Cube Pushing with PPO: Simulation in MuJoCo and Deployment on the Real SoArm100

Abdulla Aljaberi, Abdulrahman Alsuwaidi, Salim AlBlooshi
(University of Girona / Reinforcement learning)
Girona, Spain

***Abstract*—This paper presents a two-part study on learning a cube pushing skill using reinforcement learning.**

the first part of this project is the virtual policy training of the neural network in the MoJoCo simulator. The neural network was trained to push a small cube from it's position to a user defined goal position using the end-effector of a 6-DOF robotic arm (SoArm 100). we tested both DDPG and PPO networks for this training phase that implemented the following reward policies: proximity to cube and goal, cube contact, cube grasping, and cube pushing.

the second part of the project is about real life integration with the SoArm 100. for this phase a equipment used are a logitic webcam, aruco marker labeled cube, SoArm 100, checker board, and a fixed base for the manipulator. the camera intrinsics and extrinsics were calibrated using the checker board to correctly track the cube through the aruco markers labels on the cube. the checker. the fixed base was used to ensure that the manipulator arm was constantly in a fixed position and then additional aruco markers where used to configure the transform relationship between the camera and SoArm 100. finally, MUJoCo and python were used as the main interface between SoArm100 and the trained Neural Network. this report will discuss the evaluation metrics in simulation and on the real robot, challenges encountered, and provide recommendations for robust real-world pushing.

***Index Terms*—Reinforcement Learning, PPO, Robotic Manipulation, Pushing, MuJoCo, SoArm100, Sim-to-Real, Vision, DDPG**

I. Introduction

Planar pushing is a fundamental manipulation skill used for object tracking and placement. It is challenging due to the multiple factors that must be taken into consideration like friction, environment, safety conditions, object weight, contact with object, robotic manipulator dynamics, and object dynamics which all can make learning unstable if the reward is poorly.

This work has two goals:

- **Simulation:** learn a policy in MuJoCo that pushes a cube to a goal located left of the cube.
- **Real robot:** deploy the learned policy on the real SoArm100 and evaluate performance under real-world conditions.

Contributions. We provide:

- A goal-conditioned MuJoCo environment for cube pushing with a clear observation/action definition.
- PPO training with a reward design that encourages sustained pushing (not just reaching).
- A deployment pipeline for the real SoArm100 including calibration, safety, and evaluation protocol.
- Practical discussion of issues in sim-to-real for pushing.

II. Related Work

Proximal Policy Optimization (PPO) is widely used in continuous control due to its stable policy updates and strong performance in MuJoCo benchmarks [1]. Earlier actor–critic methods such as Deep Deterministic Policy Gradient (DDPG) demonstrated that deterministic policies can be effective for continuous robotic control tasks [2]. Foundational principles of actor–critic and off-policy learning are discussed in [3]. .

III. Problem Formulation

We model the task as an MDP with continuous observations s and continuous actions a. The objective is to learn a policy $\pi(a|s)$ that moves the cube to a goal location within a limited time horizon.

A. Task Definition

Let $\mathbf{p}_c$ be the cube position and $\mathbf{p}_g$ the goal position. The goal is defined as a left offset from the initial cube pose:

$$\mathbf{p}_g = \mathbf{p}_c^{(0)} + [-\Delta_x,\ 0,\ 0], \tag{1}$$

with success when $\|\mathbf{p}_c - \mathbf{p}_g\|_2 < \epsilon$.

IV. Part I: Simulation in MuJoCo

A. Robot and Scene

We use a MuJoCo scene with a 6-DoF arm model (SoArm100) and a free-moving cube on a planar table. Control runs at 20 Hz. The goal is visualized as a green site marker (`push_goal`) for debugging.

B. Observation Vector

The observation is goal-conditioned and includes:

- Joint positions $\mathbf{q} \in \mathbb{R}^6$ and velocities $\dot{\mathbf{q}} \in \mathbb{R}^6$
- Cube position $\mathbf{p}_c \in \mathbb{R}^3$
- End-effector position $\mathbf{p}_{ee} \in \mathbb{R}^3$
- Goal position $\mathbf{p}_g \in \mathbb{R}^3$

Total dimension: 21.

C. Actions

Actions are 6D continuous joint target increments:

$$\mathbf{a} \in [-1, 1]^6, \tag{2}$$

scaled by an action factor and applied as small position-control deltas.

V. DDPG

A. Overview

The DDPG method was used to train a virtual robotic arm through a MuJoCo setup. Instead of fixed motions, it tested how to learn smooth signals that produce steady, target-oriented movements across variable joints.

