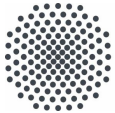


Universität Stuttgart

Intelligent Object Sorting using Deep Reinforcement Learning

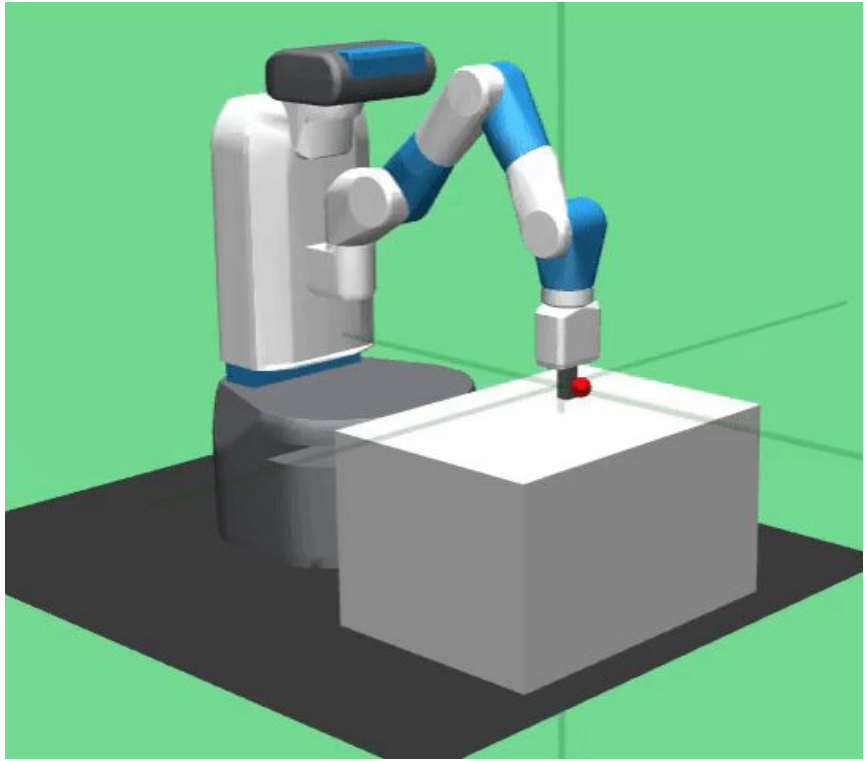
Robot & Computer Vision

Robotics Lab, Winter Term 2021-22

