UNIVERSITÉ LIBRE DE BRUXELLES

VRIJE UNIVERSITEIT BRUSSEL

Master Thesis

Object Pose Estimation on Embeded Devices

Mayur Ashok Sonawane

BRUFACE
BRUSSELS FACULTY OF ENGINEERING

Professor: Adrian MUNTEANU
Professor: Bram VANDERBORGHT
Assistant: Leandro Di Bella

Abstract

This thesis addresses the development of a 4 Degrees of Freedom (DOF) robotic arm capable of independently locating and tracking a moving cube in a dynamic, unknown environment. Traditional control approaches, which rely on pre-programmed instructions and exact environmental models, can fail in unanticipated situations. To solve these constraints, this study combines Reinforcement Learning (RL), notably Proximal Policy Optimization (PPO), with modern computer vision methods.

The study explores three distinct training setups: (1) baseline training without visual feedback, where the robotic arm relies solely on joint angles obtained through inverse kinematics; (2) training with visual feedback provided by a camera mounted on the end effector, combining visual and simulation data; and (3) training utilizing the YOLO (You Only Look Once) algorithm for real-time cube detection. Each setup aims to evaluate the impact of different sensory inputs and detection methods on the robot's performance in locating the cube's position.

The methodology involves implementing the PPO algorithm on the robotic arm within the PyBullet simulation environment and integrating YOLO for enhanced visual perception. Results demonstrate significant improvements in the robotic arm's ability to accurately find the position of the cube and adapt its movements accordingly. The robot trained with YOLO-based visual feedback showed superior performance in identifying the cube's position and efficiently navigating complex and dynamic environments.

These findings highlight the successful application of RL and computer vision techniques in enabling a robotic arm to autonomously navigate and interact in unknown environments. The insights gained from this research contribute to the advancement of autonomous robotic systems, with potential applications ranging from industrial automation to precision tasks in healthcare. Watch the simulation video at : YouTube Video

Acknowledgments

I want to extend my sincerest gratitude to my supervisor, Dr. Adrian Munteanu, and my co-supervisor, Dr. Bram Vanderborght. Their invaluable guidance, consistent support, and insightful feedback have been instrumental throughout this project. This thesis has significantly benefited from their commitment to academic excellence and their dedication to fostering my academic growth.

I am also profoundly grateful to my guide, Leandro Di Bella, for his meaningful contributions to this research. His insights, ideas, and constructive criticism have played a crucial role in guiding this work to its successful conclusion. His wisdom and support have been vital to my academic journey.

My heartfelt thanks go to my family, whose unwavering encouragement and belief in my abilities have been the foundation of my perseverance. Their sacrifices and constant support have laid the groundwork for my academic endeavors, and for that, I am deeply thankful.

I would also like to extend special thanks to my friends and classmates. Throughout the challenges of this research journey, their companionship, engaging conversations, and words of encouragement have provided both comfort and motivation. Their confidence in me has been a significant driving force, and I am fortunate to have shared this experience with such outstanding individuals.

I must also acknowledge the invaluable assistance of the VUB library staff. The library's resources, support, and welcoming environment were essential in facilitating in-depth research and the completion of this thesis. The behind-the-scenes efforts of the library team are deeply appreciated.

Lastly, I would like to thank everyone who has contributed, directly or indirectly, to the successful completion of this thesis. This work reflects the collective efforts and support of many, and I am humbled by the contributions of all who have played a part, no matter how small, in bringing this project to fruition.

Contents

List of Figures

List of Tables

Chapter I

Introduction

1.1 Background

Robotics has significantly transformed various industries by automating complex and repetitive tasks, thereby enhancing productivity, precision, and efficiency. Robotic arms, which are integral to automation, are particularly noteworthy for their ability to perform a diverse array of functions—from assembling intricate components in manufacturing to conducting delicate surgeries in healthcare. A robotic arm's effectiveness is largely determined by its Degrees of Freedom (DOF), which refer to the number of independent movements it can perform. A 4DOF robotic arm, for example, has four independent movements, allowing it to reach and manipulate objects within a three-dimensional space with a certain level of dexterity.

Traditional methods of controlling robotic arms often rely on pre-programmed instructions and precise models of the operating environment. While effective in structured settings, these methods can be limiting in dynamic and unpredictable environments where the robot must adapt to changing conditions in real-time. Reinforcement Learning (RL) offers a powerful solution by enabling robots to learn optimal actions through continuous interaction with their environment. Unlike pre-programmed approaches, RL allows the robot to explore various actions and learn from the outcomes to maximize a cumulative reward signal.

Proximal Policy Optimization (PPO) is an advanced RL algorithm that has gained prominence due to its balance between simplicity and performance. PPO enhances training stability by employing a surrogate objective function that constrains the policy update step size, thus preventing drastic changes that could destabilize learning. This makes PPO particularly suitable for continuous control tasks, such as those encountered in robotic arms, where smooth and precise movements are crucial.

Alongside advancements in Reinforcement Learning (RL), computer vision has transformed how robots perceive and interpret their surroundings. Integrating vision systems allows robots to comprehend visual inputs, which is crucial for engaging with dynamic environments. Recent progress in deep learning has greatly enhanced the efficiency and precision of object detection algorithms. YOLO (You Only Look Once) is particularly noteworthy in this field, as it excels in real-time object detection with high accuracy. By processing images in a single pass, YOLO can identify multiple objects and pinpoint their exact locations within the frame, making it well-suited for applications that demand quick decision-making.

To effectively train and test robotic control algorithms, simulation environments play a critical role. PyBullet is a popular physics simulation engine that provides a flexible and realistic platform for developing and evaluating robotic systems. Using PyBullet, researchers can simulate the physical interactions of a robotic arm with its environment, including dynamics such as friction, collisions, and gravity. This allows for extensive testing and iteration in a controlled setting before deploying algorithms on real hardware, thereby saving time and resources.

Combining the strengths of RL, computer vision, and simulation can substantially enhance the capabilities of robotic systems. By integrating PPO for control and YOLO for object detection within the PyBullet simulation environment, robots can learn optimal strategies while dynamically adapting to their surroundings through visual feedback. This synergy forms the foundation of the research presented in this thesis, which focuses on developing a 4DOF robotic arm capable of first finding the position of a moving cube in an unknown environment and then following the cube.

1.2 Problem Statement

The core challenge addressed in this thesis is the development and training of a 4DOF robotic arm to autonomously follow a moving cube. The goal is to evaluate how different sensory inputs and object detection methods affect the robot's performance. The investigation is divided into three training setups to provide a comprehensive analysis:

- **Baseline Training without Visual Feedback**: In this scenario, the robotic arm is trained using PPO without any visual input. Instead of relying on proprioceptive sensors, the robot uses joint angles obtained through inverse kinematics to follow the cube. The position of the cube is provided directly by the simulation. This setup establishes a baseline performance, highlighting the capabilities and limitations of the PPO algorithm in the absence of external sensory input.
- **Training with Visual Feedback**: This approach integrates a camera mounted on the

end effector of the robotic arm. The visual data from the camera is combined with the simulation data and used as input for the PPO algorithm. The aim is to assess how the combination of visual and simulation feedback impacts the robot's ability to find the position of and then follow the cube, providing insights into the advantages and challenges of incorporating visual perception along with simulation data.

- **Training with YOLO-based Cube Detection**: In the final setup, the YOLO object detection algorithm is used to identify and track the cube in real-time. The detected position of the cube is fed into the PPO algorithm, enabling the robot to find the position of and then follow the cube based on its visual perception. This setup explores the effectiveness of combining real-time object detection with reinforcement learning, particularly in dynamic and unpredictable environments.

In all setups, the robot first focuses on finding the position of the cube in the unknown environment and then follows the cube, without considering the orientation of the cube. Accurately determining the cube's position is crucial for the robot to effectively track and follow it.

1.3 Objectives

The primary objectives of this research are:

- **Implementation and Validation**: To implement the PPO algorithm for controlling a 4DOF robotic arm and validate its performance in a baseline scenario without visual feedback. This involves defining the state and action spaces, designing an appropriate reward function, and conducting extensive training to achieve reliable performance.
- **Impact Analysis of Visual Feedback**: To evaluate the influence of visual feedback from a camera mounted on the end effector on the robot's performance. This includes processing the visual data, integrating it with the simulation data into the PPO framework, and analyzing the robot's ability to find the position of and then follow the cube based on this combined input. The goal is to understand how visual perception affects the robot's behavior and performance.
- **Effectiveness of YOLO Integration**: To investigate the effectiveness of incorporating YOLO for real-time cube detection and its impact on the robot's tracking performance. This involves modifying the PPO algorithm to utilize the cube's detected position, assessing the robot's ability to find the position of and then follow the cube in dynamic environments, and comparing the results with the other two setups.

Accurately determining the cube's position in the unknown environment is crucial for the robot

to effectively track and follow it.

1.4 Contributions

This thesis makes several significant contributions to the fields of robotics, reinforcement learning, and computer vision:

- **Baseline Implementation**: Provides a comprehensive implementation of PPO for controlling a 4DOF robotic arm without any visual feedback. This serves as a foundational reference for evaluating the impact of sensory inputs on robotic control and establishes a performance benchmark.
- **Visual Feedback Integration**: Offers detailed insights into the integration of visual feedback from a camera mounted on the end effector. The analysis includes the effects of visual input on the robot's performance, providing a deeper understanding of how visual perception, combined with simulation data, influences robotic behavior and decision-making.
- **YOLO-based Object Detection**: Demonstrates the integration of YOLO for real-time object detection within the PPO framework. This contribution highlights the benefits and challenges of combining vision-based object detection with reinforcement learning in dynamic environments, providing a blueprint for future research in this area.
- **Use of PyBullet Simulation**: Showcases the use of PyBullet simulation to create a realistic and flexible platform for developing and testing the robotic control algorithms. This allows for extensive experimentation and refinement in a controlled environment before transitioning to real-world applications.

Chapter II

Literature Review

Robotics has experienced a remarkable evolution, transitioning from basic automated devices to sophisticated systems that can execute complex tasks with accuracy and adaptability. This progress has largely been fueled by advancements in control strategies, which are essential for facilitating seamless interactions between robots and their environments. While traditional control methods have proven effective, they often struggle with the complexities and dynamic aspects of contemporary robotic applications. As a result, researchers have been exploring this new approaches like Learning-Based Inverse Kinematics (IK), Model Predictive Control (MPC), and Reinforcement Learning (RL) to address these challenges and improve performance.

Inverse Kinematics (IK) is fundamental in robotics, involving the calculation of joint angles required to position a robot's end effector at a desired location. Traditional IK methods have been widely used but often face significant challenges when applied to robots with high degrees of freedom or in environments with complex constraints. These challenges include computational inefficiency, difficulty in real-time applications, and the need for precise models. To overcome these limitations, researchers have developed Learning-Based IK methods that leverage machine learning techniques, particularly deep learning, to enhance the flexibility, efficiency, and accuracy of IK solutions.

The study "Robust Inverse Kinematics with High-Dimensional Robot Arms Using Deep Learning[9]" introduces the Selective Inverse Kinematics (SIK)[9] framework, which utilizes deep neural networks to solve IK problems for high-dimensional robotic arms. This framework offers multiple-angle solutions for any given pose across the robot's workspace, achieving sub-centimeter accuracy. Such precision is crucial for real-time applications like obstacle avoidance and posture imitation. However, SIK's requirement for extensive computational resources during pre-training presents challenges, particularly when scalability and real-time performance are essential. This limitation highlights the need for methods that balance computational de-

mands with real-time application, a balance we aim to achieve by integrating RL with IK in this thesis.

Similarly, the paper "Adaptive Neural Network-Based Inverse Kinematics for Real-Time Applications[6]" explores the use of feed-forward neural networks to solve IK problems in a three-link planar robotic manipulator. This study demonstrates that neural networks can generate desired trajectories in Cartesian space with high accuracy and efficiency, even when trained on relatively small datasets. The emphasis on scalability and adaptability in this research aligns closely with our thesis's goal of developing a control solution that can efficiently adapt to various configurations and constraints in real-time.

As robotic systems become increasingly complex, the need for control strategies that can manage multiple constraints and optimize performance in real-time becomes more pronounced. Model Predictive Control (MPC) is particularly effective in environments where future state predictions and the management of multiple constraints are critical. MPC's ability to handle nonlinear dynamics and predict future states based on current information makes it a powerful tool for robotic control.

The first paper, "Kinematic-Model-Free Predictive Control for Robotic Manipulator Target Reaching With Obstacle Avoidance[2]" exemplifies how MPC can effectively control robotic manipulators without relying on predefined kinematic or dynamic models. This adaptability is essential for our thesis, where the robotic arm must perform in dynamic settings. However, the computational complexity associated with real-time optimization in MPC, as highlighted in this study, presents significant challenges, especially in fast-changing environments where quick decision-making is essential.

Expanding on MPC's application, the study "Model Predictive Control Design of a 3-DOF Robot Arm Based on Recognition of Spatial Coordinates[23]" integrates nonlinear MPC with visual recognition techniques to optimize the control of a three-degree-of-freedom (DOF) robotic arm. This approach demonstrates how advanced MPC techniques can significantly enhance control precision, directly supporting the objectives of our thesis. However, the increased computational complexity observed in this approach emphasizes the trade-offs between control accuracy and computational efficiency, which our research aims to address by integrating RL with IK.

Additionally, the study "Model Predictive Trajectory Tracking Control of 2 DoFs SCARA Robot under External Forces Applied to the Tip Along the Trajectory[11]" investigates the use of MPC for maintaining precise trajectory tracking in a SCARA robot despite external disturbances such as load variations and friction. While MPC can maintain accuracy under these conditions, the increased demand for input torques and computational resources highlights

the limitations of MPC in real-time applications, further justifying our exploration of RL as a complementary approach.

While MPC offers robust solutions for constraint management and performance optimization, its reliance on accurate models and high computational demands can be restrictive in dynamic and unstructured environments. This limitation is where Reinforcement Learning (RL), particularly Deep Reinforcement Learning (DRL), provides a compelling alternative. RL's ability to learn optimal control strategies through interaction with the environment makes it particularly suitable for applications where the environment is unpredictable or complex.

The paper "Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focused Mini-Review[13]" highlights the potential of DRL in enhancing robotic manipulation. The review discusses advancements in DRL algorithms, such as Deep Q-Networks (DQN) and Deep Deterministic Policy Gradient (DDPG), which have significantly improved the robustness and efficiency of robotic control. However, the challenges of sample efficiency and generalization, highlighted in this review, are critical when deploying DRL in real-world applications, such as those in our thesis. These challenges underscore the importance of selecting an RL algorithm that offers both stability and adaptability in dynamic environments, which is crucial for the success of our research.

In "Robotic Arm Control and Task Training through Deep Reinforcement Learning[7]" , the effectiveness of different DRL algorithms, including Vanilla Policy Gradient (VPG), Trust Region Policy Optimization (TRPO), DDPG, and DQN-NAF, is compared in the context of robotic arm control. This study is particularly relevant to our thesis, as it examines the impact of various reward functions and hyperparameters on the performance of these algorithms. The findings suggest that TRPO, a variant of PPO, offers a balance between learning efficiency and stability, making it a strong candidate for our research. The challenges associated with DDPG, such as instability and difficulty in hyperparameter tuning, further highlight the need for a more stable and efficient algorithm like PPO.

Moreover, the study "Variational Quantum Soft Actor-Critic for Robotic Arm Control[1]" introduces a novel approach by integrating quantum computing with the Soft Actor-Critic (SAC) algorithm to enhance DRL's learning efficiency. While this research focuses on quantum-enhanced DRL, the underlying principle of improving learning efficiency is directly applicable to our thesis. The ability to reduce the number of parameters needed for training, as demonstrated in this study, is crucial for developing a control strategy that balances performance with computational demands in real-time applications.

Furthermore, the challenges of transferring DRL policies from simulated environments to real-world applications are explored in "Towards Vision-Based Deep Reinforcement Learning for

Robotic Motion Control[22]". This study uses DQN for robotic motion control based on visual input, demonstrating DRL's potential to handle tasks requiring real-time interpretation of sensory data. However, the difficulties in transferring learned policies from simulation to reality due to discrepancies between simulated and real-world inputs highlight the need for robust DRL architectures that can generalize across different environments. This consideration is central to our thesis, as we aim to apply RL in practical scenarios where the robotic arm must operate reliably under varying conditions.

Given the strengths and limitations of the methods discussed, integrating Reinforcement Learning (RL) with Inverse Kinematics (IK) is identified as the most suitable approach for pose estimation and control of a 4DOF robotic arm in our thesis. RL's adaptability and learning capabilities are essential for enabling the robotic arm to operate effectively in dynamic environments, where traditional methods like MPC might struggle due to their reliance on precise system models.

IK, on the other hand, provides the necessary framework to calculate the joint configurations required to achieve the desired positions and orientations of the robotic arm. By combining RL with IK, we can develop a system that learns and adapts in real-time while ensuring precise control over the robot's movements. This integration is particularly important for our thesis, as it addresses the dual challenges of pose estimation and control in an efficient and scalable manner.

In proceeding with our research, we carefully considered the relevant literature to identify the most suitable RL algorithms for our approach. Initially, Deep Q-Networks (DQN) seemed like a viable option due to its success in various DRL contexts. However, as highlighted in the study "Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control[22]" , DQN's limitations in handling continuous action spaces and its instability during training posed significant challenges, making it less suitable for our application.

Instead, we turned our attention to Proximal Policy Optimization (PPO), which emerged as a more appropriate choice. PPO's design allows it to handle continuous action spaces effectively, providing the smooth and precise control necessary for the tasks involved in our thesis. Moreover, the study "Robotic Arm Control and Task Training through Deep Reinforcement Learning[7]" suggests that PPO, and its variants like TRPO, are more stable and efficient in learning, making them better suited for the dynamic and high-dimensional environments we are working with.

