

Master Thesis : Exploiting reinforcement learning to improve robotic throws

Auteur : Louette, Arthur

Promoteur(s) : Ernst, Damien

Faculté : Faculté des Sciences appliquées

Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée

Année académique : 2022-2023

URI/URL : <http://hdl.handle.net/2268.2/17705>

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.



Master's thesis

in the degree program
MSc. in Data Science and Engineering

Reinforcement learning to improve robotic throws

Submitted by

Arthur Louette

Matr. Nr.: 180981

Academic Year 2022-2023

at the University of Liège

School of Engineering and Computer Science

University supervisor: Pr. Damien Ernst

Company: GeMMe

Technical supervisor: Mr. Robert Baudinet

GeMMe

GeMMe is a distinctive research group located in Wallonia, dedicated to the innovative development and management of mineral and metallic resources. The group's research draws from traditional disciplines like mining, metallurgy, and civil engineering, but in recent years, it has shifted its focus towards unlocking the potential value in industrial solid wastes, end-of-life consumer goods, and complex georesources. The research activities of GeMMe are divided into three main units: the Georesources-GeoImaging group, the Mineral Processing & Recycling group, and the Construction Materials group. Each of these units possesses its own well-equipped experimental lab, facilitating innovation through strategic approaches.

Liège, June 9, 2023

Arthur Louette

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my thesis advisor, Professor Damien Ernst, whose expertise, understanding, and patience, added considerably to my graduate experience. I appreciate his ability to guide me throughout this research.

I would also like to express my gratitude to my internship advisor, Robert Baudinet. His macro supervision ensured the smooth progression of my internship and was instrumental to my professional development. The trust the lab placed in my abilities, along with their strategic guidance, played an essential role in shaping this enriching experience.

I would also like to thank Guillaume Drion and Pierre Sacré for serving on my thesis committee. I am sincerely grateful for their time and effort in reviewing my work and providing valuable feedback.

Special thanks go to my colleagues at the GeMMe lab for their assistance, insightful conversations, and the stimulating environment they provided during my study.

I would also like to express my gratitude to my family for their unwavering support and encouragement throughout my academic journey. Their constant belief in my abilities continued to motivate me to strive for excellence.

Finally, I wish to extend my thanks to my friends, whose continued support, encouragement and good humor made this journey an unforgettable experience.

Abstract

This Master’s thesis explores the potential of using reinforcement learning-based approaches to improve the efficiency of industrial sorting processes in the recycling industry, specifically focusing on enhancing the operational capability of robots. Current robotics have shown potential in handling and manipulating objects, however, the complexity of throwing tasks presents a significant challenge. The research conducted as part of this internship at GeMMe, a laboratory specializing in sensor-based sorting, aims to overcome this challenge by developing a control policy that enables robots to accurately throw objects into buckets, thereby increasing the speed of the sorting process.

To address this challenging task, reinforcement learning algorithms TD3, SAC, and PPO, were trained and analyzed in a simulated environment developed using PyBullet. This environment offered a simplified representation of the real-world task, serving as a safe and efficient platform for the training of agents. Furthermore, hyperparameter optimization was conducted using an Optuna study to enhance the learning process. Domain randomization was also implemented to bridge the sim-to-real gap and increase the robustness of the models to real-world variability. The final stage involved integrating the model into a working system, optimizing a communication process for effective information transfer. The performance of the models was assessed in both the simulation and real-world scenarios, offering valuable insights into their transferability and robustness.

In simulation, the best model demonstrated 94.75% accuracy while beating the pick-and-place baseline in terms of speed. However, when tested in real-world scenarios, there was a decrease in performance, but the findings showed promising potential for future improvements and the application of these methodologies in practical settings.

Table of Contents

1. Introduction	1
1.1. Context	1
1.2. GeMMe	3
1.3. Research problem and questions	5
1.4. Research goals and objectives	6
1.5. Scope and limitations of the study	7
1.6. Structure of the thesis	8
2. Literature Review	9
2.1. Theoretical Background	9
2.2. Problem formulation	13
2.3. Overview of the field	15
2.3.1. Contextual bandit	15
2.3.2. Hyperparameter tuning	17
2.3.3. Simulation	18
2.3.4. Domain randomization	20
2.4. Current state of the art	22
2.5. Gaps in the research	26
3. Methods	27
3.1. Simulation	27
3.2. Domain randomization	30
3.3. Hyperparameters tuning: Optuna	31
3.4. Model training	34
3.5. From simulation to reality	36
3.6. Experiments	38

4. Results	40
4.1. Hyperparameter optimization	40
4.2. Simulation results	42
4.3. Transfer of the results in real-world	47
5. Conclusions	52
5.1. Summary of the findings	52
5.2. Reflection	54
5.3. Recommendations for future research	55
Bibliography	56
A. Appendix	59
A.1. Algorithms	59
A.2. Hyperparameters	62
A.3. Training complements	64

1. Introduction

1.1. Context

Sorting has always been a critical step in the recycling process, as it allows the separation of different materials for further processing and reuse. In recent decades, a significant part of this task has been outsourced to developing countries due to the availability of cheap labor. However, this practice has raised concerns about the working conditions and environmental impact of these operations. The use of robots in recycling represents an opportunity to address these issues by relocating the sorting task to developed countries and improving the efficiency and sustainability of the process.

Robots have already shown significant potential in industrial applications, particularly in the handling and manipulation of objects. However, throwing items is a particularly challenging task for robots due to the need for precise control over the throwing trajectory and the variability of object properties. Developing a reliable and efficient throwing mechanism for robots would allow them to perform a wider range of sorting tasks and increase the overall efficiency and productivity of the recycling process.

I completed my internship at GeMMe, a lab that specialized in sensor-based sorting for industrial applications. This company uses advanced sensor technologies such as 3D imaging, hyperspectral imaging, X-ray transmission, and laser-induced breakdown spectroscopy (LIBS) to acquire data that drives robot sorting processes. Currently, three ABB FlexPickers are being used to grasp objects and place them into corresponding buckets, as determined by the sensor and classification algorithms.

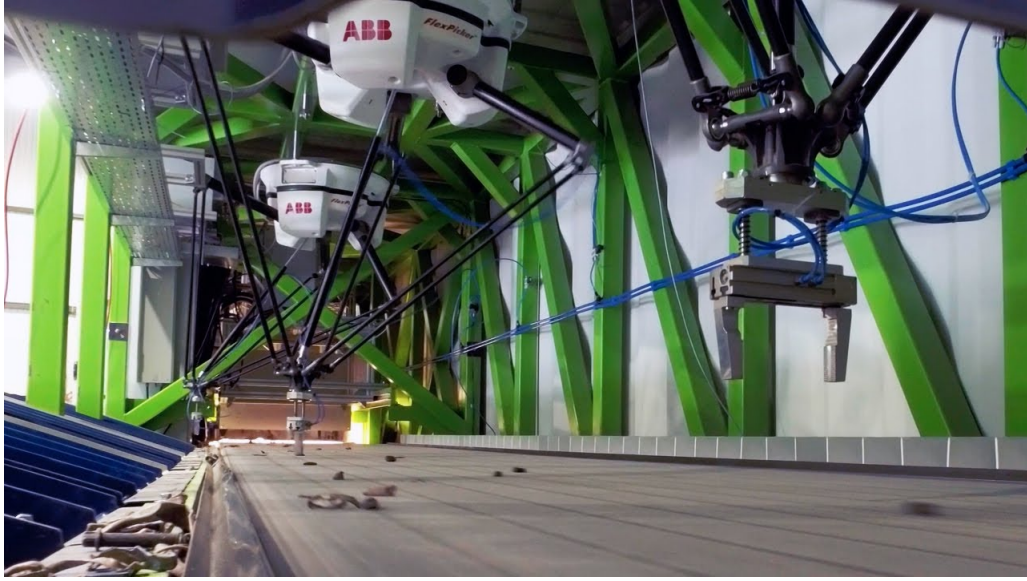


Figure 1.1.: Picture of the Pick It project taken from the website of GeMMe¹.

In the context of a previous project named PICK IT (Figure 1.1) carried out by the GeMMe lab and in collaboration with the COMET Group and CITIUS Engineering, the researchers have succeeded to sort non-ferrous scraps for the recycling industry. This project has led to the birth of MULTIPICK a wider project where sensors are used for the recognition of different metals (aluminium, copper, brass, zinc, etc.) and are sorted with robots at very high speeds (more than 1 part per second). The rate of the sorting process is a significant concern for the process to be profitable. Currently, the robots use a pick-and-place routine once the object has been classified in a category.

An ambitious idea of the lab is to develop a control policy for the robot that allows them to throw directly the object into the bucket in order to save time. A first proof of concept was developed by another engineering student, Norman Marlier, in 2019 [Marlier et al., 2019] with an ABB IRB 340 robot to throw objects to buckets with an empirical success rate of 99% in simplified conditions using reinforcement learning.

¹Image from <http://www.gemme.ulg.ac.be/> (last consulted 8th June 2023)

1.2. GeMMe

As indicated on their website², the GeMMe laboratory of the University of Liege, known for its multifaceted specialization in Mineral Processing and Recycling, Construction Materials, and Georesources-GeoImaging, served as the foundation of my internship experience. Under the aegis of Robert Baudinet, I collaborated with the data science and robotic teams to integrate innovative algorithms into the system.

In the Sensor-Based Sorting and Characterization lab, a key focus is the intersection of sorting and characterization, manifesting in the development of machine vision tools, robotic sorting systems, and pneumatic ejection systems. These real-time, high-efficiency processes are facilitated by the integration of sensors, PLCs, and grippers.

My experience was enriched through hands-on exposure to their state-of-the-art multi-sensor bench for industrial sorting, comprising a fast-moving conveyor belt, 3D scanner, an assortment of hyperspectral VNIR and SWIR cameras, multi-energy X-ray linear sensor, an analytical LIBS scanner, ABB robots, and pneumatic ejectors. The intricate fusion of these components yields comprehensive data, processed through machine learning algorithms for inference, and further utilized by a supervisory program to control the robots.

Additionally, a key aspect of my internship experience was the integration of Agile methodologies within our working processes. Regular Agile meetings fostered team communication, coordination, and collaboration. We were encouraged to iterate quickly, respond to changes, and continuously improve our workflows. Moreover, our weekly intern meetings served as an excellent platform for us to share experiences, discuss challenges, and collaboratively strategize solutions.

During my internship, I had the privilege of working alongside a highly skilled team of professionals that have greatly enriched my learning experience. Charles Baudinet, Benjamin Delvoye, and Dominik Zians, made up the formidable data science team that I collaborated with. Their expertise in various facets of data science, machine learning algorithms, and their practical application was instrumental in my development as an intern.

²<http://www.gemme.ulg.ac.be/> (last consulted 8th June 2023)

Simultaneously, I also had the opportunity to work closely with the robotics team, comprising Baptiste Dory and Robin Kloostermeyer. Their proficiency in developing robotic systems, managing industrial processes, and integrating complex components into functional units added depth to my understanding of the field.

In summary, it can be said that the GeMMe laboratory has strategically positioned itself as a leading player in the field of recycling automation. Its innovative work in sensor-based sorting and characterization is not only shaping the future of recycling practices in Belgium but also influencing the global landscape of waste management.

1.3. Research problem and questions

Following the proof of concept of Norman Marlier [Marlier et al., 2019], the objective of this master thesis is to investigate reinforcement learning-based approaches in more realistic conditions for the throws the GeMMe lab has to tackle. That means using scraps with more chaotic shapes than stones used by Norman Marlier in its proof of concept [Marlier et al., 2019]. Moreover, in the proof of concept used for throwing stones, the robot has to move in front of the bucket before throwing the stone which leads to poor results in terms of time. Indeed, it is even slower than the pick-and-place routine currently implemented. These results raise several questions: is it possible to throw the object directly from its pick position while remaining accurate in the throw? Moreover, is it possible to speed up the process to gain cycle time between the sorting of two objects?

Reinforcement learning is a promising approach for robotic throws as it enables the robot to learn how to throw objects accurately in complex and dynamic environments without the need for explicit programming.

The goal of this master thesis is to develop reinforcement learning approaches for throwing items with an ABB IRB 360 robot (Figure 1.2).



Figure 1.2.: ABB IRB 360 FlexPicker³

³Image from <https://new.abb.com/products/fr/3HAC020536-015/irb-360> (last consulted 8th June 2023).

1.4. Research goals and objectives

The goal of this master thesis is to explore the performances of reinforcement learning in a multidimensional continuous action space contextual bandit problem which consists in throwing items with an ABB IRB 360 robot. The ultimate objective of this research is to beat by a significant margin the pick-and-place routine that is currently working in the line while keeping a satisfactory success rate. To try to achieve this performance, we have decided to follow a different approach than simply running an RL algorithm that learns directly with the robot in order to boost the performance.

Indeed, for reinforcement learning in the robotic field, it is common to use a simulator. Regarding real-world robotic applications, the interactions of RL agents can be time-consuming, expensive, and potentially hazardous. The use of simulators in RL is a powerful tool for enabling agents to learn complex tasks in a safe, efficient, and cost-effective manner. It allows for faster training and safer testing. For this reason, we have decided to start my master's thesis by building a simulator of a simplified version of the environment where we can train agents.

Thereby we aim to develop for the company a whole pipeline from training agents in a simulator to fine-tuning them in the real world while using transfer learning techniques. Therefore, if the results are not plenty satisfactory and are possible to be improved, it opens the door for future works to be carried out for the GeMMe lab.

