



**UNIVERSITY
OF TURKU**

Data Collection for Humanoid Robot Control

Mechanical Engineering / Department of Mechanical and Materials Engineering

Bachelor's Thesis

Elias Patronen

26.5.2026

Turku

The originality of this thesis has been checked in accordance with the University of Turku
quality assurance system using the Turnitin OriginalityCheck service.

Subject: Mechanical Engineering

Author: Elias Patronen

Title: Data Collection for Humanoid Robot Control

Supervisors: Dr. Anam Tahir (University of Turku, Finland), Dr. Jorge Peña Queralta (Skya AI, Finland)

Number of pages: 22 pages

Date: 26.5.2026

Training generalist vision-language-action models for humanoid robot control requires large amounts of demonstration data. This data consists of robot movement data, images from cameras on the robot, and a description of the task the robot is doing. This thesis provides an overview of where this data comes from, the differences in the formats used by different datasets, and the viability of translating data between formats. Four sources of training data are examined. One dataset is produced locally using teleoperation, demonstrating a general setup and workflow for recording datasets. The formats used by these training data sources are analyzed thoroughly and the challenges of translating data from one format to another are discussed. It is found that translating data between formats which store different amounts of data can be difficult and that, for most use cases, it is not necessary to make a new format which is more general than those already available.

Keywords: Humanoid robots, robot control, vision-language-action models, data format

Oppiaine: Konetekniikka

Tekijä: Elias Patronen

Otsikko: Datan Keräys Humanoidi Robottien Ohjaukseen

Ohjaajat: Dr. Anam Tahir (Turun Yliopisto, Suomi), Dr. Jorge Peña Queralta (Skya AI, Suomi)

Sivumäärä: 22 sivua

Päivämäärä: 26.5.2026

Jotta yleiskäyttöön tarkoitettuja näkö-kieli-toimi malleja voidaan kouluttaa ohjaamaan humanoidirobotteja, vaaditaan paljon demonstraatiodataa. Tämä data koostuu robottien liikkeidatista, robotin kameroista saaduista kuvista, sekä robotin tekemän tehtävän kuvauksesta. Tämä tutkielma tarjoaa yleiskuvan siitä, mistä tämä data tulee, eridatasettien formaattien eroista, sekä mahdollisuuksista kääntää dataa eri formaattien välillä. Neljää koulutusdatan lähdettä tutkitaan. Yksi datasetti tuotetaan paikallisesti käyttäen teleoperaatiota. Näin demonstroidaan yleisesti datasettien tallentamiseen käytettyä järjestelyä sekä työnkulkua. Koulutusdatalähteiden käyttämiä formaatteja analysoidaan perusteellisesti ja haasteita datan kääntämisessä yhdestä formaatista toiseen käsitellään. Tutkielmassa löydetään, että datan kääntäminen formaatista, jolla voidaan tallentaa enemmän dataa, formaattiin, jolla voidaan tallentaa vähemmän dataa, voi olla vaikeaa sekä, että useimmille käyttötapauksille ei ole tarpeellista luoda uutta formaattia, joka olisi enemmän yleispätevä, kuin jo olemassa olevat formaatit.

Avainsanat: Humanoidirobotit, robottinohjaus, näkö-kieli-toimi mallit, dataformaatti

Contents

1	Introduction	1
2	Prerequisites	3
2.1	Vision-Language-Action Models	3
2.2	Task and Joint Spaces	4
2.3	Inverse and Forward Kinematics	4
2.4	State and Action Spaces	5
3	Methodology	7
3.1	Data Sources	8
3.2	Simulation Setup	10
3.3	Challenges	12
4	Dataset Format Analysis	14
4.1	Unitree JSON	14
4.2	TWIST	15
4.3	Reinforcement Learning Datasets	16
4.4	LeRobotDataset	17
4.5	Unification	18
5	Conclusion	20
5.1	Further Work	21
	References	23

List of Acronyms

RLDS Reinforcement Learning Dataset

VLA Vision-Language-Action

VLM Vision-Language Model

WAM World Action Model

WSL Windows Subsystem for Linux

1 Introduction

Robot control has evolved significantly in recent years. It is no longer required to program every movement manually with, for example, G-code. Nowadays artificial intelligence can be used to control robots and today's paradigm for autonomous robot movement is the use of general-purpose vision-language-action (VLA) models [1]. AI policies like these are trained on demonstrations of tasks. These are usually gathered by manually controlling the robot to do the task a number of times and recording the process. These datasets of demonstration recordings are then fed to the AI which learns to do the task based on them. To train VLA models like this, especially for controlling humanoid robots, large datasets are currently required.

In this thesis, different sources for gathering data to train VLA models to control humanoid robots are presented and the differences between formats used to store data in datasets are analyzed. To achieve the goals of producing a dataset with teleoperation, the following research questions are addressed:

- What data sources are available for training VLA models to control humanoid robots?
- What differences exist among VLA model training dataset formats?

By knowing the differences in formats used in datasets, it is possible to translate datasets into different formats. This enables validating VLA model performance and comparing different VLA models since different policy training programs accept differently formatted datasets. Being able to translate datasets into different formats also enables combining datasets of different formats to train on even bigger datasets. Some datasets are also published online, meaning the ability to translate between formats allows for not only combining these datasets into training data but also publishing datasets in collections where a unified format is required.

