Wobbler: Ball balancing robot arm

Jaydan Aladro and Guillaume Charron

April 27, 2024

Abstract

The goal of this project is to make a robot arm learning how to balance a ball while reaching a goal point in space. It will move a platform from point A to point B with a ball on top. Its goal is to maintain the ball at the center of the platform as much as possible, while moving the end-effector to the desired location. We will train the model in simulation and apply the learned algorithm to the real robot arm.

Our code: Wobbler

1 Introduction

This project aims to study the ability of a robot arm to adapt to real world physics by maintaining a round object at the center of a plate while moving the plate to a desired location. Round objects can easily roll and fall off the plate, which increases the complexity of the task. As we can see in

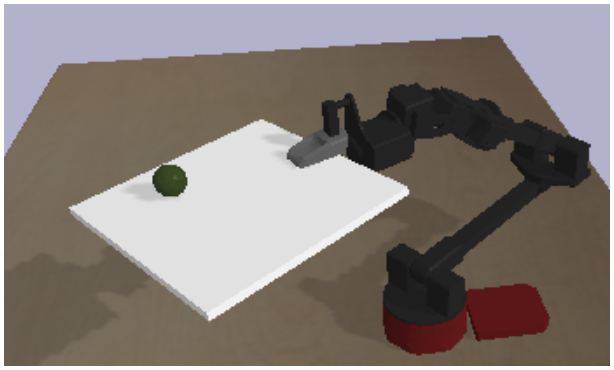


Figure 1: Example of ball on a plate in the gripper of a robot arm

Figure 1, the plate will be placed in the robot end-effector to simplify the task. Goals in terms of location (X, Y, Z) in a 3D plane in the reachable space of the robot arm will be specified for the robot to reach. This project aims to explore the capability of RL algorithms like SAC and PPO to learn how to balance a delicate object while learning to reach goal locations at the same time.

2 Related work

Earlier studies like those by [Khan et al., 2018] have utilized ROS for robot manipulator tasks, demonstrating the feasibility of similar ball-on-plate balancing acts. This foundational work paves the way for our project, which also aims to leverage such frameworks but with a heightened focus on advanced reinforcement learning techniques like SAC [Inc., 2024, Haarnoja et al., 2019] or PPO [Schulman et al., 2017] for enhanced adaptability and control precision. Additionally, methodologies

introduced by [Levine et al., 2016] and [Gu et al., 2016], involving deep visuomotor policies and domain randomization, respectively, offer valuable insights into robust policy learning that our project could adapt. These approaches, particularly the application of domain randomization, are crucial for developing models that perform well in the unpredictable dynamics of real-world settings.

What sets our project apart is the ambition to not only apply reinforcement learning algorithms in simulation but to extend these learned behaviors to physical robot systems, addressing the challenge of simulation-to-reality transfer. This effort is further distinguished by its approach to dynamic tasks, such as enabling the robot arm to catch or move with the ball, which introduces a level of complexity and real-time adaptability not commonly addressed in previous works. Our project also proposes the development of a goal conditioned algorithm to enable a user to specify a goal location for the robot to reach while maintaining the ball in balance. We aim to better understand the bridge between simulated training environments and real-world application, tackling complex, dynamic control tasks, and refining the integration of learning-based and traditional control strategies.

3 Background/math framework used

The states of the MDP for our project consists of the position of the end effector in 3D coordinates, the position of the ball relative to the center of the plate in 2D coordinates and the goal position relative to the end effector in 3D coordinates. The possible actions are the desired end effector positions and orientations. The reward function motivates the arm to keep the ball close to the center of the plate while moving the end effector towards the goal location.

