Self-Collision Aware Reaching and Pose Control in Large Workspaces using Deep Reinforcement Learning

by

Tumuçin BAL

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Sciences and Engineering

December 6, 2023

Self-Collision Aware Reaching and Pose Control in Large Workspaces using Deep Reinforcement Learning

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Tumuçin BAL

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assist. Prof. Barış Akgün (Advisor)

_____

Assoc. Prof. Erol Şahin

_____

Assoc. Prof. Emre Uğur

Date: _____

Dedicated to myself, my beloved family, Darüşşafaka, and all those people who have contributed and will continue to contribute to the science.

ABSTRACT

Self-Collision Aware Reaching and Pose Control in Large Workspaces using Deep Reinforcement Learning
Tumuçin BAL
Master of Science in Computer Sciences and Engineering
December 6, 2023

Reaching, pose control, and inverse kinematics are fundamental robotic manipulator tasks that underpin other tasks and as such, there is a vast body of related literature from various fields. Control, planning, and more recently learning fields are among the main ones. Traditional control algorithms are prone to failure around singularities and joint limits and do not naturally handle self-collisions. Planning methods are not fast enough for reactive behaviours and require additional infrastructure, including controllers, to work. Learning-based methods have emerged to tackle these issues. However, most of them do not handle arbitrary initial and target poses, ignore self-collisions, do not include orientation information in their targets, work in small workspaces and evaluate themselves with coarse success metrics.

In this thesis, we introduce a novel hybrid approach that combines Pseudo-inverse control (PinvC) and model-free reinforcement learning (RL), including state space and reward function design, to fill these gaps in the context of reaching, pose control and inverse kinematics. PinvC already calculates joint velocities given desired task-space (e.g. the end-effector pose) velocities and only requires the kinematic structure of the robot. PinvC is mostly reliable away from joint limits, singularities and when individual links are not prone to collisions. The main idea behind our approach is to use RL to handle these situations. Towards this end, we design a novel state space and reward functions. Our reward function aims to minimize position (reaching task) or pose (inverse kinematics and pose control tasks) errors, reduce self-collisions and reduce joint velocities near the target. Furthermore, we develop a curriculum learning methodology to aid learning. Lastly, we introduce a simple modification, which we call *switching* to further improve task performance.

We evaluate our approach with four simulated robots for various problem settings and compare it against traditional and learning-based approaches. Our results

show that our approach decidedly outperforms the baselines in terms of mean error, success rates at various thresholds and terminal speed for reaching tasks. In addition, we reduced the number of self-collisions across all the scenarios. Our approach achieved better results when orientation was included, but none of the methods performed very well, especially the learning baselines. We note that the learning-based methods in the literature almost always ignore orientation. As a result, we comprehensively discuss the reasons for orientation failure and potential remedies.

ÖZETÇE

Büyük Çalışma Alanlarında Derin Pekiştirmeli Öğrenme Tabanlı Kendi Kendine Çarpışma Farkındalığına Sahip Erişim ve Poz Kontrolü
Tumuçin BAL
Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans
6 Aralık 2023

Erişim, poz kontrolü ve ters kinematik, diğer görevlerin temelini oluşturan robotik manipülator görevleridir. Bu nedenle literatürde, bahsedilen görevler için çeşitli alanları kapsayan bir çok çalışma mevcuttur. Kontrol, planlama ve son zamanlardaki öğrenme metodları bu alanların arasındadır. Geleneksel kontrol algoritmaları tekillik ve eklem sınırları etrafında başarısız olma eğilimindedir ve kendi kendine çarpışma farkındalığını doğal olarak ele alamazlar. Planlama tabanlı yöntemler, reaktif davranışlar için yeterince hızlı değildir ve çalışmak için kontrolcüler dahil ek altyapı gerektirir. Bu sorunları ele almak için öğrenmeye dayalı yöntemler ortaya çıkmıştır. Ancak öğrenmeye dayalı yöntemlerin birçoğu serbest başlangıç ve hedef pozlarını ele almaz, kendi kendine çarpışma farkındalığını göz ardı eder, hedeflerinde oryantasyon bilgisi içermez, küçük çalışma alanlarında çalışır. Ek olarak bir çoğu sadece kaba başarı metrikleri ile değerlendirilmiştir.

Bu tezde, erişim, poz kontrolü ve ters kinematik konularındaki boşlukları doldurmak için modelden-bağımsız Pekiştirmeli Öğrenme (PÖ) ve yalancı-ters kontrolü birleştiren, durum uzayı ve ödül fonksiyon tasarımını içeren yeni bir hibrit yaklaşım sunuyoruz. Yalancı-ters kontrol, istenilen görev uzayı (örneğin, uç işlevci pozu) hızları verildiğinde zaten eklem hızlarını hesaplar ve sadece robotun kinematik yapılanmasına ihtiyaç duyar. Yalancı-ters kontrol, genellikle eklem sınırlarından, tekilliklerden ve bireysel bağlantı elemanlarının birbirleriyle çarpışmadan uzak olduğu durumlarda güvenilirdir. Yaklaşımımızın ana fikri, PÖ'yü yukarıda bahsedilen durumların üstesinden gelmek için kullanmaktır. Bu amaçla, yeni bir durum uzayı ve ödül fonksiyonu tasarladık. Ödül fonksiyonumuz, pozisyon (erişim görevi) veya poz (ters kinematik ve poz kontrol görevleri) hatalarını, kendine çarpmaları azaltmayı ve hedefe yakın eklem hızlarını en aza indirmeyi amaçlamaktadır. Ek olarak, öğrenmeye yardımcı olması için bir müfredat öğrenme metodolojisi geliştirdik. Son

olarak, görev performansını daha da artırmak için *anahtarlama* adını verdiğimiz basit bir değişiklik sunduk.

Yaklaşımımızı, çeşitli durumlar ve görevler için benzetim ortamında dört robot ile değerlendirdik ve geleneksel ve öğrenmeye-dayalı yaklaşımlarla karşılaştırdık. Sonuçlarımız, yaklaşımımızın erişim görevlerinde ortalama hata, çeşitli eşiklerde başarı oranları ve son hız açısından temel yaklaşımları belirgin bir şekilde geride bıraktığını göstermektedir. Ek olarak, tüm senaryolarda kendi kendine çarpma sayısını azalttık. Oryantasyon eklendiğinde yaklaşımımız daha iyi sonuçlar elde etti, ancak yöntemlerin hiçbiri (özellikle öğrenmeye-dayalı temel aldığımız yöntemler) çok iyi performans göstermedi. Literatürdeki öğrenmeye-dayalı yöntemlerin neredeyse her zaman oryantasyonu göz ardı ettiğini görüyoruz. Bu nedenle, oryantasyon başarısızlığının nedenlerini ve potansiyel çözümleri kapsamlı bir şekilde tartışıyoruz.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor, Barış Akgün, for his guidance, support, and continuous encouragement throughout the entire process of researching and writing this thesis. I would also like to thank him for his empathy and great sense of humour.

I am also profoundly grateful to Tarık Bercan Sarı and Buse Bilgin, whose support accompanied me from the beginning to the end of my master's journey. Their generosity in sharing valuable insights whenever I need guidance has been instrumental in completing this thesis.

I am grateful to KUIS AI for granting me access to High-Performance-Computing resources for this thesis. This support significantly enhanced my research capabilities and contributed to the successful completion of my thesis.

My biggest gratitude goes to my family members - Ayşe, Özgür, Nezaket, Metin, Merve, Eslem Nisa, and Erva Nur - for their constant support, love, and understanding during my academic journey. Their encouragement taught me how to overcome challenges and keep moving forward. I owe much of my success to their belief in me. I am grateful for their sacrifices, understanding, and patience, which have enabled me to pursue my dreams. I am truly blessed to have such a loving and supportive family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

PRM	Probabilistic Roadmap
RRT	Rapidly Exploring Random Trees
BiRRT	Bi-Directional Rapidly Exploring Random Trees
OSC	Operational Space Control
MPC	Model Predictive Control
PinvC	Pseudo-Inverse Control
DRL	Deep Reinforcement Learning
HER	Hindsight Experience Replay
PPO	Proximal Policy Optimization
SAC	Soft Actor Critic
IK	Inverse-Kinematics
U_6	Universal Robot UR5
P_7	Franka Emika
Iiwa	KUKA LBR Iiwa
NAF	Normalized Advantage Functions
TD3	Twin Delayed DDPG
HCP	Hardware Conditioned Policies
D4PG	Distributed Distributional Deep Deterministic Policy Gradients
CL	Curriculum Learning
URDF	Unified Robotics Description Format
MAE	Mean Absolute Error
MJV	Mean Joint Velocity
QAE	Quaternion Angle Error
RL	Reinforcement Learning
VPG	Vanilla Policy Gradient

A3C	Asynchronous Advantage Actor-Critic
GAE	Generalized Advantage Estimation
TRPO	Trust Region Policy Optimization
DoF	Degree-of-Freedom
KDL	Kinematics and Dynamics Library
SR	Success Rate
DDPG	Deep Deterministic Policy Gradient
OSC	Operational Space Control
J_7	Kinova Jaco 7-DoF
J_6	Kinova Jaco 6-DoF
SB3	Stable Baselines3

Chapter 1

INTRODUCTION

1.1 Motivation

Going from one pose to the next is a common manipulator task. In this thesis, we concentrate on two specific but highly prevalent cases; (1) reaching: controlling the end-effector to a desired 3D position while ignoring orientation, and (2) full-pose control: controlling the end-effector to a desired 6D pose. One primary application of reaching and full-pose control lies in manufacturing, where robotic arms are employed for tasks such as pick-and-place operations, assembly, grasping, transporting, etc. [Jiang et al., 2013, Mohammed et al., 2022, Franceschetti et al., 2021, Zhu and Hu, 2018]. Each of these tasks requires the robot manipulator to reach a specific pose. Note that reaching and full pose control is prevalent in many manipulator applications. Figure 1.1 illustrates an example of a Franka Emika (P_7) robot reaching a randomly selected target pose in simulation.

The earliest manipulators relied on human via-point programming, which is still common in most industrial settings. This approach is only suitable for heavily structured environments with static start and target points. The manipulators need re-programming whenever the environment changes and an expert programmer is usually required to get efficient results. This process is time-consuming and requires significant expertise to apply these changes to various robot platforms [Chang and Stone, 2013].

Since reaching and full-pose control are prevalent tasks, there is a vast body of related work spanning multiple fields such as control, optimization, planning and learning [Liu and Liu, 2021, Moll et al., 2017, Števo et al., 2014, Kazem et al., 2008, James and Johns, 2016, Franceschetti et al., 2021]. These methods have their

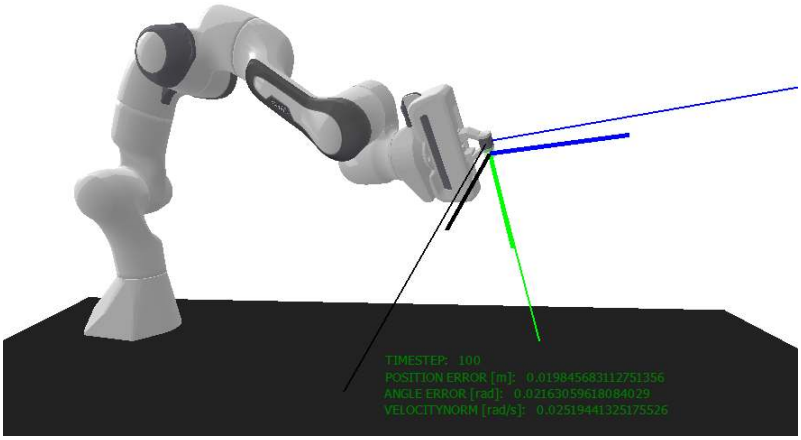


Figure 1.1: Example of a 7-DoF Panda manipulator reaching a random target pose from a random initial pose in PyBullet [Coumans and Bai, 2021] robotic simulator. The Figure includes green text that gives information about the current time step, position error in meters m, orientation error in radians rad, and mean joint velocity norm in radians per second rad/s. The steps for computing position and orientation errors are described in Chapter 5. Additionally, the definition of mean joint velocity norm is explained in Section 6.2.

own advantages and disadvantages. For example, control-based approaches have achieved great success in structured environments, especially regarding repeatability, robustness, speed and precision. [Nakanishi et al., 2008] utilizes Operational Space Control (OSC) in their work. OSC is a control methodology to control the robot's end-effector position and orientation in task space rather than controlling the robot in joint space. Moreover, OSC is useful when the task is to move the robot's end-effector to a target position or pose. The author explores OSC for velocity, acceleration, and force spaces to address redundancy in robots with a high degree-of-freedom (DoF).

In a different study, [Killpack et al., 2016] utilizes Model Predictive Control (MPC) as another control algorithm to regulate the robot's end-effector for reaching a target position in a densely cluttered environment. Control-based approaches can handle arbitrary start and target points within an envelope and offer reactivity. However, they require accurate knowledge of robot dynamics to handle constraints (joint limits, torque limits, energy expenditure, etc.) and do not naturally handle unstructured environments. Moreover, these methods require complex derivation

and computation for different robots. Besides that, industrial robotic applications often demand task-specific programming efforts. As a result, industrial robotic programmers spend considerable effort to change the current task to a desired task [Lobbezoo et al., 2021].

There is another set of algorithms, called path planning, to control robotic manipulators. These planning methods aim to move the robots from the initial configuration to another configuration. For example, the graph-based Probabilistic Roadmap (PRM) can be used to construct a map by randomly sampling collision-free vertices in space [Alatartsev et al., 2015]. A notable advantage of this approach is that the constructed graph can be reused more than once.

In addition to graph-based methods, tree-based approaches, such as Rapidly Exploring Random Trees (RRT) [LaValle et al., 2001], are also used for path planning. RRT rapidly and uniformly explores the configuration space, from a single starting point. Furthermore, the Bi-Directional Rapidly Exploring Random Trees (BiRRT) [Kuffner and LaValle, 2000] algorithm creates two trees, one rooted at the start and the other at the end. The author of [Berenson et al., 2009] extends RRT algorithms to make the robot work with several constraints, such as end-effector pose and torque constraints.

Planning-based approaches are adept at handling arbitrary start and target poses, collisions, joint limits, and most other constraints with much less effort [Berenson et al., 2009, Doerr and Linares, 2020] than control-based approaches. On the other hand, they require significant computational resources. In addition, most planning methods output a joint space trajectory, which usually needs smoothing and require additional means of control to track it. It is also not straightforward to incorporate arbitrary costs. Furthermore, quick re-planning is needed for reactive behaviour, which may not be feasible due to computational resources. There are more advanced methods that alleviate some of the smoothness and cost-related issues, but their trade-off is further computational efficiency.

In response to the challenges described above, Deep Reinforcement Learning (DRL) has emerged as an alternative to traditional methods for solving complex tasks. DRL combines the traditional Reinforcement Learning (RL) concept with

Deep Learning (DL) approaches. In the work AlphaGo [Silver et al., 2016], they train two neural networks by combining supervised learning and RL training procedures to master the Go game. This work gained attention when AlphaGo successfully defeated the human European Go champion. In recent work, AlphaTensor [Fawzi et al., 2022] introduced a DRL-based approach for multiplying arbitrary matrices. Using DRL techniques, AlphaTensor can discover a new way to do faster matrix multiplication. Moreover, it aims to reduce the number of operations for matrix multiplication.

[Kilinc and Montana, 2022] explores the robotic manipulation task using DRL without relying on human demonstrations. The initial phase of this research involves learning a locomotion policy through a simulator using Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015] combined with Hindsight Experience Replay (HER) [Andrychowicz et al., 2017]. After that, the learned locomotion policy is used to learn task-specific robotic manipulator tasks. The work presented in [Li et al., 2020b] introduces a graph-based network architecture designed to learn complex tasks through a simple Curriculum Learning (CL) approach. The network is trained using Soft Actor-Critic (SAC) [Haarnoja et al., 2018] as the baseline algorithm with HER. This approach can stack six blocks into a tower without human demonstration. However, these approaches are highly specific to a single robotic task.

The potential of DRL for solving complex control tasks leads to multiple applications such as grasping, pick and place operations, door opening, and assembly [Shahid et al., 2020, Lobbezoo et al., 2021, Inoue et al., 2017, Gu et al., 2017, Qin et al., 2019]. However, most existing literature aims to demonstrate proof-of-concept rather than general solutions due to inherent RL challenges with noisy, high-dimensional, continuous state spaces and action spaces. On the other hand, more practical and general learning-based reaching and IK methods have emerged (see Tab.2.1). Although promising, these methods still fall short regarding arbitrary start initial robot configuration and target poses, workspace volume, collision handling, singularity handling, and full-pose control. Furthermore, most of these do not use tight thresholds for evaluating success.

In this thesis, we develop a control and DRL hybrid approach that aims to be practical and general. As such, we work on large workspaces, include arbitrary start initial robot configuration and target poses, apply our approach to multiple robots without changing parameters and use tight metrics for evaluation. Our approach also involves a minimal setup, only requiring robot kinematic structure and a simulator for self-collision checks.

1.2 Contributions

This thesis introduces a hybrid method to learn a velocity controller in the joint space that can move the robot to a given end-effector target. We combine Pseudo-inverse control (PinvC) and model-free reinforcement learning to take the best of both worlds. PinvC already provides many benefits, especially away from constraints, singularities and self-collisions. Then, RL can learn to deal with these and other costs and redundancy. As a result, the RL part would not need to learn everything from scratch and can concentrate on the shortcomings of PinvC. In addition to reaching and pose control, such a controller can naturally be used for inverse-kinematics (IK), alleviating some issues (e.g. joint limits, singularities) of Pseudo-inverse IK.

To utilize both PinvC and RL, we introduce a novel hybrid learning architecture. Integrating the PinvC module into the architecture accelerates and enhances the RL component's learning of reaching, pose control, and IK, compared to pure RL learning. We also provide a minor modification near the target pose, referred to as *switching*, to further improve task success.

Our approach is designed to control the robot's end-effector, navigating from a randomly selected initial robot pose to a randomly selected target pose, including orientation information in large workspaces. To maximize the effectiveness of the RL component, we introduce a novel reward function that accounts for positional error, orientation error, joint velocity norm, and self-collision factors. This comprehensive reward function guides the learning process in reaching, pose control and IK tasks. We also design our state space to represent the task and ease the burden of the RL part. We also show that using the same architecture for various robotic platforms

is possible without changing the hyper-parameters.

The main contributions of the thesis can be summarized as follows:

- A novel state and reward formulation for reinforcement learning of reaching, pose control and IK tasks.

- Combining traditional Pseudo-inverse control with model-free RL, namely PPO [Schulman et al., 2017], to handle large workspaces for the reaching and full-pose control tasks in the joint space.

- Demonstrating a high degree of success, in terms of tight success metrics, for IK and reaching in large workspaces with arbitrary initial robot configuration and target poses for multiple robots without changing any hyper-parameters.

- Demonstrating partial success for full-pose control, a detailed discussion on the issues of incorporating orientation into the target pose and potential remedies which is frequently overlooked in the learning-based manipulator literature.

1.3 Thesis Organization

The rest of this thesis is structured as follows:

- Chapter 2 provides a detailed review of the existing literature on robotic reaching and full-pose control tasks employing DRL. This chapter examines various studies and classifies them based on key features such as the inclusion of orientation in the target pose, including random initial robot configuration and target poses, and self-collision awareness. Moreover, the studies are categorized based on the success rate (SR) threshold and corresponding SR. This chapter also highlights the similarities and differences between our work and the other studies in the literature on the same topic.

- Chapter 3 describes the background that forms the basis of our approach. We describe the Proximal Policy Optimization (PPO) [Kumar et al., 2021] algorithm and the PinvC method.

- Chapter 4 provides an overview of the robotic platforms and the simulation environment utilized in this thesis. It also explains the reaching and full-pose control tasks.

- Chapter 5 explains our hybrid approach for learning reach, pose control and IK tasks. The chapter introduces the RL formulation, which consists of action space, a novel state space and a novel reward function to handle the tasks. Furthermore, it explains the utilization of curriculum scheduling during the training phase. The chapter also explores the possible agents integrated into our architecture. Also, it gives details about the *switching* method to improve the system performance further in various metrics.

- Chapter 6 explains the training details and hyper-parameters. It outlines the details of curriculum scheduling and describes the workspaces utilized for the various robotic platforms. Moreover, it gives verbal and mathematical explanations of the metrics used to evaluate task performance.

- Chapter 7 provides the learning curves for each agent and robotic platform to understand the training process better. The numerical metrics are also shown in this chapter. Additionally, it offers a detailed analysis of the experiments. Lastly, it includes a detailed discussion on the difficulty of full pose control (including orientation targets) with RL.

- Chapter 8 emphasizes the overall results and key findings of learning reaching, pose control, and IK using a hybrid approach. It additionally discusses future research directions.

Chapter 2

LITERATURE REVIEW

Table 2.1 summarises existing studies focusing on the reaching and full-pose control tasks. The reaching task controls the end-effector to a specified 3D position while ignoring orientation. On the other hand, the full-pose control task is responsible for guiding the end-effector to a desired 6D pose. We categorize these methods based on whether they include orientation in the target pose, handle arbitrary initial/target poses, are self-collision aware, and evaluate the same architecture on different robotic platforms. Additionally, we provide details on their SR thresholds, corresponding SR, and workspace volume. The detailed explanation of categorized features is outlined as follows:

- **Orientation:** Studies are categorized based on whether they consider orientation information at the target pose. If the target pose includes orientation information, the objective is to learn the full-pose control task. Otherwise, the focus is on learning the reaching task.

