Evaluating Vision-Language-Action Models in Robotic Manipulation: Performance, Implementation, and Comparison with Deterministic Systems

Jake Kemple

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2025

Committee:

Min Chen

Clark Olson

Yang Peng

Program Authorized to Offer Degree:

Computer Science and Software Engineering

© Copyright 2025
Jake Kemple

University of Washington

Abstract

Evaluating Vision-Language-Action Models in Robotic Manipulation:
Performance, Implementation, and Comparison with Deterministic Systems

Jake Kemple

Chair of the Supervisory Committee:
Min Chen
Department of Computing & Software Systems

Robotic manipulation systems typically use deterministic policies for perception, decision-making, and task planning, which achieve millimeter-level precision but require extensive specialized development and cannot easily generalize to new tasks. Emerging vision-language-action (VLA) foundation models promise to reduce this specialized effort and inflexibility through learned multimodal reasoning. However, their practicality in the real world and associated development costs remain largely unknown.

This thesis presents a real-world comparison of a strong open-source VLA foundation model (OpenVLA-7B) against a fine-tuned deterministic control system. Both systems are evaluated on identical hardware consisting of a WidowX 250 6-DoF robotic arm, an Intel RealSense D415 camera, an NVIDIA Jetson AGX Orin edge computer, and the ROS 2 (Robot Operating System 2) framework. Each system repeatedly executes a pick-and-place robotic task under randomized initial conditions. Performance is measured using goal-oriented, object-centric metrics of accuracy, repeatability, and cycle time, adapted from ISO 9283 standards. Additionally, a qualitative analysis examines the installation effort and configuration challenges associated with each system.

The primary contributions of this research include: (i) a comparative evaluation of performance and setup complexity between robotic systems utilizing a VLA-based control policy and conventional deterministic control logic; (ii) documentation of hardware, software, and configuration challenges encountered during VLA system implementation; and (iii) qualitative insights from real-world deployment, emphasizing usability and adaptability.

Results indicate that current VLA foundation models underperform compared to deterministic control systems in terms of accuracy, repeatability, and cycle time, limiting their immediate viability for production-level robotic tasks. However, the inherent flexibility of VLA models suggests strong potential as future replacements for deterministic approaches, contingent upon improvements through fine-tuning, future optimizations, enhanced integration frameworks, and better overall performance metrics. These findings offer practical insights and set realistic expectations for developers considering transitioning from deterministic robotics systems to VLA-based implementations.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my committee chair, Professor Min Chen, for her guidance and effort, as well as my committee members, Professors Clark Olson and Yang Peng, for their time and input. I also thank my parents, Lonnie and Joey Kemple, and Trisha's parents, Tushar Dani and Monika Deo, for their support throughout this process. Finally, I want to thank my partner, Trisha, for her devoted support. This effort would not have been possible without her.

Chapter 1

Introduction

1.1 Background and Motivation

Historically, robotic manipulation system software development has required significant engineering effort, both in design and implementation [5]. Developing a consistent and reliable system, even for narrow-scoped and simple tasks in real-world robotics, has proven very challenging with a high technical barrier to entry [6]. Consider the example of a robotic arm on a factory conveyor performing a single task, such as placing an object in a specified position required for processing at a later stage. This system must execute precise movements in three-dimensional space with millimeter accuracy, account for environmental conditions such as lighting, part variation, and conveyor speed, calibrate coordinate frames and transforms, and accurately process perception data from sensors and camera vision systems. Once developed to a viable degree, any adaptation, including performing a new task, incurs non-trivial additional costs [6]. Given this complexity, the potential for AI integration to reduce development effort by abstracting implementation details, or even generalizing the system to handle numerous tasks from its initial deployment, is highly attractive and impactful.

In recent years, the introduction of large language models and transformer-based architectures [2], along with multimodal input capabilities [3], have driven breakthroughs in robotics and AI research, particularly related to task generalization [7]. Specifically, VLA models, large language models extended to process visual input and output actions, represent the state-of-the-art architecture designed to address task generalization in robotics [4]. Legacy conventional robotics systems, previously limited to deterministic control logic for perception, decision-making, and task planning or specialized application-specific models, now have the potential to replace this

logic with pretrained foundation VLA models serving as a control policy with generalization capabilities [4]. Although recent research emphasizes strong advances in cognition and perception for task generalization [4], comparative analyses between VLA models and conventional robotics architectures on common real-world tasks remain limited.

1.2 Problem Statement

This research provides a detailed analysis of the setup, deployment process, and practical considerations for a real-world robotic manipulation system utilizing the notable open-source 7-billion-parameter OpenVLA foundation model as its control policy [4]. Additionally, the study compares this implementation to a conventional robotic setup using deterministic policies without AI, focusing on performance on a defined pick-and-place manipulation task. To date, most academic research on foundation models in robotics has focused on model performance improvements using standardized simulation benchmarks, such as the LIBERO benchmark, which assesses the performance of the VLA model itself [8], as opposed to assessing precise action success or real-world performance [9]. Moreover, existing evaluations typically compare VLA models to previous iterations or competing models. This narrow focus on benchmark-driven comparisons between models leaves uncertainty regarding their real-world applicability and implementation requirements [10].

Focusing specifically on VLA models within robotics, limited research directly compares their real-world effectiveness to conventional deterministic control logic or specialized application-specific robotics solutions. While certain practical robotic applications require extremely high precision and accuracy, this research specifically focuses on goal-oriented scenarios in which VLA-based policies may be viable. This thesis aims to provide stakeholders with actionable insights into whether adopting a strong open-source VLA foundation model is practical and beneficial for their intended applications. Despite promising demonstrations of generalization capabilities across various robotic platforms in recent OpenVLA research [4], the suitability of deploying these advanced VLA foundation models in realistic production environments-considering factors such as cost, effort, performance, and practical constraints-remains insufficiently explored. A pick-and-place task was chosen for evaluation in this study to reflect a typical task in practical robotic applications. The task definition and evaluation criteria are detailed further in Section 4.2.

1.3 Research Questions and Hypotheses

This thesis focuses on three central research questions designed to provide insight into the performance and implementation implications of VLA models in robotic manipulation systems. These questions specifically address performance metrics, implementation benefits, and integration challenges. Accompanying each research question are testable hypotheses that outline the expectations and potential outcomes this research aims to evaluate.

Research Question 1: "How do robotic systems utilizing a VLA control policy compare to robotic systems utilizing deterministic control logic in terms of task execution speed, accuracy, and repeatability for common manipulation tasks?"

- **Hypothesis 1:** Robotic systems utilizing a VLA control policy will achieve comparable accuracy, repeatability, and cycle time to robotic systems utilizing deterministic control logic in tasks involving varied object arrangements.

This question and corresponding hypothesis challenges the assumption that deterministic robotic systems inherently outperform AI-based systems in speed and precision with significance. Additionally, it explores whether the generalized capabilities of VLA models demonstrate their superiority despite computational overhead, especially within scenarios involving complexity or variability.

Research Question 2: "What implementation advantages, if any, do VLA control policies offer compared to conventional deterministic control logic?"

- **Hypothesis 2:** VLA models improve resilience by allowing the robotic system to effectively recover from unforeseen circumstances or failures encountered during execution.

Research Question 3: "What specific technical and operational challenges arise when integrating VLA models into robotic manipulation systems?"

- **Hypothesis 3:** Integration of VLA models introduces technical challenges, including increased inference latency, calibration complexity, and heightened computational resource requirements.

The final question and hypothesis focus on integration challenges that arise in VLA based robotic systems. By examining these challenges, the thesis aims to offer insight into the technical barriers that future engineers and researchers can expect when adopting a VLA model based

system, and build the currently limited set of documentation regarding such models in actual implementation [11].

These research questions and hypotheses provide guidance for the empirical analyses detailed in subsequent chapters of this thesis, ensuring evaluation and meaningful contribution to the new body of knowledge in VLA Models applied in real-world scenarios.

1.4 Scope

This research is bounded by the distinct, chosen hardware and software to sufficiently run robotic manipulation tasks in such a way that both conventional software control methods and VLA-based model control could be implemented effectively. For hardware, this research is limited to robotic manipulation tasks on small wooden blocks, executed by a WidowX 250s robotic arm in a controlled indoor environment using an Intel RealSense D415 camera for visual feedback. All computation is done locally on a Nvidia Jetson AGX Orin edge computing platform, including an aftermarket 1TB NVMe SSD. Regarding software, the Orin is the running NVIDIA JetPack 6 environment (based on Ubuntu Linux), which includes optimized libraries and drivers specifically developed for Jetson hardware.

The research evaluates two primary system types: a deterministic control system using the Interbotix open-source perception packages for manipulation, and a VLA control policy-based system using the OpenVLA 7-billion-parameter model. Both system types exclusively utilize Interbotix X-Series arm control open-source packages for abstracting lower-level motor control of the WidowX in order to apply position, rotation, and gripper state movements. One specific task is experimentally evaluated on the blocks, pick-and-place, which involves object detection and movement.

1.5 Objectives

This research focuses on 3 primary objectives: (1) quantitatively compare performance metrics (accuracy, speed, and repeatability) between deterministic and VLA-based robotic manipulation systems, (2) assess and identify any other implementation advantages VLA models offer compared to conventional deterministic robotic control methods, and (3) provide recommendations

and guidance for robotics engineers considering adoption of VLA-based robotic control in practical manipulation scenarios.

1.6 Thesis Contributions

Due to the recent introduction of VLA models as a type of foundation model, there is little available research in understanding their performance, advantages, and implementation specifics when applied in real-world scenarios. This thesis addresses those gaps with the following contributions:

- **Empirical Comparative Analysis:** This thesis provides an empirical evaluation of the performance differences between VLA-based and deterministic robotic control systems, directly addressing an existing research gap.
- **Insights for Industry and Academia:** This thesis offers insights on when and how employing VLA models within robotic systems provides advantage, clarifying the conditions under which such models offer tangible benefits.
- **Practical Implementation Guidance:** This research contributes thorough guidance on setup procedures, challenges, and practical advice for deploying VLA models in real-world robotic manipulation.

Chapter 2

Background and Related Work

2.1 Deterministic Perception and Task Planning for Robot Manipulation

To effectively distinguish VLA-based systems from conventional or deterministic perception and task planning systems, it's imperative to define what "deterministic perception and task planning" means, particularly in the context of robotic manipulation. Deterministic perception and task planning refers to predictable actions toward task completion based strictly on explicitly programmed inputs, without the uncertainty of probabilistic or inference-based components. To perform actions in cartesian space, a deterministic system relies on explicitly coded instructions based on predetermined action sequences and math-based approaches [12] [13]. Deterministic decision-making methods are characterized by "user-defined trajectories", rules and behaviors of end-effectors, object detection and localization algorithms (e.g. RGB-D color segmentation, bounding boxes, both forms of conventional machine learning and computer vision) [14], and algorithmic calculation of incremental movements and rotations based on object coordinates [15].

2.1.1 Common Methods and Implementation Processes

Common deterministic robotic manipulation techniques incorporate structured programming techniques and explicit algorithmic configuration processes [15]. Typically, the approach involves clearly-defined conditional logic or math-based algorithms to guide actions in response to vision pipeline input representing the environmental state [13] [15]. This structured logic often relies on

sensory input conditions to trigger specific actions, ensuring consistent and predictable responses to expected task scenarios [15].

Various common implementation methods for manual perception, vision, and task planning include the following.

Point Cloud-Based Object Segmentation and Localization

Deterministic perception commonly employs depth sensors and structured-light cameras to produce 3D point cloud representations of objects [16]. These point clouds are explicitly processed using clustering algorithms to segment the environment into discrete objects based on spatial proximity [17]. Once segmented, the object's positions are explicitly calculated relative to a predefined reference frame, providing accurate target locations for robotic manipulation tasks.

Color-Based Object Detection and Localization

A widely-used alternative method is explicit color thresholding and contour extraction. This involves manually defining color thresholds in HSV or RGB space and applying contour detection algorithms on 2D camera images [18]. The detected contours explicitly represent the object's position and dimensions, offering a simple and efficient approach for object localization when depth data is unavailable or unreliable [18].

Explicit Cartesian Motion Planning

Deterministic task planning commonly defines explicit Cartesian goal positions or incremental positional adjustments (Δx, Δy, Δz) to guide the robotic arm's movement [19]. Target positions and orientations are explicitly identified based on manually segmented perception data (e.g., point cloud clusters or image-based detections). These clearly-defined Cartesian goals directly instruct the robot's motion, resulting in predictable and repeatable arm positioning [19].

Incremental Movement and Gripper Control Logic

In deterministic systems, incremental adjustments in Cartesian space are explicitly calculated to ensure precise control when approaching, grasping, and placing objects. Gripper actions (open or close) are triggered explicitly based on predefined logic conditions tied directly to object positions or task sequence steps [20] [15]. This incremental, scripted methodology provides

reliable execution of repetitive manipulation operations, although there has been research on how to better execute more complex scenarios with machine learning techniques like Deep Reinforcement Learning [20].

2.1.2 Advantages and Typical Use Cases

Historically, deterministic systems have demonstrated consistent success in controlled environments with simple task goals such as a repetitive manipulation action on precise object positions [21]. For example, consider a position on a factory conveyor where a robot arm is expected to repetitively join or assemble parts. In such a highly controlled environment, a system designed with deterministic perception and task planning logic will be precise, predictable, repeatable and affordable, all qualities deemed as strengths, additionally easing system debugging and validation [21]. As a result, deterministic perception and task planning to complete robotics tasks is most commonly used in industrial settings.

2.1.3 Limitations

In contrast to the advantages of deterministic perception and task planning, there are numerous drawbacks including required knowledge of the environment beforehand [22], poor generalization to unknowns [22], and limited robustness to clutter without more complex techniques [21]. When conditions change, such as the introduction of a previously unseen object task, robotic manipulation will have difficulty adapting [23] without additional learning through live interactions [24]. Finally, when the environmental conditions differ from assumptions even slightly, such as small positional deviation, performance can become brittle [22] [23]. Despite these limitations, deterministic systems remain common in industry settings due to the lack of alternatives without large complexity overhead [21] and deterministic sampling remains the best option for path planning and robotic manipulation [22].

2.2 VLA Foundation Models

VLA Models represent a distinct approach from conventional perception, decision making, and task planning in robotics and AI. Their ability to integrate visual inputs with advanced language reasoning allows for a more capable and flexible interaction with the physical world, positioning them as the potential foundation for next-generation robotics [4].

2.2.1 Contextualizing VLA Models within AI Architectures

To contextualize VLA models among other AI techniques, VLA models are a type of Neural Network (NN), which broadly encompass all deep learning models that involve neurons and learning edges between them via back-propagation [1]. Large Language Models (LLMs) are specialized NNs focused primarily on textual data, using attention mechanisms [2], while Multimodal LLMs expand this to include additional modalities like images or audio [3]. Vision-Language Models (VLMs) specifically integrate visual perception with language understanding [4]. VLAs further extend VLMs by changing the outputs of the model to be actionable position deltas, enabling direct translation from visual and natural language input into physical actions, thus uniquely supporting robotic applications [4]. See Figure 2.1 for a visualization of these relationships.

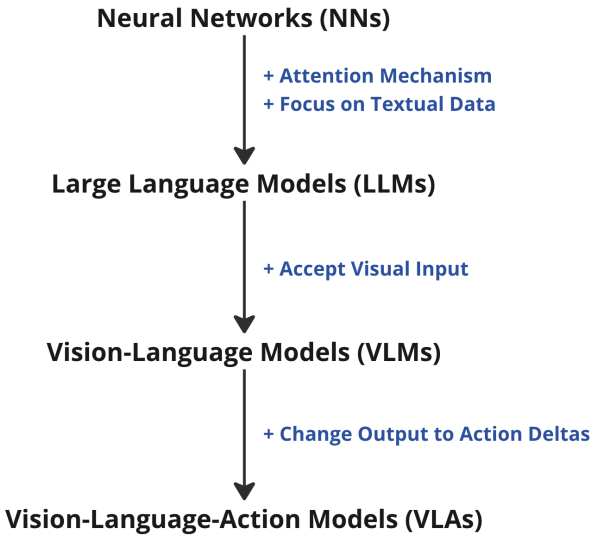

Figure 2.1: Model relationship diagram [1][2][3][4].

VLA Models represent a strong advancement in transformer-based deep learning, making them uniquely capable of strong predictive outcomes from even the most complex input data. Unlike existing language models, which rely solely on text data, VLA Models are capable of processing inputs from multiple sources, including visual data and physical sensory data [3]. Further, because of their ability to be trained on diverse inputs, VLA Models are capable of generating responses required for physical task execution from simple natural language instructions [4].