To empirically validate our choice of PPO over DQN, we draw on insights from Chang's[21] well-known CartPole experiment, which provides valuable lessons for understanding the strengths and weaknesses of these algorithms in reinforcement learning contexts. Chang's experiment did

not involve a robotic arm but rather focused on the CartPole problem, a classic reinforcement learning task where the goal is to balance a pole on a moving cart by applying forces to the cart.

In the CartPole experiment, DQN and PPO were tested on their ability to learn the balancing task. DQN, designed to handle discrete action spaces, showed some initial success but quickly encountered significant challenges. DQN required over 3 hours of training to achieve a policy that could stabilize the pole, but even then, the pole remained balanced for only 14 seconds by the 120th episode. This performance indicated that while DQN could eventually learn the task, its training was slow, unstable, and prone to producing suboptimal policies. The instability was evidenced by large fluctuations in the rewards across episodes, making DQN less reliable for tasks requiring consistent performance.

On the other hand, PPO demonstrated clear advantages in the CartPole experiment. Designed to handle continuous action spaces, PPO quickly learned an effective policy within just 7 minutes and 34 seconds. By the 120th episode, PPO was able to balance the pole for 54 seconds, significantly outperforming DQN. The stability of PPO's learning process was much higher, with rewards varying minimally across episodes, indicating that PPO was more consistent and efficient in learning compared to DQN as illustrated in Figure II.1. Additionally, PPO required fewer interactions with the environment to achieve these results, demonstrating higher sample efficiency, which is crucial for tasks where rapid learning and adaptation are needed.

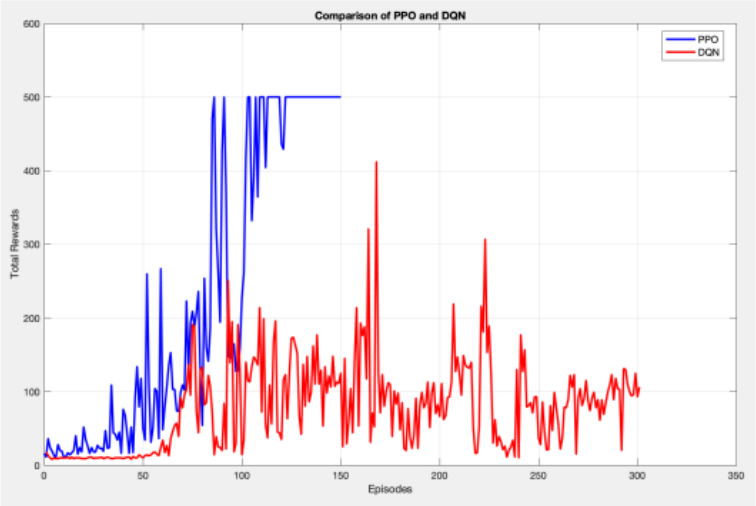

Figure II.1: Training Reward for CartPole DQN and PPO.[21]

The insights from Chang's CartPole experiment, combined with our focus on robotic control, justify the selection of PPO over DQN for our thesis. Integrating PPO with Inverse Kinematics (IK) leverages the strengths of both approaches to tackle the challenges of pose estimation

and control in a 4DOF robotic arm. PPO's reinforcement learning capabilities are well-suited for continuous control tasks, enabling the robotic arm to make precise, real-time adjustments based on feedback from its environment. Meanwhile, IK provides the necessary precision in calculating joint configurations, ensuring the arm achieves desired poses accurately. This combination allows the system to adapt dynamically, maintaining high precision while responding to environmental changes, making it ideal for the dual objectives of pose estimation and control in our thesis.

Building on existing literature and empirical studies, this thesis introduces key innovations in robotic control. First, integrating PPO with IK for the control of a 4DOF robotic arm represents a significant advancement in combining reinforcement learning with traditional control methods. While RL and IK have been explored separately, our approach uniquely combines PPO's strengths in continuous control with IK's precision.

Second, our thesis extends the application of PPO to more complex and dynamic environments, demonstrating its effectiveness not only in simulations but also in real-world scenarios where the robotic arm must adapt to changing conditions. Through extensive experiments, we provide new empirical evidence highlighting the robustness and adaptability of PPO in practical applications.

Finally, this thesis contributes to the development of autonomous robotic systems by offering a framework that can be applied to various robotic platforms beyond the 4DOF arm. The principles and methodologies developed here can be adapted to other robotic configurations, paving the way for more versatile and intelligent systems capable of autonomous operation in complex environments.

In conclusion, the integration of PPO with IK in this thesis presents a robust and efficient control strategy for real-time robotic tasks. The decision to choose PPO over DQN, supported by a thorough literature review and insights from Chang's experiments, ensures that our approach is both innovative and well-founded. By combining PPO's strengths in continuous control with IK's precision, this thesis advances the capabilities of autonomous robotic systems, setting the stage for future innovations in robotic control strategies.

Chapter III

Theoretical Background

This chapter provides the theoretical foundation necessary for understanding the methodologies used in this research. We explore key concepts in machine learning, with a particular focus on Reinforcement Learning (RL) and its application in robotic control. The chapter delves into the Proximal Policy Optimization (PPO) algorithm, a central component of this study, and discusses its integration with neural networks and simulation environments. By establishing this theoretical groundwork, we set the stage for the practical implementation and experimentation that follow in later chapters.

3.1 Introduction to Machine Learning

Machine learning (ML) represents a subset of the expansive domain of artificial intelligence. Within ML, algorithms are developed utilizing statistical techniques to extract insights from data. This field is broadly categorized into three main branches: supervised learning, unsupervised learning, and reinforcement learning.

3.1.1 Supervised Learning

This category involves discerning underlying structures or patterns within labeled data. Common supervised learning techniques encompass linear regression, support vector machines, and k-nearest neighbors. These methods are proficient at predicting outcomes based on provided examples.

3.1.2 Unsupervised Learning

Algorithms within unsupervised learning focus on identifying patterns within unlabeled data. Among these methods, clustering using k-means is a prominent approach for grouping similar data points based on inherent similarities

3.1.3 Reinforcement Learning

Reinforcement Learning In this branch, algorithms learn by interacting with an environment and making a sequence of decisions to optimize cumulative rewards. It involves learning from experience and is widely utilized in scenarios where decision-making occurs over time, such as in gaming strategies and robotics.

With advancements in computing technology, deep learning has become increasingly prominent, especially in handling large datasets and tackling complex problems. Built upon neural networks, deep learning enables the modeling of more sophisticated function approximations and abstractions than traditional machine learning algorithms. Its flexibility in working with high-dimensional, unstructured data makes deep learning applicable across various domains, including time series forecasting, autonomous driving, natural language processing, and robotic process automation. This shift in approach has driven significant progress in multiple fields, transforming how we address challenges and derive insights from data.

3.2 Reinforcement Learning

In terms of theory and practical application, Reinforcement Learning (RL) is a branch of machine learn ing that is constantly evolving. Algorithms that use reinforcement learning analyze how people behave in different scenarios and figure out how to improve that behavior [14]. As indicated in Figure III.1 RL algorithms can be categorized.

In RL, the agent begins with no specific knowledge about the best actions to take. Through repeated interactions with the environment, the agent learns which actions yield the most positive outcomes and adjusts its behavior accordingly. This process of exploration and learning from mistakes enables the agent to develop increasingly effective strategies to achieve its goals.

The core of reinforcement learning is the development of a policy, which is a strategy for choosing actions that are taken based on the current state of the environment. As the agent accumulates experience, it continuously updates its policy to maximize the cumulative rewards it receives. This approach is particularly useful in complex scenarios where the best course of action isn't immediately clear and needs to be discovered through ongoing interaction.

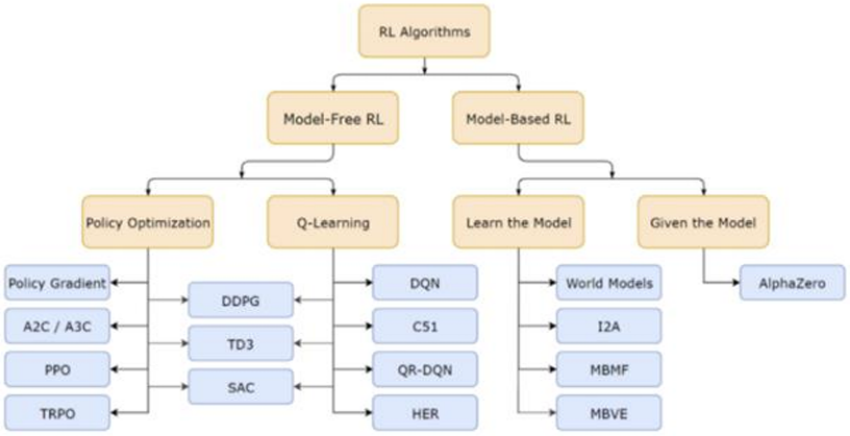

Figure III.1: Reinforcement Learning Classification.[14]

Reinforcement learning has been particularly successful in areas requiring sophisticated decision-making and adaptation. For example, it has achieved impressive results in mastering complex games like Go and chess, controlling autonomous vehicles, and optimizing various robotic tasks. Its ability to generate novel and effective solutions through extensive experimentation makes it a powerful tool for addressing challenging problems and advancing technology.

3.3 Markov Decision Process[14]

A Markov Decision Process (MDP) is a formal framework used to model decision-making problems where an agent interacts with an environment over discrete time steps[14] . This framework helps in understanding and optimizing the agent's behavior to achieve the best possible outcomes. Here's a detailed breakdown of its components:

1. **States (S)**: States represent all the possible situations or configurations that the environment can be in at any given time. For example, in a game, states could include different board configurations or player positions. Each state is denoted as s or s_t for the state at time t. The collection of all possible states is represented by S.

2. **Actions (A)**: Actions are the choices available to the agent in each state. These actions influence how the agent interacts with the environment and consequently how the environment changes. Actions are represented as a or a_t for the action taken at time t. The set of all possible actions is denoted by A.

3. **Transition Probabilities (P)**: These probabilities describe the likelihood of transitioning from one state to another after performing a specific action. For instance, if the agent is in state s and takes action a, the transition probability $P(s' \mid s, a)$ indicates the chance

of moving to state s'. These probabilities help model the uncertainty and dynamics of the environment.

4. **Rewards (R)**: Rewards quantify the immediate benefit or cost of taking a specific action in a given state. This feedback helps the agent assess the desirability of its actions. The reward function can be denoted as $R(s, a)$, representing the reward received after taking action a in state s, or as $R(s, a, s')$ if the reward depends on the transition from state s to state s' as a result of action a.

5. **Policy (π)**: A policy is a strategy that defines how the agent should act in different states. It is a mapping from states to actions and can be deterministic or probabilistic. The policy $\pi(a \mid s)$ indicates the probability of taking action a when the agent is in state s. The policy guides the agent's decision-making process and is crucial for determining how the agent should behave to achieve its goals.

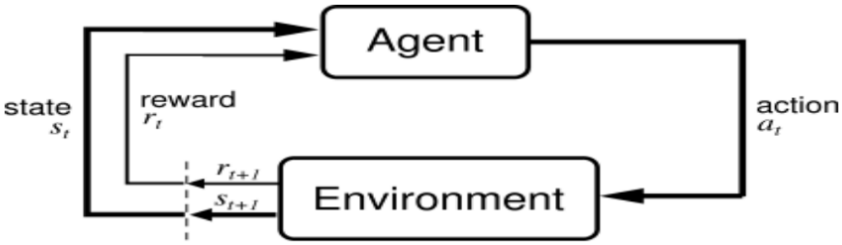

Figure III.2: Agent-Environment Interaction.[14]

The behavior of a Markov Decision Process (MDP) can be described using the Bellman Equation, which connects the value of a state to the expected rewards over time. The Bellman Equation for a policy π is expressed as:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a \mid s) \left[R(s, a) + \gamma \sum_{s' \in S} P(s' \mid s, a) V^{\pi}(s') \right]$$

Here, $V^{\pi}(s)$ represents the value of being in state s under policy π. This value is the sum of the immediate reward $R(s, a)$ for taking action a in state s and the expected future rewards, discounted by a factor γ, from the resulting states s' after taking action a.

The objective in an MDP is to find the optimal policy π^* that maximizes the expected cumulative rewards. To achieve this, we need to determine the optimal action-value function $Q^*(s, a)$. This function represents the maximum expected cumulative rewards obtainable by performing action a in the state s and then adhering to the optimal policy. The Bellman Equation for the

optimal action-value function is given by:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

In this equation, $\max_{a'} Q^*(s',a')$ denotes the maximum value among all possible actions in the subsequent state s'.

By solving the Bellman Equation, we can determine both the optimal policy and the values for states and actions that will yield the highest expected rewards in the MDP.

The advantage function $A^\pi(s,a)$ is another useful measure defined as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)[14]$$

This function measures the relative value of taking action a in state s compared to the average value of being in state s, providing a way to reduce variance in the Q-value function by using the state-value as a baseline.

The ultimate result of applying reinforcement learning is the discovery of the optimal policy, which enables the agent to adapt to its environment and maximize future rewards. The optimal value function is determined by:

$$V^*(s) = \max_\pi Q^\pi(s,a)$$

And the optimal policy itself is defined as:

$$\pi^*(s) = \arg\max_\pi Q^\pi(s,a)$$

3.4 Proximal Policy Optimization (PPO)

Reinforcement Learning (RL) involves an agent gaining decision-making skills through interactions with its surroundings, with the ultimate goal of optimizing its cumulative rewards over time. Proximal Policy Optimization (PPO) is a well-liked reinforcement learning technique that meticulously strikes a balance between taking advantage of existing knowledge and investigating novel possibilities, all the while maintaining stable policy updates. As shown in Figure III.3, PPO employs an actor-critic model in which the actor proposes actions and the critic evaluates their efficacy. This handbook provides a thorough understanding of the Actor-Critic model and PPO concepts, along with thorough explanations and mathematical insights.

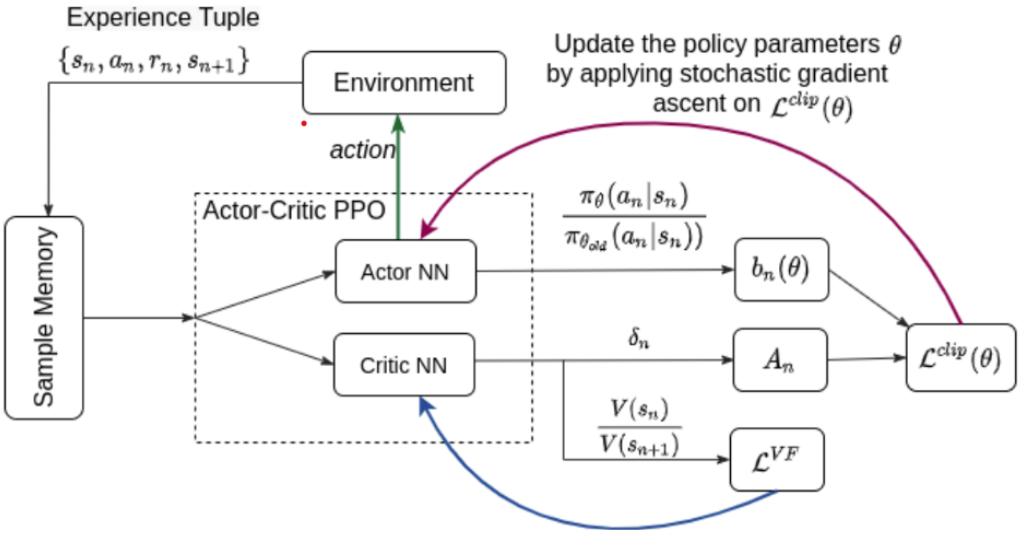

Figure III.3: PPO Architecture.[20]

The Actor-Critic[20] architecture is central to PPO, consisting of two main components: the Actor Network (π_θ) and the Critic Network (V_ϕ). The actor represents the policy, mapping states (s) to actions (a), where the policy $\pi_\theta(a \mid s)$ defines the probability distribution over actions given a state. The main objective of the actor is to maximize the expected cumulative reward. On the other hand, the critic estimates the value function $V(s)$, which represents the expected return from a state. The value function is crucial as it helps evaluate the actions chosen by the actor, thereby providing feedback to improve the policy. The Actor-Critic method leverages this feedback mechanism to guide the actor in iteratively improving its policy.

PPO[17] is a policy gradient method specifically designed to enhance the stability and efficiency of training. It introduces a clipped objective function that restricts the magnitude of policy updates, preventing large changes that could destabilize the learning process. The policy gradient theorem forms the foundation for optimizing policies in RL, with the objective of maximizing

the expected cumulative reward $J(\theta)$:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

Here, τ represents a trajectory (sequence of states, actions, and rewards), γ is the discount factor, and r_t is the reward at time step t. The gradient of $J(\theta)$ with respect to the policy parameters θ is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

where $\hat{A}_t$ is the advantage function, which represents the relative value of an action compared to the expected value. The advantage function $\hat{A}_t$[17] is defined as:

$$\hat{A}_t = Q(s_t, a_t) - V(s_t)$$

In this equation, $Q(s_t, a_t)$ is the action-value function, representing the expected return after taking action a_t in state s_t, and $V(s_t)$ is the value function, representing the expected return from state s_t. The advantage function can be further estimated using the temporal difference (TD) error[17]:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots$$

where:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

and λ is a decay parameter that balances bias and variance.

