The Thesis Committee for Steve Han
certifies that this is the approved version of the following thesis:

VR Teleoperation Interface for Learning Loco-manipulation of Humanoid Robots

APPROVED BY

SUPERVISING COMMITTEE:

Yuke Zhu, Supervisor

Etienne Vouga

VR Teleoperation Interface for Learning Loco-manipulation of Humanoid Robots

by

Steve Han

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2023

Acknowledgments

I would like to thank Mingyo Seo and Yuke Zhu for the project idea as well as the guidance they provided along the way. Many of the motivations introduced in Chapter 1 are based on their ideas.

This work is conducted in partnership with Mingyo Seo, who created the simulation, trained the baseline policies, and adapted the whole body control code. All the results in Chapter 5 besides the data collection are obtained by him.

Chapter 4 would not have been possible without the hardware team (Carlos Isaac Gonzalez and Seung Hyeon Bang), who shared their humanoid setup for us to conduct real-robot experiments and helped us when things went awry.

Finally, I would like to give thanks to Kyutae Sim for helping with the data collection and coding.

VR Teleoperation Interface for Learning Loco-manipulation of Humanoid Robots

Steve Han, MSCompSci
The University of Texas at Austin, 2023

Supervisor: Yuke Zhu

Our world is designed by humans, for humans. This makes humanoid robots the most suitable general-purpose platform to automate repetitive or dangerous tasks done by people. However, due to the complex dynamics and high degrees-of-freedom of humanoid robots as well as the shortage of demonstration data, research in robot learning for humanoids is scarce. To address these challenges, I present a VR interface named TRILL (TeleopeRation Interface for Learning Loco-manipulation) to collect human demonstrations for humanoid robots in both simulation and reality. The demonstrations are then used to train a baseline Imitation Learning algorithm that uses an underlying controller to abstract away the complexity of whole-body control. I further propose that by embedding this data collection mechanism in VR video games, we can amass a large-scale dataset of high quality human demonstrations that can drive the development of future autonomous humanoids. To illustrate the feasibility of this idea, we collect a small dataset on toy tasks in simulation

and real robot using the VR interface. We then show that the trained policy can be deployed in simulation with a reasonable success rate. A video demo of the VR teleoperation can be found here: `https://youtu.be/PNZTwtcRhVU`.

Table of Contents

Acknowledgments **iii**

Abstract **iv**

List of Tables **viii**

List of Figures **ix**

Chapter 1. Introduction **1**
1.1 Humanoid Robots 5
1.2 VR Teleoperation Interface 6
1.3 Scaling up Demonstration Dataset for Humanoids 7

Chapter 2. Background **10**
2.1 Imitation Learning 10
2.2 Whole-body Control (WBC) 11
2.3 Related Work 12
2.3.1 VR for Imitation Learning 12
2.3.2 Demonstration Interface 13
2.3.3 Learning for Humanoids 15
2.3.4 Humanoid Teleoperation 15

Chapter 3. VR Interface **18**
3.1 Architecture 19
3.2 Technical Details 21
3.2.1 OpenVR 21
3.2.2 Image Transfer from GPU to CPU 22
3.2.3 Asynchronous Message Passing 23
3.2.4 Separate Computer for VR Interface 24
3.3 Performance Analysis 24

Chapter 4. Real-Robot Experiments **28**
4.1 Infrastructure 28
4.1.1 Image Streaming From Camera to VR 30
4.2 Neural Network Training and Evaluation 31

Chapter 5. Simulation Experiments **35**
5.1 Tasks 35
5.2 Results 36

Chapter 6. Conclusion **39**

Chapter 7. Future Work **41**

Bibliography **44**

List of Tables

3.1 Simulation performance numbers. The left column denotes whether OpenCV or the VR headset is used to display the image and the number of PGS iterations per time step. The last row shows a manipulation task without locomotion. In each loop, Mujoco runs 25 times, WBC runs 5 times, and rendering runs once. The first three columns measure the time taken in each run, the fourth column shows the total time taken by each loop, while the last three columns show their percent contribution to total loop time. 27

5.1 The evaluation success rates on the 4 simulation tasks. 37

List of Figures

1.1 The proposed interface can be used to teleoperate a simulation environment (left) and a real robot (right). 2

1.2 An overview of the hierarchical learning framework used in this work. Diagram credit to Mingyo. First, we get the desired hand trajectories and walking commands from the demonstrator. Second, we send these commands to the whole-body controller to produce joint-torque actions of the robot. Third, a behavior cloning policy is trained to imitate the human demonstrations. 4

3.1 Architecture for the VR interface 19

3.2 The robot hands takes time to reach the desired controller poses (represented by the floating red and blue arrows). 25

4.1 The infrastructure used to collect demonstraions and deploy policy on the real robot. Original diagram by Mingyo. 29

4.2 150 demonstrations of a simple pick-and-place task are collected. The egocentric view for 36 of the episodes are shown. 31

4.3 During evaluation, the robot usually knocks over the temperature gun before grasping it. 32

4.4 Plotting the desired (red) vs actual (blue) position and orientation of the right hand during deployment. For each grid, the top row is position and the bottom row is velocity. 33

5.1 A successful episode from each simulation task. From top to bottom, we have opening the door, pushing the door, moving the pot, and placing the lid. The 2×5 image sequences are read from left to right and top to bottom. Note that the robot swings laterally when performing locomotion. 38

Chapter 1

Introduction

Humanoid Robotics, Imitation Learning, and Virtual Reality (VR) are some of the most exciting and rapidly developing fields in technology today. Humanoid robots developed by Agility Robotics can be used for last-mile delivery. Imitation Learning shows great promise in robotics and autonomous driving. VR is ushering in a new world called the "metaverse". While each technology is important in its own right, we believe that the marriage of these three technologies can also have far-reaching impacts. Using VR, humans can control robots in an immersive simulation, or they can remotely teleoperate robots to perform tasks in dangerous areas such as a nuclear plant. The trajectories from the human demonstrations can then be used to train Imitation Learning policies, which can enable robots to perform the same tasks autonomously. While deep Reinforcement Learning methods have been successful in teaching robot to learn from repeated interactions with an environment given a reward function, the sample inefficiency, need for online interactions, and intricacy of reward engineering make them difficult to implement in a humanoid platform. On the other hand, imitation learning skips over the requirement to create reward functions to shape a behavior and instead lets the robot learn directly from offline demonstrations.

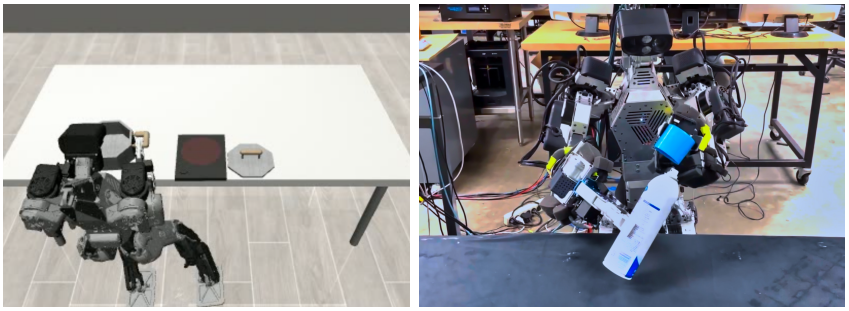

Figure 1.1: The proposed interface can be used to teleoperate a simulation environment (left) and a real robot (right).

However, due to the complexity of humanoid robots, the lack of large-scale data for training, and the difficulty of creating an intuitive interface for data collection, there has been no research on teaching humanoid robots to perform locomotion and bi-manipulation tasks to our best knowledge. There has been work to teach fixed-base robot arms through VR demonstrations [32], but humanoid robots are significantly more difficult to train due to the need to consider robot dynamics for balancing and walking. Also, although there is a large body of work on teleoperating humanoids using VR and motion-capture devices, there has been no attempt in literature to use the teleoperation data to train a neural network policy to control the humanoid autonomously. Furthermore, these "telepresence" systems are usually very complicated and require expensive hardware, which makes them unsuitable for scaling up data collection in a distributed manner.

