ME5001 Final Project Report

Low-Cost Dual-Arm Teleoperation System for Imitation Learning and ACT-based Policy Deployment

Author: WU JIAWEI
Matric Number: A0304505R

Supervisor: Prof. Chew Chee Meng
Examiner: Prof. Marcelo H Ang Jr

Department of College Design and Engineering
Master of Mechanical Engineering

April 11, 2025

Abstract

Remote dual-arm manipulation systems play an increasingly important role in tasks requiring flexible coordination and precise control, especially in complex or remote environments. However, many existing solutions rely on high-cost hardware, complex teleoperation frameworks, and high-bandwidth feedback systems, making them difficult to deploy in scalable or low-resource settings.

This project presents the design and implementation of a low-cost, real-world dual-arm teleoperation platform based on the LeRobot robotic system. The system utilizes a leader-follower control structure, in which the operator manually manipulates the leader arm to demonstrate task trajectories, while the follower arm replicates these motions. To support future autonomous learning, a synchronized multimodal data collection system has been developed. It records RGB images from dual-view camera setups (top view and side view), along with joint states and time-stamped synchronization logs, at a fixed sampling frequency. This system enables high-quality dataset generation for imitation learning tasks.

The platform is designed to support the training of transformer-based imitation learning models, such as Action Chunking with Transformers (ACT), with the goal of enabling autonomous execution of manipulation tasks. While the training and evaluation of such models are planned for future work, the current focus has been on ensuring robust data capture, low-latency control response, and accurate trajectory synchronization between the dual arms.

The proposed system lays the groundwork for an extensible research framework that supports human-in-the-loop teaching, multimodal dataset collection, and vision-guided learning in real-world dual-arm robotic control. It provides a cost-effective and modular baseline for further experimentation in imitation learning, visual servoing, and task generalization across manipulation scenarios. Future work will focus on training imitation policies using collected data and evaluating the effectiveness of vision-based feedback in improving execution accuracy under different environmental conditions.

Contents

1. Introduction

1.1 Background and Motivation

With the rapid advancement of robotics, their applications have been extensively expanded across various real-world domains, including industrial manufacturing, remote maintenance, rehabilitation, and educational training. In particular, remote robotic operation in complex and dynamic environments has revealed the limitations of traditional control paradigms based on preprogrammed paths or automated motion planning, especially in terms of flexibility, responsiveness, and operational safety. Therefore, developing a **low-cost, highly generalizable, and portable teleoperation system** has become a critical challenge for enabling robots to transition from laboratory research to broad real-world deployment.

Existing commercial teleoperation systems often rely on high-end hardware infrastructures, such as virtual reality (VR) controllers, force-feedback devices, and motion capture systems. While these devices enable precise control and immersive experiences, their high cost, complex setup, and poor environmental adaptability hinder their scalability in domains such as education, home assistance, and light industrial applications. Furthermore, imitation learning (IL), a promising approach for enhancing robot generalization through human demonstrations, typically depends on datasets collected in laboratory conditions using expensive fixed-arm platforms, force-torque sensors, and specialized vision equipment. These constraints significantly limit the applicability of IL in open and unconstrained environments [1].

To overcome these challenges, researchers have begun exploring the **integration of low-cost handheld devices with visual servoing** in teleoperation. The *Mobile ALOHA* system marks a significant milestone in this direction. It uses simple handheld controllers for bimanual teleoperation and simultaneously records user actions and camera observations, enabling the training of imitation learning models under budget-constrained conditions. Mobile ALOHA has successfully demonstrated diverse mobile manipulation tasks such as cooking, cleaning, and door operation, showcasing a practical balance between system affordability and task complexity [2].

Building on this foundation, the *GELLO* system further enhances usability through 3D-printed ergonomic design, enabling non-expert users to efficiently collect high-quality demonstrations. Compared to traditional VR or space-mouse setups, GELLO exhibits higher motion consistency and control precision while lowering the barrier to data collection for imitation learning [1].

Meanwhile, **visual servoing** has become a key enabler in robust teleoperation systems. Notably, the *VITAL* system fuses visual observations with human corrective signals to support a human-in-the-loop learning paradigm. This approach improves robot perception in dynamic environments and enhances the stability and adaptability of imitation policies [3].

The continued evolution of the ALOHA framework has also inspired a new wave of low-cost dual-arm platforms, such as *Bunny-VisionPro* and *ACE*, which integrate hand pose estimation, visual feedback, and real-time error correction mechanisms. These platforms highlight the synergistic potential of vision-guided control and high-quality data acquisition for complex bimanual manipulation tasks [4].

Motivated by these advancements, this project aims to replicate and extend the Mobile ALOHA architecture using the *LeRobot* dual-arm platform, constructing an integrated system that combines **teleoperation, visual servoing, and imitation learning**. The system features a hand-

held controller and low-cost sensory modules on the operator side, paired with RGB camera modules for real-time perception of the target and environment. On the learning side, we explore joint encoding strategies for motion and vision data to improve generalization capabilities. Our ultimate goal is to enable autonomous task execution—such as grasping, sorting, and assembly—**without relying on expensive hardware**, thereby providing a practical solution for deploying teleoperated robots in open-world scenarios.

1.2 Problem Statement

Despite recent advances in teleoperation and imitation learning, the large-scale, low-barrier deployment of such systems in real-world environments remains a formidable challenge. These limitations primarily stem from four critical issues: **high hardware and deployment costs**, **limited policy generalization and stability**, **weak integration between perception and control**, and **the lack of a full-stack, deployable learning loop**.

First, **hardware cost and deployment complexity remain prohibitive**. Most current teleoperation systems rely on high-end equipment such as VR controllers, haptic feedback modules, or motion tracking systems. While these devices support high degrees of freedom and fine-grained control, their cost and setup complexity severely limit their applicability in education, home assistance, or small- to medium-scale industrial settings [1]. Although recent systems such as *Mobile ALOHA* [2] and *ACE* [4] have demonstrated the feasibility of using simplified handheld controllers and off-the-shelf visual sensors, challenges remain in cross-platform compatibility, fine manipulation, and trajectory precision.