PPO employs a clipped objective function to ensure stable updates to the policy. The surrogate objective function is expressed as[17]:

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

where:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

is the ratio of probabilities between the new policy and the old policy. The clipping function constrains this ratio within a range defined by $1 - \epsilon$ and $1 + \epsilon$, where ϵ is a hyperparameter. This clipping mechanism prevents the policy from changing too much in a single update, thus ensuring stability during training.

The critic network's goal is to estimate the value function[17] $V(s_t)$, which represents the expected return from a state. The value function is trained by minimizing the mean squared error between the predicted value and the actual return:

$$L_V(\theta) = \mathbb{E}_t \left[\left(V_\theta(s_t) - R_t \right)^2 \right]$$

where R_t is the actual return calculated as:

$$R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

This objective ensures that the critic provides accurate feedback to the actor. To further improve the estimation of the advantage function, PPO often employs Generalized Advantage Estimation (GAE)[17], which reduces variance while maintaining low bias. The GAE formulation is given by:

$$\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

where δ_t is the TD error:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

GAE introduces the parameter λ, which controls the bias-variance trade-off. When λ is 0, GAE reduces to the one-step TD error; when λ is 1, it uses Monte Carlo estimates, incorporating all future rewards.

PPO is conceptually related to Trust Region Policy Optimization (TRPO)[17], which enforces a constraint on the magnitude of the policy update. TRPO optimizes the objective:

$$\max_{\theta} \mathbb{E}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right]$$

subject to the constraint:

$$\mathbb{E}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t) \| \pi_\theta(\cdot \mid s_t) \right] \right] \leq \delta$$

where KL denotes the Kullback-Leibler divergence, and δ is a small positive number. PPO approximates this constraint by using a clipped objective rather than explicitly enforcing a trust region.

The training procedure for the PPO algorithm follows an iterative process to update the actor and critic networks. Initially, trajectories are collected by running the current policy π_θ, which gathers sequences of states, actions, rewards, and next states. Based on these collected trajectories, advantage estimates $\hat{A}_t$ are calculated for each time step. Subsequently, the policy parameters θ are updated by maximizing the PPO objective function $L^{\text{CLIP}}(\theta)$. Alongside this, the value function is updated by minimizing the value loss $L_V(\theta)$. This process is repeated for multiple iterations, gradually improving the policy [17].

3.5 Neural Networks

A neural network acts as an approximation tool when the exact form of a function $f(x) = y$ is either unknown or too complex to compute directly. The goal is to train the neural network to approximate this function. In this scenario, the neural network learns to represent the function as $f^*(x, \theta) = y$, where θ denotes the network's parameters, including the weights and biases[8].

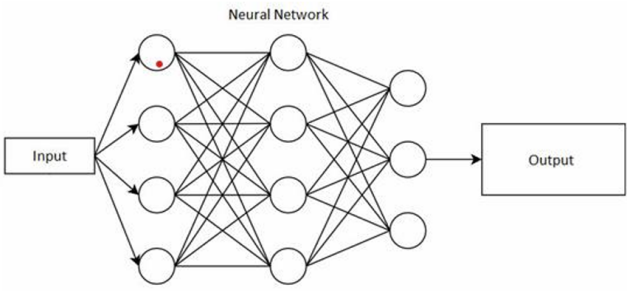

Figure III.4: Neural Networks.[8]

3.6 PyTorch[12]

Facebook AI Research's PyTorch is a potent deep learning framework known for its model construction flexibility, especially with its dynamic computation graphs. PyTorch is particularly well-suited for research circumstances where models are continually evolving, as it permits updates to the computational graph during runtime, which is not possible with static graph frameworks. The robust GPU acceleration of PyTorch complements this flexibility by making it possible to train and deploy large-scale neural networks quickly. The automated differentiation feature of the library makes backpropagation, a crucial step in deep learning model training, easier to implement. PyTorch's ability to easily interact with other Python libraries increases its usefulness in a variety of applications, such as reinforcement learning and natural language processing. Furthermore, PyTorch facilitates the transfer of models to production with TorchScript, guaranteeing a seamless transition for research models.

3.7 PyBullet[15]

A popular library for real-time physics simulations in robotics and reinforcement learning research is PyBullet. It performs exceptionally well in simulating intricate physical interactions, including collision detection, soft body simulations, and rigid body dynamics—all essential for precise robotic modeling. The creation and testing of complex robotic algorithms in a virtual setting is made possible by PyBullet's interaction with machine learning frameworks, which eliminates the need for actual prototypes during the early phases of research. This feature is especially useful for iterative testing and optimization, enabling researchers to make significant improvements to their algorithms before putting them on real robots. In addition, PyBullet is compatible with several robotics platforms and sensors, which makes it an adaptable tool for modeling a variety of robotic applications, from straightforward jobs like object manipulation to more complex scenarios involving multiple interacting robots. as we see in Figure III.5.

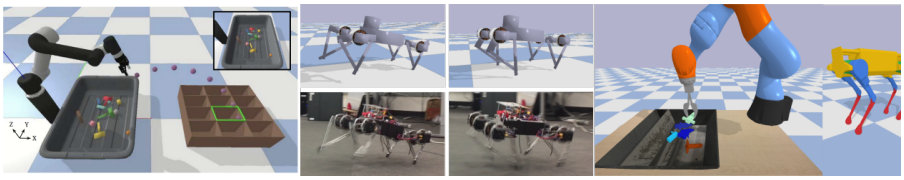

Figure III.5: Bullet Real-Time Physics Simulation.[4]

3.8 SolidWorks[5]

One of the best CAD programs for engineering and design is SolidWorks, which is used extensively to create and test 3D models. It provides a full suite of tools for motion analysis,

finite element analysis (FEA) to assess material characteristics including stress, strain, and thermal behavior, and mechanical system design. Easy modifications and improvements are made possible by SolidWorks' parametric design features, which are crucial for the iterative design processes that are common in engineering projects. When creating robotic components, where exact mechanical design and simulation are essential to guarantee operation and performance, this flexibility is very helpful. SolidWorks further makes it easier to convert CAD models into the Unified Robot Description Format (URDF), allowing these designs to be easily integrated into robotic simulation environments such as PyBullet and Gazebo. This transformation ensures that virtual models closely align with their real-world counterparts, enhancing the reliability of simulations.

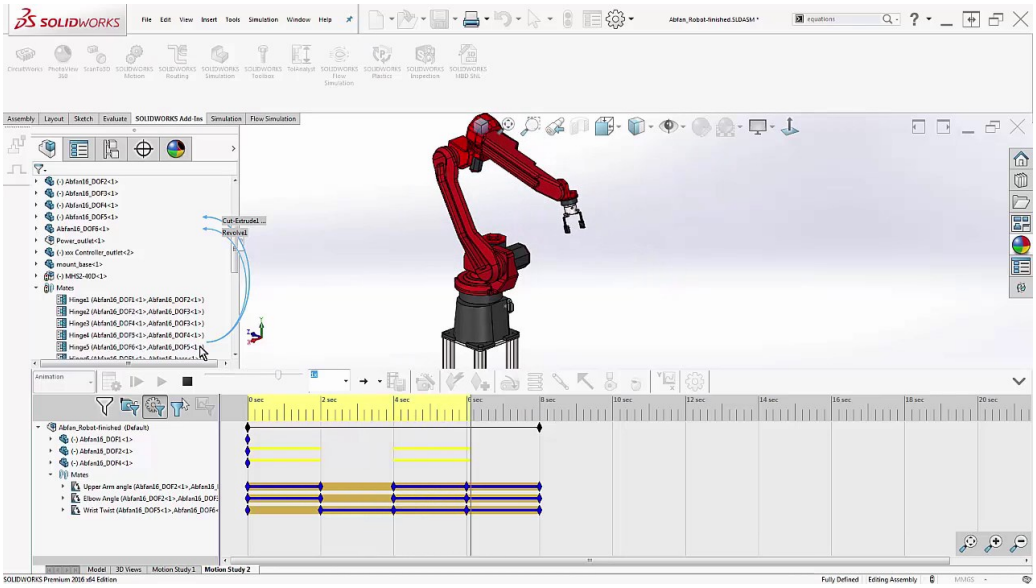

Figure III.6: Robots in Motion Simulation.[18]

3.9 URDF (Unified Robot Description Format)[16]

Within the Robot Operating System (ROS) ecosystem, a robot's physical attributes may be described using an XML language called the Unified Robot Description language (URDF). For precise simulation and control, URDF offers a standardized method for defining the robot's kinematics, dynamics, visual characteristics, and collision models. Through the use of URDF, developers may guarantee that the description of the robot remains constant from design to simulation to real-world deployment. Because of this uniformity, testing and debugging robotic systems in a virtual environment is made easier, freeing up engineers to concentrate on improving control algorithms and sensor integrations. Because of URDF's adaptability, intricate descriptions of complicated robotic systems—including multi-joint configurations and sensor locations—can be written. These descriptions are essential for advanced robotic applications.

3.10 Computer Vision[3]

In order to enable robots to interpret and comprehend visual information from their surroundings, computer vision is an essential area of artificial intelligence. Through a variety of methods for processing, interpreting, and analyzing pictures and videos, it enables computers to perform visual cognition-intensive tasks including item detection, facial recognition, and scene reconstruction. [10]Algorithms for image processing, which improve picture quality and extract pertinent information, and object identification techniques, which recognize and categorize things in an image, are essential parts of computer vision. Deep learning advances have significantly increased computer vision systems' accuracy and efficiency, to be used in surveillance, autonomous driving, or medical imaging applications[10]. This is mostly due to breakthroughs in deep learning. Advancements in technology have made it possible for robots to move and engage with their environment, enabling robots to use visual data to navigate and interact with their environments more effectively.

3.11 YOLO (You Only Look Once)[19]

Modern object identification algorithms like YOLO (You Only Look Once) are renowned for their quickness and precision while analyzing photos in real-time. In contrast to conventional techniques that utilize a sliding window method to identify objects, YOLO creates a grid out of a picture and predicts class probabilities and bounding boxes for every grid cell in a single pass. Because of its design, YOLO can identify objects at fast rates without compromising accuracy, which makes it perfect for real-time processing applications like live video analysis and autonomous cars. YOLO's capacity to identify smaller objects, lower false positives, and boost overall detection accuracy have all improved over several iterations. YOLO is one of the most popular object detection algorithms due to its speed and accuracy in both academic research and industry, particularly in fields where rapid and reliable object detection is essential.

The theoretical insights discussed in this chapter lay the essential groundwork for the research conducted in subsequent chapters. By understanding the principles of Reinforcement Learning and the mechanics of the PPO algorithm, along with their application in robotic control, we are well-prepared to implement and experiment with these techniques. This foundational knowledge is crucial for achieving the research objectives and advancing the development of autonomous robotic systems.

Chapter IV

Simulation and Training Setup

This chapter details the comprehensive setup of the simulation environment and the design of the 4DOF robotic arm used for this study. Key aspects include the creation of the robotic arm model, the setup of various training scenarios—ranging from baseline training without visual feedback to advanced training with YOLO-based real-time object detection—and the use of PyBullet for simulating a dynamic environment. These setups are essential for teaching the robotic arm to autonomously locate and follow a moving cube, ensuring robust performance in preparation for real-world deployment.

4.1 Training Setup

4.1.1 Environment Preparation

In preparing the environment for simulation, a detailed setup was implemented in PyBullet, featuring a 4-DOF robotic arm and a cube. The simulation was carefully configured with accurate physical properties, including mass, friction, joint constraints, and collision parameters, to ensure the robotic arm interacted realistically with the cube. The environmental conditions, such as gravity and lighting, were meticulously adjusted to closely replicate real-world scenarios, ensuring the robot could perform tasks like reaching and manipulating the cube with precision. Furthermore, a virtual camera was mounted on the robot's end effector to capture visual feedback, which was processed through resizing and normalization to create features that were utilized in the machine learning models during the training process.

4.1.2 Baseline Training without Visual Feedback

To establish a baseline performance for the robotic arm, training was conducted using only proprioceptive data, without the inclusion of any visual input (as illustrated in Figure IV.1. The state space was defined to include critical proprioceptive information such as joint angles, velocities, and torques, while the action space was configured to allow the robot to adjust its joint torques or positions. A reward function was designed to incentivize the robot to minimize the distance between itself and the target cube based on proprioceptive feedback alone. Training was performed using the Proximal Policy Optimization (PPO) algorithm, chosen for its robustness and stability in reinforcement learning tasks. During training, multiple episodes were conducted, allowing the robotic arm to learn how to locate and follow the cube in an unknown environment purely through proprioceptive inputs, with careful monitoring and adjustment of hyperparameters to optimize performance.

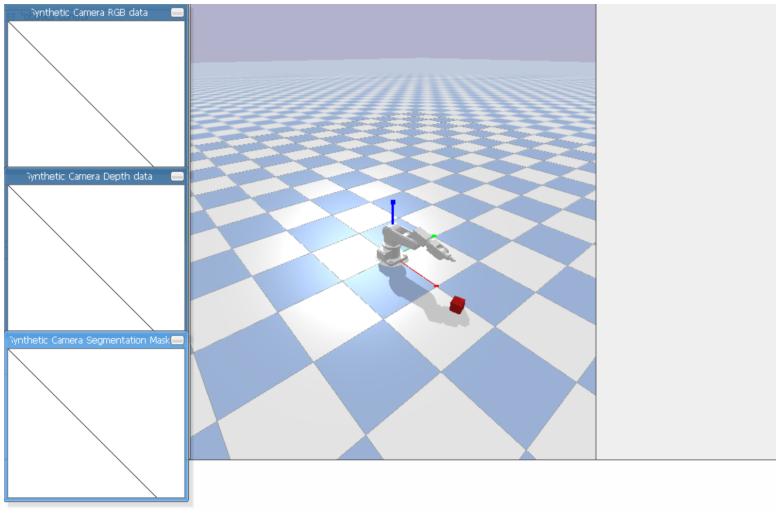

Figure IV.1: Pybullet Simulation Without Visual Feedback.

4.1.3 Training with Visual Feedback

To assess how visual feedback influences the robotic arm's ability to track the cube, training was performed by incorporating visual data captured by a camera attached to the robot's end effector, along with simulation data (as depicted in Figure IV.2. The state space was expanded to include both visual inputs and simulation data, while the action space continued to focus on joint movements. The reward function was adjusted to account for the visual feedback, encouraging the robot to follow the cube using real-time visual information accurately. The training was carried out using the Proximal Policy Optimization (PPO) algorithm, which facilitated the integration of visual and proprioceptive data. Over multiple episodes, the robot learned to locate and track the cube within the environment by effectively utilizing visual and proprioceptive

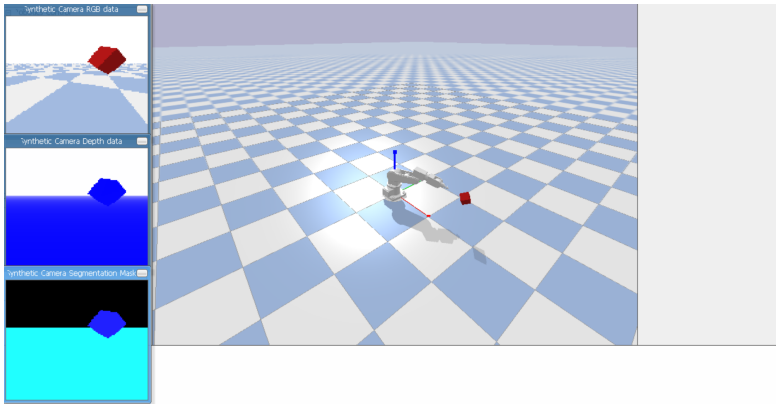

Figure IV.2: Pybullet Simulation With Visual Feedback.

feedback, with the training parameters being fine-tuned to maximize performance.

4.1.4 Training with YOLO-based Cube Detection

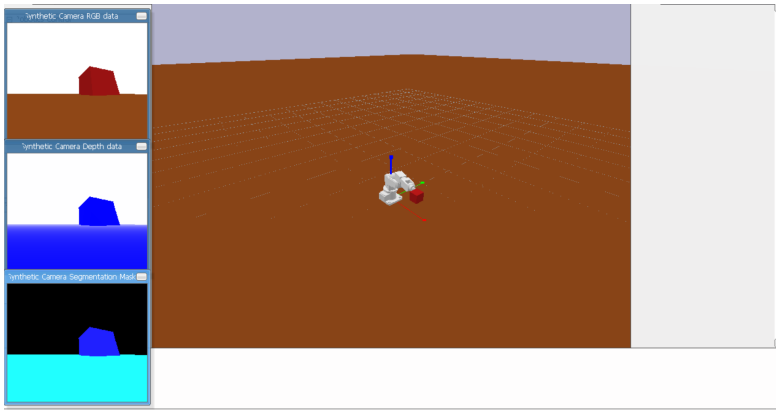

Figure IV.3: Pybullet Simulation with YOLO-based Cube Detection.

In this stage, the training process was advanced by incorporating real-time object detection using the YOLO algorithm to enhance the robotic arm's capability to track and follow the cube (as illustrated in Figure [insert figure number here]). The YOLOv8 model was utilized to detect the cube's position in real-time, and this detection data was integrated into the state space of the training environment. The action space continued to focus on the robot's joint control, similar to the previous setups. The reward function was modified to make use of the real-time data provided by YOLO, thereby encouraging the robotic arm to track the cube based on its detected location accurately. The training was conducted using the Proximal Policy Optimization (PPO) algorithm, which facilitated the seamless integration of the YOLO detection data into the reinforcement learning framework. Through numerous training episodes, the robot progressively improved its ability to follow the cube, with continuous adjustments to optimize performance.

