Universitat
de les Illes Balears

MASTER'S THESIS

WHOLE-BODY MANIPULATION USING REINFORCEMENT LEARNING

Minahil Raza

Master's Degree in Intelligent Systems (MUSI)

Specialisation: Computer Vision

Centre for Postgraduate Studies

Academic Year 2022-23

WHOLE-BODY MANIPULATION USING REINFORCEMENT LEARNING

Minahil Raza

Master's Thesis
Centre for Postgraduate Studies
University of the Balearic Islands

Academic Year 2022-23

Key words:
Whole-body manipulation, reinforcement learning, hierarchical reinforcement learning, robotics, autonomous agents

Thesis Supervisor's Name: Murtaza Hazara
Thesis Supervisor's Name: Alberto Ortiz Rodriguez

Whole-body Manipulation using Reinforcement Learning

Minahil Raza
Tutors: Murtaza Hazara, Alberto Ortiz Rodriguez
Master's Thesis in Intelligent Systems (MUSI)
Universitat de les Illes Balears
07122 Palma, Illes Balears, Spain

***Abstract*—Recent advancements in artificial intelligence(AI) have revolutionized the field of robotics. One of the most intriguing use cases in this domain is whole-body manipulation. Whole-body manipulation combines the precision of robotic manipulators with the expanded reach of mobile platforms. This thesis explores the task of autonomous whole-body manipulation using reinforcement learning (RL). By leveraging RL's ability to learn from experience and adapt to new scenarios, we aim to navigate and manipulate a robot jointly. First, we explore RL for navigation and manipulation separately. After developing a keen understanding of these tasks and training successful RL agents, we move towards joint navigation and manipulation. We conduct experiments using different training methods to combine these tasks under the paradigm of hierarchical RL (HRL) to achieve autonomous whole-body manipulation. The resulting RL agent is capable of successfully reaching a target location outside the operating range of the arm without collisions. In conclusion, we provide an example of the future potential of HRL for complex tasks within the domain of robotics.**

***Index Terms*—whole-body manipulation, reinforcement learning, hierarchical reinforcement learning, robotics, autonomous agents**

I. Introduction

The recent advances in artificial intelligence (AI) have paved the way for autonomous robots that can navigate and perform complex manipulation tasks. One particularly intriguing area of development in this field is whole-body manipulation, a remarkable capability that enables robots to interact with their environment using their entire body. Rather than relying solely on the capabilities of a robotic manipulator, mounting it on a mobile base can lead to a wide range of coordinated movements. The resulting mobile robotic manipulator combines the expansive working space of the mobile platform with the flexible operating area of the robotic arm [31].

This integration opens up a multitude of possibilities in research as well as various industries. A prime example is Little Helper [11]. Developed by Aalborg University, it pioneered the concept of industrial mobile manipulators. While primarily serving as a research platform, Little Helper has showcased its capabilities in real-world scenarios, including industrial applications such as machine tending, as well as in service robotics, notably as a host at the Scandinavian Expo [27]. Mobile manipulators can be useful in the service industry. For instance, they can be used for transporting objects [13], opening doors [4], performing cleaning duties [16], and pushing carts [21], among many other application domains.

While many works have focused on joint navigation and manipulation of robots using traditional control-based approaches, current mobile manipulation tasks present a number of unsolved challenges which are a focus of current research. Primarily, the mobile base and the fixed manipulator have been addressed as distinct research areas, making the integration of both into a cohesive whole-body control system a complex and demanding task. One approach to integration is to separate the two tasks entirely. In other words, the manipulator and the mobile platform are controlled independently as two distinct subsystems [5, 24], although this hard decoupling has a major drawback since it fails to capture the impact of the manipulation task on the locomotion task and vice versa. For instance, the movement of the arm may cause the robot wheels to slip leading to imprecise control. The better approach is to control the dynamics of the whole body. This approach is also known as whole-body control or whole-body manipulation as this strategy considers the overall dynamics of the system. Whole-body manipulation has shown exceptional results for a wide range of tasks [12, 15, 18, 19].

Recently, reinforcement learning (RL) [26] has been emerging as a popular paradigm for various tasks in robotics including manipulation and navigation. In the RL framework, the controllers do not have to be explicitly programmed for the task. Rather, near-optimal to optimal behaviors are learned explicitly through interaction with the environment. In other words, RL enables robots to learn and master complex tasks through trial and error. RL has been explored for navigation, path planning, and obstacle avoidance of mobile robots [3, 6, 7, 20, 23] as well as for the control of static manipulators [29, 30]. However, relatively few works focus on the task of whole-body manipulation [25]. This is primarily due to the complex nature of the task. Whole-body manipulation has greater degrees of freedom (DOFs) and RL must learn the effect of each DOF on the overall motion of the robot. Therefore, a large number of iterations are required for training an RL agent to successfully perform a task. Moreover, the rewards have to be tuned according to the task at hand. One of the key hindrances in RL training is to ensure that the RL agent explores in the right direction and obtains meaningful samples. If the RL agent is exploring in an efficient way, it eventually converges to optimal behavior. These challenges lead to the motivation behind this work.

A. Problem Statement

In this thesis, we investigate the potential of RL for whole-body mobile manipulation using a KELO robile [1] with a Franka Research Arm attached [9] as our target platform. The primary goal is to learn an RL policy that is able to jointly navigate and manipulate. Moreover, we want to achieve our objectives through simple reward functions, state, and action definitions. Lastly, we want to ensure that the RL agents explore efficiently and learn optimal behavior in a sample-efficient manner.

B. Research Aim and Objectives

Having discussed the background and motivation for targeting this problem, the aim of this thesis is to put forward a whole-body control approach for joint navigation and manipulation through RL. To achieve the stated aim, in this thesis, we have identified and explored the following research objectives:

- Design of well-defined RL tasks for navigation, manipulation and whole-body manipulation of the robot.
- Implementation of the corresponding environments using the OpenAI Gym's [2] APIs.
- Investigation of RLlib to enable parallelism in training, allowing effective use of multiple workers (i.e. multiple CPU threads).
- Training of the RL agents for the aforementioned tasks using different configurations, algorithms and hyperparameter values.
- Definition of the qualitative and quantitative metrics to evaluate the trained RL agents.
- Investigation of the potential and sample efficiency of hierarchical RL (HRL) for whole-body manipulation.

C. Delimitations

Due to the non-availability of the mobile manipulator at the time of conducting the research for this thesis, all the experiments and analyses were carried out within a simulated environment, where the agent was trained using RL and evaluated. The simulation environment was provided by Nokia.

D. Structure of the Thesis

This work is structured as follows: Section II contains a brief overview of RL, deep RL and HRL; Section III presents the high-level architecture of our system, detailing all the tools used for this thesis; Section IV presents the design of the RL tasks, along with the details of the neural network (NN) architectures; Section V defines the metrics used for reporting results and presents them in detail; lastly, Section VI ends with the conclusions obtained from the work done, pointing the reader to the future research directions in this domain.

II. Background

In this section, we provide an overview of the concepts and algorithms used in this thesis. First, we provide a brief overview of RL, along with its mathematical formulation. Next, we also provide a background on HRL approaches.

A. Reinforcement Learning

RL is a computational approach to learning where an *agent* interacts with an *environment* and learns to make decisions based on received feedback in the form of *rewards* or punishments [26]. The key idea behind RL is to make an agent learn without explicitly telling it which actions to take. Unlike supervised learning, RL does not require data and ground truth to learn a mapping function. RL is also different from unsupervised learning where the goal is to identify the underlying patterns in data. Hence, RL is the most suitable for interactive problems compared to supervised and unsupervised learning.

Interactive problems can be solved using RL by appropriately defining several key terms, including the following:

- **Environment:** An environment refers to a simulated or physical world in which the RL agent operates. It provides the rules, dynamics, and interactions that shape an agent's experience.
- **Agent:** An RL agent is the entity that interacts with the environment. At each step, it observes the environment and takes an action accordingly. The agent's objective is to maximize its cumulative rewards over time.
- **Observation:** As the agent interacts with the environment, changes its own state and maybe the state of the environment. For example, while navigating, the robot would be changing its position, which leads to a change in state. An observation is a part of the environment state which is visible to the RL agent.
- **Action:** The agent can take an action based upon its observations. The action depends on the task. For example, during navigation, an action can consist in the velocities of the mobile platform.
- **Policy:** The internal decision-making process of the RL agent is guided by its policy. A policy is simply a mapping from the set of states to the set of actions. When we say that the RL agent is learning, we imply that it is improving the estimate of its policy. Ultimately, the aim is to train the agent so that it learns the optimal policy.
- **Reward:** The agent learns a policy through a trial-and-error process, exploring different actions and receiving rewards. A reward is a feedback signal from the environment as a response to the agent's actions. By receiving rewards, the agent can assess the consequences of its actions and adjust its behavior accordingly. A reward function typically depends only on the state, but other more complex configurations can be adopted.
- **Reward engineering:** Designing reward is critical for the successful convergence of a policy. Rewards are the only way to encourage the good actions and prevent the bad ones. The process of tuning the reward function is called reward engineering.
- **Episode:** The RL agent interacts with the environment over discrete timesteps. An episode represents a complete sequence of interactions between the agent and the environment, typically starting from an initial state and continuing until a terminal state is reached. At the beginning of an episode, the agent perceives its initial

state from the environment. It selects an action based on its current policy and interacts with the environment by executing the chosen action. The environment transitions to a new state, and the agent receives a reward associated to the new state attained. The agent continues to observe states, choose actions, and receive rewards, progressing through the episode. The episode concludes when a specific condition is met, such as reaching a goal state or exceeding a maximum number of time steps. At the end of an episode, the agent evaluates the cumulative reward it has obtained during that episode.

B. Markov Decision Process (MDP)

An MDP allows us to define and formalize a decision-making process. An MDP can be defined as a tuple (S, A, P, R, γ) where S is a set of valid states (called the state space), A is a set of valid actions (called the action space), $P(s, a, s')$ represents the probability of transitioning from one state to the next, $R(s, a, s')$ is the reward function and γ is the discount factor for future rewards. The problem in MDPs is to find a policy π, which is a mapping from the set of observed states S to the set of actions A.

MDPs are important in RL because they allow modelling and solving sequential decision-making problems. First, they capture the sequential structure of a problem. Under the Markov assumption, the current state is supposed to contain all the necessary information to make the next decision. This property allows the RL algorithm to disregard the history of actions and states which occurred before the current one, greatly simplifying the decision-making process. Moreover, MDPs capture the stochastic nature and uncertainty of real-world problems using the concept of state transition probability, enabling RL agents to handle uncertain and dynamic environments. Lastly, MDPs provide a way for the agent to look at the long-term reward from the current state to the end of the episode, called *return*, being expressed as G_t in Equation 1 through the concept of discount::

$$G_t = \sum_{i=t}^{\infty} \gamma^i R(s_i, a_i, s_{i+1}) \tag{1}$$

where $a_i = \pi(s_i)$ is the action recommended by the policy, and γ is the discount factor, which satisfies $0 \leq \gamma \leq 1$. This equation allows us to assign importance to future rewards so that the RL agent is not merely considering the immediate consequences of its actions. The RL agent is more short-sighted in considering rewards as the value of γ is closer to 0.