Second, **existing imitation learning pipelines often struggle with generalization and policy robustness**. The prevailing paradigm follows a static, three-stage process—offline demonstration collection, centralized training, and post-deployment execution—which limits adaptability to changing environments, including novel objects, illumination variations, and spatial task shifts. This disconnect introduces covariate shift and *compounding error*: small deviations in early execution can cascade into task failure [5]. To mitigate this, VITAL [3] introduced a *human-in-the-loop visual correction framework*, enabling real-time error recovery and environmental adaptation. However, such architectures have yet to be adapted to low-cost, deployable robotic platforms.

Third, **current systems lack tightly coupled models between perception and control**. While visual information is indispensable in manipulation, many approaches treat vision as static input rather than temporally synchronized sensory feedback. This undercuts robustness in scenarios involving occlusion, distractors, or deformable objects. Zhang et al. argue that visual policies must encode high-dimensional *semantic-motion abstractions* to handle complex object interactions [6], while Wong et al. emphasize the need for *error-aware imitation policies* that adapt to state transitions arising from imperfect execution [7].

Finally, although open-source systems like *Mobile ALOHA* and *GELLO* have laid promising groundwork, most focus primarily on data collection and training, **lacking a deployable framework that completes the loop from perception to policy execution and evaluation**.In addition, a truly end-to-end pipeline—from demonstration to deployment—remains elusive, especially on affordable hardware suitable for education and research-scale environments.

In light of these limitations, this work proposes a **low-cost, vision-augmented, and generalizable closed-loop imitation learning system**, based on the *LeRobot* dual-arm platform. We design an integrated workflow encompassing handheld teleoperation, multi-modal data acquisi-

tion, transformer-based policy learning, and real-world deployment and evaluation. Our goal is to close the loop between *human demonstration, perception, learning, and control*, thereby enabling practical deployment of imitation learning methods in open-world, low-resource robotic applications.

1.3 Research Contributions

To address the deployment challenges, limited generalization, and perception-control decoupling in real-world imitation learning systems, we propose an integrated low-cost robotic learning platform based on the *LeRobot* dual-arm robot. Our work contributes across system development, multimodal data acquisition, policy training and deployment, and evaluation, with the following key technical contributions:

Development of a Low-Cost, Deployable Teleoperation Platform We design a complete teleoperation platform based on the *LeRobot Koch* dual-arm robot, enabling human demonstration, real-time control, and policy execution. The system adopts a leader-follower structure, and integrates RGB-D cameras for visual perception. Inspired by the Mobile ALOHA [2] and GELLO [1] architectures, the system significantly lowers the technical barrier for robotic learning research and facilitates low-cost deployment in education and lightweight industrial applications.

Design of a Multimodal Synchronized Data Acquisition Mechanism We implement a multimodal data collection pipeline that synchronizes RGB-D image frames, joint states, and control commands at a fixed sampling rate (20–30 Hz). Using ROS2's timestamping infrastructure and custom time labeling, we ensure precise temporal alignment across modalities, enabling efficient post-processing, annotation, and replay. This design is inspired by the asynchronous multimodal fusion approach used in VITAL [3], while extending it with real-time visual feedback logging to improve data semantic integrity.

Policy Training and Deployment with Action Chunking Transformer (ACT) We apply the Action Chunking Transformer (ACT) [8] architecture to jointly model vision-action sequences and enable multi-step policy prediction. The ACT model mitigates compounding errors through a chunk-based modeling approach, maintaining stable performance even under limited data regimes. The trained policy is successfully deployed on the real-world LeRobot platform and achieves a grasping success rate exceeding 90%, demonstrating its robustness and applicability in real environments.

Incorporation of Visual Servo Feedback for Online Error Compensation To improve the stability and adaptability of policy deployment, we integrate a vision-based servo control loop that enables real-time trajectory correction. By tracking object pose with a keypoint detection network and computing corrective actions through the inverse Jacobian, our system closes the control loop in the visual space. This implementation combines insights from stable dynamical visual servoing [9] and direct vision-based imitation strategies [10], improving robustness under disturbances such as occlusion or object shift.

Proposal of a Comprehensive Multi-Dimensional Evaluation Framework We construct a multi-dimensional evaluation pipeline for deployed policies, including success rate, trajectory deviation, and generalization robustness. Trajectory error is quantified using the Euclidean distance between predicted and actual end-effector paths, while generalization is assessed through task variations involving target object, lighting, and viewpoint. Our evaluation framework builds upon the Error-Aware Imitation Learning paradigm [7], offering a comprehensive analysis of policy reliability and resilience in diverse scenarios.

Demonstration of the Effectiveness of Vision-Modality Integration Through comparative experiments, we demonstrate that integrating visual input significantly enhances the policy's ability to generalize across spatial variations. Policies trained with joint-image observations outperform joint-only baselines in tasks involving varying object poses and viewpoints. These results align with prior findings from DepthPC [11] and Visual Imitation Made Easy [12], and provide a concrete foundation for future extensions using vision-conditioned policy frameworks such as Diffusion Policy [13].

In summary, these contributions establish a practical and extensible framework for vision-guided imitation learning on low-cost dual-arm platforms. By bridging teleoperation, multimodal data collection, transformer-based policy learning, and real-world deployment, our work offers a complete pipeline that supports reproducibility, generalization, and future extensions. In the following sections, we elaborate on the system architecture, data acquisition methodology, policy training procedures, deployment of training model and evaluation strategies that support these results.

2. Literature Review

2.1 Teleoperation Systems for Robot Learning

Imitation learning (IL) has emerged as a prominent approach for robotic policy acquisition, relying heavily on the availability of high-quality human demonstrations. As such, the design of teleoperation systems—serving as the primary human-robot interaction interface—plays a critical role in determining the quality, consistency, and expressiveness of demonstration data. Over recent years, teleoperation frameworks have transitioned from high-precision, lab-constrained systems to modular, low-cost, and deployable platforms. This evolution offers key insights into the design of scalable learning systems.

High-Fidelity Systems: Precision-Centric Architectures

Early-generation teleoperation systems primarily relied on sophisticated interfaces such as force-feedback haptic devices (e.g., PHANTOM Omni), VR controllers (e.g., HTC Vive), and motion capture suites (e.g., OptiTrack). These platforms supported fine-grained control and physically grounded interaction, particularly for surgical robotics or micro-manipulation tasks such as the dVRK system.

However, such systems are expensive, require complex setup, and present steep learning curves for non-expert users. Cognitive load increases drastically in dual-arm or mobile manipulation tasks, which often leads to reduced demonstration consistency and degraded control performance [14].