This thesis is divided into five chapters. After the introductory chapter 1, chapter 2

presents the theoretical background relevant to this study. This includes vision-language models, task and joint spaces, inverse and forward kinematics, as well as state and action spaces. Chapter 3 describes the research methodology, including the utilized data sources, simulation setup, and the challenges encountered during the implementation process. Chapter 4 provides an analysis of different dataset formats, including Unitree JSON, TWIST, Reinforcement Learning Datasets, and LeRobotDataset, followed by a discussion on dataset unification. Finally, chapter 5 concludes the thesis and presents recommendations for further study.

DeepSeek AI was used during the research process of this thesis to help with software troubleshooting and reverse engineering. It was also used at the end of the writing process to check grammar.

2 Prerequisites

2.1 Vision-Language-Action Models

Vision-language-action models are foundation models trained to control robots. They are built on top of vision-language models (VLM). VLMs produce text as an output from an input of text and image data. VLA models build on this by changing the output to be low-level robot actions [1]. The difference between VLAs and previously used robot policies is that VLAs are inherently generalist and multi-task. They work on multiple different environments and embodiments and on multiple different tasks including previously unseen ones, all with little to no fine-tuning [2].

Robot movement datasets are inherently task-specific. The raw movement data of one embodiment doing one task in one environment is not directly applicable to another embodiment doing another task in another environment, unlike with VLMs where one response to one question might be applicable to another question elsewhere. This is why VLA model training datasets must be very large and multimodal, or in other words, as variable as possible. They must include examples of as many configurations as possible. If you want the VLA model to do multiple different tasks, the dataset must include examples of that task or similar tasks being done in all those different configurations. A robot trained to only fold clothes might understand the task of picking up a coffee cup but it will not know how to push the buttons on a coffee machine. Nowadays these datasets are not only published alongside research publications in journals but also online in open-source, community-driven collections. [2]

Current research efforts in the field of robot control models are in developing better generalization and reducing the required size of training datasets. In the future, training datasets may not need to be as big as they are today. $\pi_{0.6}^*$ introduces reinforcement learning into the training of the model, reducing the needed training dataset size by making the robot learn from experience [3]. $\pi_{0.7}$ generalizes better not by increasing training dataset size but by focusing on prompt quality by specifying “not only what the

robot should do, but how it should do it” [4]. DreamZero replaces the VLA with a World Action Model (WAM) which understands and predicts the world around the robot by using a pretrained video diffusion backbone. This completely new type of robot control model needs only 30 minutes of play data to adapt to a new robot [5].

2.2 Task and Joint Spaces

Task space and joint space are ways of representing the state or current positions of all joints on a robot. Task space is a more human-understandable way of looking at joint positions, while joint space is usually better for commanding the movements of a robot.

In task space, the positions of joints are given in Cartesian coordinates, usually relative to the previous joint on the arm. This way it is possible to calculate the positions of each joint on the robot given a desired end-effector position in space. By knowing the position of an object in relation to the robot, it is possible to guide the hand of the robot to pick up said object. For classical robot tasks like pick and place or writing, only the position of the end-effector is relevant, but for whole body tasks and generalist VLAs meant to work on multiple different embodiments, it is important to also consider the positions of other joints of the robot and the position of the robot in relation to the world around it. [6]

In joint space, also known as configuration space, the positions of the robot’s joints are given by the angles of the joints. This is the only space the robot understands on a hardware level. However, it is fairly difficult for a human to intuitively control a robot by just giving it joint angles to target. No matter which representation is used throughout the process of controlling a robot, the final low-level command to control the individual motors in joints will always have to be in joint space. [6]

2.3 Inverse and Forward Kinematics

It is fairly common to only know either the task space or joint space state of the robot and want to know the other. Fortunately, it is possible to calculate joint angles from Cartesian coordinates and vice versa. Forward kinematics is the methodology for calculating the

position of a robot arm given its current parameters. By knowing the angles of the joints of the robot and the lengths of the links between the joints, it is possible to find the Cartesian coordinates of a part of the robot with respect to the base of the robot arm. Inverse kinematics, on the other hand, is the methodology for calculating the robot parameters needed for a part of the robot to be in a specific place in space. In other words, by knowing the Cartesian coordinates of a point on the robot, it is possible to calculate the joint angles for a robot of specific link lengths to have that point in that position in space. [7], [8]

Classically these concepts are thought of in the context of industrial robot arms but they can be abstracted to robots of all shapes including humanoid robots. Nowadays most kinematics calculations are done computationally since they are tedious to calculate by hand but easy for computers to compute. For example, a robot may be made to have its end-effector follow a moving point in space. The robot will only be given the point in space in relation to its own position. In this case it is important for the robot to be able to calculate kinematics efficiently since it will need to calculate the needed joint angles every time the point moves in space.

2.4 State and Action Spaces

The state space represents the current state of the robot and its environment. This can include the current positions of all the joints and links of the robot, the robot's location relative to the world around it, locations of all manipulated objects, etc. All information available could be gathered and included in the state space. However it is usually better to choose only the required information and to choose one way of representing it. This means choosing between joint space or task space for the positions of the robot's joints. [9]

The action space represents all the possible actions available to an agent. An action is the decision made by the agent based on its current policy and goal. In robot control and for VLA models, this action is usually low-level robot commands in either joint or task