4.2 Robot Design

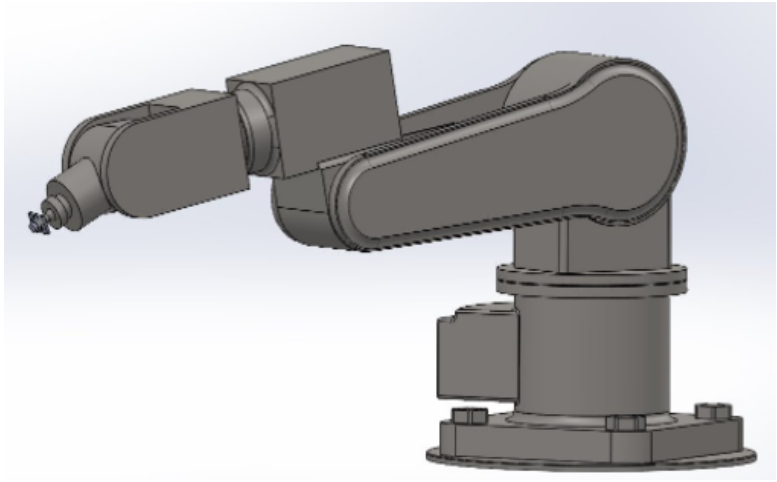

Figure IV.4: 4DOF Robot

The robotic manipulator developed for this project is a 5 Degrees of Freedom (DOF) design, optimized for precise object tracking and manipulation tasks. The robot's structure consists of four active DOFs for movement, each corresponding to a specific joint, and one fixed link that provides stability to the end effector and the camera.

The design process began with detailed modeling in SolidWorks, where the robot's base and links were constructed to ensure stability and a broad range of motion. The links, labeled as *base_Link*, *Empty_Link1*, *Empty_Link2*, *Empty_Link3*, *End_effector*, and *Camera_Link*, were engineered for smooth articulation and robust performance. The end effector, equipped with a camera, was specifically designed to interact accurately with objects, capturing visual data essential for tasks like object detection and tracking.

To translate the physical design into a functional simulation model, the 3D model from SolidWorks was converted into the Unified Robot Description Format (URDF). This process involved meticulously defining each link and joint within the URDF, ensuring that the virtual model mirrored the physical structure. The URDF configuration included detailed descriptions of the dimensions, shapes, and connections of all components, facilitating an accurate and reliable simulation environment in PyBullet.

The joints were configured as follows:

- **Joint1 (Base to Empty_Link1):** A rotational joint with a range of motion from -180° to 180°.
- **Joint2 (Empty_Link1 to Empty_Link2):** A rotational joint with a range of motion from -90° to 90°.

- **Joint3 (Empty_Link2 to Empty_Link3):** A rotational joint with a range of motion from -90° to 90°.
- **Joint4 (Empty_Link3 to End_effector):** A rotational joint with a range of motion from -90° to 90°.
- **Joint5 (End_effector to Camera_Link):** A rotational joint with a range of motion from -90° to 90°.

These joints provide the manipulator with the flexibility required to perform complex tasks within a three-dimensional workspace. The end effector's positioning and the camera's alignment are critical for maintaining the accuracy and efficiency of object detection and manipulation tasks.

The final URDF model was integrated into the PyBullet simulation environment, where the robot's kinematic and dynamic properties were tested extensively. These simulations ensured that the robot could perform as expected under various operational conditions, validating the design and allowing for any necessary adjustments before physical implementation.

So, the simulation and training setups, along with the detailed design of the robotic arm, provide a solid framework for evaluating the robot's tracking capabilities. By integrating different sensory inputs and using PyBullet, we have established a comprehensive testing environment that facilitates precise assessment and iterative improvement. This foundational work, including the careful design of the robotic arm, is critical for ensuring the robot's effective operation in dynamic and unpredictable real-world scenarios.

Chapter V

Designing of Environments and PPO Implementaion

In this chapter, we focus on the design of the simulation environments and the implementation of the Proximal Policy Optimization (PPO) algorithm to control the 4DOF robotic arm. The environments are crafted to progressively challenge the robot's capabilities, starting from basic setups to more complex scenarios involving visual feedback and real-time object detection. The chapter also delves into the technical aspects of integrating PPO with these environments, ensuring that the robotic arm can learn to perform tasks with increasing precision and adaptability.

5.1 Proximal Policy Optimization (PPO) Implementation Breakdown

The agent operates within an environment characterized by a defined state space S and an action space A. The state space includes the observed variables that describe the current condition of the environment, such as joint angles and velocities, while the action space consists of all possible actions that the agent can take, like adjusting the torque of the joints.

The Memory class plays a crucial role in the PPO algorithm by storing episodes of interactions between the agent and the environment. It holds key elements such as actions, states, log probabilities of actions, rewards, and terminal flags indicating the end of an episode. These stored experiences are vital for off-policy updates, allowing the agent to learn from past actions and improve its policy.

The PPO algorithm employs an Actor-Critic architecture, where the Actor-network determines

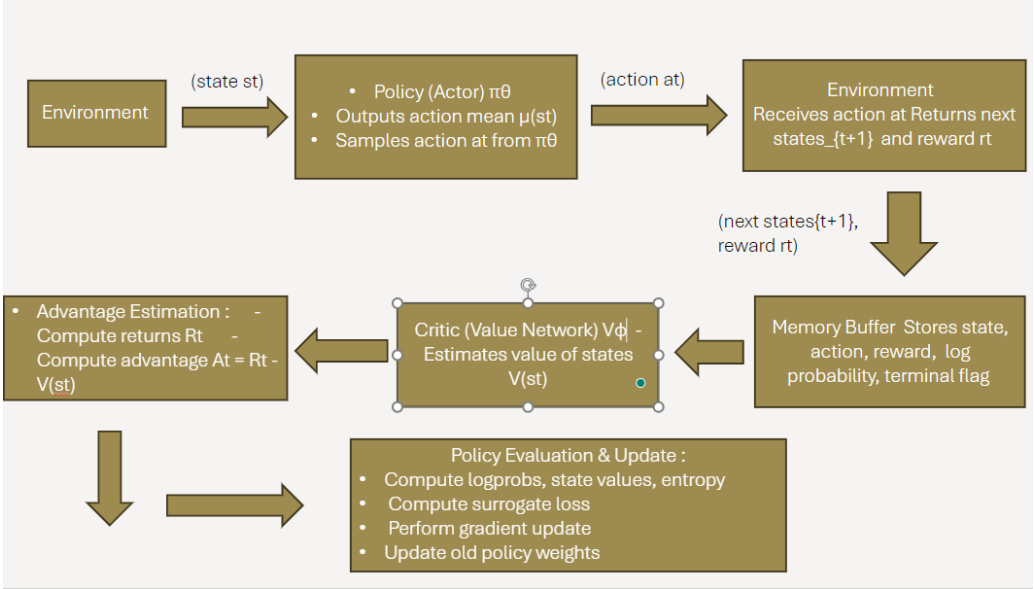

Figure V.1: PPO Workflow

the policy and outputs the mean of a Gaussian distribution over possible actions given the current state s. The network consists of an input layer that receives the state $s \in S$, multiple hidden layers with non-linear activation functions (Tanh), and an output layer that produces the mean action $\mu(s)$. The action is then sampled from a normal distribution $N(\mu(s), \sigma^2)$, where σ is a fixed standard deviation. Mathematically, this can be expressed as:

$$\pi_\theta(a|s) = N(\mu_\theta(s), \Sigma)$$

where $\mu_\theta(s)$ is the output of the Actor network, and Σ is a diagonal covariance matrix with variance σ^2. The Critic network, on the other hand, estimates the value function $V_\phi(s)$, which represents the expected return from state s. It consists of an input layer that takes the state s, hidden layers similar to those in the Actor network, and an output layer that produces a scalar value representing $V_\phi(s)$. The value function is defined as:

$$V_\phi(s) \approx \mathbb{E}[R|s]$$

where R is the return, and $V_\phi(s)$ is the Critic's estimate.

During interaction with the environment, actions are selected using the old policy $\pi_{\theta_{\text{old}}}$ to maintain training stability. Given a state s_t, the old policy network outputs the mean action $\mu_{\theta_{\text{old}}}(s_t)$, and an action a_t is sampled from the Gaussian distribution:

$$a_t \sim N(\mu_{\theta_{\text{old}}}(s_t), \sigma^2)$$

The log probability of the action is also computed for use in the policy update:

$$\log \pi_{\theta_{\text{old}}}(a_t|s_t) = -\frac{1}{2}\left(\frac{(a_t - \mu_{\theta_{\text{old}}}(s_t))^2}{\sigma^2} + \log(2\pi\sigma^2)\right)$$

The policy is updated using the collected experiences after a certain number of steps or episodes. This involves several steps, beginning with the calculation of the discounted reward for each time step t in an episode:

$$R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$$

where γ is the discount factor, balancing the importance of immediate versus future rewards. The rewards are then normalized to stabilize training, typically by subtracting the mean and dividing by the standard deviation plus a small constant to avoid division by zero:

$$\hat{R}_t = \frac{R_t - \mu_R}{\sigma_R + 1e^{-5}}$$

where μ_R and σ_R are the mean and standard deviation of the rewards within the batch. The advantage $\hat{A}_t$ is calculated to measure the relative value of an action compared to the expected value:

$$\hat{A}_t = R_t - V_\phi(s_t)$$

PPO uses a surrogate objective function with a clipped probability ratio to constrain updates and prevent excessively large changes to the policy:

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

The probability ratio $r_t(\theta)$ between the new policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$ is defined as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

The clipped objective ensures that $r_t(\theta)$ stays within the range $[1-\epsilon, 1+\epsilon]$, preventing large policy updates that could destabilize the training process.

The critic network is trained to minimize the error between the predicted value $V_\phi(s_t)$ and the actual return R_t. The value function loss is given by:

$$L^{VF}(\phi) = \mathbb{E}_t \left[(R_t - V_\phi(s_t))^2 \right]$$

Additionally, an entropy bonus is included to encourage exploration by maintaining some randomness in the action selection. The entropy of the policy π_θ is defined as:

$$H(\pi) = -\mathbb{E} \left[\log \pi(a|s) \right]$$

The total loss function optimized during training combines the clipped surrogate objective, the value function loss, and the entropy bonus:

$$L(\theta, \phi) = -L^{CLIP}(\theta) + c_v \cdot L^{VF}(\phi) - c_e \cdot H(\pi_\theta)$$

where c_v and c_e are coefficients that balance the contributions of the value function loss and the entropy term, respectively.

Gradient descent is used to update the parameters θ and ϕ of the policy and value networks, respectively. The update rules are:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta, \phi)$$

$$\phi \leftarrow \phi - \alpha \nabla_\phi L^{VF}(\phi)$$

where α is the learning rate.

After updating the policy, the old policy $\pi_{\theta_{\text{old}}}$ is synchronized with the current policy π_θ to ensure consistency in future updates:

$$\theta_{\text{old}} \leftarrow \theta$$

This process iterates over multiple episodes, continually refining the agent's policy to achieve

the desired balance between exploration and exploitation. Training continues until the agent's performance stabilizes or meets predefined convergence criteria.

5.2 Design of Environment for the Baseline Training Without Visual Feedback

The ur5GymEnv environment provides a detailed state representation that includes crucial information about both the robot arm and the object within the simulation. This state representation is essential for understanding the current configuration and interactions within the environment. The position of the end-effector is represented by a three-dimensional vector $p_{tool} = (x_{tool}, y_{tool}, z_{tool})$, which indicates the coordinates of the end-effector in the simulation space. This vector is vital for determining the proximity of the end-effector to the target object and for calculating the necessary movements to achieve the desired task outcomes.

Similarly, the position of the object is represented by the vector $p_{obj} = (x_{obj}, y_{obj}, z_{obj})$. This vector provides the target location that the robot arm must reach and serves as a reference for evaluating how close the end-effector is to the object. The observation vector o, which concatenates the end-effector and object positions into a single array, is mathematically expressed as:

$$o = \begin{bmatrix} x_{tool} \\ y_{tool} \\ z_{tool} \\ x_{obj} \\ y_{obj} \\ z_{obj} \end{bmatrix}$$

This observation vector captures the state of the environment, providing the agent with all the necessary information to make informed decisions. The observation space O is defined to ensure that the values of the observation vector are within realistic ranges, with each element constrained by $[-10, 10]$:

$$O = [-h, h]$$

where $h = [10, 10, 10, 10, 10, 10]$. This constraint ensures that the robot arm and object positions are scaled appropriately for the learning algorithm.

In this environment, the action space outlines the possible actions the agent can take to interact with the environment, particularly in adjusting the position of the end-effector. The action vector a is represented as:

$$a = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ (\text{unused dimension}) \end{bmatrix}$$

where Δx, Δy, and Δz represent the incremental changes in the end-effector's position along the x, y, and z axes, respectively. The fourth component of the vector is unused in this implementation. The action space is bounded by a range of $[-0.1, 0.1]$ for each component, ensuring that the movements are constrained to small increments, making the task manageable and promoting controlled exploration of the state space.

The new position of the end-effector p_{new} is computed by adding the action vector to the current position:

$$p_{new} = p_{current} + a$$

This new position is then used to calculate the necessary joint angles q using inverse kinematics:

$$q = IK(p_{new}, \text{oor})$$

where q represents the joint angles required to achieve the new end-effector position, and oor denotes the orientation of the end-effector.

Termination conditions in the simulation define when an episode should end. The first condition for termination is task completion, which occurs when the Euclidean distance d between the end-effector and the object falls below a certain threshold λ:

$$\text{If } d < \lambda \text{ then Task Completed} = \text{True}$$

The distance d is calculated as:

$$d = \sqrt{(x_{tool} - x_{obj})^2 + (y_{tool} - y_{obj})^2 + (z_{tool} - z_{obj})^2}$$

If this distance is below the cutoff λ, the task is considered successfully completed. Another condition for termination is the maximum number of steps, where the episode ends if the step counter exceeds a predefined maximum number N_{max}:

$$\text{If Step Counter} > N_{max} \text{ then Episode Ends}$$

This limit ensures that episodes do not run indefinitely and that the interaction with the environment remains within a controlled duration.

The reward functions in this setup are crucial for guiding the agent toward the desired behaviors during training. Four distinct reward functions are employed, each introducing different levels of complexity and evaluation criteria, which progressively refine the learning process. The use of four reward functions is particularly motivated by the need to explore and evaluate different aspects of agent learning and performance optimization. Each reward function emphasizes different elements of the task, thereby providing a comprehensive assessment of the agent's capabilities.

The first reward function, the Basic Distance-Based Reward Function, focuses on encouraging the robot to minimize the distance between its end-effector and the target. The reward is calculated as:

$$\text{Reward} = -10 \times d$$

where d is the Euclidean distance between the end-effector position p_{ee} and the target position p_{goal}:

$$d = \sqrt{(p_{ee,x} - p_{goal,x})^2 + (p_{ee,y} - p_{goal,y})^2 + (p_{ee,z} - p_{goal,z})^2}$$

This negative reward is proportional to the distance, meaning that as the distance increases, the penalty becomes more significant. The primary goal here is to incentivize the robot to reduce this distance as much as possible, effectively guiding the robot closer to the target.

The second reward function, Distance with Proximity Penalty, introduces an additional penalty if the robot is too close to the target but not exactly on it, which enhances precision in positioning. This reward is formulated as:

$$\text{Reward} = -10 \times d - 5 \times \max(0, 0.15 - d)$$

This function continues to penalize the robot based on the distance while introducing a proximity penalty for being within 0.15 units of the target without achieving precise alignment. This added term encourages the robot to achieve exact proximity rather than merely approaching the target.

The third reward function, Distance Range Reward, is designed to reward the robot for maintaining its end-effector within a specific distance range from the target, thereby promoting precision and stability in task performance. The reward is defined as:

$$\text{Reward} = \begin{cases} 10 & \text{if } 0.15 \leq d \leq 0.35 \\ -10 & \text{otherwise} \end{cases} - 5 \times d$$

This function gives a positive reward when the end-effector is within the acceptable range of 0.15 to 0.35 units from the target. If the end-effector is outside this range, a penalty is applied, discouraging the robot from either being too far from the target or too close without precision.

The fourth reward function, Distance Range with Improvement Reward, combines the concept of maintaining a specific distance range with rewarding continuous improvement over time. The reward is expressed as:

$$\text{Reward} = \begin{cases} 10 & \text{if } 0.15 \leq d \leq 0.35 \\ -10 & \text{otherwise} \end{cases} - 5 \times d + 10 \times \Delta d$$

where Δd is the change in distance from the previous step, calculated as $\Delta d = d_{previous} - d_{current}$. This reward function not only emphasizes maintaining a precise distance but also encourages the robot to continuously reduce the distance to the target with each step. By rewarding reductions in distance, it promotes an iterative refinement in the robot's behavior, driving it to achieve and maintain a closer proximity to the target over time.

These four reward functions are strategically designed to address different aspects of the robot's learning process. By progressively increasing the complexity and introducing more nuanced evaluation criteria, these functions help ensure that the robot not only learns to perform the basic task of approaching the target but also refines its movements to achieve greater precision and stability. This multi-faceted approach to reward design enables a comprehensive evaluation of the agent's capabilities, ultimately leading to the development of a more robust and effective policy.

5.3 Design of Environment for the Baseline Training With Visual Feedback

In this enhanced environment, the state representation incorporates visual feedback, providing a richer and more detailed depiction of the scenario. The vector representing the end-effector's position is given by

$$p_{tool} = (x_{tool}, y_{tool}, z_{tool}),$$

which specifies its coordinates within the simulation. This vector plays a crucial role in determining how close the end-effector is to the target object and in guiding necessary movements. Similarly, the position of the object is denoted by

$$p_{obj} = (x_{obj}, y_{obj}, z_{obj}),$$

serving as the target location the robot arm aims to reach.