The SAC [Inc., 2024, Haarnoja et al., 2019] algorithm is particularly noted for its sample efficiency and stability due to its entropy regularization component, which encourages the agent to explore more widely and ensures a more robust policy. The primary objective function for SAC can be represented as:

$$J(\pi) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right]$$

where $(s_t, a_t) \sim \rho_\pi$ denotes the state-action pairs sampled according to the distribution ρ_π under policy π, $r(s_t, a_t)$ is the reward function, α is a temperature parameter balancing the importance of the reward versus entropy and $\mathcal{H}$ represents the entropy of the policy, promoting exploration. The PPO [Schulman et al., 2017] algorithm, is renowned for its balance between computational efficiency and ease of implementation. PPO achieves stability and robustness in policy updates through a novel objective function that prevents excessively large updates. The primary objective function for PPO can be represented as:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ denotes the probability ratio of the action a_t given state s_t under the current policy π_θ to the old policy $\pi_{\theta_{old}}$, $\hat{A}_t$ is an estimator of the advantage function at time t, and ϵ is a hyperparameter that controls the clipping range, which moderates the update step to ensure moderate policy changes. This clipping mechanism is central to PPO's approach to maintaining a stable and efficient training process.
In our implementation, we combined GCRL with SAC and PPO, refer to algorithms in the appendix as 1 and 2. We used and modified CleanRL's Python implementation of PPO and SAC.

In our project, PyBullet simulates the physical interactions between the robot arm, the platform, and the ball. This simulation is used for training the RL model in a controlled environment where parameters like friction and ball size can be easily manipulated. We spent **many** hours configuring

the simulation so the arm holds a platform and a ball drops above it. For the real-world setup, OpenCV is used to track the ball in real-time using a webcam and various thresholding techniques (adding gaussian noise, conversion to HSV, applying masking, etc.), see Figure 2b.

4 Project Method

In our project, we applied Goal-Conditioned Reinforcement Learning (GCRL) to train the robot arm to balance a ball on the platform and reach a specific 3D position. The GCRL will help the arm learn to act based on both the immediate state, including the ball and arm's gripper positions, and the desired end state, defined by a vector of target coordinates and ball position. The rewards are set to encourage proximity to these target states. By diversifying the space (x, y, z) goals between episodes, the arm learns to adapt and balance achieving its position goals while maintaining the ball's stability. This method equips the robot with the ability to handle new goal combinations, enhancing its versatility in performing the dual tasks.

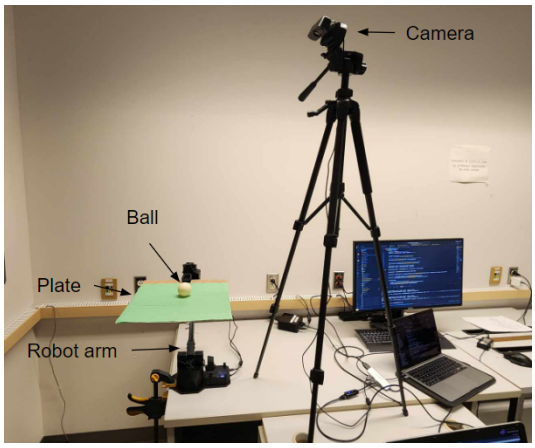
In order to obtain a more robust policy for the real life experiments, we employ various sim-to-real techniques that are implemented using Gym wrappers. We added a history of past observations to the observations at each step, alongside the last performed action to give the agent more contexts about the current state of the environment and the possible outcomes. We also added a random number of null actions at the start of each episode to cope with the eventual initialization delays, gaussian noise to the actions and observations to account for the similar noise that exists in the motors/communications in real life and a random number of repeated action to help the agent understand that the real hardware may need to receive the same action multiple times before it is actually executed.

To detect the position of the ball on the platform, we utilize computer vision techniques, specifically image segmentation and thresholding, to isolate the ball from its background. OpenCV (Open Source Computer Vision Library) is employed for this purpose, being a library extensively used for real-time computer vision applications. In a practical scenario, a camera would be mounted on the wrist of the robotic arm or on a tripod at a certain height to oversee the platform and the ball. In our project's context, we apply a mask to the camera feed to detect the ball, crucial for calculating the reward function based on the ball's distance from the center of the platform. This process entails converting the camera's RGB image into a binary mask that exclusively highlights the ball, likely through color space conversion and thresholding techniques (see Figure 2b).