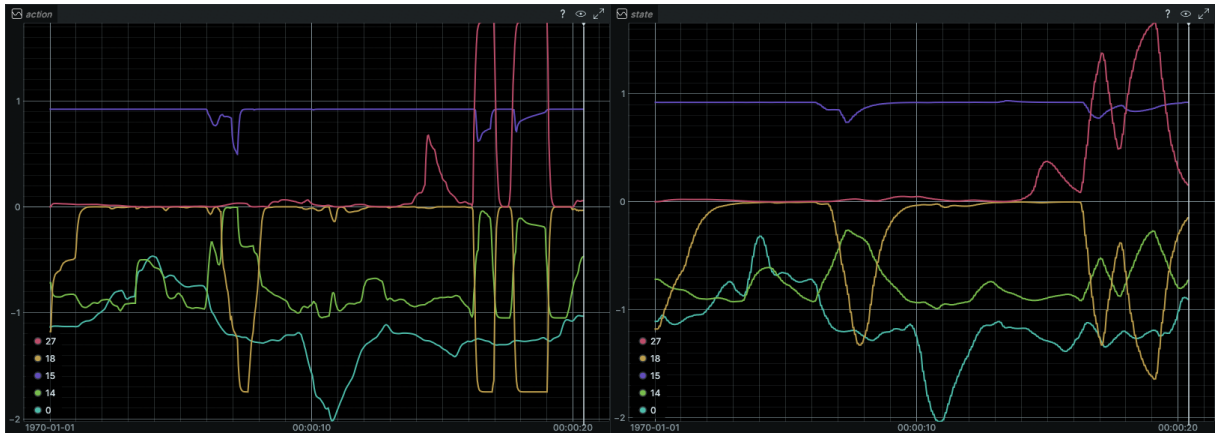


Figure 1: Action and state in a dataset. Joint angle over time for a few selected motors on the robot. Modified from a screenshot from Rerun Viewer data replay program.

space. When gathering data to train VLA models, the actions given by the demonstrator are recorded and the model is trained to output these actions based on the current state of the robot.

Figure 1 shows the difference between state and action spaces in robot control data. The method for collecting this data is shown in a later section. Both graphs show the joint angles of a few selected motors over time. On the left, the graph of action shows the movement commands given to the robot, which are calculated from hand movement data collected from the teleoperator. On the right, the graph of state shows the movements of the robot as a result of these movement commands. The graphs demonstrate why both are necessary to capture in datasets. The state does not always match the action. This is because the robot is not always able to move as fast as the actions it is given. Its movements may also be affected by objects it is carrying or obstacles it is hitting. In this way, knowing the state is a way for the VLA model to get feedback from the robot's environment, and in training it shows the model how to react to outside feedback.

3 Methodology

VLA models learn by observing demonstrations of how to perform tasks. The VLM that the VLA model is built on will understand a task description and may be able to generate text that describes what needs to be done to execute the task. It will also understand images of the task setup and be able to understand which objects are related to the task and describe how the scene may change when the task is done. The VLM will not be able to generate low-level robot actions based on its reasoning without training. The training dataset therefore consists of demonstrations of a robot doing a given task. The dataset will include the task prompt and a recording of the robot including its state at all times, what it sees at all times, and all the commands given to it during the execution of the task.

Demonstrations are mainly gathered from humans doing the task or controlling a robot to do the task. Even just video of a human doing a task can be used in training VLA models but it is fairly difficult to solve joint positions from video. An easier way is for a human to control a robot with teleoperation. This is when a robot is controlled by translating inputs from a keyboard, controller, or XR-device into robot movements. For humanoid robots, out of these options, the most practical one is using an XR-device. This way human movements can be fairly effortlessly retargeted directly to the robot. There are many programs to do this and more advanced ones like GR00T-WholeBodyControl [10] use separate foundation models which handle locomotion and keeping the robot balanced [11].

It is however not necessary for everyone who wishes to train a VLA model to gather their own multimodal datasets as many datasets are released online to the public. These often include expansive documentation about the used robots, setups, environments, tasks, and crucially the format of the dataset.

Figure 2 shows the methodology of this thesis. To analyze multiple dataset formats, one dataset is generated through teleoperation of a simulated robot with `xr_teleoperate`, another dataset recording program, TWIST2, is evaluated, and two online dataset

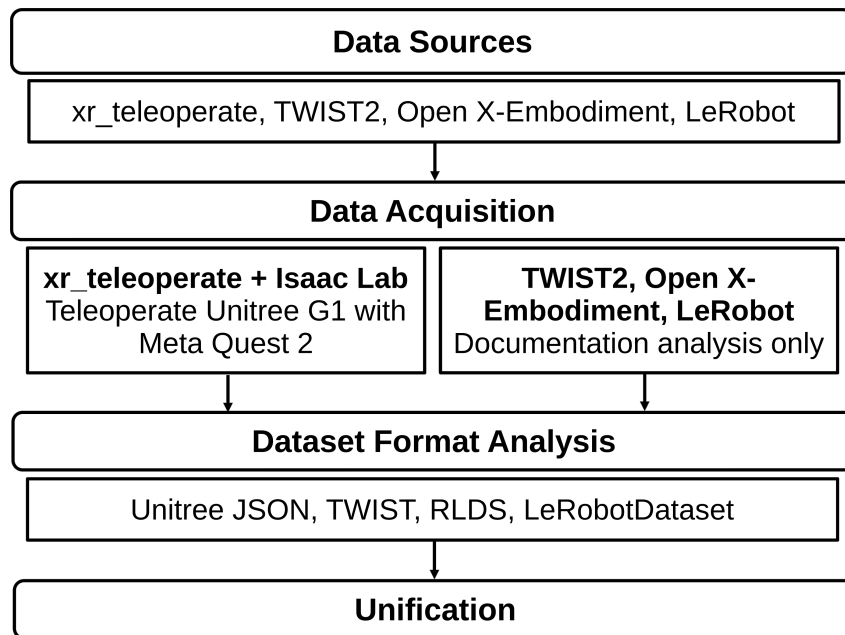


Figure 2: Methodology flowchart. Data from multiple sources, including local and online, is gathered and analyzed, followed by a discussion on unification.

sources, Open X-Embodiment and LeRobot, are examined. Analysis is followed by format comparison and discussion on translating between formats.

