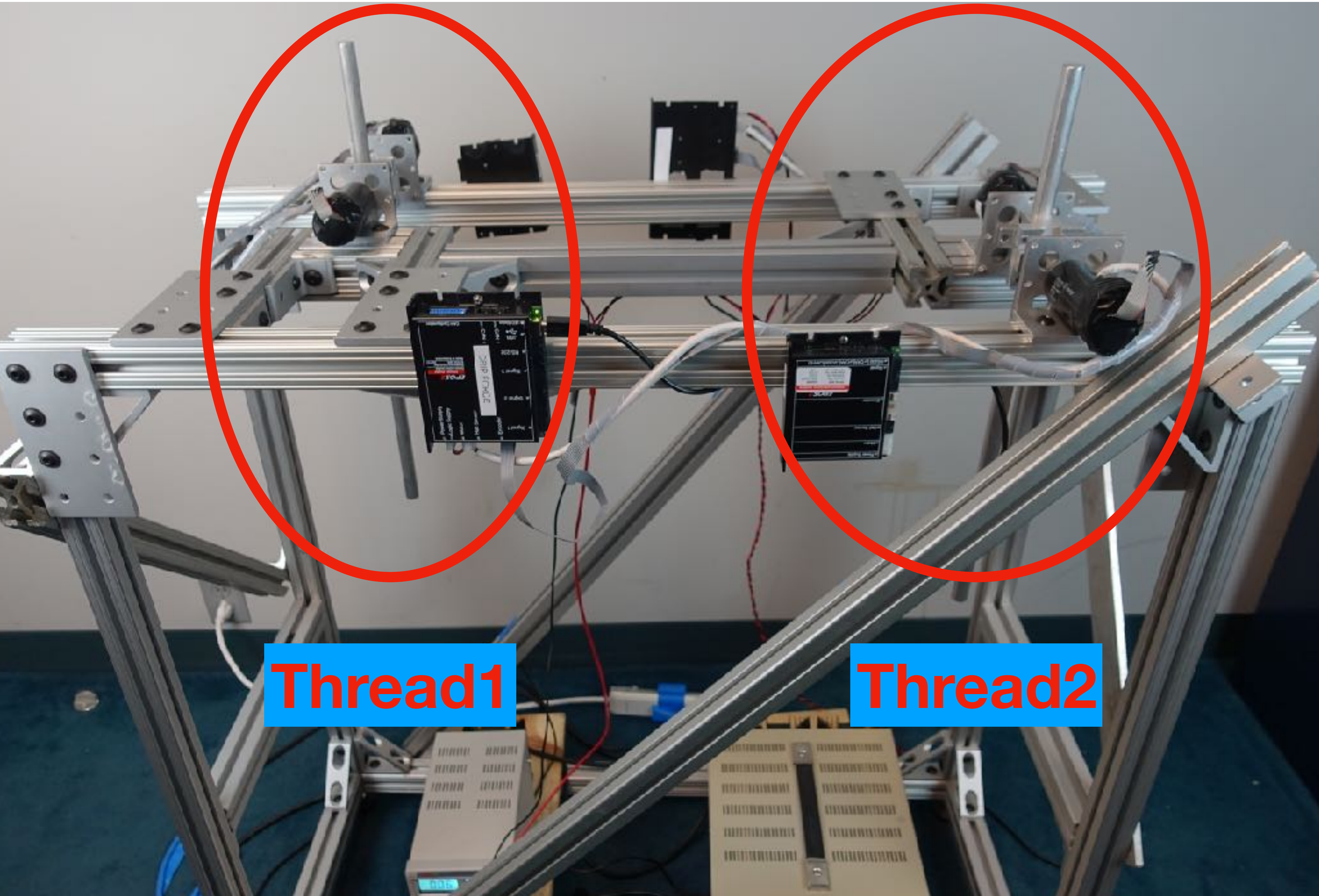


Fig. AR-A3C Structure

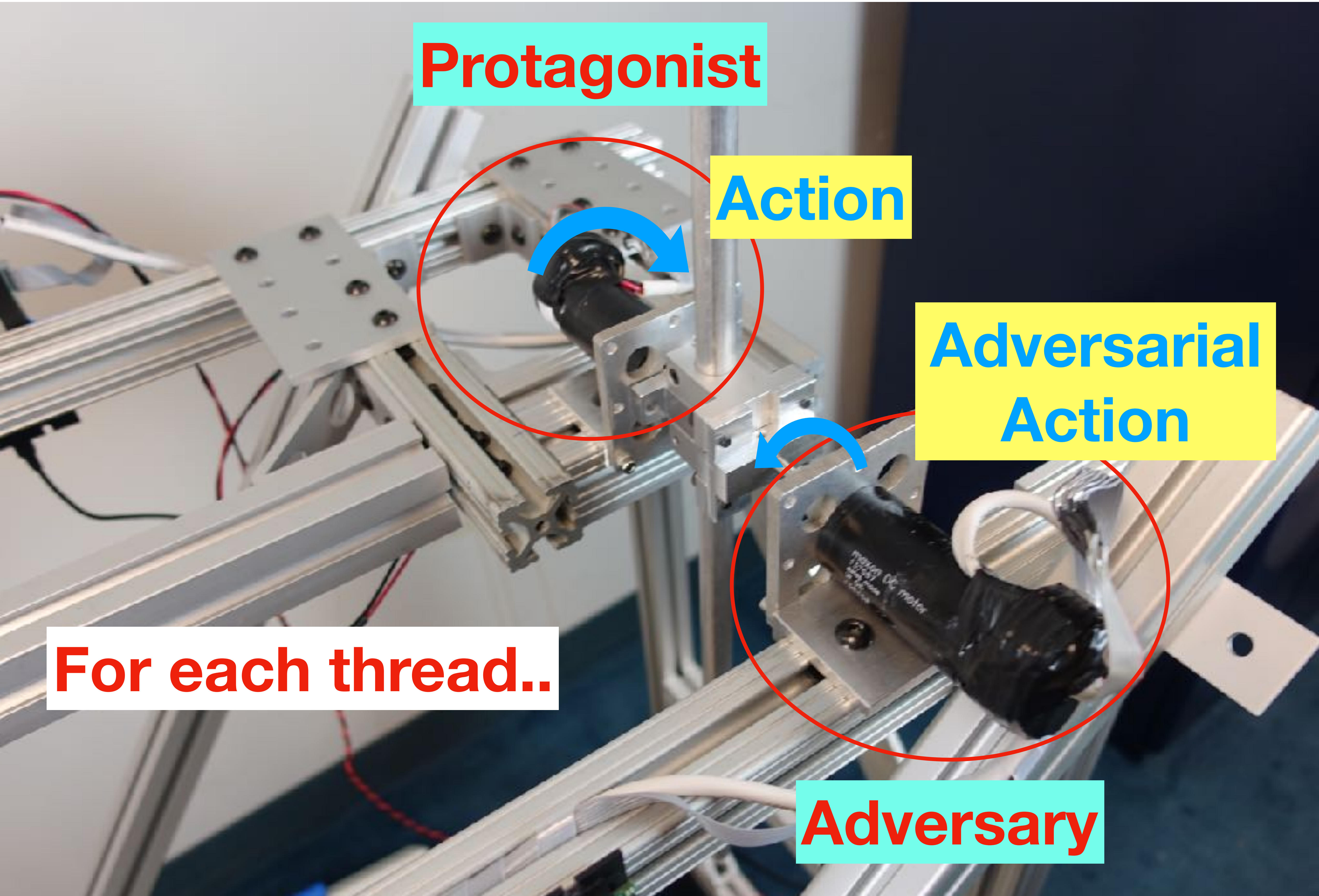
How to implement? Hardware

- what we need:
  - multi thread structure
  - within each thread, two AC pairs applying action to the same environment

Hardware AR-A3C



Hardware AR-A3C



How to implement? Software

- **Controller part**:
    - use cpp library from Maxon official
    - API includes position, velocity and current control
    - use current control mode to apply torque
    - the controller is wrapped into a environment for agent to explore
- **Algorithm part**:
    - implemented in python using Tensorflow
    - request action using ROS service
    - save model at key frame for test and for later keep training