In this thesis, I present a simple VR interface that uses the commonplace Oculus Quest 2 headset. I believe that compared to other humanoid teleoperation systems, the software architecture used in this project is uniquely

positioned to be both adaptable to VR games and accessible to the public. Although the interface is relatively simple, it is sufficient to teach a humanoid robot to perform some simple tasks. In simulation, I hypothesize that this setup can be incorporated in VR video games to massively scale up data collection. VR games contain rich interactions with the virtual world, have built-in rewards to label successes and failures, and integrate with powerful physical simulation engines. As a result, they are perfect for collecting human demonstrations to potentially teach humanoids to perform the same tasks in the real world. For example, Cook-Out is a VR game that requires players to cook in a kitchen, which is a valuable skill for humanoids to learn. In addition, on the real-robot side, we could potentially distribute the data collection by letting users with VR headsets control the robot remotely. This is similar to the RoboTurk crowd-sourcing platform for fixed-base robot arms [20]. I hope that by open-sourcing the teleoperation codebase later this summer, research in humanoid robot learning can be made more accessible.

To show that we can use the collected demonstrations to train a humanoid to perform simple tasks, we present a hierarchical approach that learns a motion policy from teleoperation demonstrations with an underlying controller. The policy outputs the desired poses of the hands, while the whole-body controller takes care of balancing the robot and tracking the desired trajectory. An overview of this design is presented in Figure 1.2, and a description is given below:

1. First, we take advantage of the similar kinematic structures between

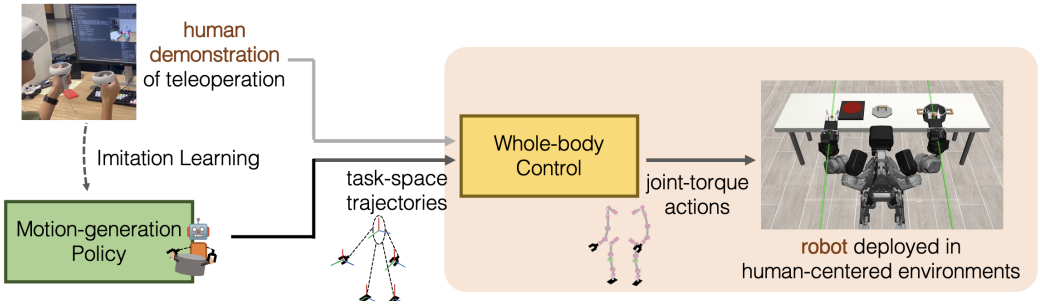

Figure 1.2: An overview of the hierarchical learning framework used in this work. Diagram credit to Mingyo. First, we get the desired hand trajectories and walking commands from the demonstrator. Second, we send these commands to the whole-body controller to produce joint-torque actions of the robot. Third, a behavior cloning policy is trained to imitate the human demonstrations.

the robot and demonstrators to collect demonstrations using immersive VR. We only track the SE(3) poses of the hands and abstract away the other joint configurations. This lets us retarget human motions to the humanoid robot, which has fewer degrees of freedom in the arms. This also makes it easier for learning, since the policy can delegate joint-space controls to the underlying controller. Even though human's 7-DOF arm could produce motions that cause singularity issues at the robot's 6-DOF arm, this can be easily corrected by the demonstrator.

2. Then, we send the desired hand trajectories and walking commands to the whole-body controller to produce joint-torque actions of the robot. The controller prioritizes the robot's stability while tracking the hand and feet trajectories, so it may not track the trajectories perfectly. However, the human demonstrator can adapt to the tracking error by observ-

ing the effects of their actions in the VR headset. The walking commands are implemented with a sequence-based DCM gait planner, which allows the user to take a step in one of the four directions or make a turn by clicking a button. Since these commands are discrete, the locomotion is not very fluid or precise.

3. Finally, a behaviorial cloning policy is trained to imitate the human demonstrations. The policy needs to learn the state-action distribution of whole-body control behaviors and adapt to it in a closed-loop manner similar to the human. During deployment, the generated trajectories are passed to whole-body control just like during demonstration. The policy achieves reasonable success rate in simple simulation tasks, but it has low success rate for more complex tasks involving loco-manipulation (manipulation and locomotion at the same time) and precise motions. Future research directions to improve the policy will be discussed in the end.

In the below sections, more details about the motivations for this project are described.

1.1 Humanoid Robots

Humanoid robots have gained a lot of attention in recent years. After Boston Dynamic's Atlas made headlines by jumping and dancing with human-like dexterity [10], Tesla also entered the market by developing the cheap and

mass-producible humanoid named Optimus. If the cost of the robot could be kept below $20,000 like Elon Musk promised [27], we would be entering a world where general purpose humanoids could replace humans for unsafe and repetitive tasks. Many startups are also getting a lot of funding recently to pursue humanoid robots. Apptronik is an Austin-based startup that aims to create humanoids that can work alongside people. One of their prototypes, DRACO 3, is the platform used for this thesis.

This interest in humanoids is justified by their versatility and social capabilities. Humanoids can be used as personal assistants, companions for the elderly, workers in factories, and first responders in disaster zones. The morphology of humanoids enables them to easily adapt to the human-centered world that we live in. Every tool, every building, and every task in our society are designed for the human form. It wouldn't make sense to redesign power tools or get rid of stairs for the convenience of robots, so creating robots that can take advantage of the existing infrastructure made for humans is extremely valuable. Also, as humans, it's easier to provide demonstrations to a robot that has a similar form as us, rather than having to train our brain to adapt to the morphology of the robot.

1.2 VR Teleoperation Interface

In order for imitation learning to succeed on robots, high-quality demonstrations are essential. Using a VR interface is desirable since it closes the observation and embodiment gap. Namely, instead of looking at the robot

externally, the human will have the same viewpoint as the robot. Instead of having to map the human body's joints to the robot's joints, the human is directly controlling the robot's joints. Because of the human brain's impressive capability to adapt, the demonstrator can quickly learn to treat the arms of the robot as their own arms, and perform tasks intuitively as they do in their own body. Indeed, there are various experiments that show that humans can start to take ownership of their virtual body, even if the body is very different from their own, if appropriate multisensory correlations are provided [15] [26].

In addition, since humanoid robots have to maintain balance while following hand trajectories, the whole-body controller may fail to track the hand trajectory perfectly. So, in order to move the robot hand to a desired position, one needs to constantly observe the effects of their actions before deciding where to move next. In experiments, we notice that humans are great at adapting to the whole-body control's tracking error in this closed-loop manner. By using the VR interface, we are essentially borrowing the human brain's power to solve the embodiment mismatch issue and perform closed-loop actions.

1.3 Scaling up Demonstration Dataset for Humanoids

Recently, we have seen the successes of training deep learning algorithms on mind-blowingly huge datasets. For example, GPT-4 [21] is believed to be trained on most texts on the internet, which enables its scalable transformer architecture to produce human-like texts. The ability of the transformer

to scale can also impact the robot learning field, since many transformer architectures for robotics have been proposed with impressive results [34] [14]. Notably, researchers at Google demonstrated that their transformer architecture is able to absorb a large dataset of multi-task demonstrations from different robots and even simulations, and the resulting model is able to generalize to unseen tasks, environments, and objects in a zero-shot manner [11]. Importantly, they demonstrated that by incorporating simulation data with the real-robot data, not only was the real-robot policy not noticeably degraded, but the generalization performance also improved significantly on objects only seen in simulation. This shows that even if the simulation isn't very realistic, the model can still absorb useful information that helps with generalization. The researchers stated that their dataset "consists of over 130k episodes, which contain over 700 tasks, and was collected with a fleet of 13 robots over 17 months." The scale of this dataset is very impressive, but the collection procedure seems very costly, and it is still small in scale compared to the billions of images used to train today's image generation models. In addition, the dataset isn't publicly available, and they are collected on single-arm wheeled robots instead of humanoids.

So, how can we scale up humanoid demonstration collection in a cheaper manner than investing manpower to collect demonstrations on the real robot? One proposal is to learn from the plethora of YouTube videos online. For example, [8] shows that by watching YouTube videos of house tours, an off-policy Q-learning algorithm can learn the semantic cues in a human environment to

improve navigation efficiency. To make it possible to learn dexterous manipulation skills from YouTube videos, [25] trained a neural network to retarget human finger poses from a video to a robotic hand.

Nevertheless, due to the difficulty of inferring 3D poses from video, the lack of proprioceptive view from the eyes of the human, and the ignorance of the demonstrator's internal state such as their joint positions, the wealth of information that online videos possess still remain out of reach for practical applications of robot learning.