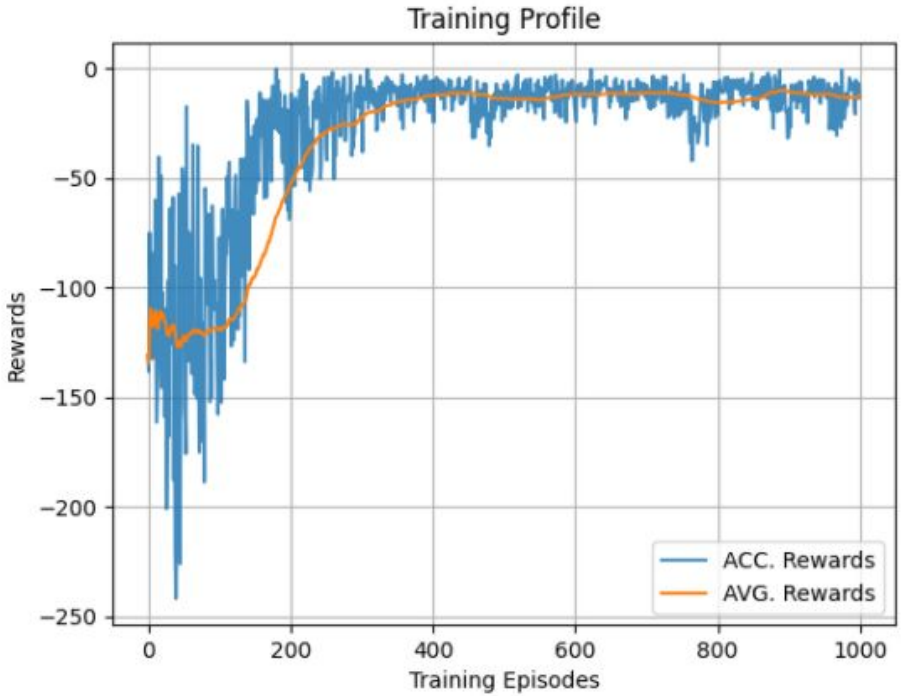
Kanishk Navale / 3437531

Olga Klimashevska / 3525388

Proof of Concept

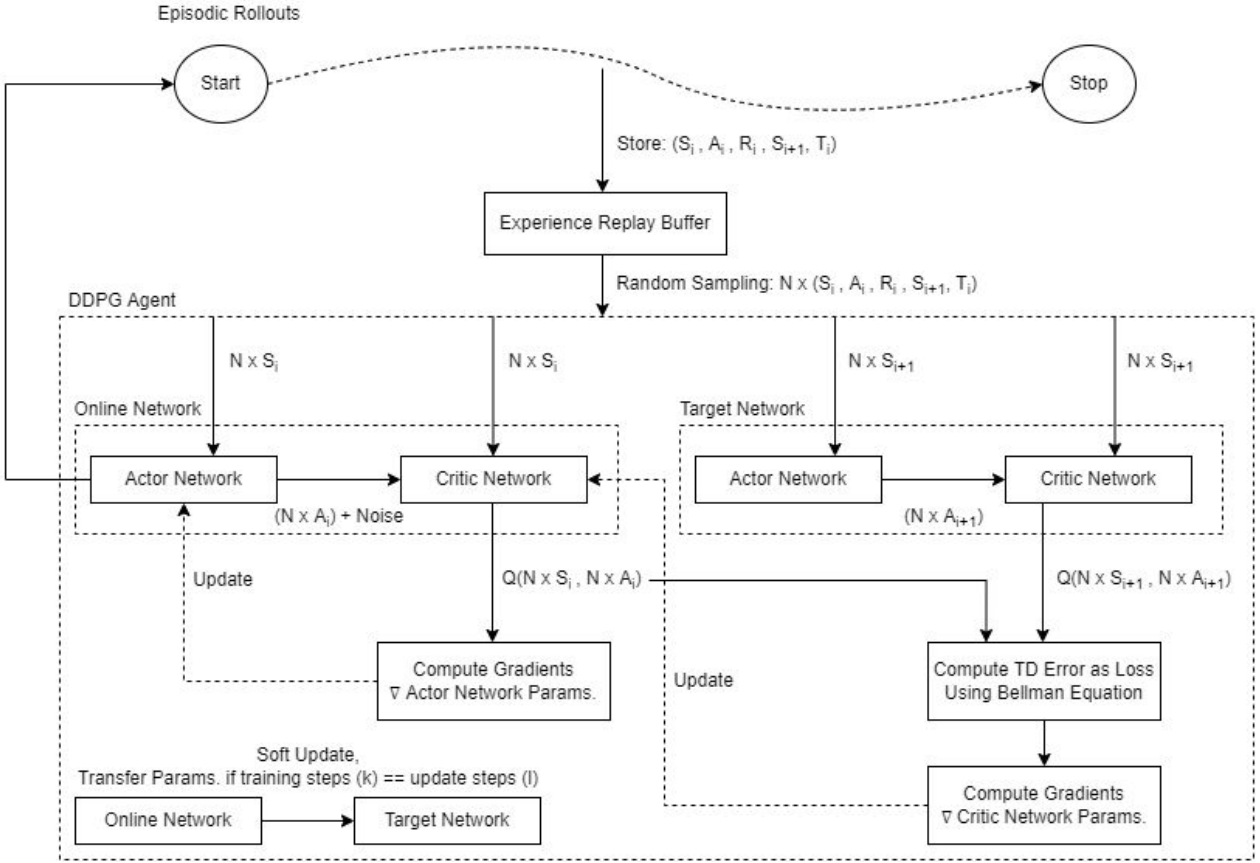


Benchmarking DDPG, TD3 & PPO on openai-gym ‘‘FetchReach-v1’’