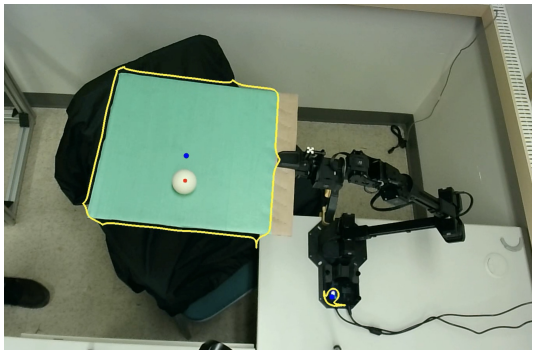
Additionally, the NUC provided with the arm gave the end-effector's positions in the 3D space in real time by calculating it through the arms current joint angles. This setup is pivotal for our system to understand and adjust the arm's position relative to the desired coordinates, thereby enabling precise control over the arm's movements and the balancing task.

The components of our reward are as follow:

- $r_1 = -\alpha_1 \cdot \exp(d_{ball\_center} - radius\_plate)$, where d_{ball_center} is the Euclidean distance between the ball and the center of the platform, $radius\_plate$ is a parameter controlling the radius of the goal center area and α_1 is a weight that determines the relative importance of this term in the reward function.
- $r_2 = \alpha_2 \cdot g_{arm\_final}$, where g_{arm_final} is 1 or 0 depending if the arm's end-effector reached the target position (x, y, z) within a $radius\_goal$ threshold, and α_2 is a weight that determines the relative importance of this term.
- $r_3 = -\alpha_3 \cdot \exp(g_{arm\_(x,y,z)} - radius\_goal)$, where $g_{arm\_(x,y,z)}$ is the Euclidean distance between the arm's end-effector and the target position (x, y, z), $radius\_goal$ is a parameter controlling



(a) Real life setup with camera placement



(b) Application of ball and plate tracking through video with OpenCV

Figure 2: Real setup and its OpenCV tracking application

the radius of the goal area, and α_3 is a weight that determines the relative importance of this term.

- $r_4 = -\alpha_4 \cdot h_{arm\_(z)}$, where $h_{arm\_(z)}$ is the height distance between the ball and the platform, and α_4 is a weight that determines the relative importance of this term. This reward term is used to force the robot to keep the ball on the platform without making it bounce in the air.
- $r_5 = -\alpha_5 \cdot t_{platform}$, where $t_{platform}$ is the absolute angle on the x axis of the platform and α_5 is a weight that determines the relative importance of this term. This component helps prevent the arm from tilting the platform forward and backward to make the ball bounce.
- $r_6 = \alpha_6 \cdot duration$, where $duration$ is the length of the episode and α_5 is a weight that determines the relative importance of this term. This component helps the robot to try to keep the ball in the center of the plate for longer instead of dropping it to minimize its negative rewards.

The total reward R at each timestep can then be expressed as: $R = r_1 + r_2 + r_3 + r_4 + r_5 + r_6$
Our SAC formula mixed with GCRL would therefore look something like:

$$J(\pi) = \mathbb{E}_{(s_t,a_t,g_t)\sim\rho_\pi}\left[R(s_t,a_t,g_t) + \alpha\mathcal{H}(\pi(\cdot|s_t,g_t))\right]$$

The primary objective function for PPO with GCRL can be represented as follows:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)\hat{A}_t(s_t,a_t,g_t,R), \text{clip}(r_t(\theta),1-\epsilon,1+\epsilon)\hat{A}_t(s_t,a_t,g_t,R))\right]$$

where $\hat{A}_t(s_t,a_t,g_t,R)$ is the estimator of the advantage at time t, which evaluates the relative value of taking action a_t in state s_t towards achieving goal g_t, using the reward R .

5 New skills learned

This project helped us learn new skills like policy training, applying code (learned policy) on real hardware and optimizing reinforcement learning algorithms by doing hyperparameters searching. We also learned how to adapt and use a simulated environment to train a reinforcement learning agent. Most importantly, we learned how to leverage the modularity of the simulators to train a more robust policy capable of bridging the gap between simulation dynamics and real-world scenarios. Additionally, we learned how to setup a real world scenario compatible to a simulated environment to deploy a trained policy on physical hardware. We also experienced first hand how sim-to-real can be a really challenging and complicated task.