Between learning from YouTube videos and collecting demonstrations on the real robot, taking advantage of the rich manipulation data from VR applications might just be practical enough to scale up humanoid data to satiate the appetite of today's deep learning methods. In VR, the human sees exactly what the robot sees, the hand poses are measured precisely by the headset and controllers, and the simulated robot provides full access to its internal states. As realistic simulation games such as Microsoft Flight Simulator rise up in popularity, we are presented with a valuable opportunity to collect high-quality human hand trajectories at scale for robot-learning research. This is not unreasonable since Meta is most likely already collecting information about the users' surroundings and actions for targeted ads [30].

Chapter 2

Background

In this chapter, the necessary background information for this work is introduced. Then, existing literature that relates to our goal will be surveyed.

2.1 Imitation Learning

In Imitation Learning, a task is demonstrated by a human instructor multiple times until a robot can imitate the demonstrated behavior and perform the task autonomously. There are many forms of Imitation Learning. The simplest form is Behavioral Cloning, where the robot learns to imitate the motion without inferring the actual goal. This becomes a supervised learning problem with the inputs being the observations and the outputs being the actions taken. This simple method is often effective [32], but it is prone to covariant shift [23], in which the errors in actions accumulate and the state deviates from the ones shown in demonstration. Another class of algorithms is Inverse Reinforcement Learning, in which the agent tries to reverse-engineer the reward function from the demonstrations. However, this is usually computationally expensive due to the requirement to run reinforcement learning as an inner-loop.

The specific Imitation Learning algorithm used in our work is Behavioral Cloning with Recurrent Neural Networks (RNN) and Gaussian Mixture Models (GMM). RNN maintains information about the past and can be viewed as injecting the inductive bias of sequentiality. Namely, it preserves time translation invariance by using the same neural network weights at each time step [6]. GMM uses a mixture of Gaussians with different means to approximate a multimodal distribution, such as the multiple ways that a human may perform a specific task.

2.2 Whole-body Control (WBC)

Whole-body control is an algorithm that aims to control the entire robot's body, including its arms, legs, torso, and head, to achieve a desired task. The specific form we are using is called implicit hierarchical whole-body controller [1], which uses an implicit hierarchy of tasks to allow for different prioritization of control depending on whether the robot is performing locomotion or manipulation. For example, we would like the robot to prioritize foot motion over hand motion when the robot is walking.

Whole-body control can be formulated as finding the optimal joint acceleration $\ddot{q}^*$ and contact forces f_r^* to minimize a loss function

$$\min_{\ddot{q}, f_r} \sum_{i=1}^{n} w_i \| J_i \ddot{q} + \dot{J}_i \dot{q}_m - \ddot{x}_i^d \|^2 + w_{f_r} \| f_r^d - f_r \|^2 + \lambda_q \| \ddot{q} \|^2 + \lambda_{f_r} \| f_r \|^2$$

- subject to the constraints of robot dynamics, contact, maximum reaction force, joint position limit, and torque limit. The first term in the loss function

represents the task space error, or the deviation of the robot's end-effectors' position and orientation from the desired values. The second term represents the contact force error. The third term regularizes the joint acceleration to control the trade-off between task performance and joint motion smoothness. The fourth term regularizes the contact force to prevent damage to the robot or the environment.

2.3 Related Work

2.3.1 VR for Imitation Learning

The work that most closely relates to ours is [32], which also involves using a consumer VR headset to teleoperate a robot. They show that by using only half an hour of demonstrations collected in VR, an Imitation Learning algorithm can be trained to perform simple manipulation tasks such as grabbing a ball or pushing a block. However, their robot has a fixed base, so they can directly use their robot's built-in Jacobian-transpose-based controller instead of a whole-body controller. Also, instead of streaming stereoscopic images to the headset, they use Unity to render the colorized point cloud from the robot's 3D camera to allow the user to look around freely in the VR world. They argue that since the robot's head has low degrees of freedom and moves slowly, controlling its movement using the headset's movement will cause motion sickness. Indeed, we are faced with the same problem. However, instead of using a point cloud that looks strange for the demonstrator, we simply disallow movements of the head. This is sufficient for our tasks, and it does not

seem to impact immersion. For learning, they also use a Behavioral Cloning algorithm. However, they are only controlling one arm of the robot, whereas we are controlling both. They are also using the depth image as an input, but we only provide stereoscopic images. Finally, we are using an RNN to preserve information from previous time steps, while they only provide the 5 most recent end-effector poses and no image history.

Another work that utilizes VR for robot learning is [4], which uses the hand-tracking capabilities of the Oculus Quest 2 headset to teach a robotic hand dexterous manipulation skills. They first retarget the human hand joints to a robotic hand with 4 fingers, then they use visual self-supervised learning combined with a simple nearest neighbor algorithm to successfully manipulate objects unseen during training. Although they are using a VR headset, they are not using stereoscopic rendering to create depth perception. Instead, they are rendering the robot's camera as a 2D video in Unity. Similarly, there is a senior thesis project teleoperating a small car in VR, and they also only use a 2D video displayed through Unity [17].

2.3.2 Demonstration Interface

There are multiple ways to collect demonstrations for imitation learning in existing literature. First, a teleoperation input device with 6 degrees of freedom such as the SpaceMouse can be used to control the position and orientation of the robot's end-effector [35]. However, it can be unintuitive for people to translate a 3D motion into the push and turn of a button, especially

if there is a need to control 2 arms at once. In addition, since SpaceMouse controls the velocity instead of position, it requires training to perform actions involving precision. Second, humans can directly hold the robot to move it in a desired trajectory by applying force [2] [24]. This is called kinesthetic teaching, but it requires the human to come into the frame to control the robot, which becomes a problem when the policy is trained on vision data. To avoid this, we can also build a replica of the robot and move the replica manually, while the main robot follows its trajectory. For example, [33] used a low-cost replica of their bi-manipulation robot to collect demonstrations for fine manipulation tasks. To do so, a human demonstrator pushes the end-effectors of the replica robot to backdrive its joints, and the resulting joint positions are issued as commands for the actual robot to follow. While this method achieved impressive results for the fixed-base robot, they cannot handle the floating-base dynamics of humanoid platforms, which requires torque control to account for the dynamics of the robot.

Since we can manipulate a replica of the robot to create demonstrations, why can't we use our own body as this replica? After all, humanoids are designed to mimic the morphology of humans. Indeed, we can directly record the kinematics of human motions [7]. Using either a camera or a motion-capture system, we can measure the angular displacement of the joints precisely, and then we can map the values to the robot's joints. However, robots can have different mass distributions and degrees of freedom than humans. So, the actions that humans perform may be impossible for robots or cause them to

lose balance. Therefore, learning a good mapping and using a whole-body controller to maintain balance are crucial to the success of this method.

2.3.3 Learning for Humanoids

As mentioned in the introduction, there does not seem to be prior work on training humanoid robots to perform manipulation tasks. Two works that cut close are [5], which uses Hidden Markov models to imitate a human demonstration, and [13], which uses motion planning to produce fluid transitions between locomotion, loco-manipulation, and manipulation. However, both works only consider a simple simulation with only kinematics and no dynamics, so their results are impossible to be applied to the real world.

Nonetheless, there are recent successes on learning humanoid locomotion. [19] adapts the Rapid Motor Adaptation work [18] to bipedal robots. It uses an adaptation module to enable the walking controller to adapt to a changing environment in real-time. [22] uses RL in an IsaacGym simulation to train a humanoid locomotion controller that responds to velocity commands robustly. Their work seems to be a more robust version of [19] where instead of explicitly adapting to the environment, the transformer implicitly infers the environment through past actions rapidly in every time step.

2.3.4 Humanoid Teleoperation

There is a recent surge in interest to remotely teleoperate robots. XPRIZE hosted the AVATAR competition in 2022, in which teams compete

to develop telepresence technology that allows a human to control a robot remotely to complete a set of tasks. Their overarching goal is for humans to be able to see, hear, touch, and interact with the world in the body of a robot, while the people themselves can be on the other side of the world. While most robots in the competition don't have legs, there are a few humanoids, and most teams are using VR technology to create an immersive visual experience.

There is a recent survey on teleoperation of humanoids [9], some of which competed in AVATAR. The authors also claim that humanoid teleoperation is a new field with high resource demand, so not many labs are working on it. The works they summarized include very complicated motion-capture and feedback setups that track the movement of users' feet and relay the sense of touch to the users. The survey unifies the teleoperation systems into a framework with 6 components. First, human measurements are taken. Then, the measurements are retargeted to the robot. After a delayed communication channel, the robot executes its controller to produce low-level commands. Finally, the robot's sensory input is retargeted to human feedback, and human teleperception such as the sense of touch is provided. This is similar to our framework, but we only have vision feedback. For actions, we only have hand trajectory, gripper control, and locomotion commands. They also mention that streaming the camera images to VR can cause motion sickness during locomotion, but this could be fixed by adopting digital image stabilization techniques.