Lightweight Systems: Toward Accessible and Modular Control

Recent works have focused on building **low-cost and user-friendly teleoperation frameworks** to democratize data collection for robot learning:

- **Mobile ALOHA** leverages joystick-based bimanual control with vision feedback for remote mobile manipulation tasks [2].
- **GELLO** introduces a 3D-printed, human-intuitive controller to replicate robot kinematics, enhancing demonstration consistency while minimizing hardware costs [1].
- **ACE** proposes a vision-based exoskeleton glove that supports a range of gripper types and maintains high generalizability [4].
- **VITAL** incorporates real-time visual feedback and human-in-the-loop correction to create closed-loop training conditions [3].
- **TeleMoMa** enables multi-modal control inputs (keyboard, joystick, VR) through modular APIs, emphasizing universality and ease of use [15].
- **MoMa-Teleop** partially automates low-level control (e.g., navigation via RL), allowing the operator to focus on high-level end-effector tasks [16].

- **RoboTurk** adopts a web-based interface using mobile devices, demonstrating the feasibility of large-scale cloud teleoperation using consumer hardware [17].

These systems represent an important shift toward "affordable + deployable + general-purpose" control paradigms. However, many of them still rely on high-throughput vision pipelines, non-standard communication protocols, or platform-specific ROS stacks, limiting their robustness and portability.

Our System Design: Lightweight, Flexible, and Deployable

Motivated by the above trends, we designed a **lightweight, modular teleoperation system** tailored for real-world, vision-guided imitation learning tasks. The core features of our system are:

- **Leader-Follower Architecture with Minimal Communication Overhead**: We adopt a master-slave control scheme using a USB game controller mapped directly to the LeRobot dual-arm interface. The system does not require VR hardware or ROS middleware, providing low-cost deployment and intuitive control for educational and research settings.
- **Multimodal Data Acquisition with Time-Synchronized Visual and Kinematic Streams**: We employ dual-camera viewpoints (top-down and side) to capture both global task layout and local interaction details. All sensor data (RGB-D frames, joint states, user inputs) are synchronized with local timestamps, ensuring alignment for subsequent training and evaluation.
- **Phase-Wise Vision Integration Strategy for Progressive Policy Fusion**: We first train joint-only ACT models to verify the learnability and generalization of low-dimensional control. Subsequently, image features (e.g., object keypoints) are introduced to support visual servoing and enhance robustness, enabling a smooth transition from action-centric to perception-driven policies.
- **ROS-Free Architecture for Portability and Low Latency**: All control and logging modules are implemented using lightweight custom code, with local queues and threaded execution, eliminating dependency on ROS. This greatly improves the system's portability across platforms and reduces runtime latency.
- **Validated on Real Hardware for Multi-Position Grasping Tasks**: The system has been deployed on a real-world LeRobot platform, executing trained ACT policies across various static grasping scenarios. Despite lacking force sensors or tactile feedback, the system demonstrates stable and reproducible control.

System Comparison and Positioning

Table 1: Comparison of Teleoperation Systems for Robot Learning

System	Control	Vision	Feedback	Complexity	Deployability	Ref
Mobile ALOHA	Joystick + video	Yes	No	High	Medium	[2]
GELLO	3D controller	No	No	Medium	Medium	[1]
VITAL	Human correction	Yes	Yes	High	Medium	[3]
TeleMoMa	Multi-modal input	Optional	No	High	High	[15]
This work	Synchronous control + RGB-D (staged)	Yes	Yes (light)	Low	High	[18]

In summary, recent developments in teleoperation systems reveal a clear shift toward cost-effective, flexible, and scalable architectures. While high-fidelity systems provide fine-grained control, their complexity and cost limit real-world applicability. In contrast, our proposed system emphasizes simplicity, modularity, and ease of deployment—traits that are essential for scalable data-driven learning. Crucially, the effectiveness of any teleoperation framework for imitation learning also depends on the quality and synchronization of the collected multimodal data, which motivates the discussion in the next subsection.

2.2 Multimodal Data Collection and Synchronization

The performance of imitation learning (IL) systems is fundamentally constrained by the **quality, structure, and temporal alignment** of the demonstration data. In real-world conditions, mismatches between image frames, control actions, and intention signals frequently lead to *semantic drift*, inconsistent supervision, and degraded policy generalization. Hence, building a **synchronized, structured, and semantically aligned data pipeline** is a critical prerequisite for robust and deployable imitation learning.

Key Challenges in Multimodal Synchronization

- **Temporal Misalignment between Perception and Action:** Vision and action data often operate at different frequencies—images typically stream at 10–30 Hz, while robot joint states and control commands update at much higher rates. Without a shared temporal reference, this leads to *perceptual-action dissonance*, where an image no longer accurately represents the robot's decision context. GELLO highlighted this challenge as a major factor in inconsistent demonstrations [1].

- **Semantic Gaps between Modalities:** Tasks such as bottle twisting or screwing show minimal visual change despite large control variations, while others exhibit significant visual changes with minor control input. This problem of *semantic modality mismatch* has been emphasized in large-scale datasets like Open X-Embodiment [19].
- **Real-Time Synchronization and Buffer Management:** VITAL addressed synchronization by employing asynchronous buffers and thread-managed schedulers to align images, joint states, and intervention signals [3].
- **Structured Representation Constraints:** XBG demonstrated that structured sliding-window encodings of image-joint sequences outperform flat inputs in multimodal policy learning.

Our System Design and Synchronization Strategy

To address these challenges, we develop a **lightweight, structured, and synchronized** multimodal data acquisition system tailored for vision-conditioned robotic policy learning. Its key components are:

- **Centralized Scheduler and Single-Loop Synchronization:** All sensory streams—including camera frames, joint states, and user input—are governed by a shared main thread. This avoids asynchronous race conditions and inter-frame jitter.
- **Dual-Camera Setup with Image-Triggered Sampling:** Two fixed RGB cameras (top-view and side-view) capture layout and manipulation perspectives. Image capture triggers joint-state logging for *event-level coupling*.
- **Raw Data Logging and Structured Bundling:** We store raw inputs and assemble structured data units (e.g., image-joint pairs, time-aligned bundles), facilitating efficient annotation, augmentation, and sequence learning.
- **ROS-Free Lightweight Infrastructure:** Instead of ROS, the system uses native scripts for device control, reducing latency, buffer overflow, and jitter during high-frequency operation.