1.5. Scope and limitations of the study

The scope of this master thesis is to explore the performances of reinforcement learning in a multidimensional continuous action space contextual bandit problem using a simulator and transfer the knowledge to the real world. However, there are several limitations to this work that must be acknowledged.

The simulator used in this study was developed in the context of this master thesis, and although it is based on the real-world environment, it is a simplified schema that may not fully capture the complexity and variability of the real environment. Additionally, the transfer learning techniques employed are relatively simple, and future work should explore more advanced techniques that may yield better performance.

Furthermore, while the simulator was used to train the agents, fine-tuning the learned policies by learning in the real-world environment was not possible within the time frame of this master thesis. This step is crucial to achieving more efficient object throws and should be a major focus of future work. As a consequence, only policies learned with the simulator have been evaluated in real settings.

Despite these limitations, the results of this study can serve as a starting point for further research in robotic throws. I encourage the GeMMe group to build upon this work and continue to explore the use of reinforcement learning for solving this task.

1.6. Structure of the thesis

To conclude this **introduction**(1), this section provides an overview of the structure and content of the master thesis, outlining the key chapters and sections that will be covered.

First, the field is introduced through a **literature review** chapter (2) starting with the theoretical background needed and the formulation of the problem. Afterwards, we will pass through an overview of the field ending with the state-of-the-art solution to solve our problem and discuss the limitations and the gap in the research with respect to our kind of problem.

Secondly, the **methodology** (3) used to throw scraps with the robots will be discussed. Starting with the simulator, passing through transfer learning techniques, hyperparameter optimization, model training and ending with the integration of the solution.

The fourth chapter will report the **results** (4) of the hyperparameter optimisation, the training of the algorithms with these optimized hyperparameters and we will finish with the performances of the algorithms with the real robots.

Finally, the Master's thesis ends up with the **conclusion** (5). Further details can be found in the appendix (A).

2. Literature Review

2.1. Theoretical Background

Reinforcement Learning (RL) is a machine learning method where an intelligent agent interacts with an environment to learn a policy that maximizes the cumulative reward (Equation 2.1). This framework is often used in robotics to construct an autonomous intelligent agent to control a robot over an infinite time horizon (Figure 2.1).

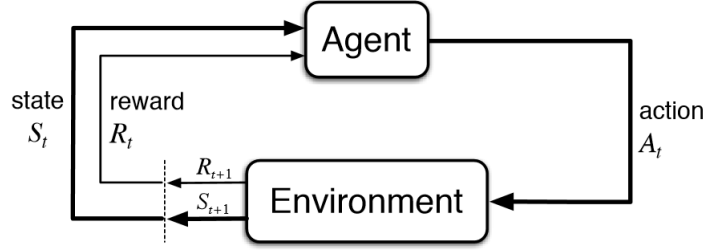
$$\underset{\pi}{\text{maximize}} \quad \mathbb{E}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim p(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \quad (2.1)$$

The objective function is defined as the expected sum of discounted rewards obtained by following the policy π in an environment with dynamics p , starting from the initial state s_0 . At each time step t , the agent selects an action a_t from the policy distribution π conditioned on the current state s_t , observes a reward $r(s_t, a_t)$ and the next state s_{t+1} according to the environment dynamics p .

The discount factor γ ($0 < \gamma < 1$) is used to weigh down future rewards relative to immediate rewards, and ensures that the sum of rewards converges to a finite value even for infinite horizons.

However, the problem simplifies in our situation as we only act at one time step for one throw. Therefore we have a time horizon of 1 and the objective function can be simplified

$$\underset{\pi}{\text{maximize}} \quad \mathbb{E}_{a \sim \pi(\cdot|s)} [r(s, a)] \quad (2.2)$$

Figure 2.1.: Agent-environment interaction in RL framework¹.

The problem becomes therefore a contextual bandit problem. Contextual bandit problems are similar to traditional RL problems but differ in that the agent only receives feedback about the reward for the action it selects in the current state, rather than in all possible states (Figure 2.2). This makes the learning problem more tractable in some cases but also limits the agent’s ability to learn about the environment. Both problems, RL and CB, involve selecting an action based on the current state or context (in CB we speak rather about context than state) and receiving a reward based on that action. In the RL framework, the goal is typically to maximize the cumulative reward over a long time horizon, whereas, in the contextual bandit framework, the goal is to maximize the immediate reward at each time step. However, when the time horizon is 1, the distinction between long-term and immediate rewards disappears, and the problems become equivalent. The benefits of contextual bandit algorithms include statistical qualities such as regret guarantees that provide a way to balance exploration and exploitation. However, it may still be relevant to examine the performance of RL algorithms as they are flexible and can adapt to CB settings. Moreover, recent works [Duckworth et al., 2023] show that RL algorithms can outperform handcrafted contextual bandit algorithms for continuous actions.

¹Image from <https://www.researchgate.net/figure/Agent-environment-interaction-in-RL-framework> (last consulted 8th June 2023).

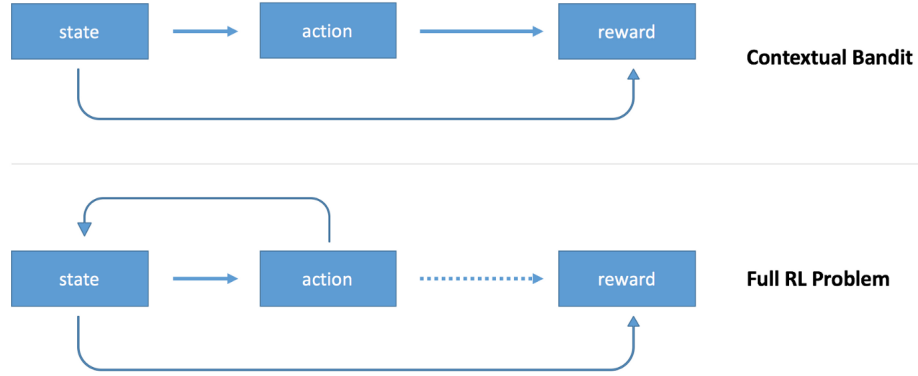


Figure 2.2.: Reinforcement learning compared to contextual bandit².

Consequently, the problem is reformulated as finding from a given position of an object and a bucket position, plus some information about the nature of the object such as its mass, for example, a place to throw the object with a direction and a given speed. However, the speed, the direction and the release position are insufficient due to physical and robotic constraints. Indeed, the ABB robot uses a trapezoidal speed curve during its movements, therefore, with only one movement instruction, the robot will stop at the end of the movement, and with the delay of the gripper to open the object will just fall vertically. Consequently, to throw an object at a given location with a given speed, we have to provide a release position where we open the gripper, a final position for the movement and the speed will be the maximal speed during the movement if it is reachable. As many positions in the space could be irrelevant, we only focus on the positions in the plane between the post-pick position and the bucket position and perpendicular to the conveyor in order to reduce the dimension of the problem. Therefore, the direction of the gripper will always be aligned with the movement and we avoid this additional parameter.

In summary, the process of throwing begins with the robot picking up the object from the conveyor belt (Figure 2.3a), followed by its ascent to the post-pick position (Figure 2.3b). The robot then transitions in a straight line to its release position, aligning the gripper with its trajectory (Figure 2.3c). Upon reaching the release position, the gripper begins to open as the robot continues moving towards its final position (Figure 2.3d). Once it reaches this position, the robot can then proceed to

²Image from <https://towardsdatascience.com/contextual-bandits-and-reinforcement-learning-6bdfeace72a> (last consulted the 8th June 2023).

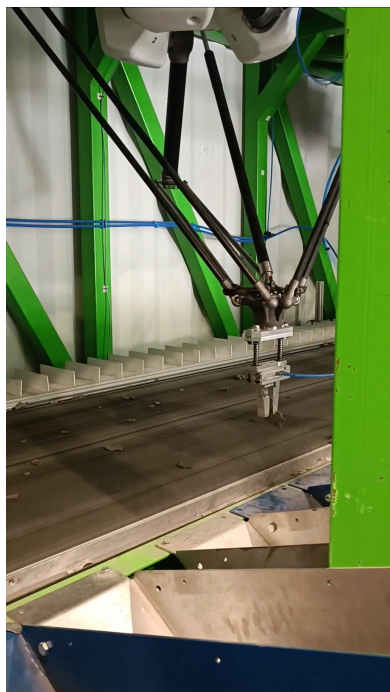
pick up another piece of scrap.



(a) The robot picks the scrap.



(b) The robot goes up to post-pick position.



(c) The gripper starts to open at the release position.



(d) The robot reaches its final position.

2.2. Problem formulation

Formally, to define properly the contextual bandit problem we have to define a context space (set of possible situations the agent can encounter), an action space (set of possible actions the agent can use) and a reward function which represent the feedback signal that the agent receives after taking an action in a given context.

Context space

Context Space \mathcal{C} :

$$\mathcal{C} = \{((x_o, y_o, z_o), (x_b, y_b), m) \in \mathbb{R}^6$$

where (x_o, y_o, z_o) is the position of the object, (x_c, y_c) is the position of the bucket and m is the weight of the object

Action space

Action space \mathcal{A} :

$$\mathcal{A} = \{((x_r, z_r), v, x_t) \in \mathbb{R}^4$$

where (x_r, z_r) is the position where we start to open the gripper, y_r is determined by the constraint that the point is in the plane between the object at the post-pick position and the bucket position perpendicular to the conveyor, v is the maximal speed during the movement if it is reachable, x_t determines the target position for the robot. Note that we only need x_t as the target position for the robot is on the line between the post-pick position and the release position. Only linear movements for the robot are considered. While the ABB FlexPicker is capable of executing more complex movements, we decided to begin with a simpler problem formulation. Our approach was to gradually increase the complexity of the problem only if initial results were promising. This strategy was preferred over starting with a complex problem and potentially finding ourselves with no progress mid-internship, thereby being forced to simplify the problem.

Reward function

The reward function is really crucial for the problem as it will guide the agent toward its final policy. In this problem, we aim to train the robot to throw objects with high accuracy and consistency while being fast. Therefore we need a fair trade-off between being fast and accurate. The two quantities we will collect are the success of a throw and the time between the robot starting its movement and reaching its target position. It is possible to weigh these two with a parameter however setting the value of the parameter is challenging and this score will not have much physical meaning. Moreover, the success of a toss where the object is far from the bucket should be more rewarded than one close to the bucket. That is the reason why the current pick-and-place policy was chosen as a baseline.

The idea is to reward the success of a toss based on the time the pick-and-place policy takes to go to the bucket. This is, therefore, a measure of the difficulty of a toss and additionally, it is expressed in seconds which is interesting in order to combine with the travel time of the robot. Moreover, as we want to be able to be faster than the pick-and-place routine the score will allow us to directly observe if the agent performs better or not than the baseline.

However, the time the pick-and-place policy would take for a given context is not straightforward and directly available. As a consequence, the idea is to build an estimator of the baseline that will be used at training time. In our case, this estimator will be a neural network.

Reward function \mathcal{R} :

$$\mathcal{R} = \begin{cases} \text{PickAndPlace}(c) - t & \text{if the toss is successful} \\ -t & \text{if the toss is missed} \end{cases} \quad (2.3)$$

where $\text{PickAndPlace}(c)$ is the time the pick-and-place policy takes for a context c and t is the time the robot takes from the post-pick position to the target position. The reward is expressed in seconds and if the reward is positive, that means that we achieve a better performance than the baseline.

2.3. Overview of the field

2.3.1. Contextual bandit

In this work, we consider a contextual bandit problem with continuous and multi-dimensional action space. Contextual bandit is a subfield of machine learning that involves making decisions based on contextual information under uncertainty, while simultaneously learning from the outcomes of those decisions. They differ from classical RL problems as the agent does not interact with the environment over a series of episodes, with the goal of maximizing the expected sum of future rewards. In contrast, contextual bandits problems typically involve a single interaction between the agent and the environment per episode, the interactions are independent and identically distributed. In addition, the agent must make a decision based on the available context, receives a reward or penalty for that decision, and then moves on to the next interaction with a potentially different context, there is no transition function (Figure 2.2).

As a consequence, CB algorithms focus more on the exploration-exploitation trade-off and are often more sample efficient. A common measure in contextual bandit problems is regret which is the difference between the reward obtained at iteration t by selecting the best action possible and the reward obtained by using the current policy. Whereas, RL algorithms focus more on finding the best policy that will return the best expected reward from interacting with the environment.

$$regret(T) = \sum_{t=1}^T r_t(c_t, a_t^*) - r_t(c_t, \pi(c_t)) \quad (2.4)$$

where a_t^* is the optimal action at iteration t in the context c_t , π is the policy of the agent and T is the number of interactions.

It is worth noting that in large continuous action space, the optimal action for a given context is often unknown and we use an estimate of this one.

Contextual bandit problems with small and distinct action spaces, also known as

multi-armed bandits with side information, have established techniques [Li et al., 2010] [Garivier and Moulines, 2011]. The trade-off between exploration and exploitation in this scenario is well-studied, and there are formal bounds on regret. However, little research has been conducted on continuous action spaces furthermore discretising continuous action space to fall back in the discrete framework is often a bad idea due to the curse of dimensionality: the number of actions increases exponentially with the number of degrees of freedom [Lillicrap et al., 2019]. Recent studies have focused on extreme classification and have used tree-based methods to select actions from a discretised action space with smoothing [Majzoubi et al., 2020]. However, these tree methods only work for unidimensional actions.