The cockpit interface for NASA's Valkyrie humanoid is a good example

of a sophisticated VR teleoperation system [12]. The interface has a complex UI that allows the operator to directly specify where to place future footsteps. There is a 3D model of the humanoid for visualization, and both a depth point cloud and a 2D RGB footage of the environment are presented to the user.

These complex teleoperation systems make sense if the goal is to create immersive telepresence or to perform critical missions in outer space. However, if the goal is to teach robots common loco-manipulation skills, these systems can be overkill. In order to scale up data collection and lower the barrier of entry for humanoid research, I opted to use a single Oculus Quest 2 as the interface. Although it is missing many components such as feet tracking and tactile sensing, it is enough for simple tasks. In addition, VR is a rapidly developing industry, so future household headsets will mostly likely incorporate more sensors. For example, the recently-released Meta Quest Pro supports leg tracking, which can be used to control the robot's legs.

Chapter 3

VR Interface

In this chapter, I'll go into the implementation details of the VR interface. The requirements for the interface are as follows:

1. It needs to report the 6-DOF poses of left and right hands as well as additional buttons for locomotion and gripper control.

2. Stereoscopic images need to be streamed and displayed on the VR headset to create depth perception for the wearer.

3. It needs to connect to both the simulation codebase written in Python and the real-robot controller written in C++.

4. The latency should be low and the computation speed should be fast so that the wearer can provide demonstrations with ease. The computation resource consumption should be low because the simulation and whole-body controller are resource-intensive.

First, an overview of the architecture is given. Then, some of the design decisions are justified. Finally, the performance of the system is analyzed.

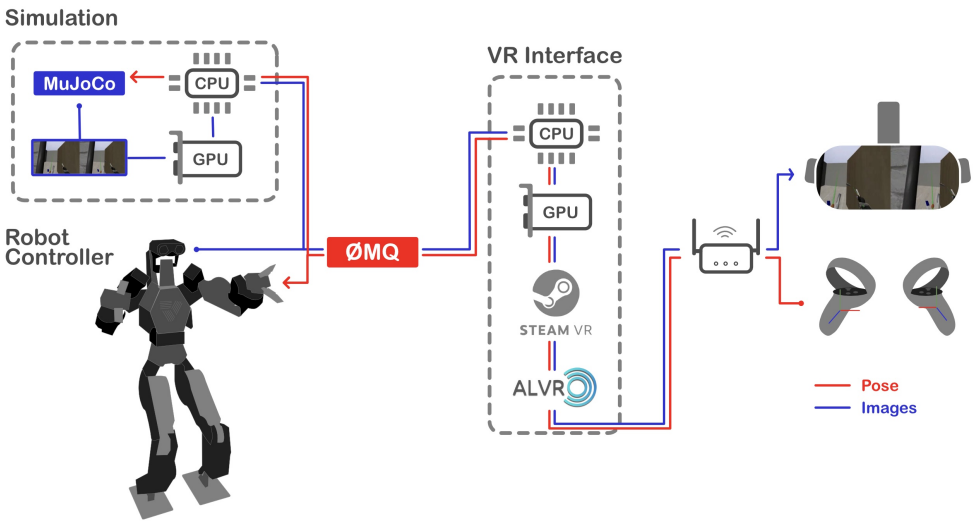

Figure 3.1: Architecture for the VR interface

3.1 Architecture

Figure 3.1 describes the software architecture. The VR interface code written in C++ runs on a separate laptop, which is connected to either the simulation desktop or the robot control station for the real robot. We use OpenVR (implemented in SteamVR) and ALVR (Air Light VR) for the communication between the laptop and the headset. OpenVR is a low-level API designed to support a wide range of VR devices. ALVR is an open-source project that allows streaming Steam VR games from the laptop to the headset via Wi-Fi. It implements technologies such as Asynchronous Timewarp and Fixed Foveated Rendering for a smoother experience. ZMQ is an asynchronous messaging library that simplies message-passing between different programs or devices.

For the simulation, we use Mujoco [28] with Python binding for the physical simulation and Robosuite [35] for the objects in the scene. We first render the scene using a virtual stereoscopic camera that is adjusted to match the interpupillary distance of the VR headset. Then, the rendered pixels are copied from the GPU to CPU and sent to the VR interface using ZMQ and ethernet. The interface code listens to the images and writes them into a GPU texture used by OpenVR. Finally, SteamVR and ALVR transmit the images through a router and displays them in the VR headset. At the same time, the interface polls the VR headset for the poses of the headset and controllers through OpenVR. It transforms the controller poses to the local frame of the headset, and they are then mapped to the poses of the robot hands in the robot's local frame. The latter mapping involves aligning the rotation axes of the controller with the axes used in the simulation. Specifically, let R_t be the transformation between the VR axes and the simulation axes, R_{vr} be rotation of the controller from its natural orientation, and R_{sim} be the desired rotation to be applied to the robot hands. Their relation can be expressed as

$$R_{sim} = R_t^{-1} R_{vr} R_t$$

The transformed hand poses are then published continuously using a ZMQ pub socket. When the simulation needs a VR command, it pulls the most recent command from the queue and sends it to the whole-body controller.

The setup for the real robot is similar. However, since the robot has a lot more components to manage, it will be described in more details in the next chapter.

3.2 Technical Details

Some design decisions were made to satisfy the requirements defined above. They are explained in this section in hopes of providing documentation and guidance for others working on similar systems.

3.2.1 OpenVR

Initially, a website was created based on WebXR and JavaScript to stream controller poses over the internet. I wanted to improve the latency by keeping the connection within a local network, but I encountered difficulty in setting up HTTPS certificates in the campus lab. After getting tired of tunneling the connection over the internet, I decided to pursue a more low-level approach. The native Oculus SDK could work, but it would limit the option to change VR systems in the future. Unity is a good option for cross-platform compatibility, but since there's only a need to stream images and controller poses, a game engine is an overkill. I also don't believe that Unity exposes the low-level functionality to directly show the stereoscopic images in the headset. Since Unity uses the low-level OpenVR API, I decided to use it directly. Although a newer API called OpenXR is gaining steam, I feel that the performance benefits of OpenXR doesn't justify its complexity compared to OpenVR.

Using OpenVR has several advantages. First, it has great performance since it is used by VR games and has direct support from VR headset manufacturers. The fact that all SteamVR games use it also makes it suitable for my

proposal to embed the demonstration collection system in video games. Second, it delegates the work of video streaming to existing technologies. Many VR headsets support direct HDMI connection from the computer's graphics card, which OpenVR can take full advantage of since it takes input images from OpenGL. Even though the Quest doesn't have an HDMI cable, it is still possible to use the Oculus Link (over an USB cable) or Oculus Air Link (over a local network) with OpenVR. These are sophisticated streaming technologies that predict the user's movements and streaming latency to render ahead of time, and they encode the frames as slices in H.264 [3]. ALVR is an open-source alternative that implements similar technologies with the added benefit of having experimental support for Linux. Using these technologies is much more performant and scalable than hand-coding an image streaming pipeline, as done by [4]. Finally, OpenVR works on a variety of Operating Systems and targets many VR headsets.

3.2.2 Image Transfer from GPU to CPU

Since our interface script is written in C++ (since the Python OpenVR binding doesn't work well) and the simulation is in Python, we need a way to transfer the rendered images between processes. First, we can use memory sharing or pybind to transfer the Mujoco simulation state. Using the simulation states, the C++ code can directly render the scene using Mujoco and pass the result directly to OpenVR within the GPU. However, since Mujoco allocates simulation data structure dynamically, it's difficult to do inter-process

memory sharing. Second, we could hypothetically share the GPU buffer rendered by the python script with the C++ script, but OpenGL contexts don't seem to allow inter-process sharing. Third, we can lose some performance and transfer the images from GPU to CPU. Once the images are in memory, we can send them over using ZMQ.

The last approach was chosen to make the design more adaptable to the real robot, where the images are always coming from memory. The rendered images have a resolution of 1096×2 by 1176, where the width is multiplied by 2 to account for both eyes. The rendering of an image takes .3 milliseconds on the RTX3090 GPU, copying it to the CPU consumes 3 milliseconds, and sending it through ZMQ uses .6 milliseconds. Overall, this number is negligible compared to the simulation and whole-body control times, so this performance is acceptable.