C. Search for the Optimal Policy

When we formulate a problem as an MDP, our goal is to find the optimal policy $\pi^*(a|s)$. Therefore, we need a way to compare two policies and determine which one is better. For this purpose, we use the concept of the *value* of a state. The value of a state $V_\pi(s)$ indicates how favorable or unfavorable it is to be in that state, after following policy π. While the return G_t represents how good or bad a state is for the agent from the time instant t until the end of the episode, $V_\pi(s)$ provides complete information about a state. We accomplish this by finding the *expected return* from a state. In other words, due to the stochastic nature of the environment, we cannot rely on a single value of the return. Rather, we focus on the expected return from each state when following a certain policy π. This expected return is called the value of a state. Similarly, the action-value of a state-action pair $Q_\pi(s, a)$ is the expected return when the agent takes an action a in a state s and follows the policy π thereafter.

The Bellman equations are a set of fundamental equations in RL that express the relationship between $V_\pi(s)$ or $Q_\pi(s, a)$ and the expected future rewards, thereby providing a way to evaluate and improve a policy. They allow us to obtain a recursive formulation for calculating the expected return without the need to add all the rewards up to the terminal state. Ultimately, they permit to define ways to find $V^*_\pi(s)$ or $Q^*_\pi(s, a)$. The Bellman expectation equation for the value function $V(s)$ represents the expected cumulative reward from being in state s and following the policy π thereafter:

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V_\pi(s')|s_t = s] \tag{2}$$

Similarly, the Bellman expectation equation for the action-value function $Q(s, a)$ represents the expected cumulative reward from taking action a in state s and following the policy π thereafter.

$$Q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_\pi(s', a')] \tag{3}$$

D. Deep Reinforcement Learning

Deep RL is a sub-field of RL that combines the fundamentals of RL with NNs to handle complex, continuous and/or high-dimensional state and action spaces. It involves using NNs as function approximators to represent the parameters of the policy and/or the value function. Thus, the goal of deep RL is to find a good policy by optimizing the parameters of the NNs involved.

When using NNs, the loss or objective function L serves as a critical steering for the learning process. It represents a measure of the agent's performance, which the learning algorithm seeks to maximize or minimize. To optimize the objective function, the most common approach is to use stochastic gradient descent (SGD) or any of its variants such as Adam [14]. The learning algorithm samples experiences (state transitions and rewards) from the environment, and, on the basis of these samples, computes the gradients of the objective function ΔL with respect to the NN parameters. These gradients indicate the direction in which the parameters should be adjusted to improve the agent's performance. The optimizer then iteratively updates the network parameters in the direction of these gradients, effectively improving the corresponding estimation of the policy or the value function over time.

The choice of the objective function L depends on the specific RL algorithm and the learning framework being used. Through the optimization process, the deep RL agent learns to improve its policy or value function and, consequently, its performance for the given task or environment.

E. Classification of RL algorithms

The RL literature provides different ways to classify RL algorithms. Based on whether the algorithms explicitly learn a model of the environment or directly learn the policy or value function without modeling the environment dynamics, two broad categories can be defined, namely model-based RL and model-free RL. The details of these categories are as follows:

- **Model-based RL:** The agent uses the observed data to learn a model of the environment allowing it to simulate future states and rewards.
- **Model-free RL:** The agent focuses on estimating the value of states or state-action pairs without explicitly modeling the dynamics of the environment.

Another method to classify RL algorithms is based on their interaction with the environment and the data collection strategy, namely on-policy RL and off-policy RL. The details of these categories are as follows:

- **On-policy RL:** On-policy RL algorithms act and update the same policy. Therefore, exploration of the environment and learning are based on the same policy.
- **Off-policy RL:** Off-policy RL algorithms, in contrast, learn and update the policy using data generated by a different policy. The agent can explore the environment using any policy, often an exploratory policy, and learn from the collected experience without being constrained to the current policy.

The choice between these categories is dependent upon the problem. Given the uncertainty and complexity of the operating environment, recent research has been focusing on model-free RL. Both on-policy and off-policy methods are commonly used. In this thesis, we focus on model-free RL. In our experiments, we use an on-policy algorithm called Proximal Policy Optimization (PPO) [22] and an off-policy method called Soft Actor-Critic (SAC) [8].

F. Proximal Policy Optimization (PPO)

PPO is a popular and efficient model-free, on-policy RL algorithm used for optimizing policy functions through small and consistent updates [22]. PPO belongs to the family of policy gradient methods and is specifically designed to address some of the challenges associated with traditional policy gradient algorithms, such as high variance and instability. The key idea behind PPO is to perform multiple updates on the policy by taking small steps to ensure that the policy remains close to the previous one, thus preventing large policy changes that can lead to instability.

One of the main components of PPO is the objective function, which consists of two terms: the clipped surrogate objective and an entropy bonus term. The clipped surrogate objective helps to limit the policy update to a *trust region*, ensuring that the new policy remains close to the old one. The entropy bonus term encourages exploration by penalizing policies that are overly certain in their actions. In other words, the entropy bonus term encourages the policy to take different actions and explore the environment in a better way.

The objective function for PPO can be defined as follows:

$$L_t(\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t, \text{clip}\Big(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\Big)\hat{A}_t\Big) - \beta\mathcal{H}(\pi_\theta(\cdot|s_t))\Big] \tag{4}$$

where $L_t(\theta)$ is the objective function at time step t with respect to policy parameters θ, $\hat{\mathbb{E}}_t$ represents the empirical expectation over a batch of experiences collected from the environment, $\pi_\theta(a_t|s_t)$ is the probability of selecting action a_t given state s_t according to the current policy with parameters θ, $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability of selecting action a_t given state s_t according to the previous policy with parameters θ_{old}, $\hat{A}_t$ is the estimated advantage function at time step t, clip$(\cdot)$ is a clipping function that constrains the policy update, ϵ is a hyperparameter that determines the extent of the clipping, β is a hyperparameter controlling the weight of the entropy bonus term and $\mathcal{H}(\pi_\theta(\cdot|s_t))$ is the entropy of the policy distribution, encouraging exploration.

G. Soft Actor-Critic (SAC)

SAC is a model-free RL algorithm designed for solving continuous action space tasks. SAC belongs to the family of actor-critic algorithms and is based on the maximum entropy RL framework. It has gained significant attention due to its effectiveness in complex environments and its ability to handle stochastic policies. We briefly explained the concept of entropy when discussing PPO. In summary, policy entropy is a concept in RL that quantifies the uncertainty or randomness in the policy's action selection. In the context of stochastic policies, policy entropy measures the average level of surprise or unpredictability in the agent's actions given different states. A policy with higher entropy implies that the agent's actions are more diverse and uncertain, promoting exploration and adaptability in complex and uncertain environments. Maximizing policy entropy encourages RL agents to explore more effectively and discover potentially better solutions during the learning process.

The key idea behind SAC is to encourage policies that are both optimal and stochastic, leading to improved exploration and better handling of uncertainties in the environment. The central objective is to maximize the expected reward while also maximizing the policy's entropy. This is achieved through the use of an objective function that combines the expected reward and an entropy regularization term. The entropy term encourages the policy to produce actions with high uncertainty, which is beneficial for exploring diverse and potentially better solutions. The SAC objective function can be expressed as follows:

$$J(\theta) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\Big[\sum_{t=0}^{T}\gamma^t\big(r(s_t,a_t) + \alpha\mathcal{H}(\pi_\theta(\cdot|s_t))\big)\Big] \tag{5}$$

where $J(\theta)$ represents the objective function to be maximized with respect to the policy parameters θ, (s_t, a_t) denotes a state-action pair sampled from the state-action distribution

ρ_π, $r(s_t, a_t)$ is the immediate reward received when taking action a_t in state s_t, γ is the discount factor determining the importance of future rewards, $\mathcal{H}(\pi_\theta(\cdot|s_t))$ is the entropy of the policy π at state s_t calculated over the set of actions, and α is a balancing parameter that controls the trade-off between the expected reward and the policy entropy.

H. Hierarchical Reinforcement Learning

HRL is an approach for solving complex tasks by decomposing them into sub-tasks or sub-goals [10]. In traditional RL, an agent interacts with an environment to learn a policy that directly maps states to actions. However, in many real-world scenarios, tasks can be large and complex, making it difficult for a single agent to learn an effective policy. Hierarchical RL addresses this challenge by introducing additional levels of hierarchy in the learning process. It involves learning multiple policies, each responsible for a different level of decision-making. These policies can be seen as controllers that manage various sub-tasks or temporal abstractions.

A two-level hierarchical structure typically consists of the following levels:

- **High-level policy (or meta-controller):** This policy is responsible for selecting and switching between different sub-policies or options. It decides which lower-level policy should be executed at any given time based on the current state of the environment.
- **Low-level policies (or options):** These policies are responsible for controlling the agent's actions within a sub-task or during a specific time interval. Each low-level policy is specialized in handling a specific part of the task or a particular situation.

HRL provides a way to decompose tasks into sub-tasks which can lead to faster learning as the agent can focus on mastering smaller components before tackling the entire task. By reusing low-level policies for different high-level tasks, the agent can save samples and improve data efficiency. Moreover, HRL naturally handles tasks that have a clear hierarchical structure, which is common in real-world scenarios such as in whole-body manipulation.

There are several ways to train policies in the HRL framework. While all the policies can be initialized at random and trained in a multi-agent setting, it is often beneficial to train low-level policies separately and reload them while training the high-level policy from scratch. In this thesis, we explore both training methods for whole-body manipulation.

III. System Architecture

In this section, we discuss the architecture for the whole-body manipulation system, focusing on the three main components, namely Simulation, OpenAI Gym environment and RLlib-based algorithm implementations. The high-level architecture of the whole-body manipulation system used to conduct experiments for this thesis is shown in Figure 1. All the tools, libraries and frameworks are open source.

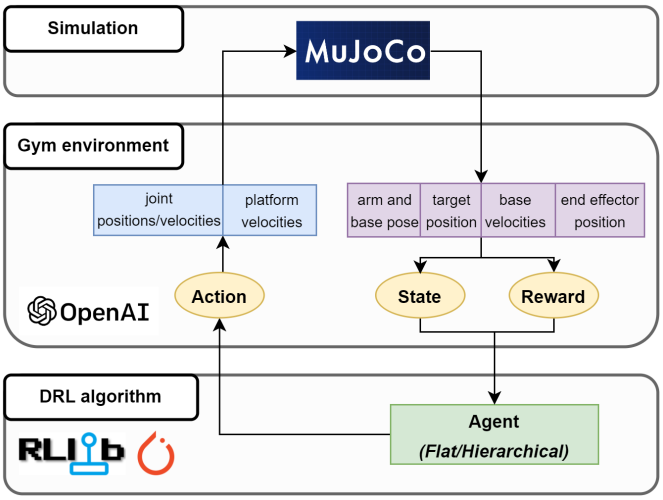


Figure 1: High-level overview of the system architecture

A. Simulation

An RL agent requires multiple interactions with the environment to learn an optimal policy. Using real robots for training poses logistic and safety concerns. Therefore, simulators are used for training RL agents before transferring them to the physical robot. It is essential to opt for a simulator that provides accurate and efficient physics simulations, incorporating realistic dynamics, contacts, and friction. There are many different simulators available for robotics simulation, each with its own pros and cons. For this thesis, we simulate the robot in *Mujoco*. It is a highly versatile and widely adopted physics engine, developed by OpenAI, commonly used for simulating complex robotic systems and physical interactions [28]. Its widespread adoption and support within the research community make it a reliable choice for creating realistic and challenging environments for evaluating the performance of RL algorithms.