- **Initial&Target:** This feature indicates whether the robot's initial configuration and target pose are randomly sampled from a workspace to perform reaching or full-pose tasks.

- **Self-Collision Awareness:** Studies are classified based on their incorporation of self-collision awareness.

- **# of Robots:** This feature shows whether the methods are evaluated on various robotic platforms. This indicates the generalizability of the proposed approaches.

- **Threshold and Success Rate:** Success rate (SR) thresholds are defined to a specific value to measure task performance. Since most studies do not consider orientation information at the target pose, we report the SR threshold in centimetres *cm* for the reaching task. An episode is considered successful if the robot's end-effector falls within the defined SR threshold.

- **Workspace:** A workspace consists of points the robot can reach. The joint angles are sampled from that workspace for task execution. The Table 2.1 categorises the workspace volume into three parts: small, medium, and large.

$$\text{Workspace} = \begin{cases} \text{Small} & \text{if } 0m^3 \leq \text{Workspace volume} < 0.4m^3 \\ \text{Medium} & \text{if } 0.4m^3 \leq \text{Workspace volume} < 0.8m^3 \\ \text{Large} & \text{if } 0.8m^3 \leq \text{Workspace volume} \end{cases}$$

In Table 2.1, it is clear that most studies fix the robot's initial configuration, ignoring self-collision awareness in reaching or full-pose control tasks. Additionally, most studies only concentrate on target positions, neglecting orientation information at the target. Furthermore, many of these studies evaluate their results using coarse metrics, often limited to a single type of robot. Note that the studies that use the same threshold cannot be compared directly due to the evaluation of different robots and significant differences in the workspace volume. Fixing the end-effector's initial position and ignoring self-collision awareness limits the practical applicability of reaching or full-pose control tasks. The lack of orientation information in the target further restricts the robot's capabilities. Our hybrid approach is designed with large workspaces, arbitrary initial and target configurations, full-pose targets, and self-collision awareness to fill the gaps in the existing literature. Our approach also aims for generality across various robotic platforms without concerns about redundancy, joint limits, or singularities.

For instance, in [Li et al., 2020c], an improved version of model-free DDPG [Lillicrap et al., 2015] is introduced for a robotic reaching task involving a six DoF configuration. The algorithm is trained initially using the first three joints of the Universal Robot UR5 (U_6) robot, followed by employing all six joints. This trained

Table 2.1: Learning-based reaching and full-pose control studies in the literature

Study	**Orientation**	**Init.&Target**	**Self-Collision Awareness**	**# of Robots**	**Threshold**	**Success Rate**	**Workspace**
[Luo et al., 2020]	✓(2D)	✗	✗	1	15cm	Sim:0.98[1] Real:0.98	no details
[Szep et al., 2022]	✗	✗	✗	1	5cm	0.744[2] 0.932	no details
[Gu et al., 2017]	✗	✗	✗	1	10cm	1.0	Small
[Plappert et al., 2018]	✗	✗	✗	1	5cm	1.0	Small
[Chen et al., 2018]	✗	✓	✗	2	2cm	H:0.92[3] E:0.88	Small
[Aumjaud et al., 2020]	✗	✗	✗	1	5cm	0.75	Small
[Li et al., 2020a]	✗	✗	✗	1	5cm	0.95	Small
[Lewis II et al., 2019]	✗	✗	✗	1	10cm	1.0[4] 0.5	Small Large
[Kumar et al., 2021]	✗	✓	?[5]	2	1cm	0.96	Medium
[Gandana et al., 2020]	✗	✗	✗	1	5cm	0.72	Medium
[Lucchi et al., 2020]	✗	✓	✓	1	10cm	Sim:0.96 Real:0.98	Large
[Weber and Schmidt, 2021]	✗	✓	✗	1	10cm	0.994	Large
[Væhrens et al., 2022]	✓	✓	✗	3	✗	footnote[6]	Large
Ours	✓	✓	✓	4	1cm	See Chapter 7	Large

[1]Top is the simulation test results, while the bottom section is the real robot results.

[2]Top trained with TD3 and bottom with SAC.

[3]Top tested with H and bottom on the E robot.

[4]Top result is with small workspace and bottom is with the full workspace.

[5]Addresses obstacle avoidance but the self-collision is unclear due to *collision spheres* placement.

[6]The success rate is nearly 1 for the Panda and Iiwa robots, and approximately 0.75 for UR5.

algorithm improves accuracy and reduces the training episode length. Termination criteria involve ensuring the robot does not collide and the end-effector reaches the target position. The robot tries to reach a random target position from a predefined initial robot configuration. The reward function depends on the positional error and is formulated exponentially. The agent receives a large positive reward when there is no collision and the end-effector reaches the target. In the presence of a collision, the agent receives a large negative number. However, the study lacks a reported SR, orientation information at the target pose, evaluation of different robotic platforms and evaluation within a large workspace.

In [Gu et al., 2017], a strategy is proposed to enhance the efficiency of DRL in complex tasks for real robots. The study employs an asynchronous Normalized Advantage Functions (NAF) algorithm [Gu et al., 2016] for a 7-DoF robotic arm. The reaching task's target is defined by position information, and the robot begins from a predetermined configuration, aiming to reach a randomly selected target position. Random target points are sampled from a $0.2m^3$ cube for simulation and a $0.4m^3$ cube for real test evaluations. The episode is successful if the distance between the end-effector and the target position is within $5cm$ for simulation and $10cm$ for real environments. The SR is determined using 5 test samples. The state space for the RL formulation includes the joint angles, joint velocities, end-effector position, and target position. The SR for the reaching task is 1. However, this study lacks orientation information at the target pose, random initial robot configuration, consideration of self-collision awareness, evaluation of various robotic platforms, evaluation within a large workspace and tight metrics for measuring the task performance.

In [Plappert et al., 2018], a diverse range of robotic control tasks including pushing, sliding, pick and place and reaching tasks are trained using the DDPG+HER algorithm. This study presents accuracy outcomes across various robotic tasks with a 7-DoF robotic arm. Three-dimensional positional coordinates define the target pose. Accuracy results are calculated after each epoch by evaluating ten samples and computing the average value. A distance threshold of $5cm$ defines success. The agent gets 0 reward points if the end-effector is at the target location within $5cm$ and

-1 otherwise. The action space is 4-dimensional, where three dimensions represent the end-effector movement in Cartesian space and the last dimension is responsible for opening and closing the gripper. The reported SR for the reaching task is 1. However, this study lacks orientation information at the target pose, random initial robot configuration, consideration of self-collision awareness, evaluation of various robotic platforms, evaluation within a large workspace and tight metrics for measuring the task performance. Additionally, the study controls the robot end-effector in Cartesian space rather than joint space.

[Mahmood et al., 2018] outlines an effective and reliable setup using DRL to learn a reaching task in a real-world environment. The author presents the setup procedure for a UR5 Reacher and UR5 Reacher 6D robotic arm. The UR5 Reacher only actuates its second and third joints, while UR5 6D actuates all six joints. Random target positions are sampled from a $0.35m^2$ two-dimensional area for UR5 Reacher, $0.14m^3$ volume for UR5 Reacher 6D. The state space for the RL formulation includes the joint angles, joint velocities and the three-dimensional position error between the end-effector and the target position. The reward is defined as the Euclidean distance between the end-effector and target positions, scaled with the action cycle time. However, this study lacks SR results, orientation information at the target pose, random initial robot configuration, consideration of self-collision awareness, evaluation of various robotic platforms, evaluation within a large workspace and tight metrics for measuring the task performance.

In [Lewis II et al., 2019], the policy is trained using DDPG for a reaching task. The state space for the RL formulation includes the joint angles, joint velocities, joint accelerations, target position and the Euclidean error between the end-effector and the target position. Desired joint angles define the action space. The reward function penalizes large actions and the Euclidean distance error between the end-effector and the target position. Additionally, the agent receives a large reward once the robot reaches the target position. The performance of the reaching task is evaluated under two different cases: unconstrained and constrained regions. In the unconstrained version, the robot's initial configuration is fixed at predefined joint angles, while random target positions are sampled from the entire workspace.

A distance threshold of $10cm$ is established to determine success. Training the robot with the first two joints yields a highly successful policy, with a SR close to 1. However, when all joints are included in the training, the SR drops below 0.5. For the constrained version, the target positions are sampled from a $0.288m^3$ volume. In this setup, the agent cannot learn a good policy to reach a target. The target positions are sampled from a $0.128m^3$ volume box for the other constrained version. In this case, UR5 Reacher successfully learns a policy to reach target positions, with the SR converging nearly to 1. However, this study lacks orientation information at the target pose, random initial robot configuration, consideration of self-collision awareness, evaluation of various robotic platforms, evaluation within a large workspace and tight metrics for measuring the task performance.

In [Aumjaud et al., 2020], a comprehensive benchmark of various model-free DRL algorithms is introduced for solving the robotic reaching task using a 6-DoF manipulator. Four distinct environment setups are employed, identified as *Env1*, *Env2*, *Env3*, and *Env4*. The policies are trained using Pybullet [Coumans and Bai, 2021] robotic simulator. The detailed configurations of these environments are outlined as follows:

- **Env1:** Pybullet simulation + fixed target position
- **Env2:** Pybullet simulation + random target position
- **Env3:** Physical robot + fixed target position
- **Env4:** Physical robot + random target position

The state space formulation consists of the position of the robot's end-effector and the joint angles. The action space is defined as the joint angles from the previous time step. The reward function is specified as the negative of the L2 norm of the vector between the end-effector and the target position. A variety of model-free DRL algorithms are employed to make a comprehensive benchmark test for solving the reaching task. All model-free algorithms tested in [Aumjaud et al., 2020] successfully learn how to reach a fixed position in Env1. However, the algorithm performance

and SR experience a significant decline in *Env2* due to the inclusion of random target positions. System performance and SR decrease globally when the learned policies are transferred to a real robot.

[Luo et al., 2020] aims to accelerate the DRL training process while enhancing the reaching task performance using DDPG on the UR5 robot platform. In their setup, the UR5 robot starts from a predetermined initial pose and aims to reach a randomly selected target pose within a defined workspace. To simplify the task, only five joints of the UR5 robot are activated, and the orientation dimension is reduced from 3D to 2D by fixing the Roll X-axis. The state space includes the joint angles, along with the current position and orientation of the end-effector. The action space is defined as the joint velocities. With dense reward settings, the success threshold is set to $15cm$ for position and 0.15 radians for orientation. The study demonstrates that the algorithm achieves a SR of 0.982 in simulation and 0.98 in a real environment. However, the one-dimensional reduction in the orientation space and fixing the initial robot configuration at each episode limit the method's generality. Additionally, this study lacks consideration of self-collision awareness, evaluation of various robotic platforms and tight metrics for measuring the task performance.

In [Szep et al., 2022], a benchmark test is conducted using various model-free DRL algorithms (DDPG, Twin Delayed DDPG (TD3) [Fujimoto et al., 2018] and SAC) on the Neurorobotics Platform. Each experiment's initial robot configuration begins at a zero position. The target positions are randomly sampled within the reachable workspace. An episode is successful if the robot's end-effector reaches the random target position within various thresholds ($0.2m$, $0.15m$, $0.1m$, $0.05m$). The study starts with DDPG, which performs well when the success threshold is set to $15cm$. However, the SR significantly drops when the threshold is set to $5cm$. To address this, the authors continue training the agent using TD3 [Fujimoto et al., 2018] and SAC algorithms. The study conducts various experiments by chaining the number of actuated joints, reward types, and training length. When all joints are actuated, the SR is 0.744 with an average distance error of $4.6cm$ for TD3. Using the SAC algorithm, the SR and average distance error improve to 0.932 and $2.1cm$,

respectively. However, this study lacks orientation information at the target pose, random initial robot configuration, consideration of self-collision awareness, evaluation of various robotic platforms, and tight metrics for measuring task performance.

In [Chen et al., 2018], a novel Hardware Conditioned Policies (HCP) framework for multi-robot transfer learning is proposed. The core concept involves controlling the robot's end-effector based on the current state and hardware properties. Thus, the state space for RL formulation includes joint angles, joint velocities and hardware properties. The study considers robot kinematic (DoF, link length, kinematic structure) and dynamics (joint damping, friction, armature and link mass) characteristics as hardware properties. The action space is defined as the torque values over all joints. Nine distinct robot types (A, B, ..., I) are designed to have varied kinematic and dynamic structures using the Sawyer robot in Mujoco [Todorov et al., 2012]. These robots have different link lengths, kinematics and dynamics. Among them, the first four are 5-DoF, the following four are 6-DoF, and the last is 7-DoF. The reaching task is defined as moving from a random initial end-effector position to a random target position. The volume of the initial robot's configuration workspace and the target workspace are $0.024m^3$ and $0.072m^3$, respectively. Upon training the various robot types (A-G and I) and testing on the H robot type, the SR is reported as 0.925. Similarly, after training on the various robot types (A-D and F-I) and testing on the E robot type, the SR is reported at 0.88. However, this study lacks orientation information at the target pose, consideration of self-collision awareness and evaluation within a large workspace.

[Lucchi et al., 2020] focuses on transferring a learned model from a simulation environment to a real robot, emphasizing self-collision avoidance. They aimed to reduce the gap between simulation and reality. The experiments are conducted on a UR10 reaching task utilizing Distributed Distributional Deep Deterministic Policy Gradients (D4PG) [Barth-Maron et al., 2018]. The study incorporates a random initial robot configuration and random target position for the reaching task in a large workspace. The authors establish a success threshold of $10cm$ to determine success. The real-world experiments closely match those achieved in the simulation environment, with SR of 0.98 and 0.96, respectively. However, their threshold is

impractical, and their targets lack orientation. They also omit singularities in the workspace, which simplifies the task of reaching. The study lacks an evaluation of various robotic platforms.

In [Weber and Schmidt, 2021], the authors employ the DDPG algorithm to learn reaching task from user-selectable starting points and random target points within a notably large workspace. An essential contribution of their work is the introduction of a novel state space representation for RL formulation to enhance task performance. The proposed state space includes joint angles, the Euclidean distance between the end-effector and the target position, a binary success that indicates whether the target is reached, and the transformed target position to each link in the base frame. The action space is defined as the joint velocities. The reward function consists of three terms. The first term penalizes the agent based on the distance between the end-effector and the target position. The second term penalizes the agent when the robot has high joint velocities near the target. The last term penalizes the agent for exceeding mechanical joint angle limits. Using their novel state space, their approach achieves a 0.994 SR, albeit with a $10cm$ threshold and an average distance error of $1.5cm$. However, the study lacks orientation information at the target pose, consideration of self-collision awareness, and evaluation of various robotic platforms. Moreover, the authors only compare their introduced novel state space representation to other representations.

[Væhrens et al., 2022] adopts the methodology (PPO and CL) and simulation environment (including obstacles) used in [Kumar et al., 2021]. The primary aim of [Væhrens et al., 2022] is to extend [Kumar et al., 2021] by incorporating orientation information into the target pose. The state space consists of the joint angles, joint velocities, end-effector position, target position, the distance from the end-effector to obstacles in the environment, and the distance from the end-effector to the target position. The reward function is composed of several terms. The first term is the distance reward between the end-effector and the target position. The agent receives a high reward as it approaches the target. The second term introduces a negative reward when the robot approaches obstacles in the environment. Additionally, the agent receives a large reward of 250 when it successfully reaches the target. Exper-

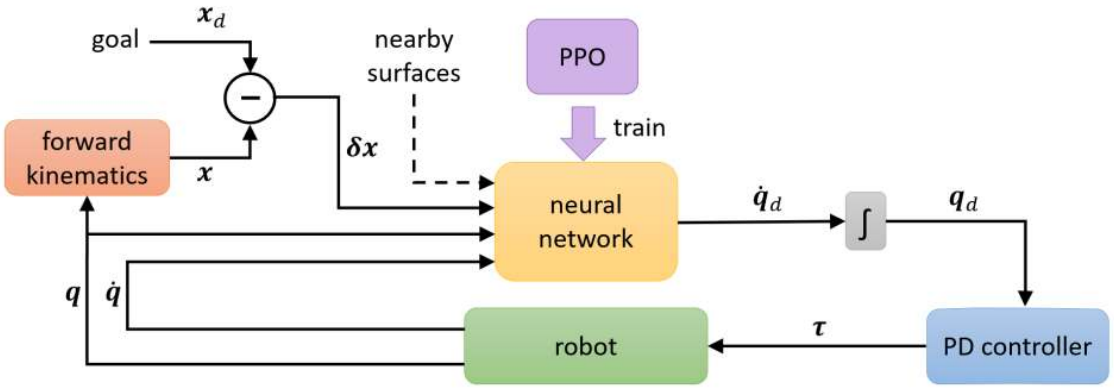


Figure 2.1: The RL arhitecture is taken from [Kumar et al., 2021] for learning reaching task using PPO model-free algorithm. Our work closely aligns with this approach in using PPO and CL. The RL architecture takes joint angles q, joint velocities $\dot{q}$, and positional error δx as inputs.Additionally, obstacles in the environment *nearby surfaces* can be integrated to the system optionally.

iments are conducted using three distinct robots: KUKA LBR Iiwa (Iiwa), Panda, and UR5. The results indicate that UR5 struggles to learn reaching and full-pose control tasks, while Iiwa and Panda learn the reaching and full-pose tasks. However, it is important to note that the study lacks a specified SR threshold, consideration of self-collision awareness, and a comparative analysis with other reaching and full-pose control algorithms. Additionally, the workspace used in their experiments does not encompass the boundary points of the robot.

[Kumar et al., 2021] employs PPO and CL to achieve accurate and smooth manipulator control. The state space in their work consists of the joint angle, joint velocities, Euclidean distance between the end-effector and the target position and the smallest vector between each link and the obstacles in the environment. The action space is defined as the joint velocities, and the reward term consists of three components. The first term encourages the end-effector to reach the target position. The second term penalizes the high joint acceleration. The third term penalizes the agent when the robot approaches an obstacle in the environment. The authors illustrate the proposed system in Figure 2.1. Their method handles arbitrary initial robot configuration and target positions. They present a remarkable SR, 0.96, with a tight $1cm$ threshold and very low distance error of $0.4cm$, comparable to OSC

baseline results. However, this study lacks orientation information at the target pose, consideration of self-collision awareness, evaluation of various robotic platforms and evaluation within a large workspace.

We incorporated the first reward term for position proposed by [Kumar et al., 2021] into our work, where the author sets λ_{pos} to 20. Additionally, we adopted the concept proposed by [Kumar et al., 2021] of minimizing the joint acceleration norm and applied it to minimize the joint velocity norm in our work. Our work is similar to [Kumar et al., 2021]'s work in using PPO and CL but with a larger workspace, orientation results, expanded state space (minus object collision avoidance) and a different reward function. Our results presented in Chapter 7 shed light on why they and the other authors may have omitted orientation altogether.

Chapter 3

PRELIMINARIES

3.1 Reinforcement Learning Basics

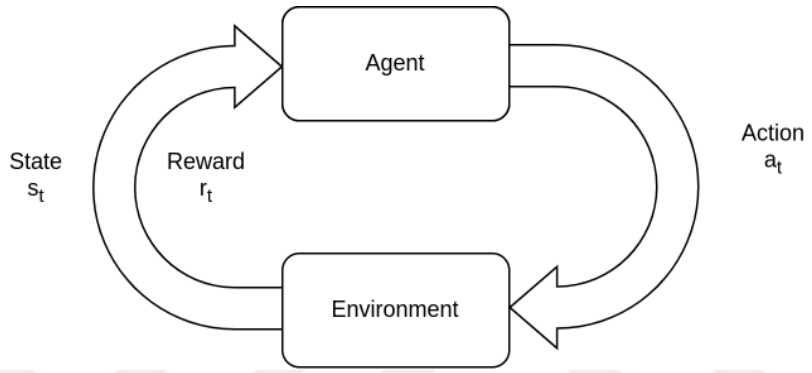


Figure 3.1: Basic representation of RL.

In reinforcement learning, an agent learns to make decisions by interacting with an environment. The agent perceives the environment state and takes an action as illustrated in Figure 3.1. After each action, the agent observes the environment, updates its state and gets/calculates its reward. The main goal of RL is to figure out a mapping from states to actions in an environment such that it maximizes its expected (discounted) cumulative reward over time. It differs from supervised learning, where the learning relies on sets of labelled data provided by a supervisor. In RL, the agent learns to act optimally by interacting within the environment without needing a supervisor. The elements of RL can be listed as follows:

- **Agent:** The agent is a system that makes decisions and takes actions in the environment. The primary goal of the agent is to maximize the expected cumulative reward.

- **State:** An environment or the agent's observations may contain too much and

potentially unnecessary information. A state represents the current situation of the environment relevant to the task at hand. The state should informative for decision-making while being as compact as possible.

- **Action:** Actions are based on the agent and the environment. The agent interacts with the environment by taking actions.

- **Reward:** Reward is a scalar quantity that scores the importance of taking action for a given state. It describes the goal(s) of a task.