The SoArm100 using DDPG was able to move in multiple directions, and manipulate the end-effector to touch and move a square block set on a flat surface. Instead of continuous motion updates, it received step-based angle targets delivered at set intervals.

B. Control and Learning Setup

Running at 20 Hz, the system sent out joint deltas - small angle changes - limited to ±0.05 radians per time step. That speed reached about 1 rad/s, which meant movement up to 57 degrees per second. Chosen this way, movements stayed steady and predictable through training.

DDPG worked well because it handles continuous actions while also giving precise control without uncertainty. DDPG uses an actor-critic setup which helps balance predictions from previous actions taken. Learning from past actions made things faster by mixing old data into each update. Noise added to each episode kept discovery active even when confidence grew.

C. Achieved Behaviors

One thing that worked well with DDPG is how smooth and arm movements were:

Joints stay under steady control, free from shaking or uncontrolled growth. Motion stays steady at levels matching how people actually handle objects. One thing stood out - DDPG learned very quickly to reach and touch the cube. additionally, instead of just touching the cube, it began to explore what came next after touching. Over time, actions started following a clear pattern, like steps in a sequence built by trial and error. With less punishment from the critic piling up, policies start fitting together better while the actor adjust in predictable ways.

One thing the study shows is how well DDPG connects complex scene data to proper joint movements in a real-like setup.

D. Training Cost and Practicality

Only when training settled down did patterns emerge - around one million simulation steps, roughly five thousand episodes. Clocking in at about fifty minutes on everyday machines, the system reached usable results without excessive demand. Useful hand-like actions took form under these conditions, proving effective exploration doesn't always need vast resources.

E. Observed Limitations

Though DDPG managed to learn simple actions, it struggled with tougher problems. It reacted strongly to how rewards were designed, often relying too much on gaps in feedback. Without limits like touch or friction penalties, outcomes got unstable fast. Such results point to well-documented flaws in free-space reinforcement setups.

TABLE I
SIMULATION RESULTS OF PPO-BASED PUSHING TASK.

Metric	Value	Notes
Success rate (%)	100	Single evaluation episode ($N = 1$)
Mean final distance (m)	0.0243	Distance to goal at success
Median steps to success	18	Successful episode only

F. Final remarks on the DDPG

DDPG worked well as a starting point for controlling robots in continuous settings. Learning stayed steady, movements felt natural, while still adapting to changes around it. Still, when things got messy - like collisions or extended timelines - the method fell short. That shortfall made clear why extra safeguards are needed, along with tougher techniques for complex tasks ahead.

VI. PPO

A. Reward Function

A key design goal is to avoid "hovering" near the cube without effective pushing. We use a dense reward:

- **Reach:** encourages the EE to approach the cube.
- **Goal progress:** encourages reducing cube-to-goal distance.
- **Push-left:** rewards leftward cube motion, gated by EE–cube proximity.
- **Success bonus:** large terminal bonus for reaching tolerance.
- **Action penalty:** encourages smooth actions.

B. Algorithm: PPO

We train PPO [1] using Stable-Baselines3 with an MLP policy (actor-critic). The policy is trained for a fixed number of timesteps and periodically saved.

C. Training Setup

Hyperparameters (example): learning rate 3×10^{-4}, $n\_steps = 2048$, batch size 64, epochs 10, $\gamma = 0.99$, $\lambda = 0.95$, clip range 0.2.

D. Simulation Results

We report:

- Success rate in simulation (%).
- Mean final distance to goal.
- Distance-to-goal curves over time.

VII. SIMULATION SETUP

The pushing task is implemented in a MuJoCo simulation environment. Figure 2 shows the simulated setup, which includes a SO-100 robotic arm, a cube, and a goal position on a flat surface.

The objective of the task is to push the cube toward the goal location. The cube to be pushed is shown in green, while

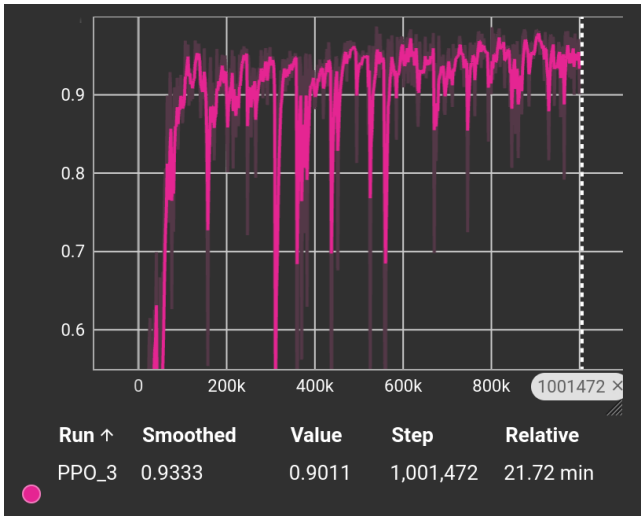

Fig. 1. Explained variance of the value function during PPO training, indicating stable and accurate value estimation.