In this thesis, we consider a mobile robotic manipulator constructed by assembling two components: a KELO robile mobile base and a Franka Research 3 (FR3) Arm, as shown in Figure 2.

The KELO robile platform has been developed by KELO robotics. It is equipped with four caster wheels, free to rotate in any direction. The motion is similar to the wheels attached to furniture such as office chairs. Each caster wheel consists of two hub wheels, which can be commanded independently. Based on the design, there are two ways of controlling the mobile platform, both in real-world and in the simulation framework:

- **Passive Wheel Control**: A force can be applied on the mobile base by specifying velocities in the X and Y directions v_x and v_y along with a rotational velocity ω. Individual wheels move under the effect of the platform force.
- **Active Wheel Control**: The velocity of each wheel is set independently. The relative velocities of the wheels produce movements along different axes.

One of the key novelties of the KELO mobile platform is

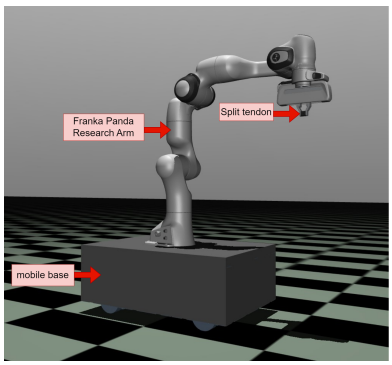


Figure 2: Three-dimensional model of the KELO Robile platform fitted with a 7-DOF FR3 Panda manipulator

the ability to move it along any specified axis. For example, if a velocity of 0.707 m/s is provided along the X and Y axes, the robot will move along a 45° line with a net velocity of 1 m/s. This design provides additional maneuverability compared to other mobile platforms.

The FR3 arm is designed to offer precise and flexible manipulation capabilities, making it well-suited for research and industrial applications. It features seven DOFs, enabling a wide range of complex movements and grasping tasks. The arm incorporates torque sensors in each joint, providing accurate force feedback and enabling safe human-robot interaction. The aforementioned seven DOFs can be controlled using various methods including Joint Position Control, Joint Velocity Control, and Joint Torque Control. For this thesis, we are primarily focusing on position and velocity control.

The simulation engine for simulating the robot is named *KeloFrankaSim*, after its components. It has been designed, developed, and tuned by Nokia Bell Labs. It has been developed using Python bindings, making it easier to incorporate it with open source Python libraries for RL. In addition to the mobile base and the FR3 arm, a split tendon has been attached to the end of the arm as well. This split tendon serves as the end effector. In robot manipulation, an end effector refers to the device or tool attached to the end of a robotic arm or manipulator that enables it to interact with the environment.

At each timestep, a control command can be applied to the robot, which includes the mobile base velocities and/or the positions or velocities of the FR3 joints. As a consequence, in the simulator, the state of the robot changes, and the corresponding data structures are updated including the joint positions, base position, and end effector position. This updated state data can be read using Python APIs.

B. OpenAI Gym

To facilitate the development and evaluation of RL algorithms, OpenAI Gym [2] has emerged as a widely-used toolkit. It comes with a collection of diverse environments ranging from Atari video games to robotic motion control. Moreover, it provides standard, well-documented APIs to design custom environments. These APIs provide communication between the environment and the learning algorithm. The *step* function takes an *action* to advance the simulation. It returns the updated state of the environment (*observation*) and the *reward* to the learning algorithm. Once a terminal state is reached, the *reset* function allows to start a new episode. Additionally, the OpenAI Gym environments provide other auxiliary functions and attributes, such as *action_space* and *observation_space*, which define the valid set of actions and observations respectively. These functions and attributes help agents understand and interact with the environment more effectively. Moreover, they simplify the job of researchers and developers, allowing them to focus on the task design instead of fiddling with the communication between the agent and the environment.

C. RLlib

RLlib [17] is an open-source library designed to provide robust support for distributed RL workloads in production environments. RLlib offers a user-friendly and flexible framework for implementing and training RL agents across a wide range of problem domains. It includes a collection of state-of-the-art algorithms, such as PPO [22] and SAC [8]. RLlib supports distributed training, allowing for efficient utilization of computational resources and accelerating the training process. By specifying the number of workers $N_{workers}$, multiple copies of the environment can be spawned, which run in parallel to accelerate training as shown in Figure 3. Within the RLlib framework, RL algorithms can be trained on multiple workers across several iterations N_{iter}. At the start of each iteration, each worker begins to advance the simulation by the maximum steps specified in the iteration or until an episode is complete. At the end of the iteration, each worker stores the collected sample in a buffer $\mathbb{B}$ which is used to update the parameters of the policy and the value function. This allows us to collect more training samples in each iteration.

RLlib also provides support for both single-agent and multi-agent settings. One of the goals of this work is to explore the potential of HRL for whole-body manipulation. The MultiAgent environment provided by RLlib has simplified in a significant way the development of this part of the work done. A multiprocessor server provided by Nokia has been used for accelerated training inside RLlib.

IV. Design and Implementation

In this section, we detail the design and development process for the description of RL tasks. We describe the navigation and manipulation tasks in detail, followed by the description of the hierarchical tasks for whole-body manipulation. Finally, we provide details of our NN architectures and the hyperparameter values used for training.

A. Parameterization of RL Tasks

Tasks are defined as specific goals and objectives that an RL agent aims to accomplish within an environment. The description of tasks involves specifying the objectives, constraints, and desired behaviors that an agent needs to learn and optimize. The definition of several parameters plays a crucial role in designing well-defined and achievable RL tasks.

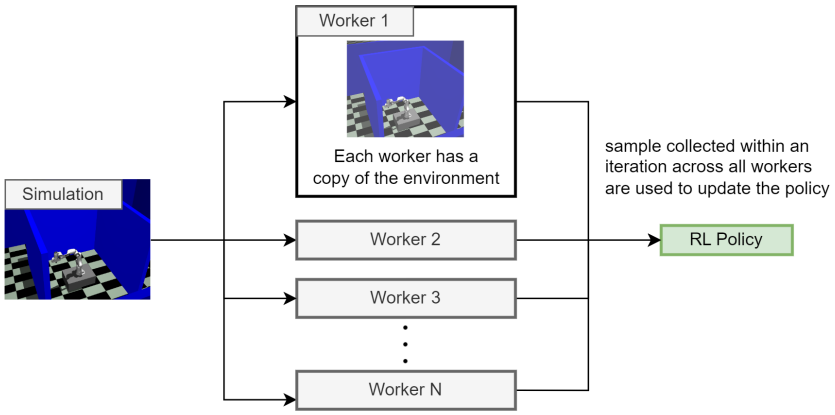


Figure 3: Representation of the RLlib training setup with multiple workers

These parameters include the reward function, state and action spaces, and the criteria for task completion. In addition to these general parameters, there are specific parameters tailored to the problem under consideration, which play a critical role in the behavior and learning progress of RL agents. These parameters include the following, each of which will be discussed in detail:

- **Criteria for Task Completion:** The criteria for task completion include the conditions under which the RL agent is deemed to have successfully achieved its objective or goal. The Gym interface must have some method to determine whether the goal has been achieved. For instance, in the context of the navigation task, the criteria for task completion can be defined in terms of the distance to the goal. When the distance to the goal becomes smaller than a specified *error threshold*, the task is considered complete.
- **Error Threshold:** The selection of the error threshold is a crucial hyperparameter, as it implicitly determines the complexity of the task. Opting for lower error margins increases the complexity of the task. Training an RL agent with much lower error margins requires a higher number of training iterations.
- **Terminal States:** When designing an RL task, it is essential to specify the termination conditions for the episode within the Gym interface. For instance, within the context of a navigation task, the episode must be terminated in the event of a collision or if the agent wanders in the opposite direction of the goal. This ensures that the agent learns from the negative consequences of its actions. Additionally, if the agent gets stuck in a state from which it cannot recover, continuous interactions with the environment can result in unnecessary consumption of computational resources. This, in turn, leads to reduced scalability and efficiency in the RL training process.
- **Environment Timestep:** The environment timestep denotes the interval used within the simulator for time-based dynamics calculations. This timestep must be sufficiently small for precise control and dynamics. After careful consideration, this value was set to be 1 millisecond(ms).
- **Control Frequency**: Within the Gym environment, each timestep denotes the interval within which the agent collects a single training sample. When this interval is extremely small, the agent must process a higher volume of samples while updating its parameters. A straightforward method to circumvent this problem is to repeat the same command for a defined number of timesteps until the subsequent control command is received. The time interval between subsequent commands can be controlled using the *control frequency*. The control frequency determines the relationship between one timestep in the Gym interface (also known as the agent timestep) and one environment timestep, i.e. the time period in environment timesteps during which a control command persists. After trial and error, in this work, a value of 10 has been found useful for the control frequency, i.e. a control command persists for 10 environment timesteps (and 10 training samples) before changing.
- **Episode Length**: The episode length denotes the duration of a single episode in the Gym interface, expressed in terms of the maximum number of agent timesteps in one episode. It has to be carefully selected to ensure that the designated task can, in fact, be completed within the specified timeframe. This was ensured by simulating the robot using a package called *urdf-tutorial*, which allows the teleoperation of the robot using a keyboard or a mouse. This package enables the user to simulate a URDF file and control each individual joint. By simulating a number of sample trajectories, we obtained estimates of the timeframe required to complete each task considered in this thesis.

The essential parameters for the simulator and the Gym interface have also been used for whole-body manipulation in this thesis. The inherent complexity of whole-body manipulation has made it a difficult problem to solve. Using the divide-and-conquer approach, the task has been broken down into more manageable sub-problems involving either navigation or manipulation. By incrementally increasing the difficulty of the task, the RL framework eventually has evolved to the point where RL can control the entire robot. Under this approach,

our focus has shifted to training RL agents for the subsequent tasks:

- **Navigation:** Navigating to a randomly selected goal location, sampled at a specified distance and orientation from the initial robot position.
- **Manipulation:** Moving the end effector to a target position within the operational space of the robot's arm.
- **Whole-body Manipulation:** Moving the end effector to a target position situated outside the operational space of the robot's arm.

We expand upon each task in the following sections.