The addition of visual feedback is achieved through images captured by a camera mounted on the end-effector. These images are processed to extract key information, such as the position of the cube within the image frame. The processed image data is then integrated into the observation vector, enriching the agent's perception with both proprioceptive and visual information. The complete observation vector o is expressed as:

$$o = \begin{bmatrix} x_{tool} \\ y_{tool} \\ z_{tool} \\ x_{obj} \\ y_{obj} \\ z_{obj} \\ \text{processed\_image} \end{bmatrix}$$

This vector encapsulates the state of the environment, combining the positions of the end-effector and the object with visual data, thus providing a comprehensive basis for decision-making. The observation space O is defined to ensure all elements of the observation vector are within realistic bounds, specifically:

$$O = [-h, h]$$

where

$$h = [10, 10, 10, 10, 10, 10, 1, \ldots, 1],$$

which accommodates the inclusion of visual feedback.

The action space remains consistent with the baseline environment, allowing the agent to adjust the position of the end-effector. The action vector a is defined as:

$$a = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ (\text{unused dimension}) \end{bmatrix}$$

Here, Δx, Δy, and Δz represent the changes in the end-effector's position along the x, y, and z axes. Each component of the action vector is constrained within $[-0.1, 0.1]$ to maintain controlled and precise movements. The new position of the end-effector p_{new} is calculated by adding the action vector to the current position:

$$p_{new} = p_{current} + a$$

This position is then used to compute the required joint angles q using inverse kinematics:

$$q = IK(p_{new}, \text{oor}),$$

where oor represents the end-effector's orientation.

The termination conditions are extended to consider the visual feedback. An episode is deemed successful if the Euclidean distance d between the end-effector and the object falls below a predefined threshold λ:

$$\text{If } d < \lambda \text{ then Task Completed} = \text{True}.$$

This distance d is determined by:

$$d = \sqrt{(x_{tool} - x_{obj})^2 + (y_{tool} - y_{obj})^2 + (z_{tool} - z_{obj})^2}.$$

Episodes are also limited by a maximum step count N_{max}, ensuring they don't go on indefinitely:

$$\text{If Step Counter} > N_{max} \text{ then Episode Ends.}$$

The reward function for this environment is designed to motivate the robot to follow the cube using both proprioceptive and visual feedback. The reward structure has been adjusted to reflect the additional visual information, with modifications to the acceptable distance range and reward values. These changes aim to assess whether visual feedback improves the robot's performance in tracking the cube.

For the baseline training, the acceptable distance range was set between 0.15 and 0.35 units. With visual feedback, this range is refined to 0.2 to 0.4 units, hypothesizing that visual data allows for greater precision. The reward function is formulated as:

$$\text{Reward} = \begin{cases} 10 & \text{if } 0.2 \leq d \leq 0.4 \\ -10 & \text{otherwise} \end{cases} - 5 \times d + 10 \times \Delta d$$

Here, Δd represents the improvement in distance from the previous step, calculated as

$$\Delta d = d_{previous} - d_{current}.$$

This function incentivizes the robot to not only maintain proximity to the target but also to continuously improve its positioning accuracy by rewarding reductions in distance.

5.4 Design of Environment Training with YOLO-based Cube Detection

In this environment, the YOLOv8 model is employed to detect and track the cube in real-time, providing precise location data within the visual field of the robotic arm. The camera, mounted on the end effector, continuously captures RGB images of the environment, referred to as I_{RGB}. These images are preprocessed by resizing them to the input dimensions required by YOLOv8 and normalizing the pixel values to suit the model's requirements.

After preprocessing, the images are fed into the YOLOv8 model, which detects the cube and provides the coordinates of a bounding box around it. The center of the bounding box, representing the 2D position of the cube in the image frame, is calculated using the coordinates of the top-left corner (x_1, y_1) and the bottom-right corner (x_2, y_2). The center coordinates are determined as follows:

$$x_c = \frac{x_1 + x_2}{2}$$
$$y_c = \frac{y_1 + y_2}{2}$$

Simultaneously, depth information is captured by the camera, allowing for the extraction of the depth value Z_c in the bounding box's center:

$$Z_c = I_{depth}(x_c, y_c)$$

The transformation from 2D image coordinates to 3D world coordinates is performed using the intrinsic parameters of the camera. These intrinsic parameters include the focal lengths f_x and f_y, ,in addition to the principal points c_x and c_y. The focal lengths f_x and f_y are calculated based on the camera's field of view (FOV) and the image resolution, using the formula:

$$f_x = f_y = \frac{\text{Image Width}}{2 \cdot \tan\left(\frac{\text{FOV}}{2}\right)}$$

The principal points c_x and c_y represent the optical center of the image and are typically located at the center of the image, calculated as:

$$c_x = \frac{\text{Image Width}}{2}$$
$$c_y = \frac{\text{Image Height}}{2}$$

With these intrinsic parameters, the transformation of the 2D image coordinates and the depth value into 3D world coordinates is carried out. The camera coordinates are first computed as:

$$X_{camera} = \frac{(x_c - c_x) \cdot Z_c}{f_x}$$
$$Y_{camera} = \frac{(y_c - c_y) \cdot Z_c}{f_y}$$

$$Z_{camera} = Z_c$$

These camera coordinates are then converted to the world coordinate system by considering the camera's orientation relative to the end effector, resulting in:

$$X_c = Z_{camera}$$

$$Y_c = -X_{camera}$$

$$Z_c = Y_{camera}$$

This process ensures that the cube's position in the 3D world coordinate system is accurately determined.

The state space in this environment includes the 3D positions of both the end-effector and the cube. The position of the end-effector is represented by the vector $p_{tool} = (x_{tool}, y_{tool}, z_{tool})$, while the position of the cube is represented by $p_{obj} = (x_{obj}, y_{obj}, z_{obj})$. These positions are combined into an observation vector o, which provides a comprehensive snapshot of the environment:

$$o = \begin{bmatrix} x_{tool} \\ y_{tool} \\ z_{tool} \\ x_{obj} \\ y_{obj} \\ z_{obj} \end{bmatrix}$$

The observation space O is defined to ensure that all values are within realistic and appropriately scaled bounds for effective learning, expressed as:

$$O = [-h, h], \quad h = [10, 10, 10, 10, 10, 10]$$

The action space defines the range of possible movements available to the robotic arm. Actions are represented as a vector $a = [\Delta x, \Delta y, \Delta z]$, which indicates incremental changes in the end-effector's position along the x, y, and z axes. These movements are constrained within the range $[-0.1, 0.1]$ to maintain control and precision. The new position of the end-effector p_{new} is calculated by:

$$p_{new} = p_{current} + a$$

Inverse kinematics is then applied to determine the joint angles q required to achieve this new position:

$$q = IK(p_{new}, \text{oor})$$

where oor denotes the orientation of the end-effector.

Termination conditions in the simulation ensure that episodes end under specific circumstances. Task completion is determined by the Euclidean distance d between the end-effector and the cube:

$$d = \sqrt{(x_{tool} - x_{obj})^2 + (y_{tool} - y_{obj})^2 + (z_{tool} - z_{obj})^2}$$

An episode is considered successfully completed if this distance falls below a predefined threshold λ:

$$\text{If } d < \lambda \text{ then Task Completed} = \text{True}$$

Additionally, episodes are terminated if the step counter exceeds a maximum number N_{max}:

$$\text{If Step Counter} > N_{max} \text{ then Episode Ends}$$

During testing, it was observed that inaccuracies in bounding box detection were most pronounced when the object was within a specific distance range from the camera, particularly between 1 and 1.6 units away. These inaccuracies led to systematic errors in the detected 3D position, which directly impacted the calculated distance between the end-effector and the cube. To correct these errors, an offset of 0.7 units is applied to the distance measurement when the detected distance falls within this range:

$$d_{corrected} = \begin{cases} d - 0.7 & \text{if } 1 \leq d \leq 1.6 \\ d & \text{otherwise} \end{cases}$$

This correction helps ensure that the robot's estimation of the actual distance to the object is more accurate, leading to more reliable reward calculations.

The reward function assesses the robot's performance, guiding it toward effective interaction

with the cube. The reward is calculated based on the corrected Euclidean distance $d_{corrected}$, adjusted for bounding box inaccuracies. The reward function is expressed as:

$$\text{Reward} = 40 - 5 \times d_{corrected} \quad \text{if } 0.25 \leq d_{corrected} \leq 0.85$$

$$\text{Reward} = -25 - 5 \times d_{corrected} \quad \text{otherwise}$$

In addition to the distance-based reward, the function includes terms for cube detection and penalties for collisions. The final reward function is given by:

$$\text{Reward} = 75 \times \text{cube\_detected} - 30 \times \text{not\_cube\_detected} - \text{collision\_penalty}$$

where cube_detected and not_cube_detected are binary indicators of whether the cube is detected, and collision_penalty is applied if a collision occurs.

5.5 Hyperparameters in Reinforcement Learning

Hyperparameters are essential components in the design and training of reinforcement learning (RL) agents. They significantly influence how an agent explores its environment, updates its knowledge, and balances exploration and exploitation. Proper configuration of hyperparameters is crucial for achieving effective learning and optimal performance. This section provides an in-depth examination of the key hyperparameters used in the RL training process, explaining their roles and impacts on the training dynamics.

5.6 Hyperparameters in Reinforcement Learning

Hyperparameters are essential components in the design and training of reinforcement learning (RL) agents. They significantly influence how an agent explores its environment, updates its knowledge, and balances exploration and exploitation. Proper configuration of hyperparameters is crucial for achieving effective learning and optimal performance. This section provides an in-depth examination of the key hyperparameters used in the RL training process, explaining their roles and impacts on the training dynamics.

`-mel` (Maximum Episode Length) defines the maximum number of timesteps allowed per episode. The length of an episode directly affects the training process. Longer episodes provide

more opportunities for interaction with the environment, enabling the agent to learn more effectively, but they also increase training time and computational cost. On the other hand, shorter episodes may speed up training but might not give the agent sufficient time to accomplish its goals or learn effective strategies.

`-lp` (Learning Parameter) is a task-specific parameter that influences thresholds or goals within the reward function. It defines criteria for success, such as a distance threshold for completing a task. Adjusting the learning parameter alters the difficulty of the task and how the agent perceives and responds to different states, impacting its learning efficiency and effectiveness.

`-seed` parameter sets the random seed for reproducibility. Fixing the seed ensures that experiments are reproducible by controlling the randomness in various aspects of training, including environment interactions and policy initialization. This is crucial for debugging, comparing results across different runs, and ensuring consistent training outcomes.

`-max_episodes` sets the maximum number of episodes for training the agent. This parameter limits the total training time. More episodes provide more learning opportunities but also increase training time and computational cost. Balancing the number of episodes is crucial for efficient training and achieving optimal performance.

`-update_timestep` parameter defines the number of timesteps after which the policy is updated. The frequency of policy updates based on collected experiences influences the learning process. Frequent updates can lead to faster learning but may result in noisy updates, while infrequent updates offer more stable learning but may slow down convergence.

`-action_std` specifies the standard deviation for the action distribution, controlling the exploration-exploitation trade-off by defining the variance in the actions taken by the agent. Higher values promote exploration by encouraging diverse actions, while lower values make actions more deterministic, focusing on exploiting learned strategies.

`-K_epochs` determines the number of epochs for updating the policy per batch of data. This parameter affects how many times the policy is refined using collected experiences. More epochs can lead to better policy refinement and improved learning but also increase training time and resource consumption.

`-eps_clip` is the clipping parameter for PPO, which limits the change in policy during updates. It helps stabilize training by constraining the ratio between the new and old policies. Smaller clipping values ensure more conservative policy updates, reducing the risk of instability during training.

`-gamma` represents the discount factor for future rewards. It balances the importance of immedi-

ate versus future rewards, where a value close to 1 makes the agent consider long-term rewards, encouraging strategic planning, while a value close to 0 emphasizes immediate rewards, leading to more short-term behavior.

`-lr` (learning rate) is the parameter for the optimizer, controlling the rate at which the model parameters are updated. A high learning rate can speed up convergence but may cause instability, while a low learning rate provides more stable updates but might slow down learning.

`-betas` refers to the coefficients for the Adam optimizer's moving averages of gradients. The parameters β_1 and β_2 control the decay rates of the first and second moments of gradients, respectively. Typical values such as $(0.9, 0.999)$ provide a balance between stability and learning efficiency, with adjustments affecting optimization dynamics and convergence properties.

`-loss_entropy_c` is the coefficient for the entropy term in the loss function. This parameter encourages exploration by adding an entropy term to the loss function, where a higher value increases exploration by penalizing deterministic policies, while a lower value reduces the emphasis on exploration. `-loss_value_c` is the coefficient for the value function loss term, balancing the importance of the value function loss relative to other loss terms. A higher value prioritizes accurate value predictions, improving learning stability and performance, but may lead to overfitting if set too high.

So, the design of the simulation environments and the implementation of the PPO algorithm are critical to the success of this research. By carefully structuring the training scenarios and fine-tuning the PPO parameters, we have established a robust framework that enables the robotic arm to learn and perform complex tasks efficiently. This chapter provides the groundwork for the experimental results, demonstrating the effectiveness of the designed environments and the PPO implementation in enhancing the robot's operational capabilities.

Chapter VI

Reasults and Discussion

In this chapter, we present the results of the experiments conducted to evaluate the performance of the 4DOF robotic arm across different training scenarios. These scenarios include baseline training without visual feedback, training with visual feedback, and training with YOLO-based object detection. Additionally, a video demonstration is provided to visually showcase the robot's capabilities in real-time, highlighting key moments from the experiments. The chapter discusses these results in detail, analyzing the effectiveness of the Proximal Policy Optimization (PPO) algorithm in improving the robot's ability to track and follow a moving cube. The discussion also interprets the data, identifies key patterns, and compares performance across the different setups to draw meaningful conclusions.

6.1 Baseline Training With- out Visual Feedback

6.1.1 Hyperparameter Exploration and Refinement

Initially, we set the discount factor to 0.92 and the learning rate to 0.02. The discount factor determines how much the agent values future rewards, while the learning rate controls the size of the weight updates during neural network training. These values were chosen based on theoretical considerations and common practices, but the agent's learning did not show the desired convergence.

Understanding the crucial impact of hyperparameters on the learning process, we decided to systematically explore different configurations to find the optimal settings. This involved varying one hyperparameter at a time while keeping the others constant, allowing us to isolate and analyze the effect of each parameter on the agent's performance. Through iterative experiments and careful analysis of the learning behavior, we aimed to identify parameter settings

that would facilitate convergence.

This process of systematic exploration is akin to conducting a scientific experiment, where each configuration acts as a trial to evaluate the agent's performance under different conditions. We found that even minor adjustments in hyperparameters could significantly influence the learning curve. This underscored the sensitivity of the learning process to parameter choices and the importance of meticulous parameter tuning.

Refining Hyperparameters

Through extensive experimentation and systematic tuning, we identified a set of hyperparameters that significantly enhanced the agent's learning performance and ensured successful convergence. The following table presents the optimal configuration for each reward function, highlighting crucial factors such as max episode length, action_std, update_timestep, discount factor, K_epochs, learning rate, eps_clip, loss_entropy_c, and action_bound.

Hyperparameter	**Reward Function 1**	**Reward Function 2**	**Reward Function 3**	**Reward Function 4**
Max Episode Length	150	170	200	200
Action Std	1.0	1.0	1.0	1.0
Update Timesteps	500	700	1000	1000
Discount Factor (γ)	0.99	0.99	0.99	0.99
K Epochs	50	35	20	20
Learning Rate	0.01	0.001	0.0001	0.0001
EPS Clip	0.2	0.17	0.15	0.15
Loss Entropy Coefficient	0.01	0.014	0.02	0.02
Action Bound	1.0	0.5	0.1	0.1

Table VI.1: Hyperparameters for Each Reward Function for Baseline Training.

Impact of Hyperparameters on Agent Convergence

- **Max Episode Length**:
  - For Reward Function 1, a max episode length of 150 provided the agent with ample time to explore and refine its policy. For Reward Functions 2 through 4, extended lengths of 170 and 200 were essential to accommodate more complex task dynamics and ensure thorough learning across diverse scenarios.
- **Action Std**:
  - A consistent action standard deviation of 1.0 was utilized across all reward functions, facilitating a stable exploration strategy that balanced the trade-off between exploration and exploitation.

- **Update Timesteps**:
  - The Update Timesteps were adjusted to 500 for Reward Function 1, optimizing the frequency of policy updates to match the complexity of the task. For Reward Function 2, 700 timesteps allowed for more frequent updates, aligning with its specific requirements. For Reward Functions 3 and 4, a setting of 1000 timesteps effectively managed policy learning while supporting a broader exploration phase.
- **Discount Factor (γ)**:
  - A discount factor of 0.99 was consistently applied to all reward functions. This high discount factor emphasized the importance of long-term rewards and encouraged the agent to focus on future gains, fostering a more effective learning process.
- **K Epochs**:
  - The number of epochs was set to 50 for Reward Function 1 and 35 for Reward Function 2. These values were chosen to provide sufficient policy updates for the more complex tasks. For Reward Functions 3 and 4, a reduced number of 20 epochs was sufficient to achieve convergence while maintaining computational efficiency.
- **Learning Rate**:
  - Learning rates varied according to the complexity of the reward functions: 0.01 for Reward Function 1, and more conservative rates of 0.001, 0.0001, and 0.0001 for the subsequent functions. These settings controlled the speed of policy updates and ensured stability in the learning process.
- **EPS Clip**:
  - The clipping parameter (eps_clip) was set to 0.2 for Reward Function 1 to manage larger policy updates effectively. For the remaining functions, values of 0.17 and 0.15 helped stabilize training by limiting policy changes to prevent excessive variance.
- **Loss Entropy Coefficient**:
  - The entropy coefficient was set to 0.01 for Reward Function 1 and 0.014 for Reward Function 2, with values of 0.02 for Reward Functions 3 and 4. This parameter was crucial for balancing exploration and exploitation, with higher values encouraging broader exploration.
- **Action Bound**:
  - Action bounds were set to 1.0 for Reward Function 1, 0.5 for Reward Function 2,

and 0.1 for Reward Functions 3 and 4. These settings ensured that actions remained within feasible limits, contributing to stable and reliable policy learning.