6 Experiments and Analysis

6.1 Domain Randomization

In our simulation, domain randomization was applied at each timestep to introduce minor variations to the more significant ones introduced at each reset, as detailed in Table 1a. These subtle shifts are designed to mimic the nuances and unpredictability of real-world dynamics. The ranges for parameter sampling, as presented in the table, were established through a methodical approach that included the analysis of simulation dynamics, insights gathered from community forums, and extensive iterative experimentation. This process ensured a balance between realistic variation and computational tractability.

6.2 Reward engineering

Our primary objective is to maintain the ball's position at the center of the plate, a goal that is reflected by assigning significant weight to the ball's distance from the center, as this directly correlates with our main task of maintaining balance. The weights of the reward components were adjusted based on what we observed from the robot's performance during the learning process, ensuring that the reward system adapts to encourage the correct balance of behaviours. While the main focus has been on balancing, it has become apparent in retrospect that the weighting attributed GCRL aspects could have been increased. This adjustment may prevent the scenario where the agent overemphasizes balance at the cost of other important tasks. The final weights applied in our experiments are detailed in Table Table 1b.

Table 1: Simulation Parameters and Environment Rewards

(a) Domain Randomization (Step and Reset)

Dynamics Type	Reset	
	Min	Max
Ball **position**	plate min(x,y)	plate max(x,y)
Ball **size**	0.4	0.9
Ball **mass**	0.00005	0.005
Ball **rolling friction**	0.1	1
Ball **lateral friction**	0.2	0.8
Ball **spinning friction**	0.01	0.5
Plate **lateral friction**	0.1	0.6

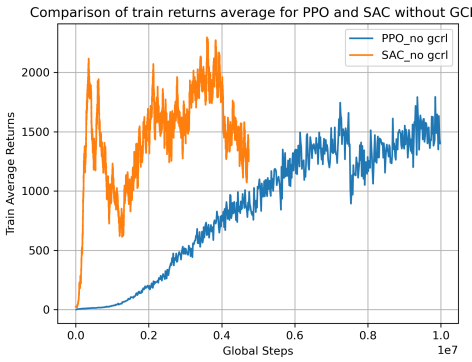
(b) Environment's Reward Components

Reward Type	Reward Weight
Ball distance from center of plate	1
Duration of ball on the plate	0.5
Height of ball from plate	0.5
Over tilting plate (y,z)	0.2
Distance of gripper from goal	0.5
Gripper reaching the goal	2

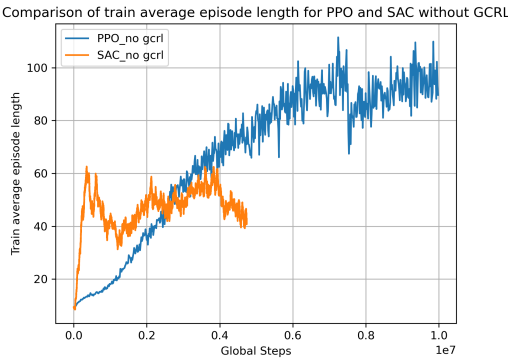
Note: The coloured rewards weights are only applied when we set GCRL in our environment.

6.3 RL algorithms

1. **Comparison between SAC and PPO**
   As we can see in Figure 3, SAC is less stable than PPO, especially in the ball-balancing only task. Despite SAC achieving reasonably high average returns in this task, the average episode lengths for both this and the goal-conditioned task are significantly shorter than those achieved with PPO.
   This behavioral discrepancy can likely be traced back to SAC's tactical preference for executing smaller, more conservative movements such as less tilting of the platform and reduced bouncing of the ball. While such a strategy is adept at garnering immediate high rewards by maintaining a semblance of balance, it inherently lacks the capacity for dynamic correction. Consequently,