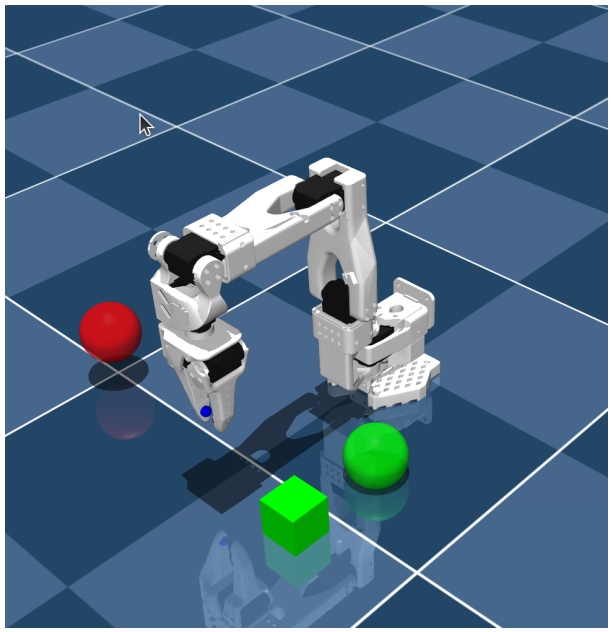

Fig. 2. MuJoCo simulation environment used for the cube pushing task with the SO-100 robotic arm.

the goal position is visualized using a green sphere. A red reference marker is included for visualization purposes.

The robot is controlled using continuous joint position commands. The simulation provides full state information, allowing safe and efficient training of the reinforcement learning policy before real-world deployment.

VIII. Part II: Real SoArm100

The first step to transfer the results of the simulation is to replicate the scenario from the simulation. The Idea is to build an setup that will hold the SoArm100, camera in a fixed position so that it ensure that the calibration and syncing process is consistent.

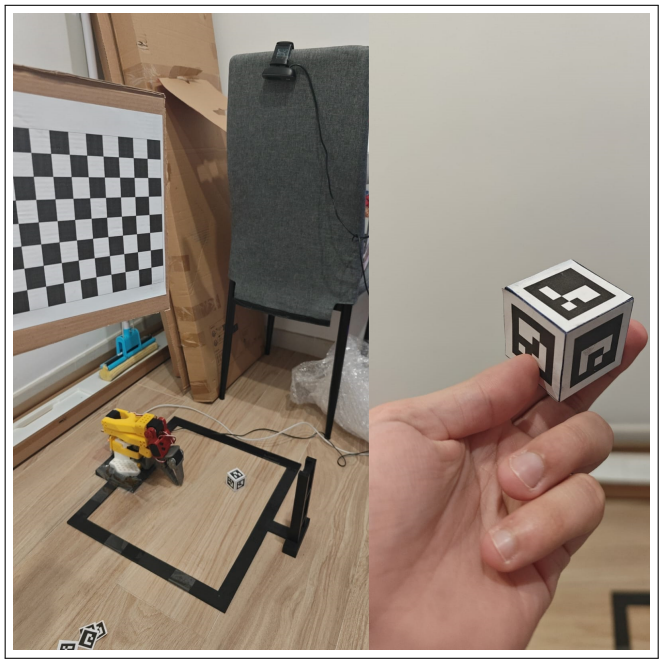

Fig. 3. Simulation training curve (exported from TensorBoard).