Coupling of Multimodal Inputs and Policy Design

The structure of demonstration data directly affects model design. We adopt a **phased fusion strategy**:

- **Phase I: Joint-Only Policy Learning:** We first train policies using only joint states to evaluate baseline generalization.
- **Phase II: Vision-Augmented Fusion:** Vision signals (e.g., keypoints, RGB patches) are added to improve robustness. Evaluation spans object pose, lighting, and viewpoint variations.
- **Phase III: Redundancy-Aware and Masked Learning (Future work):** Drawing on Masked Imitation Learning [20], we assess the contribution of each modality through masking and ablation, leading to efficient multimodal fusion.

Although we only compelete Phase 1 & 2, our system delivers structured, low-latency, and semantically aligned multimodal data for robotic imitation learning. Unlike prior pipelines with undefined fusion boundaries, we create a clear *perception-control-learning* interface, supporting scalable and generalizable policy training.

2.3 Imitation Learning Policies

2.3.1 Why Imitation Learning?

A central question in robotic intelligence is:

> *How can we enable robots to perform purposeful and generalizable tasks in unstructured, uncertain, and dynamic real-world environments?*

To this end, multiple learning paradigms have been explored, most notably Reinforcement Learning (RL) and Imitation Learning (IL). While RL has made remarkable progress in simulation and theoretical AI, its practical applicability to real-world robotic systems remains limited. In contrast, IL has emerged as a more efficient, deployable, and stable framework, especially suited for service, educational, and industrial robotics.

Theoretical Strengths, Practical Limitations: Why Not RL? RL is fundamentally designed for interactive agents maximizing cumulative reward. While theoretically optimal, RL suffers from several practical bottlenecks when applied to physical robot systems:

- **Reward Engineering is Fragile:** Designing dense and informative reward functions for multi-stage tasks like grasping, rotation, and assembly is extremely difficult. Even slight reward misalignment can lead to unintended behaviors [21].
- **Low Sample Efficiency:** RL often requires millions of environment interactions to converge to a usable policy. This is impractical on real hardware due to execution time, system wear-and-tear, and safety risks [22].
- **Unstable Learning and Safety Risks:** RL policies are sensitive to initialization, disturbances, and training parameters. They may collapse during learning or fail abruptly at deployment, compromising system safety [23].
- **Strong Environment Dependency:** RL typically requires resettable environments or high-fidelity simulators. Real-world resets are hard to implement, and sim-to-real transfer often leads to performance degradation.

Conclusion: RL is powerful in simulation but fails to close the engineering loop in scenarios that demand low interaction cost, explicit task structure, and robust deployment.

Imitation Learning: Efficient, Controllable, and Deployable Imitation Learning (IL) offers a supervised alternative by learning policies directly from expert demonstrations, bypassing the need for reward signals or environmental exploration.

Key advantages of IL include:

- No Reward Function Needed — IL eliminates the need for hand-tuned reward design, learning directly from human behavior.
- High Sample Efficiency — IL policies can be trained with only a few minutes of demonstration data, in contrast to RL's need for millions of samples [24].
- Stable and Predictable Training — Based on supervised learning, IL models are well-behaved, robust to noise, and easy to validate—ideal for hardware-constrained systems.
- Modality-Agnostic Integration — IL naturally supports multimodal inputs (vision, language, proprioception) and human-in-the-loop corrections, as seen in systems like VITAL [3].
- Fast Deployment and Domain Transfer — IL policies generalize well across platforms and tasks, making them ideal for rapid prototyping and deployment.

Table 2: RL vs. IL: A Three-Dimensional Engineering Comparison

Dimension	**Reinforcement Learning (RL)**	**Imitation Learning (IL)**
Learning Mode	Reward-driven exploration	Supervised behavior cloning
System Compatibility	Requires resets/simulators	Compatible with offline data
Task Alignment	Long-horizon, ambiguous goals	Structured, goal-directed tasks
Deployment Complexity	High (hardware/safety/compute)	Low (fast training/inference)
Sample Efficiency	Low	High

Why IL Aligns with Our System Objectives This project aims to develop a dual-arm manipulation system that is efficient to train, easy to deploy, and compatible with multimodal perception. The target tasks—e.g., static-object grasping, block stacking—are short-horizon, goal-explicit, and sensor-structured, making IL particularly suitable.

IL is well aligned with our architecture in the following ways:

- Natural Integration with Leader-Follower Teleoperation — Human-controlled demonstrations directly produce training data without extra instrumentation.
- Compatible with Pre-Defined Tasks — IL does not require on-policy data collection, making it ideal for scripted or repetitive tasks.
- Modality Scalability — Supports phase-wise training: starting with joint-state-only policies, then extending to vision-based inputs.
- Low Hardware Requirement — IL models can be deployed on CPUs or edge devices without GPU acceleration.

- Supports End-to-End Experimentation — The data collection → training → deployment pipeline can be closed within a real-world setup.

Summary: While Reinforcement Learning offers theoretical optimality, its real-world applicability is severely limited by interaction cost, reward design, instability, and sim-to-real issues. In contrast, Imitation Learning provides a practical, robust, and efficient solution, especially for robot learning systems that require quick prototyping, safe execution, and flexible integration of human demonstration. Within the context of this research—where multimodal data collection, real-time deployment, and structured task execution are key—IL is not just a viable choice; it is the necessary one.

2.3.2 Behavior Cloning: From Foundations to Frontiers

Behavior Cloning (BC) is one of the earliest and most widely adopted imitation learning approaches. At its core, BC formulates policy learning as a **supervised regression problem**, mapping states s_t to actions a_t by minimizing the prediction error between a learned policy and expert demonstrations. Unlike reinforcement learning (RL), BC requires neither reward engineering nor interactive rollouts, making it structurally simple, data-efficient, and straightforward to deploy.

Due to these properties, BC is often used as:

- A **baseline** for early-stage policy validation;
- The foundation for training **large-scale behavioral datasets** (e.g., Open X-Embodiment);
- An **initialization or auxiliary loss** module in more advanced frameworks like Diffusion Policies or Residual IL.

Despite its simplicity, BC remains highly relevant in modern robotics due to its compatibility with **multimodal inputs**, **low-latency inference**, and **hardware-friendly deployment**.