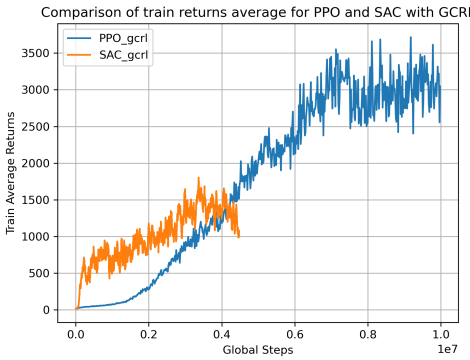


i) Training average returns

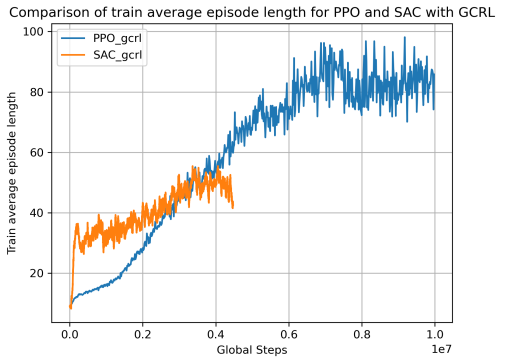


ii) Training average episode lengths

a) Only ball balancing task



i) Training average returns



ii) Training average episode lengths

b) Ball balancing task and goal conditioning

Figure 3: textbfComparison between SAC and PPO. The top row presents results for the ball balancing task only, while the bottom row includes results with goal conditioning (reaching a point in the 3D space while balancing). On the left are training average returns, and on the right are training average episode lengths. In both tasks, SAC (orange) is less stable than PPO (blue), achieving lower average episode lengths.

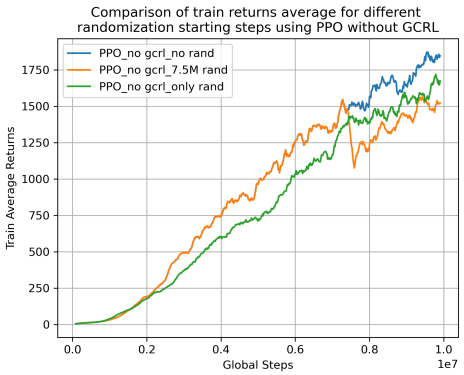
when an unexpected perturbation occurs, like a sudden jerk of the arm, SAC's predilection for subtlety becomes its downfall. Without the ability to execute larger, more compensatory movements, SAC is unable to recover the ball's trajectory effectively, leading to a higher propensity for the ball to be dropped or control to be lost in contrast to PPO's more robust response mechanism (see video *Simulations/SAC/rl-video-episode-7800.mp4* in the drive linked below for a good example).

Furthermore, SAC has a harder time learning to move to a point in the 3D space (goal) while balancing the ball. This could be explained by SAC's heavy relying on entropy as part of its reward mechanism. It's encouraged to explore and can lead to more efficient learning in some contexts, but it will also result in more conservative actions that, although initially yield high rewards, might not contribute to the robustness required for the compound task of balancing and reaching a goal simultaneously. PPO, by contrast, tends to take bolder steps due to its clipping objective which may better handle the multi-faceted nature of the tasks, as seen by the longer episode lengths. Additionally, PPO's objective function, with its separate clipping range for the advantage estimator, allows it to more effectively integrate and balance the multiple objectives of positioning and balancing, leading to more sustained control over the ball, as evidenced by

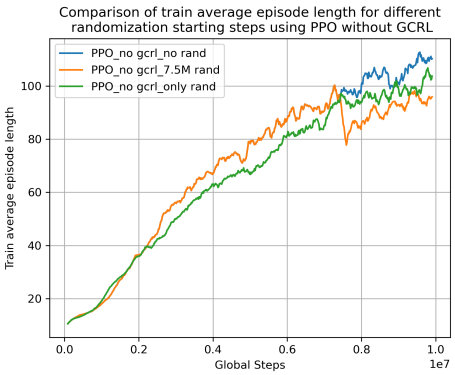
the stable task performance metrics.