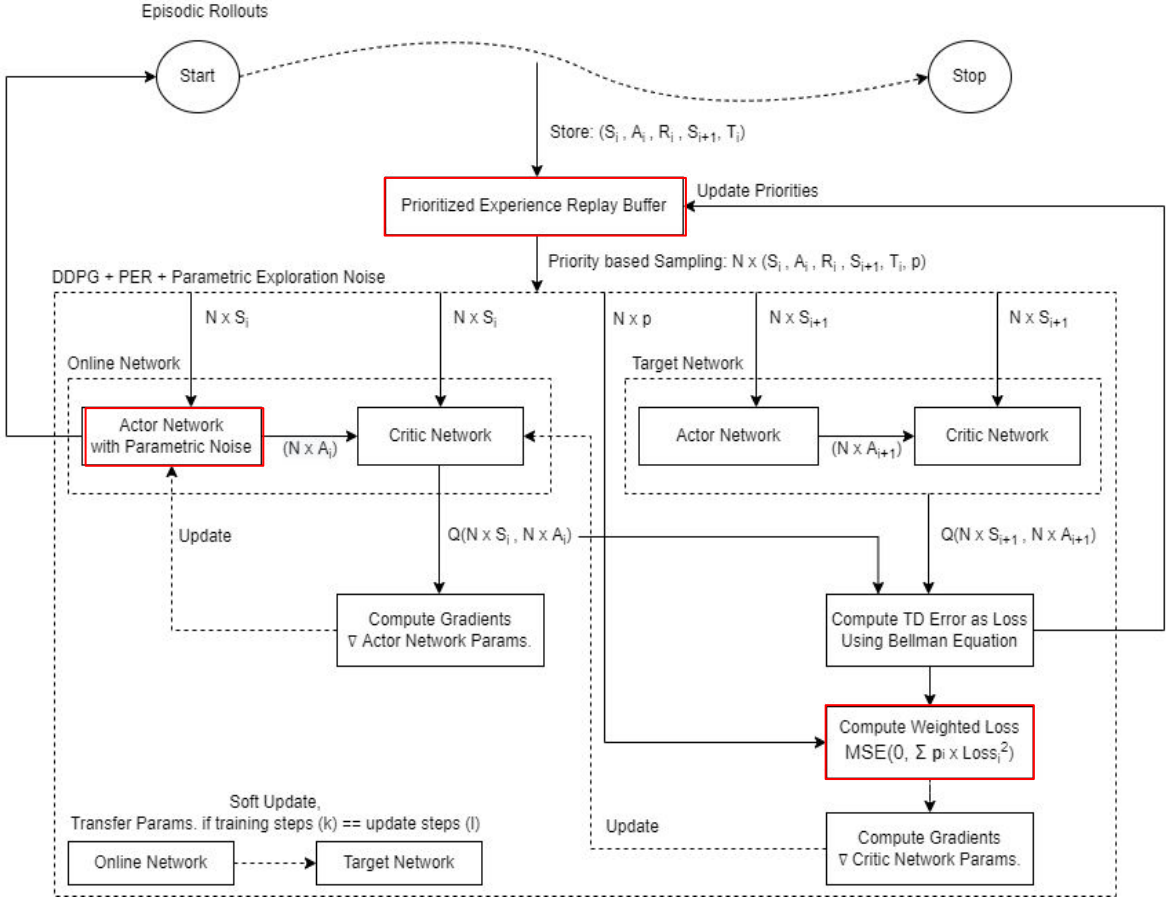


Training Profile of the DDPG Agent

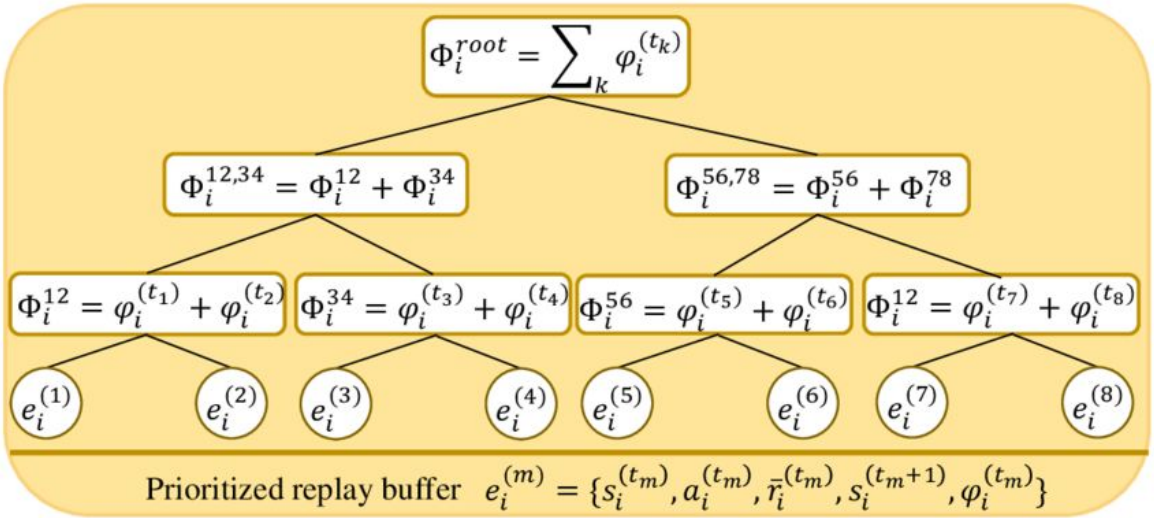
Vanilla DDPG Algorithm



Engineering a better DDPG Algorithm

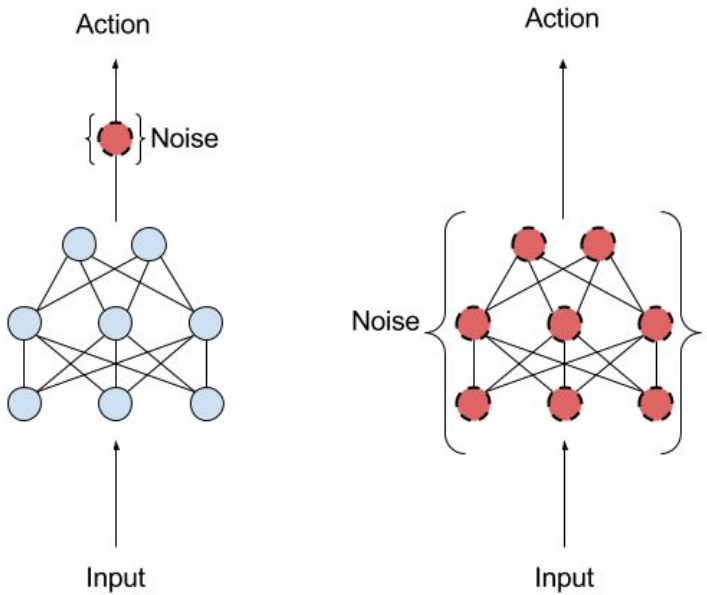


Prioritized Experience Replay Buffer



- Random Sampling of experiences is not used.
- Instead, newest experiences are sampled first.