3.1 Data Sources

To produce an example dataset for analysis, the `xr_teleoperate` repository was used. It includes all the software needed to teleoperate a Unitree humanoid robot using an XR-device. It supports the Unitree G1, H1, and a small list of different grippers and dexterous hands. Apple Vision Pro, Pico 4 or Meta Quest 3 can be used as input devices. In addition to being able to teleoperate a real robot, a simulated robot can also be used. This particular teleoperation software stack can only be used for teleoperation of the hands and arms while locomotion is handled by the robot’s own motion controller using the joysticks of the XR-device or an external controller. [12]

There are multiple programs in the repository. Teleimager streams camera data from the robot to the XR-device. Vuer captures movement data of the head of the teleoperator from the XR-device headset and hands from either the controllers or using hand tracking. A program controls the robot’s movements based on the captured XR-device movement

data and given robot configuration. Another program records all captured data. All programs are controlled by a single interface program.

The simulator that `xr_teleoperate` connects to is `unitree_sim_isaac`lab. The simulator is made to simulate Unitree robots and is built on Isaac Lab. It includes configurations for the Unitree G1, H1, and different actuators. It also comes ready with various tasks and environments. The simulator is capable of data collection, data playback, data generation, and model validation. [13]

The second source used for dataset format analysis is TWIST2, which is a humanoid robot teleoperation data collection framework. It is built to be portable and holistic, allowing data collection of whole body teleoperation using an XR-device. The framework not only collects movements in space, but utilizes two motion trackers on the ankles of the teleoperator to capture complex whole body movements. TWIST2, however, only supports a single setup and does not offer data collection from a simulated robot, which is why it was not used in this work to produce an example dataset. Support is limited to the Pico 4 Ultra XR-device and a Unitree G1 robot with 29 DoF equipped with two Dex3-1 dexterous hands with an additional requirement of a custom add-on neck. [14]

The third source used for dataset format analysis is the Open X-Embodiment dataset, a collection of 72 existing datasets. The dataset collection includes datasets of variable tasks collected with many different robot configurations in many different environments. It does not, however, include any data collected on humanoid robots. Still, most datasets are recorded on single or dual-arm robots and as such can be used for training humanoid robots to manipulate objects with their arms. The format used by the dataset is standardized but because of varying configurations, not all datasets are quite the same. An example dataset, `ucsd_kitchens` [15], was used to help with the analysis of the format. [16]

The last analyzed dataset source is LeRobot. It is a PyTorch-based library for use in robotics. It includes datasets, pretrained models, tools for training models, tools to

evaluate models, tools to collect training data, and tools to control robots. LeRobot is developed and maintained by Hugging Face and integrates with their platform for dataset and model sharing. The library is very flexible and supports a number of robots, AI policies, and setups with the ability to add anything that is not already provided in the library. The caveat of LeRobot being a library is that end-to-end programs must be built on top of it for usage in the real world. Still, the library comes with a standardized dataset format used by numerous datasets published on Hugging Face Hub. Some programs built on LeRobot are also available online, such as `unitree_lerobot` [17] which, among other things, allows for translation of data gathered with `xr_teleoperate` into the format used by LeRobot. [18], [19]

Some other projects are also worth mentioning though they were not analyzed thoroughly for this work. XRoboToolkit is a framework for robot teleoperation developed by PICO [20]. In their work, the tool was found to be suitable for collecting demonstration data for VLA model training but the framework does not come with ready-made programs for teleoperation data collection. It also does not enforce a dataset format and because of that could not be analyzed for this work. One program built on XRoboToolkit is TWIST2 [14]. Another program worth mentioning is GR00T WholeBodyControl which enables data recording of whole-body teleoperation of humanoid robots in the real world or in simulation [10]. However, due to issues described later, this program was not analyzed. The last project considered but found unsuitable for analysis is RoboVerse, which is a platform that unifies multiple simulators to be accessible through a single interface [21]. It, however, does not include a teleoperation stack nor does it define a dataset format, though a program could be built on top of RoboVerse for data collection with teleoperation.

3.2 Simulation Setup

The simulation and teleoperation of the robot were done on a mid-range home computer and a consumer Meta Quest 2 XR-device. This computer meets or exceeds the specifications required by `xr_teleoperate` and `unitree_sim_isaac` other than that it has

less VRAM than what is recommended by the simulator. However, since a home computer was used some of the encountered issues could have potentially been avoided by using a machine more widely used in academic and enterprise environments. The XR-device was not listed as compatible but this only caused tracking inaccuracy which did not affect the results of this work.

Xr_teleoperate and unitree_sim_isaacclab were installed by following the install instructions in their respective documentation. Setup and configuration were left at their default and example values to avoid complications. This means that the task given to be demonstrated on the recording was set as picking up a cylinder and placing it into a basket. The simulated environment for this task is a desk inside a simple warehouse.