The Covariate Shift Challenge The primary challenge in BC lies in *covariate shift*: the policy is only trained on the expert trajectory distribution, but during deployment, even slight deviations can lead the robot into *out-of-distribution (OOD) states*. Without a recovery mechanism, this causes **compounding errors**, where incorrect predictions accumulate and destabilize the trajectory.

This issue becomes more severe in long-horizon tasks such as stacking, remote picking, or obstacle avoidance. The policy may fail to recover once it leaves the expert manifold, leading to a phenomenon known as *policy collapse*.

Table 3: Covariate Shift in Behavior Cloning: Manifestation and Mitigation

Level	**Problem Manifestation**	**Root Cause**	**Mitigation Strategy**
Data-level	OOD states → inaccurate actions	Narrow demonstration coverage	Data diversification, PU filtering
Prediction-level	Small error → drift	Weak boundary generalization	Horizon-aware training, adaptive loss
Behavior-level	Divergence → mission failure	Lack of self-correction	Residual policy, human feedback

Recent Advances: From Pure Supervision to Robust Structure To overcome BC's inherent limitations, researchers have explored several directions for **enhancing robustness, generalization, and structure-awareness**:

- **Positive-Unlabeled Behavior Cloning (PU-BC):** Filters high-risk states and focuses learning on reliable demonstrations [25].
- **Preference-Driven BC:** Incorporates pairwise human preferences (e.g., FDPP) to guide the policy toward better-than-average behavior [26].
- **Residual and Hybrid IL:** Uses BC as a warm-start, then applies RL finetuning or correction layers [27].
- **Structural Regularization and Memory-Augmented BC:** Introduces memory banks or trajectory-consistency losses to enforce temporal stability.

These innovations reflect a shift from naive regression toward *behaviorally grounded, structure-aware policy learning*.

Role of BC in Our Framework In our project, BC plays a foundational role. While its limitations are well-recognized, it remains an **essential baseline for validating input encodings, model capacity, and data quality**.

Specifically, we employ BC to:

- Train joint-only and joint+vision policies as baselines;
- Compare execution robustness, trajectory smoothness, and viewpoint generalization;
- Serve as a precursor to the ACT transformer, which builds on BC's strengths;
- Validate our synchronized data pipeline in a lightweight inference setup.

Summary: Behavior Cloning remains a cornerstone of imitation learning due to its simplicity, efficiency, and broad applicability. While vulnerable to covariate shift in high-stakes scenarios, ongoing advances have significantly enhanced its robustness. BC thus serves as both a *benchmark and a building block* for the advanced policy models adopted in this work.

2.3.3 ACT vs. Diffusion Policy: Sequence vs. Sample

As robotic manipulation tasks become increasingly complex and temporally extended, simple behavioral cloning approaches often fail to capture long-range dependencies and structured transitions. In response, two advanced paradigms have emerged in imitation learning:

1. **Structured modeling strategies** (e.g., ACT), which emphasize *task-phase segmentation* and *temporal abstraction*;
2. **Generative modeling strategies** (e.g., Diffusion Policies), which focus on *distributional modeling* of action trajectories under diverse multimodal conditions.

Although both frameworks may employ Transformers as the sequence backbone, their core **design principles**, **model assumptions**, and **deployment suitability** diverge significantly.

ACT: Structure-Aware Policy Learning

Action Chunking with Transformers (ACT) [8] addresses the difficulty of modeling long-horizon manipulation by dividing action sequences into *semantic chunks*, such as reach–grasp–lift–place. Each chunk is treated as a phase-specific behavior module, allowing both local and global dependencies to be captured.

- **Chunk-level abstraction**: Encodes structured units of motion with high-level temporal semantics;
- **Hierarchical modeling**: Combines MLP/RNN for intra-chunk dynamics with a Transformer for inter-chunk sequencing;
- **Direct, single-step inference**: Enables low-latency prediction without sequential sampling.

Strengths:

- Aligns with **natural task decomposition** (e.g., assembly, pick-and-place);
- Fast and deployable in embedded systems;
- Flexible in integrating vision and state inputs through tokenized embeddings;
- Easily extendable to dual-arm coordination (e.g., Bi-ACT, HACT-Vq).

Diffusion Policies: Distributional Trajectory Generation

Diffusion Policy (DP) [13] adapts ideas from generative denoising models to robot action generation. Here, actions are modeled as noisy trajectories that are progressively refined through conditional sampling based on visual and proprioceptive context.

- **Gaussian noise initialization**: Begins from random trajectories;
- **Multi-step denoising**: Iteratively reconstructs expert-like behaviors using a learned reverse process;
- **Conditioned sampling**: Visual and state inputs guide the trajectory reconstruction.

Advantages:

- Learns **multi-modal behavioral distributions**, capturing diverse execution styles;
- Scales to complex sensory inputs (images, point clouds, language);
- Can incorporate reward-free optimization, human preferences, or hybrid training).

Limitations:

- **High inference latency** due to iterative sampling steps;

- **High system cost** (GPU + multi-stage scheduler);
- **Limited interpretability** and debugging difficulty in real-world failures.

Table 4: ACT vs. Diffusion Policy: Comparison of Structured vs. Generative Modeling

Dimension	**ACT (Structured Modeling)**	**Diffusion Policy (Generative Modeling)**
Objective	Learn task phases and transitions	Learn trajectory distributions
Modality Integration	Joint/image token embedding	Conditional input sampling
Training Regime	Supervised (MSE/classification)	Multi-step denoising
Inference Cost	Low (single-step)	High (iterative sampling)
Deployment Feasibility	Real-time, CPU-friendly	GPU-required, high latency
Interpretability	High (semantic chunks)	Low (black-box)
Notable Works	ACT, Bi-ACT, HACT-Vq	DP, FDPP, HDP, ResiP

Why We Choose ACT for Our System Given the system constraints and application goals of this project—low-latency control, limited hardware resources, and multimodal data inputs—**ACT is highly aligned with our deployment and modeling needs**:

- Leverages chunked datasets derived from image-joint synchronization;
- Supports real-time inference without recursive sampling;
- Enables seamless integration of vision-guided servoing modules;
- Facilitates policy generalization and multi-task scalability through structured design;
- Allows for modular evaluation of perceptual contribution via ablation studies.