2.2.2 Pretraining, Generalization, and VLA Models in Robotics

VLA Models are pretrained on vast multimodal datasets consisting of corresponding pairs of text and images; this enables the models to learn associations between objects, actions, and descriptions [4]. VLA models, such as OpenVLA, exhibit strong generalization capabilities in zero-shot settings, effectively performing tasks without task-specific fine-tuning when the tasks fall within the scope of their pretraining data. For tasks outside this scope, fine-tuning enables adaptation to new environments and robot embodiments, enhancing performance in novel scenarios [4]. This is particularly important in the context of robotics, where conventional systems rely heavily on deterministic control algorithms, which lack the capability to handle most complex real-world tasks [24].

In contrast, VLA Models are more capable of handling ambiguities and generating appropriate actions based on high-level instructions without prior knowledge of the environment [4]. This makes them a promising candidate for a wide range of applications in real-world settings, such as object manipulation. Prominent VLA foundation models such as PaLM-E, RT-X, and OpenVLA have been developed to extend language-based reasoning into the physical world, where real-time decision-making and interaction with objects are required [4] [7] [25].

2.2.3 OpenVLA, BridgeV2 Dataset, and Implementation Resources

OpenVLA, specifically the OpenVLA-7B model, is an open-source VLA foundation model explicitly developed for robotic manipulation tasks [4]. It was pretrained on the BridgeData V2 dataset, which contains multimodal robotic demonstrations across various manipulation tasks, environments, and robot embodiments, including the WidowX 250 6-DoF arm. The BridgeData V2 dataset emphasizes diverse visual inputs and corresponding action labels, enabling OpenVLA to achieve zero-shot generalization to novel manipulation scenarios without task-specific training [26]. Additionally, the companion `bridge_robot_data` repository provides basic preprocessing scripts, fine-tuning examples, and initial deployment code. However, it offers limited guidance on detailed integration requirements, system configuration, and practical deployment considerations.

2.2.4 Potential Limitations of VLA Models

Although VLA models promise impressive generalization, their practical deployment still faces four major hurdles.

1. Computational complexity remains high: contemporary models span from the 7-billion-parameter OpenVLA to the 562-billion-parameter PaLM-E, pushing memory consumption and floating-point operations to the limits of affordable edge hardware [4] [7].

2. Inference latency is another bottleneck: on an NVIDIA Jetson AGX Orin, an un-quantised OpenVLA forward pass averages roughly 840 ms, and even INT4 quantization only reduces this to about 336 ms, far below the 20-30 Hz control rates achieved by deterministic manipulation systems [27] [28].

3. Resource constraints extend beyond pure compute: large-scale training corpora such as the Open X-Embodiment dataset contain over a million trajectories and require significant storage and memory bandwidth, posing practical challenges for data handling, streaming, and fine-tuning [25].

4. Calibration sensitivity persists because action tokens are interpreted in metric robot space; millimeter-scale camera-arm miscalibrations can cascade into centimeter-scale task errors, necessitating meticulous hand-eye calibration and, in many cases, per-task fine-tuning or task-specific models [23].

Collectively, these issues, compute load, latency, hardware appetite, and calibration overhead, form the most immediate barriers separating today's VLA research prototypes from drop-in industrial deployments.

2.3 Performance Metrics for Robotic-Arm Tasks

Robotic manipulation is conventionally evaluated using three core metrics: accuracy, repeatability, and cycle time, as defined by the ISO 9283:1998 standard [29]. These metrics are typically robot-centric, focusing on how precisely and consistently the tool-center-point reaches a commanded pose [29]. However, this thesis adapts those definitions to be object-centric, evaluating success based on the final position and orientation of the manipulated object rather than the robot's joints or end-effector. This shift aligns more directly with real-world goals, where task

success is defined by what the robot achieves in the environment, not how its body moves [30]. Compared to simulation benchmarks like LIBERO, which emphasize model generalization and internal consistency [8], object-centric measurements provide a more practical and tangible assessment of a system's effectiveness, especially for physical deployments. They better capture real-world variability, error accumulation, and task-level outcomes [30], making them a more meaningful standard for evaluating manipulation systems across policies with fundamentally different setups .

In practice, accuracy measures how closely the robot achieves the desired final placement or orientation of an object [31] [32]. For example, after rotating a block, accuracy is simply how close the block's final orientation matches the intended orientation. Repeatability evaluates whether the robot can achieve this accurate outcome consistently across multiple trials [31] [32]. Finally, cycle time measures how long it takes for the robot to complete a single manipulation task, defined clearly as the time from issuing the initial movement command to the robot completing the goal task via observation.

These adapted metrics align closely with the approach of existing robotics studies, who commonly utilize similar metrics such as average error, variability between trials, and task completion speed to evaluate whether robotic arms meet industrial or practical requirements [31]. Moreover, these metrics are straightforward and achievable with our available hardware and software: the WidowX joint encoders, the RealSense D415 camera, and standard ROS 2 logging tools [33]. In clear terms, accuracy answers "Did the robot place or orient the object correctly?", repeatability answers "Can it consistently achieve the same results?", and cycle time addresses "How quickly can the robot complete the given task?" [32].

Using these practical and accepted measures, this thesis clearly compares the deterministic and VLA-based robotic systems, ensuring the evaluation process is robust, realistic, and achievable within the constraints of this research.

2.4 Existing VLA Performance Research

Recent VLA-focused research has primarily showcased foundational models capable of impressive generalization, often evaluated through standardized benchmarks and controlled experimental setups. Notable flagship studies such as OpenVLA [4], PaLM-E [7], RT-X [25], and VIMA [34] emphasize VLA models' ability to interpret natural language commands and perform

diverse manipulation tasks without extensive task-specific fine-tuning. For instance, OpenVLA greatly improves task generalization in the real world, compared to earlier multimodal models like VIMA [34], which only operated in simulation environments. OpenVLA was shown to demonstrate strong performance in pick-and-place, object stacking, and basic tool handling scenarios [4]. Similarly, PaLM-E and RT-X underscore adaptability, demonstrating effectiveness across varied tasks and environmental conditions [7] [25].

Common performance findings from these studies show VLA models consistently excel in flexibility and adaptability, successfully managing novel tasks, objects, and instructions without explicit reprogramming [7]. These models are typically evaluated on accuracy in task completion, speed of execution, and ability to adapt to different contexts, reflecting their potential for real-world applications beyond benchmark tasks [4].

However, a clear research gap exists concerning real-world applicability. Current VLA studies typically measure performance against older or competing AI models rather than comparing directly with conventional deterministic robotics approaches. Furthermore, there is minimal existing documentation or guidance detailing challenges with practical implementation of these models into real-world robotic systems.

This thesis directly addresses these gaps in two important ways. First, it provides a concrete and practical comparison between deterministic and VLA-based robotic manipulation methods in realistic and controlled conditions, specifically evaluating accuracy, speed, and repeatability. Second, it offers critical insights into the actual implementation process, clearly outlining setup complexities and operational challenges. This research broadens the understanding of VLA model performance in realistic scenarios and serves as a practical guide for future research and practical applications of VLA-based systems.

Chapter 3

System Design and Implementation

This chapter describes the configuration and architecture of the systems being evaluated. The complete architecture, including both deterministic and VLA-based methods integrated through ROS 2 middleware, is depicted in Figure 3.1. Subsequent sections describe the details necessary to execute the architecture described.

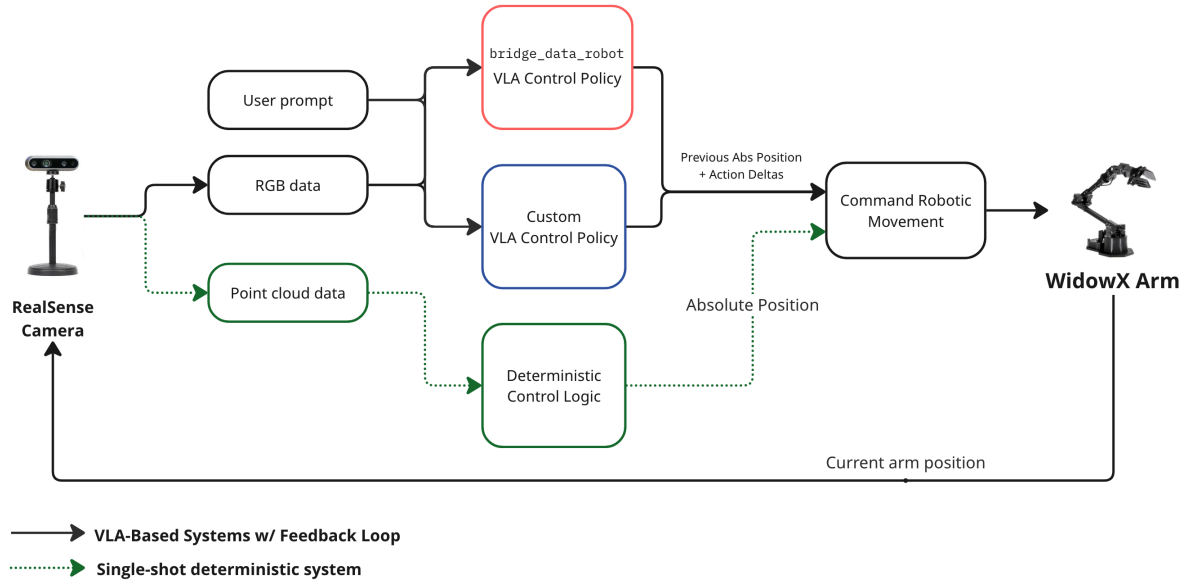

Figure 3.1: Overall system architecture, across three policy implementations

3.1 Hardware and Software Configuration

This section outlines the hardware components, software dependencies, and configuration steps necessary for the robotic manipulation experiments. Both the deterministic and AI-based (OpenVLA) systems share a common hardware setup and foundational software infrastructure, which are detailed below in this section. Distinctive system specifics for each system are described in Sections 3.2 and 3.3.

3.1.1 Hardware Configuration

The experimental system consists of three primary hardware components: the NVIDIA Jetson AGX Orin, the WidowX 250 6-DoF robotic arm, and the Intel RealSense D415 camera. An additional NVMe SSD was also installed on the Orin to support large foundation model storage requirements.

Jetson AGX Orin 64GB

The NVIDIA Jetson AGX Orin was selected primarily for its large GPU-accessible memory capacity essential for AI inference with large models, particularly for running the 7-billion-parameter OpenVLA model [4]. While it was suggested that a minimum GPU memory requirement of 16GB provided by the Jetson Orin NX would be possible with optimizations to run OpenVLA, optimal performance could be achieved by utilizing 32-64GB of GPU memory with the Jetson AGX Orin [28]. With 64GB of total RAM available for GPU usage, the AGX Orin exceeds minimum requirements and provides sufficient compute. Additionally, the Orin is recommended by NVIDIA as the preferred hardware platform for running the OpenVLA models [28].

Setup was straightforward, with configuration involving connecting the power supply via USB-C and performing a flash installation of the JetPack 6.2 SDK using NVIDIA's SDK Manager software.

WidowX 250 6-DoF Arm (Interbotix X-Series)

The WidowX 250 robotic arm was specifically chosen due to its inclusion in the Open X-Embodiment training dataset that was used to train the OpenVLA foundation model [25]. This enabled zero-shot capability running OpenVLA, without further model fine-tuning needed.

Additional benefits as a case to choose the WidowX 250 arm include its precise 6-degrees-of-freedom, ROS 2 compatibility, and an active open-source package suite provided by the vendor for rapid prototyping and experimental manipulation task execution.

Setup procedures included positioning the robotic arm centrally within a workspace meeting the following requirements: a flat, anti-slip surface of approximately 4.5 feet in diameter with a recommended vertical clearance of three feet [35]. To avoid confusion or performance degradation for perception, the environment should provide consistent indoor lighting conditions. Additionally, the arm's electronic enclosure and base was weighted down using diving weights to keep the base of the arm secure during arm movements, especially at the limits of its degrees of freedom. The provided gripper end-effectors were attached using the included hardware, followed by connecting the power and USB cables directly into the Jetson AGX Orin.

Intel RealSense D415 Camera

For perception tasks, the Intel RealSense D415 camera was chosen due to its compact form factor, accuracy in RGB imaging, and ease of ROS 2 integration. Mounted on an adjustable stand, the camera was placed optimally to capture both the robotic arm's workspace and its end-effector, allowing for vision-based manipulation. Like the robotic arm, the camera was connected directly to the AGX Orin via USB.

NVMe SSD Storage Configuration

Given the substantial storage requirements for large VLA models and associated Hugging Face caches, an aftermarket 1TB NVMe SSD was installed on the Orin. Setup involved formatting the NVMe drive using the ext4 filesystem, mounting it at /mnt/nvme, and configuring persistent mount settings in /etc/fstab. Appropriate permissions were explicitly set to ensure compatibility and writable access for all AI model-related data. Environment variables (HF_HOME, TRANSFORMERS_CACHE) were permanently configured to direct model downloads and caching directly onto the SSD, bypassing storage limitations of the Orin's native storage.

3.1.2 Software and Middleware Setup

A detailed and reproducible software foundation was established to support both deterministic and VLA-based systems, focusing on compatibility and performance. The full repository with all

custom code and links to adapted versions of Interbotix and BridgeV2 implementations can be found at the jakekemple/CyberLimb repository on Github [1]. Adapted pseudocode of evaluation scripts for each setup are found in Appendix B.

Operating System and ROS 2

The Jetson AGX Orin runs JetPack 6.2, an NVIDIA-optimized Ubuntu-based environment specifically tailored for Jetson devices. ROS 2 Humble [33] was installed to provide a decoupled middleware for pub-sub based node communication through topics. Essential packages (ros-humble-desktop) and ROS dependencies (rosdep) were installed to ensure easy package management.

While ROS 2 was chosen for this research due to its adoption as an industry standard in robotics middleware, it is important to acknowledge that alternative middleware architectures exist and may offer tradeoffs. ROS 2's publish-subscribe communication protocol and decentralized node structure provide the flexibility and modularity necessary to implement both the deterministic and VLA-based systems described in this work. However, this design may also introduce latency or timing unpredictability in specific scenarios. This thesis does not evaluate whether ROS 2 was the optimal choice compared to alternatives, but its use shaped the system architecture by encouraging a modular structure and simplifying integration across hardware and model components.

Interbotix ROS 2 Packages

The foundational software framework for the WidowX robotic arm consisted of a comprehensive suite of Interbotix ROS 2 packages [33]. These packages provided a robust software ecosystem, handling essential low-level motor control, inverse kinematics, trajectory generation, sensor integration, perception tasks, and visualization utilities.

Core Interbotix Robot Operation and SDK Packages

These core packages formed the infrastructure enabling arm operation:

- interbotix_xs_msgs: Standardized ROS 2 message definitions specifically created for Interbotix robotic systems, facilitating communication across modules.

[1] https://www.github.com/jakekemple/CyberLimb

- interbotix_xs_sdk: The primary software development kit (SDK) providing functionality for initializing hardware, managing communication with the Dynamixel servos, and abstracting hardware-level details into higher-level interfaces.
- interbotix_xs_driver: Low-level ROS 2 driver directly interfacing with the WidowX arm hardware, handling servo communication and status monitoring. Builds directly on top of the dynamixel_workbench_toolbox [36].
- interbotix_x_ros_control: Integration of the ROS 2 control framework, enabling advanced real-time control interfaces, precise servo management, and standardized robot state management.
- interbotix_xs_modules: Convenient higher-level utilities for common robotic arm operations, including inverse kinematics computations, trajectory execution routines, and gripper control.
- interbotix_common_modules: Shared utilities and functionality employed broadly across Interbotix packages, reducing redundancy and simplifying maintenance.
- dynamixel_workbench_toolbox: A third-party library integrated as a dependency for low-level Dynamixel servo communication and management, essential for controlling the Dynamixel-based actuators within the WidowX arm.

Perception-Related Packages (Core for VLA-Based System)

These specialized perception packages were integral to the image-processing pipeline used by robotic manipulation systems:

- interbotix_perception_modules: Offered ROS 2 interfaces for perception-related functionalities, including depth-based clustering, object localization via point cloud processing, and AR tag-based camera-to-robot frame calibration.
- interbotix_perception_pipelines: Provided ROS 2 pipelines orchestrating multiple perception modules into workflows for tasks such as object segmentation and pose estimation.
- interbotix_perception_msgs: Standardized perception-specific ROS 2 message definitions, allowing for communication between perception modules and other subsystems.

Support and Visualization Packages

Although optional for core system functionality, these packages were installed to provide additional utilities including visualization and debugging:

- interbotix_xs_rviz: Offered enhanced RViz visualization tools specifically tailored for Interbotix robotic arms, simplifying robot state monitoring, trajectory visualization, and real-time debugging.
- interbotix_tf_tools: Provided specialized utilities simplifying management of ROS 2 coordinate frame transformations (TF2), particularly useful for debugging and visualization of robotic arm and sensor data frames.
- interbotix_common_sim: Included utilities and integration support for simulated robotic operation, enabling easier testing and debugging of robot control logic without direct hardware dependency.