3.2.3 Asynchronous Message Passing

The simulation uses a ZMQ sub socket to get actions from the pub socket in the interface script. Usually this is done synchronously, meaning that the simulation waits for the interface to provide a response after requesting. However, this approach usually takes 70 milliseconds for the message round-trip. This delay can be reduced to .1 milliseconds by using an asynchronous approach. In this case, the sender and receiver simply run at their own pace, and the ZMQ threads in the background takes care of getting the messages ready. When the simulation requests an action, the ZMQ simply gets the most

recently received message in its queue and returns it. Since the interface runs at a higher frequency than the simulation, there will never be a case where no action is available. Also, by setting the "conflate" option in ZMQ, we can reduce the need of a queue by only keeping the most recent message.

3.2.4 Separate Computer for VR Interface

At first, the simulation and interface scripts ran on the same desktop computer. However, we noticed that the scripts were competing for resources, causing performance degradation. We chose ALVR since it's the only option on Linux to connect to VR, and its wireless design enables remote teleoperation. However, encoding the stream in real time consumes CPU power needed for the whole-body controller. When running, the simulation and whole-body controller takes up 1500% of the CPU, while ALVR and SteamVR consume around 200%. After decoupling the VR interface to a separate laptop, we noticed a performance improvement of around 10%.

3.3 Performance Analysis

The performance of the VR Interface is acceptable. Using asynchronous message passing and a separate laptop for VR interface, receiving images and sending commands are both sub-millisecond operations. The biggest contributor to latency is ALVR, which adds about 70 milliseconds of latency. However, this is nothing compared to the latency introduced by the slow simulation. In the simulation loop, for every rendering and communication with the inter-

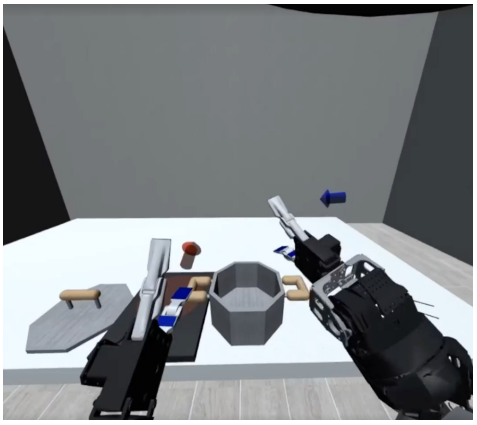

Figure 3.2: The robot hands takes time to reach the desired controller poses (represented by the floating red and blue arrows).

face, we run 25 simulation steps and 5 whole-body control computations. In comparison, on the hardware, the whole-body control code is run 20 times between every two action inputs. This means that in simulation, the whole-body control code may not have reached the desired action before a new action is given. Indeed, the slow tracking speed of the whole-body control introduces a large delay between the movement of the controller and the robot's hands reaching the desired poses. We qualitatively assessed this by rendering the desired poses as arrows in simulation as seen in Figure 3.2, and we found that for fast motions, the robot can take up to a second to reach the desired state.

Another big issue with the current setup is speed. The current simulation loop can only achieve around 10 fps in the contact-rich kitchen environment on a Linux machine with Intel i9-10900KF CPU and RTX 3090 GPU. This is also the reason we can't simply increase whole-body control compu-

tation frequency - it will just make the frame rate worse. Since a humanoid robot capable of walking requires an accurate physical simulation, we are using Mujoco with the RK4 integrator and PGS numerical solver at 50 iterations per time step, where the time step is set at 2 milliseconds. We have also tried running the numerical solver at 10 iterations per time step, sacrificing some precision for speed. For debugging, we tried rendering the images using OpenCV instead of sending them to the VR headset. The performance numbers from these different setups are shown in Table 3.3. Since the quickest loop time is 70 milliseconds, there is no hope that our setup can reach 60 fps. From the table, we see that using OpenCV to display images takes quite a lot of time, whereas sending the images to VR is faster. When walking is not involved, the whole-body controller time is reduced drastically, while manipulation tasks involving contact slows down Mujoco. Overall, Mujoco and whole-body control are the biggest culprits for performance issues, but the rendering performance with the copy from GPU to CPU can also be improved.

However, this is not the end of the story for potentially embedding our setup in a video game. As mentioned, our simulation codebase is written in Python. When the C++ codebase from the hardware team is used, each WBC computation is reduced from 5 to .45 milliseconds. In fact, the hardware team has their own simple simulation in PyBullet for debugging purposes. As mentioned before, they run 20 simulation steps and 20 WBC calls per loop. On the same machine that runs the simulation code, their loop is able to execute 40 times per second with the VR interface (so the WBC and simulation execute

Table 3.1: Simulation performance numbers. The left column denotes whether OpenCV or the VR headset is used to display the image and the number of PGS iterations per time step. The last row shows a manipulation task without locomotion. In each loop, Mujoco runs 25 times, WBC runs 5 times, and rendering runs once. The first three columns measure the time taken in each run, the fourth column shows the total time taken by each loop, while the last three columns show their percent contribution to total loop time.

	Mujoco (ms)	Render (ms)	WBC (ms)	Total (ms)	Mujoco	Render	WBC
CV, 50	1.72	30	7.7	120	35%	27%	31%
CV, 10	.8	30	7.7	100	21%	33%	39 %
VR, 50	1.72	8	6	92	48%	8%	34%
VR, 10	.92	8	6	70	33%	11%	44%
VR, 50 no walking	2.12	8	5	90	57%	8%	25%

at 800 hz). Needless to say, one of our current tasks is to substitute our Python code with their C++ code.

Chapter 4

Real-Robot Experiments

In this chapter, the details of how our experiments are conducted on the real robot are introduced. However, due to the complexity of the robot system, the infrastructure built for collecting demonstrations and deploying the neural network policy will be described first. Specifically, a real-time system was built to integrate the camera, grippers, the whole-body controller, and the VR headset. Then, the data collection and neural network policy evaluation results (or lack of thereof) are discussed.

4.1 Infrastructure

Figure 4.1 shows the infrastructure for collecting demonstrations and deploying trained policies on the real robot. The VR Interface script is the same as before, but more components are added to interface with the grippers, camera, and the robot-control desktop. The whole-body controller running on the desktop is a modified version of PnC [1], named RPC (Robot Planning and Control), created by the hardware team.

When collecting demonstrations, the camera footage is streamed from the robot's stereo camera to the VR headset. Both RPC and the gripper

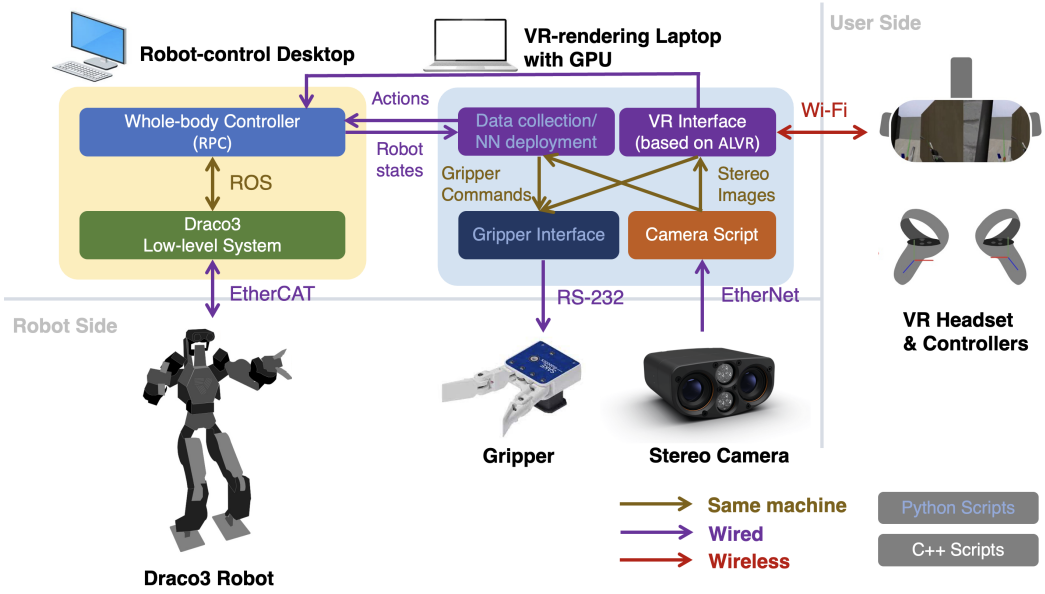

Figure 4.1: The infrastructure used to collect demonstraions and deploy policy on the real robot. Original diagram by Mingyo.

interface subscribe to VR commands to execute desired actions. The data collection script then pulls robot states from RPC and camera footage from the camera script and saves the resulting data into an HDF5 file. The saved data is then post-processed and used to train the neural network. During deployment, the observations (robot states and camera footage) are processed in real-time and fed into the neural network. The processing involves converting hands and feet poses to local frame of the robot and normalizing and resizing images. Then, the neural network inference is performed on the GPU laptop, and the output commands are issued to the grippers and robot. To protect the robot, the output commands are restricted by a 3D bounding box, and the maximum arm movement speed is restricted.