2. **Domain randomization**

   When comparing domain randomization introduced after 7.5M steps and from the beginning of the training (with a training without randomization as baseline) like shown in Figure 4, we can see that delayed randomization does not always reach higher performance than only randomization. Even if the values before adding randomization are higher, the significant drop in performance caused by this addition makes the final performance lower than a training done only with randomization for the ball balancing only task. This could be explained by the fact that the arm is not acclimated to these new changes and has never experienced them so it can't adapt well For the goal-conditioned task, the random goal positions generated each episode introduce greater variability in outcomes (e.g., the arm inadvertently reaching the goal, or the goal materializing adjacent to the starting position). This variability implies that additional experiments and fine-tuning are required to discern a more distinct pattern in the data for this specific task.

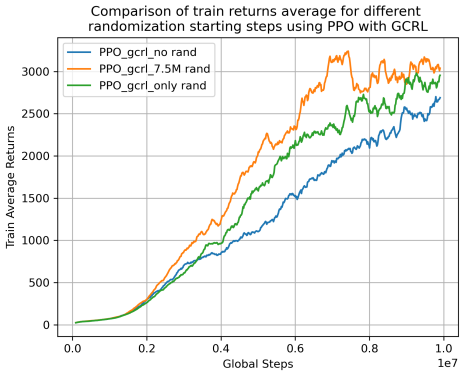


i) Training average returns

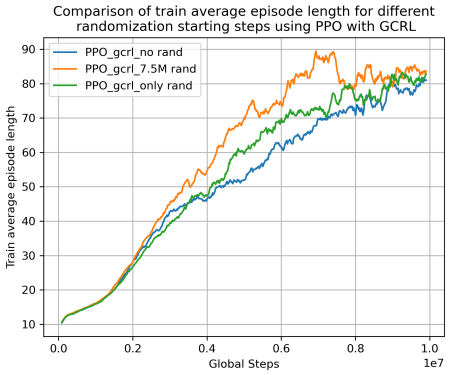


ii) Training average episode lengths

a) Only ball balancing task



i) Training average returns



ii) Training average episode lengths

b) With goal conditioning

Figure 4: **Comparison of different randomization starting steps**. The top row presents results for the ball balancing task only, while the bottom row includes results with goal conditioning. On the left are training average returns, and on the right are training average episode lengths. Results are smoothed using a moving average of size 10. Randomization introduces at 7.5M steps provides better results than at the beginning for the goal conditioned task, but not for the only ball balancing task.

3. **Transfer to real-world**
   When transferring the learned policies to the real world setup and hardware, a variety of challenges arose. The distance obtained with the ball tracking script (as shown in Figure 2) was in pixel references and had to be translated to real world units that the policy was used to. To do that, we simply scaled the distance by the ratio of the detected plate width and the plate width in the simulation. Additionally, the inverse kinematics function provided with the WidowX hardware was systematically refusing the end effector orientation outputted by the policy. To be able to make the robot move and execute the policy's action, we had to scale down the orientation values by a factor of 0.01 (obtained by trial and error). Finally, the communication delay alongside the action execution delay of the robot arm caused the ball to fall off the plate because the reaction time of the arm was not fast enough to recover from an executed action that made the ball move rapidly (as shown in the videos that can be accessed with the link in the next section). To cope with this problem, a delay or a damping method would've been necessary during training to force the agent to learn smaller action that would be executed slower and in a smoother way on the real hardware.

7 Video Results

Here you will find a Drive link with video results from our 8 experiences in simulation and videos of our real life application: Video results

8 Conclusions

Combining all the discussed methodologies allowed for the creation of a system where the robot arm, guided by PPO or SAC, learns to balance and move the ball to target locations based on visual inputs and simulated physics. This approach showcase the power of combining RL for decision making, computer vision for perception, and physics simulation for realistic interaction modeling in robotics. PPO has been identified as more suitable for the explored problem of ball balancing and goal reaching tasks, while the introduction of domain randomization at a later step has been shown to be less efficient than introducing domain randomization from the beginning of the training. Although some challenges were faced during the sim-to-real transfer, many lessons have been learned from this end-to-end experience and future work based on the explored tasks and experiences could increase the performance of the real world hardware by a significant amount, for instance by adding methods to cope with the communication and action execution delays, and obtain smoother policies.