- Then, sample experiences priorities wise based on TD errors.
- It enables the agent to relearn the experiences to get better rewards.

Parametric Exploration Noise



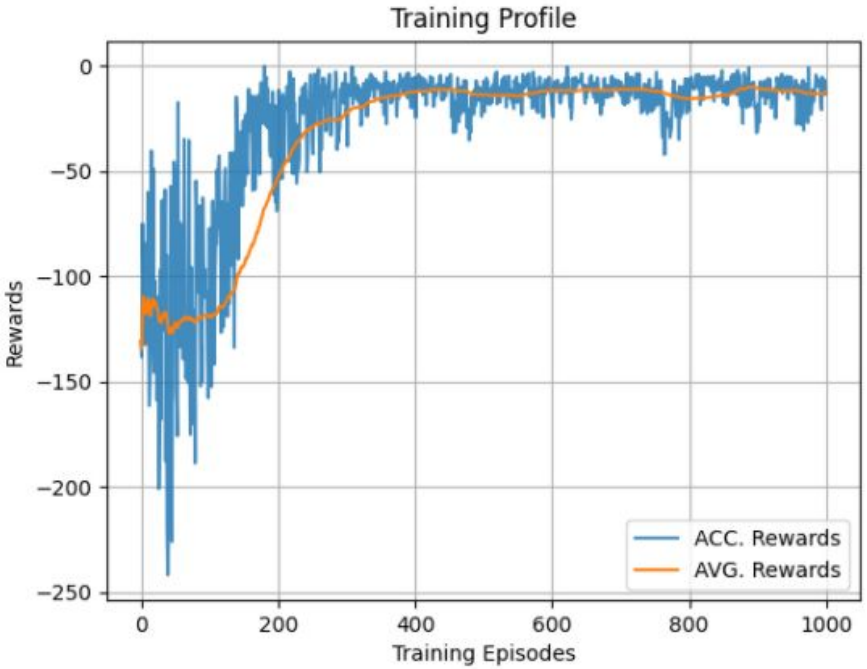
- Added Gaussian Noise is not used for exploration (left image).
- Instead, a new clone network of actor is created with noisy weights.
- This ‘Noisy Actor’ produces noisy action for exploration.