This comprehensive configuration is a testament to the intricate relationship between hyperparameters and the agent's learning process. Through deliberate experimentation and meticulous parameter tuning, we identified the optimal combination that guided the agent toward achieving its learning objectives, culminating in the desired convergence.

This experience underscores the intricate interplay between hyperparameters and an agent's learning trajectory. It highlights the importance of systematic experimentation and iterative optimization to fine-tune parameter settings. By gaining insights from the simulation and refining our parameter choices, we enhanced the agent's performance and achieved convergence, marking a significant milestone in the reinforcement learning journey.

6.1.2 Rewards

The analysis of the agent's performance over the initial 500 episodes provides critical insights into how different reward functions shape learning and behavior. The trends observed during this period are key to understanding the early stages of the agent's development.
As illustrated in Figure VI.1, the rewards associated with the Basic Distance and Collision Penalty function start with a significant penalty, averaging around -3000 in the first few episodes. This reflects the agent's initial difficulties in avoiding collisions and minimizing the distance to the target. However, by episode 100, there is a noticeable improvement, with rewards increasing to approximately -1500. This trend indicates that the agent quickly adapts to basic navigational tasks, learning to reduce penalties associated with collisions. As the episodes progress, the rewards stabilize around -1000 by episode 500, suggesting that while the agent has made substantial progress, further improvement is limited by the constraints of this reward structure.

Figure VI.1: Training_Rewards_500 for Reward Function 1.

In the case of the Distance Range and Penalty for Proximity function, shown in Figure VI.2, the rewards begin at around -2500, reflecting the challenge of balancing proximity to the target with the need to avoid collisions. Throughout the first 100 episodes, the rewards improve, fluctuating between -1500 and -1000. This fluctuation suggests that the agent is continuously adjusting its strategy in response to the competing demands of proximity and safety. By episode 500, the rewards show a slight improvement but remain within the range of -1200 to -1000, indicating ongoing challenges in consistently maintaining the optimal distance from the target.

Figure VI.2: Training_Rewards_500 for Reward Function 2.

For the Preferred Distance Range with Heavy Penalties function, depicted in Figure VI.3, the rewards initially register around -3000 due to frequent deviations from the preferred distance range of 0.15 to 0.35 units. However, by episode 200, the rewards improve significantly to approximately -1000, as the agent learns to operate more effectively within the specified range. By episode 500, the rewards trend closer to -500, indicating a substantial reduction in penalties as the agent becomes more proficient in maintaining the desired distance. Nonetheless, occasional drops in rewards to lower values suggest that precise control within this range remains challenging, even as the agent's overall performance improves.
The Distance Range with improved reward function, shown in Figure VI.4, demonstrates a more consistent upward trajectory in rewards over the 500 episodes. Starting with rewards around -2000 in the initial episodes, the agent steadily improves, with rewards increasing to approximately -500 by episode 300. By the end of the 500 episodes, the rewards approach zero, reflecting the agent's successful learning and continuous improvement in maintaining proximity to the target. This steady increase in rewards suggests that this function effectively reinforces both immediate and long-term objectives, leading to sustained progress throughout the training process.

The analysis of the agent's performance over 1,000 episodes offers a deeper understanding of how

Figure VI.3: Training_Rewards_500 for Reward Function 3.

Figure VI.4: Training_Rewards_500 for Reward Function 4.

different reward functions impact long-term learning and behavioral refinement. The trends observed in each figure highlight the effectiveness and challenges associated with each reward structure.

As illustrated in Figure VI.5, the rewards for the Basic Distance and Collision Penalty function show a significant improvement in the early episodes, rising from approximately -3000 at the start to around -1000 by episode 1000. This initial phase of rapid learning indicates that the agent quickly adapts to minimizing collisions and reducing the distance to the target. However, beyond episode 1000, the rewards stabilize within the range of -1000 to -800 for the remaining 9000 episodes. This stabilization indicates that the agent has reached a performance plateau where it effectively minimizes collisions and maintains proximity to the target, but further refinement of its policy is limited. The absence of further upward movement suggests that the reward function lacks the additional gradients needed to push the agent toward more refined behavior.

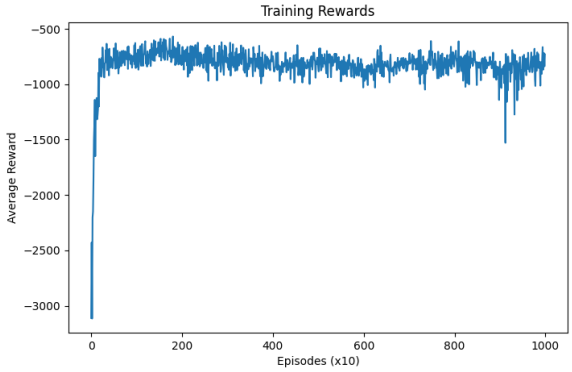

Figure VI.5: Training_Rewards_10000 for Reward Function 1.

In Figure VI.6, the rewards associated with the Distance Range and Penalty for Proximity function start at approximately -2500 and improve to around -1200 by episode 1000. However, the rewards exhibit significant fluctuations between -1750 and -1000 from episode 2000 onwards. These fluctuations suggest that the agent struggles to consistently balance the competing demands of maintaining an optimal distance from the target while avoiding penalties. The lack of convergence towards a more stable reward trajectory suggests that the complexity introduced by the proximity penalty significantly challenges the agent's ability to optimize its policy. The continued variability in rewards over the full 10,000 episodes highlights the difficulty in achieving a stable and optimized policy under this reward structure.

Figure VI.7 shows that the rewards for the Preferred Distance Range with Heavy Penalties function increase sharply from around -3000 to near zero by episode 2000. However, in the subsequent episodes, the rewards fluctuate around zero, with occasional dips to approximately

Figure VI.6: Training_Rewards_10000 for Reward Function 2.

-500. These dips indicate that while the agent has generally learned to operate within the preferred distance range, it occasionally fails to maintain the required precision, particularly when faced with varying environmental conditions. The fluctuations observed in the rewards suggest that the heavy penalties for deviations from the preferred range continue to challenge the agent, leading to periodic instability in its performance.

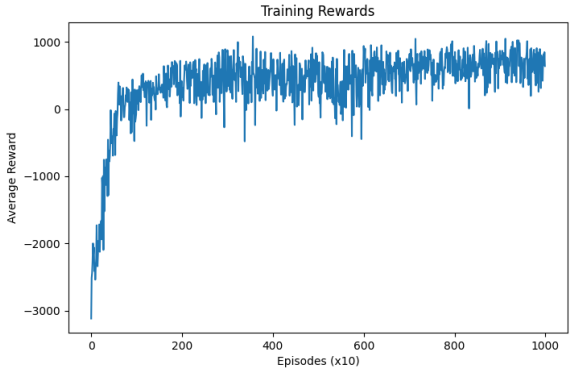

Figure VI.7: Training_Rewards_10000 for Reward Function 3.

In Figure VI.8, the Distance Range with improved reward function results in a steady and consistent improvement in rewards over the 10,000 episodes. The rewards start at approximately -2000 and gradually increase, crossing into positive territory by around episode 3,000. By the final episodes, between 9,000 and 10,000, the rewards stabilize around +500. This continuous upward trajectory underscores the effectiveness of this reward function in encouraging not only the maintenance of optimal performance but also ongoing refinement. The consistent improvement observed across the full training period suggests that this reward function is particularly well-suited for promoting long-term learning and adaptation, enabling the agent to achieve and sustain high-performance levels.

The remarkable performance of the fourth reward function across important training parameters

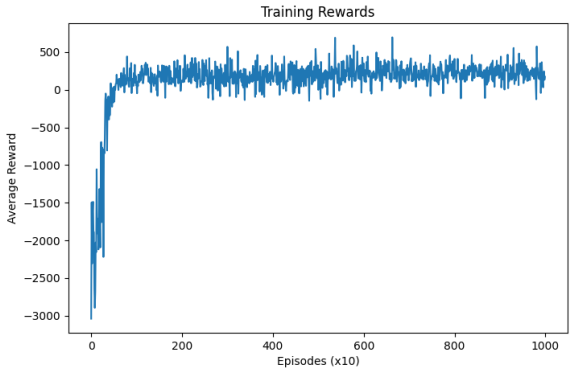

Figure VI.8: Training_Rewards_10000 for Reward Function 4.

led to the choice to concentrate on it. In early tests, this reward function continuously performed better than the others by enabling quick learning and preserving motion stability in the robot. It is the best option for further in-depth research as it offers a well-balanced incentive structure that prompts the robot to keep its motions consistent and reduce the distance it takes to reach the destination.

6.1.3 Positional Errors

The evaluation of smoothed positional errors throughout the training process provides critical insights into the robot's ability to accurately position itself about the target cube. The trends observed in both the initial 500 episodes and the full 10,000 episodes are key to understanding the effectiveness and refinement of the agent's learning process.

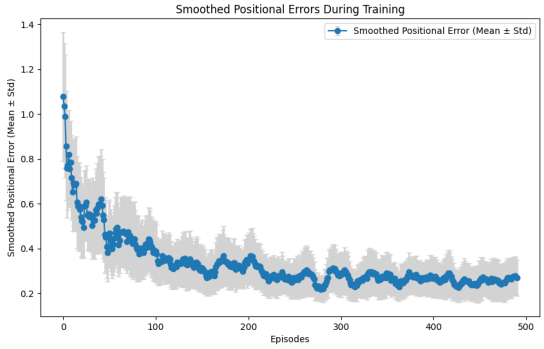

Figure VI.9: Smoothed_Positional_Errors_500.

As depicted in Figure VI.9, the smoothed positional errors show a significant decrease during the initial stages of training. In the first 100 episodes, the errors drop sharply from approximately 1.0 to around 0.3. This steep decline indicates a rapid improvement in the robot's positioning accuracy, demonstrating the effectiveness of the PPO algorithm in enabling the robot to quickly

learn and adapt its movements to reduce positional inaccuracies. The critical observation here is the speed at which the agent adjusts its behavior, reflecting a strong initial learning phase.

After this initial period, the error values begin to stabilize between 0.3 and 0.4, indicating that the robot has reached a level of performance where further reductions in positional errors are more gradual. The stabilization suggests that the robot consistently maintains a higher degree of accuracy in its positioning, though it has entered a phase where additional improvements are more incremental. This trend highlights the agent's transition from rapid learning to fine-tuning its performance, with error stabilization marking the beginning of a phase focused on maintaining and slightly enhancing precision.

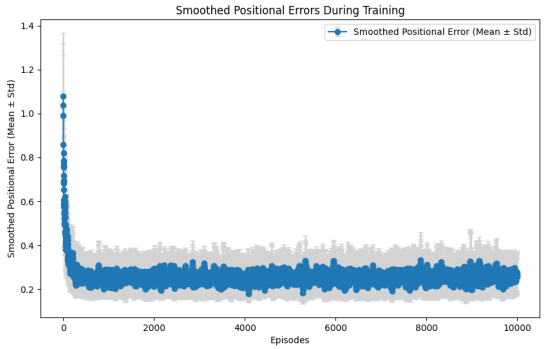

Figure VI.10: Smoothed_Positional_Errors_10000.

When extending the analysis to the full 10,000 episodes, as shown in Figure VI.10, the trend of decreasing positional errors continues, albeit at a slower rate compared to the initial episodes. The errors gradually decrease over time, stabilizing around 0.2 by the end of the training period. This ongoing reduction, although more subtle, indicates that the robot continues to refine its ability to position itself accurately relative to the target, even after the initial learning curve has plateaued.

A key observation in the later stages of training is the reduction in the standard deviation, which narrows as the episodes progress. This reduction in variability underscores the increased consistency in the robot's performance, suggesting that the robot not only improves its average accuracy but also becomes more reliable in consistently achieving precise positioning. The narrowing standard deviation indicates that the agent's movements are becoming more repeatable, leading to a more stable and predictable behavior over time.

6.1.4 Target Distance

The analysis of average target distance over episodes provides crucial insights into the robot's capability to maintain proximity to the target cube throughout the training process. By exam-

ining this measure across both the initial 500 episodes and the full 10,000 episodes, we gain a deeper understanding of how the robot's control and precision evolve over time.

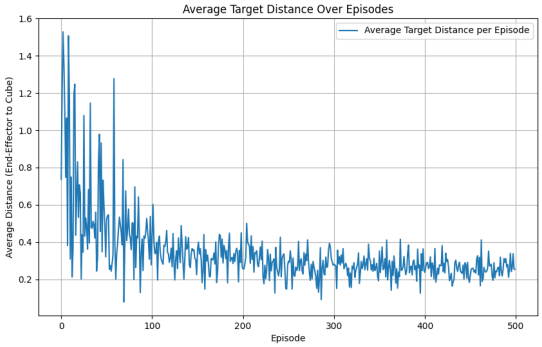

Figure VI.11: Target_Distance_over_Episodes_500.

As illustrated in Figure VI.11, the average target distance shows a sharp decline during the first 100 episodes, dropping from approximately 1.5 units to around 0.3 units. This significant decrease indicates that the robot rapidly learns to adjust its movements to maintain closer proximity to the cube. The early steep decline in distance reflects the effectiveness of the learning algorithm in enabling the robot to quickly understand and execute the task of minimizing the distance to the target.

As the training progresses beyond the initial 100 episodes, the average distance stabilizes around 0.3 units, with reduced fluctuations. This stabilization suggests that the robot has achieved a certain level of precision and control in its positioning. The reduction in variability over the later episodes indicates that the robot's behavior becomes more consistent, reflecting a refined understanding of how to maintain proximity to the cube with minimal deviation.

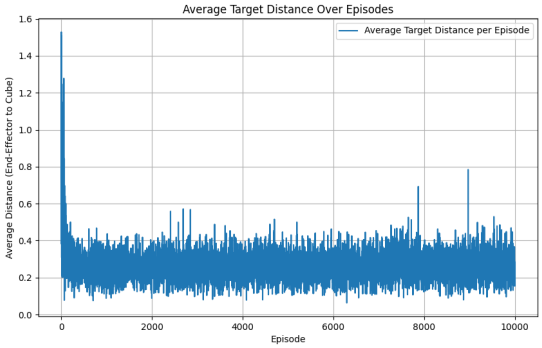

Figure VI.12: Target_Distance_over_Episodes_10000.

Extending the analysis to the full 10,000 episodes, as shown in Figure VI.12, reveals that the robot consistently maintains an average target distance of approximately 0.26 units. This

consistent performance indicates that the robot has not only learned to stay close to the target but also maintains this proximity over an extended period, demonstrating long-term stability in its behavior.

Throughout the 10,000 episodes, minor spikes in the average distance are observed, particularly around certain episodes. These spikes likely correspond to instances where the robot engages in exploratory behavior or makes adjustments to its strategy, potentially in response to changes in the environment or internal policy updates. Despite these occasional deviations, the overall trend shows a strong ability to maintain a close and consistent distance to the target, underscoring the robustness of the robot's learned policy.

6.2 Baseline Training With Visual Feedback

6.2.1 Hyperparameter Fine-Tuning

Initial attempts to apply the same hyperparameters from Baseline Training Without Visual Feedback yielded unsatisfactory results in achieving convergence within this environment. As a response, an iterative process of trial and error ensued, leading to numerous simulations that explored different hyperparameter combinations. After meticulous fine-tuning, a set of hyperparameters were identified that finally achieved convergence for Baseline Training With Visual Feedback as shown in the table :

Hyperparameter	**Value**
Max Episode Length	160
Action Std	1.0
Update Timesteps	1100
Discount Factor ()	0.99
K Epochs	23
Learning Rate	0.0015
EPS Clip	0.2
Loss Entropy Coefficient	0.02
Action Bound	0.1

Table VI.2: Hyperparameter Values for Visual Feedback.

6.2.2 Rationale for Choosing the 4th Reward Function

The 4th reward function was selected due to its superior performance in initial trials, offering the highest and most consistent rewards. Its balanced structure rewards the agent for maintaining

an optimal distance from the target, achieving close proximity, continuously improving its distance to the target, and penalizing deviations and collisions. This comprehensive approach ensures effective and optimal navigation by the agent.

6.2.3 Initial 500 Episodes Analysis with Visual Feedback

Rewards

Figure VI.13: Training_Rewards_500.

The rewards observed over the initial 500 episodes, as depicted in Figure VI.13, reflect the agent's performance as it integrates this additional sensory input into its learning process. At the start of the training, the agent begins with highly negative rewards, around -3000. However, as training progresses, there is a notable and rapid improvement in rewards, demonstrating that the agent quickly learns to leverage visual feedback. This early rise in rewards indicates the agent's growing proficiency in using visual cues to avoid collisions and minimize the distance to the target. The visual feedback enables the agent to make more accurate adjustments, accelerating its ability to adapt to the environment.

Throughout the 500 episodes, rewards continue to increase with some fluctuations, which suggest ongoing refinement in the agent's strategies. These variations reflect the dynamic nature of the learning process, where the agent experiments with different actions and learns from the resulting outcomes. Despite these fluctuations, the overall trend is upward, indicating a consistent improvement in performance as the agent becomes more adept at integrating visual information into its decision-making process. By the later episodes, the rewards approach closer to zero, signifying that the agent has effectively learned to balance the various elements of the reward function, such as maintaining proximity to the target while avoiding penalties

for collisions. The reduction in variability towards the end of the training period suggests that the agent's behavior becomes more stable and reliable, a clear indication that visual feedback has contributed to a more refined and consistent performance.