Full code for Control and Test: https://github.com/guzhaoyuan/epos2

How to implement?
Training Process

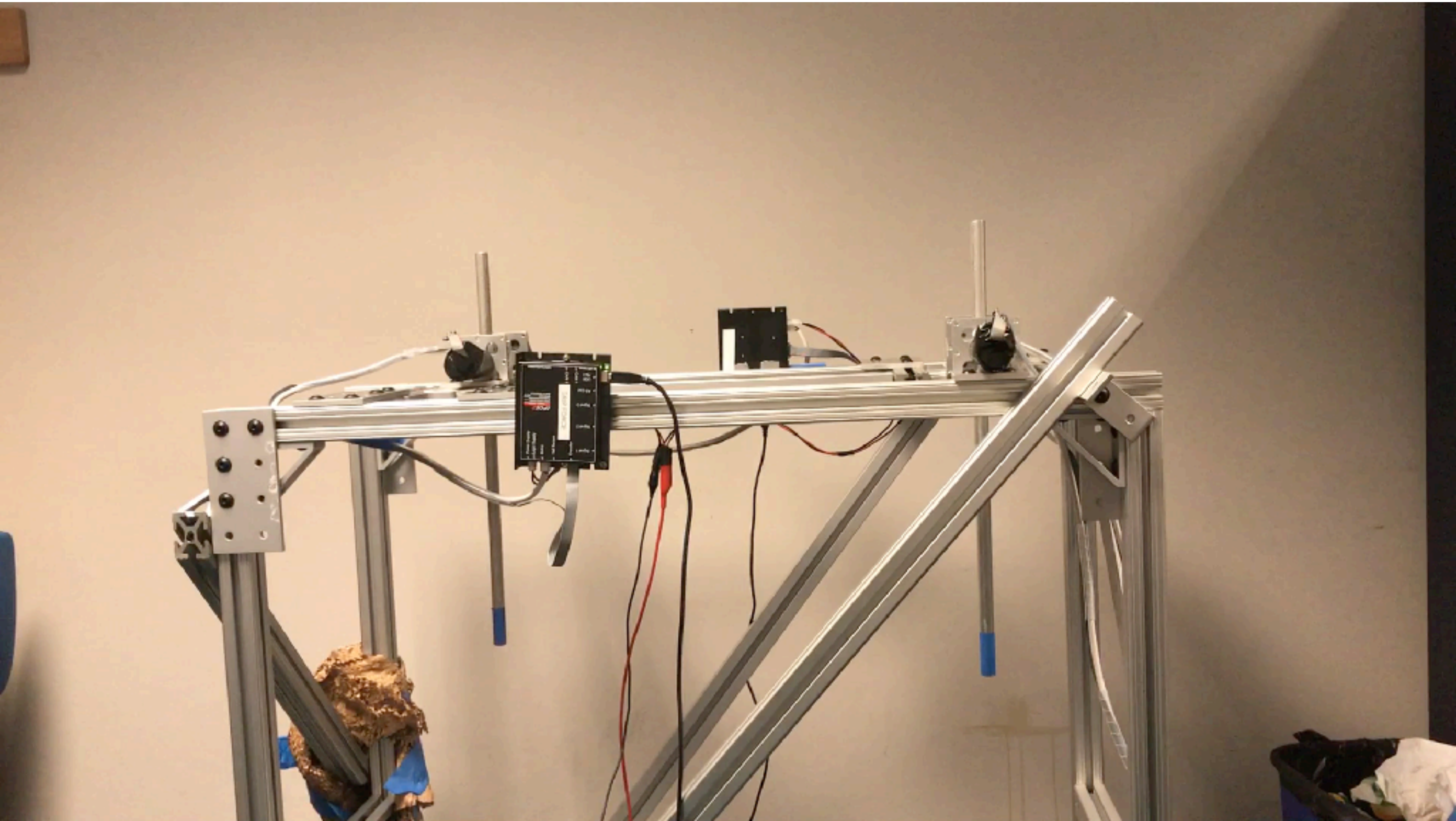
1. Implement algorithm using TensorFlow, training on simulation environment: Mujoco Pendulum from OpenAI Gym

2. Transfer the model trained on simulation to hardware

3. Train simultaneously on two hardware pendulums

4. Further training for several episodes (like 200 episodes) to converge the Policy on new model

DEMO



Appendix A
Programming with epos2

1. Follow install instruction to have library and driver installed

2. USB number: system assign a USB number on plugin according to order, like USB0, USB1

3. Node ID: the DIP switch on Epos2 card indicate the node number, like 1,2,3…

4. the controller init a handler based on USB number

5. write command use handler together with Node ID to control

6. make sure the usb are pluged in to port in right order and the node id is set correctly

Full code for Control and Test: https://github.com/guzhaoyuan/epos2

Appendix B
Encountered Hardware Problem

- shaft alignment, need fine tune to align shaft of the two servo and reduce friction

- cooling problem, the servo get super hot while continuous training - apply ice to cool servo down

- coupling friction not enough and slips - add more pressure use clamp coupling instead of side screw

- the pendulum rotate at fast speed may hurt people or damage the platform - soft limit the angular velocity