The chosen robot configuration is a Unitree G1 with 29 DoF [22] paired with Dex3-1 dexterous hands [23]. Other configurations are available but this is the default and example configuration given in the documentation for xr_teleoperate. The cameras were also left at their default configuration which in the case of the simulated robot includes one head camera (Realsense D435i built-in on G1) and one camera on each wrist (Realsense D405 recommended). The cameras recommended by the documentation for real world deployment are the built-in Realsense D435i depth camera or an externally mounted stereo camera for the head, and Realsense D405 depth cameras for the wrists. Ready-to-print 3D models for the mounts of the cameras are available in the xr_teleoperate repository.

This setup was used to produce an example dataset that has one episode of data. Datasets meant for actually training VLA models have numerous episodes with the possibility of even having multiple configurations and tasks. For the purposes of analyzing the format of the dataset, this is unnecessary and one episode is enough.

To produce a dataset, after setup, the simulator and teleoperation program are started. The simulation and data transfer need to be started, after which video from the camera of the robot will stream to the display of the XR-device and movements of the XR-device will

cause the robot to move. In the simulator, the task is already set up in front of the robot. Enabling recording will start saving all the movements of the teleoperator and robot. The teleoperator then needs to complete the task, after which the recording can be stopped. To record multiple episodes, the process can be repeated rapidly by resetting the task between recordings. The process is similar on a real robot, though the robot, environment, and task need to be set up and reset manually.

3.3 Challenges

Attempts were made to install GR00T-WholeBodyControl for recording a sample dataset of teleoperation data but these attempts were unsuccessful. Many different configurations were tried, and all had issues. The documentation of the software lists the Ubuntu 22.04 Linux distribution as the recommended operating system but installation was first tried on Windows 11 using Windows Subsystem for Linux (WSL). There were a few issues with this setup but, as an example, a custom kernel needs to be compiled to be able to pass inputs into the simulator to control the robot [24]. There were also issues with getting GPU hardware acceleration working inside the container running in the WSL virtual machine, and it was decided to change the operating system of the computer to Ubuntu 22.04. However, this did not solve all issues as the simulator did not run properly with the used configuration and it was ultimately decided not to use GR00T-WholeBodyControl at all. The problem is most likely with the hardware configuration on the used computer, showing that software for recording teleoperation data is still in early stages of development and might not work on all hardware yet.

Many issues were also found while installing `xr_teleoperate` and `unitree_sim_isaacgym` as some of the documentation was incomplete and some dependencies had wrong or incompatible versions. In most cases, these are fairly trivial to fix by applying fixes from the issues section of the repository but it is still worth noting that depending on the point in time when the installation instructions are followed, manual fixes may need to be applied. As an example, `Vuer` is a piece of software in the repository that streams video from the robot to the VR headset. Video from the eyes of

the robot should stream on port 60001. However, with the used default setup, port 60001 shows the camera feeds of the hands of the robot. This makes teleoperation fairly difficult as the operator cannot see from the perspective of the robot. This can be fixed by changing the video configuration of the teleoperation program so that the head camera is mono instead of stereo, which is the default. The cause of this issue was found to be that the default configuration of Vuer assumes the use of a stereo camera for the head while the simulator only uses a mono camera and the documentation that guides the reconfiguration of Vuer for the simulator did not mention this at the time of installation.

4 Dataset Format Analysis

The basic format of all the examined VLA training data sources is the same. A single dataset can include multiple tasks and environments and even robot configurations. However, a single chunk of a dataset or the simplest possible dataset consists of demonstrations recorded on one robot configuration in one environment doing one task. Each dataset is made up of individual episodes where each episode stores all the recorded data for one instance of completing the given task. An episode consists of one or more trajectories depending on how many movements the task requires. In most of the examined dataset formats, each episode is stored in its own folder or file, though more compressed formats like LeRobotDataset V3.0 can store multiple episodes in one file. The data captured in the datasets is discrete, meaning, alongside metadata, each episode is made up of individual steps or frames. The rate at which data is captured is left up to the user but a very typical framerate is 30 frames per second. Each frame stores information captured during that point in time including the state, action, and other possible observations like data from the cameras. The details of what this data includes, what other data is recorded, and how it is all stored vary greatly across all the examined dataset formats.

4.1 Unitree JSON

Detailed documentation for the Unitree JSON format [17] used by `xr_teleoperate` is not available, so reverse engineering was used on the produced example dataset to understand the format. The basic structure is the same as other examined formats but since it is made for specific Unitree robot configurations, it is not made to be as flexible as some other formats. Episodes are stored in their own folders on disk and in each episode, there is one data file and folders for audio, color camera data, and depth camera data. The data file is stored in JSON format while color camera data is stored as JPEG images. In this format, unpopulated fields in the data are not removed but instead simply left blank. Metadata about the episode, including camera resolutions and task

description, is stored in the header of the data file. The rest of the data file is a list of all the frames where each frame lists all the data captured at that specific point in time. Each frame lists an index number, color camera image file name, depth camera image file name, states, actions, tactile sensor states, audios, and sim state. The state and action both list the joint angles, joint velocities, and torques for both arms, both end-effectors, and the body. Data for each of the state and action fields is saved as a tensor. The sim state lists information about the simulation like the name of the scene and robot and object root poses.