B. Navigation

In this task, the RL objective is to learn a policy π_{nav} that effectively advances the robot's mobile platform towards a specific goal location. At the beginning of each episode, a random goal is selected, positioned 6 meters away from the robot's initial position $P_{0_{base}}$ and at a randomized orientation. The goal position is specified by a point G_{xy} in the XY-plane. The agent receives observations and samples actions to navigate the robot towards the goal. The maximum duration of an episode is selected to be 12 seconds (i.e. $T_{\text{sim}} = 12000$). In the Gym environment, we specify a control frequency of 10, leading to one action being repeated for 10 simulation timesteps ($T_{\text{ctrl}} = 10$). We specify the maximum episode length to be 1200 agent timesteps ($T_{\text{limit}} = T_{\text{sim}}/T_{\text{ctrl}} = 1200$). After this timeframe, the episode is truncated, and a new goal is sampled for the next episode. In the following, we define the relevant elements of the RL strategy:

- **Observation Space:** In order to navigate to a randomly selected goal, the robot needs information about the goal. This information is provided in the form of a relative distance d_g and an angle θ_g to the goal. Additionally, information about the robot's velocities at the previous timestep is added to the navigation observation o_{nav}. This aids the robot in selecting achievable velocities at the current timestep. Our objective is to find the navigation policy π_{nav} which is a mapping function f to determine the current action a_t given the previously stated inputs:

$$a_t = f(d_g, \theta_g, v_x, v_y, \omega) \tag{6}$$

  Since the policy π_{nav} is parameterized using a NN, we normalize the inputs to have a similar range. Specifically, d_g is normalized by a look ahead distance of 10 meters, representing the maximum permissible distance the robot can stray from the goal. Similarly, θ_g is limited within $[-\pi, \pi]$. Moreover, the velocities are already constrained within the interval $[-1, +1]$ in the form of limits imposed on the mobile platform in simulation.
- **Action Space:** The RL agent needs to select the appropriate action in each state to reach the goal successfully. The action a_t depends on the type of the control. As described in Section III, the mobile platform can be controlled using passive or active wheel control. Consequently, the following three action spaces are defined for this navigation task, based on the type of control used for the mobile platform:
  - **Action Space 1 (AS1):** For passive wheel control, the action space is three-dimensional, consisting of the velocities in the X and Y directions v_x and v_y along with a rotational velocity ω. The action space is continuous and each velocity value is constrained within the range $[-1, +1]$ (m/s). This action space formulation dictates that a valid action consists of three values, each restricted to a magnitude not surpassing 1 m/s.
  - **Action Space 2 (AS2):** For active wheel control, the action space consists of four dimensions. It is continuous with each value constrained within the range $[-60, +60]$ (radians per second). These four actions correspond to individual velocities of the four wheels of the KELO robile platform. It is important to note that controlling individual wheels intensifies the task complexity. The action space is of higher dimensionality. Moreover, the RL agent must learn the corresponding relationship between the wheel velocities and the global base movement. Note that we provide the same velocity to the right and left hub wheels under this definition.
  - **Action Space 3 (AS3):** Alternatively, another strategy for active wheel control is to have an 8-dimensional continuous action space constrained within $[-60, +60]$ (radians per second). These eight actions correspond to individual velocities of the eight hub wheels of the KELO robile platform. This task is significantly more complex than AS1 and AS2, as the RL agent needs to learn the relationship between the movement of 8 wheels and the global movement of the entire platform.
- **Reward Function:** An RL agent tries to maximize the cumulative reward through the learning process. Therefore, the reward functions play a critical role in implicitly determining the optimization objective for the agent. A reward is computed at each agent timestep t indicating whether the current action led to a more favorable state or not. Therefore, after careful analysis and experiments, the reward function in Equation 7 is proposed.

$$R = \begin{cases} R_{goal}, & \text{if } \ d_{g_t} \leq 0.02 \\ R_{wander}, & \text{if } d_{g_t} > d_{LA} \\ G(d_{g_{t-1}}, d_{g_t}), & \text{otherwise} \end{cases} \tag{7}$$

  We want to incentivize the RL agent to reach the goal. Consequently, the agent gets a positive reward when it reaches its goal ($R_{goal} = 20$). A robot is considered to have reached the goal when the distance between the goal and the robot is less than or equal to 0.02 meters ($d_g \leq 0.02$). However, our aim is to facilitate faster learning by providing dense rewards. In addition to rewarding the agent upon reaching the goal, we also want to encourage the agent to move closer to the goal with each passing timestep. Therefore, we use a delta-based reward kernel $G(d_{g_{t-1}}, d_{g_t})$, which is the difference between the distance to the goal at the previous timestep and the one at the current timestep. If the current action resulted in

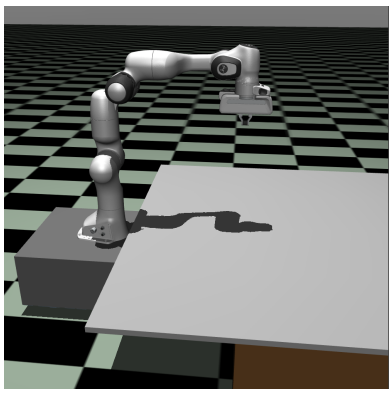


Figure 4: Environment setup for manipulation and whole-body manipulation tasks

the robot moving closer to the destination, the reward is positive. Otherwise, the agent gets a negative reward. However, if the agent wanders in the opposite direction over a look-ahead distance of 10 meters ($d_{LA} = 10$), a penalty of $R_{wander} = -20$ is applied.

- **Episode Reset:** At the beginning of each episode, the agent is placed at an initial position selected at random. The episode terminates if the timestep limit is met or the robot wanders away from the goal (greater than 10 meters).

C. Manipulation

In this task, the RL objective is to learn a policy π_{man} that effectively manipulates the joints of the FR3 arm to move the end effector to a target position. The environment for the manipulation task consists of a table placed near the robot, as shown in Figure 4. The specified target position is located slightly above the tabletop. In addition to reaching the target position, the RL agent must also ensure that the end effector does not hit the table. The target position is within the operational range of the arm, eliminating the need to move the mobile base. That is to say, we exclusively focus on manipulation and provide zero velocities to the mobile base. The target position is sampled at random between a radius of 0.58 to 0.68 meters from the base of the arm, as shown in Figure 6. Note that the maximum range of the arm is 0.855 meters. We only consider an area in front of the robot. Therefore, the target position is sampled with an orientation of $[-1.2, +1.2]$ (radians) with respect to the robot's base frame. After simulating the robot using the *urdf-tutorial* package, it was found that a duration of 7 seconds is sufficient to reach the target position within the specified area (i.e. $T_{\text{sim}} = 7000$). Similar to navigation, we use a control frequency of 10. This implies that a single manipulation action is repeated for 10 simulation timesteps ($T_{\text{ctrl}} = 10$). As a result, the time limit in the Gym interface is specified to be 700 ($T_{\text{limit}} = 700$). Episodes lasting longer than that time period were truncated.

The RL agent can move the end effector by manipulating the joints of the arm. We chose a target position on a plane with a height of 0.437 meters. This plane location is slightly above the top of the table placed adjacent to the robot. In the following, we define the relevant elements of the RL strategy:

- **Observation Space:** The agent requires information about the arm's joint, the end effector's position, and the target position. Therefore, the manipulation observation o_{man} must include the position of the arm joints Q_{pos}, position of the end effector EE_{pos} and the target position G_{xy}. Consequently, the observation space is represented as follows:

$$o_t = (Q_{pos}, EE_{pos}, G_{xy}) \tag{8}$$

  In Equation 8, $Q_{pos} = [q_1, q_2, q_3, q_4, q_5, q_6, q_7]$ is a 7-dimensional vector comprising the rotational positions of all the arm joints. Each joint position is constrained between 0 and 2π radians. EE_{pos} is the location of the end effector in the 3-dimensional space. Finally, G_{xy} represents the target position in the XY plane. Note that the z-coordinate of the target position has been omitted because we are considering movements along a plane at a fixed height.
  Since we want to move the end effector to a position along a fixed plane, not all degrees of freedom are required. Planar movement is possible by manipulating joints 1, 2 and 4 of the arm. A description of these joints and their function is provided as follows:
  - **Joint 1** is the base joint of the arm and provides rotation around the vertical axis. It allows the arm to rotate horizontally, enabling it to cover a wide workspace. Joint 1 is responsible for the base rotation of the entire arm.
  - **Joint 2** is the joint allowing rotation in the vertical plane. It is responsible for lifting the arm up or down. Joint 2 provides vertical motion, which is orthogonal to the rotation provided by joint 1.
  - **Joint 4** is the elbow joint of the FR3 arm. It provides rotation about the axis formed by the end effector of the previous joints. Joint 4 allows the arm to rotate around its own axis, enabling it to change the orientation of the end effector. This allows the arm to extend and reach locations that are further away.
- **Action Space:** This thesis explores both position and velocity control of the arm. Consequently, the action a_t depends on the specified control mode. The action spaces for the two control modes are discussed in detail as follows:
  - **Velocity Control:** For velocity control, the action consists of velocities for joints 1, 2, and 4. We simplify the problem by considering a discrete action space. The policy π_{man} would provide an integer output which is mapped to the values $[-0.5, 0, 0.5]$ (rad/s) for each joint. A velocity of -0.5 leads to an anti-clockwise rotation along the joint while a velocity of 0.5 leads to a clockwise rotation. Providing a 0 velocity stops the joint from rotating. The size of the discrete action space N can be found using the number of actions J and the possibilities for each action A according to $N = J^A$. Since we

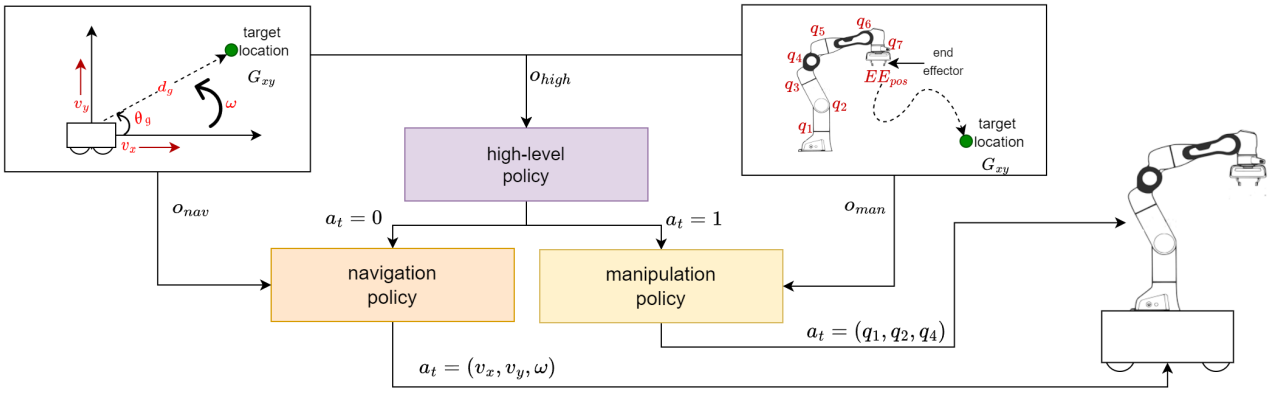


Figure 5: Architecture of the HRL Setup for Whole-body Manipulation

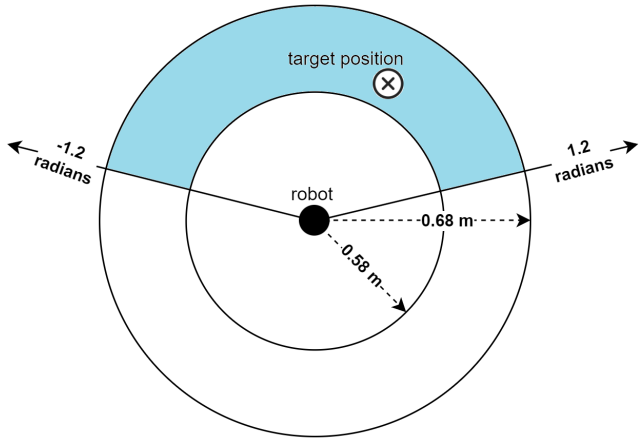


Figure 6: A new target position is sampled from the space indicated in blue at the start of each episode

have 3 joints and 3 possible action values for each, the action space consists of 27 permutations.