Moreover, recent works in the field have shown that RL algorithms can outperform CB algorithms when they are trained enough [Duckworth et al., 2023]. As in our situation, we freely generate data from a simulator, it makes sense to use state-of-the-art methods in RL to tackle a contextual bandit problem with continuous and multidimensional action space. In addition, recent RL algorithms have incorporated advanced techniques to avoid exploiting too much their current policy at the expense of exploration. Proximal Policy Optimization (PPO) [Schulman et al., 2017] and Soft-Actor critic (SAC) [Haarnoja et al., 2018] algorithms for instance introduce an entropy term in the loss function to encourage exploration. Furthermore, ensuring decision-making performance in vast, continuous action spaces has been difficult to achieve, resulting in a substantial disparity between theoretical approaches and practical implementation even if recent work try to make it practical [Zhu et al., 2022]. Therefore in this work, we have decided to discard contextual bandits algorithms such as Gaussian processes [Krause and Ong, 2011] and only use algorithms that are known to perform well on high-dimensional action space such as SAC, PPO and TD3.

Contextual bandits and RL problems raise several challenges compared to supervised machine learning. Indeed, in classical supervised learning the ideal action to associate with a particular context is learned from ground truth labels whereas in the RL settings, we only have access to the reward produced by the agent’s interaction with the environment. As a consequence, the data generated are biased with the policy of the agent and might lead to local minima when trying to optimize the policy

[Sutton et al., 1999]. Modern RL algorithms have therefore implemented techniques to avoid this problem such as gradient clipping used in PPO [Schulman et al., 2017] in order to limit the update of the policy at each gradient descent optimization step.

2.3.2. Hyperparameter tuning

Hyperparameter optimization is a critical and challenging task in the field of Reinforcement Learning (RL), as it directly affects the performance of RL models. The goal of hyperparameter optimization is to find the best set of hyperparameters that can optimize the performance of a model on a specific task. In the past, researchers mainly used brute-force methods to optimize hyperparameters, such as grid search, which is a time-consuming and computationally expensive method.

Grid search is one of the earliest and most straightforward hyperparameter optimization techniques. It involves specifying a grid of hyperparameters and evaluating the model’s performance on each combination of hyperparameters. Although grid search is simple and intuitive, it suffers from the curse of dimensionality. As the number of hyperparameters increases, the search space grows exponentially, making the grid search infeasible for complex models.

To overcome the limitations of grid search, researchers have proposed various approaches to hyperparameter optimization, such as random search. Random search is a simple yet effective approach that randomly samples hyperparameters from a pre-defined range. Unlike grid search, which exhaustively searches the entire space, random search focuses on a smaller subset of the search space. This makes it a more efficient approach for hyperparameter optimization in a high-dimensional space [Bergstra et al., 2011]. However, [Bergstra and Bengio, 2012] shows that random search is unreliable for training some complex models.

In recent years, Bayesian optimization has emerged as a popular approach to hyperparameter optimization. Bayesian optimization employs probabilistic models to construct a surrogate function that approximates the performance of the model as a function of hyperparameters. The surrogate function is iteratively optimized to find the optimal hyperparameters. One of the most popular libraries for Bayesian optimization is Optuna [Akiba et al., 2019].

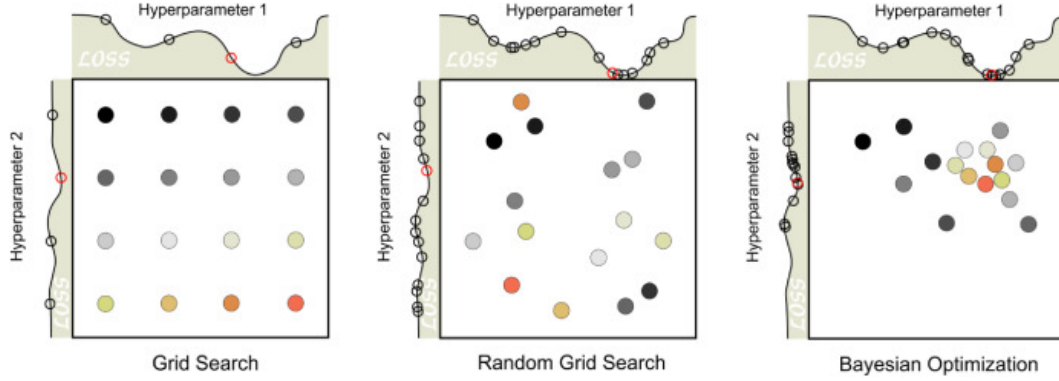


Figure 2.4.: Overview of hyperparameter optimization techniques reprinted from [Passos and Mishra, 2022].

Optuna is an open-source library for hyperparameter optimization that implements several state-of-the-art algorithms, such as Tree-structured Parzen Estimator (TPE) and Successive Halving (SH). Optuna is built on top of Python and is easy to use, making it a popular choice for many researchers in the field of RL.

In conclusion, hyperparameter optimization is a crucial task in RL, and various approaches have been proposed over the years to address it. Grid search, random search, and Optuna are some of the most commonly used methods in the field. While grid search and random search are simple and intuitive, they can be inefficient for complex models. Optuna, on the other hand, is a powerful library that provides efficient and effective methods for hyperparameter optimization which is the reason why it was selected to tune the hyperparameters of the different models in this thesis.

2.3.3. Simulation

In the robotic field when it comes to reinforcement learning, it is common to use a simulator. Indeed, when it comes to real-world robotic applications, the interactions of RL agents can be time-consuming, expensive, and potentially hazardous. The use of simulators in RL is a powerful tool for enabling agents to learn complex tasks in a safe, efficient, and cost-effective manner. It allows for faster training and safer testing. For this reason, we have built a simulator of a simplified version of the environment where we can train agents. It also encourages the usage of RL

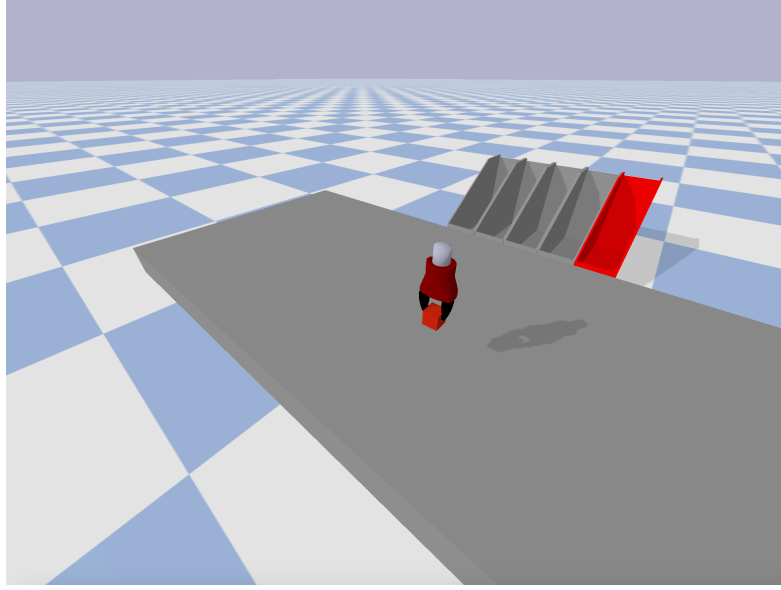


Figure 2.5.: Overview of a PyBullet environment.

algorithms at the expense of CB algorithms as with the simulator we are able to train on thousands of episodes and dismiss the sample efficiency problem of RL algorithms.

The selection of a suitable physics simulator is crucial in achieving high efficiency and accuracy in robotic applications. After reviewing the paper [Collins et al., 2021] to acquire a broad view of the field, PyBullet appeared as the ideal choice for simulating scrap throws.

PyBullet[Coumans and Bai, 2021] is an open-source 3D physics simulator that provides accurate and efficient simulations of physical interactions. It has gained popularity among researchers and practitioners in the field of robotics due to its robustness, versatility, and high-performance computing capabilities. PyBullet offers a wide range of features that make it an excellent choice for simulating throwing objects at high speeds.

Firstly, PyBullet is equipped with a Bullet Physics Engine that can simulate rigid bodies, soft bodies, and deformable bodies. This engine is optimized for multi-core processors and GPUs, making it capable of handling complex physics simulations with high accuracy and speed. Therefore, PyBullet can simulate the physical properties of objects such as weight, size, and shape accurately, allowing for efficient

throw simulations.

Secondly, PyBullet provides a python API that makes it easy to integrate with other libraries such as Gym, a toolkit for developing and comparing reinforcement learning algorithms, and PyTorch. This feature allows researchers to use PyBullet as a building block for developing custom gym environments for reinforcement learning. The flexibility of the API also makes it easy to customize the environment to suit specific needs, such as adjusting the velocity and trajectory of the robot.

Finally, PyBullet is actively maintained by a large community of developers and researchers, ensuring that it is up-to-date with the latest advancements in physics simulations. This means that any issues or bugs are quickly resolved, and new features and improvements are regularly added.

2.3.4. Domain randomization

A drawback of using a simulator to train robotic policies is the reality gap as mentioned in [Ibarz et al., 2021]. Indeed the simulated environment does not represent exactly the reality and the policies learned in such environments can lead to poor results in real conditions. This step of transferring policies learned in simulation to real-world settings is called the sim-to-real transfer. Several methods exist to bridge this reality gap and achieve better sim-to-real transfer [Weng, 2019]. Such methods can be a better simulation but also domain randomization and domain adaptation (Figure 2.6). The purpose of domain randomization is to provide enough simulated variability at training time such that at test time the model is able to generalize to real-world data. While domain adaptation refers to a group of techniques used in transfer learning to adjust the data distribution in a simulated environment so that it more closely matches the real environment. This is achieved by applying a mapping or regularization method to the task model. In image-based reinforcement learning tasks, such techniques often involve adversarial loss or the use of generative adversarial networks (GANs). The goal of domain adaptation is to improve the agent’s ability to perform well in the real environment, even if it was trained on a different simulated environment.

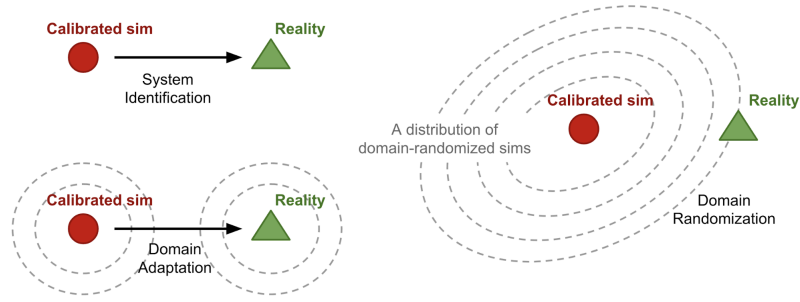


Figure 2.6.: Sim2Real approaches reprinted from [Weng, 2019]

In this work, we use deep RL algorithms to learn the policies in simulation. As it was shown that domain randomization improves the performance of deep neural networks when transferring to reality [Tobin et al., 2017], we use domain randomization with various dynamics parameters of the robot and the environment. The randomized parameters can be found in the next chapter.

2.4. Current state of the art

RL algorithms have made significant progress in recent years, allowing them to handle continuous and high-dimensional action spaces. More precisely, deep reinforcement learning (RL) algorithms have emerged as a powerful tool for solving complex decision-making problems in a wide range of domains, from robotics over game playing to finance. The emergence of deep RL can be attributed to several factors, including the availability of large amounts of data, the development of powerful computing resources, and the advances in deep learning techniques. Deep RL algorithms combine the principles of RL, which involves learning from trial and error, with deep neural networks, which are capable of learning complex representations of data. This combination allows deep RL algorithms to learn directly from raw sensory inputs, such as images or sounds, and to make decisions based on high-dimensional state spaces. Among the state-of-the-art algorithms, Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Twin Delayed Deep Deterministic Policy Gradients (TD3) have shown impressive results in solving RL problems with continuous action spaces.

PPO

PPO [Schulman et al., 2017] is an off-policy algorithm that uses a clipped objective function to ensure that the policy updates are not too large, improving stability. PPO also uses a value function as a baseline to reduce the variance of the gradient estimates. The proximal policy optimization algorithm updates its policy as follows:

$$\theta_{k+1} = \arg \max_{\theta} \mathbb{E}_{s, a \sim \pi_{\theta_k}} [L_{PPO}(s, a, \theta_k, \theta)] \quad (2.5)$$

The goal is to maximize the objective given by:

$$L_{PPO}(s, a, \theta_k, \theta) = \min \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)} A^{\pi_{\theta_k}}(s, a), \quad \text{clip} \left(\frac{\pi_{\theta}(a | s)}{\pi_{\theta_k}(a | s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right) \quad (2.6)$$

where ϵ is a hyperparameter limiting the update of the policy and A is the advantage function being positive if the action a needs to be chosen more frequently in the given

state s by the policy π and vice-versa.