9 How was the work divided?

Jaydan Aladro: Implemented the learning algorithms (PPO and SAC) using CleanRL and added the goal-conditioned version of the environment. Added random goal location and visualisation for the target goal. Tuned the hyperparameters of the reward components and domain randomization components. Worked on real life implementation tuning and fixing.

Guillaume Charron: Modified the provided simulator for the robot arm to include the custom reward function (distance to the center of the plate), the plate in the gripper and the rolling ball with customizable dynamics. Implemented the different wrappers for the sim-to-real generalization methods (gaussian noise, history buffer, etc.). Implemented a ball tracking script for the real-world setup using OpenCV and integrated it with the project code. Worked on real life implementation tuning and fixing.

References

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, 2016.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

The MathWorks Inc. Train sac agent for ball balance control, 2024. URL https://www.mathworks.com/help/reinforcement-learning/ug/train-sac-agent-for-ball-balance-control.html.

Khasim Khan, Revanthraghuram Konda, and Ji-Chul Ryu. Ros-based control for a robot manipulator with a demonstration of the ball-on-plate task. 2:113–127, 01 2018. doi: 10.12989/arr.2018.2.2.113.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

A PPO and SAC algorithms

Goal-conditioned version of SAC algorithm:

Algorithm 1 SAC with goal-conditioned RL

Init $\phi' \leftarrow \phi$ and $\theta' \leftarrow \theta$ to a random network and $D \leftarrow \{\}$
while true **do**
  Choose a goal $g \sim p(g)$
  for $t \in 0, ..., T$ **do**
    take some action $\mathbf{a}_t$ from $\pi(\mathbf{a}_t|\mathbf{s}_t, g, \theta)$ and receive $\{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}\}$, add to D
    Sample batch of data $\{\mathbf{s}_i, \mathbf{a}_i, r_i, g, \mathbf{s'}_i\}_{i=1}^N$ from D
    $y_i \leftarrow r_i + \max_{a'} Q(\mathbf{s'}_i, \mu(\mathbf{s'}_i, g, \theta'), g, \phi')$
    Update critic by minimizing the loss: $\frac{1}{N}\sum^N ||(Q(\mathbf{s'}_i, \mathbf{a}_i, \phi, g) - y_i)||^2$
    $\theta \leftarrow \theta - \hat{\nabla}_\theta J_\pi(\theta)$ (Update actor using the sampled policy gradient)
    $\alpha \leftarrow \alpha + \lambda \nabla_\alpha J(\alpha)$
    Update $\theta' \leftarrow \rho\theta' + (1-\rho)\theta$ and $\phi' \leftarrow \rho\phi' + (1-\rho)\phi$
  end for
end while

Goal-conditioned version of PPO algorithm:

Algorithm 2 PPO with goal-conditioned RL

Init $\phi' \leftarrow \phi$ and $\theta' \leftarrow \theta$ to a random network and $D \leftarrow \{\}$
while true **do**
  Choose a goal $g \sim p(g)$
  for $t \in 0, ..., T$ **do**
    take some action $\mathbf{a}_t$ from $\pi(\mathbf{a}_t|\mathbf{s}_t, g, \theta)$ and receive $\{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}\}$, add to D
    Sample batch of data $\{\mathbf{s}_i, \mathbf{a}_i, r_i, g, \mathbf{s'}_i\}_{i=1}^N$ from D
    $y_i \leftarrow r_i + \gamma V(\mathbf{s'}_i, g, \phi')$
    for each epoch
      Calculate $\hat{A}_i = y_i - V(\mathbf{s}_i, g, \phi)$
      Update critic by minimizing the loss: $\frac{1}{N}\sum^N ||(\hat{A}_i)||^2$
      Update policy by maximizing the clipped objective:
        $\mathcal{L}^{CLIP}(\theta) = \mathbb{E}_i\left[\min(r_t(\theta)\hat{A}_i, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i)\right]$
    Update $\theta' \leftarrow \rho\theta' + (1-\rho)\theta$ and $\phi' \leftarrow \rho\phi' + (1-\rho)\phi$
  end for
end while