A. Hardware Setup

- SoArm100 robot arm (6-DoF) with position control interface.
- A 3D printed Arena to mount the SoArm100 arm
- A cube of known size with Aruco markers for each face
- A Logitec webcam used for the SoArm100 vision

B. Calibration and Frame Alignment

The goal is to first calibrate the camera to the current setup, ensure that the SoArm100 is synced between the simulation joint values and the real motors, and align the frames so that the real SoArm100 position is reflected onto the simulation and vise-versa. The step-by-step process is as follows:

- A 7x9 checkerboard was used to estimate the camera intrinsics and extrinsic
- Using the serial library and a custom script to calibrate the SoArm100 joint values from the motors and simulation to ensure both represent the same angles
- Using the Aruco markers and the Kabsch Algorithm to estimate the necessary transformation to align the camera world frame with the simulation world frame
- enforcing the Z value for the cube to be half the height, to reduce noise from the camera estimation

The result is an synced simulation that moves as the real SoArm100 moves. An additional feature is the ability to control the real SoArm100 from the simulation using the control sliders for each joint.

C. Safety and Hardware Constraints

The goal is to implement software-level protections that safeguard the mechanical integrity of the SoArm100 during deployment. These constraints prevent high-torque impulses

and ensure that the control signals remain within the physical capabilities of the actuators. The technical measures are as follows:

- Applying an action scaling factor to the model's output to ensure that joint target increments are small enough to prevent abrupt, high acceleration movements.
- Utilizing the numpy.clip function to enforce hardware joint limits, preventing the actuators from reaching mechanical limits or over-extending to unwanted locations
- Implementing a global velocity limit on the servo commands to cap the maximum execution speed, intended to keep the SoArm100 within a safe operating range

The result is a controlled execution environment that reduces the risk of unwanted behaviors during policy implementation. While the idea is to have the safeguards to preserve the SoArm100 integrity, and avoid damaging the hardware.

D. Sim-to-Real verification

Once the simulation setup was complete, some optimizations and adjustments were necessary in order to ensure that there is no problems once the AI is deployed for testing. Some of the aspects the needed verification are:

- tuning: adjusting the movements between the simulation and real SoArm100 to ensure proper mimicked motion
- **R**obustness: test across different cube starting positions and small disturbances.
- **D**omain randomization: randomize friction/mass in simulation, lighting and minor tweaks in the initial position of the SoArm100.

E. Proposed Real SoArm100 Evaluation Protocol

The goal is to establish a standard framework for quantifying the Sim-to-Real transfer quality, mirroring the performance metrics used in the MuJoCo simulation. The testing methodology focuses on three core indicators:

- Determining the Real Success Rate by recording the percentage of actual trials where the cube is successfully pushed into the defined goal within the episode time limit.
- Quantifying precision via Final Distance Error, using the Euclidean distance between the cube's final position and the target position at the end of the trial as a metric.
- Assessing Qualitative Contact Dynamics by visually inspecting the policy's ability to ensure stable grip contact and maintain a consistent push trajectory without missing the object.

The result is a projected set of benchmarks that would allow for a direct numerical comparison between the simulated baseline and the physical performance of SoArm100's, identifying specific areas of degradation in the transfer process.

F. Problems Encountered in Real Deployment

Although the simulation and robot were ready for actual trials, unfortunately, sudden movement broke the base joint due to an attempt to sync with the simulation. This means that in reality, no real attempts were made in order to get the data to benchmark the result of the PPO network. Things to note regarding the situation:

- Camera does not have the best depth perception, which cause a small misalignment between the cube and the simulation
- Initial Arm calibration is needed for every run to ensure proper syncing
- Delay between the simulation and hardware would create some gap when running the model.

The gap between the simulation and hardware was not fully realized to the result of the early termination of the robot, which forced the implementation to be halted.

TABLE II
REAL ROBOT EXPERIMENTAL RESULTS (TESTING ABORTED).

Metric	Value	Notes
Success rate (%)	0.0%	0/1 trials (Aborted)
Mean final distance (m)	N/A	Trials terminated
Failure modes	Structural Failure	Base fracture due to jitter

Fig. 4. SoArm100 broken on the floor after it decided enough is enough

IX. Discussion

Simulation results show that PPO can learn stable reaching and effective pushing with the proposed reward shaping. Real-world deployment highlights sensitivity to friction and calibration; conservative action scaling and careful frame alignment are essential. Randomizing cube start positions improves robustness, and domain randomization is a promising next step.

X. Conclusion

We presented a two-part pipeline: (I) training a cube pushing policy in MuJoCo using PPO with dense reward shaping, and (II) deploying it on the real SoArm100 with calibration and safety constraints. The contact-gated push-left term is important to produce sustained pushing rather than hovering. The Arm broker due to sudden movement which resulted in the lack of acquired data for the real robot. Future work will focus on stronger generalization across goals, more systematic domain randomization, and vision-based cube tracking. Other points to consider is hardware response to the simulation outputs to avoid damaging the hardware.

References

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2016.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.