Additional ROS utilities and tools such as RViz, Xacro, and joint-state publishers complemented these packages, ensuring state visualization, robot description management, and efficient debugging throughout development and experimentation. This extensive Interbotix software stack provided a comprehensive foundation, reducing initial configuration complexity and supporting streamlined deployment of both deterministic and AI-based robotic manipulation experiments.

Camera Integration

Integration of the Intel RealSense D415 camera was facilitated through realsense2_camera, an official ROS 2-compatible driver. This package publishes calibrated RGB image data, which feeds directly into the perception pipeline of both systems. Configuration included verifying USB communication, udev rule creation, and functional testing via Intel's provided utilities (rs-enumerate-devices, realsense-viewer).

CUDA and PyTorch

CUDA 12.5 was installed to utilize GPU acceleration on the Jetson AGX Orin. Additionally, a GPU-compatible PyTorch 2.6 installation was obtained from a community-maintained Jetson-specific wheel distribution, essential for executing the OpenVLA model efficiently.

AI and Vision Dependencies

Critical software dependencies for running the VLA-based methods included Hugging Face's Transformers library (transformers==4.40.1), PyTorch's vision library (torchvision), and essential image processing libraries (numpy, opencv-python, and pyrealsense2).

3.1.3 General Utilities and Debugging

The system configuration included logging, diagnostics, and debugging tools inherent to ROS 2. Diagnostic logging provides efficient troubleshooting during experiments and precise identification of issues related to hardware or robotic manipulation accuracy.

This systematic hardware and software setup established a robust and reproducible baseline environment for the empirical evaluation of both deterministic and AI-driven robotic manipulation methods. Detailed specifics on perception and planning components unique to each system are presented in the subsequent Sections 3.2 (OpenVLA-based AI system) and 3.3 (Deterministic system).

3.2 VLA-Based System

This section describes two distinct approaches explored for integrating the OpenVLA model into the robotic manipulation system. Both approaches utilized the same underlying hardware and ROS 2 software infrastructure; however, each approach differed significantly in architecture and integration methods. An initial integration attempt introduced custom components for multimodal perception, natural language processing, and learned action generation, but encountered substantial implementation difficulties, proving out-of-the-box use of OpenVLA isn't possible without structured configuration code between the Interbotix arm control packages and the OpenVLA model. Subsequently, a second, successful integration was developed by modifying and patching the official `bridge_data_robot` repository associated with the Bridge V2 dataset [26], enabling effective real-world evaluation of the OpenVLA model.

3.2.1 OpenVLA Model Specifics

In both implementations, the OpenVLA model was integrated using the ROS 2-based framework to enable zero-shot robotic manipulation through natural language instructions combined with

visual perception. The model was selected due to its ability to generalize across diverse robotic tasks, its zero-shot deployment capabilities, and compatibility with the WidowX 250s robotic arm used in the experiments [4]. OpenVLA fundamentally differs from conventional robotic control approaches by employing a multimodal transformer architecture that processes visual and natural language inputs to produce action deltas for control commands to consume.

Specifically, the OpenVLA model consists of the following architectural components [4]:

- Vision Encoder: Utilizes DINOv2 and SigLIP to extract meaningful visual embeddings from RGB images captured by the Intel RealSense camera.
- Language Encoder: Processes the provided natural language instruction using a LLaMA-7B language model.
- Fusion Layer: Combines visual embeddings and language embeddings into a unified multimodal representation.
- Action Decoder: Outputs a sequence of seven discrete action tokens representing positional (Δx, Δy, Δz), rotational (Δroll, Δpitch, Δyaw), and gripper (open/close) commands.

These action tokens were mapped to continuous robot-specific commands through linear scaling and detokenization. A high-level, inference-in-the-loop flow continuously captured camera images, combined them with natural language prompts, executed inference on the inputs to obtain tokens, decoded the tokens to actionable commands, and executed movements on the robot in a tightly integrated feedback loop. Refer to Figure 3.2 from the OpenVLA paper for a visual representation of this loop.

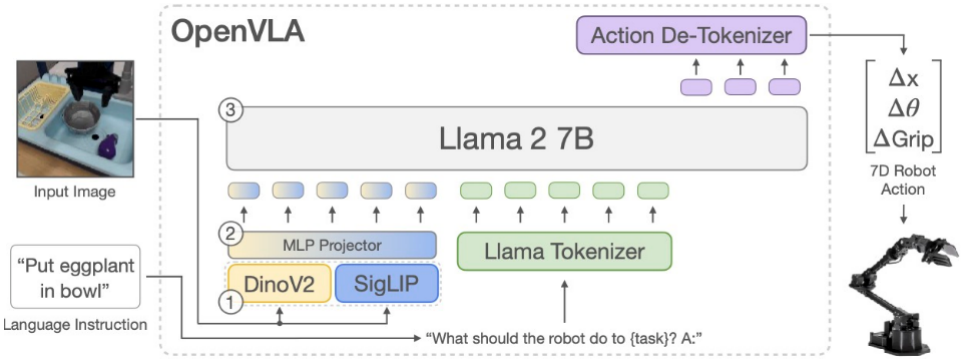

Figure 3.2: OpenVLA architecture diagram. Reproduced from OpenVLA paper [4].

3.2.2 Initial VLA Integration Attempt (Custom ROS 2 Architecture)

The initial approach to integrating the OpenVLA model into the robotic manipulation system relied on custom-developed ROS 2 nodes and inference pipelines. The purpose of this implementation was to test whether OpenVLA could be implemented 'out of the box' by simply applying outputted action deltas to exposed Interbotix package functions for resolving deltas to appropriate joint trajectories. However, serious challenges arose related to inference latency, calibration accuracy, and implementation complexity. The custom architecture demonstrated a successful inference-in-the-loop pipeline, but fails to result in meaningful task progress. Consequently, this method proved unsuitable for final experimental evaluations, motivating the development of a subsequent, improved approach described in Section 3.2.3.

Integration into ROS 2 was achieved through three custom nodes, detailed below and shown in Figure 3.3. Each node assumed a separate responsibility within the VLA-system pipeline. A ROS 2 launch file orchestrated the simultaneous startup of these nodes, ensuring correct node initialization, parameterization, and namespace management to maintain clarity within the ROS 2 environment.

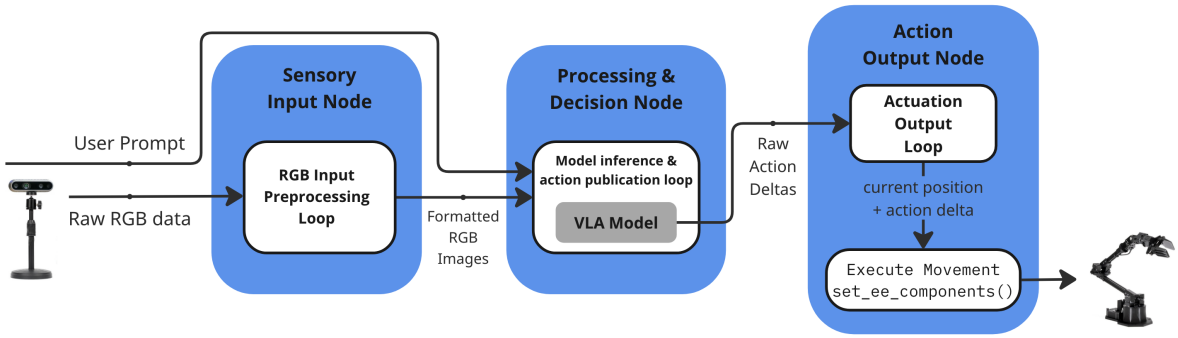

Figure 3.3: Custom OpenVLA ROS 2 Integration diagram.

Sensory Input Node

This node continuously captured RGB frames from the RealSense D415 camera using the pyrealsense2 library, configured to stream RGB-formatted data at 5Hz to maintain real-time responsiveness. Images were published on the /camera/image_raw topic as ROS 2's standard sensor_msgs/Image messages, removing the unnecessary overhead of format conversions.

Processing Decision Node

This node was subscribed to the camera image stream and natural language instruction topics, executing OpenVLA model inference. Image frames were preprocessed into PIL format and combined with the natural language instruction (e.g., "put block onto paper dot") into a structured prompt. The OpenVLA model produced seven discrete action tokens, subsequently decoded to real-world actionable robot pose and gripper command deltas. The action deltas vectors were then published as ROS 2 String messages to the /openvla_action topic.

Action Output Node

This node received action delta vectors and translated them into robotic motions. It performed critical coordinate transformations to map incremental actions from the camera's reference frame into the robot's base frame using inverse kinematics and transformations managed by ROS 2's tf2 library. This node directly commanded low-level Interbotix control modules (interbotix_xs_modules) to smoothly execute positional and rotational adjustments, as well as gripper actuation based on model outputs.

ROS 2 Launch File and Execution Procedure

A dedicated ROS 2 launch file orchestrated node initialization and execution order. The `xsarm_control.launch.py` file first initialized the Interbotix core arm controllers. After a short delay to ensure hardware readiness, the AI-specific ROS 2 nodes (Sensory Input Node, Processing Decision Node, and Action Output Node) launch. This structured launch file enables consistent startup behavior for all required nodes to run the AI-system experiment.

3.2.3 Successful VLA Integration Based on bridge_data_robot Adaptation

The second integration approach employed a modified version of the official `bridge_data_robot` repository, originally designed to create the necessary experiment conditions and evaluate models trained on the Bridge V2 dataset. This repository provided undocumented information about how to configure communication and control of the arm, such as explicit workspace boundaries, initial robot poses, and precise action scaling parameters, which were essential for accurately deploying the OpenVLA model on the WidowX 250 robotic arm. Required patches were applied to ROS 1 scripts within the repository to enable their execution within the ROS 2 environment,

removing integration challenges and preserving low-level functionality. These control processing layers are depicted in Figure 3.4.

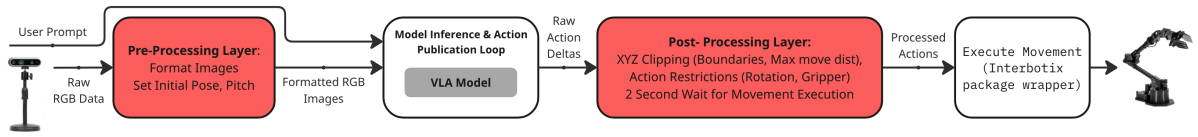

Figure 3.4: Abstracted `bridge_data_robot` Implementation Architecture

bridge_robot_data Evaluation Script Architecture

The updated architecture used an evaluation script (`run_bridgev2_eval.py`) to run the pick-and-place experiment. At startup, the evaluation script configured robot parameters (e.g. camera_topics) through a Python data class, enabling reproducibility without modifying core code. Refer to the pseudoscope block in Appendix B.2 understand the adapted evaluation script logic.

During execution, the evaluation loop continuously captured RGB images from the camera via ROS 2 topics. Each image and a natural-language task prompt (e.g., "place the block onto the paper dot") were processed by OpenVLA to output discrete action tokens encoding positional, rotational, and gripper adjustments. These normalized actions were explicitly rescaled to real-world Cartesian coordinates within defined workspace bounds. Cartesian commands were converted to joint-space movements using Interbotix inverse kinematics and executed through ROS 2 publishers (`/wx250s/commands/joint_group`, `/wx250s/commands/joint_single`). The arm motion was configured to run in non-blocking mode, meaning the latest action delta outputs were acted upon immediately to provide smooth and continuous task completion. Refer to the code block in Appendix B.2 for the adapted evaluation loop script.

Task Execution and Motion Control Logic

At each episode's start, the robot arm was initialized to an explicitly configured end-effector position and orientation to make the robot appear more natural and minimize occlusions. The initial Cartesian position was set approximately 30 centimeters in front of the robot base ([0.3, -0.09, 0.26]), with an intentional pitch orientation offset (quaternion [0, -0.259, 0, -0.966]). This downward pitch was chosen to provide the camera with an optimal viewing angle toward the workspace, enhancing visual clarity for object detection and manipulation. Following the initialization, each control step processed a 7-dimensional action vector (dx, dy, dz, d_roll,

d_pitch, d_yaw, gripper). Movements were constrained to prevent risky or unpredictable jumps in joint position:

- Translation: ±5 cm per axis
- Rotation: ±0.25 radians per axis
- Gripper: normalized [0.0 closed, 1.0 open]

Action vectors exceeding these limits were clipped and clipping events were logged. Actions were applied incrementally to the current robot pose, creating a new target pose represented as a 4×4 transformation matrix. Before execution, poses were also clipped within workspace bounds (e.g., `[0.05, -0.25, -0.02, -1.57]` lower and `[0.50, 0.30, 0.35, 1.57]` upper limits). Resulting joint targets were published to ROS 2 topics with durations of 2.0 seconds per action. A low control frequency of 0.5 Hz was chosen to ensure smooth motion and allow time for new image inputs. Gripper actions were represented by binary open/close states with a value threshold of 0.5.

Episode Lifecycle and Reset Logic

Each evaluation episode began with a systematic robot reset to a predefined neutral, random, or user-defined pose. After reaching the initial pose, the system paused briefly (1.5–2.0 seconds) for sensor stabilization before execution began.

Runtime Safety and Exception Handling

The evaluation system incorporated robust safety mechanisms, including explicit workspace boundary enforcement, motion error handling, and camera timestamp synchronization checks (tolerance of $\tilde{0}.1$ seconds). Violations raised exceptions and were logged to prevent unsafe conditions.

Logging and Data Capture

Detailed logging enabled quantitative and qualitative analysis. When activated, each episode captured images, joint states, actions, and timestamps. Captured frames could be automatically compiled into MP4 videos for review. A timer started right before the inference execution and ended at the end of an episode, outputting the elapsed time after each step. This allowed a record of when any step resulted in a notable action (e.g. block was picked up).

Advantages and Practical Improvements

This integration offered a few changes, differences, and improvements over the initial, custom VLA-based system attempt:

- Explicitly clarified previously undocumented workspace configurations (e.g. downward pitch for natural gripper orientation).
- Minimal ROS 1-to-ROS 2 patches reused proven vendor PID and kinematics logic.
- Enhanced stability due to precise action scaling and workspace bounds.

In summary, the adapted `bridge_robot_data` repository significantly improved the OpenVLA model deployment, provided clarifications of previously obfuscated configurations, and enabled stable, reproducible real-world robotic evaluations. The patching and adjustments to the original `bridge_robot_data` codebase were kept minimal and targeted solely at enabling compatibility with the ROS 2 environment. These adaptations were not expected to materially impact system performance or inference latency. However, the project was challenging to implement and required an intensive understanding of underlying dependencies to interpret the functionality or properly configure tasks with custom modifications

3.3 Deterministic Control System

This section outlines the deterministic robotic manipulation system, emphasizing algorithm based perception methods, predefined task planning logic, and detailed software configuration implemented without AI inference. The deterministic approach utilized standard robotics perception techniques provided by the Interbotix ROS 2 perception packages and custom adaptations based on demonstration scripts provided by Interbotix.

3.3.1 Architecture and Distinctive Components

The deterministic configuration distinctly consists of the following key components:

Perception Pipeline

This pipeline utilized classical methods, including color-based segmentation, contour extraction, point-cloud clustering, and centroid calculations to localize and identify target objects explicitly.

RGB images from the Intel RealSense D415 camera were processed by standard computer vision (OpenCV) methods integrated with ROS 2 modules. No AI-based solutions such as transformer-based vision models or learned embeddings were utilized, which simplified computational complexity and reduced latency.

Predefined Task Logic

These were tasks executed based on scripted decision-making logic operating directly on explicit perception results. A deterministic state machine managed task progression, ensuring predictable and repeatable execution without probabilistic inference-based components.

Interbotix Control Modules

Low-level motor control, inverse kinematics calculations, and trajectory execution were identical to the AI-based configuration, utilizing Interbotix ROS 2 modules. The deterministic system differed only in high-level control decisions, not in the underlying motor execution.

3.3.2 Deterministic Software Configuration

The deterministic robotic system relied exclusively on open-source ROS 2 packages for perception and arm control, as detailed below. The software pipeline was explicitly configured for deterministic manipulation, incorporating predefined object localization and robot command logic.

ROS 2 Middleware Infrastructure

- Middleware: ROS 2 Humble provided inter-node communication, action handling, and parameter configuration [33].

RealSense Camera Integration

- ROS 2 Package: realsense2_camera
- Hardware: Intel RealSense D415
- Function: Integrated camera streamed calibrated RGB-D data continuously into the deterministic perception pipeline.

Point Cloud and Object Clustering

- ROS 2 Package: InterbotixPointCloudInterface
- Function: Depth data processed into 3D point clouds; explicitly segmented clusters represented individual objects. Object centroid positions provided precise Cartesian coordinates relative to the robot's base frame.

Coordinate Transformation and Frame Calibration

- ROS 2 Package: InterbotixArmTagInterface
- Calibration: Camera-to-robot-base frame transformations explicitly calibrated via AR markers (ArUco tags) affixed to the robot arm.
- Transformation Method: Marker poses converted from camera coordinates to robot-base coordinates using ROS 2's tf2 library, ensuring accurate robotic manipulation.