In contrast, while Diffusion Policies excel at modeling trajectory richness and diversity, their high computational demands, complex parameter tuning, and poor interpretability pose significant barriers for on-device, closed-loop deployment in our real-world robotic platform.

Summary ACT and Diffusion Policies represent two advanced directions in imitation learning: *structure-aware vs. generative modeling*.

While Diffusion Policies push the frontier in learning from rich, diverse datasets, their inference bottlenecks limit their practicality in many real-time scenarios.

In contrast, **ACT offers a principled, scalable, and efficient strategy** for robotic manipulation tasks—especially when system resources, interpretability, and deployability are critical, as in this study.

2.3.4 Integrating Vision and State for Multimodal Imitation Learning

In early imitation learning systems, policies were typically trained on **low-dimensional proprioceptive inputs**, such as joint positions, velocities, and end-effector poses. These *joint-only policies* are advantageous for their simplicity, stability, and compatibility with robot control interfaces. They require no synchronization between sensory modalities and are lightweight to train and deploy.

However, this simplicity comes at the cost of **limited perceptual awareness**. Joint-only policies are inherently blind to external context and thus struggle in the following scenarios:

- **Poor spatial generalization**: Minor variations in object positions can lead to task failure;
- **Lack of semantic understanding**: Policies cannot distinguish between target objects or interpret visual goals;
- **Sensitivity to uncertainty**: Unable to adapt to occlusion, lighting shifts, or cluttered environments.

For instance, a joint-only policy for needle insertion failed almost entirely when the needle-hole position was slightly perturbed—highlighting the fragility of such models in high-precision manipulation tasks.

As robotic systems are deployed in more complex, dynamic settings, **pure control-driven policies are insufficient**. The field is thus rapidly shifting toward *perception-driven, multimodal policy architectures*.

The Shift to Multimodal Integration A growing body of work demonstrates that **environmental perception is key to robust policy generalization**. Integrating **visual modalities** (RGB, depth, segmentation) allows policies to reason about object location, scene layout, and semantic context.

Table 5: Representative Vision+State Multimodal Policy Systems

System	Input Modalities	Core Mechanism	Generalization Benefit
VIOLA [28]	Joint + RGB + proposals	Attention-guided vision	Robust to occlusion
Open X-Embodiment [29]	Image + State + Language	Multi-platform dataset	Transfer across tasks
RoboFlamingo [30]	Joint + Image + Text	Prompt-conditioned control	Fast task adaptation
X-IRL	Joint + Depth + Latent visuals	Contrastive latent fusion	Robust to unpaired data

These systems reflect a broader trend: **shifting from joint-only to vision+state paradigms**, enabling policies to move beyond fixed environments toward *semantically grounded and spatially adaptive behaviors*.

Core Challenges in Multimodal Fusion Despite its promise, multimodal policy learning presents several modeling and deployment challenges:

- **Temporal misalignment**: Vision and joint data often have mismatched frequencies;
- **Semantic dissonance**: Differences in modality representations can cause conflicting gradients;
- **Increased complexity**: Fusion modules demand more compute and memory;
- **Perception fragility**: Policies can fail under occlusion, lighting change, or domain shift.

To address these, researchers have proposed:

- Late fusion and multi-stage alignment;
- Semantic-guided attention (e.g., VIOLA [28]);
- Masked learning strategies (e.g., Masked IL [20]);
- Auxiliary consistency losses (e.g., state-prediction loss).

Our System Design: A Phased Fusion Strategy To balance interpretability, stability, and expressiveness, we adopt a **stage-wise integration of visual modalities** into our policy pipeline:

- **Phase I: Joint-only ACT** – Train the ACT model solely on proprioceptive inputs; – Establish chunk-level structure and benchmark baselines.
- **Phase II: Vision + State ACT** – Integrate top-view and side-view RGB via CNN encoders; – Fuse with joint tokens in ACT Transformer; – Evaluate spatial generalization and object robustness.

To sum up, Imitation learning is shifting **from control-centric to perception-aware policy design**. The evolution from joint-only to vision+state policies reflects a fundamental increase in agent capability to *understand, adapt, and generalize*. While multimodal fusion introduces complexity, it is becoming essential for real-world robotic autonomy. Our **phased fusion strategy** provides a modular and resource-aware pathway toward scalable, visual imitation learning.

2.3.5 Summary of Imitation Learning Policies

Imitation Learning (IL), a paradigm centered on learning from human demonstrations, has emerged as one of the most practical and scalable approaches in robotic policy modeling. Compared to Reinforcement Learning (RL), IL exhibits distinct advantages in training efficiency, deployment simplicity, and safety, especially in real-world scenarios where reward engineering is infeasible, environment resets are costly, or rapid deployment is required.

The development of IL methods has evolved from basic **Behavior Cloning (BC)** to more sophisticated **structure-aware policies** such as **Action Chunking with Transformers (ACT)** [8],

and **generative approaches** like **Diffusion Policy (DP)** [13]. BC remains a fundamental technique for system pretraining and baseline setup due to its simplicity and efficiency, but suffers from inherent limitations in robust generalization and error recovery. To address this, structure-aware models like ACT leverage Transformer encoders and action chunking schemes to capture long-horizon temporal dependencies while maintaining a balance between model expressiveness and computational efficiency. In contrast, Diffusion Policy excels at modeling diverse and expressive action distributions, yet it incurs high inference costs and hyperparameter sensitivity, making it more suitable for simulation-rich or resource-intensive applications.

In parallel, the transition from **low-dimensional joint-only inputs** to **multimodal policies combining vision and state** has become a key trend in enabling real-world generalization. Visual inputs significantly enhance the policy's ability to understand spatial layouts, semantic context, and task variations, providing a foundation for handling object occlusion, deformation, and perception ambiguity. However, multimodal fusion introduces new challenges in modality alignment, model complexity, training stability, and real-time execution, which must be addressed via staged training strategies and structured encoder design.

Overall, the evolution of IL reflects a dual-axis progression: from *supervised regression* $\rightarrow$ *temporal abstraction* $\rightarrow$ *conditional generative modeling*, and from *state-driven control* $\rightarrow$ *perceptual fusion* $\rightarrow$ *semantic-guided execution*. Based on this trajectory, our study adopts the ACT framework as the core policy model due to its deployment efficiency and structural interpretability, while gradually integrating multimodal inputs—from joint-only to vision-state representations—to build a practical, extensible, and robust robotic imitation learning system.