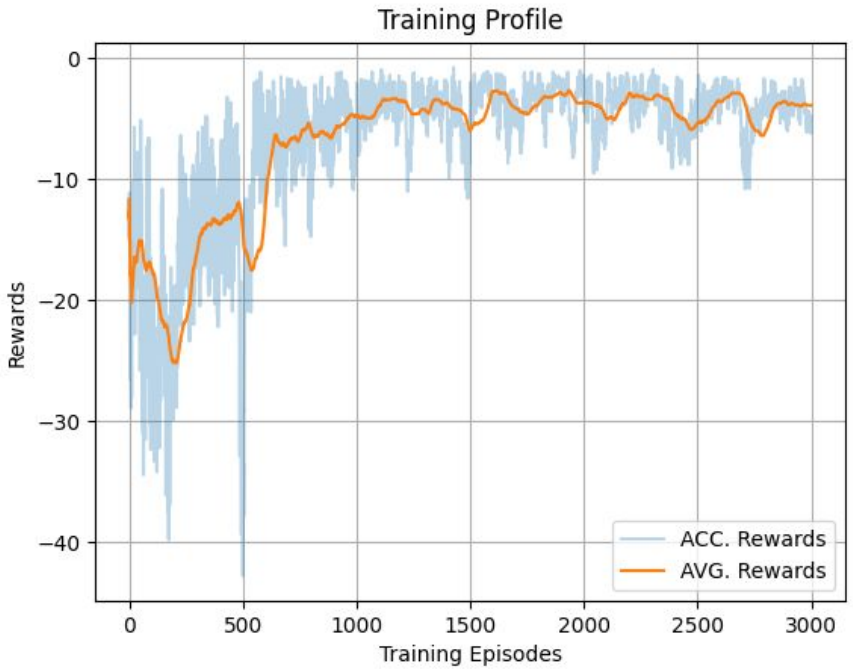
Building State & Action for Interaction

- For each play step in a game,
  - Build: state = Current Robot TCP(x, y, z) | Target Location P(x, y, z)
  - Compute: action = actor.choose_noisy_action(state)
  - Get: next_state, reward, done = env.step(action)
  - Reward = -1.0 * Euclidean Distance (Current Robot TCP, Target Location P)

- DDPG Agent is optimized to maximize the reward for each play step over the games.

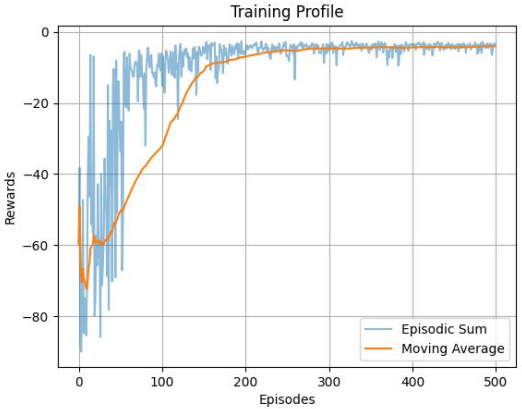
Comparison of Vanilla DDPG & Our DDPG

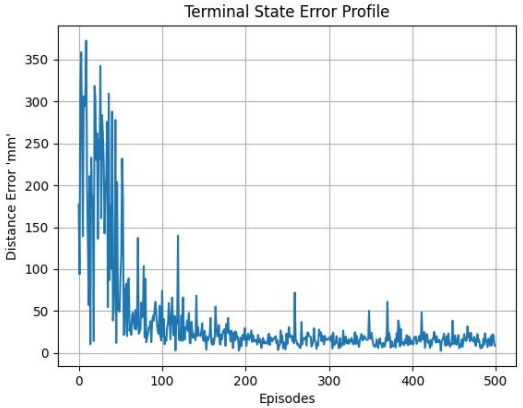




Result: The DDPG Agent is 5 times better (metric: training rewards) with PER & Parametric Exploration Noise.