- **Policy:** A policy stands as a core element in RL algorithms, often referred to as the brain of a RL agent. It functions as a mapping from states to actions. The policy may be *deterministic* policy where the policy π is a mapping from the current state s_t to action a_t. This *deterministic* policy chooses specific action a_t for a given state s_t and is depicted as:

$$a_t = \pi(s_t) \tag{3.1}$$

  The policy may also be *stochastic*. Such a policy defines a distribution over actions given a state s_t and is depicted as:

$$a_t \sim \pi(. \mid s_t) \tag{3.2}$$

- **Value/Q-Value/Advantage:** Value is the long term utility of a state, Q-Value (or action/state-action value) is the long term utility of a state-action pair and Advantage is the long term utility difference of the value and the Q-value of an action for a given state. These values represent the expected cumulative reward of a given policy.

In this thesis, we focus on online RL and learning a policy which fits our tasks and environment. Note that there are other RL flavors such as offline RL where we do not have access to the environment during learning, model-based RL where we also learn a transition model and use it to update and/or regularize the policy, inverse RL where we learn a reward function from observed trajectories, and value

learning where we only learn the long term utilities of states for example to be used in planning. There are also certain problems with multi-dimensional rewards where the policy learning is guided by pareto optimality.

3.2 Policy Gradient Theorem

The Policy Gradient Theorem provides a framework for optimizing policy parameters for maximizing cumulative rewards. The policy parameters are adjusted for a better decision-making process within an environment. The policy parameters are updated in the direction that increases the cumulative reward. Policy gradient theorem[1] assumes a *stochastic* policy, denoted as $\pi(a_t|s_t, \theta)$, which represents the probability of taking action a_t at state s_t given the parameter vector θ. Let the policy be parameterized by θ. The general equation summarizing the parameter update is as follows: [Sutton and Barto, 2018]

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta_t} \hat{J}(\theta_t) \tag{3.3}$$

where, α denotes the learning rate, θ_t represents the policy parameters at time t, ∇ is the gradient operator with respect to the polixy parameters θ_t, and $\hat{J}(\theta_t)$ is the objective function. This equation iteratively updates the policy parameters to enhance performance by maximizing the objective function.

The following steps outline the derivation of the Policy Gradient Theorem for discrete actions (continuous version is analogous and summation is replaced with integration), as referenced by [Sutton and Barto, 2018]. The primary objective is to maximize state-values through policy learning. To initiate the proof, we begin by calculating the derivative of the state value of the policy with respect to the policy parameters. Note that the value function $v_\pi(s)$ of a *stochastic* policy is the sum of the policy q-values $q_\pi(s, a)$ weighted by action probabilities $\pi_\theta(a \mid s)$. The q-values serve as a representation of the value of taking action a in state s. The gradient of the state-value function with respect to the policy parameters can be expressed as:

[1]There is also a deterministic policy gradient (DPG) theorem [Silver et al., 2014].

$$\nabla_\theta v_\pi(s) = \nabla_\theta \left[\sum_a \pi_\theta(a \mid s) q_\pi(s,a) \right], \quad \text{for all } s \in \mathcal{S} \tag{3.4}$$

Given the multiplication structure of the above equation, it is possible to rearrange the notation for summation and apply the product rule of calculus.

$$\nabla_\theta v_\pi(s) = \sum_a \left[\nabla \pi_\theta(a \mid s) q_\pi(s,a) + \pi_\theta(a \mid s) \nabla q_\pi(s,a) \right] \tag{3.5}$$

At this point, we can substitute the action-value function at the last term with a transition probabilities $p\left(s', r \mid s, a\right)$, reward r and the state-value function of the next state $(r + v_\pi(s')$.

$$\sum_a \left[\nabla_\theta \pi_\theta(a \mid s) q_\pi(s,a) + \pi_\theta(a \mid s) \nabla_\theta \sum_{s',r} p\left(s', r \mid s, a\right) \left(r + v_\pi\left(s'\right)\right) \right] \tag{3.6}$$

It is possible to omit the reward term in the equation since the reward does not depend on the policy parameters θ.

$$\nabla_\theta v_\pi(s) = \sum_a \left[\nabla_\theta \pi_\theta(a \mid s) q_\pi(s,a) + \pi_\theta(a \mid s) \sum_{s'} p\left(s' \mid s, a\right) \nabla_\theta v_\pi\left(s'\right) \right] \tag{3.7}$$

Upon looking at the equation above, a recursive pattern becomes clear. This pattern means that the differentiation of the state-value function relies on the differentiation of the following state-value function, both described under policy π_θ. After a series of mathematical manipulations, the resulting equation takes the following form:

$$\nabla_\theta v_\pi(s) = \sum_{x \in S} \sum_{k=0}^{\infty} \Pr(s \to x, k, \pi) \sum_a \nabla \pi(a \mid x) q_\pi(x,a) \tag{3.8}$$

The variable k denotes the time step. At this point, we can move towards concluding the derivation. In the context of episodic scenarios, the objective function can be defined as the value of the initial start state s_0 of each episode.

$$\begin{aligned} \nabla \hat{J}(\theta_t) &= \nabla v_\pi\left(s_0\right) \\ &= \sum_s \left(\sum_{k=0}^{\infty} \Pr\left(s_0 \to s, k, \pi\right) \right) \sum_a \nabla \pi(a \mid s) q_\pi(s,a) \\ &\propto \sum_s \mu(s) \sum_a \nabla \pi(a \mid s) q_\pi(s,a) \end{aligned} \tag{3.9}$$

Here, $\mu(s)$ is the normalized state distribution probabilities starting from state s_0, affected by the transition probabilities. Thus, it is possible to express the equation 3.9 in expectation form as follows:

$$\nabla \hat{J}(\theta_t) = \mathbb{E}_\pi \left[\sum_a q_\pi \left(S_t, a\right) \nabla \pi \left(a \mid S_t, \boldsymbol{\theta}\right) \right] \tag{3.10}$$

Given that the above equation includes a summation within an expectation, we multiply and divide the inner term by $\pi \left(a \mid S_t, \boldsymbol{\theta}\right)$ while maintaining equality. This manipulation eliminates the summation over all actions by substituting A_t instead of a. Furthermore, we can replace the action-state value with the return G_t. The final derivative of the objective function equation can be formulated as follows:

$$\nabla \hat{J}(\theta_t) = \mathbf{E}_\pi \left[G_t \nabla \log \pi \left(A_t \mid S_t\right) \right] \tag{3.11}$$

At this point, we have a tool for optimizing the policy parameters and it is known as Vanilla Policy Gradient (VPG). This equation can be used in Equation 3.3 to iteratively update the policy parameters.

3.3 PPO

The VPG approach described in equation 3.11 has some drawbacks. VPG often suffers high variance problems in its gradient estimates due to the multiplication of rewards. This can lead to differences in actions produced by the policy from one episode to another. This variance problem leads to noisy and problematic convergence, resulting in non-monotonic performance and stability issues. To address these challenges, researchers have developed several variations of the VPG algorithm, such as Asynchronous Advantage Actor-Critic (A3C) [Mnih et al., 2016], Generalized Advantage Estimation (GAE) [Schulman et al., 2015b], Trust Region Policy Optimization (TRPO) [Schulman et al., 2015a], and PPO [Schulman et al., 2017]. These algorithms aim to achieve efficient, stable and steady improvements. For our experiments, we employed PPO due to its proven success in learning the reaching task [Kumar et al., 2021].

The objective function presented in equation 3.12 utilizes the ratio of a new policy to an old policy and the advantage function within an expectation $\hat{E}_t[.]$. This specific objective function is referred to as the "surrogate" function. [Schulman et al., 2015a] maximizes the following objective function subject to a constraint to avoid large policy update.

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta \left(a_t \mid s_t \right)}{\pi_{\theta_{\text{old}}} \left(a_t \mid s_t \right)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right] \tag{3.12}$$

Where $r_t(\theta)$ is ratio of probabilities of taking an action a_t given the state s_t under the current policy $\pi_\theta(a_t \mid s_t)$ to the old policy $\pi_{\theta_{old}}(a_t \mid s_t)$. This ratio serves as a measure of the distance between the two policies. Additionally, $\hat{A}_t$ represents the advantage function, which assesses how advantageous it is to be in the particular state s_t compared to the predicted average value of being in that state. The advantage function $\hat{A}_t$ is calculated using the following expression.

$$\hat{A}_t = V_t^{\text{target}} - V_{\text{old}}(s_t) \tag{3.13}$$

where, $V_{\text{old}}(s_t)$ is the value of being in the state s_t predicted by the old value network. Finally, V_t^{target} is calculated as follows:[Bick and Wiering, 2021]

$$V_t^{\text{target}} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n V_{\text{old}}(s_{t+n}) \tag{3.14}$$

where $\gamma \in (0, 1]$ denotes the discount factor, and r_t represents the earned reward during the transition from state s_t to s_{t+1}.

However, using only the equation 3.12 to update the policy could lead to extreme policy updates, causing unstable training. In addition, VPG, as its name suggests, uses a vanilla gradient which is affected by how the policy is parameterized. Natural gradients break this dependency. [Schulman et al., 2017] suggests a clipped version to prevent this issue and ensure more stable updates while also approximating the natural gradient as follows:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right] \tag{3.15}$$

In Equation 3.15, the first term, $r_t(\theta)\hat{A}_t$, represents the unclipped objective, while the second term, $\text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t$, corresponds to the clipped objective. The clipped component modifies the entire objective function by constraining

the probability ratio $r_t(\theta)$ to fall within the range of $1+\epsilon$ and $1-\epsilon$. Here, ϵ is the emprical value to limit the policy update and generally it is set to 0.2 as suggested in [Schulman et al., 2017]. Finally, the minimum expression outputs the minimum value between the unclipped and clipped objective to make a boundary for policy updates. This approach ensures that policy updates remain within reasonable bounds, preventing extreme updates.

Table 3.1 illustrates the different scenarios encountered in the primary objective function $L^{\text{CLIP}}(\theta)$, as detailed in [Bick and Wiering, 2021]. The sign of the objective function depends on the sign of the objective output presented in the fourth column. As the probability ratios $r_t(\theta)$, $1-\epsilon$ and $1+\epsilon$, are positive, the objective function shares the same sign as the advantage function across all cases.

In the first two cases, the clipped objective function is not applied since the probability ratio falls within the range $[1-\epsilon, 1+\epsilon]$, resulting in the objective function output being $r_t(\theta)A_t$. A positive advantage function indicates that taking action a_t in state s_t is better than the average expectation. In this scenario, the policy is adjusted to increase the probability of taking that action in the given state, driven by the positive advantage function and the presence of an acceptable probability ratio. When the advantage is negative, it implies that taking the action is worse than the average expectation. In such instances, the policy is modified to decrease the probability of taking that action given the state. The presence of the gradients can be observed in Figure 3.2 within the interval where the probability ratio is between $[1-\epsilon, 1+\epsilon]$.

In the case where the probability ratio is less than $1-\epsilon$, it indicates that the probability of taking an action given the state under the current policy is lower than under the old policy. When the advantage function is positive, the unclipped part of the objective function takes the minimum value, resulting in the objective output becoming $r_t(\theta)A_t$ with a positive sign. As a result, clipping is not employed. In this circumstance, the policy is updated to maximize the probability of taking that action because the advantage function is positive, and the probability ratio is less than $1-\epsilon$. However, when the advantage function is negative, the clipped part of the objective function is utilized to ensure that the gradient is zero. This

Case	$r_t(\theta) > 0$	A_t	**Objective output**	**Clipped objective**	**Sign of objective**	**Gradient**
1	$r_t(\theta) \in [1-\epsilon, 1+\epsilon]$	+	$r_t(\theta)A_t$	✗	+	✓
2	$r_t(\theta) \in [1-\epsilon, 1+\epsilon]$	-	$r_t(\theta)A_t$	✗	−	✓
3	$r_t(\theta) < 1-\epsilon$	+	$r_t(\theta)A_t$	✗	+	✓
4	$r_t(\theta) < 1-\epsilon$	-	$(1-\epsilon)A_t$	✓	−	0
5	$r_t(\theta) > 1+\epsilon$	+	$(1+\epsilon)A_t$	✓	+	0
6	$r_t(\theta) > 1+\epsilon$	-	$r_t(\theta)A_t$	✗	−	✓

Table 3.1: The Table presents various cases to understand the behavior of the objective function $L^{\text{CLIP}}(\theta)$. This Table is taken from [Bick and Wiering, 2021]. And the Table explores the output and gradient of the objective function under different probability ratio ranges and advantage function signs. The first column enumerates cases from 1 to 6. The second column displays the range for the probability ratio of the new policy $\pi_\theta(a_t \mid s_t)$ to the old policy $\pi_{\theta_{\text{old}}}(a_t \mid s_t)$. The third column denotes the sign of the advantage function for a given training example. The fourth column is the output of $L^{CLIP}(\theta)$, as described in equation 3.15 for a given probability ratio and advantage sign. The fifth column indicates whether the clipped function in the main objective function is utilized. Finally, the last column illustrates the presence of the gradient of the clipped objective function $L^{\text{CLIP}}(\theta)$.

approach prevents further reduction in the probability of taking that action. The gradients within the specific region where the probability ratio is less than $1-\epsilon$ can be observed in Figure 3.2.

In the last two cases where the probability ratio is greater than $1+\epsilon$, showing that the probability of taking an action given the state under the current policy is lower than that under the old policy. Clipping is employed in this case because the terms in the clipping part take the minimum value due to the positive sign of the advantage function. So, the objective output becomes $(1+\epsilon)A_t$ after clipping. There is no need to update the current policy for case 5 since the current policy already has a high probability of taking an action given a state, coupled with a positive advantage. The main objective function avoids to update the policy, resulting in a zero gradient in this case. However, for the last case, the policy is updated to decrease the probability of taking an action given state due to the negative advantage. The gradients within

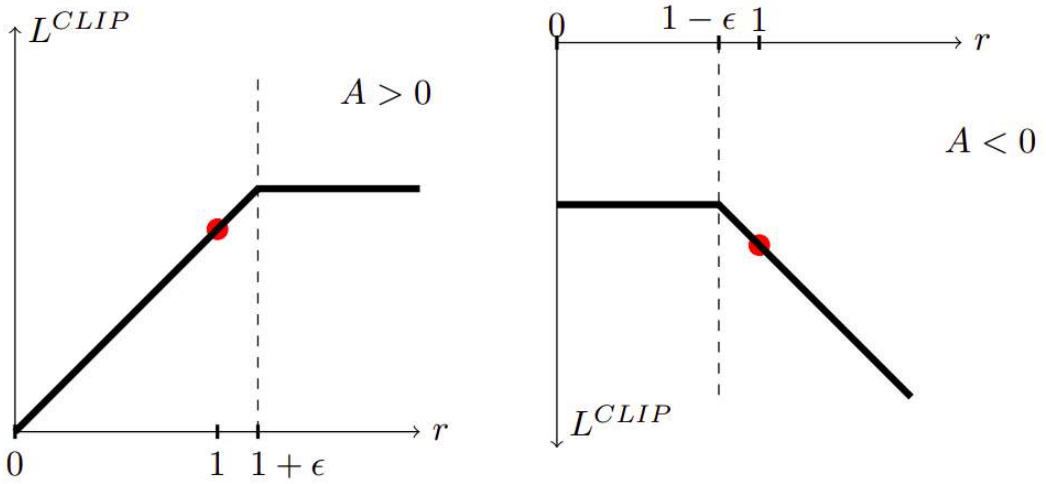


Figure 3.2: Plots illustrating the objective function $L^{\text{CLIP}}(\theta)$ in a piece-wise form to provide insight into its behavior. The plots are taken from [Schulman et al., 2017]. The plots are drawn as a function of the probability ratio $r_t(\theta)$. The red dots represent special points where the probabilities of taking an action for a given state under the current policy and the old policy are equal. ϵ is a hyper-parameter and generally it is set to 0.2. These plots offer valuable observation of the objective function and the gradient at specific regions.

the specific region where the probability ratio is less than $1 + \epsilon$ can be observed in Figure 3.2.

In addition to the primary objective function $L^{CLIP}(\theta)$, as presented in Equation 3.15, there exists an objective function for the value state network. The following objective function, expressed in expectation form, aims to minimize the squared error for the value state network. The loss function for value state network is defined as:[Schulman et al., 2017]

$$L^{VF}(\theta) = (V_\theta(s_t) - V_t^{\text{target}})^2 \tag{3.16}$$

where, $V_\theta(s_t)$ is the value state network and V_t^{target} target state value defined in equation 3.14. Since we have two objective function, the following combined objective function with the entropy loss can be used to update the networks parameters.

$$L_t^{CLIP+VF+S} = \hat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + s_2 S[\pi_\theta(s_t)]] \tag{3.17}$$

where c_1, c_2 are coefficients, and S denotes entropy bonus. The entropy bonus makes the policy to be more exploratory to discover new actions.

Algorithm 1 Pseudo-code of PPO with Clipped Objective

Input: Randomly initialize policy and value network with shared weights θ
for $k = 0, 1, 2, \ldots$ **do**
    number_steps = 0
    Reset roll-out buffer RB
    while number_steps < max_roll-out_steps **do**
        Calculate (s_t, a_t, r_t, s_{t+1}) under the current policy $\pi(\theta)$ and add to RB
        Compute advantages A_t based on GAE
        number_steps $\leftarrow$ number_steps + 1
    end while
    for epoch $\in$ range(number_epochs) **do**
        for roll_out data $\in RB$ **do**
            Calculate the probability ratio: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
            Calculate the clipped surrogate loss:
            $L^{CLIP}(\theta) = \frac{1}{N}\sum_{t=1}^{N} \min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)$
            Calculate value loss
            $L^{VF}(\theta) = \frac{1}{N}\sum_{t=1}^{N}\left(V_\theta(s_t) - (V_t^{target})^2\right)$
            Compute entropy loss
            $L^S(\theta) = \frac{1}{N}\sum_{t=1}^{N} \pi_\theta(a_t \mid s_t) log(\pi_\theta(a_t \mid s_t))$
            Compute the total loss
            $L^{CLIP+VF+S}(\theta) = L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S(\pi_\theta(s_t))$
            Update the policy parameters by maximizing $L^{CLIP+VF+S}(\theta)$
            Update the policy by maximizing the above objective function with stochastic gradient ascent
        end for
    end for
end for

The pseudo-code of training the agent using PPO with clipped objective can be seen in Algorithm 1. The algorithm begins by initializing both the policy and velue network with shared parameters θ. During the simulation, experiences are

collected by interacting with the environment under the current policy $\pi(\theta)$ and the advantages A_t are computed using the GAE. After that the policy is updated by maximizing the total objective function for a specified number of epochs. This update involves calculating the probability ratio $r_t(\theta)$, clipped surrogate loss $L_t^{CLIP}(\theta)$, value network loss $L_t^{VF}(\theta)$ and entropy loss $L_t^S(\theta)$. The formulation for calculating the entropy loss is taken from [Bick and Wiering, 2021].After that all computed loss values are added to have total loss to update the policy parameter with stochastic gradient ascent. And finally, the new parameters are assigned to policy and value network.

3.4 Inverse Kinematics and Pseudo-inverse

FK describes the task-space configuration (e.g. end-effector pose) for a given joint configuration of a robot. For a vast majority of robot tasks, the inverse relationship is desired; calculating the joint configuration of a robot given a desired task-space configuration. This process known as IK.

Calculating IK is usually not straightforward. For open-link kinematic chains, such as robot manipulators used in this thesis, FK has a straightforward closed-form. However, achieving a closed-form solution for IK can be challenging and, frequently, impossible. This is especially true for redundant manipulators where the DoF of the robot is greater than the task space dimensionality. For example, the full end-effector pose space is six-dimensional and any manipulator with more than six joints is considered redundant[2]. This leads to infinitely many solutions.

A common way to calculate IK is to work in the velocity space by taking the derivative of the FK equation with respect to time. When we take the derivative of the FK equation, the Jacobian matrix J establishes a linear relationship between the joint velocity $\dot{q}$ and the end-effector velocity vector $v_e = [\dot{p}_e^T, \omega_e^T]^T$, where $\dot{p}$ is the translational velocity and ω is the rotational velocity[3].

$$\boldsymbol{v}_e = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{3.18}$$

[2]We omit the case when the joint space dimensionality is less than the task space dimensionality.

[3]For the rest of this chapter, we are assuming that the task space is the full-pose space.

For redundant manipulators, the Jacobian matrix is rectangular, with unequal rows and columns. Thus, the it can not be directly inverted to determine joint velocities [Spong and Vidyasagar, 2008]. To address this issue, Pseudo-inverse solutions have been introduced to handle non-square Jacobian matrices. The Pseudo-inverse methods provide an approximate solution for obtaining desired joint velocities when no unique solution exists.