Robotic Arm Manipulation

- ROS 2 Package: InterbotixManipulatorXS
- Function: Low-level control for inverse kinematics, trajectory generation, and servo control, converting explicit Cartesian pose commands into executable trajectories.

3.3.3 Deterministic Task Implementation and Configuration

The pick-and-place task was adapted from Interbotix's provided demonstration scripts, closely mirroring AI-driven tasks for comparative evaluation.

Pick-and-Place Task Implementation

The tasks were executed using color-based segmentation perception. Objects were detected using predefined HSV color thresholds, contour extraction, and centroid calculations to determine object coordinates. There is hard-coded logic directing the robot to grasp objects and transfer them to a predefined location. Configuration involved includes calibration of HSV thresholds, static transforms for camera-robot coordinate frames, and predefined coordinates or geometric placements for each action.

Pick-and-Place Script Adaptation

The deterministic experiments utilized adapted versions of Interbotix's provided demonstration scripts, explicitly configured to match experimental conditions. Refer to the code block in Appendix B.1 for the core logic, adapted from the Interbotix `pick_place.py` script. Additionally, see Figure 3.5 for an abstracted flow diagram of the script content.

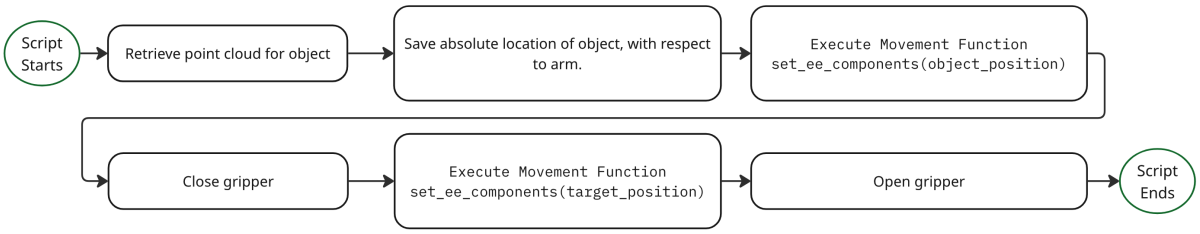

Figure 3.5: Deterministic control logic script

The scripts include the following steps:

1. The arm, point-cloud interface, and AR-tag interface were instantiated.
2. Calibration of reference frames via transforms between camera and robot frames using AR markers (`find_ref_to_arm_base_transform`).
3. Object localization via point-cloud clustering. The visual scene was segmented into discrete clusters corresponding to individual objects, sorted systematically to provide ordered object coordinates.
4. Predefined Robot Motion Execution: The arm moved above each object, descended to grasp calculated object positions, grasped the object, and moved to predetermined locations. Cartesian coordinates and orientations (`set_ee_pose_components`) defined each incremental action, ensuring deterministic execution.

3.3.4 ROS 2 Launch File and Execution Procedure

The deterministic system utilized a dedicated ROS 2 launch file adapted directly from Interbotix demonstration scripts, sequentially initializing the core Interbotix arm control drivers (xsarm_control.launch.py), perception nodes for color segmentation and contour extraction, and

task-specific logic nodes controlling robot trajectories based on predefined inputs and conditions. Parameters and nodes were clearly documented in structured YAML files, simplifying adjustments and ensuring reproducibility.

3.3.5 Comparative Configuration for Evaluation

To ensure accurate comparisons, deterministic task scripts were configured to replicate OpenVLA experimental conditions. Object placements, workspace dimensions, lighting conditions, and task definitions were maintained consistently, differing only in underlying methodologies used for perception and task planning and camera placement to support these different perception methods. This alignment facilitated empirical comparisons on accuracy, repeatability, execution speed, and robustness to object placement variations, providing a clear baseline against AI-based VLA models.

Chapter 4

Experiment Setup and Trials

This chapter defines the experimental setup used to evaluate both deterministic and VLA-based robotic manipulation systems on a representative pick-and-place task. Through a controlled series of 10 trials per system, performance was assessed using standardized industrial robotics metrics-accuracy, repeatability, and cycle time-adapted from ISO and prior robotics literature. The deterministic system followed a traditional control script with fixed coordinates, while the VLA-based system operated on natural-language instructions parsed by the Bridge V2 foundation model. Experimental conditions were carefully standardized, with randomized block positions, consistent workspace initialization, and calibrated vision and manipulation pipelines. This rigorous setup allowed for direct comparison of the two systems' capabilities and limitations, both quantitatively and qualitatively, in preparation for the results presented in the following section.

4.1 Experiment Scenarios

To evaluate the performance of both deterministic and VLA-based robotic manipulation systems, a representative and moderately complex task, pick-and-place, was selected. This task required the robot to perceive an object, plan and execute the necessary motions to grasp it, identify an appropriate target location, and complete the placement with precision and accuracy. Pick-and-place is a foundational operation in both industrial and service robotics, making it an ideal benchmark for comparing traditional control methods with emerging VLA-based approaches. All trials were conducted using standardized 1-inch by 1-inch colored wooden blocks, a common object choice in manipulation benchmarks.

In each trial, a single block was placed at a randomized position within a 28 cm by 20 cm workspace situated directly in front of the robotic arm and camera. The robot then executed the task based on either a deterministic command with fixed target coordinates (issued via the Interbotix demo script), or a natural-language instruction parsed and interpreted by the OpenVLA model-for example, “Put block onto paper dot.” This setup enabled a direct and meaningful comparison between rigid, pre-programmed logic and flexible, language-conditioned inference.

4.2 Evaluation Setup

4.2.1 Controlled Factors

To rigorously evaluate performance, each of these tasks were repeated under controlled conditions:

- **Number of Trials:** The task was executed ten times for each approach (deterministic vs. OpenVLA).
- **Randomized Block Positions:** For each trial, the block was randomly repositioned within a predefined workspace to ensure robustness and avoid biased outcomes.
- **Randomized Block Orientation:** To evaluate the adaptivity of the system, the orientation of the block at its initial position was randomized. The final orientation of placement for the pick-and-place was not considered, only the center-point of the block.
- **Measurement Apparatus:** To measure as accurately as possible, 1mm graph paper was utilized.
- **Block Color and Shape:** The blocks used for all trials were 1 inch by 1 inch (25.4mm by 25.4 mm) wooden blocks of 6 colors. The same distribution of colors were used across trials for the deterministic and VLA-based system.
- **Consistent Initial Conditions:** Each trial begins from a standardized initial pose of the robotic arm to ensure comparability.

Note that camera position is not a control variable, since the two systems rely on fundamentally different visual inputs of RGB image streams for the VLA-based system and point

cloud data for the deterministic system. Maintaining the same camera pose across both would compromise functionality, since each system requires a different perspective to operate correctly. In order to measure both systems and their competence in task completion, the appropriate camera positions were placed for each individual system.

4.2.2 Metric Definitions

Performance metrics were collected using methodologies adapted from established literature [32, 31, 29], specifically tailored to the manipulated objects as detailed in Section 2.3. These ISO-derived metrics emphasize task-oriented outcomes and successful task completions, in contrast to model-centric performance measures such as those presented by LIBERO [8], traditionally used to evaluate VLA models and are independent of accuracy or precision of the final outcome. Utilizing ISO metrics provided a clear contextual framework for comparing VLA-driven performance to the deterministic control system, aligning with standard industrial and mechanical robot evaluation criteria.

For pick-and-place, the primary metrics were accuracy, repeatability, and cycle time. See Table 4.1 for details and important formulae.

More specifically, two accuracy measures were tracked. First, the final center point of the block for ten trials, on a coordinate plane centered on the target dropping point, was tracked. Then, the average of these coordinates across the ten trials were computed, to achieve the bias vector of the robot. Finally, the accuracy was the Euclidean (pick-and-place) between the achieved average final position of the target blocks (bias vector) compared to the intended target placement position.

By contrast, repeatability evaluated the consistency of the outcomes, measured by the standard deviation of the actual outcomes, regardless of how close they are to the target. Thus, high repeatability meant the robot consistently reached nearly the same outcome, even if systematically off from the target [32]. There were two repeatability metrics: axis repeatability and repeatability radius. Axis repeatability is an axis-wise standard deviation of the final positions of the blocks across all trials, represented as a vector. Repeatability radius used the standard deviation in x and y to compute the radius of inclusion for 99 percent of dropped points - that is, the radius around the target point that the arm is 99 percent likely to drop the block in.

Lastly, cycle time was clearly defined and measured as the elapsed time from issuing the initial movement command or written query to the moment the robot successfully completes the

manipulation, verified through visual observation with the RealSense D415 camera. This was measured automatically and precisely through time stamped events logged by ROS 2.

	Description	**Formula**
Bias vector	Average vector of all the dropped points.	$(\bar{x}, \bar{y})$ where $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$
Accuracy	Distance from average vector p to desired center q.	$d(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$
Axis Repeatability	Standard deviation axis-wise of all the dropped points.	$\sigma_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}$ and $\sigma_y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}$
Repeatability Radius	The radius of inclusion for 99 percent of dropped points.	$\sigma = 3\sqrt{\sigma_x^2 + \sigma_y^2}$
Cycle Time	Average time to completion, across N trials.	$\bar{c} = \frac{1}{n}\sum_{i=1}^{n} c_i$

Table 4.1: Pick-and-place Evaluation Metrics

4.2.3 Equipment Requirements

After following setup instructions and procedure in Section 3, the experiment setup finally used the following hardware and software components.

- Hardware
  - WidowX 250 Robot Arm (6-degree-of-freedom)
  - Intel RealSense D415
  - Secondary laptop
  - Jetson AGX Orin
- Software
  - ROS 2

  - RViz
  - Google Sheets (or other spreadsheet software)

- Physical Materials
  - April Tag
  - Colored 1 inch by 1 inch (25.4 mm by 25.4 mm) blocks
  - Painter's tape
  - 1mm graph paper
  - Pen

4.3 Key Execution Commands

The following commands were used throughout the experiment procedure and should be referenced when certain commands are required. Commands will be labeled in accordance to their utility.

4.3.1 Common Commands

1. Enable arm torque: `ros2 service call /wx250s/torque_enable interbotix_xs_msgs/srv/TorqueEnable {"cmd_type: 'group', name: 'arm', enable: true"}`
2. Disable arm torque: `ros2 service call /wx250s/torque_enable interbotix_xs_msgs/srv/TorqueEnable {"cmd_type: 'group', name: 'arm', enable: false"}`

4.3.2 Deterministic System Commands

1. Launch camera node (on separate computer if needed): `ros2launchrealsense2_camerars_launch.py align_depth:=true depth_module.depth_profile:=640x360x15 rgb_camera.color_profile:=640x360x15 pointcloud.enable:=true initial_reset:=true`

2. Throttle passthrough topic setup to reduce frames per second of camera output (on a separate computer if needed): `ros2runtopic_toolsthrottlemessagescamera/camera/depth/color/points1.0camera/camera/depth/color/pointst`

3. Launch perception pipeline: `ros2launchinterbotix_xsarm_perceptionxsarm_perception.launch.py robot_model:=wx250s use_pointcloud_tuner_gui:=true use_armtag_tuner_gui:=true load_configs:=false`

4. Run Pick and Place script (in a separate tab): `cd ~/interbotix_ws/scripts` and `python3 pick_place.py`

4.3.3 VLA-Based System Commands

1. Launch camera node:
```
ros2 launch realsense2_camera rs_launch.py \
camera_name:=camera_stream \
frame_id:=world \
fps:=30 \
reconnect_timeout:=0.0 \
enable_sync:=false \
initial_reset:=false \
pointcloud.enable:=false \
align_depth.enable:=false \
color_qos:=DEFAULT \
depth_qos:=DEFAULT \
infra_qos:=DEFAULT \
gyro_qos:=DEFAULT \
accel_qos:=DEFAULT
```

   - **Camera Topic:** `/camera/camera_stream/color/image_raw`
   - **Replaced topic in:** `/mnt/nvme/cyberlimb-bridge/bridge_data_robot/widowx_envs/widowx_env_service.py`

2. Launch Arm Control SDK: `ros2 launch interbotix_xsarm_control xsarm_control.launch.py robot_model:=wx250s use_rviz:=false load_configs:=false`

3. Run OpenVLA Eval Script (Should run cached openvla base model): `python /mnt/nvme/cyberlimb-bridge/openvla/experiments/robot/bridge/run_bridgev2_eval.py --model_family openvla --pretrained_checkpoint openvla/openvla-7b`

4.4 Evaluation Procedure

Each trial followed these clear and repeatable steps in order to gain meaningful insights. Two separate initialization procedures occurred for the pick-and-place tasks, varied across the deterministic and VLA-based systems. Then, the experiment trials took place, and finally metrics were read. In total, 10 trials were run for each robotic system and task, totaling 20 trials.

4.4.1 Deterministic Pick-and-Place

1. **Initialization of Workspace:**
   (a) Configure robot arm and camera as shown in Figure 4.1. Cross-reference with the camera view in Figure 4.2.
   (b) Place a sheet of 1mm graph paper on workstation in front of robot arm.
   (c) Tape this sheet down by making loops of painter's tape and sticking them underneath the corners of the paper. This ensures the tape is not visible to the cameras during execution.
   (d) Outline a 1 inch by 1 inch (25.4 mm by 25.4 mm) square on the paper. This is the desired dropping point (as configured in the deterministic code).
   (e) Mark its center with a pen or permanent marker. This will be the origin for a simple Cartesian plane, which will help with simple measurement of distances to goal for pick-and-place. Note that the x and y directions were of a typical coordinate grid, from the perspective of the robot arm. See Figure 4.3 for clarification.
   (f) Place the red block outside of the printer paper randomly somewhere on the workstation, within a 28 cm by 20 cm area next to the paper.
2. **Calibration:**
   (a) Safely disable arm torque (if currently torqued).

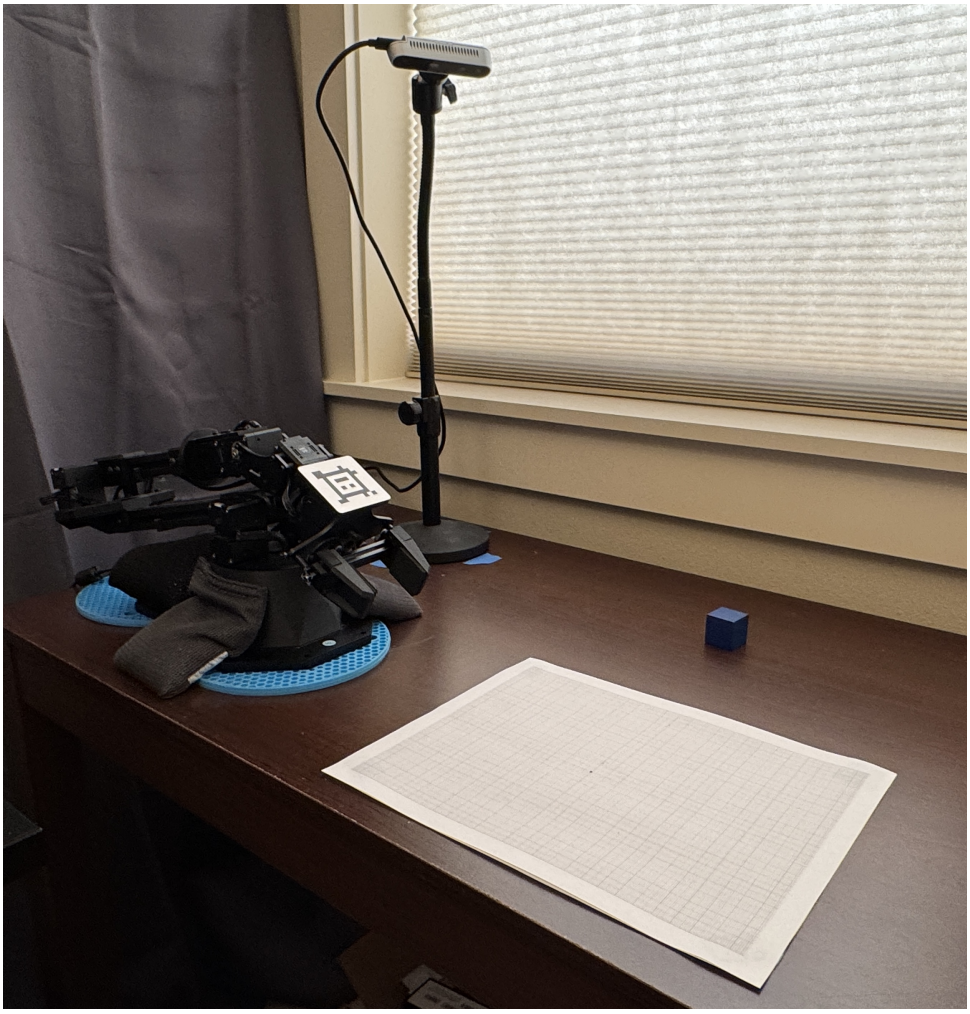

Figure 4.1: Deterministic Pick and Place Setup (required for point cloud rendering).