The example dataset does not fill all the fields available in the dataset format since the configuration used for recording is for dual-arm object manipulation. The configuration does not include depth cameras, microphones, or tactile sensors, so the fields for them are left blank. It also does not include locomotion or movement of the body of the robot nor does it provide the possibility to record the velocities or torques of the joints, meaning all those fields are left blank as well. The configuration included three cameras so in each frame the color camera field lists the file names of the images from all of the cameras. State and action for the configuration are listed in a specific order. For both arms, the order of joint angle values is shoulder pitch, roll, and yaw, elbow, and wrist roll, pitch, and yaw. For the left end-effector, the order is three values for thumb joints, two for middle finger joints, and two for index finger joints, while for the right end-effector the middle finger and index finger are swapped. Order of joints for all configurations is determined by and readable from the robot configuration code of the program.

4.2 TWIST

TWIST2 and its predecessor TWIST use the TWIST dataset format. The format is standardized for policy learning on the specific Unitree G1 configuration of TWIST2. The basics of the format are the same as the other analyzed formats. The format is most similar to Unitree JSON with data being organized on disk in the same way. Unlike with Unitree JSON, TWIST only supports color camera data. Metadata and movement data are saved in a JSON format similar to that of Unitree JSON with minor differences in the

available fields. Each frame of data includes an index number, filename of the image captured by the camera for that frame, and the state and action of the body, hands, and neck. The order of movement data in the 35-dimensional body state and action tensors is the following: base x and y velocities, base z coordinate, base roll and pitch, base yaw velocity, joint positions of the body. The state and action of the hands and the neck include just the joint positions and are represented by 7-dimensional and 2-dimensional tensors respectively. This means data in TWIST format is in both joint space and task space since task space data is needed to keep track of the position of the robot in relation to the world around it. No example dataset was used in the analysis of this format as the format is designed to not be flexible and is documented clearly. [14]

4.3 Reinforcement Learning Datasets

The format used by episodes in Open X-Embodiment is the Reinforcement Learning Datasets (RLDS) episode format. Each episode contains steps and metadata but most of what is stored in those steps and metadata is left up to the user generating the dataset. The only mandatory fields in the dataset are the self-descriptive `is_first` and `is_last` boolean fields in each step. Optional fields in each step include the observation, action, and reward. The observation includes everything that the robot can observe at a given point in time including the state and camera data. Episodes in RLDS format are stored as dictionaries where each episode is a readable data object. Numerical data like state and action are stored as tensors that have as many dimensions as needed for the robot configuration. [25]

An example of the RLDS format's style in the Open X-Embodiment dataset collection is the UCSD Kitchen dataset [15]. The collection is very varied but this dataset was chosen for its combination of small size and data collection method of VR-based human teleoperation which is the same as the other examined dataset sources. The robot used is only a single 6-DoF xArm robot arm but for format analysis purposes this is not an issue. This dataset also documents its format clearly, which is good for format analysis. In the UCSD Kitchen dataset, state is saved as a 21-dimensional tensor that includes

joint angles, joint velocity, and joint torque. Action is stored as an 8-dimensional tensor that includes end-effector position and orientation, if the gripper is open or not, and an episode termination action. This means that while state is stored in joint space, action is stored in task space.

State and action representation is not defined by the format so other datasets in the collection might store them differently from the chosen example. The state and action representation is also very different from all the other analyzed dataset formats since they store different parameters. Another big difference from other analyzed formats is that RLDS includes an optional field for a reward because it is originally made for reinforcement learning applications. The other formats also do not serialize camera data into the same data stream as the other information and instead opt to store it in separate files.

4.4 LeRobotDataset

The format used by LeRobot is called LeRobotDataset, which is a standardized format made for storing robot learning data. Like RLDS, LeRobotDataset is made to be flexible and can accommodate many different configurations and use cases. Files in this format are structured so that metadata including configurations and task descriptions are stored in their own files, and movement data and camera data are stored in their own files. The newer v3.0 differs from older versions in that, instead of one episode per file, data from multiple episodes is stored in standard-sized files to save disk space.

Movement data is stored in Apache parquet files and camera data is stored in MP4 files.

Movement data is stored in discrete frames which contain key-value pairs for all recorded parameters. Crucially, this includes index numbers and timestamps to find the corresponding frames in the camera data video file. The recorded parameters also include the state and action which are both tensors with as many dimensions as is needed for the used robot configuration. [18], [19]

An example of this dataset format is the custom example dataset recorded with

`xr_teleoperate` translated into LeRobot format with the conversion utility in the `unitree_lerobot` repository [17]. In this example dataset, the state and action are both 29-dimensional tensors that list joint angles for both arms and end-effectors in the same order they appear in the original dataset. The translation omits the `sim state` field present in the original dataset. The translation process also takes all the individual camera images and renders them into a single video for each camera.

4.5 Unification

Translating data from one format to another is possible as demonstrated by data in the Unitree JSON format being able to be translated into the LeRobotDataset format using `unitree_lerobot`. Translating data from one format to another at a minimum only requires reformatting the way the data is written and in some cases rendering images to video or vice versa. Translating movement data stored in task space to its joint space representation and vice versa is also possible with forward and inverse kinematics enabling translation between formats with different movement data representations. The better documentation a dataset format has, the easier it will be to design a simple translator like this.

An issue arises when two formats store different data. `Unitree_lerobot` includes a utility to translate from Unitree JSON to LeRobotDataset but does not include a utility for translation in the other direction. While this might partially be due to a lack of reason to store data in Unitree JSON format as `xr_teleoperate` does not include tools for policy training like LeRobot, another reason is that datasets in Unitree JSON include more data than those same datasets translated into LeRobotDataset. This is because the `sim state` field of the Unitree JSON format is dropped during the translation process though the LeRobotDataset format could be made to include this data. However, this is not the case for all dataset formats. Some dataset formats store more data than others and translation from a format which stores less data to a format which stores more data will either result in an invalid dataset or filling missing data fields with null or made-up data. Similarly, as an example, TWIST stores movement data from its custom neck add-on.