3.4.1 Right Pseudo-Inverse Solution

As previously described, a robot manipulator is said to be redundant when the DOF of the robot is greater than the required task DOF. Thus, the Jacobian matrix becomes a rectangular matrix where the number of columns exceeds the number of rows, resulting in infinitely many solutions. To adress this complexity, a constrained linear optimization problem is constructed as stated in [Bruno et al., 1994]. This involves constructing a quadratic cost function with Lagrange multipliers as follows:

$$g(\dot{\boldsymbol{q}}, \boldsymbol{\lambda}) = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{W}\dot{\boldsymbol{q}} + \boldsymbol{\lambda}^T (\boldsymbol{v}_e - \boldsymbol{J}\dot{\boldsymbol{q}}) \tag{3.19}$$

where $\dot{\boldsymbol{q}}^T$ represents the transpose of the joint velocities, W is the weight matrix and λ^T is the transpose of the Lagrange multiplier. The objective is to minimize both the error and the norm of joint velocities. This is achieved by computing the gradient of the quadratic cost function with respect to the joint velocities $\dot{q}$ and Lagrange multiplier λ and setting them to zero.

$$\left(\frac{\partial g}{\partial \dot{\boldsymbol{q}}}\right)^T = \mathbf{0}, \quad \left(\frac{\partial g}{\partial \boldsymbol{\lambda}}\right)^T = \mathbf{0},$$

After solving the quadratic constrained problem above, assuming that J has full rank, the solution for joint velocities with weighting matrix W can be determined as:

$$\dot{\boldsymbol{q}} = \boldsymbol{W}^{-1}\boldsymbol{J}^T \left(\boldsymbol{J}\boldsymbol{W}^{-1}\boldsymbol{J}^T\right)^{-1} \boldsymbol{v}_e \tag{3.20}$$

When the weight matrix W is set as the Identity matrix, the solution in Equation 3.20 is referred to as the right Pseudo-inverse solution, which minimizes the norm of joint velocities.

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^T \left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1} \boldsymbol{v}_e = \boldsymbol{J}^{\dagger}\boldsymbol{v}_e \tag{3.21}$$

3.4.2 Damped Least Square Solution

The right Pseudo-inverse solution presented in Equation 3.21 is applicable only when the Jacobian has full rank. This equation becomes unstable when the manipulator is near a singularity or at a singularity. In such scenarios, the calculated joint velocities can become extremely large (infinite in the case of a singularity), leading to failure. The damped-least-squares inverse is introduced to address the singularity concern. The modified version of equation 3.21 can be reformulated as:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^T \left(\boldsymbol{J}\boldsymbol{J}^T + k^2 \boldsymbol{I} \right)^{-1} \boldsymbol{v}_e = \boldsymbol{J}^* \boldsymbol{v}_e \tag{3.22}$$

where, k is the damping factor and it helps to regularize the problem and improve the stability of the soution.The computation of joint velocities for a given end-effector velocity is possible in equation 3.22 even when the manipulator is near a singularity or at a singularity.

Chapter 4

ROBOTS, SIMULATION ENVIRONMENT AND TASKS

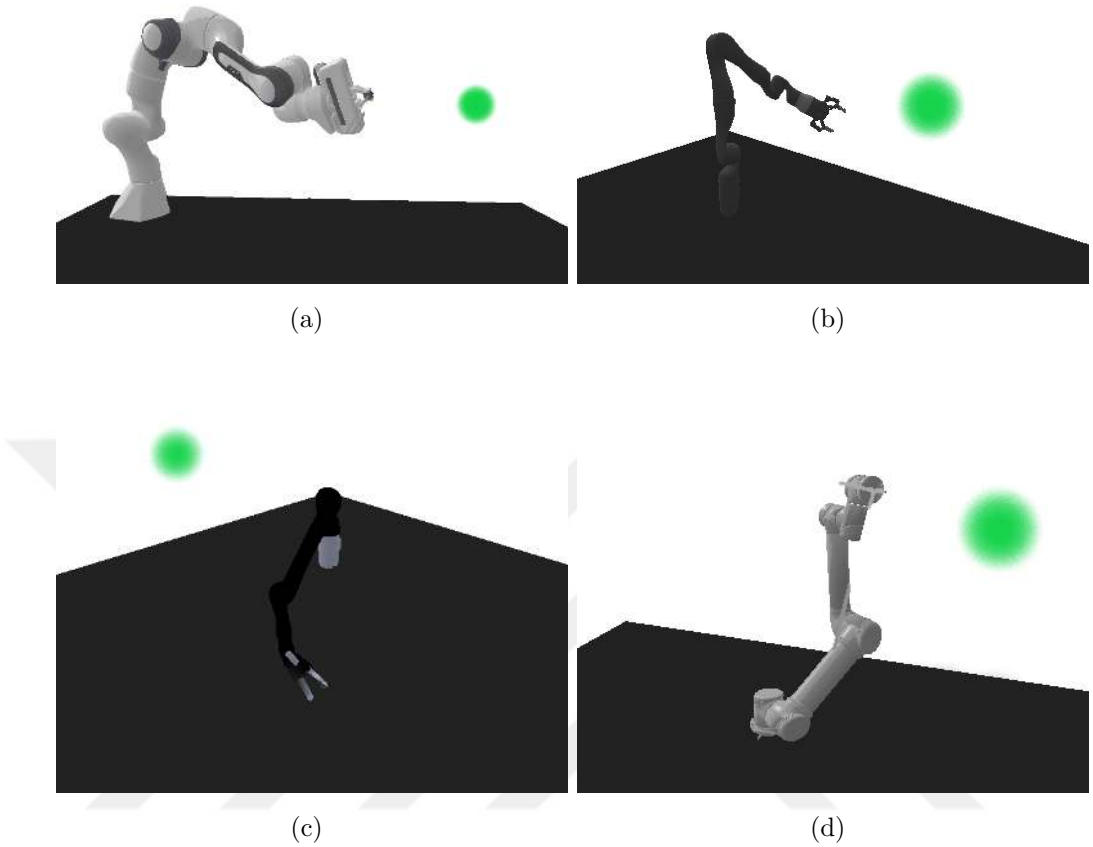
One of the contributions of this thesis is the training of a hybrid model with constant hyper-parameters across various robotic platforms. Two distinct 7-DoF robotic manipulators, namely Panda 7-DoF and Kinova Jaco 7-DoF (J_7), as well as two 6-DoF manipulators, namely UR5 6-DoF and Jaco 6-DoF (J_6), were employed for this purpose. In Figure 4.1 and 4.2, each type of robot is illustrated performing the reaching and full-pose control tasks, respectively.

The training and evaluation processes were employed using the PyBullet simulation engine [Coumans and Bai, 2021]. Additionally, creating environments for reaching and full-pose control for controlling the robot in joint space was designed through the panda-gym framework [Gallouédec et al., 2021]. The tasks employed in this thesis are listed below:

- **Reaching:** In this task, the main objective is to move the robot's end-effector to a target. Here, the target only contains 3D position information. The robot's initial configuration and target positions are randomly sampled within a workspace.

- **Full-Pose:** Similar to the Reaching task, the robot's end-effector is required to reach a predefined target. However, in this case, the target pose includes additional orientation information along with the 3D position.

- **Inverse-Kinematics:** This is almost the same as the Full-Pose task, but it does not prioritize the robot's trajectory. This task involves the iterative integration of joint velocities to compute joint configurations instead of sending the velocities to the robot. As such, collision handling in the intermediate steps is not a must.

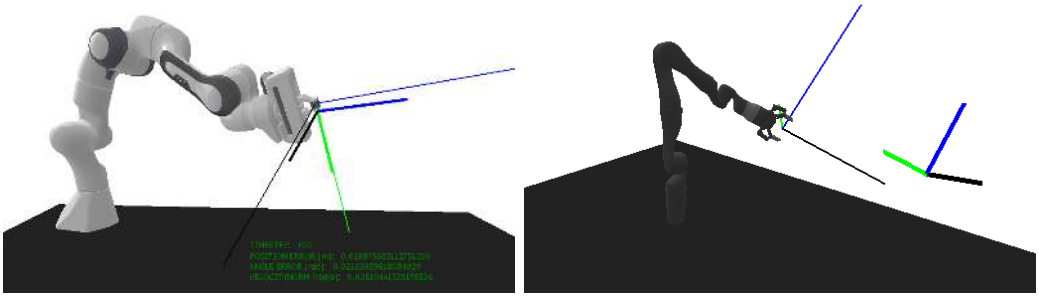


(a) (b)

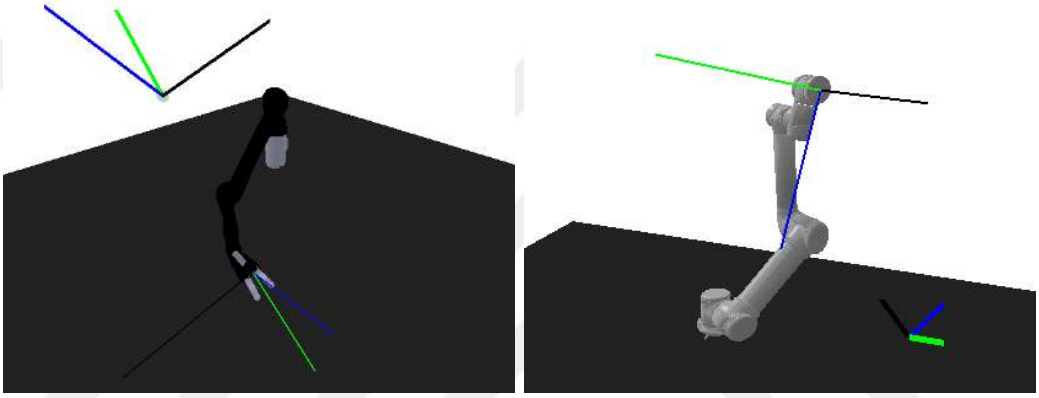
(c) (d)

Figure 4.1: Visualization of Reaching task on: (a) Panda, (b) Jaco 7-DoF, (c) Jaco 6-DoF, (d) UR5. The size of the green circle does not convey any specific information. The green circle represents the target position that the robots aim to reach. The initial robot configurations and target positions are randomly sampled within a defined workspace. The details of workspace creation can be found in Chapter 6.3.

Panda-gym utilizes the Panda 7-DoF robotic manipulator to accomplish the tasks mentioned. Our study expanded this framework by incorporating three additional robotic platforms, as illustrated in Figures 4.1 and 4.2.



(a) (b)



(c) (d)

Figure 4.2: Visualization of Full-Pose control task on: (a) Panda, (b) Jaco 7-DOF, (c) Jaco 6-DOF, (d) UR5. In the full-pose control task, random target poses are selected within the robot's workspace. Each pose includes three-dimensional position and orientation information. The robot's objective is to reach the target position and align its end-effector axis with the target axis.

.

Chapter 5

METHODOLOGY

5.1 Proposed Method

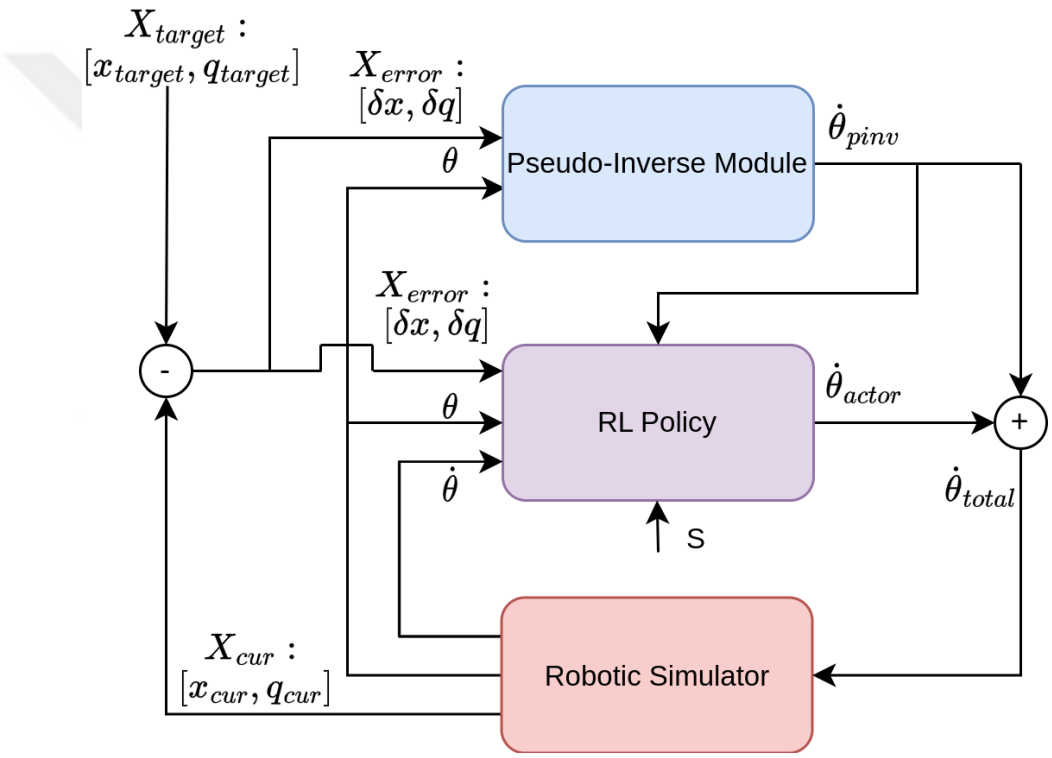


Figure 5.1: The overall architecture for learning the reaching, full-pose control and IK tasks. It consists of two main modules: the Pseudo-Inverse module and the RL Policy module. Both modules are responsible for calculating joint velocities. Subsequently, these joint velocities are summed up and sent to the robotic simulator.

We introduce a novel hybrid approach for learning reaching and posing by combining Pseudo-inverse control and model-free reinforcement learning. Our main aims are to handle large workspaces and to involve arbitrary initial and target poses. In addition, our method can work with multiple robots, only requiring a kinematic description and, if desired, the curriculum schedule. Our system was designed to op-

erate robotic systems by providing a Unified Robotics Description Format (URDF) file and a defined workspace.

The inference flow-chart of our approach is presented in Figure 5.1. The training version is similar but includes reward calculation, curriculum scheduling, rollouts and model updates. Our approach takes the desired target configuration (X_{target}), configuration difference between the current configuration (X_{cur}) and the target (X_{error}), and current joint positions (θ) and velocities ($\dot{\theta}$) as input. The RL part additionally utilizes the joint velocities calculated by the Pseudo-inverse ($\dot{\theta}_{pinv}$) and a binary success flag (S) and outputs *corrective* joint velocities ($\dot{\theta}_{actor}$). Finally, the joint velocities are added ($\dot{\theta}_{total}$) and sent to the robot.

We represent the configurations as $X = [x, q]^T$, where x is the position represented as a 3D vector and q is the orientation represented as a unit quaternion. Then the configuration difference is denoted as $X_{error} = [\delta x, \delta q]$. For reaching tasks, the configuration only includes the position component, and the Pseudo-inverse is only calculated with δx.

In the following parts of the section, we describe our state and action spaces, reward function and CL approach for the RL part. We also present four agents that can be trained by adding/removing certain parts of our formulation. In addition, we introduce two modifications that our method can employ post-training. We omit Pseudo-inverse control in the interest of space.

5.2 State Space

The state space contains all the essential information within an environment that enables a robot to make decisions and accomplish desired tasks. In this thesis, states are chosen so the robot can learn to reach a random target pose from a random starting pose. The state, denoted as s, is represented by the following tuple (omitting the temporal indices):

$$s : [\theta, \dot{\theta}, \delta x, S, \dot{\theta}_{pinv}, \delta q]$$

The state consists of joint angles θ, joint velocities $\dot{\theta}$, a binary variable S indicating goal achievement and the error between the end-effector and the goal position δx.

The binary success flag S is set to 1 if $\delta x < 0.01m$ and 0 otherwise. The position error is computed by calculating the Euclidean distance between the end-effector and the goal position. These are the minimum state variables to train the architecture. When the Pseudo-inverse module is activated to learn a policy, $\dot{\theta}_{pinv}$, joint velocities computed by the Pseudo-inverse module are added into the state.

The orientation difference, δq, which quantifies the difference in orientation between the end-effector and the target, is integrated into the state when the target pose includes orientation information. q_{cur} represents the current orientation information of the end-effector, while q_{target} represents the target orientation information of the target pose, both expressed in quaternions. The orientation difference is calculated as $\delta q = q_{cur} q_{target}^{-1}$ (utilizing quaternion multiplication and quaternion inverse). q_{cur} represents the current orientation of the end-effector and q_{target} denotes the target orientation.

5.3 Action Space

The action space is the set of joint velocities $\dot{\theta}$ corresponding to the robot. It means that the trained policy learns optimal values of joint velocities for a given state. This representation does not change based on the target robot task. As such, it has the same dimensionality as θ. However, what they represent can differ based on whether the Pseudo-inverse part is included. If included, actions represent corrective joint velocities. We can also train an RL agent without the Pseudo-inverse part with the same formulation, in which case the actions are directly the joint velocities. This would make the action space equivalent to the one by [Kumar et al., 2021].

5.4 Reward Function

The reward function comprises four terms, each serving a distinct purpose. We incorporated the first reward term for position proposed by [Kumar et al., 2021] into our work, where the author sets λ_{pos} to 20. However, during our experiments, we observed sub-optimal learning behavior when λ_{pos} was set to 20. To address this, we conducted hyper-parameter tuning to determine the optimal value for λ_{pos}, as

detailed in Table 6.2. Additionally, we adopted the concept proposed by [Kumar et al., 2021] of minimizing the joint acceleration norm and applied it to minimize the joint velocity norm in our work. The complete reward function is calculated as follows:

$$r = e^{\left(-\lambda_{pos}\|(\delta x)\|^2\right)} - \frac{\lambda_{velocity}\|\dot{\theta}\|}{1 + \|\delta x\|} - \lambda_{coll}c + e^{\left(-\lambda_{ori}\beta_q^2\right)} \tag{5.1}$$

where, λ_{pos}, $\lambda_{velocity}$, λ_{coll}, and λ_{ori} represent constant coefficients corresponding to each reward term. The specific values assigned to these constants can be found in Table 6.2. The first exponential term offers a high reward when positional error δx is small. This term guides the end-effector to move to the target position.

The second term ensures a smooth and accurate approach around the target region, which is lacking in multiple related approaches [Li et al., 2020c, Gu et al., 2017, Plappert et al., 2018, Mahmood et al., 2018, Gandana et al., 2020, Aumjaud et al., 2020]. This term penalized the agent if the robot has high joint velocity. There is a minus sign in front of this term because we do not want high joint velocity near the target position. The distance normalization, $1 + \|\delta x\|$, in the denominator is introduced to emphasize the importance of joint velocity near the target position. When the positional error is very large, then there is no need to pay attention to joint velocity. Because we also want the robot to decrease the positional error quickly. This approach allows the robot have large velocities when the end-effector is far from the target position.

The third term penalizes self-collisions between the robot links where c is a binary variable denoting self-collisions. If there is any self-collision, this takes on a large negative value and is 0 otherwise.

The fourth term serves the same purpose as the first term but for the orientation component. The β_q is the scalar angular difference in radians between the target and the current orientation, calculated from the scalar part of δq. The maximum reward is earned from the last term when the orientation angle error is zero.

The third term is omitted if the aim is just to train an IK model, and the fourth is omitted if the target task is reaching.

With the exception of the first term, our reward function presented in equation

5.1 can be considered a novel reward formulation.

5.5 Curriculum Learning

The primary objective of using CL is to learn complex robotic tasks from starting a simple environment and increase the complexity of the environment step-by-step. We integrate CL, which is beneficial to learn reaching tasks.[Kumar et al., 2021] To achieve this objective, we systematically expand the learning environment by creating larger regions, newer ones encompassing the older ones, and progressively train the RL part. This expansion in the regions increases the difficulty of the task. The details of the created workspace are described in Chapter 6.3.

5.6 Task Specific Formulations and Possible Agents

Throughout this section, we mention that specific components can be omitted for certain tasks and ablation. The full formulation as described is designed for pose control. For IK, self-collisions can be ignored (while checking the collisions at the target pose at the end). For reaching, orientation-related state and reward components can be removed.

In addition to the tasks, we can also train different agents, mainly as baselines and ablation. We describe these agents below. Note that the first two are analogous to the agents of [Kumar et al., 2021] and are treated as competitive baselines.

1. **Pure RL Agent (Agent 1):** Direct RL training without the Pseudo-inverse component and curriculum scheduling. In this configuration, the Pseudo-Inverse component, as illustrated in Figure 5.1, is excluded from the overall architecture for policy training.

2. **Pure RL Agent with Curriculum (Agent 2):** Curriculum training of RL without the Pseudo-inverse component. This setup combines **Agent 1** with a curriculum learning schedule.

3. **Hybrid Agent (Agent 3):** The pseudo-inverse and the RL module are used

(See 5.1) without curriculum scheduling. This thesis introduces this setup for addressing the reaching, full-pose control, and IK tasks.

4. **Hybrid Agent with Curriculum (Agent 4):** Both the pseudo-inverse and the RL output is used with curriculum scheduling. This setup combines **Agent 3** with a curriculum learning schedule.

5. **Raw Pseudo-Inverse (Agent 5):** This setup uses pseudo-inverse method to calculate the desired joint velocities. This agent serves as a traditional baseline without any learning component.

5.7 Switching

In our evaluations, we have observed that our approach falls short of the raw Pseudo-inverse control for tight success thresholds while outperforming it for looser ones. This suggests that the RL part can better take the end-effector to the vicinity of the target but fails to reach it precisely. This led us to integrate a minor modification to the system, which we call *switching*; We switch to raw Pseudo-inverse control when the Euclidean distance between the end-effector and the target position is below a threshold (i.e. only utilize θ_{pinv}). It is important to note that this is only employed during inference and not during training. This *switching* modification is not used in pure RL methods as these methods do not include Pseudo-inverse modules.

Chapter 6

EXPERIMENTS

6.1 Training Details

The experiments were performed on the Panda Gym simulator [Gallouédec et al., 2021] with a simulation time step of 0.01 seconds. Experiments were conducted using the Stable Baselines3 (SB3) framework [Raffin et al., 2021]. The Kinematics and Dynamics Library (KDL) [Orocos,] package and its damped-least squares implementation were utilized for the Pseudo-inverse module. Four robots with similar degress-of-freedom (dof) but different kinematic topologies were selected; Panda with 7-DOF (P_7), Kinova Jaco with 6-DOF (J_6) and 7-DOF (J_7), and UR5 with 6-DOF (U_6).