4.1.1 Image Streaming From Camera to VR

Unlike simulation, the parameters of the camera on the robot cannot be easily adjusted. This means that the interpupillary distance and FOV of the camera does not match those of the human eyes, causing dizziness in the demonstrator [31]. Although cropping the images helps with adjusting the convergence distance of the cameras, the difference in interpupillary distance cannot be completely fixed. In addition, there is only RGB data from the left camera, whereas the right camera only has grayscale images. Instead of colorizing the right image from the left image using computer vision techniques, which can produce artifacts that are easily noticeable by the human, we opted to only display grayscale images in the headset. Although the experience isn't very smooth, the depth perception provided by stereoscopic images is still enough to complete many manipulation tasks.

The camera is the MultiSense S7 model from Carnegie Robotics. It comes with a low-level driver library with minimal documentation. RGB image streaming is achieved by executing callback handlers for luma and chroma images on separate threads and merging them together into RGB. Similarly, left and right luma images are streamed in separate threads, which are then synchronized and stitched together to form stereoscopic images. This multi-threaded C++ code is designed to incur minimal image-copying and is thus very efficient.



Figure 4.2: 150 demonstrations of a simple pick-and-place task are collected. The egocentric view for 36 of the episodes are shown.

4.2 Neural Network Training and Evaluation

Our Neural Network takes in the following inputs: stereoscopic images resized to 800×200, positions and velocities of the 26 joints in sin and cos form, the SE(3) poses of the hands and feet in local frame, and the current state machine of the robot, such as balancing or walking forward. A Resnet is trained to extract the image features, and an RNN is used to process the features as well as other inputs to produce an output distribution. The continuous outputs such as hand trajectories are fed into a GMM to produce manipulation commands, while discrete outputs such as locomotion commands and grippers are rounded to either 0 or 1.

To train the network, 150 demonstrations are collected on a simple pick-and-place task involving grabbing a temperature gun and dropping it in a box,

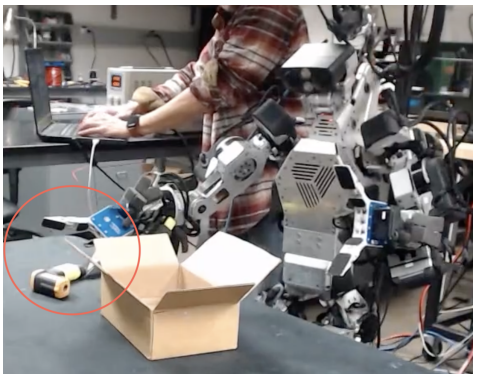

Figure 4.3: During evaluation, the robot usually knocks over the temperature gun before grasping it.

as shown in Figure 4.2. Since the hardware team is working on replacing the force-torque sensors in the robot's ankles, the robot cannot walk yet. Instead, the task only involves the robot balancing on two feet and doing manipulation with its hands.

However, the evaluation of the trained policy is not very successful. Out of 20 evaluations, the robot is only able to pick up the temperature gun in 1 episode. As shown in Figure 4.3, the robot fails to locate the object and often knocks it over.

We hypothesize that the tracking error of the whole-body controller is partially responsible for the difficulty to learn precise manipulation. Figure 4.4 shows whole-body control's tracking error of the right hand during deployment. The red line is the desired pose, and the blue line is the actual pose. Each bump corresponds to a new episode, and the large spikes on the orientation are due to resetting the robot after a failed episode. Since the blue line changes as

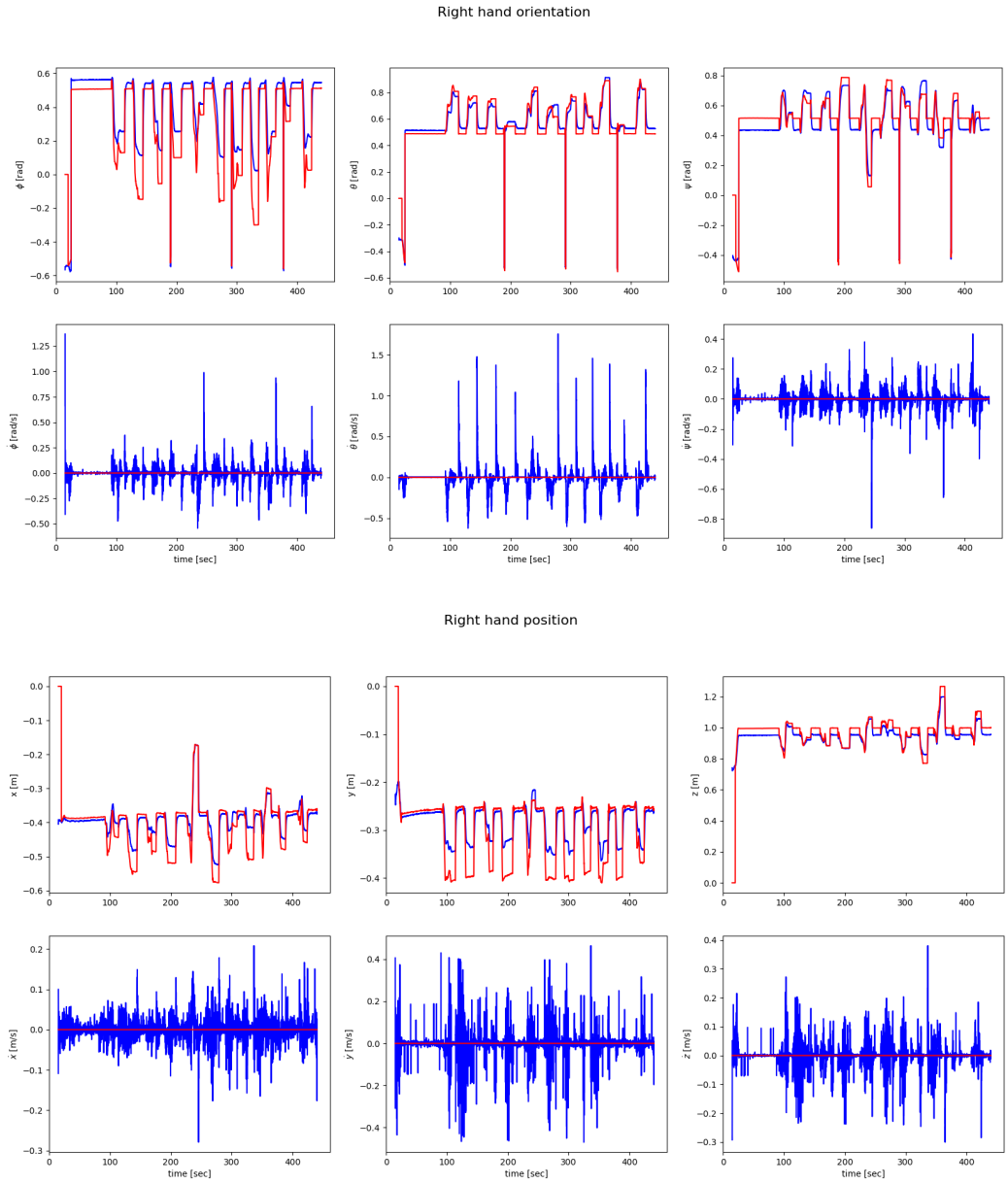

Figure 4.4: Plotting the desired (red) vs actual (blue) position and orientation of the right hand during deployment. For each grid, the top row is position and the bottom row is velocity.

soon as the red line changes, we know that the tracking is very fast, unlike what we have in simulation. However, the steady-state error is quite large. In the y direction, there can be an up to 10 centimeters error for the hand position. Steady-state error is caused by the lower weight assigned to the hand-tracking task in comparison to the center-of-mass task. In order to maintain balance, the whole-body controller refuses to extend the arm too far forward. This can be fixed by assigning a higher weight to the hand task, but this would trade off the stability of the robot.