SAC

SAC is another off-policy algorithm that uses an entropy regularizer to encourage exploration and avoid premature convergence to suboptimal policies. The entropy of a policy π parametrized with parameters θ is defined as:

$$\mathcal{H}(\pi_\theta(\cdot|s)) = \mathbb{E}_{a \sim \pi_\theta(\cdot|s)} [-\log \pi_\theta(a|s)] \quad (2.7)$$

In entropy-regularized reinforcement learning, the agent gets a bonus reward at each time step proportional to the entropy of the policy. This changes the reinforcement learning problem, with the objective function now including the entropy bonus. The new objective function is:

$$\pi^* = \arg \max_{\pi} E_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t)) \right) \right] \quad (2.8)$$

where $\alpha > 0$ is the trade-off coefficient. The value functions are also slightly different in this setting. V^π now includes the entropy bonuses from every timestep, while Q^π includes the entropy bonuses from every timestep except the first. The connection between V^π and Q^π is given by:

$$V^\pi(s) = E_{a \sim \pi} [Q^\pi(s, a) + \alpha H(\pi(\cdot|s))] \quad (2.9)$$

and the Bellman equation for Q^π is:

$$Q^\pi(s, a) = E_{s' \sim P} E_{a' \sim \pi} [R(s, a, s') + \gamma (Q^\pi(s', a') + \alpha H(\pi(\cdot|s')))] = E_{s' \sim P} [R(s, a, s') + \gamma V^\pi(s')] \quad (2.10)$$

SAC's objective function in one step RL is therefore defined as:

$$V^\pi(s) = \mathbb{E}_{a \sim \pi} [r(a) + \alpha \mathcal{H}(\pi(\cdot|s))] \quad (2.11)$$

where \mathcal{H} is the entropy function, and α is a hyperparameter that controls the trade-

off between exploration and exploitation.

TD3

Twin Delayed Deep Deterministic Policy Gradient (TD3)[Fujimoto et al., 2018] is an off-policy, model-free reinforcement learning algorithm designed for continuous control tasks. TD3 builds upon the Deep Deterministic Policy Gradient (DDPG) algorithm [Silver et al., 2014] by addressing its overestimation bias and instability issues. TD3 introduces several improvements, such as using twin critics, delayed policy updates, and target policy smoothing, which lead to better performance and stability.

TD3 trains two Q-functions, Q_{ϕ_1} and Q_{ϕ_2} , concurrently using mean square Bellman error minimization. This is similar to DDPG but with some differences. One difference is target policy smoothing, which adds clipped noise to actions used to form the Q-learning target based on the target policy. The target actions are clipped to lie in the valid action range. This serves as a regularizer for the algorithm and helps avoid incorrect sharp peaks in the Q-function. The target actions smoothing equation is:

$$a'(s') = \text{clip}(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}}), \quad \epsilon \sim \mathcal{N}(0, \sigma) \quad (2.12)$$

Another difference is clipped double-Q learning, which uses the smaller Q-value for the target and helps fend off overestimation in the Q-function. Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a smaller target value. The clipped double-Q learning equation is:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')) \quad (2.13)$$

and both Q-functions are learned by regressing to this target. The Q-function loss equations are:

$$L(\phi_1, \mathcal{D}) = E_{(s,a,r,s',d) \sim \mathcal{D}} \left(Q_{\phi_1}(s, a) - y(r, s', d) \right)^2 \quad (2.14)$$

$$L(\phi_2, \mathcal{D}) = E_{(s,a,r,s',d) \sim \mathcal{D}} \left(Q_{\phi_2}(s, a) - y(r, s', d) \right)^2 \quad (2.15)$$

The policy is learned by maximizing Q_{ϕ_1} , but updated less frequently than the Q-functions to dampen volatility. The policy update equation is:

$$\max_{\theta} E_{s \sim \mathcal{D}} [Q_{\phi_1}(s, \mu_{\theta}(s))] \quad (2.16)$$

which is similar to DDPG. To facilitate exploration, noise is added to actions at training time, typically uncorrelated mean-zero Gaussian noise. To get higher-quality training data, you may reduce the scale of the noise over the course of training. However, noise is not added to actions at test time.

In our settings, the Q-functions only depend on the current context or state and the action in this context which simplifies the problem.

Implementation details

The pseudo-code for each algorithm can be found in the appendix A and the implementation of each algorithm can be found on the GitHub page of Stable Baseline 3³.

³<https://github.com/DLR-RM/stable-baselines3>

2.5. Gaps in the research

Despite the recent progress, there are still several gaps in the research. One of the main gaps is the lack of theoretical analysis and formal bounds on the performance of RL algorithms in CB problems with continuous action spaces. Another gap is the need for more empirical studies to compare the performance of different RL algorithms and CB algorithms in handling continuous action spaces. Finally, there is a need for more research on the scalability and efficiency of RL algorithms in handling large-scale CB problems with continuous action spaces.

3. Methods

3.1. Simulation

In order to learn efficient control policies, we developed a PyBullet simulator of the sorting line. The simulator is based on the OpenAI Gym API [Brockman et al., 2016]. The OpenAI Gym API has become a de facto standard in the reinforcement learning community. The OpenAI Gym’s simple and uniform interface allows for testing diverse reinforcement learning algorithms in a standardized manner, enabling easy comparison and benchmarking. Crucially, PyBullet’s compatibility with the Gym API allows us to leverage the powerful algorithms implemented in Stable Baselines 3, a set of high-quality implementations of reinforcement learning algorithms. These state-of-the-art algorithms enable more efficient training, better performance, and a faster path to results. Thereby it is used to enhance the overall quality of this work.

Running an episode with the OpenAI Gym API generally follows a pattern of initializing the environment, iterating over a set number of steps, taking an action at each step, observing the result, and finally recording the outcome of the episode. However, in our settings, we only have one action to take therefore an episode is decomposed as follows:

Initialization

The environment is made of 4 principal components all in *urdf* format (Figure 3.1). A conveyor, 5 buckets, the object to throw and the gripper. At initialization, a bucket is randomly chosen as the target bucket and an object spawns in a random location on the conveyor. In order to handle the grasping of the objects easily, we only use cubes as objects but the center of mass and the mass is randomized. The

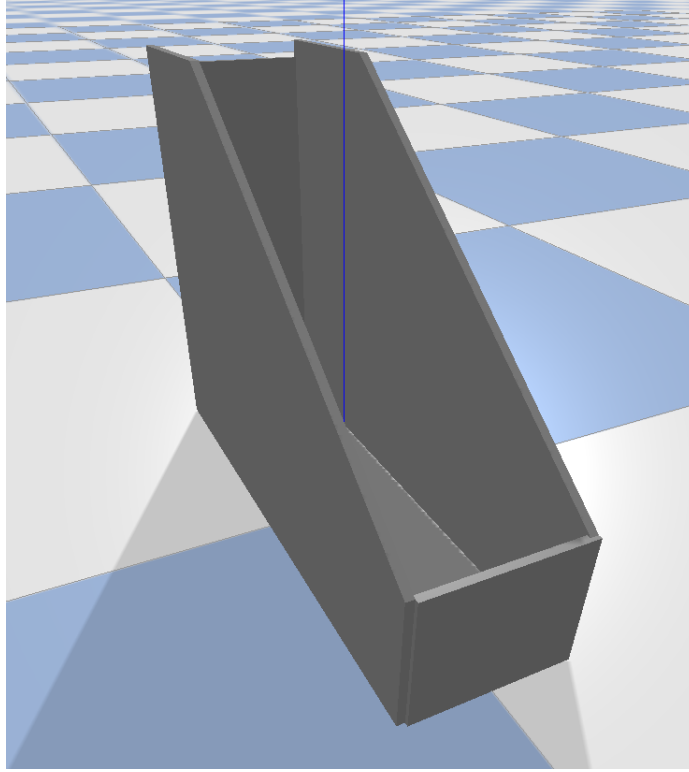


Figure 3.1.: Illustration of a bucket in URDF format in PyBullet.

pick of the cube can be therefore hard coded. Once the cube is picked, the episode starts with an initial observation (context) of the environment. The context contains the object position after the pick, the target position which is the position of the bucket, and the mass of the object.

Select Action

The agent selects an action based on the context. The action is composed of the release position, the target position and the maximal speed of the gripper. The implementation of the SAC, PPO and TD3 algorithms of Stable Baselines 3 [Raffin et al., 2021] are used as agents.

Apply Action

The selected action is applied to the environment using the “step()” function. This returns the new observation, the reward for the action, a boolean indicating if the

episode has finished (which is always the case in our setting), and additional info such as the travel time and the success of the throw:

```
observation, reward, done, info = env.step(action)
```

The dynamic of the system works through the PyBullet API to control the robot and the displacement is computed through an external library¹ provided by Tom Ewbank, researcher for the IntegrIA team. Indeed the velocity of the robot follows a trapezoidal curve governed by physical constraints such as its acceleration. Therefore, thanks to this library, a function computes the forces that are applied to the robot to match reality. The technical information about the robot was provided by the ABB documentation.

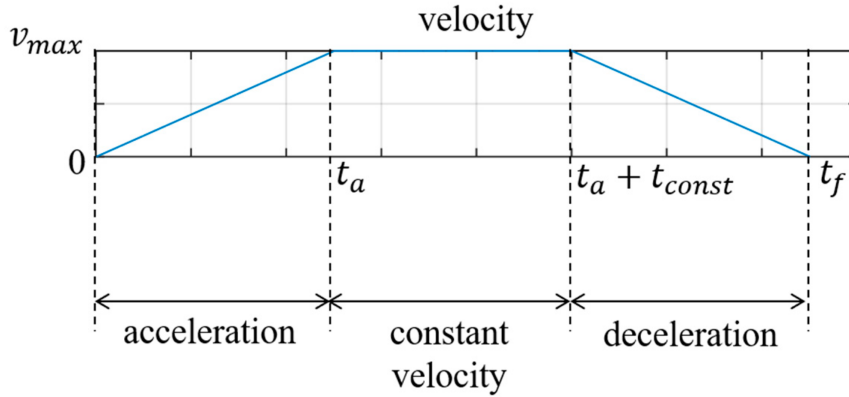


Figure 3.2.: Velocity curve of the robot (Image from [Yoon et al., 2019]).

The step ends when the object hits the ground or falls into the bucket.

Record Outcome:

Record the outcome of the step, which includes the reward, the success of a throw and the travel time. This information is stored for later analysis or for training the agent.

¹https://gitlab.uliege.be/integria/orocos_kinematics_dynamics/

3.2. Domain randomization

As mentioned in the literature review section 2, domain randomization can be used to improve the sim-to-real transfer. As a consequence, the following environment parameters have been randomized:

1. **Position of the buckets:** At each episode, the x position of the conveyor and the buckets is randomized. In a similar manner, the position of the buckets among the y-axis is randomized along the conveyor.

$$x \sim \mathcal{U}(0.6, 1.1), \quad y \sim \mathcal{U}(-1, 1)$$

2. **Opening delay of the gripper:** The gripper in reality has a delay when we want to open it. The gripper used on the ABB Flexpicker is a Festo HGPL-14-40-A-B. According to the technical documentation, the opening delay is 171ms. Therefore, the opening delay is randomized around this value:

$$d \sim \mathcal{U}(150, 190),$$

where d is the delay.

3. **Object:** The size and the mass of the object which is always a cube is randomized. The mass m is uniformly distributed between 0.01 and 2 kg. The side length a of the cube is uniformly distributed between 3 and 6 cm. Moreover, for the center of mass c , since it's a two-dimensional variable it can be shifted vertically and horizontally. The shift follows a uniform distribution in a square region of $\pm \frac{a}{4}$. We can represent this as two separate variables, c_x and c_y , which represent the horizontal and vertical displacement of the center of mass respectively

$$m \sim \mathcal{U}(0.01, 1), \quad a \sim \mathcal{U}(3, 6),$$

$$c_x \sim \mathcal{U}\left(-\frac{a}{4}, \frac{a}{4}\right), \quad c_y \sim \mathcal{U}\left(-\frac{a}{4}, \frac{a}{4}\right).$$

3.3. Hyperparameters tuning: Optuna

Optimizing the hyperparameters of RL models is a crucial step in order to avoid local minima and instability in the results. It tends to improve the results. Even if the default parameters of the stable baseline 3 models are already tuned, we have decided to tune the hyperparameters of the different models for this specific task.

This practice is really common in reinforcement learning. RL Zoo, a Python library for training and optimizing models with Gym environments, [Raffin, 2020] also used Optuna to optimize hyperparameters on a given set of environments. Therefore, we implemented a simplified version to tune the hyperparameters of the SAC, PPO and TD3 algorithms.

Optuna uses two main principal components: **an optimizer and a pruner**. First, given a search space of hyperparameters, some sets are first randomly sampled. Each set is used to train the agent on a given budget of episodes which is called a trial. At the end of the training, the score is computed in an evaluation environment with a predefined number of episodes. Afterwards, the optimizer determines the next set to explore given the previous scores obtained by the previous set of hyperparameters.

In this study, a Tree-structured Parzen Estimator (TPE) is used as an optimiser. TPE is a sequential model-based optimization (SMBO) algorithm. It utilizes Bayes' rule to update the probability model based on the observed data. It seeks to find the optimal hyperparameters x to maximize the objective function y . To suggest a new set of hyperparameters, TPE defines two distributions, $l(x)$ and $g(x)$, based on previous trial results. $l(x)$ represents a Gaussian Mixture Model associated with the hyperparameters that provided good results, where y (the objective value) is greater than y^* (the best observed value), and $g(x)$ is for the hyperparameters where y is less than or equal to y^* [Bergstra et al., 2011].

The median pruner is used as the second component for the optimization. Median pruning is a strategy where if the best intermediate result of a trial is below the median of intermediate results of previous trials at the same step, that trial is pruned. This is based on the assumption that if a trial's performance is poor in the early stages, it will likely remain poor until the end.

The parameters considered for each algorithm are detailed in the respective tables

below. For the SAC and PPO algorithms, we use the generalized state-dependant-exploration (gSDE) [Tobin et al., 2017] which facilitates the exploration in continuous action space. It is not used for TD3 as it was not implemented in the stable baseline 3 library.