– **Position Control:** For position control, the action consists of three displacement values. Similar to velocity, we consider a discrete action space that would be mapped to $[-0.001, 0, 0.001]$. A displacement of -0.001 leads to an anti-clockwise change in the rotational position of the joint while that of 0.001 leads to a change in the clockwise direction. Providing a 0 does not change the position of the joint. Since we have 3 joints and 3 possible action values for each, the action space consists of 27 actions. Since the target position is sampled from a limited space, we also limit the range of motion of joints 1,2, and 4 as shown in Table I to facilitate faster learning.

Table I: Functional Range of FR3 joints

Name	Maximum Range	Selected Range
Joint 1	(-2.8973, 2.8973)	(-1.30, 1.30)
Joint 2	(-1.7628, +1.7628)	(0, 1.60)
joint 3	(-3.0421, -0.1518)	(-2.80, -0.16)

- **Reward Function:** The reward function must encourage the RL agent to move the end effector to the target position. Therefore, the agent gets a positive reward when it achieves its objective ($R_{target} = 100$). The distance between the end effector and target position d_{targ} is monitored at each agent timestep. The task is considered complete when the distance is less or equal to 0.01 meters. This margin of error can be modified depending on the level of accuracy desired. It is important to note that the number of steps required to train the agent exponentially grows as a higher precision is specified. In addition to rewarding the agent upon reaching the goal, we also want to encourage the agent to move closer to the target position with each passing timestep. Therefore, we use a delta-based reward kernel $G(d_{targ_{t-1}}, d_{targ_t})$ which is simply the difference between the distance from the end effector to the target position at the previous timestep and the one at the current timestep. Each episode is 1000 timesteps in the gym interface (10 seconds) which are sufficient to reach the goal. Moreover, we want to penalize the agent if it hits the end effector against the table or its own body. Therefore, for such undesirable scenarios, the agent is penalized with $R_{collision} = -100$. Equation 9 describes all the aforementioned in a formal way:

$$R = \begin{cases} R_{target}, & \text{if } d_{targ} < 0.01 \\ R_{collision}, & \text{if } F_{EE} > 200 \\ G(d_{targ_{t-1}}, d_{targ_t}), & \text{otherwise} \end{cases} \quad (9)$$

- **Episode Reset:** At the beginning of each episode, the arm is reset to an initial position Q_0, as shown in Figure 4. The episode terminates in one of the following terminal states:
  – the end effector is at the target position
  – the number of steps in the episode has exceeded T_{limit}
  – the robot falls over because the arm is being moved beyond the joints' range of motion or the end effector makes contact with a surface (robot body or table)

  For detecting the last scenario, a force sensor is used in

MuJoCo. When the end effector hits an object, the force F_{EE} has a value greater than 200.

D. Whole-body Manipulation

In this task, the RL objective is to learn simultaneous navigation and manipulation. The preceding exploration of navigation and manipulation tasks in Sections IV-B and IV-C has provided insights into the essential components of the problem, including reward functions, termination conditions, action spaces and observation spaces. Merging these insights, we now move on toward whole-body manipulation. RL-based whole-body manipulation can be approached in the following two methods:

- **Flat RL Setup:** This setup uses a single RL policy to control both the arm and the mobile platform.
- **HRL Setup:** This setup breaks the problem into simpler sub-problems, namely navigation and manipulation. The RL agent uses a separate policy for each sub-problem. Finally, a high-level control policy selects which low-level policy should be used in which state.

HRL's ability to deconstruct the complex and intricate task of whole-body manipulation into manageable sub-level objectives that we have already explored greatly simplifies the problem. The agent learns the task in a structured manner, where each level of the hierarchy focuses on different levels of abstraction and complexity. As mentioned previously, we decompose the task into navigation and manipulation and use a high-level policy π_{high} to choose between the two sub-policies (π_{nav} and π_{man}). Each policy operates with its distinct NN, observation space, and action space.

The environment for this task is the same as the one in Section IV-C. However, in addition to sampling the target position, we randomly select the initial pose of the mobile base $P_{0_{base}}$ within 2.5 meters from the edge of the table. The arm always has an initial position of Q_0 at the start of the episode. Within this setup, the target position G_{xy} would be outside the reach of the arm most of the times. Occasionally, we may reset the robot's position to a location close to the table, to ensure diversity in the training scenarios. The maximum duration of an episode is 20 seconds, which is equivalent to 20,000 timesteps in the simulation. In the Gym environment, this equals to an episode limit of 2000 steps ($T_{limit} = 2000$).

In the following, we define the relevant elements of the RL strategy:

- **Observation Space:** We define three observation spaces, one for each policy. We use the same observation space for the navigation policy o_{nav} as described in Section IV-B. The observation space for the manipulation policy o_{man} has the same terms as described in IV-C. However, since the mobile base pose is constantly changing, we transform the end effector and target positions into the reference frame of the mobile base. Therefore, we consider the position $P_{xy} = (p_x, p_y)$ and yaw ϕ of the mobile base using the following transformation matrix:

$$\begin{bmatrix} x_{\text{robot}} \\ y_{\text{robot}} \end{bmatrix} = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x_{\text{world}} - p_x \\ y_{\text{world}} - p_y \end{bmatrix} \tag{10}$$

where $(x_{\text{robot}}, y_{\text{robot}})$ are the coordinates of the point in the reference frame of the robot while $(x_{\text{world}}, y_{\text{world}})$ are the coordinates of the point in the world frame.
Lastly, the observation space for the high-level policy o_{high} consists of the relative distance and angle to the target position, mobile base velocities, target position, and the end effector position as indicated in Equation 11. A key difference is that here we do not transform the target and end effector positions into the robot's frame.

$$o_t = f(d_g, \theta_g, v_x, v_y, \omega, EE_{pos}, G_{xy}) \tag{11}$$

- **Action Space:** The action space of the navigation and manipulation policies are the same as described previously. The high-level policy has a discrete action space consisting of two options: $a_t = 0$ corresponds to navigation while $a_t = 1$ corresponds to manipulation.
- **Episode Reset:** At the beginning of each episode, the arm is set to an initial position Q_0. The mobile base pose $P_{0_{base}}$ is randomly selected. The episode terminates in one of the following terminal states:
  - the end effector is at the target position
  - the number of steps in the episode has exceeded the limit T_{limit}
  - a high force is detected at the end effector $F_{EE} > 200$, indicating collision
  - the robot collides with the table while navigating
- **Reward Function:** In the HRL setup, we can use different methods for learning the policies, with the step function and reward calculation being dependent on the chosen method. For this thesis, we experiment with the following two methods:
  - Training the high-level policy with pre-trained low-level policies
  - Training all three policies from scratch in a multi-agent HRL setup.

  We discuss the two methods in detail in the following sections:

1) HRL with Pre-trained Policies: In this training method, we leverage the manipulation and navigation policies obtained from our prior experiments. The high-level policy π_{high} receives an observation o_{high} from the environment. Based on this observation, the high-level policy can either choose to navigate or manipulate. Depending on the chosen action, the corresponding low-level policy is used to get an action that moves the robot's platform or arm. As a consequence of this action, the end effector's position as well as the state of the environment change. Thus, the agent receives the next observation o'_{high} from the environment.

If the high-level policy chooses to navigate ($a_t = 0$), the platform position might also have changed. In order to accurately assess the impact of the robot's action, we only compute the navigation reward when $a_t = 0$. The robot receives a penalty for hitting the table and for moving far away from the target position. Moreover, it gets a positive reward for moving closer to the target position. This reward function encourages the robot to move to a location such that the target position is within the reach of the arm. Equation 12

next describes all the aforementioned in a formal way:

$$R = \begin{cases} R_{collision}, & \text{if the robot hits the table} \\ R_{wander}, & \text{if } d_{g_t} > d_{limit} \\ G(d_{g_{t-1}}, d_{g_t}), & \text{otherwise} \end{cases} \quad (12)$$

On the other hand, if the high-level policy chooses to manipulate ($a_t = 1$), we also use the reward function of Equation 9. Eventually, the high-level policy learns to use these pre-trained policies to achieve its objective and maximize the reward. The training method is summarized in Algorithm 1.

2) Multi-agent HRL: Alternatively, in this method, we use a MultiAgent environment provided by RLlib where all the policies are trained from scratch. The reward functions for the lower-level policies are the same as described in Equations 9 and 12. Aside from using randomly initialized lower-level policies, the high-level corresponds to multiple agent timesteps in the Gym environment. In other words, the actions taken at a higher level of the hierarchy encompass a series of lower-level actions or interactions with the environment.

At the beginning of each episode, the high-level policy receives an observation and makes a choice between navigation and manipulation. Based on this decision, the selected lower-level policy is executed for either N_{low} steps or until a terminal state is reached. We have selected a value of 20 for N_{low}. The lower-level agent receives a reward for each of these steps. Thus, the low-level agent obtains up to 20 samples to train the policy. On the contrary, only one step of the high-level policy has been completed. The reward for the top-level policy is a sum of the rewards received for these 20 steps. The training method is summarized in Algorithm 2. Notice that we use o_{high}, o_{nav} and o_{man} to represent the observation at the current received from the environment for the three policies. Similarly, we denote the observations at the next timestep using o'_{high}, o'_{nav} and o'_{man}.

E. Policy Parameterization

In this thesis, our goal is to learn a policy to maximize the rewards for each of the specified tasks. In all of the conducted experiments, NNs are the cornerstone for formulating the policy within the RL framework. Consequently, all the policies are parameterized by NNs. Hence, finding an optimal policy is equivalent to learning the optimal parameters for its NN. We use the same architecture for all the policies trained in this thesis. The input layer contains the same number of parameters as the dimensions of the observation space. Similarly, the output layer size is dependent upon the size of the action space. We use two hidden layers of size 64 each.