The same hyper-parameters, presented in Table 6.2, were used across all the training scenarios for all the robots. We empirically selected these parameters. One curious case was using the tanh activation function, which outperformed the non-saturating ReLU activation in our early trials.

We trained all agents across various robotic platforms, encompassing both position and full-pose scenarios, without accounting for self-collisions. This training involved four distinct robot platforms and four different agents, as detailed in Section 5.6, with both reaching and full-pose tasks. Consequently, this resulted in $4 \times 4 \times 2 = 32$ training instances. Additionally, we trained hybrid agents specifically for position and full-pose cases, considering self-collision, amounting to $2 \times 4 \times 2 = 16$ instances. Notably, the count is halved (from 4 to 2) because only hybrid agents were utilized for self-collision tests. Therefore, in total, we had 48 training instances.

Table 6.1: Hyper-parameters

Hyper-parameter	**Value**
Lambda($\lambda_{position}$)	100
Orientation constant ($\lambda_{orientation}$)	50
Velocity constant ($\lambda_{velocity}$)	0.1
Collision constant (λ_{coll})	10
Discount factor (γ)	0.95
Hidden units	128
Number of hidden layers	3
Hidden layer activation	tanh
# steps to update the policy	1000
Episode length	1000
# epochs	10
Batch size	2048
Learning rate	0.0003
# of parallel training environments	100
# of samples for final evaluation	1000
Raw Pseudo-Inverse Switching threshold	5cm

Table 6.2: Hyper-parameters employed for training the policy across all robotic platforms. The provided information includes the hyper-parameters for the reward function, as outlined in Section 5.4. Additionally, it explains values utilized for the PPO class in SB3.

6.2 Metrics

In the context of the reaching task, it is essential to establish specific metrics to quantify task performance. These metrics are computed at the last time step of each episode. The following metrics can be used to evaluate the system's performance.

- **Root Mean Square Error (RMSE):** RMSE is the square root of the average squared differences between the predicted and actual values. The signif-

icant errors are subject to more penalties because of the squared term. The error term in the parentheses denotes the Euclidean distance between the end-effector and the target position. The mathematical formulation for RMSE is as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \left(\mathrm{x}_{cur,i} - \mathrm{x}_{target,i}\right)^2}{N}} \tag{6.1}$$

where, $x_{\mathrm{cur},i}$ refers to the current 3-dimensional position information of the end-effector, $x_{\mathrm{target},i}$ denotes the 3-dimensional position information of the target at the last time step of episode i and N stands for the total number of episodes used for evaluation.

- **Mean Absolute Error (MAE):** MAE is the average of the absolute differences between predicted and actual values. The error term in the parentheses denotes the Euclidean distance between the end-effector and the target position. It is a measure of the average error in the model's predictions. The mathematical equation for MAE is:

$$\mathrm{MAE} = \frac{1}{N} \sum_{i=1}^{n} |x_{cur,i} - x_{target,i}| \tag{6.2}$$

- **Success Rate (SR):** The SR is a metric used in robotics for evaluating the accuracy of a robotic manipulator in reaching a desired target. It quantifies the percentage of attempts in which the manipulator's end-effector can reach a target within a defined threshold.

$$SR = (\text{ number of successful trials / total number of trials }) * 100 \tag{6.3}$$

- **Mean Joint Velocity (MJV):** It is used to measure the stability of the robot while performing tasks. The magnitude of the velocity being low when reaching the point is as critical as the distance of the end-effector to the point. The mathematical equation for mean joint velocities $\dot{q}$ norm is:

$$|\dot{q}|_{\mathrm{avg}} = \frac{1}{N} \sum_{i=1}^{n} |\dot{q}| \tag{6.4}$$

- **10cm-30 degrees SR:** An episode is regarded as successful if the positional and orientation errors between the target pose and the end-effector pose are less than 10 cm and 30 degrees, respectively. This metric is primarily employed for orientation tests to evaluate the performance of full-pose control.

- **5cm-10 degrees SR:** An episode is regarded as successful if the positional and orientation errors between the target pose and the end-effector pose are less than 5 cm and 10 degrees, respectively.

6.3 Workspaces

A workspace should be defined to select the robot's initial configuration and the target pose randomly within its boundaries. In our study, we employ the KDL package to generate the workspace for each robot. The process involves uniformly sampling the robot's joint angles within their defined limits. The end-effector's position and orientation are computed for adding into the workspace using FK. The uniformly sampled joint angles and their corresponding end-effector positions and orientations are then stored. The URDF file must be provided to the KDL package to compute the FK result.

Each robot's workspace is divided into four curriculum regions. Each curriculum region covers the previous curriculum region points. The last of these regions is called the *main workspace*. In the horizontal (x-y) plane, the radius of the workspace for each robot and each curriculum region is set to the maximum reachable distance by the manipulator. This means that we include the points even at the boundaries of the reachable workspace in the (x-y) plane. For the Jaco 6-DoF robot, the height z is constrained within the range of $-0.5m < z < 0.5m$, and for other robots, it is constrained within $-0.1m < z < 0.9m$ for each curriculum region. To exclude self-collision cases, we remove all points within a circle centred at (0,0,0) with a radius of $0.2m$ from the workspace. The front of the robot is sliced into symmetric pieces of 40, 100, 140, and 180 degrees to generate the four curriculum regions.

Figure 6.1 illustrates the curriculum regions for the Panda robot. The curriculum regions plot for other robots can be found in Appendix A. As illustrated in Figure

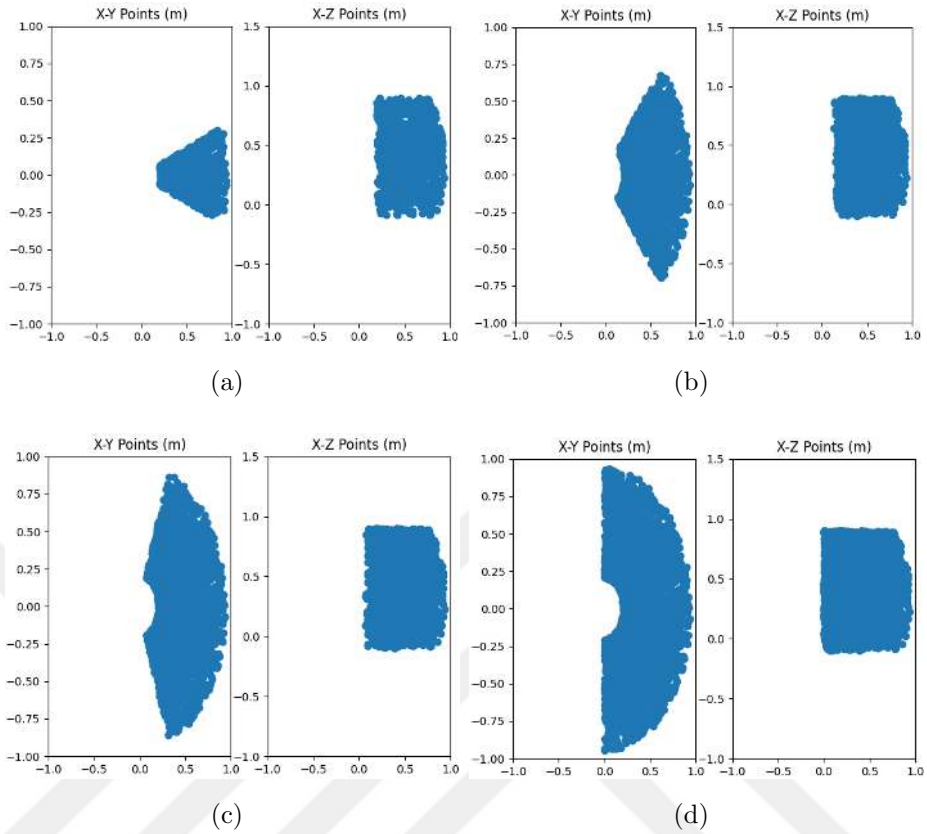


Figure 6.1: Curriculum regions for Panda (a) First curriculum region, (b) Second curriculum region, (c) Third curriculum region and (d) Fourth curriculum region. Each sub-figure shows points in horizontal (x-y) and vertical (x-z) planes.The initial curriculum region represents the smallest region for the Panda robot.The training procedure starts with using the first region. The last curriculum region is called *main workspace*. The curriculum regions plot for other robots can be found in Appendix A.

6.1, we modify only the front degrees when generating the subsequent curriculum region. The *main workspace* of each robot in 2-dimensional space is illustrated in Figure 6.2. The color representation of the *main workspace* is illustrated in Figure A.6 Additionally, the range for each dimension and the volume size of the workspace are presented in Table 6.3.

We initiate the training of the RL architecture using the smallest curriculum region. To progress to the following curriculum region, we implement a callback function that evaluates the task performance during the training phase, following the methodology outlined in the study by Kumar et al. [Kumar et al., 2021]. We

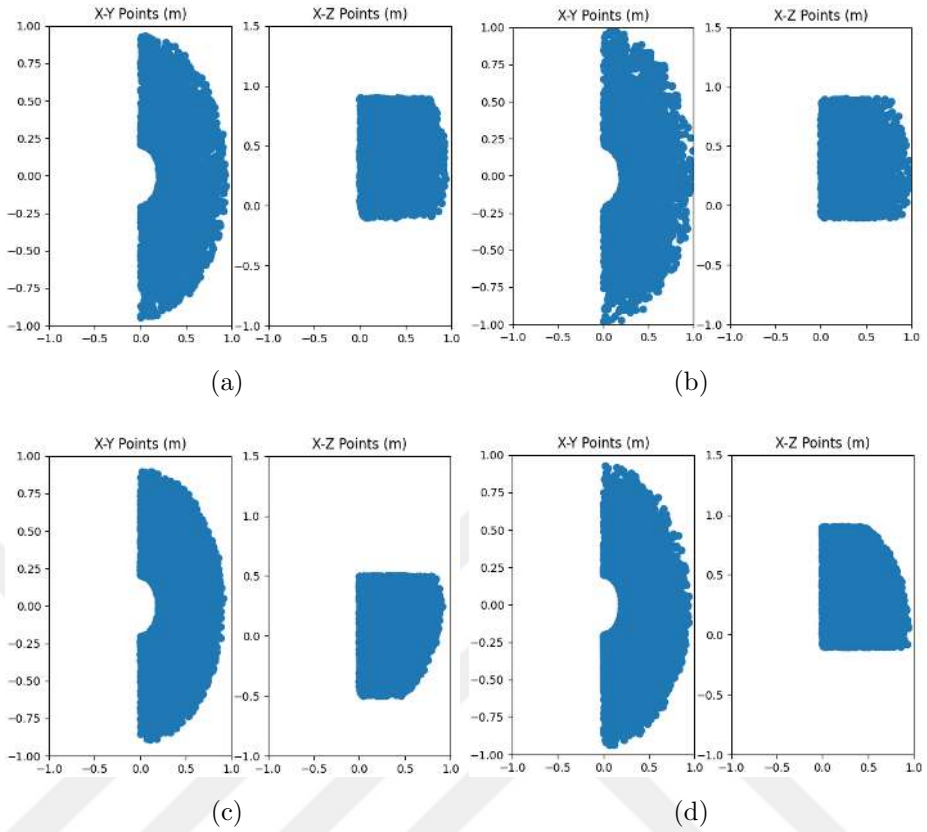


Figure 6.2: *Main workspace* of (a) Panda, (b) Jaco 7-DoF, (c) Jaco 6-DoF (d) UR5. *Main workspace* is the last curriculum region for each robot platform. The final evaluation (metric results) to measure the reaching and full-pose control performance,is conducted based on the points within these main workspaces.

evaluate reaching or full-pose task performance during the training phase using 500 episodes at every 2 million training time steps within the current curriculum region. In tests where self-collision awareness is not considered, we move on to the subsequent region if the MAE in position and the norm of MJV at the final state are below certain thresholds. We set the MAE threshold to $5cm$ and the MJV norm threshold to $0.15rad/s$ for scheduling without self-collisions. In contrast, while considering self-collision awareness, we terminate the episode in case of a collision. As a result, checking the MAE and MJV norms is no longer helpful for moving on to the subsequent curriculum region. In this scenario, we set the number of training time steps to a specific number when collisions are considered. While considering self-collisions, the agent uses the subsequent curriculum region after 3

	$X_{range}(m)$	$Y_{range}(m)$	$Z_{range}(m)$	Volume (m^3)
Panda	[0, 0.93]	[-0.93, 0.93]	[-0.1, 0.9]	1.29
Jaco 7-DoF	[0, 1]	[-1, 1]	[-0.1, 0.9]	1.5
Jaco 6-DoF	[0, 0.9]	[-0.9, 0.9]	[-0.5, 0.5]	1.1
UR5	[0, 0.93]	[-0.93, 0.93]	[-0.1, 0.9]	1.29

Table 6.3: The lower and upper values for each axis (X, Y, and Z) within the *main workspace* are determined by uniformly sampling the robot joint angles and subsequently computing the end-effector position and orientation using FK. The lower and upper boundaries in each axis are determined by extracting the minimum and maximum values in each axis. The lower and upper values for each axis can be visually assessed in Figure 6.2. Additionally, the Table presents the *main workspace* volume for each robot type.

million training time steps.

Figure 6.4 illustrates the histogram of the position and orientation space within the *main workspace* for the Panda 7-DoF. The histogram is constructed with 100 bins with equal-width bins in a given range. Table 6.3 details the range of values for the position space. The Roll and Yaw angles span from $-\pi$ to π, while the Pitch angles span from $-\pi/2$ to $\pi/2$ due to the kinematic structure of the robot.

Observing the histograms for the X-axis and Y-axis, it is evident that boundary points occur less frequently compared to the inner regions within the *main workspace*. Despite uniformly sampling joint angles and calculating the end-effector's position and orientation using FK, the position and orientation space do not seem uniformly sampled. The same observation can be seen in Figure 6.4b where the orientation space also demonstrates non-uniform distribution. The histogram plot of position and orientation space for other robots can be found in Appendix A.

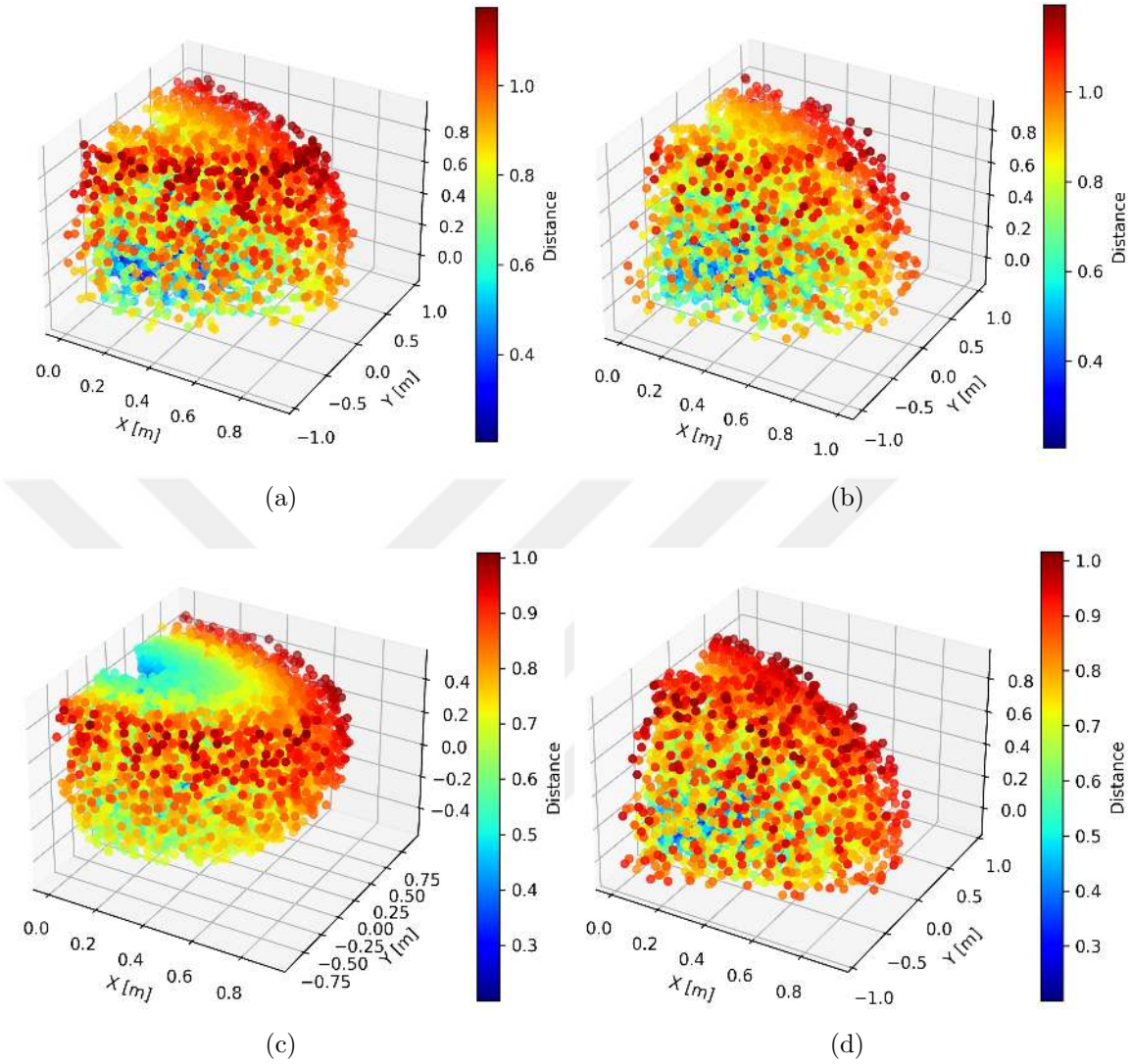
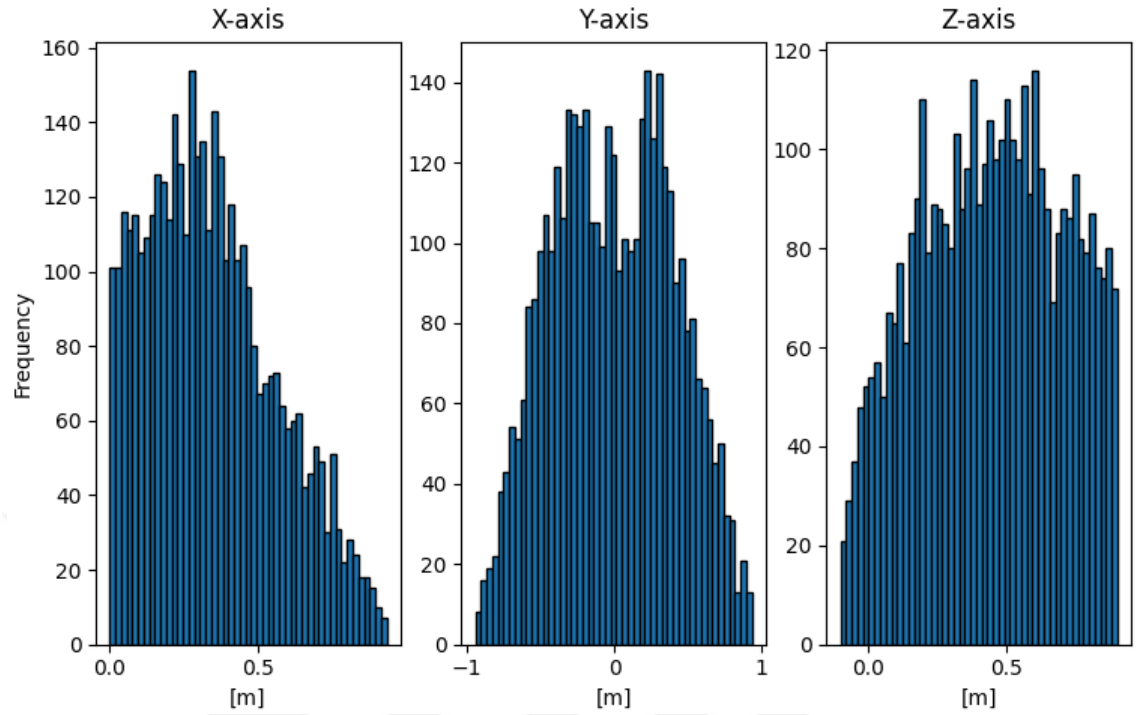
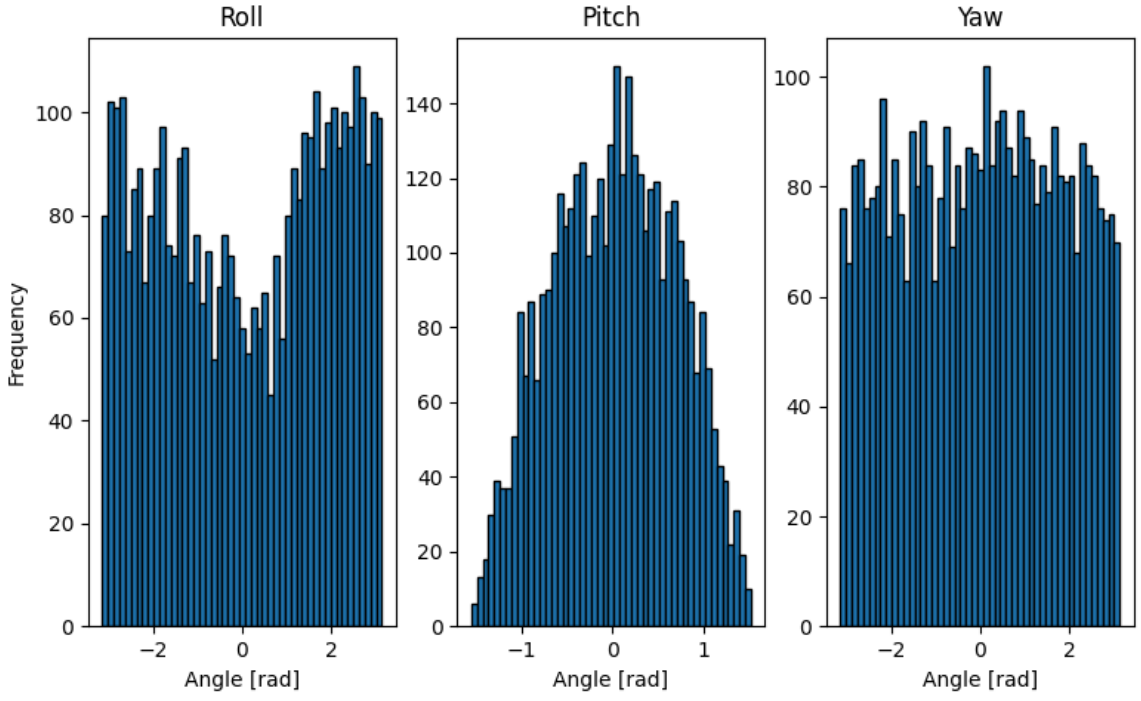


Figure 6.3: *Main workspace* of (a) Panda, (b) Jaco 7-DOF, (c) Jaco 6-DOF (d) UR5. The color representation of the *main workspace* in three-dimensional space for each type of robot. The distance between the points and the origin is reflected in the color representation.