TD3 Parameter	Search Space
Learning Rate γ	[1e-5, 1e-2]
Batch Size	{16, 32, 64, 100, 128, 256, 512}
Target smoothing coefficient τ	{0.001, 0.005, 0.01, 0.02}
Training frequency	{1, 4, 8, 16, 32, 64}
Noise type	{ornstein-uhlenbeck, normal, None}
Noise standard deviation	[0,1]
Number of hidden units per layer	{[256, 256], [400, 300]}

Table 3.1.: Search Space for TD3 Parameters

SAC Parameter	Search Space
Learning Rate γ	[1e-5, 1e-2]
Batch Size	{16, 32, 64, 100, 128, 256, 512}
Target smoothing coefficient τ	{0.001, 0.005, 0.01, 0.02}
Training frequency	{1, 4, 8, 16, 32, 64}
Learning start	{0, 100, 500, 1000}
Initial log σ	[-4, 1]
SDE sample frequency	{-1, 8, 16, 32, 64}
Number of hidden units per layer	{[256, 256], [400, 300]}

Table 3.2.: Search Space for SAC Parameters

PPO Parameter	Search Space
Learning Rate γ	[1e-5, 1e-2]
Batch Size	{16, 32, 64, 100, 128, 256, 512}
Entropy coefficient	[1e-9, 0.05]
Clip range	{0.1, 0.2, 0.3, 0.4}
Number of steps per rollout	{8, 16, 32, 64, 128, 256, 512, 1024, 2048}
Number of epochs	{1, 5, 10, 20}
GAE coefficient λ	{1, 5, 10, 20}
Max gradient norm	{0.3, 0.5, 0.6, 0.7, 0.8}
Value function coefficient	[0.25, 0.75]
Initial log σ	[-4, 1]
SDE sample frequency	{End of the rollout, 8, 16, 32, 64}
Number of hidden units per layer	{[256, 256], [400, 300]}
Activation function	[Relu, Tanh]

Table 3.3.: Search Space for PPO Parameters

The budget of trials is set to 100 for each algorithm. The model ceased random sampling after the initial 5 startup trials and performed 2 evaluations during training. The training process was budgeted for 40,000 episodes. The hyperparameters that were not tuned during the study get the default value of the Stable Baseline 3 implementation².

²<https://github.com/DLR-RM/stable-baselines3>

3.4. Model training

The Stable Baseline 3 library includes a simple way to train agents. Once the environment and the agent with the desired hyperparameters are initialized it is straightforward to train the agent thanks to the high-level API of SB3. Moreover, as training machine learning models is often computationally heavy, we use the GPU cluster of the University of Liege to perform the training.

In order to ensure effective and insightful supervision during the training process, we employ WeightsAndBiases. This framework plays a big role by collecting and organizing a wealth of valuable information regarding the training dynamics. By continuously monitoring key performance indicators, WeightsAndBiases gives us with a comprehensive understanding of the learning progress, enabling us to make informed decisions and optimize the training procedure. WeightsAndBiases tracks the reward to measure the training’s efficacy, while monitoring the success rate provides insights into the model’s proficiency and ability to adapt, enhancing the overall training process.

To train the agent, we first need to have the reward function. As mentioned in equation 2.3, the reward function is made of an estimator which is a neural network in this case. The reward function encapsulates the tradeoff between the success of a throw and the time. The neural network in the reward function aims to estimate the time of the pick-and-place routine. The more time the pick-and-place reward takes the more a successful throw is supposed to be valuable.

The neural network of the reward function is a PyTorch implementation of a multilayer perception (MLP). It takes the context as input and outputs the estimated time of the pick-and-place (PaP) routine in simulation. The MLP is made of two hidden layers with 100 neurons each and can be visualize in figure 3.3. It is important to note that even if the PaP routine represents the current policy used in reality, it still differs. Moreover, the peak speed of the PaP in simulation is fixed at 3m/s for stability reasons and keeping and success of 100% which is slower than the 10m/s in reality. The reward function was trained over 300,000 episodes on GPU and more information can be found in the appendix A.

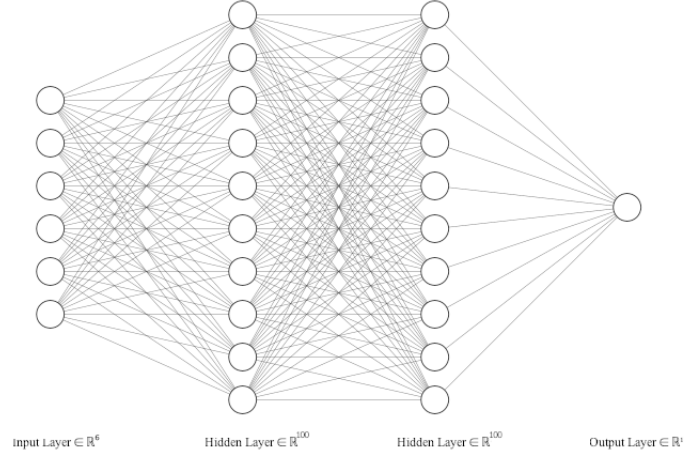


Figure 3.3.: Neural network architecture used for the reward function in order to infer the time taken by the pick-and-place baseline.

Once the reward function was trained, we trained 3 different algorithms (TD3, PPO and SAC) with 2 sets of hyperparameters. On one hand, we used the default parameters of the Stable Baselines 3 algorithms using when available the generalized state-dependant-exploration as we are in a continuous action space setting. On the other hand, we used the algorithms with the tuned hyperparameters obtained by the Optuna study described before. It also allows to visualize if the tuned hyperparameters bring superior performances even if the training is stochastic.

3.5. From simulation to reality

Once the agents were trained, we needed a way to integrate the models into the sorting line. First, the information gathered by the robots might not be in the right format: units, offsets in the coordinates compared to simulation... Therefore, we first implemented a preprocessing module that allows us to convert the information for the context gathered by the robot to understandable data for the RL model. Afterwards, a postprocessing module converts the action of the model to an executable action by the robot.

Currently, the model is running on a remote PC that communicates with the line's computer (Pick it computer). The connection is a TCP/IP connection where the Pick It computer acts as the client and the PC with the model as the server. The link is established through the classical 3-way handshake by the client and the PC runs the model when it receives a context, sends back the corresponding action and waits until the next object. The process is shown in Figure 3.4.

Moreover, two new scripts in RAPID, the programming language of the ABB robots, were implemented in order to handle the TCP/IP connection and another one to control the robot and execute the action computed by the model.

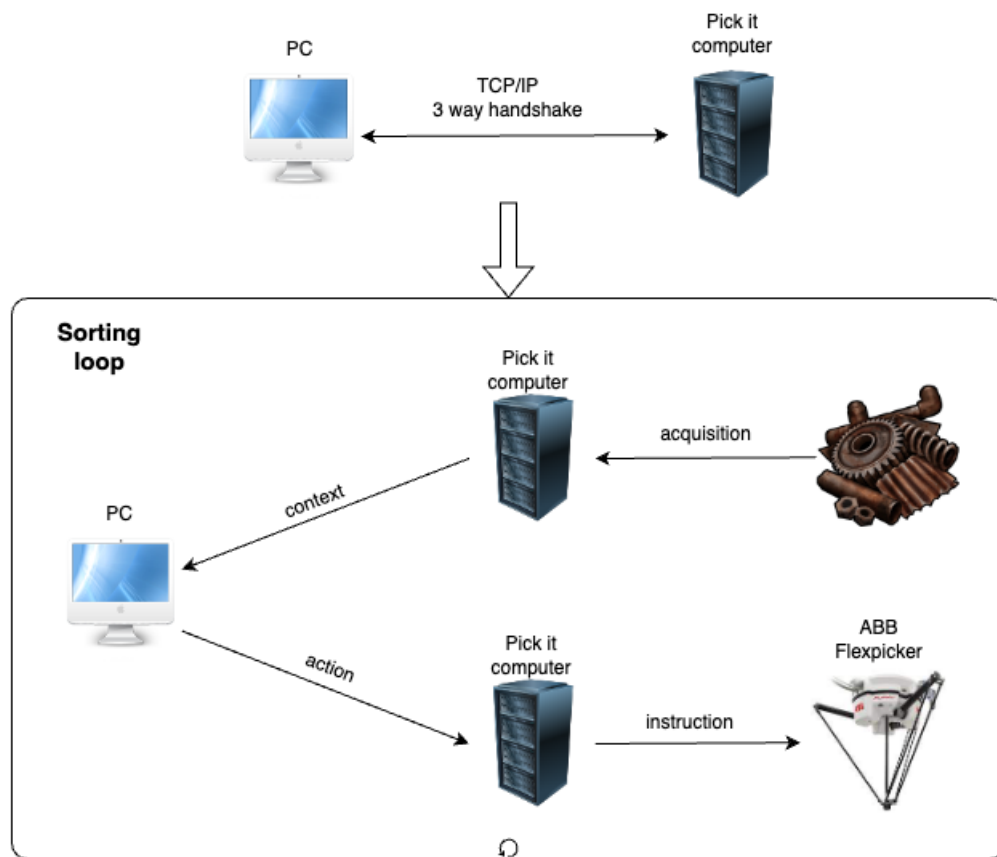


Figure 3.4.: Schema of the communication process to exchange information about actions and contexts between the Pick it computer and the remote PC with the RL model.

3.6. Experiments

Experiment 1

In the simulation, we perform a first experiment with 10,000 episodes where we report the success rate, the distance ratio in order to evaluate if the agent is able to throw objects and the reward.

Experiment 2

To evaluate if the models transfer well in reality and compare them to the pick-and-place baseline, we carry out an experiment with 5 samples of scraps and 2 agents. The 2 agents are the default pick and place routine, and our best model in simulation. The 5 samples are represented in Figure 3.6. For each sample, we run each agent and throw it in each of the 5 buckets. We test for each scenario 3 different positions on the conveyor: close, middle, and far from the buckets (Figure 3.5). Therefore, for each agent, we have 75 test throws.

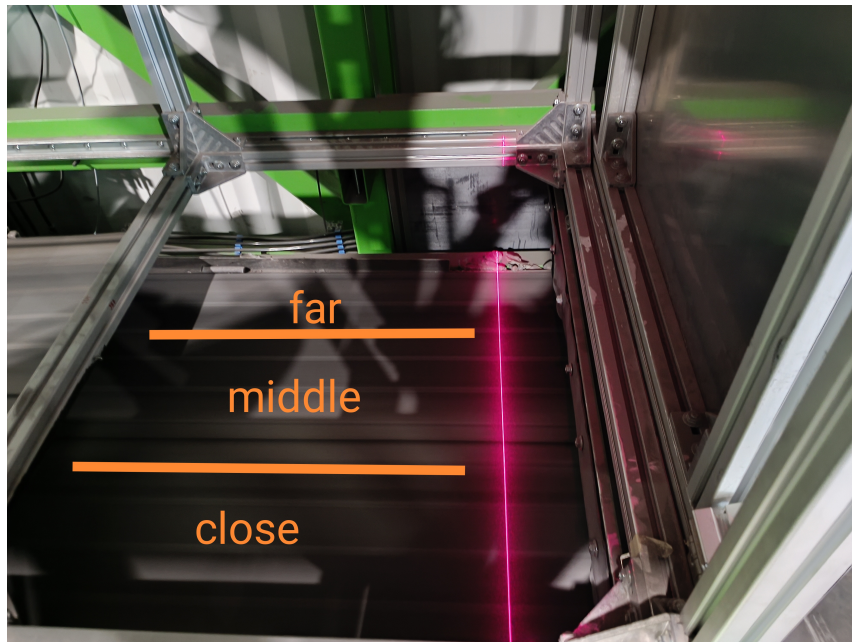


Figure 3.5.: Conveyor zones used for Experiment 2.



(a) Scrap 1: 15g

(b) Scrap 2: 30g

(c) Scrap 3: 223g



(d) Scrap 4: 51g

(e) Scrap 5: 37g

Figure 3.6.: Samples of Experiment 2.

4. Results

4.1. Hyperparameter optimization

For each algorithm, the study outcomes are reported based on 100 trials. The resultant parameters obtained from these trials are employed in Experiment 1, i.e. for complete training in simulation, and compared with the default parameters of Stable Baseline 3. Subsequently, the most favorable hyperparameters obtained after completing training are retained for the best algorithm in simulation to assess its behavior with the robot. The hyperparameters resulting from the Optuna study are presented in tabular format for reference 4.1, 4.2 and 4.3.

Table 4.1.: PPO optimized hyperparameters.

Params	Value
Learning rate	0.0067
Batch size	32
Entropy coefficient	6.92e-08
Clip range	0.4
Number of steps per rollout	256
Number of epochs	5
GAE coefficient λ	0.95
Max gradient norm	0.8
Value function coefficient	0.49
SDE sample frequency	16
number of hidden units per layer	[400, 300]
Initial log σ	-0.52
Activation function	Tanh

Table 4.2.: TD3 optimized hyperparameters.

Params	Value
Learning rate	0.0066
Batch size	512
Target smoothing coefficient τ	0.02
Training frequency	8
Noise type	ornstein-uhlenbeck
Noise standard deviation	0.673
Number of hidden units per layer	[256,256]

Table 4.3.: SAC optimized hyperparameters.

Params	Value
Learning rate	0.0016
Batch size	16
Learning starts	100
Training frequency	4
Target smoothing coefficient τ	0.005
Initial log σ	-0.075
SDE sample frequency	8
Number of hidden units per layer	[256,256]

Since these results were derived from trials comprising 40,000 episodes, it is possible that they may not surpass the performance of the default parameters but are anticipated to yield improvements. It is conceivable that optimizing over trials with a higher number of episodes may be more suitable. Consequently, the prospect of parallelizing computations becomes an enticing option for this specific task. Leveraging a database and the Optuna interface allows for effortless parallelization and implementation. However, due to time limitations within the scope of this study, this aspect was not pursued. More information about the study can be found in the appendix A.