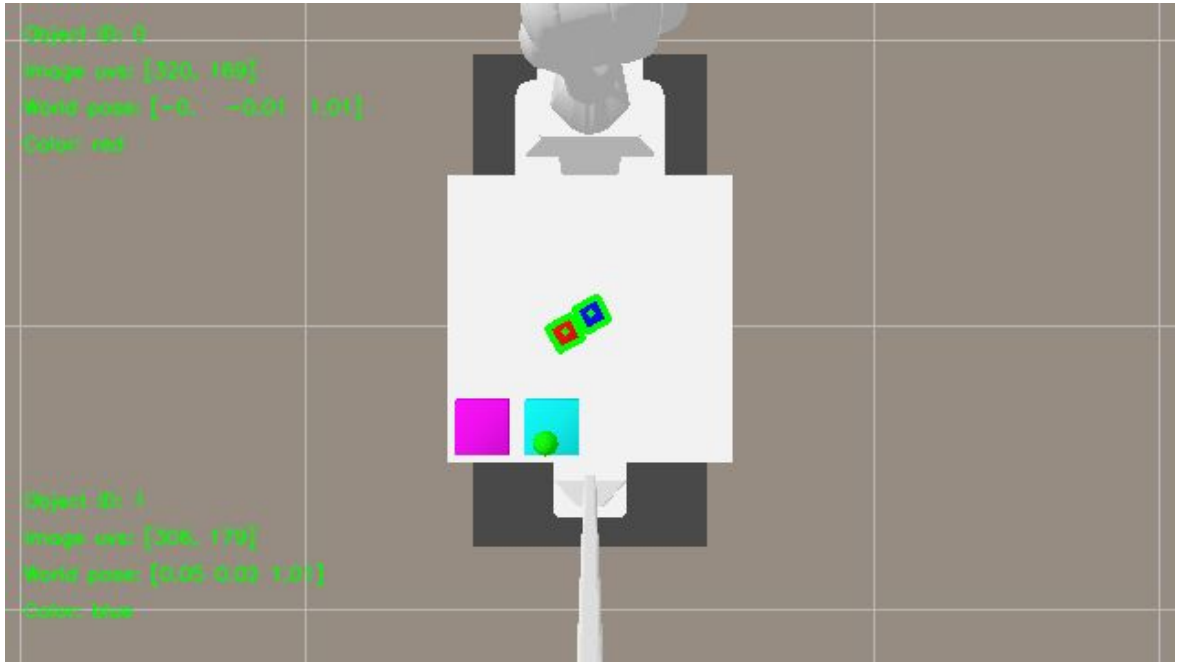
Training DDPG Agent for Robot Motion





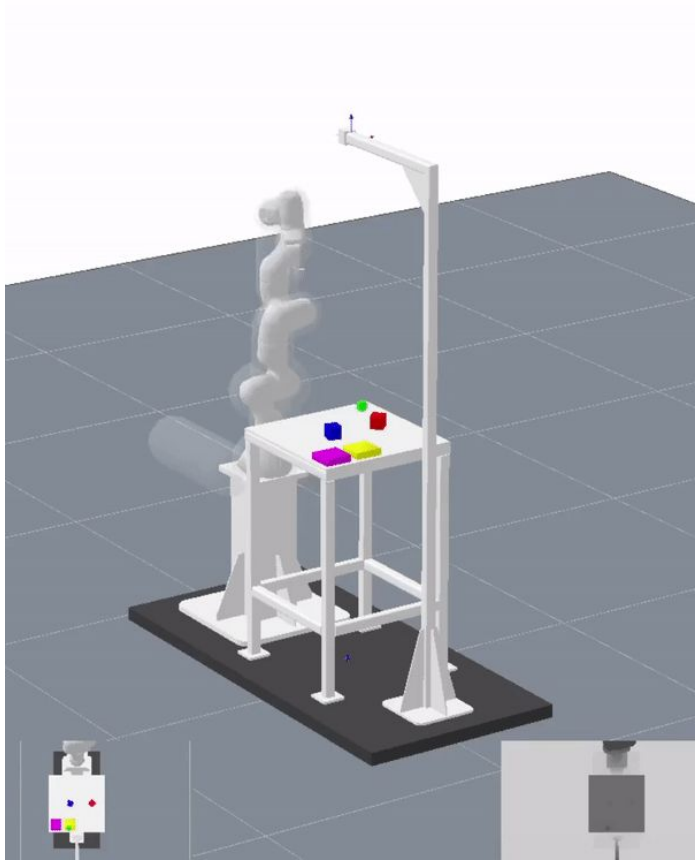
1. SolidWorks is used to develop the meshes.
2. Meshes are exported & imported in ‘rai’ after processing .urdf files.
3. The ‘gym’ wrapper is used to create ‘reach_gym’ to train the robot.
4. It takes 1Hr. to train the robot for 500 episodes.
5. The robot reaches any point in the Cartesian space with error of ±5mm.
6. The training occupies 3GB of GPU.

Pose Estimation Pipeline

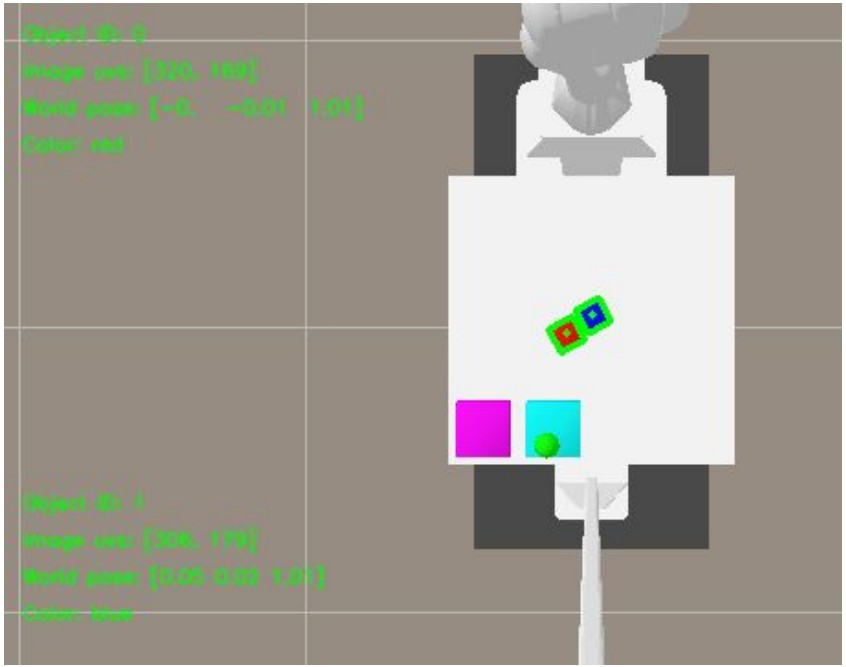


- Pose estimation is done for ‘red’ & ‘blue’ objects.
- Depth & RGB image is used to compute the use.

Object Tending



Process Logging




```
[
    {
        "Object ID": 0,
        "Camera Coordinates [u, v]": [
            320,
            169
        ],
        "World Coordinates [x, y, z]": [
            -0.0022170670613970446,
            -0.00854486748731096,
            1.0097603467432426
        ],
        "Color": "red"
    },
    {
        "Object ID": 1,
        "Camera Coordinates [u, v]": [
            306,
            179
        ],
        "World Coordinates [x, y, z]": [
            0.04528890767445167,
            0.02470116320227714,
            1.0080491988625047
        ],
        "Color": "blue"
    }
]
```


- We store the image from the Pose Estimation Pipeline as .png image.
- Also, the detailed object data in a ‘.json’ file for debugging purposes.