Translation of data from the TWIST format into another format could mean having to lose this data if the format being translated into cannot accommodate this data.

One reason to translate data from one format to another is to use that data with different tools. These include, for example, policy training tools and simulator environments both of which need definitions of the robots. A teleoperation data recording program will have the configuration of the used robot but after translating the data into another format, the program using that format might not have the same configuration available. In cases like this, the configurations must be made to allow using the translated data with the program. As an example, the utility to translate datasets from Unitree JSON format to LeRobotDataset format only includes the option for one simulator configuration, which is the Unitree G1 with Dex1 grippers. By editing the constants.py file in the same folder as the conversion script, it is possible to add other simulator configurations like the one used in producing the example dataset, the Unitree G1 with Dex3-1 dexterous hands. While this simple fix does make the utility convert datasets successfully, these datasets may not successfully evaluate for deployment and further steps are required to evaluate translation success.

There are also cases when movement data from one robot should be applied to another robot. For example, dual-arm movement data from a robot in the Open X-Embodiment dataset might be needed for training of a Unitree G1. Retargeting of motion data like this is possible and is practically the same as what is needed to apply movement data from an XR-device onto a robot. In this case too, it is important to make sure all configurations are correct so that movement from, for example, an elbow is not applied onto a shoulder.

5 Conclusion

In this thesis, sources for data to train VLA models to control humanoid robots were presented and the dataset formats used by these sources were analyzed. Training data consists of camera data and robot movement data paired with a task prompt. This data can either be manually recorded or data recorded by others may be found online. Data sources presented in this thesis include teleoperation software `xr_teleoperate` and `TWIST2`, the public dataset collection `Open X-Embodiment`, and the software library `LeRobot`. Analysis of the four formats used by these different sources found the formats to all be very similar on a high level but with some differences in the details. The episodic nature of data and the recording of camera data, state, and action is the same in all dataset formats, while there are differences in the containers used to store the data and the exact details of what motion data is recorded and in what representation it is recorded. By analyzing even more formats, more variables could have been found.

Some dataset formats, like `Unitree JSON` and `TWIST`, were found to be less flexible, while other formats, like `RLDS` and `LeRobotDataset`, were found to be very flexible. Less flexible formats are usually made for very specific use cases, while flexible dataset formats are made specifically to be general formats for use with as many configurations as possible. Of the examined formats, `LeRobotDataset` should be considered when choosing a general dataset format due to its good flexibility, space efficiency, and integration with `Hugging Face Hub`.

Libraries like `LeRobot` are already able to support most robot configurations so there is little reason to make a more general format for even wider use. Generalizing the library and its format to, for example, accommodate autonomous vehicle driving would require all the tools made with the library to also support the new addition. This makes little sense as tools to train VLA models do not have many similarities with tools to train autonomous vehicle driving policies. In theory, just the format could still support this addition while the tools remain separate but this leaves no reason for the format to

accommodate both use cases.

Similarly, while many tools could use general formats, many niche use cases, like TWIST, which is made for one configuration, opt to use their own formats instead because it is easier and allows for quicker development. When choosing between a general or custom format, it is worth thinking about whether or not the general format needs to be made to support every niche use case. If only one project needs a feature from the format, it might not be feasible for a general format library to require all tools to support this feature.

The robotics and artificial intelligence fields are evolving quickly and the needs of robot control policy training dataset formats are constantly changing with them. A format which supports every need now might not yet support anything that is wanted in the next generation of robots and AI. As such, new formats come and go and sticking to one for too long could hinder development speeds.

The speed of development was also found to be evident in the state of the used software. The issues encountered during the installation process of software used to gather the example dataset show that this software is not made for everyday use. The hardware and software support of this software is limited to specific configurations and trying to use anything outside of those configurations will result in nonoperational software. As these processes leave laboratories and enter industry and eventually home consumer use, the software used in these processes will mature and support more configurations.

5.1 Further Work

With the knowledge presented in this thesis, it is possible to record a full dataset of robot movement data. Multiple data recording programs can be used to record datasets in different formats. The next logical step is to take this data and train or fine-tune a VLA model on it. With the tools available, this should be fairly straightforward, though for successful results, the quality of data in the dataset should be examined. With data in different formats, different VLA models could be trained and their performance could be compared. One dataset could even be translated into multiple formats and different

training programs could be used to train multiple instances of the same VLA model. This would allow for analyzing the performance of different dataset formats in VLA model training.

Further research efforts in the field could be used to examine how VLA performance is affected by whether the movement data in the training datasets is represented in joint space or task space. Most VLA models now are trained on joint space data since the lowest-level commands given to the joints of the robot have to be in joint space. However, since joint space representations can be calculated from task space representations of joint positions, the dataset on which models are trained could include only task space representations of movement data. For example, a fully task space dataset that included only the positions of end-effectors and the body in relation to the world could be used fairly interchangeably for multiple similar robots by calculating the joint space data while using the dataset.

Another similar problem worth researching is seeing how VLA performance is affected by whether the VLA model does its thinking with joint space data or task space data. Currently, models output joint space low-level robot actions since that is what the robot hardware requires. If the robot instead output robot actions in task space, the joint space representation could be calculated on the fly. Changing to task space representations inside the model and in the datasets could allow for better generalization between embodiments, though it would introduce higher computation requirements and complexity.