Table II: NN Parameters Common for all Policies

Hyperparameter	Value
Number of hidden layers	2
Number of hidden units per layer	64
Optimizer	Adam
Batch size	4000
Discount factor	0.99
Number of runs used for plot averages	5
Confidence Interval for plots	95%

V. Experimental Results

In this section, we analyze the experimental results for the tasks described in Section IV. First, we evaluate the training progress and results of the RL agents with respect to the following metrics:

- **Average Return:** Return refers to the cumulative sum of rewards obtained by an RL agent over a single episode through its interactions with the environment. It quantifies the agent's ability to achieve its goals and maximize accumulated rewards over time. A higher average return signifies better performance, indicating that the RL algorithm is making more informed decisions and achieving higher rewards in the given environment. During the training process, the return plateaus, indicating that the policy has converged.
- **Training time (Train Time):** Training time refers to the total amount of time required to train the RL agent. This metric is dependent upon the resources employed for training and the type of algorithm used. To ensure a consistent and fair comparison of different RL algorithms, we maintain the same hardware and software settings for training (one 8-core processor and one GPU).
- **Environment Steps (Env Steps):** Environment steps refer to the total number of interactions between an RL agent and the environment during the training process. Lower environment steps indicate that the algorithm learns effectively with fewer interactions, which is desirable in scenarios where obtaining real-world samples is costly or time-intensive.

The training progress of the RL agents has been visualized in Figure 7. In order to obtain reliable and statistically significant results, we conducted a series of experiments with our RL agents. Each experiment was repeated five times, with metrics recorded for each run. Additionally, we calculated the confidence intervals for each metric to quantify the uncertainty and variance in the results. Confidence intervals at a 95% confidence level are reported, providing a range within which the true population values are likely to lie. The shaded regions represent this confidence interval.

A. Results for the Navigation Task

The results for passive wheel control have been visualized in Figure 7a. During initial experiments with PPO for passive control, no activation function (linear) was used in the last layer of the NN (PPO-linear). This resulted in sub-optimal performance while testing the trained model. Since the range of velocities for AS1 is $[-1, 1]$, the hyperbolic tangent *tanh* was used as the activation function in the last layer with PPO (PPO-tanh). This improved the performance. In addition, we experimented with SAC as well and used tanh activation in the output layer (SAC-tanh). The results indicate that PPO has better performance as compared to SAC, with the same policy architecture. Moreover, during multiple experiments, PPO gave similar results (less variance) as compared to SAC. Lastly, PPO is a simple algorithm to train. Therefore, it was much faster to train as compared to SAC. For active wheel control, we tried PPO with and without tanh for both, AS1 and

Algorithm 1 HRL with Pre-trained Policies

Inputs: Pre-trained policies π_{nav} and π_{man}
Hyperparameters: $N_{\text{iter}}, N_{\text{workers}}, T_{\text{ctrl}}, T_{\text{limit}}$
Init: GymEnv for each worker, function estimator for π_{high} and buffer for storing batch samples $\mathbb{B}_{\text{high}}$
for $i = 1, 2, ..., N_{\text{iter}}$ **do**
  for $w = 1, 2, ..., N_{\text{workers}}$ **do**
    Sample and set $P_{0_{\text{base}}}$. Set arm pose to Q_0
    Sample and set target position G_{xy}
    Set `done = False` and $n_{ep} = 0$
    while not `done` **do**
      Get observation for high-level agent o_{high}
      Sample $a_{\text{high}} \sim \pi_{\text{high}}(\cdot|o_{\text{high}})$
      Increment n_{ep} by 1.
      if $a_{\text{high}} = 0$ **then**
        Get navigation observation o_{nav}
        Sample $a_{\text{nav}} \sim \pi_{\text{nav}}(\cdot|o_{\text{nav}})$ and perform a_{nav} for T_{ctrl} simulation timesteps. Get reward R.
        if $n_{ep} \geq T_{\text{limit}}$ or collision **then**
          Set `done = True`
        end if
      else
        Get manipulation observation o_{man}
        Sample $a_{\text{man}} \sim \pi_{\text{man}}(\cdot|o_{\text{man}})$ and perform a_{man}. Get reward R.
        if n_{ep}limit or $F_{\text{EE}} > 200$ **then**
          `done = True`
        end if
      end if
      Store $(o_{\text{high}}, a_{\text{high}}, R_{\text{high}}, o'_{\text{high}})$ in $\mathbb{B}_{\text{high}}$
    end while
  end for
  Update π_{high} using $\mathbb{B}_{\text{high}}$. Discard samples in $\mathbb{B}_{\text{high}}$ collected before the policy update
end for
Outputs: Trained policies π_{nav} and π_{man}

Table III: NN Architecture and Training Details

Task	Size of Input Layer	Input parameters	Size of Output Layer	Output parameters
Navigation	5	$d_g, \theta_g, v_x, v_y, \omega$	**AS1**: 3 **AS2**: 4 **AS3**: 8	**AS1**: Platform velocities v_x, v_y, ω **AS2**: Individual velocities for caster wheels **AS3**: Individual velocities for hub wheels
Manipulation	13	$Q_{pos}, EE_{pos}, G_{xy}$	27	**Velocity Control**: joint velocities $\dot{q_1}, \dot{q_2}, \dot{q_4}$ **Position Control**: joint displacement $\delta q_1, \delta q_2, \delta q_4$
High-Level	10	$d_g, \theta_g, v_x, v_y, \omega, EE_{pos}, G_{xy}$	2	**0** (navigation) or **1** (manipulation)

AS2. The results have been visualized in Figure 7b. Similar to the previous case, linear activation resulted in poor performance for both action spaces (PPO-4wheel_linear and PPO-8wheel_linear). When we use the hyperbolic tangent (tanh) activation function, we get a comparatively higher reward (PPO-4wheel_tanh and PPO-8wheel_tanh). Upon visualization in MuJoCo with a random goal, it was observed that the robot was pushed in the direction to the goal. However, the policy did not manage to learn how to reach the goal. These insights are summarized in Table IV.

The two best performing models for the navigation task, PPO-tanh and SAC-tanh, have been tested for different goals. We provided different goals to the RL agents, situated at a distance of 5 meters from the robot. The robot started at the same initial pose, i.e. (0,0) with zero yaw. We have performed 10 runs for each goal and recorded the following metrics:

- **Peak Deviation:** Maximum deviation from the shortest path to the goal in meters.

Table IV: RL Training Results for Navigation Task (in the table, M stands for 10^6)

Exp	Algo.	Control	Average return	Train time (hours)	Env steps
1	PPO-tanh	passive	25.9±0.0	2.3±0.2	12M
2	SAC-tanh	passive	21.9±14.1	10.6±0.5	20M
3	PPO-linear	passive	6.5± 9.1	2.5±0.2	300M
4	PPO-tanh	active-4	12.1 ± 30.1	15.1±0.2	300M
5	PPO-tanh	active-8	14.8 ± 15.1	17.1±0.3	300M
6	PPO-linear	active-4	-1.9 ± 0.4	15±0.1	300M
7	PPO-linear	active-8	-1.9 ± 0.2	17±0.2	300M

Algorithm 2 MultiAgent RL Training Setup

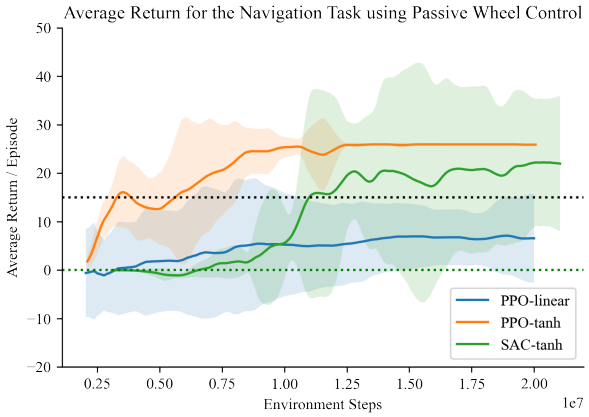
Hyperparameters: $N_{\text{iter}}, N_{\text{workers}}, T_{\text{ctrl}}, T_{\text{limit}}, N_{\text{low}}$
Init: MultiAgentEnv for each worker, function estimators for $\pi_{\text{high}}, \pi_{\text{nav}}$ and π_{man}, buffers for storing batch samples $\mathbb{B}_{\text{high}}, \mathbb{B}_{\text{nav}}, \mathbb{B}_{\text{man}}$
for $i = 1, 2, ..., N_{\text{iter}}$ **do**
  for $w = 1, 2, ..., N_{\text{workers}}$ **do**
    Sample and set $P_{0_{\text{base}}}$. Set arm pose to Q_0
    Sample and set target position G_{xy}
    Set `done = False` and $n_{ep} = 0$
    while not `done` **do**
      Get observation for high-level agent o_{high}
      Sample $a_{\text{high}} \sim \pi_{\text{high}}(\cdot|o_{\text{high}})$
      Set $n_{\text{low}} = 0$ and $R_{\text{high}} = 0$
      if $a_{\text{high}} = 0$ **then**
        while not `done` and $n_{\text{low}} < N_{\text{low}} - 1$ **do**
          Get navigation observation o_{nav}
          Sample $a_{\text{nav}} \sim \pi_{\text{nav}}(\cdot|o_{\text{nav}})$ and perform a_{nav} for T_{ctrl} simulation timesteps. Get reward R_{nav}.
          Store $(o_{\text{nav}}, a_{\text{nav}}, R_{\text{nav}}, o'_{\text{nav}})$ in $\mathbb{B}_{\text{nav}}$
          Increment R_{high} by $0.3 \times R_{\text{nav}}$, n_{low} by 1 and n_{ep} by 1.
          if $n_{ep} \geq T_{\text{limit}}$ or collision **then**
            Set `done = True` and store $(o_{\text{high}}, a_{\text{high}}, R_{\text{high}}, o'_{\text{high}})$ in $\mathbb{B}_{\text{high}}$
          end if
        end while
      else
        while not `done` and $n_{\text{low}} < N_{\text{low}} - 1$ **do**
          Get manipulation observation o_{man}
          Sample $a_{\text{man}} \sim \pi_{\text{man}}(\cdot|o_{\text{man}})$ and perform a_{man}. Get reward R_{man}.
          Store $(o_{\text{man}}, a_{\text{man}}, R_{\text{man}}, o'_{\text{man}})$ in $\mathbb{B}_{\text{man}}$
          Increment R_{high} by $0.7 \times R_{\text{man}}$, n_{low} by 1 and n_{ep} by 1.
          if $n_{ep} \geq T_{\text{limit}}$ or $F_{\text{EE}} > 200$ **then**
            `done = True`
            Store $(o_{\text{high}}, a_{\text{high}}, R_{\text{high}}, o'_{\text{high}})$ in $\mathbb{B}_{\text{high}}$
          end if
        end while
      end if
    end while
  end for
  Update $\pi_{\text{high}}, \pi_{\text{nav}}$ and π_{man} using $\mathbb{B}_{\text{high}}, \mathbb{B}_{\text{nav}}$ and $\mathbb{B}_{\text{man}}$
  Discard samples in $\mathbb{B}_{\text{high}}, \mathbb{B}_{\text{nav}}$ and $\mathbb{B}_{\text{man}}$ collected before updating the policy
end for
Outputs: Trained policies π_{nav} and π_{man}

- **Time:** Time taken to reach the goal in seconds.
- **Trajectory Length:** Length of the trajectory from the initial position to the goal in meters.