(b) Make sure the Realsense ROS node is running. If not, launch the Camera Node.

(c) Then, run the perception pipeline script. RViz should appear, with live camera feed, as setup via topic subscription.

(d) Ensure the external laptop has the camera feed publishing. Adjust the frame rate and resolution if the feed is not sufficiently fast. Throttle with a passthrough topic if needed. Refer to Section 4.3.2 for the deterministic execution commands for this passthrough.

(e) Physically lift and position the arm such that the april tag is visible to the camera feed.

(f) Enable arm torque.

(g) In RViz, go to the snapshot pop-up window and run calibration for 10 snapshots. This should save the transform required from the camera view to the robot arm view.

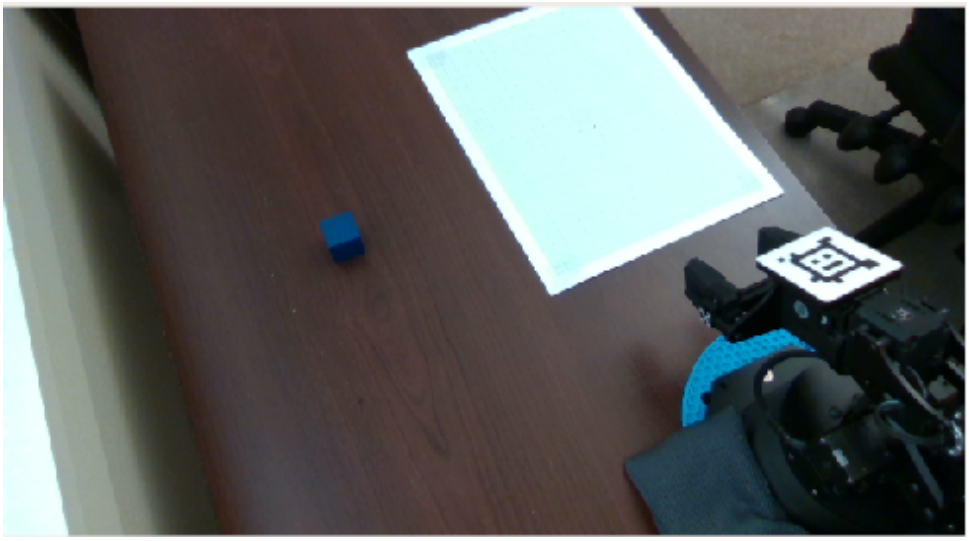

Figure 4.2: Deterministic Pick and Place Setup from camera view.

(h) Safely disable arm torque (hold it before torquing off so it does not collapse abruptly).

(i) Now, calibrate the filters in the Pointcloud Tuner GUI window. Set the bounding box (crop box) and filters such that the arm is not in the crop box area, the block appears clearly in the crop box , and the point clouds are visible without the residual impacts from tabletop surface. Recommended settings from this experiment are in Table 4.2, based on camera calibration in step 2f. Setup should look like Figure 4.4.

Tuner Setting	**Parameter**	**Value**
Crop Box Filter	X min [m]	-0.20
Crop Box Filter	X max [m]	0.08
Crop Box Filter	Y min [m]	-0.26
Crop Box Filter	Y max [m]	0.13
Crop Box Filter	Z min [m]	0.44
Crop Box Filter	Z max [m]	0.67
Voxel Grid Filter	Leaf Size [m]	0.004

Tuner Setting	Parameter	Value
Segmentation Filter	Dist Threshold [m]	0.006
Segmentation Filter	Max Iterations	50
Radius Outlier Removal Filter	Min Neighbors	5
Radius Outlier Removal Filter	Radius Search [m]	0.010
Cluster Filter	Min Size	50
Cluster Filter	Max Size	1000
Cluster Filter	Tolerance [m]	0.020

Table 4.2: PointCloud Tuner GUI Recommended Settings

3. **Task Issuance:** Torque on arm. Then issue deterministic command to the robot via pick and place script. Refer to Section 4.3.2 for these deterministic system commands.
   (a) Launch Camera Node on separate laptop.
   (b) Launch throttled topic (optional).
   (c) Launch perception pipeline. Ensure point clouds look correct according to calibration procedure.
   (d) Run pick and place script.
4. **Execution:** Robot executes the manipulation task.
5. **Recording Outcome:**
   (a) Measure the x and y distance between the dot drawn during setup and the center of the red block. One can obtain the center by outlining the block first on the paper,

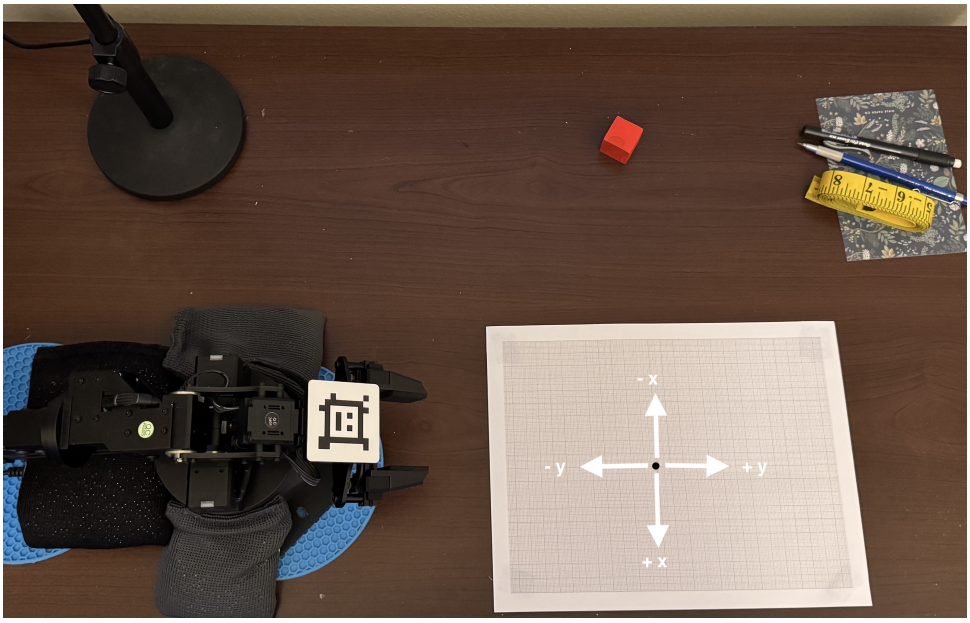

Figure 4.3: Coordinate system used with respect to robot arm

marking the center of the square, and then measuring. See Figure 4.5 for an example.

(b) Log this information on a spreadsheet software.

(c) Log the precise ROS 2 timestamps for start and completion of task on your spreadsheet.

- Start Time: Timestamp at which the robot controller publishes the first trajectory command.
- End Time: Timestamp when the Interbotix motion_done callback or equivalent signal is triggered.

6. **Metric Calculation:** Compute accuracy (or success rate), repeatability, and cycle time for each trial based on the formulae from Table 4.1.

7. Reset a different colored block to a random location outside the printer paper (cycle between any available colors with equal frequency).

8. Run steps 3-7 for nine additional trials, varying the color of the blocks.

9. **Analysis:** Aggregate results across trials to obtain statistical significance. Use tools in the spreadsheet for these computations of mean, standard deviation, etc. in accordance to the formulae in Table 4.1.

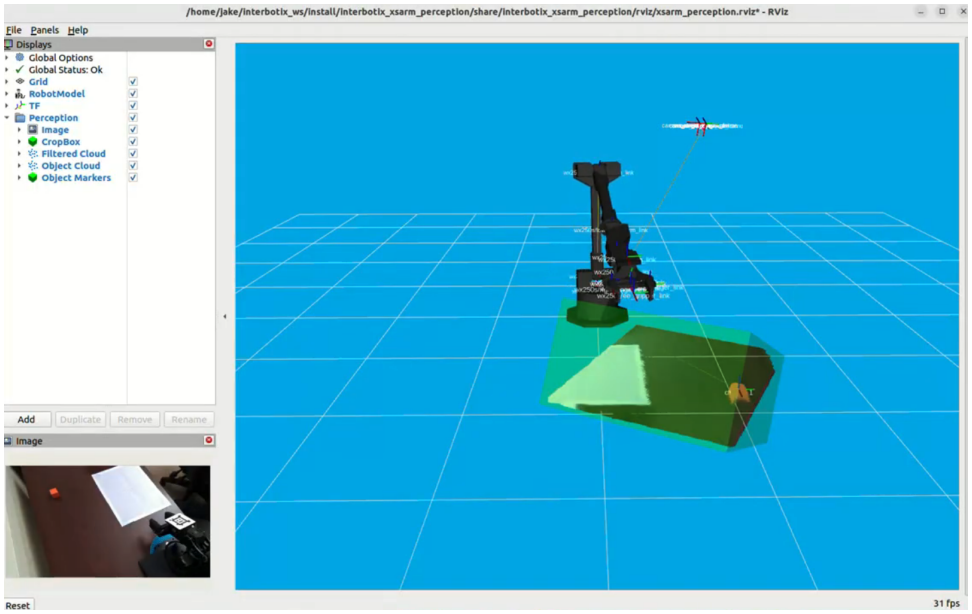

Figure 4.4: RViz Point Cloud GUI with crop box and filters

4.4.2 VLA-Based Pick-and-Place

1. **Initialization of Workspace:**

   (a) Follow the initialization steps 1a - 1d from the deterministic pick-and-place, configuring the robot as in Figure 4.6. The camera view should appear like the rollout frames in Figure 4.7, which can be validated through the Realsense Node visualizer. Note that this mimics the Bridge V2 training dataset as exactly as possible.

   (b) Place the red block outside of the printer paper randomly somewhere on the workstation.

2. Ensure the VLA policy has a maximum of 80 steps - the system does not have its own task completion mechanism, so the robot would operate forever without exit criteria. After many trials, a reasonable set of steps was determined to be 80 (stopping at around 160 seconds).

3. **Task Issuance:** Run the VLA procedure for a single trial. Refer to VLA-specific execution commands in Section 4.3.2.

   (a) Launch the camera node

   (b) Launch the arm control SDK

   (c) Run the open VLA eval script

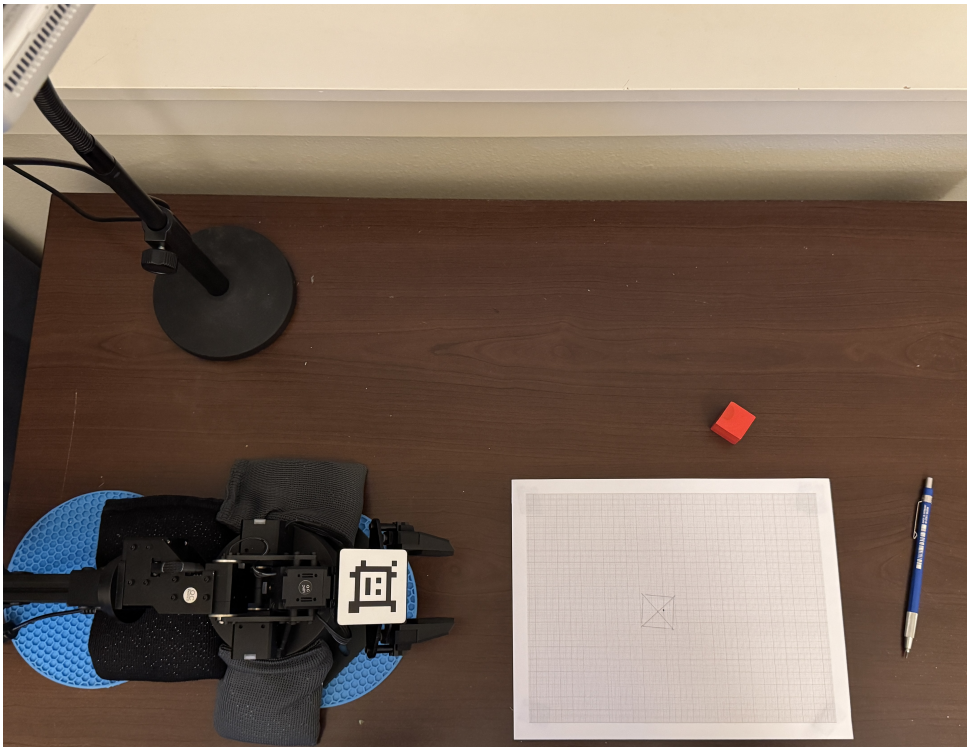

Figure 4.5: Example outlined box at the trial measurement stage

(d) Type the natural language command "Put block on paper dot."

4. **Execution:** Robot executes manipulation task.

5. Follow step 5 from the deterministic pick-and-place procedure for noting results.

6. Repeat step 3 and 4 for nine additional trials.

7. Follow step 9 from the deterministic pick-and-place for calculating metrics and logging.

Figure 4.6: VLA-Based Pick and Place Setup (required for inference).

Figure 4.7: Rollout frames from a trial of the VLA policy to demonstrate camera view.

Chapter 5

Setup and Installation Complexity Analysis

This chapter evaluates and compares the overall setup and installation complexity, as well as practical challenges associated with implementing both deterministic and VLA-based robotic manipulation systems. This analysis provides insight into the real-world feasibility and potential for adoption of VLA models when compared to conventional deterministic control systems. Refer to Chapter 7 for suggested improvements to setup and installation challenges described in this chapter.

5.1 Comparative Analysis of System Installation Complexity

5.1.1 Deterministic System Installation and Configuration

As described in Chapter 3, the deterministic system was built upon open-source ROS 2 packages primarily provided by Interbotix, alongside standard middleware and camera drivers. The initial installation benefited from organized documentation, pre-written demonstration scripts, and step-by-step walkthroughs. Despite these advantages, a few complexities arose during task-specific adaptation. The provided demonstration scripts were sensitive to precise environmental configurations, requiring near-exact placement of the robotic arm, camera, and objects for manipulation success. As a result, deviations from the original demonstration environment required adjustments.

Key challenges encountered include precisely updating arm trajectories whenever the workspace layout changed, tuning color segmentation thresholds within OpenCV due to variations in lighting or object coloration, and recalibrating camera-to-robot frame transformations

whenever the physical position of the camera or arm shifted. These repetitive manual interventions highlighted the core limitation: a deterministic system, while predictable and reliable once configured, lacked flexibility and adaptability, necessitating manual reconfiguration whenever task parameters or environmental conditions changed.

Another related discovery was that the complexity associated with implementing deterministic methods increased significantly as tasks became more complex. For simpler scenarios, manual recalibration was manageable. However, as tasks began to include additional objects, require higher precision, or involve variable object orientations, configuring the perception and planning routines demanded more effort. Each new manipulation scenario required manual updates to robot trajectories. As a result, the deterministic approach lacked robustness to environmental changes and task variability, significantly limiting its adaptability.

Despite these limitations, the deterministic control system offered a significant advantage in terms of computational resources. It required substantially less computing power than the inference-based VLA system, making deterministic control logic appealing for hardware-constrained environments or applications with strict computational budgets. While VLA models depend on GPUs and substantial GPU-accessible memory, deterministic methods have no such requirements.

Overall, while deterministic control systems provided predictable and reliable execution once configured, their rigidity and extensive manual recalibration requirements represented barriers, especially within dynamic robotic manipulation environments.

5.1.2 VLA-Based System Installation and Configuration

In comparison, installation and configuration of the OpenVLA-based robotic system proved complex and time-intensive in different ways. While the hardware configuration process remained largely consistent across both systems, numerous software and configuration challenges unique to the VLA-based implementation emerged. Notably, substantial effort was required to set up and configure the inference loop, which includes timing and management of image capture frequency, model inference frequency, and robotic action execution rates. Unlike deterministic setups, no established guidelines or frameworks existed for easily configuring the inference loop logic tailored to the hardware constraints of the experimental setup.

Though this research intentionally limited the scope to evaluating zero-shot performance, real-world deployments would likely require decent effort fine-tuning the model to optimize task

accuracy. Fine-tuning demands computational resources and hyperparameter tuning, aspects beyond this study's scope but important to acknowledge as notable effort and complexity.

Another complexity arose from GPU resource constraints. The 7-billion-parameter OpenVLA model required substantial GPU memory access for inference, severely restricting the system to higher-end computing hardware such as the NVIDIA Jetson AGX Orin. While transferring pre-trained and fine-tuned models onto the Orin would likely be straightforward, training and iterative updates could prove difficult or unreasonably time consuming without access to external GPU clusters or cloud infrastructure.

Despite the challenges of resource demands, limited documentation, and hardware-specific inference loop configuration, the OpenVLA model demonstrated the capacity to replace a large portion of system complexity by abstracting manual perception and task planning. In addition to removing the burden of manual perception and task planning, OpenVLA also delivered generalization and adaptability to unseen and changing environments. This benefit would likely prove invaluable in dynamic environments when implemented correctly.