4.2. Simulation results

In this section, we outline the findings from our first experiment, in which we trained each algorithm through one million episodes utilizing two sets of hyperparameters. These hyperparameters were derived from the default parameters of Stable Baseline 3 and from the Optuna study previously described.

The training was facilitated by the Stable Baseline 3 library and the corresponding 'learn' function associated with each algorithm. Post-training, we evaluated the performance of each algorithm on a test set comprising 10,000 episodes. The key data points collected for each trial include the mean reward with its standard deviation, the success rate, the average action time, and the distance ratio. The latter is a metric designed to measure the portion of the robot's maximum displacement completed prior to tossing the object in the bucket. Therefore, a distance ratio close to 0 consists in tossing the object whereas values close to 1 correspond to pick and place.

The accumulated results for all six agents can be found in Table 4.4.

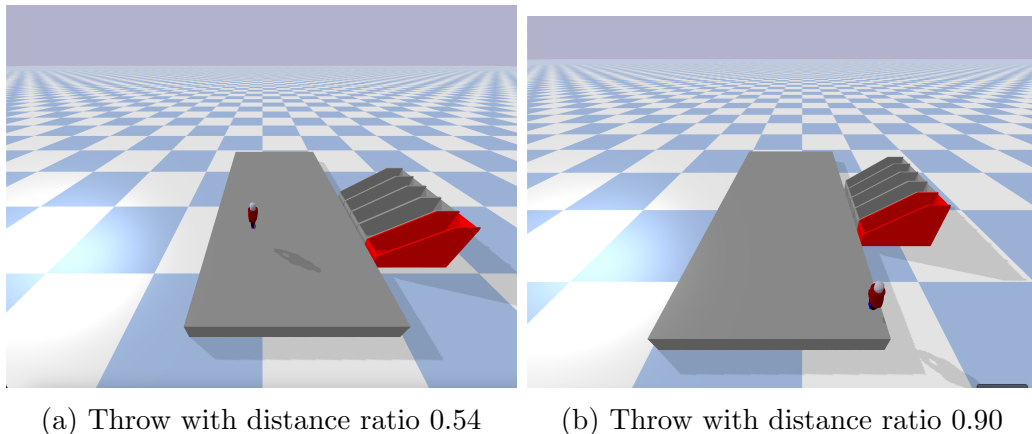
Table 4.4.: Results of Experiment 1.

Algorithm		Reward(s)	Success	Time(s)	Distance ratio
TD3	Default	0.239 ± 0.154	94.75%	0.390 ± 0.124	0.74 ± 0.24
	Optuna	0.233 ± 0.163	94.90%	0.387 ± 0.110	0.72 ± 0.25
SAC	Default	0.234 ± 0.140	96.98%	0.399 ± 0.108	0.74 ± 0.23
	Optuna	0.231 ± 0.156	95.31%	0.391 ± 0.111	0.73 ± 0.25
PPO	Default	-0.08 ± 0.131	43.84%	0.391 ± 0.086	0.72 ± 0.27
	Optuna	-0.015 ± 0.282	46.13%	0.273 ± 0.176	0.46 ± 0.34

Interestingly, we find that the highest performance is achieved using the default parameters of TD3. It is also observed that the default parameters of SAC yield better performance than the optimized ones, even though the results are very closely matched. More generally, the performance of TD3 and SAC, using both sets of parameters, show similar results. However, SAC leans towards more accurate tosses, being 2% more precise than TD3, albeit slightly slower by an average of 10 milliseconds. These four agents manage to cover an average of three-quarters of the total

distance between the object and the bucket.

It’s important to highlight that in our simulations, we model a wider variety of scenarios in which the object can be further away from the bucket than what is usually encountered in real-world situations. This is outlined in detail in the domain randomization section. These simulated situations pose increased risks for the robot when attempting to throw the object if it’s not directly in front of the bucket. This behaviour is clearly depicted in the final policy adopted by the agents. For instance, in situation 1 (Figure 4.1a), the robot learns to quickly throw the object, while in situation 2 (Figure 4.1b), it adopts a safer strategy, nearly placing the object directly into the bucket. Consequently, the evaluation of the distance ratio tends to be skewed towards higher values than what would typically be observed in reality.



We also observe that PPO struggles to develop efficient policies, even though the algorithm with the optimized parameters shows improved performance. This is illustrated in Figure 4.2, which captures the evolution of the reward throughout the training, where PPO appears to flounder without being able to enhance its policy.

Furthermore, both SAC and TD3 agents attain comparable performances after one million episodes. However, it is noteworthy that the algorithms with the optimized parameters learn significantly faster. In contrast, those with the default parameters from Stable Baseline 3 take longer to catch up, and surprisingly, even slightly surpass the others eventually. This particular trend of faster training with optimized hyperparameters can be more clearly viewed in Figure 4.3, which provides a closer look at the success rate of the algorithms during the initial 10,000 episodes.

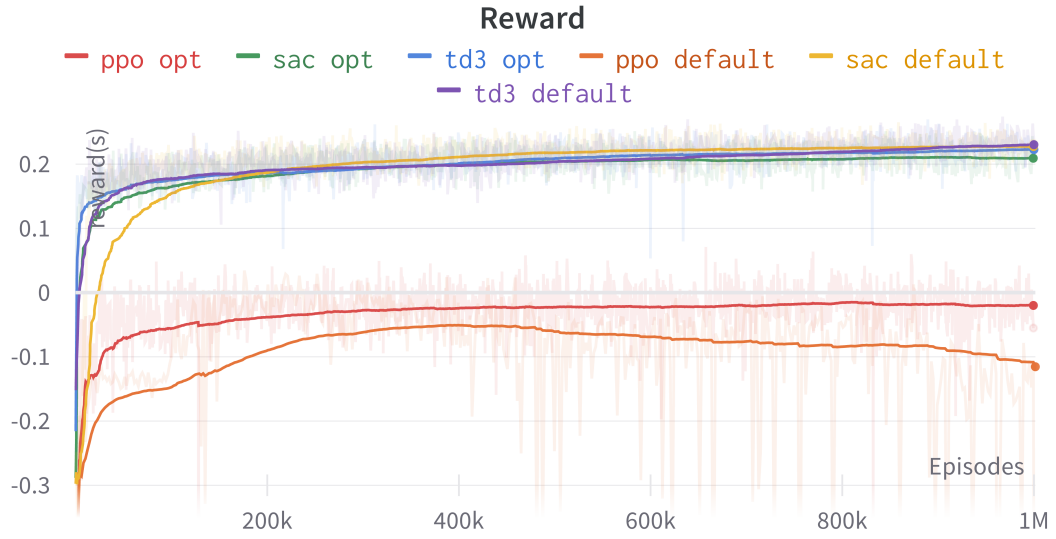


Figure 4.2.: Reward evolution during training.

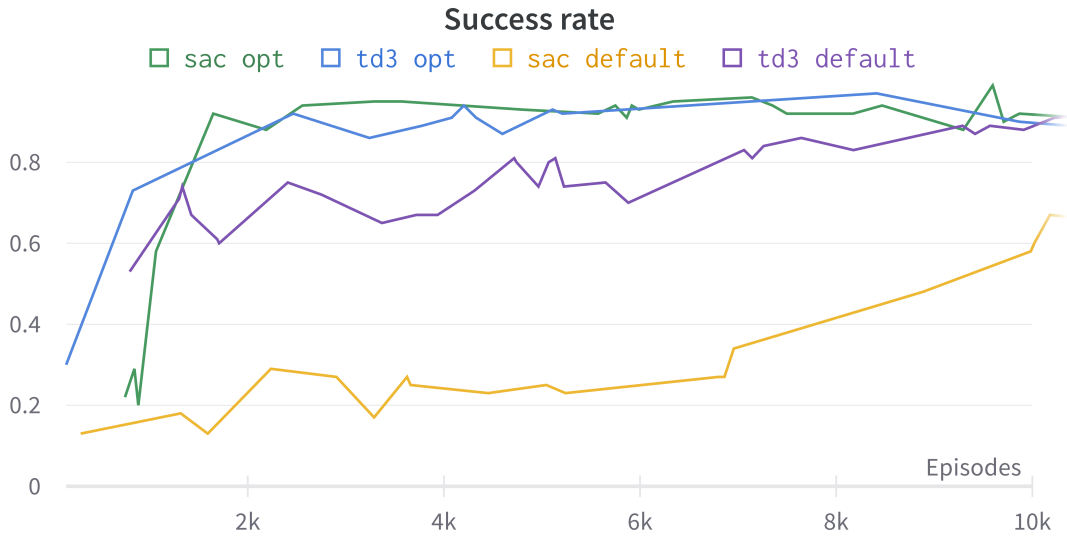


Figure 4.3.: Success rate evolution during training.

To quantify an agent's propensity to toss the object as opposed to merely placing it into the bucket, we examine the histogram depicting the distribution of the distance ratio for each algorithm, as illustrated in Figure 4.4. The data for this analysis was gathered during the evaluation phase of 10,000 episodes.

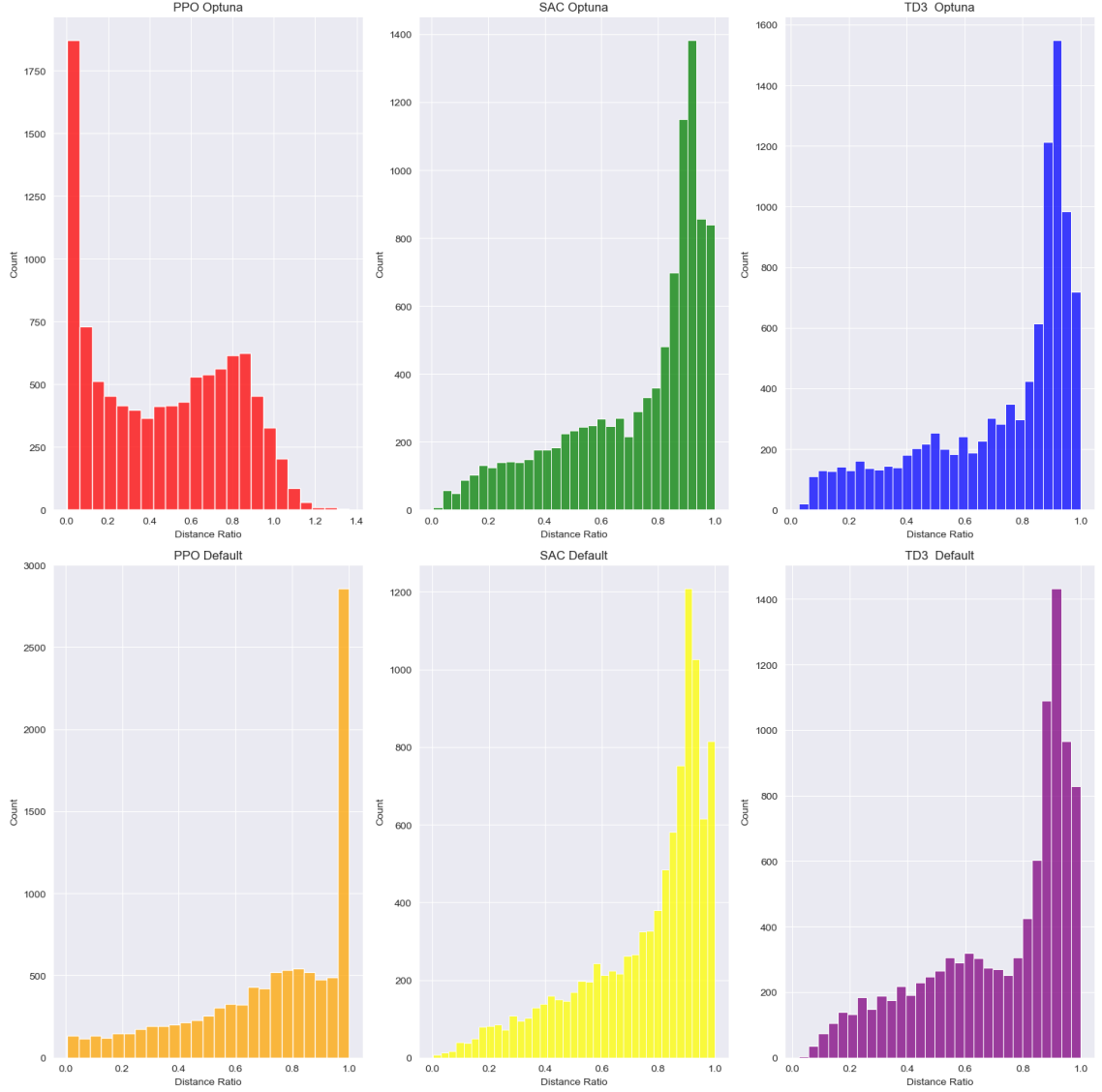


Figure 4.4.: Distance ratio distribution for each agent.

Upon analysis, we find that a significant number of tosses by the proficient agents closely resemble a pick-and-place operation. However, while maintaining a high success rate, these superior agents also manage to ensure a good proportion of tosses with a distance ratio between 0.4 and 0.8. The best performance in this regard is demonstrated by the TD3 agent with default parameters, as corroborated by the reward scores. A short demo can be found on Youtube¹.