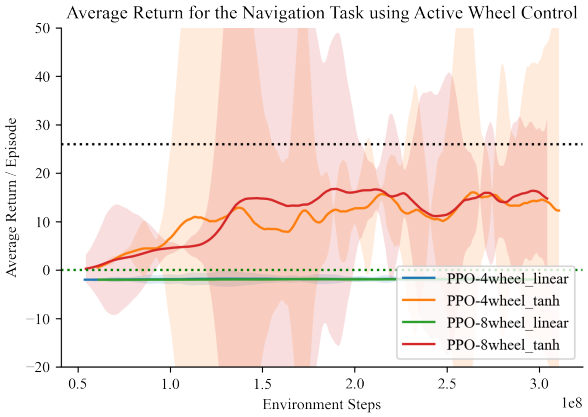
The mean and standard deviation of these metrics across the 10 trials are presented in Table V. Moreover, we visualize the trajectories as a qualitative measure of performance in Figure 8. We can see that the friction in the environment causes the robot to skid from a straight line path. However, both RL agents have learnt to apply corrective commands to overcome errors caused by friction. Except for a goal that is perpendicular to the initial orientation of the robot, PPO exhibits a lower deviation from the straight-line path. It is important to note that the straight line behaviour was not explicitly encouraged in the reward function. The RL agents implicitly figured out that the shorter path they take, the faster they would accomplish the objective.

Table V: Evaluation Results for the Navigation Task

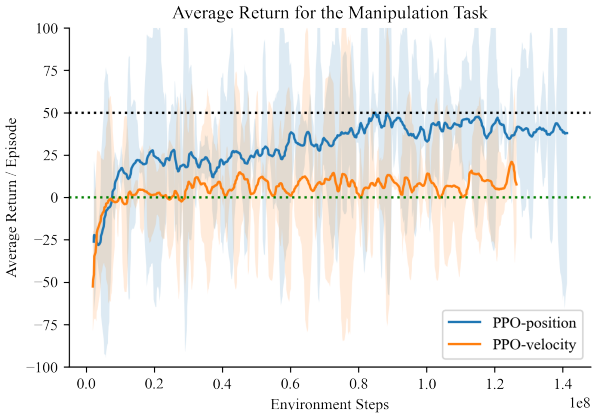
Algo.	Target Pos.	Peak Deviation	Time (sec)	Trajectory length (m)
PPO	(5,0)	0.42±0.02	4.89±0.16	5.46±0.01
SAC	(5,0)	1.02±0.01	5.60±0.01	5.96±0.01
PPO	(0,5)	0.80±0.01	5-24±0.02	5.62±0.03
SAC	(0,5)	0.46±0.01	4.56±0.01	5.52±0.01
PPO	(4.2,4.2)	0.85±0.01	5.23±0.02	5.82±0.01
SAC	(4.2,4.2)	1.23±0.1	5.23±0.01	5.87±0.01
PPO	(-4.2,4.2)	0.78±0.04	5.28±0.10	6.79±0.01
SAC	(-4.2,4.2)	1.15±0.0	5.45±0.01	5.13±0.01



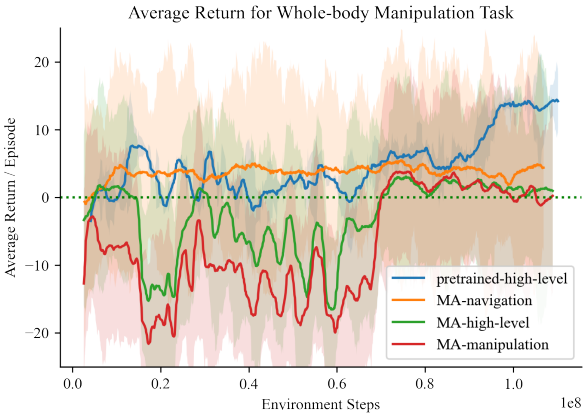
(a) Training results for passive-wheel navigation



(b) Training results for active-wheel navigation



(c) Training results for manipulation



(d) Training results for whole-body manipulation

Figure 7: Comparison of performance of different RL agents for the three tasks considered in this thesis

B. Results for the Manipulation Task

The training results for manipulation are visualized in Figure 7c. Upon analysis, we notice that position control yields a higher reward than velocity control. These training results are also summarized in Table VI.

Table VI: RL Training Results for the Manipulation Task (in the table, M stands for 10^6)

Exp	Control	Average return	Train time (hours)	Env steps
1	Position	37.9±20.2	20.2±0.02	126M
2	Velocity	7.6±14.1	17.3 ± 0.01	132M

For the testing of the manipulation agents, the robot starts at an initial location of (0,0). The arm is always at the same starting position Q_0, similarly to training. Then, we select goals at random at a height of 0.437 meters. We perform 20 trials for each type of control, with each trial running for a maximum of 10 seconds. We report the following metrics in Table VII:

- The **success rate**, as the percentage of trials in which the RL agent successfully moves the end effector within a 0.035-meter distance to the target location in 10 seconds.
- The **collision rate**, as the percentage of trials in which the end effector collides with the table.
- The **average error**, as the average distance between the end effector and the target location at the end of each trial.

Table VII: Evaluation Results for the Manipulation Task

Exp	Control	Success rate	Collision rate	Avg. Error (m)
1	Position	0.9	0.0	0.029 ± 0.01
2	Velocity	0.7	0.0	0.033 ± 0.01

We can notice that RL-based position control is more precise and has a higher success rate as compared to velocity-based control. Both agents have learned not to collide with the table. Sample trajectories of the two agents as well as the variations in joint position are visualized in Figure 9. Notice that RL-based position control is faster and more accurate. However, velocity-based control is smoother.

C. Results for the Whole-body Manipulation Task

For whole-body manipulation, we tried two training methods. The training results are summarized in Figure 7d. When

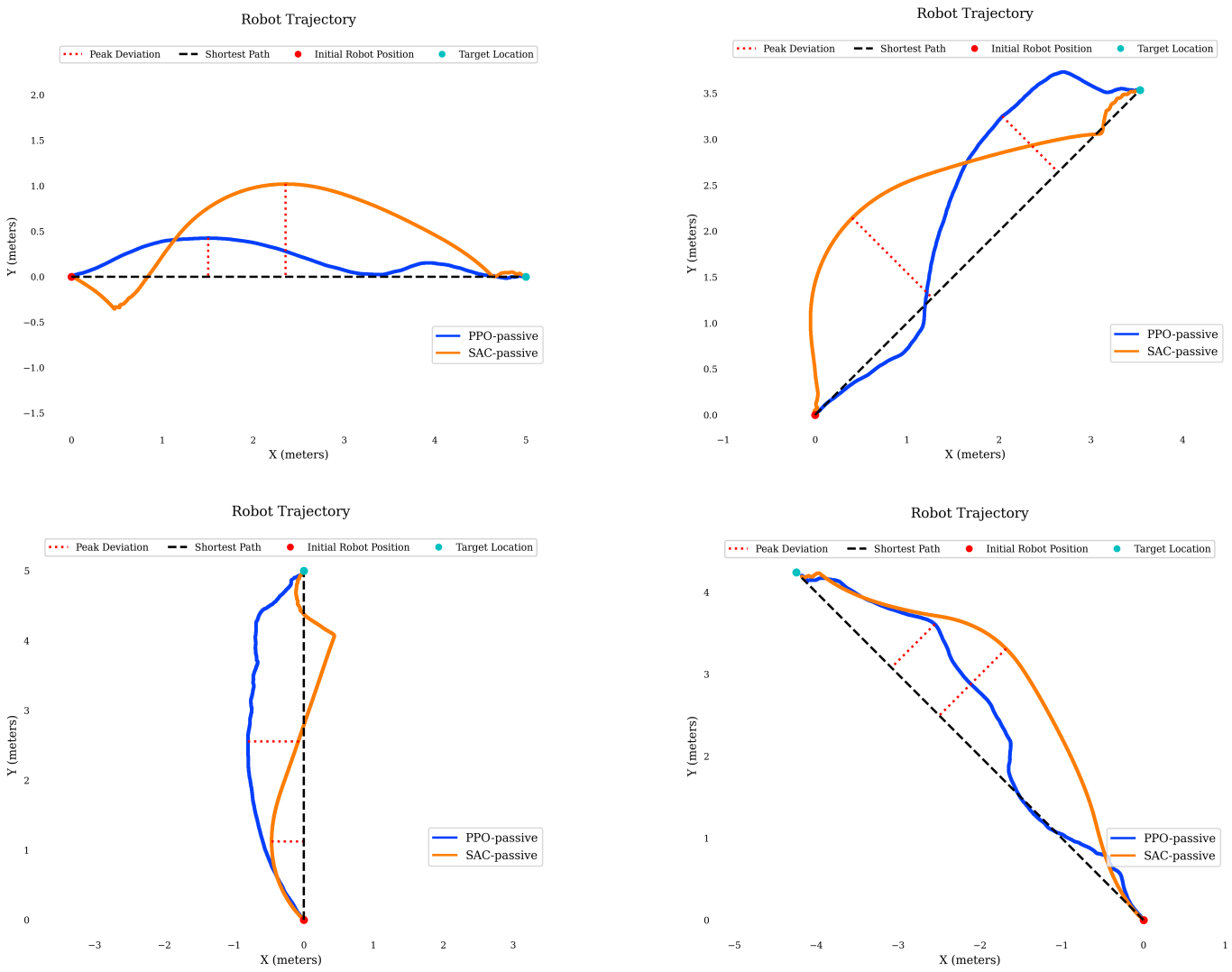


Figure 8: Trajectory followed by the PPO and SAC agents to reach goals at different angles.