5.2 Challenges and Issues Encountered

5.2.1 VLA System Setup Challenges

An early oversight in the setup process was selecting hardware without fully understanding GPU memory requirements dictated by model size. Contrary to initial assumptions, substantial GPU-accessible memory was essential not only for training but also for running inference. A model such as OpenVLA, containing approximately 7 billion parameters, necessitated substantial GPU memory resources for loading model weights during inference. Consequently, the chosen NVIDIA Jetson AGX Orin was essential, representing additional cost and system complexity compared to compute required for a deterministic system. Another concern is the lack of inherent task-completion detection mechanisms, as the OpenVLA model does not maintain contextual memory or state; consequently, without a custom-built method such as visual confirmation of a post-task state (e.g., moving a block into a designated area is complete), manual supervision, or integration of a vision-based monitoring system, the inference loop could continue indefinitely.

Custom ROS 2-Based Implementation

Although OpenVLA's generalized capabilities initially suggested minimal task-specific configuration, practical implementation revealed numerous complexities when attempting to build a custom solution. Precise calibration of camera-to-robot-base frame transformations, inference latency management, Cartesian space boundaries, rotation tolerances, and gripper management were all necessary for task success. Trial-and-error demonstrated that successful deployment required explicit alignment with the training dataset's arm configuration, as well as tailoring to the current operational environment. These challenges motivated the discovery and implementation of the adapted `bridge_data_robot` method.

bridge_data_robot

Adaptation Several challenges emerged when adapting the `bridge_data_robot` repository to implement a functioning VLA system. Although the vendor-recommended installation procedure involved Docker containers, incompatibilities between GPU usage within Docker and Jetpack 6 rendered this approach unfeasible. Furthermore, the repository's codebase was originally implemented in ROS 1, requiring substantial manual modifications to ensure compatibility within the ROS 2 experimental environment. This adaptation required significant manual effort and highlighted the absence of comprehensive, readily available frameworks or examples for deploying VLA systems with broad platform support.

5.2.2 Environment Setup Issues Common to Both Systems

For both setups, issues emerged while initially configuring the software environment. The Jetson AGX Orin came preinstalled with Jetpack 5, which proved incompatible with several of the latest essential software dependencies, including critical ROS 2 and GPU utilization libraries. This incompatibility forced an upgrade to Jetpack 6, requiring the non-trivial process of flashing the operating system.

Additionally, attempts to use Docker containers to standardize and simplify dependency management were ultimately unsuccessful. Docker proved brittle, particularly when managing direct hardware connections (such as camera or robot arm USB ports). Frequent hardware state changes (e.g., different port assignments, device identifiers) introduced instability and unpredictable behaviors in containerized environments. The complexity in implementing Docker

containers further increased when attempting to install and use GPU utilization software on Jetpack 6, undermining its usability in this robotic hardware deployment.

In another attempt to increase portability, running virtual machines was also deemed unworkable since Interbotix packages explicitly lack support for VM environments. ROS 2 officially provides robust Tier 1 support only for Ubuntu Linux distributions, complicating attempts to adapt or deploy on alternative operating systems. While ROS 2 provides acceptable Tier 1 support to Windows, limitations specific to Windows exist, reinforcing the necessity of Ubuntu Linux as the system of choice.

Another difficulty discovered was in the complexity of installing Interbotix packages with the provided installation scripts. Available scripts were explicitly written for AMD64 architectures or Raspberry Pi 4B ARM64 architectures. Despite the Jetson AGX Orin being ARM64-based, existing scripts required manual adaptation, as they were optimized solely for Raspberry Pi hardware. As a result, a custom installation process was required to install Interbotix packages on the Orin.

5.2.3 Dependency Management Complexities

Managing dependency relationships between software packages emerged as another critical source of complexity. Frequent version mismatches between software dependencies created vast overhead during initial environment setup and troubleshooting. Due to dependency conflicts encountered during initial setup, explicit version pinning was employed to ensure reliable software reproducibility. Given the reliance on Interbotix packages, outdated documentation or delayed updates in Interbotix packages complicated installation and demanded necessary adaptations.

Chapter 6

Results and Analysis

6.1 Results of Trials

This section presents both the quantitative and qualitative results from the pick-and-place trials for comparing the deterministic control logic and the VLA-based (`bridge_data_robot` adaptation) control policy. Across all measured performance metrics, including accuracy, repeatability, and cycle time, the deterministic control logic outperformed the VLA control policy. The deterministic system exhibited a high degree of precision, completing tasks with minimal deviation from the target with consistent execution times. In contrast, the VLA-based system showed greater variability in its actions, with longer cycle times and less accurate placements, driven largely by the probabilistic nature of the underlying foundation model. Nonetheless, the VLA system demonstrated some advantages in adaptability and error recovery in certain scenarios. The following subsections detail the quantitative performance results and expand on key qualitative observations and tradeoffs observed across both systems.

6.1.1 Quantitative Results of Pick-and-Place

Overall, the deterministic control system was notably more accurate and precise than the VLA-based system. The deterministic control system got within an average of 3.24 millimeters of the target, while the VLA-based system got only within an average of 54 millimeters. The respective bias vectors were very different in magnitude as a result.

The deterministic control system also completed tasks in much less time, with the VLA-based system taking on average 50 seconds longer than the deterministic system to complete the

task. The repeatability of the VLA-based model also suffered greatly; the repeatability radius was almost ten times that of the deterministic system.

It is important to note that all position measurements were performed manually using 1mm-resolution graph paper, with block centers marked and measured by hand. This introduces an estimated measurement uncertainty of approximately $\pm$1–2 mm per axis, due to the limitations of human estimation, visual alignment, and parallax. While this margin is small relative to the large differences observed between systems (e.g., 54 mm vs. 3 mm), it should be considered when interpreting fine-grained differences or near-threshold results.

Refer to Table 6.1 for the aggregated metric breakdowns for these trials. Refer to Appendix A for full trial data (Table A.1 and Table A.2).

Metric	**VLA Control Policy**		**Deterministic Control Logic**	
	x	**y**	**x**	**y**
Bias Vector (mm, mm)	-45.13	30.33	1.22	-3.00
Axis Repeatability (mm, mm)	16.14	27.35	0.42	4.47
Accuracy (mm)	54.37		3.24	
Repeatability Radius (mm)	95.271		13.48	
Mean Cycle Time (s)	64.22		14.09	

Table 6.1: Quantitative Comparison Between VLA-based and Deterministic Policies

6.1.2 Qualitative Results and Observations

During the trials, there were many qualitative considerations beyond the metrics reported above. While a custom VLA model was initially implemented (see Section 3.2.2) as a comparative baseline with identical node setup to the deterministic system, it ultimately failed to perform in any capacity and was excluded from formal analysis. This system is used as a comparison to contextualize the VLA system behavior, but the primary comparison should be between only the deterministic and VLA `bridge_data_robot` adapted systems in Section 3.2.3. Refer to Table 6.2 for a complete qualitative summary of these two systems.

Aspect	VLA-Based System (bridge_data _robot)	Deterministic System
Adaptability	Can generalize to a variety of prompts and objects; occasionally able to recover from failures.	Rigid behavior based on pre-defined logic; no capacity for recovery or adaptation.
Consistency	Low; probabilistic behavior leads to high variance in cycle time and placement.	High; consistent task execution when environment is unchanged.
Prompt Sensitivity	Highly sensitive to prompt wording; different prompts may cause vastly different behaviors.	Not applicable; uses fixed logic paths without language input.
Environmental Robustness	Some resilience to object variation, but still fails with certain colors or occlusions.	Brittle; small shifts in camera or object orientation require full recalibration.
Failure Modes	May recover from initial failure (e.g., Trial 7), but sometimes fails unpredictably or hesitates (e.g., Trial 8).	Fails deterministically if input or environment deviates; failure causes are predictable.
Accuracy and Cycle Time	Variable due to hesitation, retries, or unintended motion; sometimes inflates cycle times.	High accuracy and fast cycle time when calibrated; stable across trials.

Table 6.2: Qualitative Comparison of VLA-Based and Deterministic Systems

Qualitative Evaluation of Deterministic System

The deterministic control logic succeeded in 10 out of 10 trials with very strong performance. It demonstrated highly predictable behavior under ideal conditions, but one key limitation was its brittleness in the face of slight environmental variation. This brittleness came from the fact that the system depended on point cloud data from the depth camera, loaded fully before execution. These point clouds did not refresh live as the robot moved, instead relying on the positioning of the blocks at the start of the trial. So, if any unexpected events occurred such as the gripper pushing the block into a suboptimal position, the arm could not react.

Additionally, in some trial attempts, delays or gaps in this data due to camera latency of point cloud data caused the robot to fail to initiate movement and required full recalibration. As a result, in Figure 6.1 of an example deterministic control trial, we needed to rely on an external camera (rather than RealSense camera imagery itself) to document the trials.

The system's grasping accuracy was also highly sensitive to object orientation; even slight rotations or misalignment often caused the gripper to latch onto the edge of the block rather than centering on it, leading to imprecise placement during the drop phase.

Finally, any minor shift in the robot's base or the position of the vision system (e.g., from bumping the table or moving the camera slightly) required a complete manual recalibration to restore accuracy. These factors illustrate the deterministic system's fragility when applied outside tightly controlled environments, limiting its utility in dynamic or unstructured settings despite its otherwise consistent performance.

Qualitative Evaluation of VLA-Based System

The VLA-based robotic system, evaluated using the OpenVLA foundation model, successfully completed 9 out of 10 pick-and-place trials, demonstrating promising, but inconsistent, capabilities.

An important tradeoff emerged in the VLA system's inherent probabilistic nature, which led to noticeable variability in performance. Unlike the deterministic system, which exhibited stable, repeatable behavior, the VLA-based system showed considerable variance in accuracy, cycle times, and placement repeatability. In particular, some trials demonstrated extended hesitation or indecision during execution, impacting success. For example, in Trial 3, the robot grasped and lifted the object but delayed the drop action, moving erratically for several seconds on the paper before completion of the drop. This behavior inflated the cycle time and reduced overall placement accuracy.

The probabilistic properties of the VLA control policy did have beneficial outcomes in some trials, such as its ability to recover from failed actions. In Trial 7, after a poor grasp attempt, the model was able to reassess and retry, eventually succeeding with a corrected motion (but inflating the cycle time, similar to Trial 3). See Figure 6.2 for the video frames of this trial, captured from a separate camera that showed the nuance in movement more clearly than the RealSense camera rollout frames.

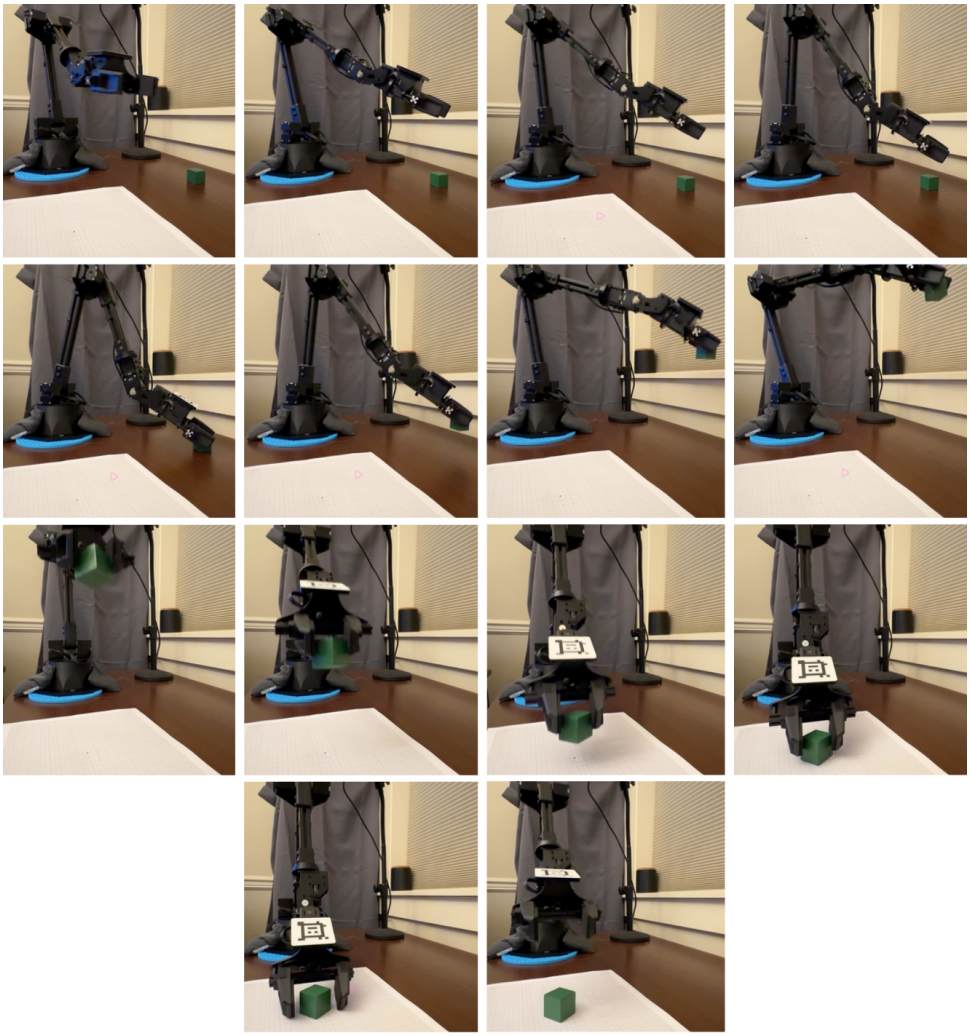

Figure 6.1: Frames from a successful trial of the deterministic control logic

However, other trials showed that this is not a reliable benefit because recovery is not guaranteed when movement is probabilistic. Trial 8, for example, failed to recover and complete the task despite multiple retries. The system falsely assumed it had successfully grasped the block and proceeded to open and close the gripper without any pattern, ultimately failing due to inability to detect state. This example highlights the VLA model's adaptive but non-deterministic behavior, which, while sometimes advantageous, can also result in unpredictable failure modes. See Figure 6.3 for frames of this trial.

Another key variable influencing performance was prompt sensitivity. Prior to final trials, some freeform experimental trials took place with varying prompts to find the most effective wording. The same task phrased with different prompts (e.g., *"Pick up block and place on center of white paper"* vs. "*Put block onto center of paper"* vs *"Put block onto paper dot"*) resulted

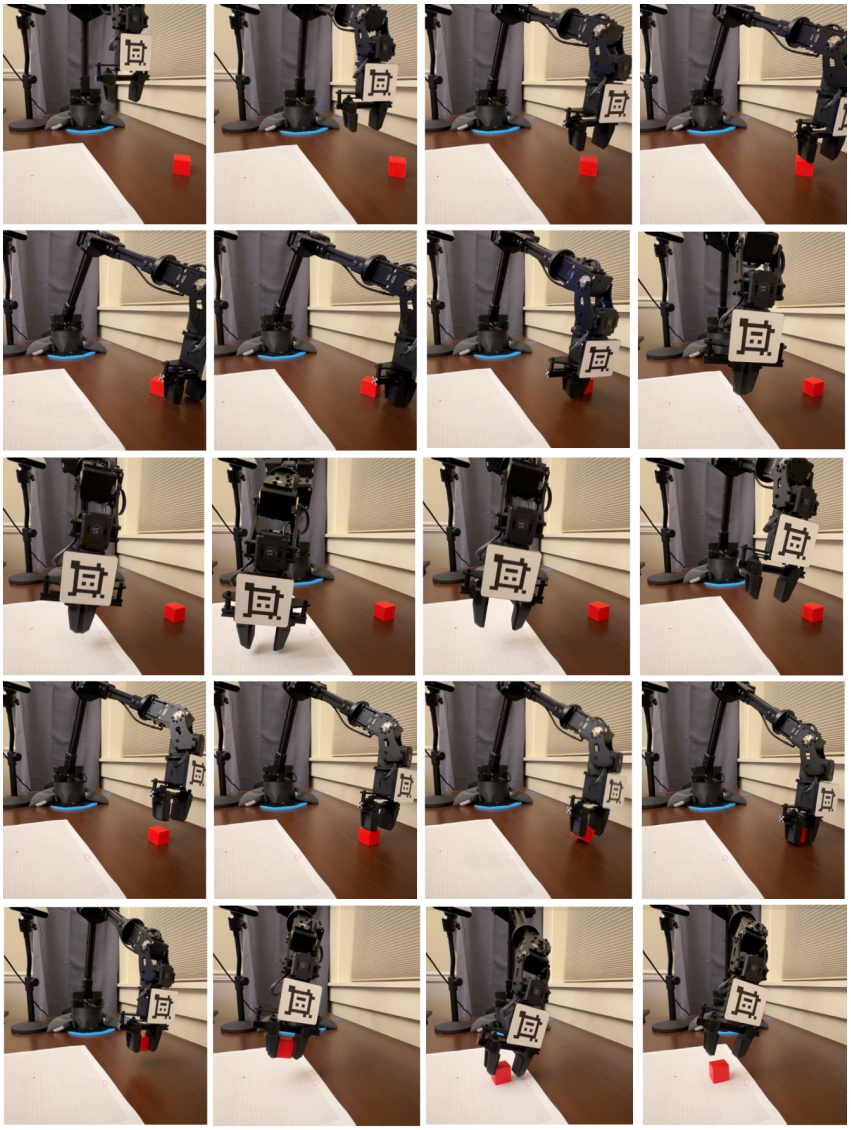

Figure 6.2: Video frames from a trial of the VLA policy, with full retry on block pickup

in widely varying robot behavior, as demonstrated in full trial accounts in Appendix A.3. *"Put block onto paper dot"* achieved the correct outcomes most often, while *"Pick up block and place on center of white paper"* produced hesitation, poor grasps, or total misinterpretation of the task.