.



(a)



(b)

Figure 6.4: Histogram of (a) Position space for Panda, (b) Orientation space for Panda. The position and orientation space of the *main workspace* for Panda. Each plot consists of 100 bins to show the frequency of the values. The range of the position values are already described in Table 6.3. The roll and yaw angles are in the interval of $[-\pi, \pi]$ radians. And the pitch angles is in the $[-\pi/2, \pi/2]$ radians due to the kinematic structure of the robot. The histogram plot of position and orientation space for other robots can be found in Appendix A.

Chapter 7

RESULTS

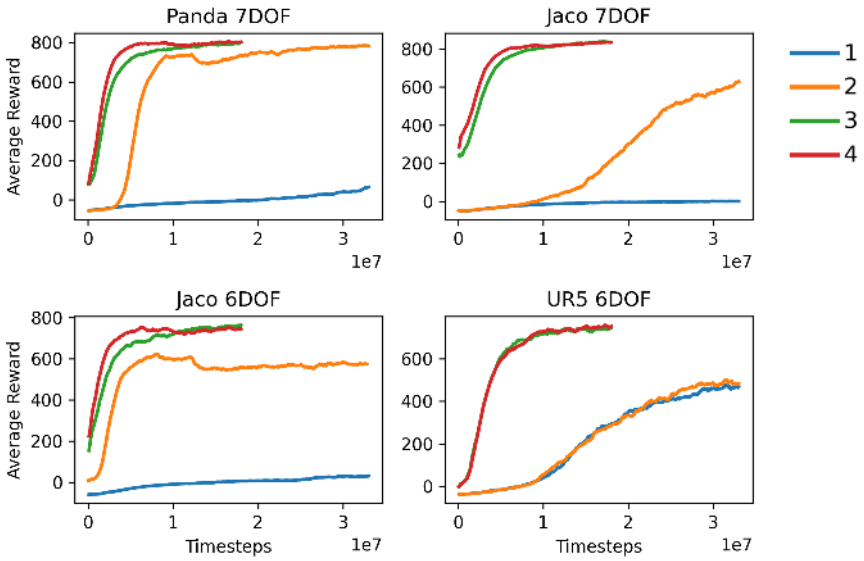
7.1 Learning Curves

Figure 7.1 presents the learning curves for the collision-ignored reaching task, corresponding to the scenario outlined in Table 7.1. The training duration for the pure RL Agent was set to 33 million time steps. We trained our hybrid approach for 18 million time steps, since integrating a Pseudo-inverse module significantly improved convergence rate and returns.

Agent 2 (pure RL Agent with CL) outperformes Agent 1 (pure RL agent) globally due to the curriculum schedule. However, despite the schedule, achieving a successful policy is challenging due to the large workspace coverage. In contrast, experiments with hybrid Agents 3 and 4 (curriculum trained hybrid agent) demonstrate faster and more successful policy learning compared to Agents 1 and 2. As a result, Agents 1 and 2 were not trained for the self-collision-enabled tests (See Figure 7.3 and 7.4).

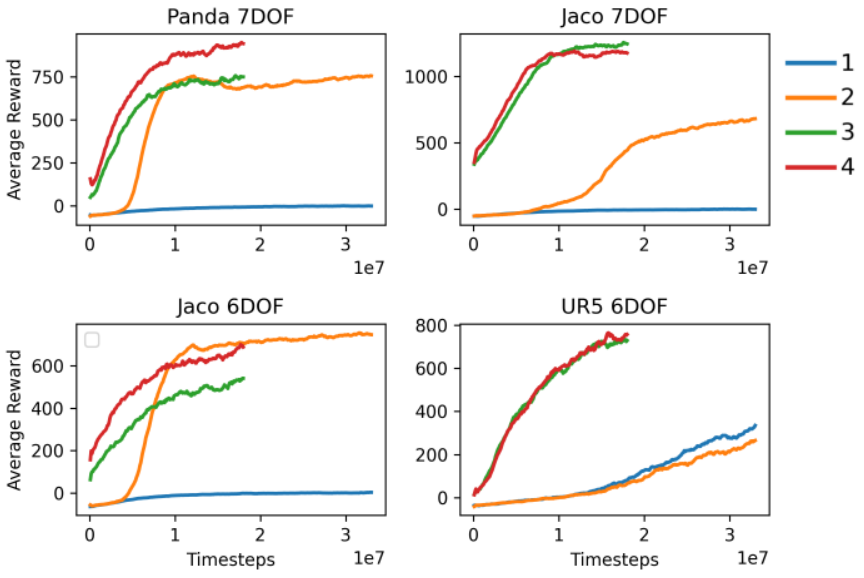
Figure 7.2 showcases the learning curves for the collision ignored full-pose control task. (Same scenario as Table 7.2). Agents 3 and 4 start with a higher initial average reward due to the inclusion of the Pseudo-inverse module, in contrast to Agents 1 and 2. Agents 3 and 4 displayed efficient policy learning across all robot platforms with fewer training time steps, except for Jaco 6-DOF. Interestingly, Agent 2 showed good performance for Jaco 6-DOF. However, a direct comparison is challenging due to the reward function's complexity. Agent 2 primarily derives its reward from the mean norm of joint velocity and Euclidean distance between the end-effector and the target pose. Agent 2 performs poorly on the 10cm30 and 5cm10 metrics, as reported in Table 7.2.

Figure 7.3 illustrates the learning curve for the reaching task with self-collisions (Same scenario as Table 7.3). and Figure 7.4 presents the learning curve for the full-



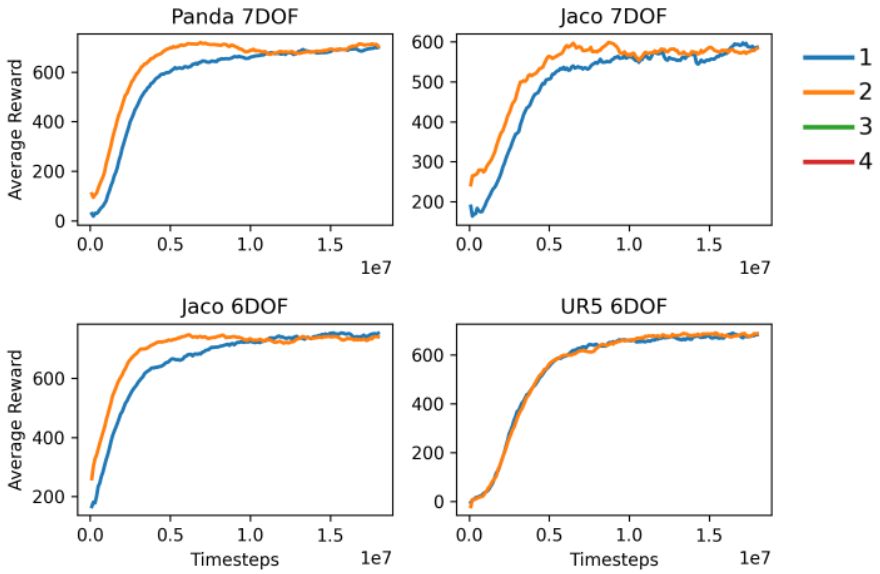
(a)

Figure 7.1: Learning curves for the test where the target only contains position information and self-collisions are ignored. The x-axis represents the number of training time steps, while the y-axis illustrates the average reward. Each agent is represented by different colors and the curves are smoothed to convey the trends.



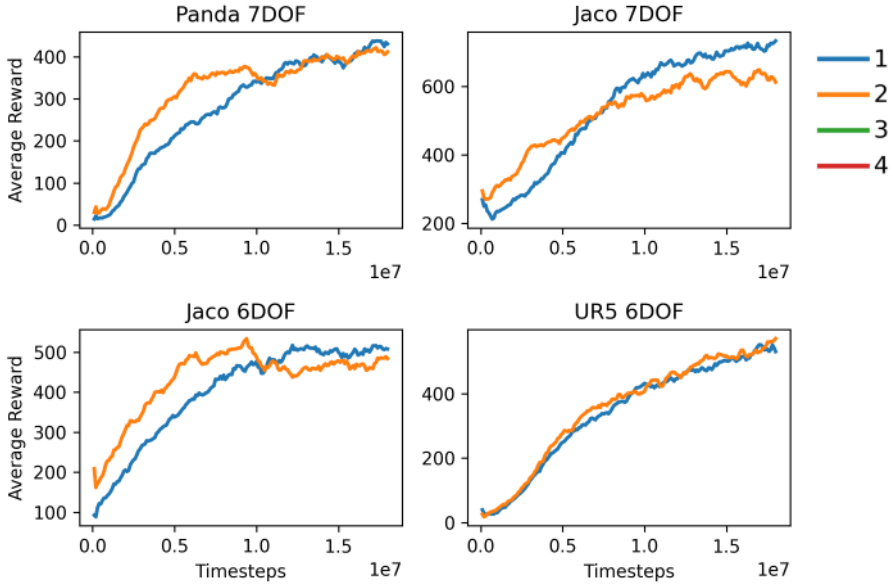
(a)

Figure 7.2: Learning curves for the test where the target contains position and orientation information and self-collisions are ignored.



(a)

Figure 7.3: Learning curves for the test where the target only contains position information and self-collisions are considered.



(a)

Figure 7.4: Learning curves for the test where the target contains position and orientation information and self-collisions are considered.

pose control task with self-collisions (Same scenario as Table 7.4). In both Figures, Agents 3 and 4 show comparable performance. Notably, experiments with Agents 1 and 2 were omitted from the learning curve as they demonstrated poor performance in tests when self-collisions are considered.

Overall, we can see that the training performance of the hybrid agents is vastly superior to pure RL agents and that they do no require curriculum scheduling. Next, we evaluate all the approaches in test scenarios.

7.2 Evaluation

We utilize mae, SR at 1cm and 5cm thresholds, and mean joint velocity norms at the episode end as our metrics for the to evaluate the reaching task. For the full-pose control task, we utilize SR at 10 cm-30 degrees and 5cm-10 degrees thresholds and mean joint velocity norms. See Chapter 6.2 for details. For collisions, we drop the mae metric and add collision rate as the ratio of episodes that end with collision to total episodes. Note that if the collision are ignored, training and testing episodes continue even if a collision occurs between the robot's links, and the episodes are immediately terminated otherwise.

The raw Pseudo-inverse agent serves as a traditional baseline, and Agents 1 and 2 serve as learning baselines. However, we do not train pure RL agents when considering self-collisions, due to their poor performance in the no-collision scenarios.

We utilize multiple Tables to present our results due to the number of evaluation scenarios. Results for the scenarios that ignore self-collisions are presented in Table 7.1 for the reaching task and in Table 7.2 for the full-pose control task. In the self-collision ignored scenarios, testing (and training) episodes continue even if a collision occurs between the robot's links. Results for the scenarios that consider self-collisions are presented in Table 7.3 for reaching task and in Table 7.4 for the full-pose control task. The episodes are stopped when a collision is detected in these scenarios.

In these Tables, the robot types are denoted as follows: Panda (P_7), Jaco 7DOF (J_7), Jaco 6DOF (J_6), and UR5 (U_6). The agent numbers indicate which training

Table 7.1: Evaluation results for Position Targets without Considering Self-Collisions for 1000 random test points

R[1]	**Agent**[2]	**MAE**[3]	**SR**[4]**(1cm)**	**SR(5cm)**	**MJV**[5]
P_7	Pseudo	3.2	0.845	0.906	0.006
	1	52	0.000	0.073	0.077
	2	3.1	0.816	0.965	0.056
	3/Sw[6]	1.4/0.9	0.858/0.938	**0.97**/**0.97**	0.091/0.004
	4/Sw	1.5/**0.8**	0.778/**0.947**	**0.97**/**0.97**	0.065/**0.001**
J_7	Pseudo	0.5	0.941	0.973	0.002
	1	62.7	0.00	0.001	0.043
	2	23.2	0.067	0.569	0.123
	3/Sw	0.6/**0.06**	0.877/**0.98**	**1/1**	0.047/**0.000**
	4/Sw	0.7/**0.06**	0.867/**0.98**	**1/1**	0.052/**0.000**
J_6	Pseudo	3.8	0.823	0.866	**0.006**
	1	50.3	0.001	0.031	0.034
	2	6.8	0.677	0.902	0.118
	3/Sw	2.8/**2.3**	0.748/**0.85**	0.917/**0.921**	0.055/0.007
	4/Sw	3.3/2.8	0.72/**0.85**	0.907/0.912	0.044/0.007
U_6	Pseudo	5.1	0.77	0.824	**0.004**
	1	23.3	0.086	0.538	0.11
	2	21.1	0.081	0.543	0.097
	3/Sw	5.1/**4.1**	0.415/0.782	0.848/**0.85**	0.053/0.012
	4/Sw	5/**4.1**	0.449/**0.791**	**0.85**/**0.85**	0.057/0.011
Avg.[7]	Pseudo	3.1	0.844	0.892	**0.004**
	1	47	0.021	0.16	0.066
	2	13.5	0.41	0.744	0.098
	3/Sw	2.4/**1.8**	0.724/**0.89**	**0.93**/**0.93**	0.061/0.005
	4/Sw	2.6/**1.8**	0.703/**0.89**	**0.93**/**0.93**	0.054/**0.004**

[1] Robot abbreviations, [2] See Sec. 5.6 for specific agents, [3] Mean absolute error in cm, [4] Success rate, [5] Mean joint velocities norm in rad/s at the end of the episode, [6] Switching modification as described in Sec. 5.7, [7] Average values over the robots. Best results are highlighted in bold.

agent was used, as described in Section 5.6. Agents 1 and 2 do not incorporate the Pseudo-inverse module, while agents 3 and 4 include this module. Agents 2 and 4 are trained with a curriculum schedule.

Table 7.2: Evaluation results for Full-Pose Targets without Considering Self-Collisions for 1000 random test points

R	**Agent**	**10cm30**[1]	**5cm10**	**MJV**
P_7	Pseudo	0.595	0.502	0.148
	1	0.00	0.00	**0.052**
	2	0.014	0.001	0.065
	3/Sw	**0.644**/0.635	**0.54**/**0.54**	0.185/0.152
	4/Sw	0.616/0.598	0.514/0.508	0.153/0.14
J_7	Pseudo	0.896	0.865	0.055
	1	0.00	0.00	0.037
	2	0.01	0.00	0.121
	3/Sw	**0.94**/**0.94**	**0.93**/0.929	0.053/0.041
	4/Sw	**0.94**/**0.94**	**0.93**/0.925	0.052/**0.036**
J_6	Pseudo	0.402	0.371	0.612
	1	0.00	0.00	**0.029**
	2	0.021	0.002	0.070
	3/Sw	**0.443**/0.439	0.374/**0.38**	0.561/0.573
	4/Sw	0.381/0.389	0.336/0.343	0.6/0.618
U_6	Pseudo	0.535	**0.473**	0.193
	1	0.002	0.001	**0.099**
	2	0.002	0.00	0.115
	3/Sw	**0.55**/**0.55**	0.425/0.452	0.183/0.170
	4/Sw	0.549/0.544	0.434/0.46	0.187/0.187
Avg	Pseudo	0.607	0.552	0.252
	1	0.000	0.000	**0.054**
	2	0.011	0.000	0.092
	3/Sw	**0.64**/**0.64**	0.569/**0.576**	0.245/0.234
	4/Sw	0.623/0.619	0.554/0.559	0.248/0.245

[1] The success rate when the position distance is below 10cm, and the angular difference is below 30 degrees.

For the reaching task, Tables 7.1 and 7.3 show similar results across the agents with hybrid agents outperforming the other ones. In addition, the *switching* method improved results, particularly regarding the 1 cm and 5 cm SR. Most methods were able to get small joint velocities.

Table 7.3: Evaluation results for Position Targets with Considering Self-Collisions for 1000 random test points

R	Agent	SR(1cm)	SR(5cm)	MJV	Colli[1]
P_7	Pseudo	0.771	0.824	**0.004**	0.105
	3/Sw	0.66/**0.87**	0.898/0.899	0.072/**0.004**	**0.07**/**0.07**
	4/Sw	0.701/**0.87**	**0.9**/**0.9**	0.069/0.005	**0.07**/**0.07**
J_7	Pseudo	0.674	0.691	0.0006	0.309
	3/Sw	0.46/**0.74**	0.621/**0.75**	0.04/**0.0001**	0.393/**0.25**
	4/Sw	0.464/**0.74**	**0.75**/**0.75**	0.041/**0.0001**	**0.25**/**0.25**
J_6	Pseudo	0.818	0.859	**0.005**	0.02
	3/Sw	0.807/0.848	0.908/**0.91**	0.047/0.007	0.018/0.016
	4/Sw	0.78/**0.853**	**0.91**/**0.91**	0.042/**0.005**	0.013/**0.012**
U_6	Pseudo	0.646	0.693	**0.003**	0.077
	3/Sw	0.356/**0.68**	0.768/**0.77**	0.047/0.011	**0.039**/**0.039**
	4/Sw	0.377/**0.68**	**0.77**/**0.77**	0.056/0.013	0.04/**0.039**
Avg	Pseudo	0.727	0.766	**0.003**	0.127
	3/Sw	0.57/**0.78**	0.798/**0.83**	0.051/0.005	0.132/**0.09**
	4/Sw	0.58/**0.78**	**0.83**/**0.83**	0.052/0.005	**0.09**/**0.09**

[1] Self-collision rate. Lower is better

We evaluated the same 1000 random test points for the full-pose control task as well. Tables 7.2 and 7.4 show the results with and without enabling the self-collisions, respectively. For both Table results, agent 3 with *switching* achieves better outcomes regarding the 10cm30 and 5cm10 SR, except for the 5cm10 threshold on the UR5 robot. On the other hand, agent 4 achieves better position and orientation results than the raw Pseudo-inverse in most cases. However, agent 4 fails regarding the 10cm30 and 5cm10 SR for the Jaco 6DOF robot and the 5cm10 threshold for the UR5 robot.

Table 7.2 highlights agent 1 with the minimum mean norm of joint velocities globally. Nevertheless, it is crucial to note that lower joint velocities alone are insufficient. We aim for higher SR in the 10cm30 and 5cm10 thresholds when orientation information is incorporated into the target pose. This explains why agents 1 and 2 did not perform well in Table 7.2, struggling to achieve high SR. In contrast, agents

Table 7.4: Evaluation results for Full-Pose Targets with Considering Self-Collisions for 1000 random test points

R	Agent	10cm30	5cm10	MJV	Colli
P_7	Pseudo	0.391	0.326	0.189	0.216
	3/Sw	0.46/**0.459**	0.333/**0.353**	0.213/0.2	**0.14/0.14**
	4/Sw	0.446/0.439	0.321/0.344	**0.18/0.18**	0.156/0.16
J_7	Pseudo	0.523	0.515	0.009	0.454
	3/Sw	0.352/0.538	0.348/0.529	0.012/**0.004**	0.642/0.452
	4/Sw	**0.54/0.54**	**0.535/0.535**	0.024/0.009	**0.439**/0.442
J_6	Pseudo	0.399	0.369	0.61	0.015
	3/Sw	**0.44/0.44**	0.382/**0.392**	**0.545**/0.56	0.015/0.014
	4/Sw	0.391/0.39	0.346/0.352	0.58/0.592	**0.013/0.013**
U_6	Pseudo	0.387	**0.327**	0.142	0.142
	3/Sw	0.399/0.396	0.272/0.298	0.149/0.156	0.093/0.093
	4/Sw	**0.41/0.41**	0.278/0.31	**0.136**/0.137	**0.085/0.085**
Avg	Pseudo	0.425	0.384	0.237	0.206
	3/Sw	0.414/**0.46**	0.333/**0.393**	**0.229**/0.23	0.223/0.176
	4/Sw	0.449/0.447	0.37/0.385	0.232/0.231	**0.173**/0.175

3 and 4 demonstrate performance very close to the raw Pseudo-inverse results regarding the mean norm of joint velocities and perform better results in terms of SR globally.