Nonetheless, the policy should learn to compensate for the tracking error, as it has done in simulation. We have also noticed that the policy doesn't vary its actions when the object is placed in different places. We are currently still working on debugging the neural network deployment, but unfortunately, due to the frequent need to repair the robot, we are not able to solve it by the thesis due date. However, the difficulty of performing real-robot experiments on a humanoid platform is the very reason we started out with simulation. Indeed, our method achieves much better results in simulation.

Chapter 5

Simulation Experiments

Knowing that real-robot experiments are expensive and error-prone, we developed a simulation in Mujoco as a test bed for our algorithms. In simulation, we can also access data that is not accessible in the real world, scale up demonstrations quickly, and have a controlled environment with reproducibility to test new methods. In the following sections, the tasks used for evaluation are introduced, and the results are presented and discussed.

5.1 Tasks

The first environment is a door in a hallway. It contains two tasks: opening the door and walking through the door. For the first task, the robot needs to grab the handle, turn it, and push the door open. Partial success is defined by the robot grasping the handle. This is a relatively easy task involing single-arm manipulation. The second task is a loco-manipulation task, where the robot needs to walk forward while pushing the door open. Partial success is given if the robot opens the door but doesn't walk through it. The initial pose of the robot is randomized based on a normal distribution for both tasks, with a standard deviation of 2 centimeters for the position and .07 radians for

the orientation.

The second environment is a kitchen with two tasks: moving a pot and placing a lid. The first task is a loco-manipulation task requiring the robot to lift up a pot with two hands, walk a step towards a stove, and put the pot on the stove. Partial success is granted for lifting the pot. For the second task, the robot should grab the lid next to the pot and put it on the pot. Grasping the lid constitutes a partial success. This task requires precision. The positions of the pot, stove, and lid are randomized along with the initial pose of the robot. The stove's position is uniformly distributed from -5 to 5 centimeters, while the pot and lid are placed from -3 to 3 centimeters away from the stove. The pot and lid are made to be light since the robot dynamic model currently does not consider external forces.

We collected 200 demonstrations for each of the 4 tasks. The neural network architecture and training are identical to those introduced in Chapter 4.

5.2 Results

The evaluation results for each task are presented in Table 5.1. Image sequences of successful episodes are presented in Figure 5.1.

A common failure mode for the pot task is the robot falling down when moving the pot. When the robot is side-stepping, the robot's arms tend to swing sideways to maintain balance. However, if the robot is holding the pot

	Success	Partial success	Fail
Open door	80%	13 %	7%
Walk through door	74 %	13%	13%
Moving pot	8 %	53 %	40%
Move lid	8 %	53 %	40 %

Table 5.1: The evaluation success rates on the 4 simulation tasks.

with two hands, the frictions on the pot handle constrain the arm motion. After lifting the pot, if the hands are not kept steady during side-stepping, the robot can easily fall.

The low success rate for moving the lid can be explained by the precision needed to grasp the lid's handle, which requires inserting the robot's gripper below the handle and above that lid. Also, when the lid is being put on the pot, occlusion occurs since the robot's arm is blocking the view of the pot. When the robot's arms are made invisible, we found that the success rate is 10% higher.

Similar to the hardware experiments, we hypothesize that the whole-body controller tracking error is partially responsible for the imprecise manipulation. It's also not clear if the model is able to successfully extract depth information from the stereoscopic images. SimNet [16] demonstrates that this is indeed possible, but they're using a stereo matching neural network crafted specifically for the purpose of extracting depth information, whereas we are simply training a ResNet in an end-to-end manner directly on the downstream task.

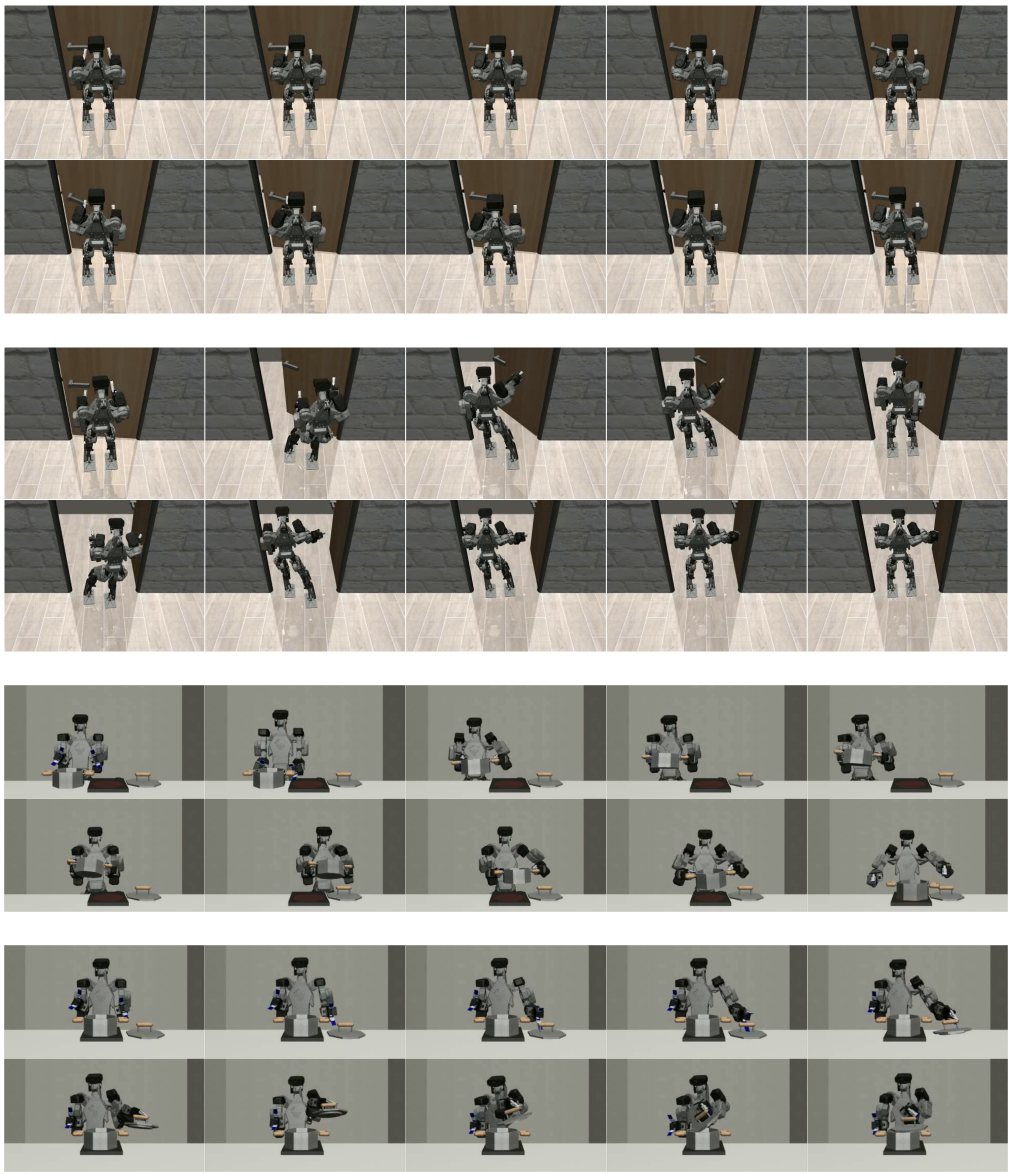

Figure 5.1: A successful episode from each simulation task. From top to bottom, we have opening the door, pushing the door, moving the pot, and placing the lid. The 2×5 image sequences are read from left to right and top to bottom. Note that the robot swings laterally when performing locomotion.

Chapter 6

Conclusion

In the introduction, we hypothesized that VR can be used as an intuitive interface to teach loco-manipulation skills to humanoids, and that the data collection can be massively scaled up by collecting demonstrations from video games. How are these hypotheses holding up against the data?