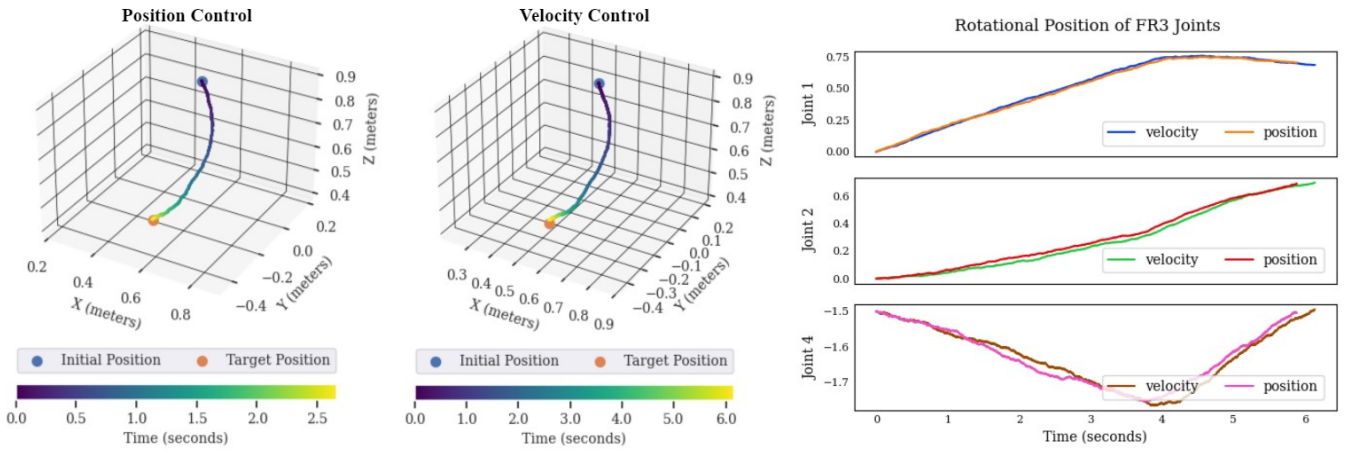


Figure 9: Performance Analysis of RL-based position and velocity control for the manipulation task

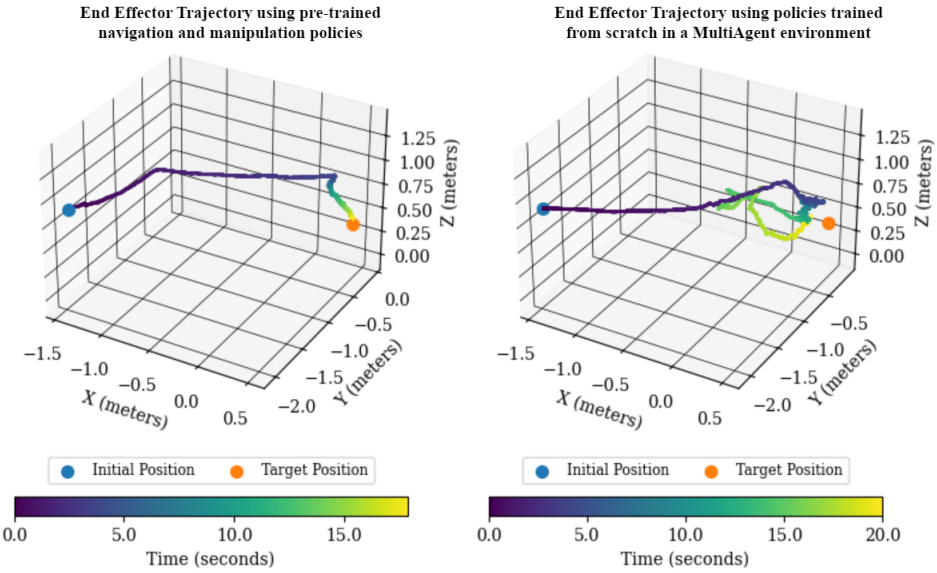


Figure 10: Analysis of the performance of HRL agents for whole-body manipulation

training the top-level policy with pre-trained π_{nav} and π_{man}, we only have one curve for average return (represented in blue color). In the multi-agent scenarios, we train the π_{high}, π_{nav} and π_{man} from scratch. The progress is indicated by the green, orange, and red curves respectively. We get better results with pre-trained policies. Notice that we are using position control due to its higher precision.

Table VIII: Training Results for the HRL agents (in the table, M stands for 10^6)

Exp	Policy	Average return	Train time (hours)	Env steps
1	High-level (pre-trained)	14.3±2.9	30.1±0.1	95M
	High-level (multi-agent)	0.07±14.9	34.8±0.0	110M
2	Manipulation	4.34±7.2	34.8±0.0	110M
	Navigation	34.8±0.0	14.4 ± 0.01	110M

To evaluate the policies trained using the two methods, we use randomly sampled goals. The robot starts 2.5 meters away from the target position. The evaluation episode is allowed to run for a maximum of 20 seconds. The episode terminates with the successful completion of the task (when the end effector is at the target position), when the robot collides with the table or when the time runs out. The end effector trajectories are visualized in Figure 10. The high-level policy trained with pre-trained low-level policies provides better results. The robot successfully moves the end effector to the target position. However, when all the policies are trained from scratch, the robot learns to move closer to the table, but the manipulation policy is not learnt accurately. These results are summarized in Table IX.

Table IX: Evaluation Results for the HRL agents

Exp	Policy Type	Trajectory Length	Avg. Error (m)
1	HRL-pretrained	2.2 ± 0.01	0.035 ± 0.01
2	HRL-multiagent	4.4± 0.05	0.17 ± 0.01

VI. Conclusions and Future Work

In this thesis, we present an HRL-based approach to tackle the complex task of whole-body manipulation. Leveraging the complementary nature of precise manipulation and expansive navigation capabilities of the KELO mobile base mounted with an FR3 arm, we investigate the potential of RL algorithms for autonomous navigation and manipulation. We experiment with different algorithms and configurations for these two tasks. By defining well-structured and simple reward functions, termination conditions, observation spaces and action spaces, we train an RL agent which can successfully perform these tasks separately. With a foundational understanding of these tasks, we define a hierarchical structure where a high-level policy acts as a controller above the navigation and manipulation policies. Our results demonstrate that HRL can seamlessly orchestrate the decision-making processes required for whole-body manipulation, effectively addressing complex scenarios beyond the scope of individual tasks. We have shown that amongst several different training methods for HRL, using pre-trained policies yields better results.

This thesis provides a groundwork for RL-based whole-body manipulation while opening numerous avenues for future research. One crucial direction is to enhance the adaptability and robustness of the RL agents through real-world testing. It would also be interesting to explore more complex tasks such as pick-and-place using HRL. Multiple levels of hierarchy can also be explored for more complicated whole-body manipulation tasks.

References

[1] Kelo robotics. Available online: https://www.kelo-robotics.com/products/, Last accessed: Sep 21, 2023.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[4] W. Chung, C. Rhee, Y. Shim, H. Lee, and S. Park. Door-opening control of a service robot using the multifingered robot hand. *IEEE Transactions on Industrial Electronics*, 56(10):3975–3984, 2009.

[5] L.-P. Ellekilde and H. I. Christensen. Control of mobile manipulator using the dynamical systems approach. In *IEEE International Conference on Robotics and Automation*, pages 1370–1376, 2009.

[6] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *IEEE International Conference on Robotics and Automation*, pages 5113–5120, 2018.

[7] A. Francis, A. Faust, H.-T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T.-W. E. Lee. Long-range indoor navigation with prm-rl. *IEEE Transactions on Robotics*, 36(4):1115–1134, 2020.

[8] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[9] S. Haddadin, S. Parusel, L. Johannsmeier, S. Golz, S. Gabl, F. Walch, M. Sabaghian, C. Jähne, L. Hausperger, and S. Haddadin. The franka emika robot: A reference platform for robotics research and education. *IEEE Robotics Automation Magazine*, 29(2):46–64, 2022.

[10] M. Hutsebaut-Buysse, K. Mets, and S. Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022.

[11] M. Hvilshøj, S. Bøgh, O. Madsen, and M. Kristiansen. The mobile robot "little helper": Concepts, ideas and working principles. In *IEEE Conference on Emerging Technologies Factory Automation*, pages 1–4, 2009.

[12] S.-H. Hyon, Y. Ida, J. Ishikawa, and M. Hiraoka. Whole-body locomotion and posture control on a torque-controlled hydraulic rover. *IEEE Robotics and Automation Letters*, 4(4):4587–4594, 2019.

[13] J. Jiao, Z. Cao, N. Gu, S. Nahavandi, Y. Yang, and M. Tan. Transportation by multiple mobile manipulators in unknown environments with obstacles. *IEEE Systems Journal*, 11(4):2894–2904, 2015.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] V. Klemm, A. Morra, L. Gulich, D. Mannhart, D. Rohr, M. Kamel, Y. de Viragh, and R. Siegwart. LQR-assisted whole-body control of a wheeled bipedal robot with kinematic loops. *IEEE Robotics and Automation Letters*, 5(2):3745–3752, 2020.

[16] D. Leidner, A. Dietrich, M. Beetz, and A. Albu-Schäffer. Knowledge-enabled parameterization of whole-body control strategies for compliant service robots. *Autonomous Robots*, 40:519–536, 2016.

[17] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International conference on machine learning*, pages 3053–3062, 2018.

[18] M. V. Minniti, F. Farshidian, R. Grandia, and M. Hutter. Whole-body MPC for a dynamically stable mobile manipulator. *IEEE Robotics and Automation Letters*, 4(4):3687–3694, 2019.

[19] K. Nagasaka, Y. Kawanami, S. Shimizu, T. Kito, T. Tsuboi, A. Miyamoto, T. Fukushima, and H. Shimomura. Whole-body cooperative force control for a two-armed and two-wheeled mobile robot using generalized inverse dynamics and idealized joint units. In *IEEE International Conference on Robotics and Automation*, pages 3377–3383, 2010.

[20] K. Nakhleh, M. Raza, M. Tang, M. Andrews, R. Boney, I. Hadžić, J. Lee, A. Mohajeri, and K. Palyutina. SACPlanner: Real-world collision avoidance with a soft actor critic local planner and polar state representations. In *IEEE International Conference on Robotics and Automation*, pages 9464–9470, 2023.

[21] X. Ren, Y. Liu, Y. Hu, and Z. Li. Integrated task sensing and whole body control for mobile manipulation with series elastic actuators. *IEEE Transactions on Automation Science and Engineering*, 20(1):413–424, 2022.

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

[23] H. Shi, L. Shi, M. Xu, and K.-S. Hwang. End-to-end navigation strategy with deep reinforcement learning for mobile robots. *IEEE Transactions on Industrial Informatics*, 16(4):2393–2402, 2019.

[24] D. H. Shin, B. S. Hamner, S. Singh, and M. Hwangbo. Motion planning for a mobile manipulator with imprecise locomotion. In *International Conference on Intelligent Robots and Systems*, volume 1, pages 847–853, 2003.

[25] C. Sun, J. Orbik, C. M. Devin, B. H. Yang, A. Gupta, G. Berseth, and S. Levine. Fully autonomous real-world reinforcement learning with applications to mobile manipulation. In *Conference on Robot Learning*, pages 308–319, 2022.

[26] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. A Bradford book. MIT Press, 1998.

[27] Technalia. Mobile robot manipulation use case analysis. Available online: https://ec.europa.eu/research/participants/documents/, Last accessed: Sep 21, 2023.

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

[29] C. Wang, Q. Zhang, Q. Tian, S. Li, X. Wang, D. Lane, Y. Petillot, and S. Wang. Learning mobile manipulation through deep reinforcement learning. *Sensors*, 20(3), 2020.

[30] T. Welschehold, C. Dornhege, and W. Burgard. Learning mobile manipulation actions from human demonstrations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3196–3201, 2017.

[31] W. Yuan, Y.-H. Liu, C.-Y. Su, and F. Zhao. Whole-body control of an autonomous mobile manipulator using model predictive control and adaptive fuzzy technique. *IEEE Transactions on Fuzzy Systems*, 31(3):799–809, 2023.