Additionally, adding a collision constant to the reward function makes the policy to learn self-collision awareness. As can be seen from the Table, this approach results in a decrease in the number of self-collisions when compared to the raw Pseudo-inverse results.

The collision ignored results given by Tables 7.1 and 7.2, specifically the 1cm SR, show the potential of using the methods for IK. If the task space is only the positions, hybrid agents are the best, closely followed by the PinvC baseline. If the task-space is the end-effector pose, then the hybrid agents and PinvC baseline are comparable.

7.3 Result Discussion

Hybrid agents outperform Pure RL agents in almost all the metrics. This is readily seen by comparing their average metrics in Tables 7.1 and 7.2 (Agents 3-4 vs 1-2). Individual robot metrics and the training curves (Figure 7.1) echo these results. Learning to reach in a large workspace is challenging. It can be argued that the Pseudo-inverse guides the hybrid agents better than just random exploration. On this note, Pseudo-inverse agent also outperforms pure RL agents.

CL significantly helps Pure RL agents while it does not have a similar effect on hybrid agents. Tables 7.1 and 7.2 show the performance gap between agents 1 and 2. The Panda and Jaco 6-DOF results in Tables 7.1 for the curriculum pure RL agent show competitive 5cm SR. On the other hand, Figure 7.1 shows slightly better convergence rates for the curriculum hybrid agent and Tables 7.3 and 7.4 show better collision rates.

Ignoring self-collisions is beneficial for IK. The SR decreases across the board when we consider self-collisions. Collisions must be considered for reaching or pose control, but for IK, there is no problem with intermediate joint configurations being in collision.

Hybrid agents vs Pseudo-inverse: This comparison is more nuanced than the previous ones. For the position only case (Tables 7.1 and 7.3), the hybrid agents have lower mae scores and better 5cm SR, which imply that they are better at getting in the vicinity of the target. In addition, the curriculum trained hybrid agent has a slight edge in collision rates. On the other hand, Pseudo-inverse agent has better 1cm SR and, for the collision ignored case, lower terminal joint velocities. This implies that the Pseudo-inverse method is better at precisely positioning if it gets close to the target. When we look at the full-pose targets (Tables 7.2 and 7.4), agents are more or less on par, with hints of the same conclusions as the position only case.

Integrating switching boosts hybrid agent performance for most cases and never decreases performance. The boost is significant for 1cm SR and mean joint velocity norm for the position only cases (Tables 7.1 and 7.3). It seldom improves

5cm SR since the end-effector may move in and out of the 5 cm threshold without *switching*. These results imply that **Hybrid RL agents with switching is the best option for reaching.** Unfortunately, when orientation is included, *switching* does not have a clear benefit.

Full-Pose Control is difficult in a large workspace and more work is required towards this end. Tables 7.2 and 7.4 paint a bleak picture compared to the position only case. Only the Jaco 7 Dof metrics without collisions show favourable results for the hybrid *switching* agents. Still, agent 3 with *switching* achieves better outcomes than Pseudo-inverse regarding the 10cm30 and 5cm10 SR, except for the 5cm10 one for the UR5 robot.

Mean joint velocity norm is very high for Full-Pose Control case. The mean joint velocity norm shows large values in the context of full-pose control, as evident from the data presented in Tables 7.2 and 7.4. This phenomenon is attributed to the large workspace associated with the full-pose control task. Here, all the agents encounter challenges aligning the end-effector's orientation with the target pose orientation.

The robots with 7-DOF show better learning performance than 6-DOF robots due to having one more DOF. This observation becomes clear while examining raw Pseudo-inverse metrics for 6 and 7 DOFs. Tables 7.1 and 7.2 illustrate that 7-DOF robots display lower mae and higher SR in positional and orientation tasks than 6-DOF robots. On a global scale, the performance improvement observed with the hybrid agent with *switching* mechanisms for 7-DOF robots is better than that observed with 6-DOF robots. This difference comes from the advantage of having one more extra degree of freedom for 7-DOF robots. Additionally, for the same reason, we observe a higher number of self-collisions in 7-DOF robots compared to 6-DOF robots, as illustrated in Tables 7.3 and 7.4.

7.4 Full-Pose Control Discussion

We were not able to get good full-pose control results. The lack of studies in the literature also point to this being a larger challenge than it first seems. Based on

our results and experience, we posit that the following issues are keeping us from better full-pose control results and offer potential remedies:

- Robot Kinematics: All of the robots we employed had non-spherical wrists. This implies that small desired changes in end-effector orientation might require large changes in the joint configuration. In such states, PinvC outputs large velocities that may dominat the RL component. In addition, large velocities risk self-collisions and joint limit violations. A more informative state space and reward functions may be needed. For example robot topology can be included in the state somehow (e.g. via graph-based neural networks) and manipulability metrics can be incorporated in the state and/or rewards.

- Immediate Effort on Orientation: Both the PinvC method and the RL agents try to satisfy the orientation immediately. For certain cases, this clashes with the reaching part of the task. A gradual increase in orientation reward as the end-effector gets closer to the target, similar to the joint velocity case, can be incorporated into the reward function.

- Lack of Inductive Bias in the Policy Representation: A multi-layer perceptron architecture may not be able to learn a policy to handle an entire state space even with large number of parameters due to lack of inductive bias. A better architecture, perhaps incorporating graph-neural networks to define the relationship between the current link configurations, would potentially perform better.

- Simple *Switching* Mechanism: One remedy to the above problem is learning multiple multiple controllers to handle different parts of the state space or handle different behaviors (e.g. coarse reaching towards the target with less emphasis on orientation and a fine positioning for full-pose control). These would require to an engineered or a learning-based method to perform *switching*. Hierarchical RL methods maybe incorporated towards this end. As a lower-hanging fruit. The simple *switching* idea can be improved with better *switching* conditions to perform better.

- Orientation Success in the Curriculum: We did not include tight orientation conditions for curriculum scheduling. Making sure that the robot has learned to orient itself better at a target pose before moving on the next workspace may be able to help with performance.

- Position and Orientation Space: Uniformly sampling the joint angles and then computing the FK does not ensure a uniformly sampled end-effector position and orientation space. The curriculum regions may be modified to have uniformly sampled position and orientation space to achieve better results for full-pose control tasks.

Chapter 8

CONCLUSION AND FUTURE WORK

We introduced a hybrid learning approach that combines Pseudo-inverse control and a model-free RL algorithm to learn reaching and pose control robotic tasks, along with IK. The method works by learning corrective joint velocities on top of the Pseudo-inverse ones to guide the end-effector to a target configuration with the option of *switching* to full Pseudo-inverse for better terminal guidance.

Our approach is designed with large workspaces, arbitrary initial and target configurations, full-pose targets and self-collision awareness in mind. We also aimed for generality so that our approach can be used in an off-the-shelf manner without worrying about redundancy, joint limits, singularities, hyper-parameter tuning and even curriculum scheduling since it had minimal benefits.

We conducted a comprehensive evaluation of our method using multiple robots and various problem settings (curriculum, configuration, self-collision) with tight success metrics. Thanks to the Pseudo-inverse component, our results showed that we can handle large workspaces for reaching tasks. Our approach significantly outperforms a pure RL baseline with or without CL. It also outperforms a traditional baseline with the inclusion of *switching*. All of the results were obtained with the same set of hyper-parameters.

Even though our hybrid approach performed better on average than the rest for full-pose control, there is much more work to be done. The reward function can be updated, for example, to dampen orientation tracking far away from the pose by normalizing it with the distance, similar to the joint velocity case. Curriculum scheduling can be modified to incorporate orientation error. Furthermore, a switching-like idea can be incorporated where the agent switches from reaching to pose control depending on the relative pose of the end-effector wrt the target. State space expansion or re-formulation may also be needed to solve this problem.

We sacrificed obstacle avoidance and real-robot experiments while keeping self-collision avoidance to focus on multiple robots, multiple agents and various ablation settings. We instead rely on the results of existing work (e.g. [Kumar et al., 2021, Lucchi et al., 2020]) to argue that real-robot transfer is feasible with minimal fine-tuning and obstacle avoidance can be incorporated by augmenting the state space. An interesting future work is incorporating generic perception for better obstacle avoidance.

BIBLIOGRAPHY

[Alatartsev et al., 2015] Alatartsev, S., Stellmacher, S., and Ortmeier, F. (2015). Robotic task sequencing problem: A survey. *Journal of intelligent & robotic systems*, 80:279–298.

[Andrychowicz et al., 2017] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, 30.

[Aumjaud et al., 2020] Aumjaud, P., McAuliffe, D., Rodríguez-Lera, F. J., and Cardiff, P. (2020). Reinforcement learning experiments and benchmark for solving robotic reaching tasks. In *Workshop of Physical Agents*, pages 318–331. Springer.

[Barth-Maron et al., 2018] Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Tb, D., Muldal, A., Heess, N., and Lillicrap, T. (2018). Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*.

[Berenson et al., 2009] Berenson, D., Srinivasa, S. S., Ferguson, D., and Kuffner, J. J. (2009). Manipulation planning on constraint manifolds. In *2009 IEEE international conference on robotics and automation*, pages 625–632. IEEE.

[Bick and Wiering, 2021] Bick, D. and Wiering, M. (2021). *Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization*. PhD thesis, Master's thesis, 2021.[Online]. Available: https://fse. studenttheses. ub ....

[Bruno et al., 1994] Bruno, S., Lorenzo, S., Luigi, V., and Giuseppe, O. (1994). Robotics: modelling, planning and control, 2010. *Cited on*, 1.

[Chang and Stone, 2013] Chang, G. A. and Stone, W. L. (2013). An effective learning approach for industrial robot programming. In *2013 ASEE Annual Conference & Exposition*, pages 23–159.

[Chen et al., 2018] Chen, T., Murali, A., and Gupta, A. (2018). Hardware conditioned policies for multi-robot transfer learning. *Advances in Neural Information Processing Systems*, 31.

[Coumans and Bai, 2021] Coumans, E. and Bai, Y. (2016–2021). Pybullet, a python module for physics simulation for games, robotics and machine learning.

[Doerr and Linares, 2020] Doerr, B. and Linares, R. (2020). Motion planning and control for on-orbit assembly using lqr-rrt* and nonlinear mpc. *arXiv preprint arXiv:2008.02846*.

[Fawzi et al., 2022] Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R Ruiz, F. J., Schrittwieser, J., Swirszcz, G., et al. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53.

[Franceschetti et al., 2021] Franceschetti, A., Tosello, E., Castaman, N., and Ghidoni, S. (2021). Robotic arm control and task training through deep reinforcement learning. In *International Conference on Intelligent Autonomous Systems*, pages 532–550. Springer.

[Fujimoto et al., 2018] Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR.

[Gallouédec et al., 2021] Gallouédec, Q., Cazin, N., Dellandréa, E., and Chen, L. (2021). panda-gym: Open-source goal-conditioned environments for robotic learning. *arXiv preprint arXiv:2106.13687*.

[Gandana et al., 2020] Gandana, C. E., Disu, J. D., Xie, H., and Gu, L. (2020). Analyzing different unstated goal constraints on reinforcement learning algorithm for reacher task in the robotic scrub nurse application. In *2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pages 42–47. IEEE.

[Gu et al., 2017] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE.

[Gu et al., 2016] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep q-learning with model-based acceleration. In *International conference on machine learning*, pages 2829–2838. PMLR.

[Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

[Inoue et al., 2017] Inoue, T., De Magistris, G., Munawar, A., Yokoya, T., and Tachibana, R. (2017). Deep reinforcement learning for high precision assembly tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 819–825. IEEE.

[James and Johns, 2016] James, S. and Johns, E. (2016). 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759*.

[Jiang et al., 2013] Jiang, L., Huo, X., Liu, Y., and Liu, H. (2013). An analytical inverse kinematic solution with the reverse coordinates for 6-dof manipulators. In *2013 IEEE International Conference on Mechatronics and Automation*, pages 1552–1558. IEEE.

[Kazem et al., 2008] Kazem, B. I., Mahdi, A. I., and Oudah, A. T. (2008). Motion planning for a robot arm by using genetic algorithm. *Jjmie*, 2(3):131–136.

[Kilinc and Montana, 2022] Kilinc, O. and Montana, G. (2022). Reinforcement learning for robotic manipulation using simulated locomotion demonstrations. *Machine Learning*, pages 1–22.

[Killpack et al., 2016] Killpack, M. D., Kapusta, A., and Kemp, C. C. (2016). Model predictive control for fast reaching in clutter. *Autonomous Robots*, 40:537–560.

[Kuffner and LaValle, 2000] Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE.

[Kumar et al., 2021] Kumar, V., Hoeller, D., Sundaralingam, B., Tremblay, J., and Birchfield, S. (2021). Joint space control via deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3619–3626. IEEE.

[LaValle et al., 2001] LaValle, S. M., Kuffner, J. J., Donald, B., et al. (2001). Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 5:293–308.

[Lewis II et al., 2019] Lewis II, W. C., Moll, M., and Kavraki, L. E. (2019). How much do unstated problem constraints limit deep robotic reinforcement learning? *arXiv preprint arXiv:1909.09282*.

[Li et al., 2020a] Li, B., Lu, T., Li, J., Lu, N., Cai, Y., and Wang, S. (2020a). Acder: Augmented curiosity-driven experience replay. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4218–4224. IEEE.

[Li et al., 2020b] Li, R., Jabri, A., Darrell, T., and Agrawal, P. (2020b). Towards practical multi-object manipulation using relational reinforcement learning. In *2020 ieee international conference on robotics and automation (icra)*, pages 4051–4058. IEEE.

[Li et al., 2020c] Li, Z., Ma, H., Ding, Y., Wang, C., and Jin, Y. (2020c). Motion planning of six-dof arm robot based on improved ddpg algorithm. In *2020 39th Chinese Control Conference (CCC)*, pages 3954–3959. IEEE.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Liu and Liu, 2021] Liu, S. and Liu, P. (2021). A review of motion planning algorithms for robotic arm systems. In *RiTA 2020: Proceedings of the 8th International Conference on Robot Intelligence Technology and Applications*, pages 56–66. Springer.

[Lobbezoo et al., 2021] Lobbezoo, A., Qian, Y., and Kwon, H.-J. (2021). Reinforcement learning for pick and place operations in robotics: A survey. *Robotics*, 10(3):105.

[Lucchi et al., 2020] Lucchi, M., Zindler, F., Mühlbacher-Karrer, S., and Pichler, H. (2020). robo-gym–an open source toolkit for distributed deep reinforcement learning on real and simulated robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5364–5371. IEEE.

[Luo et al., 2020] Luo, S., Kasaei, H., and Schomaker, L. (2020). Accelerating reinforcement learning for reaching using continuous curriculum learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[Mahmood et al., 2018] Mahmood, A. R., Korenkevych, D., Komer, B. J., and Bergstra, J. (2018). Setting up a reinforcement learning task with a real-world

robot. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4635–4640. IEEE.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA. PMLR.

[Mohammed et al., 2022] Mohammed, M. Q., Kwek, L. C., Chua, S. C., Al-Dhaqm, A., Nahavandi, S., Eisa, T. A. E., Miskon, M. F., Al-Mhiqani, M. N., Ali, A., Abaker, M., et al. (2022). Review of learning-based robotic manipulation in cluttered environments. *Sensors*, 22(20):7938.

[Moll et al., 2017] Moll, M., Kavraki, L., Rosell, J., et al. (2017). Randomized physics-based motion planning for grasping in cluttered and uncertain environments. *IEEE Robotics and Automation Letters*, 3(2):712–719.

[Nakanishi et al., 2008] Nakanishi, J., Cory, R., Mistry, M., Peters, J., and Schaal, S. (2008). Operational space control: A theoretical and empirical comparison. *The International Journal of Robotics Research*, 27(6):737–757.

[Orocos,] Orocos. Orocos kdl. `https://github.com/orocos/orocos_kinematics_dynamics`. Accessed: [Insert Date].

[Plappert et al., 2018] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*.

[Qin et al., 2019] Qin, B., Gao, Y., and Bai, Y. (2019). Sim-to-real: Six-legged robot control with deep reinforcement learning and curriculum learning. In *2019*

4th International Conference on Robotics and Automation Engineering (ICRAE), pages 1–5. IEEE.

[Raffin et al., 2021] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, 22(1):12348–12355.

[Schulman et al., 2015a] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

[Schulman et al., 2015b] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

[Shahid et al., 2020] Shahid, A. A., Roveda, L., Piga, D., and Braghin, F. (2020). Learning continuous control actions for robotic grasping with reinforcement learning. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 4066–4072. IEEE.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr.

[Spong and Vidyasagar, 2008] Spong, M. W. and Vidyasagar, M. (2008). *Robot dynamics and control*. John Wiley & Sons.

[Števo et al., 2014] Števo, S., Sekaj, I., and Dekan, M. (2014). Optimization of robotic arm trajectory using genetic algorithm. *IFAC Proceedings Volumes*, 47(3):1748–1753.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Szep et al., 2022] Szep, M., Lauenburg, L., Farkas, K., Su, X., and Zang, C. (2022). Reinforcement learning for solving robotic reaching tasks in the neurorobotics platform. *arXiv preprint arXiv:2210.17138*.

[Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE.

[Væhrens et al., 2022] Væhrens, L., Álvarez, D. D., Berger, U., and Bøgh, S. (2022). Learning task-independent joint control for robotic manipulators with reinforcement learning and curriculum learning. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1250–1257. IEEE.

[Weber and Schmidt, 2021] Weber, J. and Schmidt, M. (2021). An improved approach for inverse kinematics and motion planning of an industrial robot manipulator with reinforcement learning. In *2021 Fifth IEEE International Conference on Robotic Computing (IRC)*, pages 10–17. IEEE.

[Zhu and Hu, 2018] Zhu, Z. and Hu, H. (2018). Robot learning from demonstration in robotic assembly: A survey. *Robotics*, 7(2):17.