6.2 Additional Considerations for VLA-Based System

While VLA models offer an abstraction for general-purpose robotic control, deploying them in real-world scenarios still requires substantial scaffolding and system-level constraints that diminish their advertised efficacy. As described in Section 3, two implementations were explored: a custom VLA setup designed to mimic the structure of the deterministic system, and the Bridge

Figure 6.3: Frames from a failed trial of the VLA policy

V2-based ROS 2 patch integrated with OpenVLA. The custom setup, which attempted a more direct zero-shot deployment without system tuning or controller design, failed to complete the task in any trials. In contrast, the `bridge_data_robot` adaptation, backed by heavy-weight control logic and trajectory handling from the tools written by the Bridge V2 dataset team [26], was able to perform the task more reliably, albeit with high variability in performance metrics.

A core limitation stemmed from the inherent need for a heavy-weight motion controller that could intelligently interpret and apply the model's delta predictions to real-world movements. Unlike deterministic policies, which produce pre-calibrated trajectory plans using known transforms and inverse kinematics, VLA-based outputs required post-processing and integration layers to achieve stability, task completion, and bounded movement. Simply feeding VLA action deltas into the Interbotix SDK functions for arm movement (i.e. the lightweight custom ROS nodes built during this research in Section 3.2.2) proved ineffective, because they lacked contextual tuning, trajectory smoothing, or certain fixed degrees of freedom (e.g. downward pitch at all times).

Furthermore, an important mismatch existed between the evaluation setup and the distribution of the model's training data. Foundation models like OpenVLA appear to have been trained primarily on examples where the end-effector is already close to the object at the time of inference as opposed to being in a neutral or sleep position away from the object. As a result, when starting from the default arm pose used in deterministic scripts (further from the object and less constrained), the model was prone to producing vague or unsafe motion sequences unless guided by additional restrictions. These limitations raise an important point: zero-shot may be possible in new scenarios but not without constraints enforced by setup dependencies and the architectural effort involved in making these systems work in practice.

6.3 Analysis of Results

6.3.1 Best Control Logic by Metrics

Quantitatively, the deterministic system outperformed the VLA-based system across all measured categories. It achieved a mean deviation of 3.24 mm from the target point, while the VLA-based system deviated by an average of 54.37 mm. Repeatability metrics followed a similar trend: axis repeatability and repeatability radius were tighter for the deterministic system (13.48 mm vs 95.27 mm), reflecting its ability to execute consistent movements when starting from a fixed initial pose. The deterministic system also demonstrated a far lower average cycle time (14.09 seconds), completing tasks more than 350% faster than the VLA-based system (64.22 seconds), which often hesitated, got stuck, or retried mid-task.

The deterministic system, while rigid, achieved high scores on both accuracy and repeatability. Its task success hinged on environmental calibration and consistent setup. When these conditions were met, the system was reliably precise. In contrast, the VLA-based system resulted in high variance between trials, leading to poor repeatability.

6.3.2 The Case for a VLA Policy Despite Poor Performance

The VLA-based system demonstrated task resilience: in some trials, it succeeded not through precision, but through retrying or adjusting its approach. This behavior suggests that while deterministic systems perform better when accuracy is paramount, VLA systems may have advantages in dynamic or less constrained environments, provided sufficient model improvements.

Unlike the deterministic control system, which either succeeded or failed based on strict conditions, the VLA-based system occasionally recovered from failed grasp attempts or adjusted its behavior based on the prompt. Taken together, these findings underscore the central tradeoff in adopting VLA-based control: greater flexibility and adaptability to dynamic environmental changes come at the cost of predictability, repeatability, and robustness.

However, the complexities currently involved in deploying VLA policies outweigh their benefits, particularly given their lower performance compared to deterministic solutions. Issues such as prompt sensitivity, GPU hardware constraints, limited integration support with widely used robotics hardware, and the absence of robust, production-ready implementations made the VLA-based system significantly more challenging to implement than deterministic control logic.

6.3.3 Final Verdict

From a task success standpoint, the deterministic system was the clear winner. It completed all 10 trials with high precision and consistent execution. The VLA-based system succeeded in 9 of 10 trials, but often with significantly degraded performance. The outcome of this experiment reinforces the view that deterministic systems can achieve millimeter-level precision, but VLA-based systems, while promising for future generalization and low-code setups, still currently require extensive support infrastructure and further refinement to match the effectiveness of deterministic control systems in structured tasks. Nonetheless, the adaptability observed in the VLA-based system, particularly when paired with more tuned prompts and supportive control architectures, points to a meaningful avenue for future research and development.

6.4 Implications to Hypotheses

These findings ultimately answer the initial research questions, as follows.

6.4.1 Hypothesis 1: Comparable Performance

The first hypothesis was: Robotic systems utilizing a VLA control policy will achieve comparable accuracy, repeatability, and cycle time to robotic systems utilizing deterministic control logic in tasks involving varied object arrangements.

There was no evidence to support this hypothesis through the experiment. The deterministic control logic outperformed the VLA-based system in all measured dimensions: it was

significantly faster, more accurate, and more repeatable. VLA control policies, though currently limited, may approach comparable performance with further tuning, larger models, and more targeted training data, as seen in emerging work like OpenVLA-OFT, a more complex model that recently emerged in April of 2025 [37].

6.4.2 Hypothesis 2: VLA Policy Resilience

The second hypothesis was: VLA models improve resilience by allowing the robotic system to effectively recover from unforeseen circumstances or failures encountered during execution.

This was partially supported by the experiment results. The VLA system showed the ability to recognize failure or pivot based on environmental scenarios, coming to a point of completion despite taking longer to get there. The primary implementation advantage of VLA models lies in their ability to generalize to natural language instructions and exhibit live adaptation to circumstance, whether that be noticing a failure and trying from scratch, or attempting an action again if something unexpected occurred initially (the block moved, gripper did not close, a barrier was reached, etc.). In this way, the VLA-based policy was able to interpret high-level goals and stay true to the initial instruction, offering a potential pathway to more flexible human-robot interaction in the future. These qualitative benefits provide partial support for Hypothesis 2, though they remain constrained by model limitations and environmental sensitivity.

6.4.3 Hypothesis 3: Integration Challenges

The third hypothesis was: Integration of VLA models introduces technical challenges, including increased inference latency, calibration complexity, and heightened computational resource requirements.

This hypothesis was strongly supported by the experimental results. Integrating the VLA model revealed critical technical hurdles in many aspects. First, inference latency was substantial, with OpenVLA-7B averaging over 60 seconds per task cycle due to model size, vision processing, and arm control frequency between steps. Calibration complexity also proved to be a major issue; the model's reliance on precise camera-to-robot transforms meant that even minor misalignments could lead to task failures, requiring repeated recalibration efforts.

Second, compatibility issues compounded these challenges. The primary reference implementation for OpenVLA, found in the Bridge V2 repository, was originally developed for ROS

1 and relied on a containerized, server-client architecture that did not adapt easily into the ROS 2 framework or the Jetpack 6.2 SDK. Adapting it for a modern ROS 2 environment required extensive patching, rewriting of launch files, and the creation of new interface layers to bridge between model output and Interbotix's ROS 2-compatible arm control packages. This was not straightforward, as many of the tools and configurations assumed ROS 1 behaviors (e.g., topic naming conventions, service handling), which are incompatible with ROS 2 out of the box.

Third, the VLA-based system setup is novel enough that robotics software, standard AI libraries, and new edge computing devices like the Jetson AGX Orin do not easily or natively support each other. Extensive patching and modification were required to adapt the Interbotix packages, RealSense drivers, perception pipelines, and execution code for compatibility with the Jetson AGX Orin, as most existing tools and documentation assumed traditional hardware setups, such as Raspberry Pi.

Finally, managing software dependencies for GPU acceleration, model loading, and camera interfacing presented obstacles, primarily due to the Jetson AGX Orin's ARM64-based architecture and the limited compatibility of its relatively new JetPack 6.2 software environment. Many software packages lacked ARM64 support, had minimal compatibility with JetPack 6.2, or included undocumented dependency mismatches. Docker-based approaches were challenging because pre-built ARM64-compatible containers supporting JetPack 6.2 were scarce, and Virtual Machine solutions were unsupported by Interbotix's open-source modules. Consequently, manual dependency management and custom patching were necessary to achieve compatibility.

Chapter 7

Insights and Recommendations

Based on the findings and detailed analysis presented in this thesis, this chapter offers practical insights and actionable recommendations aimed at improving the implementation process of VLA-based robotic systems. These recommendations directly address challenges and complexities encountered in real-world deployment scenarios, including model integration, dependency management, inference optimization, and calibration practices.

7.1 Simplifying Future VLA System Setup

7.1.1 Guidelines for Streamlined VLA System Design

Future implementations of VLA-based robotic systems would benefit greatly from robust frameworks or explicit guidelines for managing the inference loop cycle (image capture $\rightarrow$ inference $\rightarrow$ action decoding $\rightarrow$ robotic actuation $\rightarrow$ loop completion). Intelligent functions providing real-time feedback about optimal frequencies for inference and robotic control actions would greatly reduce manual configuration effort. During experiments, precise timing between image capture and actuation presented as a significant variable, as even minor state mismatches likely contributed to latency accumulation or degraded task performance. Synchronizing the inference-actuation loop with timing-aware logic or feedback-driven control thresholds could improve overall system stability.

Further, robotics software frameworks designed for modular "plug-and-play" integration of VLA components into existing robotic stacks would provide significant practical utility. Such

frameworks would allow engineers to incrementally integrate or replace deterministic perception and control modules with VLA models, enabling lower-risk, staged deployments.

7.1.2 Reducing Barriers to VLA Integration in Robotics

To encourage broader adoption beyond research environments, comprehensive documentation is required. Documentation that clearly explains how to integrate VLA models into robotics frameworks such as ROS 2 would dramatically lower adoption hurdles, simplifying both initial configuration and ongoing development. Additionally, clear guidelines outlining hardware constraints, resource requirements, and integration considerations as presented throughout this research would assist non-academic users in making informed decisions during system selection and setup.

Additionally, integration frameworks should abstract away common but tedious implementation details such as camera preprocessing, boundary adjustments, and prompt-template management. These aspects consumed significant development time in this project and remain largely unsupported by existing tooling.

7.2 Practical Considerations and Limitations

7.2.1 Performance Concerns

While VLA foundation models like OpenVLA-7B demonstrate promising flexibility and task generalization, they currently underperform deterministic systems in accuracy, repeatability, and execution speed. Practitioners should continue favoring deterministic methods unless their applications can tolerate higher error rates and longer cycle times, or they are prepared to invest in further optimization techniques and substantial fine-tuning. Fine-tuning remains essential for adapting VLA models to real-world robotic applications, particularly when system characteristics or tasks differ from those seen during the model's pretraining [4].

In addition to these performance limitations, experiments revealed sensitivity to language phrasing in prompts (e.g., "grab" vs. "pick up"). This sensitivity disproportionately affected success rates and indicates the need for prompt libraries, guidelines, or templated prompts to reduce variability in task execution and to increase awareness of the impact of prompt choices.

7.2.2 Operational Configuration Requirements

Reliable VLA deployment requires precise configuration of several control-layer parameters. These include defining workspace bounds in Cartesian space, initializing the end-effector to a known pose with a downward pitch for visibility, and rescaling model outputs, typically normalized action vectors, into real-world units. Per-step limits on translation (e.g., ±5 cm), rotation (e.g., ±0.25 radians), and gripper states help prevent unstable motions. Each action is clipped to remain within safe bounds before being converted to joint targets and published at low frequencies (e.g., 0.5 Hz) to allow for completion of previous steps. These constraints were found to be necessary for stable and consistent execution of VLA-generated actions.

7.2.3 Integration Cost and Tech Debt

Implementing VLA systems involves substantial integration and resource overhead, including GPU-based compute with high memory capacity, precise calibration, custom inference loop development, and robust robotic control logic. These challenges are neither trivial nor well-supported by existing development tools.

This project showed that even minor changes to configuration details—such as camera calibration or workspace setup—could result in significant performance degradation, despite using identical prompts and workspace boundaries. These fragile dependencies highlight the importance of incorporating mid-process diagnostics, such as analyzing image embeddings, attention maps, and confidence scores, into future debugging and system workflows.

Engineers should anticipate trial-and-error during integration, particularly in areas like prompt engineering, workspace configuration, and timing synchronization. Successful deployment also depends on access to suitable compute hardware and a strong understanding of the system's end-to-end behavior.

Chapter 8

Conclusion

This thesis evaluated the performance, implementation advantages, and integration complexities of VLA models relative to conventional deterministic robotic systems. Quantitative analysis clearly showed deterministic systems outperforming VLA-based systems in accuracy, repeatability, and execution speed. However, qualitative findings highlighted the VLA system's advantages in adaptability and task flexibility, traits that are absent in deterministic solutions.

In practical terms, deterministic methods remain superior for structured environments requiring precise, consistent, and rapid robotic manipulation. Conversely, despite current limitations, VLA models demonstrate clear promise in dynamic, exploratory contexts where adaptability and natural-language interaction are useful. This research emphasizes that significant enhancements in infrastructure, fine-tuning, and model architectures are still needed for VLA systems to achieve production-grade robustness. By providing a comparative real-world evaluation, this thesis addresses a gap beyond existing benchmark-driven studies.

8.1 Research Limitations

Several limitations of this research should be acknowledged. First, the experiments were confined to a single pick-and-place scenario with uniformly shaped and colored blocks, limiting the generalizability of the conclusions. Second, evaluations focused exclusively on one foundation model (OpenVLA-7B) and one deterministic control system, leaving room for broader comparative analyses with other emerging models and control approaches. Additionally, practical constraints, such as computational resources and inference latency, restricted the scope of experimentation. Although the NVIDIA AGX Orin and WidowX arm provided suitable research-grade

hardware, different hardware selections might yield variations in implementation complexity and performance outcomes.

8.2 Future Work

Future work should expand task complexity and diversity, including irregularly shaped objects, cluttered environments, and dynamic scenarios. Investigations employing optimized VLA models, such as the recent OpenVLA-OFT variant [37], or fine-tuned versions with environment-relevant training data, are strongly recommended to continue evaluating future efficacy.

Additional research into modular software frameworks designed specifically for VLA integration into robotic systems would greatly simplify adoption and provide clearer guidelines for overcoming implementation challenges. This research highlights both the current challenges and promising potential of VLA models, laying groundwork for future advancements in adaptable robotic systems.

Bibliography

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[3] D. Zhang, Y. Yu, J. Dong, C. Li, D. Su, C. Chu, and D. Yu, "Mm-llms: Recent advances in multimodal large language models," 2024. [Online]. Available: https://arxiv.org/abs/2401.13601

[4] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, and R. e. a. Rafailov, "Openvla: An open-source vision-language-action model," in *Proceedings of The 8th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, P. Agrawal, O. Kroemer, and W. Burgard, Eds., vol. 270. PMLR, 06–09 Nov 2025, pp. 2679–2713. [Online]. Available: https://proceedings.mlr.press/v270/kim25c.html

[5] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 3, 02 2001, pp. 2523 – 2528.