The next chapter will elaborate on the training pipeline, implementation platform, and evaluation metrics for this policy framework, validating its effectiveness across real-world tasks and deployment environments.

2.4 Vision-Based Feedback and Visual Servoing

What is Visual Servoing? Visual servoing refers to a class of closed-loop control systems where visual input is continuously incorporated to guide robot behavior. The fundamental principle is to extract task-relevant information—such as relative position, orientation, or movement—from camera data, and convert it into low-level motion commands that guide the robot toward a desired pose or trajectory [31].

Depending on how the visual information is processed and integrated into the control loop, visual servoing is commonly divided into three categories:

Position-Based Visual Servoing (PBVS) reconstructs the 3D pose of the target in Cartesian space and then computes control actions in task space. It provides clear geometric interpretability but heavily relies on accurate camera calibration and is sensitive to visual noise or occlusion.

Image-Based Visual Servoing (IBVS) directly utilizes 2D image features—such as centroids, corners, or contours—as input and performs control in the image plane. IBVS is computationally efficient and robust to partial occlusion, but it may suffer from local minima, scale inconsistency, and visual singularities.

Hybrid Visual Servoing (HVS) combines aspects of both PBVS and IBVS, using partial 3D estimation alongside image-based control to balance precision and robustness, and has gained traction in recent mobile manipulation studies.

Role of Vision Feedback in Imitation Learning Systems In imitation learning (IL), most policies are traditionally trained in an open-loop setting, where the execution is determined entirely by the initial observation. However, this can lead to policy drift and failure in dynamic or partially observable environments. Incorporating visual feedback during execution enables robots to perform closed-loop correction based on the current state, substantially improving their robustness and adaptability.

For instance, the **Mobile ALOHA** system integrates wearable eye-in-hand RGB cameras to record human demonstrations and enable task transfer across diverse scenarios such as drawer opening and food slicing [8]. In **VITAL**, vision is used to detect policy failure and trigger human correction signals in real time, establishing a human-in-the-loop visual policy adjustment mechanism [3]. The **VIOLA** framework applies attention-based perception modules to focus on object-centric visual features, filtering out background noise and improving generalization across environments [32].

These examples demonstrate that vision is no longer a passive input modality but an active component in enabling closed-loop policy learning and deployment in imitation learning systems.

Challenges of Vision-Based Control Despite its benefits, incorporating visual servoing into robotic learning systems introduces several technical challenges:

- **Processing latency:** Real-time visual feedback requires rapid image acquisition, processing, and actuation. On resource-constrained platforms (e.g., edge devices), this latency can hinder responsiveness.
- **Occlusion and feature tracking loss:** Real-world manipulation often causes self-occlusion or external occlusion, making it difficult to maintain continuous visual feature tracking—especially in IBVS-based frameworks.
- **Perceptual error amplification:** Any inaccuracies in visual estimation may be looped back into the control system and cause oscillations or trajectory instability.
- **Nonlinear joint-vision mapping:** The transformation from image features to joint control commands is often nonlinear and discontinuous, which can result in ambiguous or non-smooth servoing behavior.

Recent Approaches and System Designs To overcome the limitations above, researchers have developed advanced visual servoing architectures:

- **Keypoint-based visual control:** Systems like Keypoint-VS [33] use lightweight keypoint detectors combined with IBVS control laws to track object motion more robustly.
- **Deep Visual Servoing (DVS):** End-to-end networks predict motor commands directly from raw RGB inputs, removing the need for handcrafted feature extraction [34]. DVS models can generalize across objects and tasks in cluttered environments.
- **Active Vision and Adaptive Framing:** Recent systems such as **AV-ALOHA** use wearable or mobile cameras whose viewpoints are adjusted according to task phases, thereby reducing occlusion and maintaining visual consistency during execution [2].

These efforts reflect a broader shift from static visual sensing to dynamic, task-driven perception integration.

Our System Design: Perception-Assisted Servoing In our system, we implement a lightweight and modular visual servoing framework to complement policy execution in real-world robotic settings:

- **Camera setup:** We employ a dual RGB-D configuration (top-view and side-view), allowing multi-angle observation of the robot and objects during manipulation. This setup reduces occlusion and enables spatially consistent visual tracking.
- **Error-triggered correction:** Visual feedback is not used as primary input to the policy, but as a verification tool—tracking end-effector trajectory drift or task error, and triggering low-level correction using Jacobian-based updates.
- **Multimodal policy coupling:** The ACT policy is initially trained with joint-only inputs. Vision is introduced gradually in a second training phase, where RGB features are encoded and fused via token embeddings for policy refinement.
- **Synchronized logging:** All visual frames and joint trajectories are timestamped and synchronized at the ROS2 interface level. This ensures consistent, replayable training samples for policy learning.

Summary Visual servoing plays an increasingly vital role in enabling adaptive, perception-aware robotic manipulation. In contrast to purely reactive or preprogrammed systems, our design incorporates closed-loop visual feedback in a lightweight, modular manner that enhances the generalizability and fault tolerance of the imitation learning pipeline.

This lays the foundation for the multimodal training procedures described in the following chapters, where synchronized visual and joint inputs are fused for robust task execution and policy learning.

2.5 Evaluation Metrics for Imitation Learning Policies

Why Evaluation Matters in Imitation Learning Unlike reinforcement learning (RL), which has access to explicit reward functions as performance signals, imitation learning (IL) operates without intrinsic objective feedback. As a result, evaluation metrics in IL act as a *pseudo-reward mechanism*—they define not only how good a policy is, but also which properties the learning process prioritizes.

As IL models evolve toward multimodal fusion (e.g., vision and proprioception), generative modeling (e.g., Diffusion Policy), and structured policy design (e.g., ACT), traditional scalar metrics such as success rate or mean error are no longer sufficient. Comprehensive evaluation must capture *temporal coherence, perceptual robustness, modality-specific contributions*, and policy behavior under dynamic or partially observable conditions [35].

Furthermore, in real-world deployment (e.g., mobile or dual-arm robots), performance must be interpreted alongside **latency, memory consumption, smoothness, and recoverability**. In this

sense, evaluation becomes a diagnostic tool for the policy's deployability and real-world generalization.