¹<https://youtu.be/g3ExOPykhU>

In conclusion, we have successfully trained policies in a simulated environment capable of effectively tossing objects. In the forthcoming section, we will assess the practical efficacy of these policies under real-world conditions.

4.3. Transfer of the results in real-world

This section aims to delve into the outcomes of Experiment 2, which assesses the transferability of the policies learned in the simulation to real-world robots. We will hence evaluate the success rate of the best algorithm (TD3 with the default hyperparameters of Stable Baseline 3) and the time taken to throw the object, comparing these metrics with those of the current pick-and-place program.

For the existing program tasked with placing the object in the corresponding bucket, we attain a success rate of 100%, provided the scrap has been successfully picked up.

Table 4.5 provided below displays the time taken by the pick-and-place program for each scrap, at each position, and for every bucket.

Table 4.5.: Pick and Place times.

Bucket	Position	Time (ms)				
		Scrap 1	Scrap 2	Scrap 3	Scrap 4	Scrap 5
Bucket 1	Close	436	463	468	502	444
	Middle	480	477	437	429	439
	Far	475	484	542	476	502
Bucket 2	Close	454	460	466	427	466
	Middle	430	451	435	446	481
	Far	445	454	478	482	476
Bucket 3	Close	422	384	403	419	397
	Middle	418	415	409	441	433
	Far	435	431	442	432	432
Bucket 4	Close	336	357	341	376	372
	Middle	377	395	376	416	373
	Far	431	434	450	453	408
Bucket 5	Close	383	282	291	284	365
	Middle	331	399	335	402	391
	Far	454	372	454	356	364

The average time taken to place an item is approximately 421 ms. Under the current policy, there's no discernible difference between the times taken to place individual pieces. The average placement times for each piece are 420, 417, 422, 423, and 423 milliseconds respectively, just as we would expect.

It's important to note that the pieces first pass by bucket 5 before reaching bucket

1. As each piece is conveyed one after another, the robot swiftly picks up each one as soon as it enters its operational zone. This setup leads us to anticipate shorter placement durations for bucket 5 in comparison to bucket 1, for instance.

The results for the TD3 algorithm are presented in Table 4.6. It yields an average processing time of 494 ms, which is 73 ms slower compared to the standard pick-and-place procedure. Although unexpected, this is primarily attributable to one particular problem.



Figure 4.5.: Throw with the Flexpicker using TD3 algorithm.

The inference of the robot’s action requires transmitting the piece’s position at the pick point, as depicted in Figure 3.4. However, this position only becomes known once we transition to the robot’s reference system, that is, after the piece has been grasped. Consequently, the communication process necessary for inferring the action and retrieving it back introduces a delay in the movement.

To mitigate this delay, we employed a strategy that involves sending the necessary information during the object’s grasping process. Specifically, after successful grasping at the conveyor level, the robot ascends to a post-pick position. We then dispatch the information immediately after the grasping is completed, leveraging this upward

movement to calculate the action and relay it back to the robot, minimizing any potential waiting time.

Despite our best efforts, this workaround was not entirely adequate. A potential solution could be to exploit the conveyor tracking program to infer the object’s position before picking it up, thereby eliminating the delay entirely. However, this hasn’t been implemented in this study due to the need for expert assistance with ABB robots, which was unavailable within the timeframe of the master’s thesis.

Table 4.6.: TD3 times.

Bucket	Position	Time (ms)				
		Scrap 1	Scrap 2	Scrap 3	Scrap 4	Scrap 5
Bucket 1	Close	536	532	545	511	538
	Middle	534	551	540	547	541
	Far	544	553	569	552	574
Bucket 2	Close	491	423	487	400	493
	Middle	524	524	512	543	410
	Far	491	528	539	538	569
Bucket 3	Close	453	462	451	468	470
	Middle	471	424	405	486	475
	Far	542	528	525	541	523
Bucket 4	Close	351	432	464	442	459
	Middle	482	442	505	474	473
	Far	447	547	521	545	511
Bucket 5	Close	440	454	421	428	432
	Middle	451	460	452	473	476
	Far	503	511	502	537	543

Once again, we note a negligible variance in the time taken for handling different pieces, regardless of their diverse masses that should, in theory, influence the actions undertaken. The average processing times for pieces 1 through 5 are 484, 491, 496, 499, and 499 milliseconds, respectively.

When compared to the pick-and-place routine, the standard deviation between the varying positions—close, middle, and far—ranges from 23 ms to 35 ms. This range suggests a slight increase in variability for the algorithm as opposed to the pick-and-place process, albeit the difference is not significant.

A similar assessment for the various buckets indicates a range from 44 ms to 30 ms. This observation reveals that the algorithm’s performance is less impacted by the

bucket’s positioning. However, interpreting these results is challenging given the close proximity of the variances. Thus, a definitive conclusion cannot be confidently drawn from this data.

Table 4.7.: TD3 success.

Bucket	Position	success 1, fail 0				
		Scrap 1	Scrap 2	Scrap 3	Scrap 4	Scrap 5
Bucket 1	Close	1	0	1	1	1
	Middle	1	0	1	1	1
	Far	1	1	1	1	1
Bucket 2	Close	0	0	1	1	1
	Middle	1	0	1	1	1
	Far	1	1	1	1	1
Bucket 3	Close	1	1	1	1	1
	Middle	1	0	1	1	1
	Far	1	1	1	1	1
Bucket 4	Close	1	0	1	0	0
	Middle	0	0	1	1	1
	Far	0	0	0	1	1
Bucket 5	Close	0	0	0	0	1
	Middle	1	1	0	1	1
	Far	0	0	0	1	0

When it comes to the success rate of the tosses (Table 4.7), the TD3 algorithm accomplishes a success rate of 69.33%. Depending on the positioning—close, middle, and far—the algorithm attains success rates of 60%, 72%, and 76% respectively. This suggests that the algorithm encounters difficulties with tosses that are too close to the bucket, primarily due to the reduced angle which often results in the scraps hitting the side of the bucket. Pieces located in the middle of the conveyor, on the other hand, demonstrate the highest success rate, likely due to an optimal balance between alignment with the bucket and a suitable distance. However, pieces further along the conveyor exhibit lower success rates; additional data would be required to corroborate this observation.

In terms of the success rate relative to the buckets, the respective success rates are 40%, 46.67%, 93.33%, 80%, and 86.67%. Given that bucket 5 is the closest to the incoming stream of pieces and bucket 1 is the farthest, it appears that the farther the bucket, the greater the difficulty due to increased throwing distance and a more constrained throwing angle.

Lastly, when examining the success rate conditioned on the type of scrap, we see respective success rates of 66.67%, 33.33%, 66.67%, 93.33%, and 86.67% for each scrap type, based on 15 throws per type. It's evident that scrap 2, with its more irregular shape, presents greater challenges for accurate tossing as compared to the more compact scraps 4 and 5, for instance. Scrap 1 and 3 act as intermediate pieces and already shows less efficiency than scrap 4 and 5 which achieve quite promising scores.

In addition, a video of a throw performed during the experiment can be viewed on YouTube². Furthermore, a video showcasing the TD3 algorithm applied to a classical sorting procedure was recorded and can be accessed through this link³. The video provides visual evidence of the robot's performance, demonstrating that it achieves a commendable level of efficiency, although it still encounters challenges when handling certain scraps.

When contrasting the performance of the algorithm in real-world settings with its simulation results, there's a noticeable decline in both the success rate and speed. This gap between simulation and reality was anticipated; however, even without fine-tuning for real-world conditions, the results are already encouraging. Furthermore, a comparison of the average processing time of TD3 in simulation, which is 390ms, reveals it outperforms the standard pick-and-place routine's average time of 421ms. This is particularly noteworthy given that the simulation posed greater challenges due to increased distances and complexity.

²<https://youtube.com/shorts/JSFymAOig4Y>

³<https://youtube.com/shorts/ONAUlJNZ6AA>

5. Conclusions

5.1. Summary of the findings

This master’s thesis has focused on the development of a simulated environment using PyBullet, during which policies for three key deep reinforcement learning algorithms, namely TD3, SAC, and PPO, implemented by stable baseline 3, were learned and analyzed. The understanding of these algorithms was enhanced, contributing to a broader comprehension of their operations and unique characteristics.

A reward function was engineered utilizing both employee knowledges and deep learning methodologies. This combination facilitated the creation of a capable function that was instrumental in directing the algorithms towards beneficial policy learning with a fair tradeoff between the success of a throw and its speed. This complex process required a detailed understanding of the domain but the outcome was a meaningful enhancement in the comprehension and overall performances of the models.

The study also delved into hyperparameter optimization, a critical aspect of improving policy learning. By adjusting the parameters influencing the learning process thanks to an Optuna study, we expected an observed increase in the models’ ability to learn superior policies but finally observed an increase in the speed to learn these policies but similar performance for the two best algorithms which were TD3 and SAC. This exploration substantiated the role of hyperparameter tuning in improving learning algorithm performance in simulation but also for further research in learning efficient policies in real-world settings where the episode budget is smaller.

This thesis explored domain randomization to address the sim-to-real gap. The technique enabled the introduction of diverse simulated environments where the

positions of the scraps and the buckets but also the physical properties of the scraps changed. It is expected to promote a robustness in the models that could potentially increase their resilience to real-world unpredictability.

The process of integrating the model into the Pick It program involved implementing and optimizing a communication process. This stage was critical for ensuring an efficient information transfer that would allow the full utilization of the learning algorithms in a functional context.

Performance assessments were conducted in both simulation and real-world scenarios. These evaluations provided important insights into the transferability of the algorithms and the robustness of the model in a real-world context. Despite a slight decrease in performance when moving from simulation to reality, the findings demonstrated potential for further refinement and application in real-world scenarios.

To summarize, this master’s thesis represents a notable contribution to the GeMMe lab by delivering valuable insights and methodologies that can be used to train agents, thereby enhancing the speed of the sorting process. While the current results may not surpass the pick-and-place routine—given a lower success rate and slower operation due to the existing communication process—the potential for significant improvements remains apparent. These could be realized through fine-tuning the algorithms in real-world conditions, incorporating sensors, and advancing the communication process.

In particular, the integration of sensors could enable us to determine whether the scraps have been accurately deposited in the correct bucket, and if the toss timing can be measured effectively. Such enhancements would allow us to implement the reward function used in simulation in a real-world context, potentially facilitating direct learning without reliance on the simulator, except as an initial step or ‘warm start’. This approach could offer a pathway to substantial gains in the effectiveness of the sorting process.

5.2. Reflection

In the process of developing a pipeline to train agents and subsequently deploy them in a real-world scenario, there were potential areas for further enhancement.

Indeed, the first semester was dedicated to building the simulator, leveraging data and the robotics engineering expertise available at the GeMMe lab. This collaboration facilitated a detailed understanding of the robot’s operations, leading to the creation of an effective formulation. However, a more accurate alignment with reality could have been achieved after I fully understood the robot’s code during the internship in the second semester.

For instance, the implementation of a module was required to transition between the reference system of the simulator and that of the robot, and vice versa. This step might have been unnecessary if there had been an initial comprehensive understanding of the robot’s reference system and its particular working area. A more accurate depiction of these aspects in the simulator might have resulted in a closer match to reality, potentially improving the performance of the trained agents when deployed.

Moreover, the exploration of hyperparameter optimization using Optuna was initiated later in the thesis process. Given the computational intensity of this step, the delay resulted in less time for setting up a study with a greater number and longer duration of trials. Such an approach might have improved the performance of the algorithms, rather than merely hastening the rate at which the algorithm plateaus.

Furthermore, the implementation of parallelization could have significantly increased our capacity to run an expansive study, with numerous workers simultaneously exploring the hyperparameter space. However, establishing parallelization required a database that could interface with the Alan cluster, a requirement that couldn’t be met within the given timeframe.

5.3. Recommendations for future research

In order to achieve competitive performances, future research should explore the opportunity to fine-tune policies learned in simulation in real conditions. Algorithms such as TD3 and SAC can be used to learn online or data could be collected to train offline algorithms such as PPO even if the latter struggled to learn efficient policies with the hyperparameters of this study.

Further improvements could be realized through problem reformulation, enabling the agent to select multiple points during its displacement. This could broaden the scope of potential actions and accommodate a multi-step reinforcement learning setting. Such a reformulation could also incorporate the real-world constraints of the robot, as well as a more precise depiction of the bucket's position and characteristics.

A well-known result of machine learning algorithms is that the more data is available the better. Therefore, giving a better description of the scraps to the algorithm could help instead of just giving its mass. Data such as a 3D description obtained by the 3D camera could help the algorithm to understand the shape of the scrap and anticipate its physics during the throw. Even better a description of the way the gripper has picked the object can help the algorithm as different picking angles will result in different behavior. However, the difficulty with this approach is to obtain the same data in simulation for example as it would require a dataset of objects close to scraps and images similar to images collected in reality but it still remains possible with more time and investment.

Finally, exploring sim2real techniques could also help to close the reality gap when learning in simulation. Exploring techniques such as domain adaptation, improving the domain randomization technique used and calibrating better the simulation could be interesting tracks. It would also be relevant to perform an ablation study in our case to quantify the added value of these techniques.

Bibliography

- [Akiba et al., 2019] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- [Bergstra et al., 2011] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.
- [Bergstra and Bengio, 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).
- [Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.
- [Collins et al., 2021] Collins, J., Chand, S., Vanderkop, A., and Howard, D. (2021). A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431.
- [Coumans and Bai, 2021] Coumans, E. and Bai, Y. (2016–2021). Pybullet, a python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>.
- [Duckworth et al., 2023] Duckworth, P., Lacerda, B., Vallis, K., and Hawes, N. (2023). Reinforcement learning for bandits with continuous actions and large context spaces.
- [Fujimoto et al., 2018] Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477.