[6] L. Wan, "Robotic control systems and real-world challenges," *OpenThink*, Oct 2023. [Online]. Available: https://blogs.dal.ca/openthink/robotic-control-systems-and-real-world-challenges/

[7] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, and B. I. et al., "Palm-e: An embodied multimodal language model," 2023. [Online]. Available: https://arxiv.org/abs/2303.03378

[8] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone, "Libero: Benchmarking knowledge transfer for lifelong robot learning," 2023. [Online]. Available: https://arxiv.org/abs/2306.03310

[9] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, and S. von Arx et al., "On the opportunities and risks of foundation models," 2022. [Online]. Available: https://arxiv.org/abs/2108.07258

[10] M. Yang, J. Chen, Y. Zhang, J. Liu, J. Zhang, Q. Ma, H. Verma, Q. Zhang, M. Zhou, I. King, and R. Ying, "Low-rank adaptation for foundation models: A comprehensive review," 2024. [Online]. Available: https://arxiv.org/abs/2501.00365

[11] C. L. Goues, S. Elbaum, D. Anthony, Z. B. Celik, M. Castillo-Effen, N. Correll, P. Jamshidi, M. Quigley, T. Tabor, and Q. Zhu, "Software engineering for robotics: Future research directions; report from the 2023 workshop on software engineering for robotics," *Report from the 2023 Workshop on Software Engineering for Robotics*, 2024. [Online]. Available: https://arxiv.org/abs/2401.12317

[12] Misc., "Moveit 2 motion planning configuration," *Interbotix ROS2 Documentation*, 2024. [Online]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros2_packages/moveit_motion_planning_configuration.html?highlight=moveit

[13] ——, "Perception pipeline configuration," *Interbotix ROS2 Documentation*, 2024. [Online]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros2_packages/perception_pipeline_configuration.html?highlight=perception+pipeline

[14] A. Borad, "Understanding object localization with deep learning," *eInfochips*, Apr 2024. [Online]. Available: https://www.einfochips.com/blog/understanding-object-localization-with-deep-learning/

[15] K. Lynch and F. C. Park, *Modern Robotics Mechanics, planning, and control*. Cambridge University Press, 2024, pp. 1–10.

[16] M. Authors, "Depth sensing technologies," *FRAMOS*, Jul 2023. [Online]. Available: https://www.framos.com/en/products-solutions/3d-depth-sensing/depth-sensing-technologies

[17] K. Fang, K. Xu, Z. Wu, T. Huang, and Y. Yang, "Three-dimensional point cloud segmentation algorithm based on depth camera for large size model point cloud

unsupervised class segmentation," *Sensors*, vol. 24, no. 1, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/1/112

[18] H.-C. Kang, H.-N. Han, H.-C. Bae, M.-G. Kim, J.-Y. Son, and Y.-K. Kim, "Hsv color-space-based automated object localization for robot grasping without prior knowledge," *Applied Sciences*, vol. 11, no. 16, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/16/7593

[19] Misc., "Introduction to descartes path planning," in *ROS Industrial Training documentation*. ROS Industrial Consortium, 2017. [Online]. Available: https://industrial-training-master.readthedocs.io/en/melodic/_source/session4/Descartes-Path-Planning.html

[20] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, "Deep reinforcement learning for the control of robotic manipulation: A focussed mini-review," *Robotics*, vol. 10, no. 1, p. 22, Jan. 2021. [Online]. Available: http://dx.doi.org/10.3390/robotics10010022

[21] L. Palmieri, L. Bruns, M. Meurer, and K. O. Arras, "Dispertio: Optimal sampling for safe deterministic motion planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, p. 362–368, Apr. 2020. [Online]. Available: http://dx.doi.org/10.1109/LRA.2019.2958525

[22] R. Gayathri and V. Uma, "Performance analysis of robotic path planning algorithms in a deterministic environment," *International Journal of Imaging and Robotics*, vol. 19, no. 4, pp. 83–108, 2019.

[23] Y. Hu, Q. Xie, V. Jain, J. Francis, J. Patrikar, N. Keetha, S. Kim, Y. Xie, T. Zhang, H.-S. Fang, S. Zhao, S. Omidshafiei, D.-K. Kim, A. akbar Agha-mohammadi, K. Sycara, M. Johnson-Roberson, D. Batra, X. Wang, S. Scherer, C. Wang, Z. Kira, F. Xia, and Y. Bisk, "Toward general-purpose robots via foundation models: A survey and meta-analysis," 2024. [Online]. Available: https://arxiv.org/abs/2312.08782

[24] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," 2020. [Online]. Available: https://arxiv.org/abs/1907.03146

[25] A. O'Neill, A. Rehman, A. Gupta, A. Maddukuri, A. Gupta, and A. P. et al., "Open x-embodiment: Robotic learning datasets and rt-x models," 2024. [Online]. Available: https://arxiv.org/abs/2310.08864

[26] H. Walke, K. Black, A. Lee, M. J. Kim, M. Du, C. Zheng, T. Zhao, P. Hansen-Estruch, Q. Vuong, A. He, V. Myers, K. Fang, C. Finn, and S. Levine, "Bridgedata v2: A dataset for robot learning at scale," in *Conference on Robot Learning (CoRL)*, 2023. [Online]. Available: https://arxiv.org/abs/2308.12952

[27] R. B. A. V. Omar Aboul-Enein, Ya-Shian Li-Baboud, "Design and implementation of a closed-loop mobile manipulator control system, technical note (nist tn)," *National Institute of Standards and Technology*, 2023. [Online]. Available: https://doi.org/10.6028/NIST.TN.2258

[28] Misc., "Openvla – vision/language action models for embodied robotics," *NVIDIA Developer Blog*, 2025. [Online]. Available: https://www.jetson-ai-lab.com/openvla.html

[29] ——, *Manipulating industrial robots — Performance criteria and related test methods*, 2nd ed. International Organization for Standardization, 1998, vol. 9283.

[30] S. Ferraro, P. Mazzaglia, T. Verbelen, and B. Dhoedt, "Focus: object-centric world models for robotic manipulation," *Frontiers in Neurorobotics*, vol. Volume 19 - 2025, 2025. [Online]. Available: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2025.1585386

[31] A. Sirinterlikci, M. Tiryakioğlu, A. Bird, A. Harris, and K. Kweder, "Repeatability and accuracy of an industrial robot: Laboratory experience for a design of experiments course," *Technology Interface Journal*, vol. 9, 01 2009.

[32] A. Joubair, "What are accuracy and repeatability in industrial robots?" *Robotiq Blog*, 2014. [Online]. Available: https://blog.robotiq.com/what-are-accuracy-and-repeatability-in-industrial-robots

[33] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, 01 2009.

[34] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, and e. a. Yongqiang Dou, "Vima: General robot manipulation with multimodal prompts," 2023. [Online]. Available: https://arxiv.org/abs/2210.03094

[35] Misc., “Widowx-250 6dof — interbotix x-series arms documentation,” *Interbotix ROS2 Documentation*, 2024. [Online]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/specifications/wx250s.html

[36] ——, “Dynamixel workbench toolbox,” *Interbotix X-Series Arms Documentation*, 2024. [Online]. Available: https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros_interface/ros2/overview/dxl_wb.html

[37] M. J. Kim, C. Finn, and P. Liang, “Fine-tuning vision-language-action models: Optimizing speed and success,” *arXiv preprint arXiv:2502.19645*, 2025. [Online]. Available: https://arxiv.org/abs/2502.19645

[38] I. Robotics, “pick_place.py script for ros2 manipulation,” https://github.com/Interbotix/interbotix_ros_manipulators/blob/main/interbotix_ros_xsarms/interbotix_xsarm_perception/scripts/pick_place.py, 2024.

Appendix A

Raw Trial Data

A.1 Raw Data from Deterministic Trials

Trial	x (mm)	y (mm)	Cycle Time (s)	Block Color
1	1	-5	14.09	Red
2	1	-7	14.11	Orange
3	2	5	14.09	Yellow
4	2	-3	14.08	Green
5	1	-8	14.08	Blue
6	1	3	14.09	Purple
7	1	-2	14.09	Red
8	1	-7	14.09	Orange
9	1	0	14.08	Yellow
10	1	-6	14.09	Green

Table A.1: Deterministic System Trial Results

A.2 Raw Data from VLA-Based Trials

Trial	x	y	Cycle Time (s)	Block Color	Comments
1	-49	3	46.02	Red	Touched block at 15 seconds. Picked up at 24 seconds. Dropped at 46 seconds.
2	-54	24	41.85	Orange	Picked up at 21 seconds. Dropped at 41 seconds.
3	-35	43	99.23	Yellow	Picked up at 51 seconds (tried and failed to pick twice). Placed on paper at 81 seconds but picked it up again. Actual drop is 99 seconds.
4	-21	34	43.26	Green	Picked up block at 19 seconds. Dropped on paper at 43 seconds.
5	-69	92	52.03	Blue	Picked up at 27 seconds. Dropped at 52 seconds.
6	-28	5	65.45	Purple	Picked up at 22 seconds. Tried picking and placing many times (after initially getting it at 65 seconds). Moved the block a few mm in a circle while trying to place.
7	-65	29	114.07	Red	Tried again. Picked on the retry at 94 seconds. Dropped on paper at 114 seconds.
8	X	X	X	Orange	Failed to complete within 80 steps (160 seconds). Did not pick up block; gripper kept opening and closing. Went over to paper as if it had the block.
9	-40	36	50.47	Yellow	Picked up at 30 seconds. Dropped at 50 second mark.
10	-50	7	68.21	Green	Grabbed block at 19 seconds. Dropped at 38 seconds.

Table A.2: VLA-Based System Trial Results (Prompt: "Put block onto paper dot")

A.3 VLA-based Open-Ended Prompt Exploration (pre-trials)

Trial	x (mm)	y (mm)	Cycle Time (ms)	Block Color	Comments
1	X	X	X	Red	Prompt: "pick up block and place on center of white paper" Did very poorly. Found center of paper with gripper closed. Did not take the block with it.
2	X	X	X	Orange	Prompt: "pick up block and place on center of white paper" Also did very poorly. Found center of paper with gripper closed, but did not take the block with it.
3	X	X	X	Yellow	Prompt: "Pick up block" Was able to identify the block and attempt to pick it up. Did not succeed.
4	X	X	X	Green	Prompt: "Pick up block" Was able to identify the block and pick it up.
5	X	X	X	Green	Prompt: "Pick up block and place on paper" Was able to pick up block and place on paper (but gripper did not open).
6	X	X	X	Green	Prompt: "Put block onto center of paper" Picked up at 15 seconds. Did place on paper but gripper did not open.
7	X	X	X	Red	Prompt: "Place block onto center of paper" Picked up, placed on side of paper, but gripper did not open.

Trial	x (mm)	y (mm)	Cycle Time (ms)	Block Color	Comments
8	X	X	X	Red	Prompt: "Put block onto black dot" Wiggled around but ultimately did nothing.
9	X	X	X	Red	Prompt: "Put block onto black dot" Touched block at 44 seconds.
10	X	X	X	Green	Prompt: "Put block onto black dot" Picked block up at 29 seconds. Does not get to the paper (went ahead instead of towards paper) but did drop the block this time, around 56 seconds. Corrected mistake by trying to pick up again.
11	-56	0	43	Green	Prompt: "Put block onto center of paper" Picked up block at 18 seconds. Touched paper at 41 seconds, dropped at 43 seconds.
12	X	X	X	Green	Prompt: "Put block onto dot at center of paper" Grabbed block at 19 seconds. Dropped at 38 seconds (in vicinity of paper, but not close to block).
13	-50	7	68	Green	Prompt: "Put block onto paper dot" Tried to pick up but dropped at 20 seconds. Officially picked up at 40 seconds. Dropped on paper at 68 seconds.
14	-21	34	43	Green	Prompt: "Put block onto paper dot" Picked up block at 19 seconds. Dropped on paper at 43 seconds.

Trial	**x (mm)**	**y (mm)**	**Cycle Time (ms)**	**Block Color**	**Comments**
15	X	X	X	Orange	Prompt: "Put block onto paper dot" Thought it had the block when it went to the paper. Took 60 seconds for all the operations, but did not pick the block up. At 100 seconds it tried again and almost dropped it (but did not make it).

Table A.3: VLA Prompt Engineering Trials: Effects of Different Prompts on Grasping and Placement Behavior

Appendix B

Code

B.1 Deterministic Script: Interbotix Adaptation

```
    # Start up the API
    robot_startup(global_node)

    # set initial arm and gripper pose
    bot.arm.go_to_sleep_pose()
    bot.gripper.release()

    # get the ArmTag pose
    # armtag.find_ref_to_arm_base_transform()
    bot.arm.set_ee_pose_components(x=0.3, z=0.2)

    # get the cluster positions
    # sort them from max to min 'x' position w.r.t. the ARM_BASE_FRAME
    success, clusters = pcl.get_cluster_positions(
        ref_frame=ARM_BASE_FRAME,
        sort_axis='x',
        reverse=True
    )

    if success:
        # pick up all objects and set them
        # down at the specified goal point
        for cluster in clusters:
            x, y, z = cluster['position']
```

```
            print(x, y, z)

            # execution-time measurement starts
            start_time = time.perf_counter()

            bot.arm.set_ee_pose_components(x=x, y=y+0.015, z=z+0.15,
                pitch=0.5)
            bot.arm.set_ee_pose_components(x=x, y=y+0.015, z=z+0.022,
                pitch=0.5)
            bot.gripper.grasp()
            bot.arm.set_ee_pose_components(x=x, y=y, z=z+0.3, pitch
                =0.5)
            bot.arm.set_ee_pose_components(x=0.326, y=-0.013, z=z+0.3)
            bot.arm.set_ee_pose_components(x=0.326, y=-0.013, z=0.03,
                                           roll=0.021, pitch=0.5, yaw
                                              =-0.051)
            bot.gripper.release()
            bot.arm.set_ee_pose_components(x=0.326, y=-0.013, z=0.2)

            # execution-time measurement ends
            exec_ms = (time.perf_counter() - start_time) * 1000
            print(f'Execution time: {exec_ms:.1f} ms')
    else:
        print('Could not get cluster positions.')

    bot.arm.set_ee_pose_components(x=0.3, z=0.2)
    bot.arm.go_to_sleep_pose()
    robot_shutdown(global_node)
```

Listing B.1: Adapted Deterministic Pick and Place Script [38]

B.2 VLA Script: bridge_data_robot Adaptation

```
class GenerateConfig:
    model_family: str = "openvla" # Model family
    # Initial end-effector pose [x, y, z]
```

```
    init_ee_pos: List[float] = field(default_factory=lambda: [0.3,
        -0.09, 0.26])
    # Initial end-effector orientation [roll, pitch, yaw, gripper]
    init_ee_quat: List[float] = field(default_factory=lambda: [0,
        -0.259, 0, -0.966])
    # Workspace bounds [x, y, z, roll, pitch]
    bounds: List[List[float]] = field(default_factory=lambda: [
            [0.05, -0.25, -0.02, -1.57, 0],
            [0.50, 0.30, 0.35, 1.57, 0],
        ])
    camera_topics: List[Dict[str, str]] = field(default_factory=lambda:
         [{"name": "/camera/camera_stream/color/image_raw"}])
    blocking: bool = False # wait for motion to be complete before new
        movement command?
    max_episodes: int = 50 # Max number of episodes to run
    max_steps: int = 80 # Max number of timesteps per episode
    control_frequency: float = 0.5 # WidowX control frequency (set
        lower for smoother motion)

def evaluate_model_in_real_world(cfg: GenerateConfig) -> None:

    # [OpenVLA] Set action un-normalization key
    cfg.unnorm_key = "bridge_orig"
    model = get_model(cfg) # Load OpenVLA model
    # [OpenVLA] Get Hugging Face processor
    processor = get_processor(cfg)

    # Initialize the WidowX environment
    widowx_env = get_widowx_env(cfg, model)

    # Start evaluation
    task_label = ""
    episode_idx = 0
    while episode_idx < cfg.max_episodes:
        # Get task prompt from user
        task_label = get_next_task_label(task_label)

        # Reset environment
        reset_result = widowx_env.reset()
```

```
observation = reset_result

# Setup
step = 0
# Robot control frequency: time duration between control
    commands
step_duration = 1.0 / cfg.control_frequency
# Timer for episode elapsed time from first inference
episode_start_time = None

# Start episode
print(f"Starting␣episode␣{episode_idx+1}")
last_timestamp = time.time()
while step < cfg.max_steps:
    try:
        current_timestamp = time.time()
        # If the time between control commands is greater than
            step duration,
        # execute the control command
        if current_timestamp > last_timestamp + step_duration:
            last_timestamp = time.time()

            # Refresh the camera image and proprioceptive state
            observation = refresh_obs(observation, widowx_env)

            # Get preprocessed image
            observation["full_image"] = get_preprocessed_image(
                observation)

            # Start episode timer right before first inference
            if episode_start_time is None:
                episode_start_time = time.time()
                print("Starting␣episode␣timer")

            # Query model to get action
            action = get_action(
                cfg,
                model,
                observation,
```

```
                        task_label,
                        processor=processor,
                    )

                    # Execute action
                    step_result = widowx_env.step(action)
                    observation = step_result
                    step += 1

                    # Print elapsed time since first inference
                    if episode_start_time is not None:
                        elapsed_time = time.time() - episode_start_time
                        print(f"Episode elapsed time since first
                            inference: {elapsed_time:.2f} sec")

        # Continue to next episode automatically
        episode_idx += 1

if __name__ == "__main__":
    evaluate_model_in_real_world()
```

Listing B.2: Adapted VLA-Based Pick and Place Script [26]