Appendix A

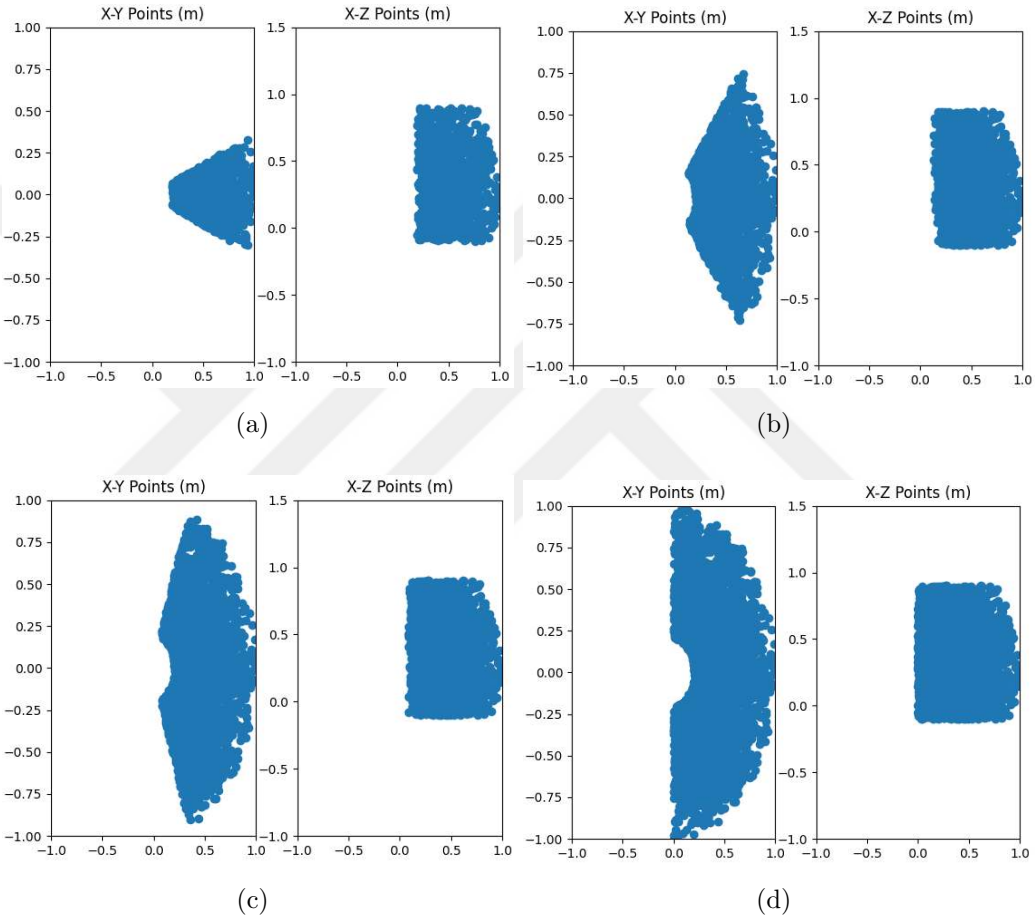


Figure A.1: Regions for Jaco 7-Dof (a) First, (b) Second, (c) Third, (d) Fourth

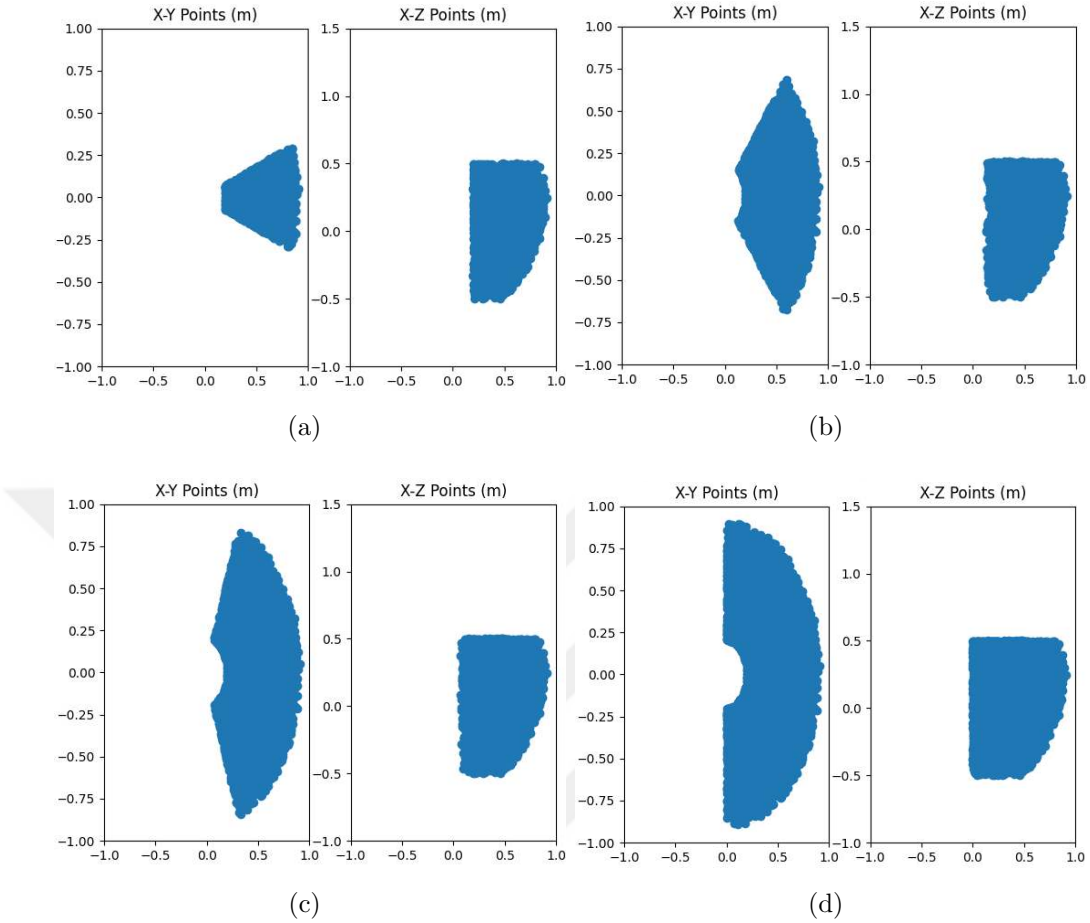


Figure A.2: Regions for Jaco 6-Dof (a) First, (b) Second, (c) Third, (d) Fourth

. .

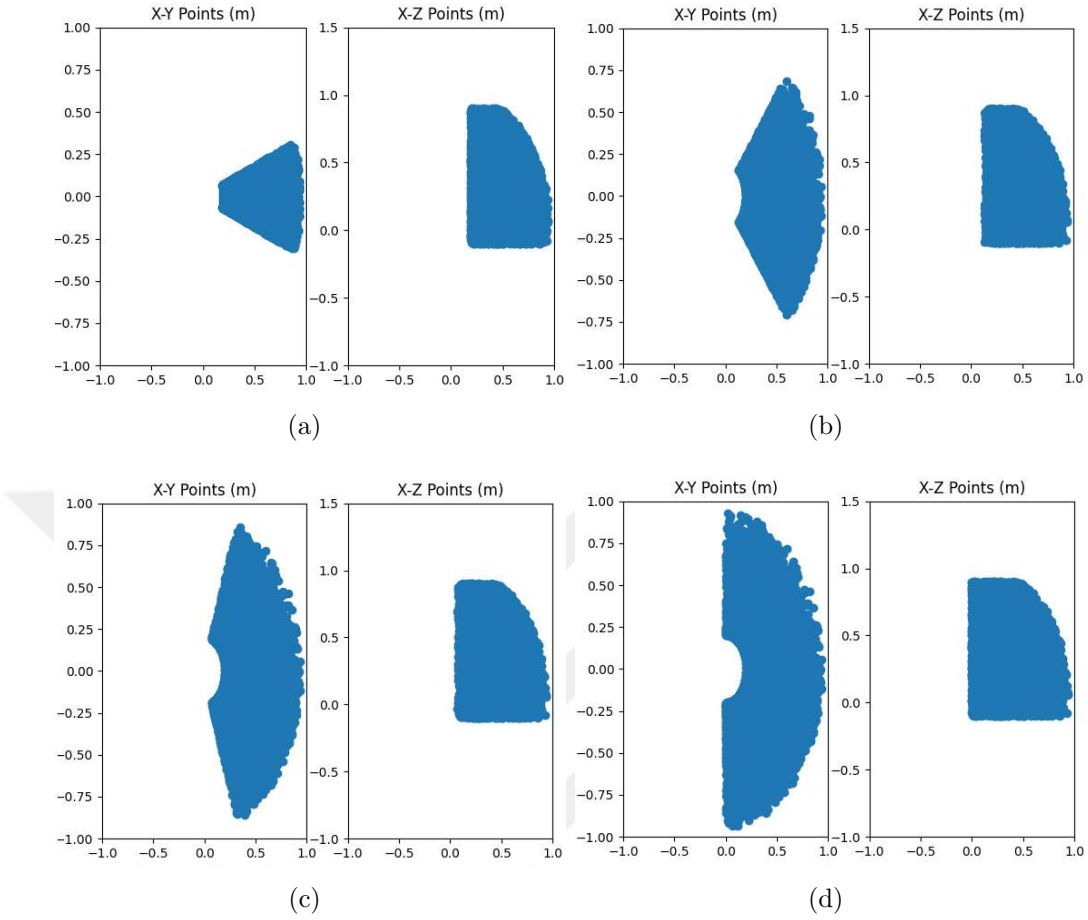
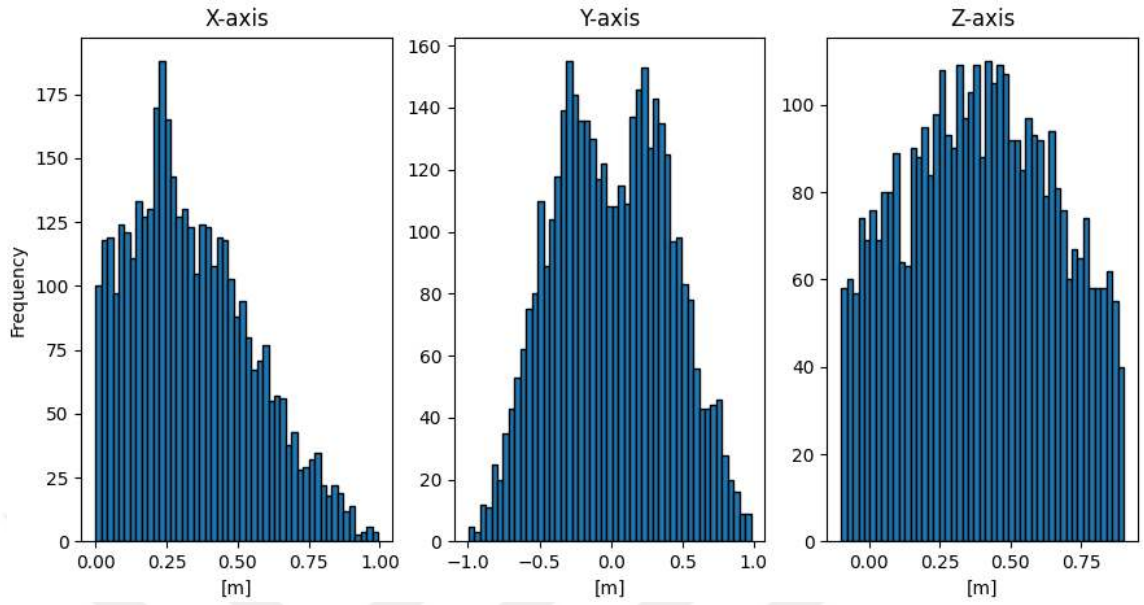
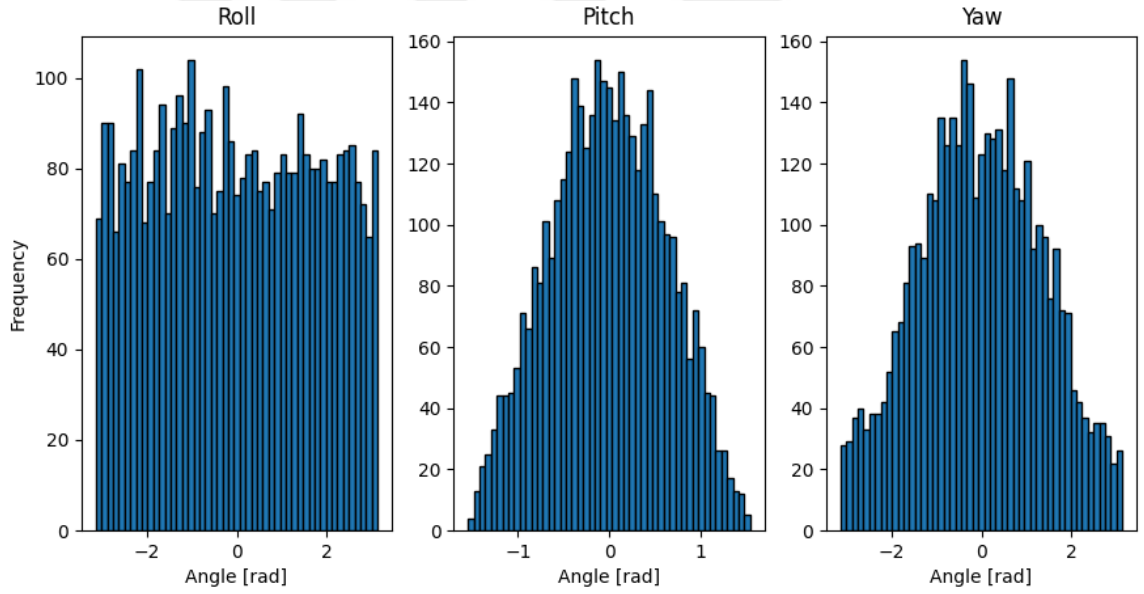


Figure A.3: Regions for UR5 6-Dof (a) First, (b) Second, (c) Third, (d) Fourth

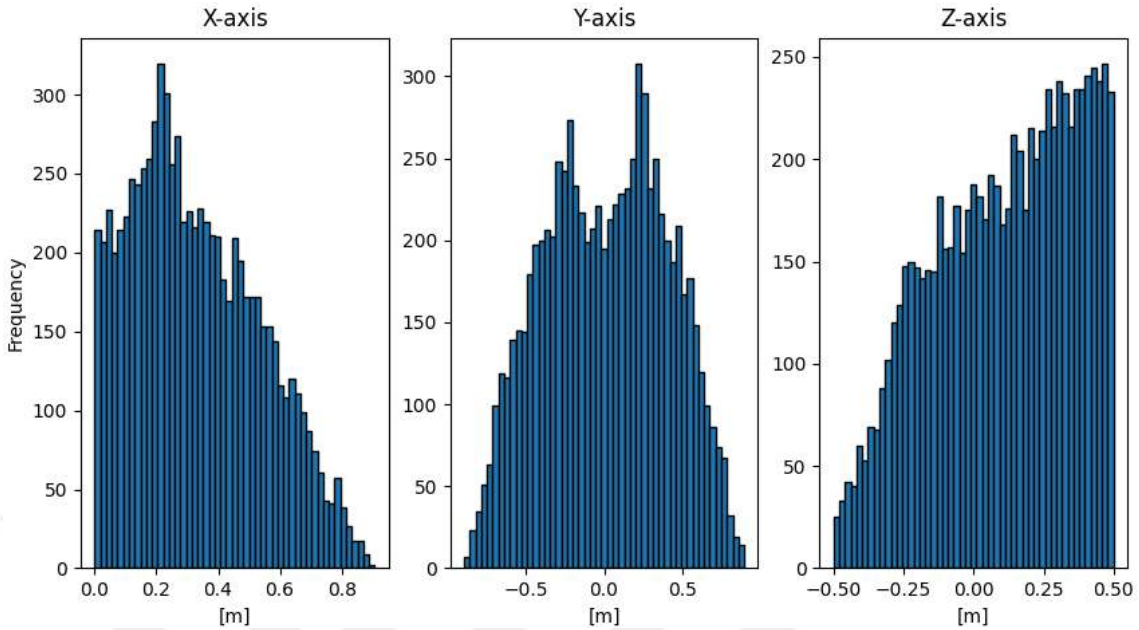


(a)

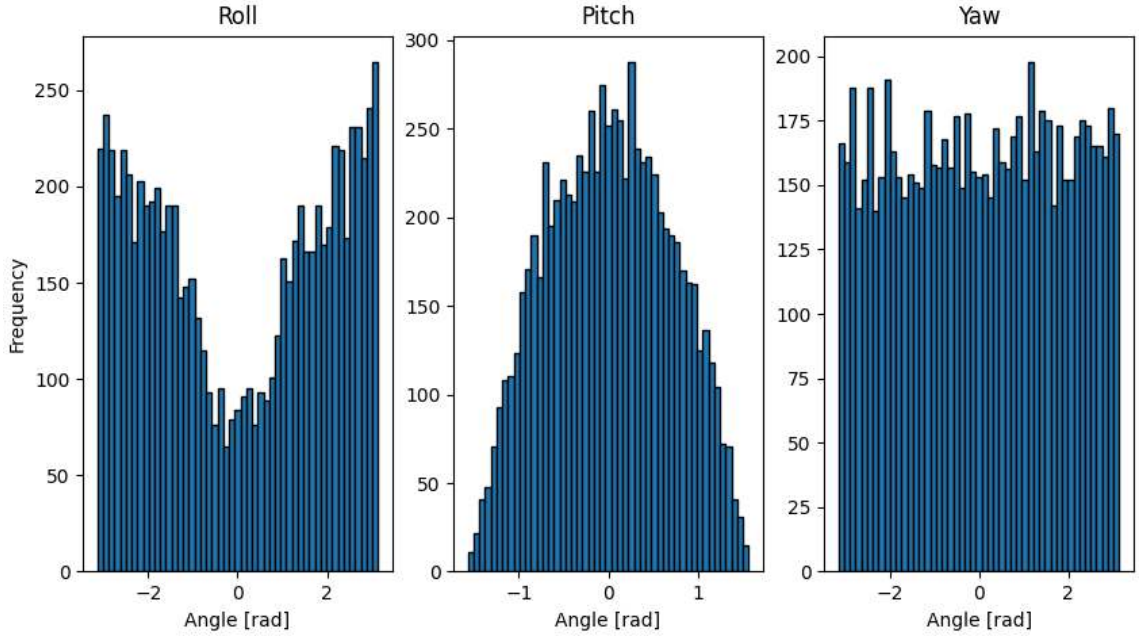


(b)

Figure A.4: Histogram of (a) Position space, (b) Orientation space. The position and orientation space of the *main workspace* for Jaco 7-DoF. Each plot consists of 100 bins to show the frequency of the values. The range of the position values are already described in Table 6.3. The roll and yaw angles are in the interval of $[-\pi, \pi]$ radians. And the pitch angles is in the $[-\pi/2, \pi/2]$ radians due to the kinematic structure of the robot.

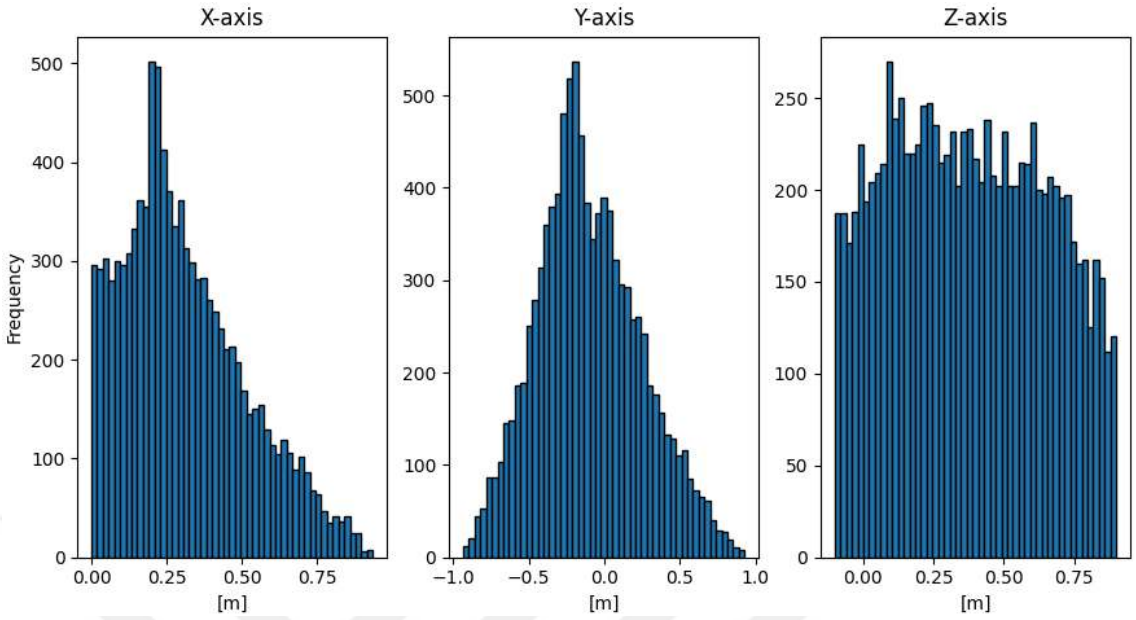


(a)

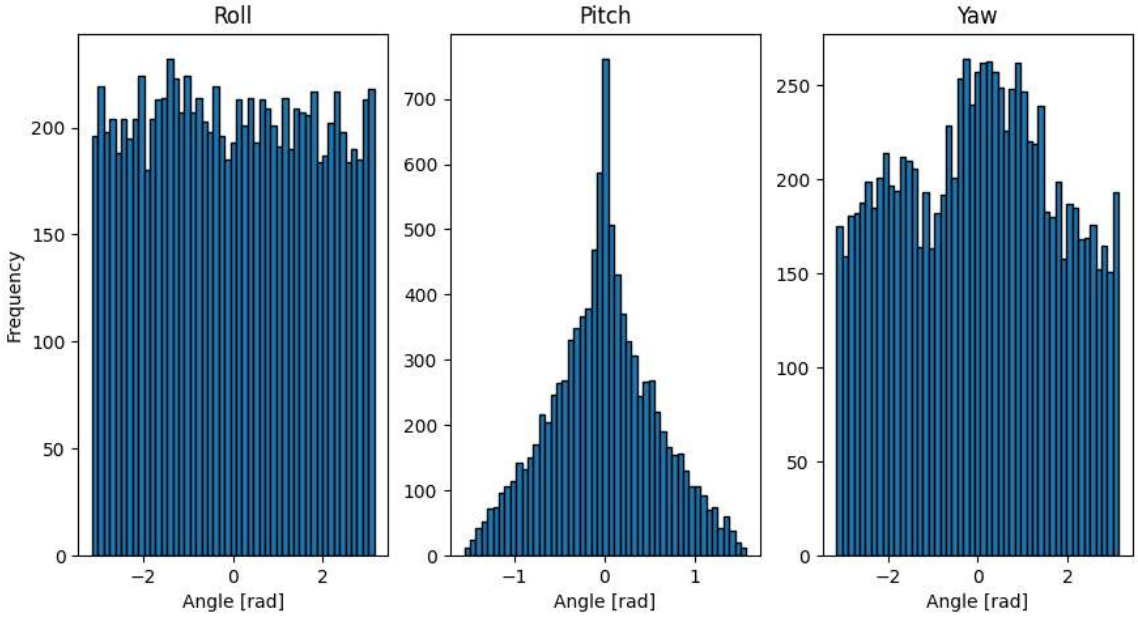


(b)

Figure A.5: Histogram of (a) Position space, (b) Orientation space. The position and orientation space of the *main workspace* for Jaco 6-DoF. Each plot consists of 100 bins to show the frequency of the values. The range of the position values are already described in Table 6.3. The roll and yaw angles are in the interval of $[-\pi, \pi]$ radians. And the pitch angles is in the $[-\pi/2, \pi/2]$ radians due to the kinematic structure of the robot.



(a)



(b)

Figure A.6: Histogram of (a) Position space, (b) Orientation space. The position and orientation space of the *main workspace* for UR5. Each plot consists of 100 bins to show the frequency of the values. The range of the position values are already described in Table 6.3. The roll and yaw angles are in the interval of $[-\pi, \pi]$ radians. And the pitch angles is in the $[-\pi/2, \pi/2]$ radians due to the kinematic structure of the robot.