References

- [1] A. Brohan, N. Brown, J. Carbajal, *et al.*, *Rt-2: Vision-language-action models transfer web knowledge to robotic control*, 2023. arXiv: 2307.15818 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2307.15818>.
- [2] F. Capuano, C. Pascal, A. Zouitine, T. Wolf, and M. Aractingi, *Robot learning: A tutorial*, 2026.
- [3] Physical Intelligence, A. Amin, R. Aniceto, *et al.*, $\pi_{0.6}^*$: *A vla that learns from experience*, 2025. arXiv: 2511.14759 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2511.14759>.
- [4] Physical Intelligence, B. Ai, A. Amin, *et al.*, $\pi_{0.7}$: *A steerable generalist robotic foundation model with emergent capabilities*, 2026. arXiv: 2604.15483 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2604.15483>.
- [5] S. Ye, Y. Ge, K. Zheng, *et al.*, *World action models are zero-shot policies*, 2026. arXiv: 2602.15922 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2602.15922>.
- [6] Q.-C. Pham and F. Suárez-Ruiz, *Introduction to Open-Source Robotics*. 2021. [Online]. Available: <https://www.osrobotics.org/osr/> (visited on 05/09/2026).
- [7] Engineering Simplified, *Robotics 101*. [Online]. Available: <https://www.youtube.com/playlist?list=PL1YrgW7R0FofBqPGiWAmTqIwDc5SrZrA> (visited on 03/17/2026).
- [8] K. Hauser, *Robotic Systems (draft)*. University of Illinois at Urbana-Champaign, 2023. [Online]. Available: <https://motion.cs.illinois.edu/RoboticSystems/> (visited on 02/24/2026).
- [9] E. Aljalbout, “Learning state and action spaces for robot learning,” Dissertation, Technical University of Munich, Dec. 2023. [Online]. Available: https://mediatum.ub.tum.de/doc/1729775/9t2zk02kqws1azj7t7iwqid90.Ealjalbout_dissertation2024.pdf.
- [10] NVlabs/*GR00t-WholeBodyControl*, Feb. 2026. [Online]. Available: <https://github.com/NVlabs/GR00T-WholeBodyControl> (visited on 02/09/2026).
- [11] Z. Luo, Y. Yuan, T. Wang, *et al.*, *Sonic: Supersizing motion tracking for natural humanoid whole-body control*, 2025. arXiv: 2511.07820 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2511.07820v2>.
- [12] Unitree Robotics, *XR-Teleoperate: An open-source teleoperation framework and data collection toolkit for embodied intelligence*, https://github.com/unitreerobotics/xr_teleoperate, Accessed: 2026-02, 2024.
- [13] *Unitreerobotics/unitree_sim_isaac*lab, Feb. 2026. [Online]. Available: https://github.com/unitreerobotics/unitree_sim_isaac (visited on 02/10/2026).

- [14] Y. Ze, S. Zhao, W. Wang, et al., *Twist2: Scalable, portable, and holistic humanoid data collection system*, 2025. arXiv: 2511.02832 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2511.02832>.
- [15] G. Yan, K. Wu, and X. Wang, “ucsd kitchens Dataset,” Aug. 2023.
- [16] E. Collaboration, A. O’Neill, A. Rehman, et al., *Open x-embodiment: Robotic learning datasets and rt-x models*, 2025. arXiv: 2310.08864 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2310.08864v9>.
- [17] *Unitreerobotics/unitree_lerobot*, Feb. 2026. [Online]. Available: https://github.com/unitreerobotics/unitree_lerobot (visited on 02/10/2026).
- [18] R. Cadene, S. Alibert, A. Soare, et al., *Lerobot: State-of-the-art machine learning for real-world robotics in pytorch*, <https://github.com/huggingface/lerobot>, 2024.
- [19] R. Cadene, S. Alibert, F. Capuano, et al., “Lerobot: An open-source library for end-to-end robot learning,” in *The Fourteenth International Conference on Learning Representations*, 2026. [Online]. Available: <https://arxiv.org/abs/2602.22818>.
- [20] Z. Zhao, L. Yu, K. Jing, and N. Yang, *Xrobotoolkit: A cross-platform framework for robot teleoperation*, 2025. arXiv: 2508.00097 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2508.00097>.
- [21] H. Geng, F. Wang, S. Wei, et al., *Roboverse: Towards a unified platform, dataset and benchmark for scalable and generalizable robot learning*, 2025. arXiv: 2504.18904 [cs.R0]. [Online]. Available: <https://arxiv.org/abs/2504.18904>.
- [22] Unitree Robotics. “Humanoid robot g1_humanoid robot functions_humanoid robot price | unitree robotics.” (2024), [Online]. Available: <https://www.unitree.com/g1> (visited on 04/05/2026).
- [23] Unitree Robotics. “Unitree dex3-1 power control dexterous hand, operate everything - unitree robotics.” (2024), [Online]. Available: <https://www.unitree.com/Dex3-1> (visited on 04/05/2026).
- [24] Microsoft, *Gamepad support in WSL2 (/dev/jsx) · issue #7747 · microsoft/WSL*, Publication Title: GitHub. [Online]. Available: <https://github.com/microsoft/WSL/issues/7747> (visited on 02/09/2026).
- [25] S. Ramos, S. Girgin, L. Hussenot, et al., *Rlds: An ecosystem to generate, share and use datasets in reinforcement learning*, 2021. arXiv: 2111.02767 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2111.02767>.