Target Distance

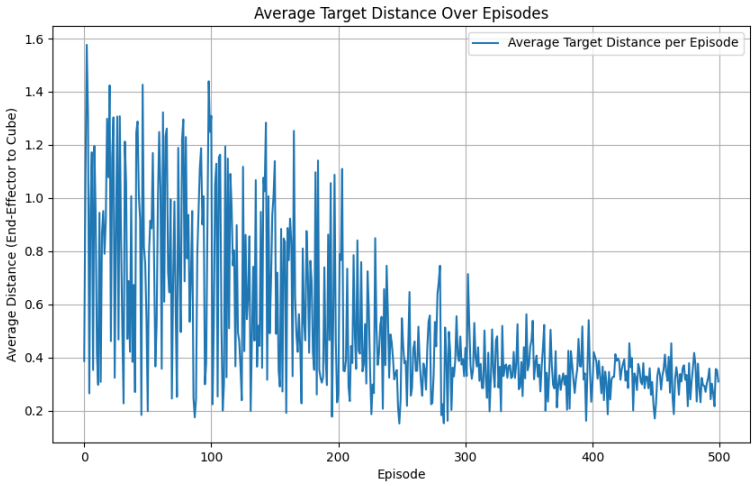

Figure VI.14: Target_Distance_over_Episodes_500.

At the start of training, as shown in Figure VI.14, the average target distance is approximately 1.6 units, which reflects the robot's initial difficulty in accurately navigating and positioning itself using visual and simulation feedback. This relatively large distance underscores the challenges faced by the robot in the early stages of training, as it struggles to effectively integrate the feedback into its control strategies. As training progresses, a noticeable and consistent decrease in the target distance is observed. By the end of the first 500 episodes, the distance has significantly reduced to around 0.4 units. This substantial reduction highlights the robot's increasing ability to close the gap to the target. The improvement in maintaining proximity to the target indicates that the robot is effectively utilizing the visual feedback to adjust its movements with greater precision.

The consistent decline in target distance throughout the episodes suggests that the robot is not only learning to stay close to the target but is also prioritizing this aspect of its performance. This focus on minimizing distance is critical for the successful completion of the task, as remaining near the target is a key indicator of the robot's ability to execute its objectives effectively.

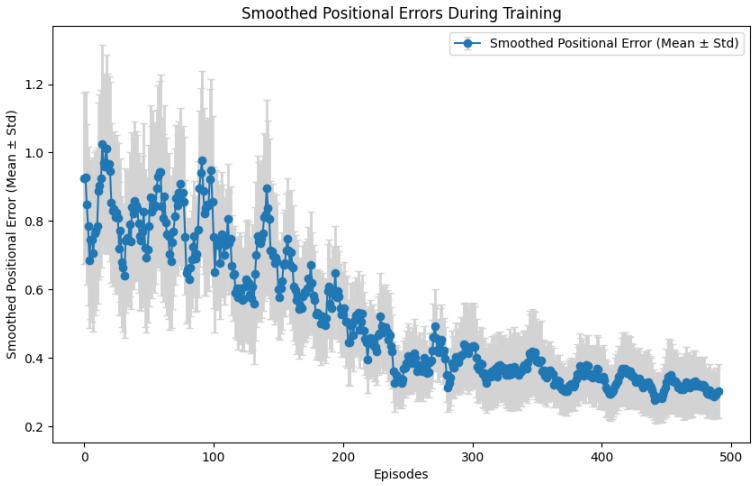

Figure VI.15: Smoothed_Positional_Errors_500.

Positional Errors

Positional errors are another important parameter that indicates how accurately the robot is positioning itself relative to the target. These errors start rather high as shown in Figure VI.15, averaging approximately 1.2 units, which is to be expected given the robot's first difficulty in efficiently using the feedback. As training progresses, there is a considerable reduction in location errors, particularly by the middle of 500 episodes. By the end of this phase, the errors had decreased to around 0.4 units. This reduction demonstrates that the robot is improving at understanding feedback and making precise modifications to its position, resulting in more regulated and accurate movements.

6.2.4 Full 10,000 Episodes Analysis with Visual Feedback

Rewards

The analysis of rewards throughout the full 10,000 episodes, as depicted in Figure VI.16, highlights the agent's learning process and optimization of behavior when utilizing visual feedback. In the initial 500 episodes, the agent shows a rapid improvement, with rewards increasing sharply from approximately -2000 to near-zero. This significant early gain indicates that the agent quickly adapts to the environment, effectively using visual feedback to minimize positional errors and enhance performance.

As the training progresses from 500 to 3000 episodes, the rewards continue to increase steadily. During this phase, the fluctuations observed earlier begin to stabilize, suggesting that the agent is refining its strategies and consistently maintaining an optimal distance from the target. This period of steady improvement reflects the agent's ability to consolidate its learning, leveraging

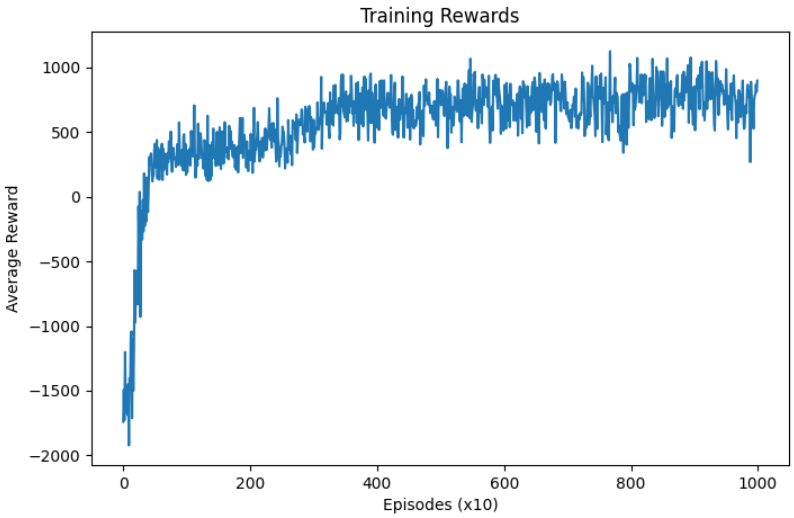

Figure VI.16: Training_Rewards_10000.

visual feedback to make precise adjustments and optimize its actions further. In the final stretch from 3000 to 10,000 episodes, the rewards remain consistently positive, averaging between 0 and 500. Occasional spikes reaching up to 1000 suggest moments where the agent employs particularly effective strategies. The reduced variability in rewards during this phase indicates a high degree of convergence, with the agent achieving a stable and optimized performance. The continued use of visual feedback allows the agent to fine-tune its strategies, ensuring sustained improvements and reliable task execution over a prolonged period.

Target Distance

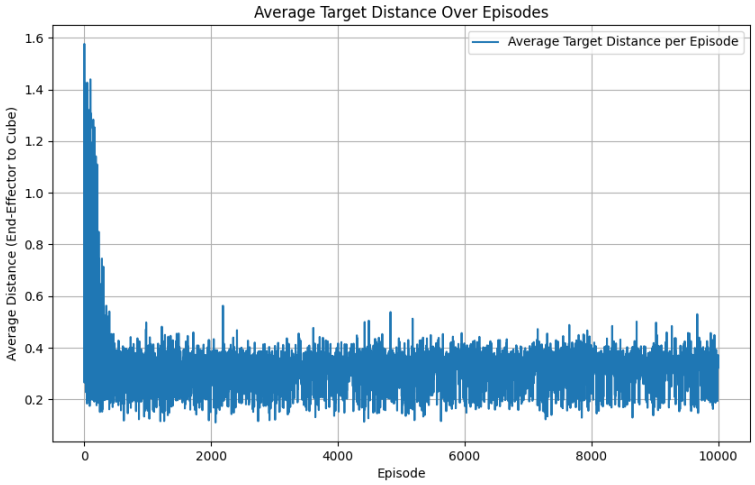

Figure VI.17: Target_Distance_over_Episodes_10000.

The analysis of target distance over 10,000 episodes, as shown in Figure VI.17, reflects the

robot's ongoing learning and adaptation process. In the early stages of training, the robot rapidly adjusts to the feedback mechanisms, resulting in a significant decrease in the target distance from approximately 1.6 units to around 0.3 units. This sharp reduction indicates the robot's ability to quickly utilize the provided visual and simulation feedback to refine its positioning relative to the target. As training progresses, the target distance stabilizes around 0.3 units, demonstrating a consistent performance across the remaining episodes. This stability suggests that the robot has developed a reliable strategy for maintaining an optimal distance from the target, effectively balancing its use of visual and simulation feedback to fine-tune its movements. The consistency of this metric over thousands of episodes indicates that the robot has successfully generalized its learned behaviors, ensuring it can consistently maintain proximity to the target across various scenarios within the training environment. This sustained accuracy in maintaining a close target distance over a prolonged period underscores the effectiveness of the training process. It reflects the robot's capability to adapt its strategies effectively and apply its learned behaviors reliably, even as it encounters a diverse set of conditions throughout the extensive training period.

Positional Errors

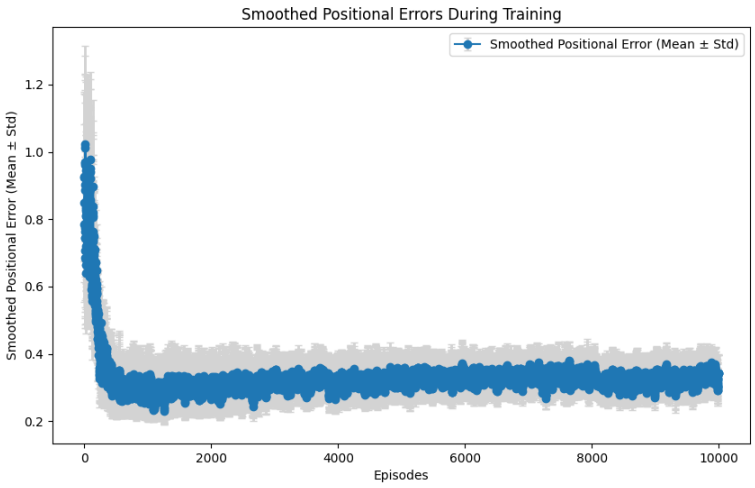

Figure VI.18: Smoothed_Positional_Errors_10000.

The analysis of smoothed positional errors across the full 10,000 episodes, as illustrated in Figure VI.18, highlights the robot's continuous improvement in spatial accuracy. Initially, there is a significant reduction in positional errors, building on the progress observed during the first 500 episodes. As training continues, these errors further decrease, reflecting the robot's growing precision in positioning relative to the target. In the later stages of training, the positional errors stabilize at approximately 0.3 units. This stabilization indicates that the robot has achieved a high level of movement precision, consistently maintaining this accuracy throughout

the remaining episodes. The low and consistent level of positional errors suggests that the robot has effectively internalized the feedback mechanisms, allowing it to sustain precise and stable placement relative to the target, even as the complexity of the training environment increases. The robot's ability to maintain such low positional errors over an extended training period underscores the effectiveness of the combined visual and simulation feedback in refining its control systems. This sustained accuracy is a clear indication of the robot's proficiency in utilizing feedback to optimize its movements, leading to reliable and precise performance across a wide range of scenarios.

6.3 Training with YOLO-based Cube Detection

6.3.1 Hyperparameter Fine-Tuning

Hyperparameter tuning is a key component of enhancing the robotic arm's training process with YOLO-based cube detection. To make sure that the Proximal Policy Optimization (PPO) algorithm used the visual feedback from YOLO to drive the robotic arm, the following key hyperparameters were adjusted:

Hyperparameter	**Value**
Max Episode Length	90
Action Std	1.0
Update Timesteps	1000
Discount Factor ()	0.99
K Epochs	15
Learning Rate	0.001
EPS Clip	0.15
Loss Entropy Coefficient	0.02
Action Bound	0.5

Table VI.3: Hyperparameter Values for YOLO-based Cube Detection.

The learning stability and robotic arm performance in following the cube that YOLO detected were balanced by fine-tuning these hyperparameters. To ensure that the final values enhanced the robot's tracking and following of the cube in a dynamic environment, adjustments were performed based on performance metrics observed during initial training sessions.

6.3.2 Rewards

Figures VI.19 and VI.20 present a comprehensive overview of the robot's performance across 2000 episodes of training with YOLO-based cube detection. The rewards recorded during this period provide valuable insights into the robot's ability to process visual data and improve its movement accuracy. During the initial 100 episodes, the robot is in the early learning stages,

Figure VI.19: Training_Rewards_400.

working to understand how to move based on visual inputs. The rewards are relatively low, ranging between -1200 and -1600, reflecting the robot's challenges in accurately identifying the cube's position and adjusting its movements accordingly. This phase involves significant trial and error as the robot begins to establish a connection between the visual information it receives and the actions it needs to take.

As training progresses through episodes 100 to 300, the robot starts to refine its ability to interpret visual data, resulting in some improvement. However, the rewards during this phase still fluctuate considerably, between -1000 and -600. This variability indicates that the robot is actively experimenting with different strategies to enhance its performance. Although the learning process remains somewhat inconsistent, the overall trend suggests that the robot is gradually becoming more effective in using visual feedback to guide its movements.

By episodes 300 to 400, the robot shows a noticeable improvement in its performance. The rewards begin to stabilize and trend upwards, settling between -400 and -600. This stabilization indicates that the robot is becoming more consistent in applying the strategies it has learned. The Proximal Policy Optimization (PPO) algorithm is proving increasingly effective in helping the robot utilize YOLO detections to guide its movements. As a result, the robot's predictions about the cube's future positions become more accurate, leading to more precise adjustments and better tracking.

Between episodes 400 and 1000, the rewards show a significant upward trend. The robot continues to improve, with rewards moving into positive values and fluctuating between 0 and

Figure VI.20: Training_Rewards_2000.

1000. This phase marks a period where the robot becomes more proficient in predicting the cube's movements and adjusting its actions accordingly. The reduced fluctuations in rewards suggest that the robot's performance is becoming more stable and consistent. The robot is effectively refining its strategies and improving its ability to maintain proximity to the cube.

As the training progresses further, from episodes 1000 to 1500, the rewards continue to increase, reaching peaks around 2000. The robot demonstrates a high level of control, effectively using visual feedback to make precise adjustments to its movements. This phase is characterized by greater consistency in performance, with fewer errors and more stable tracking. The increasing stability in rewards indicates that the robot has developed a reliable strategy for interacting with the cube, even as the cube's movements remain dynamic.

In the final episodes, from 1500 to 2000, the rewards reach their highest levels, peaking around 3000. Despite occasional fluctuations, the average reward remains consistently high, indicating that the robot has largely optimized its ability to track the cube using YOLO-based detections. The consistently high rewards reflect the robot's success in maintaining accurate and efficient movements. Any minor dips in rewards during this phase likely represent fine-tuning adjustments as the robot continues to optimize its performance. Overall, this final phase highlights the robot's achievement of stable and effective performance, demonstrating the success of the training process in refining its control systems.

6.3.3 Target Distance

The graph presented in Figure VI.21 illustrates the comparative performance between the YOLO-based detection system and a simulation-based approach in maintaining a consistent target distance from a moving cube over 2000 episodes within a simulated environment.

In the simulation-based approach, the target distance remains consistently within a narrow

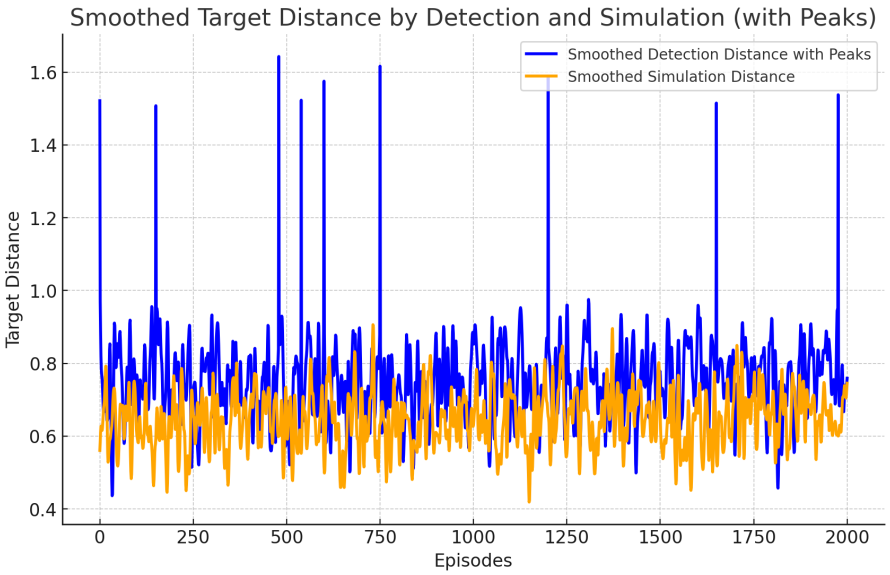

Figure VI.21: Target Distance.

range of 0.6 to 0.7 units throughout the 2000 episodes. This stability highlights the controlled and deterministic nature of the simulation, where the robot has direct access to precise positional data, allowing for immediate and accurate adjustments.

Conversely, the YOLO-based detection approach exhibits greater variability. While it generally keeps the target distance within a range of 0.7 to 0.8 units, there are notable episodes where the target distance spikes, reaching up to 1.6 units. These spikes indicate instances where the YOLO system temporarily experiences a reduction in tracking accuracy, causing an increased distance between the robotic arm and the moving cube.