- [Garivier and Moulines, 2011] Garivier, A. and Moulines, E. (2011). On upper-confidence bound policies for switching bandit problems. In Kivinen, J., Szepesvári, C., Ukkonen, E., and Zeugmann, T., editors, *Algorithmic Learning Theory*, pages 174–188, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905.
- [Ibarz et al., 2021] Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. (2021). How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721.
- [Krause and Ong, 2011] Krause, A. and Ong, C. (2011). Contextual gaussian process bandit optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.
- [Li et al., 2010] Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM.
- [Lillicrap et al., 2019] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2019). Continuous control with deep reinforcement learning.
- [Majzoubi et al., 2020] Majzoubi, M., Zhang, C., Chari, R., Krishnamurthy, A., Langford, J., and Slivkins, A. (2020). Efficient contextual bandits with continuous actions. *CoRR*, abs/2006.06040.
- [Marlier et al., 2019] Marlier, N., Louppe, G., Bruls, O., and Dislaire, G. (2019). Robotic throwing controller for accelerating a recycling line. In *Proceedings of the Robotix Academy Conference for Industrial Robotics (RACIR) 2019*. F.R.S.-FNRS - Fonds de la Recherche Scientifique, Robotix Academy.
- [Passos and Mishra, 2022] Passos, D. and Mishra, P. (2022). A tutorial on automatic hyperparameter tuning of deep spectral modelling for regression and classification tasks. *Chemometrics and Intelligent Laboratory Systems*, 223:104520.

- [Raffin, 2020] Raffin, A. (2020). Rl baselines3 zoo. <https://github.com/DLR-RM/rl-baselines3-zoo>.
- [Raffin et al., 2021] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.
- [Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.
- [Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Beijing, China. PMLR.
- [Sutton et al., 1999] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press.
- [Tobin et al., 2017] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907.
- [Weng, 2019] Weng, L. (2019). Domain randomization for sim2real transfer. *lilian-weng.github.io*.
- [Yoon et al., 2019] Yoon, H. J., Chung, S. Y., Kang, H. S., and Hwang, M. J. (2019). Trapezoidal motion profile to suppress residual vibration of flexible object moved by robot. *Electronics*, 8(1).
- [Zhu et al., 2022] Zhu, Y., Foster, D. J., Langford, J., and Mineiro, P. (2022). Contextual bandits with large action spaces: Made practical.

A. Appendix

A.1. Algorithms

Proximal Policy Optimization (PPO)

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

Figure A.1.: PPO algorithm reprinted from OpenAI Spinning Up¹

¹<https://spinningup.openai.com/en/latest/algorithms/ppo.html> (last consulted 9 June 2023)

Soft Actor-Critic (SAC)

Algorithm 1 Soft Actor-Critic

```

1: Input: initial policy parameters  $\theta$ , Q-function parameters  $\phi_1, \phi_2$ , empty replay buffer  $\mathcal{D}$ 
2: Set target parameters equal to main parameters  $\phi_{\text{targ},1} \leftarrow \phi_1, \phi_{\text{targ},2} \leftarrow \phi_2$ 
3: repeat
4:   Observe state  $s$  and select action  $a \sim \pi_\theta(\cdot|s)$ 
5:   Execute  $a$  in the environment
6:   Observe next state  $s'$ , reward  $r$ , and done signal  $d$  to indicate whether  $s'$  is terminal
7:   Store  $(s, a, r, s', d)$  in replay buffer  $\mathcal{D}$ 
8:   If  $s'$  is terminal, reset environment state.
9:   if it's time to update then
10:    for  $j$  in range(however many updates) do
11:      Randomly sample a batch of transitions,  $B = \{(s, a, r, s', d)\}$  from  $\mathcal{D}$ 
12:      Compute targets for the Q functions:

```

$$y(r, s', d) = r + \gamma(1 - d) \left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

```

13:    Update Q-functions by one step of gradient descent using

```

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2 \quad \text{for } i = 1, 2$$

```

14:    Update policy by one step of gradient ascent using

```

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s) \right),$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.

```

15:    Update target networks with

```

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho) \phi_i \quad \text{for } i = 1, 2$$

```

16:    end for

```

```

17:  end if

```

```

18: until convergence

```

Figure A.2.: SAC algorithm reprinted from OpenAI Spinning Up²

²<https://spinningup.openai.com/en/latest/algorithms/sac.html> (last consulted 9th June 2023)

Twin Delayed Deep Deterministic Policy Gradient (TD3)

Algorithm 1 Twin Delayed DDPG

- 1: Input: initial policy parameters θ , Q-function parameters ϕ_1, ϕ_2 , empty replay buffer \mathcal{D}
- 2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta, \phi_{\text{targ},1} \leftarrow \phi_1, \phi_{\text{targ},2} \leftarrow \phi_2$
- 3: **repeat**
- 4: Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$, where $\epsilon \sim \mathcal{N}$
- 5: Execute a in the environment
- 6: Observe next state s' , reward r , and done signal d to indicate whether s' is terminal
- 7: Store (s, a, r, s', d) in replay buffer \mathcal{D}
- 8: If s' is terminal, reset environment state.
- 9: **if** it's time to update **then**
- 10: **for** j in range(however many updates) **do**
- 11: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from \mathcal{D}
- 12: Compute target actions

$$a'(s') = \text{clip}(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}}), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

- 13: Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

- 14: Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2 \quad \text{for } i = 1, 2$$

- 15: **if** $j \bmod \text{policy_delay} = 0$ **then**
- 16: Update policy by one step of gradient ascent using

$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

- 17: Update target networks with

$$\begin{aligned} \phi_{\text{targ},i} &\leftarrow \rho \phi_{\text{targ},i} + (1 - \rho) \phi_i & \text{for } i = 1, 2 \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho) \theta \end{aligned}$$

- 18: **end if**
 - 19: **end for**
 - 20: **end if**
 - 21: **until** convergence
-

Figure A.3.: TD3 algorithm reprinted from OpenAI Spinning Up³

³<https://spinningup.openai.com/en/latest/algorithms/td3.html> (last consulted 9th June 2023)

A.2. Hyperparameters

At the end of the Optuna study, several pieces of information can be retrieved in order to analyze the results and provide guidance for better fine-tuning. The parameter importance plot⁴ for each study is reported below to guide a new study with better parameter value ranges and better sets of hyperparameters to optimize.

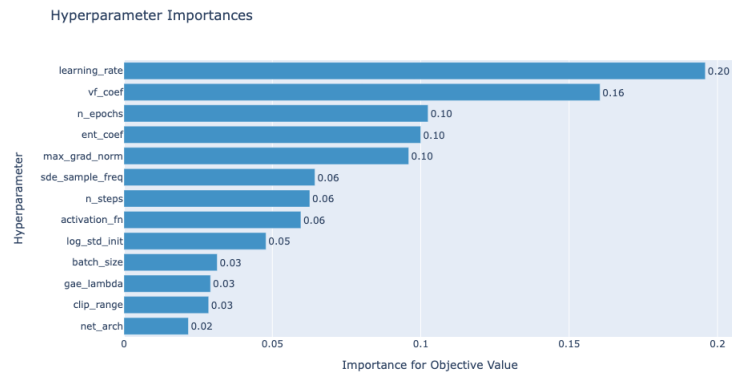


Figure A.4.: Parameter importance for PPO

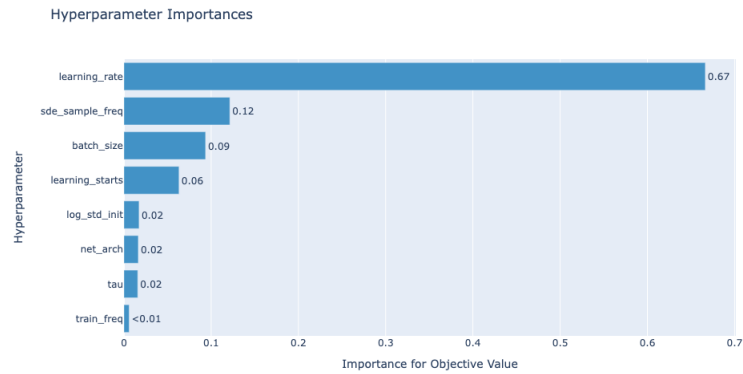


Figure A.5.: Parameter importance for SAC

⁴For more information: <https://optuna.readthedocs.io/>

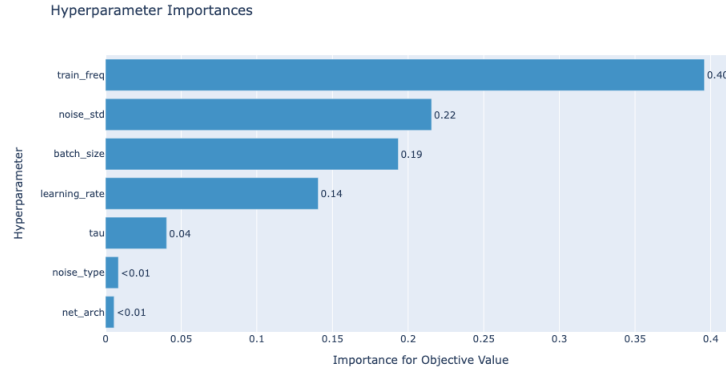


Figure A.6.: Parameter importance for TD3

As we can observe, the learning rate plays a big role, especially as the number of episodes for each study is quite limited (40k episodes). This might lead to an overestimation of the set of parameters with a large learning rate as they might lead more rapidly to decent policies, but these policies will be stuck in local minima. A solution to avoid this problem is to provide a larger episode budget for each trial.

A.3. Training complements

The reward neural network used was trained using the mean squared error loss over 300,000 episodes and was evaluated over 10,000 episodes with a mean error of 28 ms. The histogram of the error is represented below:

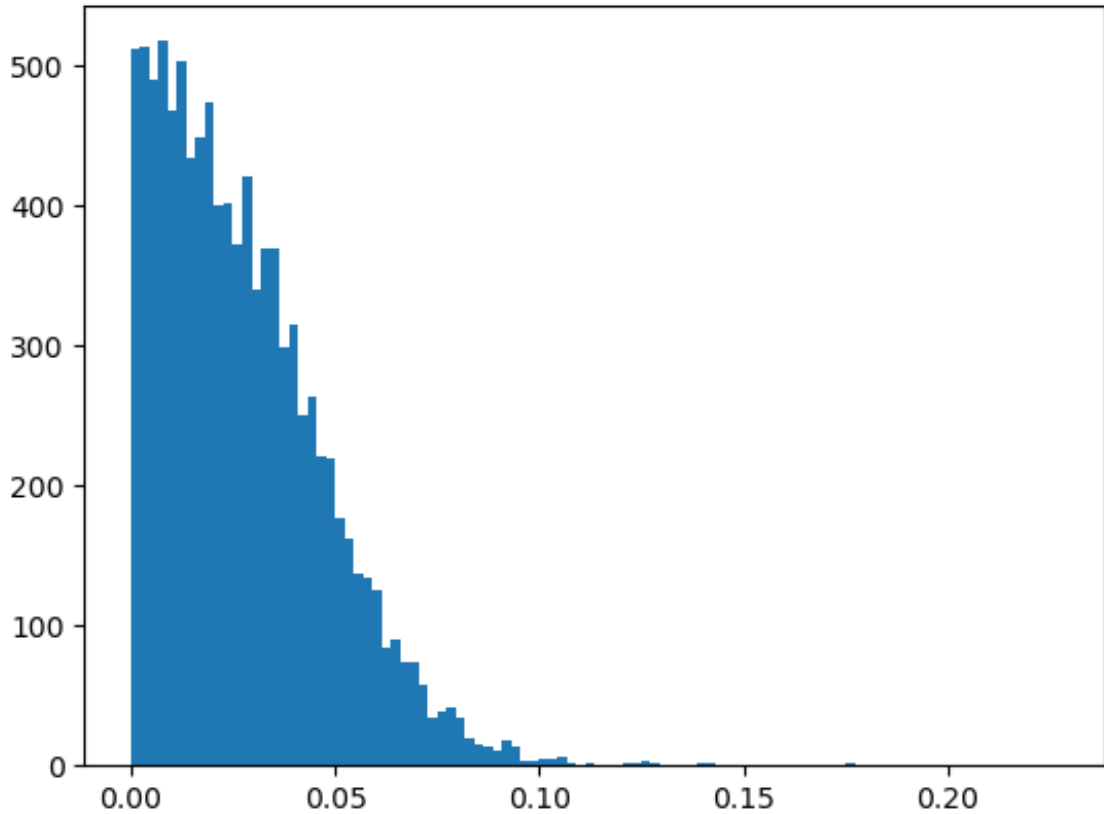


Figure A.7.: Histogram of the errors made by the reward neural network over 10,000 episodes, with the absolute error on the x-axis (in seconds) and the count on the y-axis.

Although the mean error of 28 ms obtained during the evaluation is not negligible, it was deemed sufficient for the intended purpose of the reward function. The error, represented in the histogram (Figure A.7), demonstrates that the majority of errors fall within an acceptable range for the task at hand. The evaluation results indicated that the reward function adequately captures the desired criteria and provides meaningful feedback to guide the learning process of the reinforcement learning algorithms employed in the sorting process. Therefore, despite the presence

of some errors, the performance of the reward function was considered satisfactory and suitable for the purposes of this study.