In the experiments, we found that VR is indeed a natural interface for teleoperating humanoid robots. We were able to easily collect hundreds of loco-manipulation demonstrations in simulation and on the real robot. We also confirmed that by using a whole-body controller to abstract away the low-level joint-torque controls, it is feasible for a baseline behavioral cloning algorithm to learn the high-level commands to perform simple tasks in simulation. However, the policy struggles to compensate for the tracking error produced by whole-body control when precise motion is required. Also, we have not yet successfully deployed the policy on the real robot. Nevertheless, this is the first attempt in literature to teach humanoids loco-manipulation tasks, and we believe that the next few iterations of our method can achieve greater success.

The performance of our Python-based simulation is very far away from

providing a smooth experience that video games demand. However, by switching to the C++ implementation of whole-body control, using a headset that directly connects to a computer's GPU, and optimizing our Mujoco simulation, we are confident that we can achieve a smooth simulation experience. Another option is to get rid of the whole-body control and simulated robot alltogether and simply record the player's motions and observations. I believe that demonstrations collected this way can still be invaluable to humanoid loco-manipulation research in the future.

Chapter 7

Future Work

We spent most of the year building up the infrastructure for conducting humanoid research, but the actual research is just getting started. I plan to spend the summer to continue working on this project. Here are the items currently on my list - besides open-sourcing the interface code and speeding up the simulation.

We have several hypotheses for why the baseline method isn't performing well, and we plan to conduct experiments to verify them. First, we want to verify if providing depth information instead of stereoscopic images helps with the performance. If so, we could either provide the depth information using the depth camera on the robot or pre-train the Resnet to predict depth information before solving the downstream task. Second, we don't know if the neural network is able to infer ground-truth object positions. If the object positions are known and the WBC error is nonexistent, then the task can be solved trivially. Therefore, if we eliminate the difficulty caused by understanding the positions of objects, then we can see how difficult it is to learn the WBC dynamics. If the performance of the policy improves dramatically when the objects' ground-truth positions are provided, then we can train a

module such as SimNet [16] to provide this information to the policy. Finally, most tasks only require engaging one hand at a time. Since the policy doesn't understand that the other hand is useless, it might overfit to the random motions of the useless hand. We can verify this by manually reducing the loss of the idle hand.

Learning the WBC's tracking error is one of the main challenges of our hierarchical approach. We can further tune the WBC hyperparameters to reduce the steady-state tracking error. We can also restrict the bounding box to prevent commands from going outside the reachable range. These methods can improve the tracking performance, but they won't completely eliminate the issue. Since we are providing positions and velocities of each joint to the model, it's very possible for the policy to overfit to the WBC's behavior on specific joint configurations during the 200 demonstrations. Since WBC is highly non-linear, it may not be possible to learn its behavior fully using imitation learning. Therefore, we can try to train a neural network to predict the WBC's behavior using a self-supervised approach, where the robot generates random hand motions and measure the WBC's behavior, which could provide sufficient data for generalization.

There are also many things from related work that we should try. In HoloDex [4], the authors used self-supervised learning to learn a low-dimensional embedding from high-dimensional images. By using data augmentation, the image encoder can be trained to recognize only the important features of the image. Since our data from the real robot is small in scale, it's

very possible for our image encoder to overfit to the lighting in the room and irrelevant objects in the background, which can be solved by data augmentation. Also, we currently have to re-train the encoder for every new experiment, whereas their representation learning approach allows us to re-use the image encoder to save computational resources. Their k-nearest neighbors implementation for the downstream task is also worth trying as a baseline.

Finally, the recent work on learning locomotion of humanoids [22] with RL has numerous implications on our project. First, the fact that their transformer is able to learn the robot kinematics and dynamics purely from past observations means that it can easily learn the whole-body control behavior. We are currently implementing a transformer architecture to replace our RNN. Second, we plan to use RL to fine-tune our policy. After all, the policy can only do so well by imitating the human behavior while being completely ignorant of the task objective and environment interactions. The reward functions in their paper - which takes a whole page to enumerate - account for almost all the scenarios that can be dangerous to deploy on the real-robot, which is why they were able to perform sim-to-real. If we optimize our simulation performance and take inspirations from their reward functions, we could harness the power of combining RL and IL, which is a popular research area recently [29].

Bibliography

[1] Junhyeok Ahn, Steven Jens Jorgensen, Seung Hyeon Bang, and Luis Sentis. Versatile locomotion planning and control for humanoid robots. *Frontiers in Robotics and AI*, 8, 2021.

[2] Barış Akgün, Maya Cakmak, Karl Jiang, and Andrea Lockerd Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4:343–355, 2012.

[3] Volga Aksoy, Irad Ratmansky, and Amanda Watson. How does oculus link work? the architecture, pipeline and aadt explained. Oculus Developer Blog, 2019.

[4] Sridhar Pandian Arunachalam, Irmak Güzey, Soumith Chintala, and Lerrel Pinto. Holo-dex: Teaching dexterity with immersive mixed reality, 2022.

[5] Tamim Asfour, Florian Gyarfas, Pedram Azad, and Rudiger Dillmann. Imitation learning of dual-arm manipulation tasks in humanoid robots. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 40–47, 2006.

[6] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti,

David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[7] A. Billard and D. Grollman. Robot learning by demonstration. *Scholarpedia*, 8(12):3824, 2013. revision #138061.

[8] Matthew Chang, Arjun Gupta, and Saurabh Gupta. Semantic visual navigation by watching youtube videos, 2020.

[9] Kourosh Darvish, Luigi Penco, Joao Ramos, Rafael Cisneros, Jerry Pratt, Eiichi Yoshida, Serena Ivaldi, and Daniele Pucci. Teleoperation of humanoid robots: A survey, 2023.

[10] Boston Dynamics, YouTube. Do you love me? `https://www.youtube.com/watch?v=fn3KWM1kuAw`, 2020.

[11] Anthony Brohan et al. Rt-1: Robotics transformer for real-world control at scale, 2022.

[12] Steven Jens Jorgensen et al. Cockpit interface for locomotion and manipulation control of the nasa valkyrie humanoid in virtual reality (vr), 2022.

[13] Paolo Ferrari, Marco Cognetti, and Giuseppe Oriolo. Humanoid whole-body planning for loco-manipulation tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4741–4746, 2017.

[14] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts, 2022.

[15] Konstantina Kilteni, Antonella Maselli, Konrad P. Kording, and Mel Slater. Over my fake body: body ownership illusions for studying the multisensory basis of own-body perception. *Frontiers in Human Neuroscience*, 9, 2015.

[16] Thomas Kollar, Michael Laskey, Kevin Stone, Brijen Thananjeyan, and Mark Tjersland. Simnet: Enabling robust unknown object manipulation from pure synthetic data via stereo, 2021.

[17] Alexis Koopmann, Kressa Fox, Phillip Raich, and Jenna Webster. Virtual reality teleoperation robot. `https://my.ece.utah.edu/~kstevens/4710/reports/vr-robot.pdf`.

[18] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots, 2021.

[19] Ashish Kumar, Zhongyu Li, Jun Zeng, Deepak Pathak, Koushil Sreenath, and Jitendra Malik. Adapting rapid motor adaptation for bipedal robots, 2022.

[20] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation, 2018.

[21] OpenAI. Gpt-4 technical report, 2023.

[22] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. Learning humanoid locomotion with transformers, 2023.

[23] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011.

[24] John Schulman, Jonathan Ho, Cameron Lee, and P. Abbeel. Learning from demonstrations through the use of non-rigid registration. In *International Symposium of Robotics Research*, 2013.

[25] Aravind Sivakumar, Kenneth Shaw, and Deepak Pathak. Robotic telekinesis: Learning a robotic hand imitator by watching humans on youtube, 2022.

[26] Mel Slater, Daniel Pérez Marcos, Henrik Ehrsson, and Maria Sanchez-Vives. Towards a digital body: the virtual arm illusion. *Frontiers in Human Neuroscience*, 2, 2008.

[27] Tesla, YouTube. Tesla AI day 2022. https://www.youtube.com/watch?v=fn3KWM1kuAw, 2022.

[28] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[29] Caroline Wang, Garrett Warnell, and Peter Stone. D-shape: Demonstration shaped reinforcement learning via goal conditioning, 2023.

[30] XpertVR. Vr data collection: Traditional and contemporary data types. https://xpertvr.ca/vr-data-collection/, 2022.

[31] Gao Z, Hwang A, Zhai G, and Peli E. Correcting geometric distortions in stereoscopic 3d imaging, 2018.

[32] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation, 2018.

[33] Tony Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023.

[34] Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. Viola: Imitation learning for vision-based manipulation with object proposal priors, 2023.

[35] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning, 2022.