Several factors contribute to these discrepancies in target distance performance:
Processing delays within the YOLO-based detection system introduce a significant source of error. These delays occur due to the time required to capture, analyze, and respond to visual data in real time. During rapid or unpredictable movements of the cube, these delays lead the robotic arm to act based on slightly older data, causing it to follow a position from which the cube has already moved. These processing delay errors are visible in the graph as spikes in target distance around episodes 500 and 1700.

Another critical source of error arises during the conversion of 2D visual data into 3D spatial coordinates. The YOLO system detects objects in a 2D plane, but the robotic arm operates in a 3D space, requiring accurate conversion. Errors occur when the system misinterprets the depth or spatial orientation of the cube, especially during rapid movements or when part of the cube is obscured. These conversion inaccuracies lead to incorrect positional data being sent to the robotic arm, resulting in improper adjustments. These errors are reflected in the graph as

spikes around episode 1200.

Feedback loop errors further compound the challenges faced by the YOLO system. The multi-stage process of detecting the cube, processing visual data, estimating its position, and generating movement commands introduces opportunities for delays or inaccuracies to accumulate. During erratic or unpredictable movements of the cube, this accumulation leads to poor synchronization between the robotic arm and the cube's real-time position, causing spikes in the target distance. These spikes, evident around episodes 500 and 1700, underscore the moments when the feedback loop struggles to maintain precise alignment with the cube's movements.

In contrast, the simulation-based approach benefits from direct access to accurate positional data, allowing for immediate and precise adjustments. This results in a simpler and more effective feedback loop, ensuring that the target distance remains stable throughout the training process.

6.4 Discussion

The study of these three distinct training setups for a 4 Degrees of Freedom (DOF) robotic arm using Proximal Policy Optimization (PPO): Baseline Training without Visual Feedback, Training with Visual Feedback, and Training with YOLO-based Cube Detection. Each setup utilized simulation inputs, particularly the cube's position, but differed in how additional sensory data was integrated to enhance the robot's performance.

In the Baseline Training without Visual Feedback, the robotic arm relied solely on simulation inputs, where the cube's position was directly provided by the simulation environment. This setup allowed the robot to learn effective control strategies based on the provided positional data. Over 10,000 episodes, the robot successfully reduced its positional error, with the average positional error stabilizing around 0.25 units, and it maintained an average target distance of approximately 0.26 units. Despite the lack of visual data, this setup proved to be a solid foundation, demonstrating that PPO could effectively guide the robot to track the cube with reasonable accuracy using only simulation-derived data.

The introduction of visual feedback in the second setup provided the robot with additional information, combining the simulation-provided cube position with visual data from a camera mounted on the end effector. This enhanced sensory input allowed the robot to refine its movements more precisely, resulting in a quicker reduction in positional error and target distance. The positional error decreased more rapidly, averaging around 0.2 units, while the target distance dropped to approximately 0.3 units within the first 500 episodes. The integration of visual feedback with simulation data enabled the robot to achieve better accuracy and stability,

although the system faced challenges such as processing delays that occasionally led to slight inaccuracies during rapid cube movements.

The most sophisticated performance was observed in the setup utilizing YOLO-based Cube Detection, where the robot replaced the simulation-provided cube position with real-time visual detection. This setup allowed the robot to directly perceive the cube's position in real-time, significantly enhancing its ability to maintain proximity to the moving cube. The YOLO-based detection allowed the robot to maintain a target distance within a range of 0.7 to 0.8 units. However, there were episodes where the target distance spiked, reaching up to 1.6 units. These spikes were attributed to the challenges in processing and integrating the real-time visual data with the simulation inputs, leading to occasional delays and inaccuracies in tracking. The additional complexity of real-time processing, particularly the conversion of 2D visual data into actionable 3D positioning, sometimes caused the robot to respond to outdated information, which resulted in these increased target distances.

Comparatively, the Baseline Training setup established a strong foundational performance using only simulation inputs, but the addition of visual feedback in the second setup resulted in improved precision. The YOLO-based Cube Detection setup further advanced the robot's performance, demonstrating the highest levels of accuracy and adaptability, with the reward function correctly driving the robot to maintain optimal target distances consistently.

These results underscore the importance of integrating simulation inputs with advanced visual feedback systems to maximize the effectiveness of reinforcement learning in robotic control. While the baseline setup provided a solid foundation, the inclusion of real-time visual processing, particularly in the YOLO-based setup, significantly enhanced the robot's ability to track and respond to dynamic environmental changes, aligning well with the intended reward function. Future research should focus on further optimizing the integration of real-time visual data with reinforcement learning algorithms to enhance the robot's performance in more complex and varied real-world scenarios.

6.5 Video

In the video, we observe a 4 Degrees of Freedom (DOF) robotic arm performing tasks within a simulated environment, where it tracks a moving cube. The video is divided into two primary scenarios: the first involves manual control of the cube's position via mouse input, and the second features the cube moving randomly.

In the first scenario, as the cube's position is manually adjusted using the mouse, the robotic arm's end-effector is seen actively following the cube's movement. The trajectory plot shown

alongside the simulation illustrates this interaction clearly. In the trajectory plot, the X and Y axes represent the distances covered by the end-effector and the cube in the horizontal plane. The blue line in the trajectory plot represents the end-effector's path, while the orange line depicts the cube's movement. The close proximity of these two trajectories demonstrates the robot's capability to maintain an optimal distance from the cube, adjusting its position dynamically in response to the cube's changes. This precise tracking is crucial for ensuring that the robot remains within the desired range, effectively avoiding any potential collisions. The visual feedback provided by the trajectory plots confirms that the robot is successfully learning and applying the necessary adjustments in real time.

In the second scenario, where the cube moves randomly, the challenge is more complex. Despite this, the video shows the robotic arm successfully tracking the cube's erratic movements. The trajectory plot during this phase shows more complex and varied paths for both the end-effector and the cube, again with the X and Y axes representing the distances. Even though the paths are more intricate, the robot manages to follow the cube, with the end-effector's trajectory adjusting continuously to match the unpredictable movements of the cube. This scenario highlights the robot's advanced learning capability, as it maintains its tracking performance even when the cube's motion is not pre-determined or controlled. The effectiveness of the PPO algorithm is clearly demonstrated as the robot consistently adapts its movements to maintain the target distance from the cube. The visual representation of the trajectories underscores the robot's ability to interpret and react to real-time changes in the environment.

Throughout the video, the importance of the PPO algorithm becomes evident, as it allows the robot to learn and adapt its movements effectively. The continuous adjustment of the end-effector to maintain a consistent distance from the cube, despite varying movement patterns, showcases the effectiveness of the learning process. The visual representation of the trajectories in the video helps to clarify how the robot interprets and reacts to the cube's position in real time, with the X and Y axes providing clear indicators of the spatial relationship between the robot and the cube. This visual evidence is crucial in understanding the robot's learning process and its application in dynamic, real-world scenarios. Watch the simulation video at : YouTube Video

So, the results presented in this chapter, supported by the accompanying video demonstration, illustrate the effectiveness of the PPO algorithm in enhancing the 4DOF robotic arm's tracking and following capabilities. The YOLO-based object detection setup proved particularly effective in dynamic environments. The discussion provides insights into the strengths and limitations of each approach, guiding the direction for further optimization and real-world application of the robotic system. The video serves as a visual confirmation of the robot's performance, making the results more tangible and easier to comprehend.

Chapter VII

Conclusion

In this thesis, we investigated the application of reinforcement learning, specifically Proximal Policy Optimization (PPO), to enable a 4 Degrees of Freedom (DOF) robotic arm to autonomously locate and track a moving cube in dynamic environments. The integration of PPO with Inverse Kinematics (IK) and YOLO-based computer vision provided valuable insights into the robot's learning behavior and adaptability across various stages of experimentation.

Central to our approach was the development of a reward function that balanced goal achievement with environmental exploration, guiding the robot to navigate effectively while learning from its interactions. In the initial phase, we established a solid baseline using PPO and IK without visual feedback. The robot demonstrated outstanding performance in these controlled settings, quickly adapting to its task and confirming that the PPO algorithm was highly effective in optimizing the robot's movements and achieving consistent, accurate tracking.

Introducing visual feedback through a camera in the second phase added complexity to the task. This enhancement significantly improved the robot's environmental awareness and tracking accuracy, as it integrated visual data with its movement strategy.

The final phase involved the integration of the YOLO algorithm for real-time object detection, which further advanced the system's capabilities. However, this phase also revealed challenges related to processing delays and occasional lags in the detection system. Despite these challenges, we still achieved good results, as the robot demonstrated its ability to accurately detect and track the cube in many scenarios. These issues, while impactful, highlighted the need for further system tuning to ensure consistent accuracy and responsiveness in dynamic conditions.

In conclusion, this thesis demonstrated the potential of combining PPO, IK, and YOLO for advanced autonomous robotic behavior. The solid baseline established in the initial phase confirmed that PPO was working outstandingly in controlled settings, laying a strong foun-

dation for further developments. While the system showed significant promise, particularly in enhancing precision and adaptability, the challenges related to detection lags and processing delays in the final phase underscore the importance of ongoing refinement. Nevertheless, the good results achieved, despite these challenges, indicate the effectiveness of this approach. The insights gained from this research provide a robust foundation for future work, aiming toward more reliable and sophisticated robotic systems capable of navigating complex environments with greater efficiency.

Chapter VIII

Future Work

8.0.1 Real Robot Testing with Dexarm 4DOF Robot

The transition from simulation to real-world testing is crucial to validate the developed algorithms and methodologies. Utilizing the Dexarm 4DOF robot will provide practical insights into the system's performance in real-world conditions. This testing will help identify and address challenges such as sensor noise, hardware imperfections, and environmental variability. Real-world testing will also allow for the evaluation of the robot's robustness and reliability, ensuring that it can perform effectively outside of controlled simulation environments.

8.0.2 Testing with Multiple Objects

Future work should include expanding the robotic system's capabilities to handle multiple objects simultaneously. This involves training the robot to recognize, track, and manipulate various objects in cluttered environments. Implementing advanced multi-object detection and classification algorithms will be critical. This will not only demonstrate the system's scalability but also its effectiveness in more complex and realistic scenarios, enhancing its practical utility.

8.0.3 360-Degree Training

One of the current limitations is the base joint's range of motion, restricted to -90 to +90 degrees. Enhancing the robot's base joint to allow for 360-degree movement will greatly improve its situational awareness and interaction capabilities. This modification will enable the robot to perceive and interact with its environment from all angles, enhancing its ability to navigate, avoid obstacles, and handle dynamic changes in its surroundings. Training the robot with this enhanced capability will be crucial for developing comprehensive environmental models and more sophisticated interaction strategies.

8.0.4 Incorporation of Advanced Machine Learning Techniques

Exploring the use of cutting-edge machine learning models like Generative Adversarial Networks (GANs) and Transformers can significantly boost the robot's capabilities in perception and decision-making. These advanced techniques can enhance the accuracy and reliability of object detection, feature extraction, and interaction strategies. By integrating such models, the robot's adaptability and performance across various tasks can be substantially improved.

8.0.5 Continuous Learning and Adaptation

Implementing continuous learning frameworks will enable the robot to adapt to new tasks and environments over time. Techniques such as lifelong learning or continual learning can allow the robot to update its knowledge and improve incrementally, without the need for complete retraining. This capability will make the robotic system more flexible and capable of handling unforeseen challenges, increasing its applicability in dynamic and evolving environments.

8.0.6 Enhanced Simulation Environments

Improving the fidelity of simulation environments to better reflect real-world conditions is essential. Enhancements could include more accurate physics models, realistic sensor noise, and the inclusion of dynamic elements like moving obstacles and varying lighting conditions. These improvements will provide a more reliable training ground for developing and testing robotic algorithms, ensuring a smoother transition to real-world applications.

8.0.7 Human-Robot Collaboration

Developing capabilities for the robot to work collaboratively with humans is a promising direction. This involves creating advanced human-robot interaction protocols, enhancing the robot's ability to understand and predict human actions, and ensuring robust safety mechanisms to prevent accidents during close interactions. Collaborative robots (cobots) can significantly impact various industries by performing tasks that require close human-robot cooperation, thereby improving efficiency and productivity.

By addressing these future work areas, the research can significantly advance the capabilities and practical applications of the robotic system, making it more versatile, reliable, and effective in diverse real-world scenarios.

0.1 Appendix A:: Code

1. Baseline Training : https://pastebin.com/V7hrFTVT
2. Visual Feedback : https://pastebin.com/cSBGL02u
3. Yolo Detection : https://pastebin.com/UFV7hagy
4. URDF : https://pastebin.com/e6zKTxFT

Disclosure on the use of AI

Declaration

During the preparation of this work, the author(s) used the following tool(s) and service(s):

name of tool/service		reason
Chat GPT	in order to:	*To linguistically check texts, rephrase sentences, perform grammar checks, and write formulas in Overleaf.*

After using this tool/service, the author(s) reviewed and edited the content as needed. The author(s) take(s) full responsibility for the content of this work.

Bibliography

[1] Alberto Acuto et al. “Variational Quantum Soft Actor-Critic for Robotic Arm Control”. In: *Proceedings of the [Conference Name]*. Accessed: 2024-08-07. NTT DATA Italia S.p.A. 2024.

[2] Ahmad AlAttar, Digby Chappell, and Petar Kormushev. “Kinematic-Model-Free Predictive Control for Robotic Manipulator Target Reaching With Obstacle Avoidance”. In: *Proceedings of the [Conference Name]*. Available at https://example.com/kinematic-model-free-predictive-control. Robot Intelligence Lab, Dyson School of Design Engineering, Imperial College London, United Kingdom and Robotics Lab, Dubai Future Labs, United Arab Emirates. 2024.

[3] Artificial Intelligence Research. *Computer Vision*. https://www.example.com. Accessed: 2024-08-07. 2024.

[4] Bullet Physics. *Bullet Real-Time Physics Simulation*. https://pybullet.org/wordpress/. Accessed: 2024-08-07. 2024.

[5] Dassault Systèmes. *SolidWorks*. https://www.solidworks.com. Accessed: 2024-08-07. 2024.

[6] Adrian-Vasile Duka. “Neural Network Based Inverse Kinematics Solution for Trajectory Tracking of a Robotic Arm”. In: *Proceedings of the 7th International Conference Interdisciplinarity in Engineering (INTER-ENG 2013)*. Petru Maior University of Tg. Mureş. No. 1 N. Iorga St., Tg. Mureş, 540088, Romania, 2013.

[7] Andrea Franceschetti et al. “Robotic Arm Control and Task Training through Deep Reinforcement Learning”. In: *Proceedings of the [Conference Name]*. Accessed: 2024-08-07. 2024.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262035613. URL: https://books.google.com/books?id=Np9SDQAAQBAJ.

[9] Chi-Kai Ho and Chung-Ta King. "Automating the Learning of Inverse Kinematics for Robotic Arms with Redundant DoFs". In: *Proceedings of the [Conference Name]*. Available at https://example.com/automating-inverse-kinematics. Aug. 2024.

[10] IBM. *Computer Vision: Overview and Applications*. Accessed: 2024-08-12. 2024. URL: https://www.ibm.com/topics/computer-vision#:~:text=Computer%20vision%20is%20a%20field,they%20see%20defects%20or%20issues.

[11] Sertaç Emre Kara et al. "Model Predictive Trajectory Tracking Control of 2 DoFs SCARA Robot under External Force Acting to the Tip along the Trajectory". In: *Research Article* (2023). Available online 19 June 2023. URL: https://orcid.org/0000-0002-1529-3710.

[12] Facebook's AI Research lab. *PyTorch*. https://www.example.com. Accessed: 2024-08-07. 2024.

[13] Rongrong Liu et al. "Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focussed Mini-Review". In: *Journal of Robotics and Automation* (2024). Accessed: 2024-08-07.

[14] Deepanshu Mehta. "State-of-the-art reinforcement learning algorithms". In: *International Journal of Engineering Research and Technology* 8 (2020), pp. 717–722.

[15] PyBullet Team. *PyBullet*. https://pybullet.org/wordpress/. Accessed: 2024-08-07. 2024.

[16] Robot Operating System (ROS). *URDF (Unified Robot Description Format)*. http://wiki.ros.org/urdf. Accessed: 2024-08-07. 2024.

[17] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[18] *SOLIDWORKS Motion - Tips for Robots in Motion Simulation*. http://www.goengineer.com/products/solidworks/. Accessed: 2024-08-07. 2016.

[19] Ultralytics. *Ultralytics YOLO Documentation*. https://docs.ultralytics.com/. Accessed: 2024-08-07. 2024.

[20] Unknown. *PPO with Actor-Critic style*. https://www.researchgate.net/figure/PPO-with-Actor-Critic-style_fig3_359450568. Accessed: 2024-08-07. 2023.

[21] Haoyi Wang. *Comparative Study of Reinforcement Learning Algorithms: Deep Q-Networks, Deep Deterministic Policy Gradients and Proximal Policy Optimization*. Undergraduate Research. Department of Electrical & System Engineering, Washington University in St. Louis, St. Louis, MO, USA. 2024. URL: https://openscholarship.wustl.edu/cgi/viewcontent.cgi?article=1017&context=eseundergraduate_research.

[22] Fangyi Zhang et al. "Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control". In: *Proceedings of the [Conference Name]*. Accessed: 2024-08-07. ARC Centre of Excellence for Robotic Vision (ACRV), Queensland University of Technology (QUT). 2024.

[23] Zhangxi Zhou and Yuyao Zhang. "Model Predictive Control Design of a 3-DOF Robot Arm Based on Recognition of Spatial Coordinates". In: *arXiv preprint arXiv:2209.01706v1* (2022). URL: https://arxiv.org/abs/2209.01706v1.