Categories of Evaluation Metrics We categorize IL evaluation into five major dimensions, each with representative indicators:

Table 6: Evaluation Metrics for Imitation Learning Policies

Category	**Representative Metrics**	**Purpose**
Accuracy	Mean Absolute Error (MAE), DTW alignment	Fit to expert trajectories (offline)
Success Rate	Task success, first-success latency	Real-world task completion
Generalization	Novel-object performance, prompt transfer	Robustness under environmental shift
Stability	Trajectory smoothness, jitter index	Deployment safety and actuation consistency
Interpretability	Activation heatmaps, failure attribution	Model debugging and introspection

Importantly, *offline accuracy does not guarantee online success*. High MAE or low DTW may mask erratic behavior during execution due to instability or overfitting. Cross-dimensional validation is therefore essential.

Metric Usage in Recent Systems

Modern IL systems apply multiple evaluation strategies:

- **ACT** models use chunk-wise trajectory visualization and alignment metrics to study how temporal abstraction affects control precision [8].
- **Diffusion Policy (DP)** adds behavioral entropy and diversity scores to reflect generative range and sampling stability [13].
- **VIMA** proposes zero-shot task success and prompt consistency as core indicators of generalization across goals [36].
- **RoboTurk** and **RoboNet** utilize crowdsourced ratings and user interactions to evaluate fluency and human-likeness [17].

These systems show that evaluation is shifting from static correctness to *semantic behavior and adaptability analysis*.

Challenges in Evaluating Advanced Policies Key challenges in IL policy evaluation include:

- **Modality ambiguity**: In multimodal models, it is hard to isolate whether performance gains come from vision, state, or model scale.

- **Structure invisibility**: Models like ACT encode task phases, but standard metrics like MAE do not capture semantic alignment.
- **Stochastic sampling noise**: Diffusion models produce varied outputs; accurate evaluation requires Monte Carlo estimates.
- **Lack of unified benchmarks**: No dataset or benchmark supports both generative and structured IL models for fair comparison.
- **Ignored deployment constraints**: Inference time, memory load, and hardware feasibility are under-reported despite industrial relevance.

Our Evaluation Design Strategy We propose a **layered evaluation framework** for IL policies, aligned with both training objectives and deployment goals:

Table 7: Our Evaluation Strategy for IL Models

Layer	**Metrics**	**Purpose**
Offline	MAE, DTW, chunk entropy, cosine loss	Supervised fit to demonstrations
Online	Success rate, time to success, jitter score	Deployment performance
Generalization	Zero-shot success on novel views, occlusion robustness	Robustness analysis
Interpretability	Token-level saliency, activation maps, ablation study	Understanding model behavior
Efficiency	FPS, memory profile, model size (MB)	Feasibility on edge deployment

This design allows us to compare structured (e.g., ACT) and generative (e.g., DP) policies across not only performance, but deployment constraints and behavioral transparency.

Summary: Toward Evaluation as a Feedback Loop In modern IL, evaluation is not an endpoint but a recursive tool for understanding and refining policy architectures. A well-constructed evaluation pipeline bridges **training** $\rightarrow$ **behavior** $\rightarrow$ **deployment**, enabling us to iterate IL designs toward systems that are **not only accurate, but also stable, interpretable, and real-world deployable**.

2.6 Summary and Positioning of Our Work

This chapter has systematically reviewed the critical components that constitute a teleoperated, multimodal, and deployable imitation learning system for robotic manipulation. Beginning with teleoperation interfaces (Section 2.1), we examined the progression from high-cost VR systems toward low-cost, handheld input methods that improve accessibility and deployability. Section 2.2 explored multimodal data collection strategies, highlighting the importance of synchronized visual

and proprioceptive signals for robust policy training. Section 2.3 analyzed the evolution of imitation learning strategies—from classical behavior cloning to transformer-based action chunking (ACT) and generative approaches like Diffusion Policy—revealing the trade-offs between expressiveness, stability, and inference efficiency. Section 2.4 discussed visual servoing as a key feedback mechanism, emphasizing its growing role not only in perception but also in control stability and error correction. Finally, Section 2.5 surveyed evaluation methodologies, calling for multi-dimensional metrics that better reflect deployment feasibility, robustness, and cross-task generalization.

Despite these advancements, several pressing limitations remain in the current landscape:

- **First**, many high-performing policy architectures are computationally demanding and difficult to deploy on resource-constrained platforms, limiting their applicability in real-world teleoperation settings.
- **Second**, most existing systems lack a unified engineering pipeline that connects human demonstration, data acquisition, policy training, visual feedback, and policy evaluation in an integrated and reproducible loop.
- **Third**, although visual inputs are widely used, their *semantic contribution to execution-time control*—especially in the context of feedback, error correction, and spatial generalization—remains underexplored and loosely integrated into policy design.

In response to these gaps, our work proposes a lightweight, modular robotic learning system based on the **LeRobot** dual-arm teleoperation platform. Our system integrates:

1. A real-time *leader–follower teleoperation interface*;
2. A *multimodal data collection module* with synchronized joint and dual-camera vision streams;
3. A *transformer-based Action Chunking policy (ACT)* capable of handling long-horizon structured tasks;
4. A staged *visual feedback mechanism* that supports performance correction and spatial generalization.

This design prioritizes practical deployability, training modularity, and perceptual grounding, with an evaluation protocol that spans offline prediction accuracy, online success rates, and visual–state modality ablation. By anchoring our contribution in the existing literature while addressing its methodological and system-level blind spots, we aim to deliver an accessible, end-to-end framework for imitation-based robotic manipulation in low-cost teleoperation settings.

The following chapter will introduce the system architecture and implementation details of our proposed approach, including the system design, training pipeline, deployment procedures, evaluations and results.

3. System Design

3.1 Hardware Architecture

3.1.1 LeRobot Dual-Arm Setup and Degrees of Freedom

The robotic platform adopted in this project is the **LeRobot Koch dual-arm system** developed by WOWROBO. This platform is designed to support low-cost embodied intelligence research, particularly in dual-arm teleoperation and imitation learning tasks.

The system comprises two independent 6-degree-of-freedom (6-DOF) robotic arms, each consisting of six serially connected revolute joints. This configuration provides sufficient flexibility for precise 3D manipulation, enabling each arm to perform reaching, grasping, and placement tasks within a compact workspace. The arms are mounted side by side on a rigid base, and their kinematic structure allows for both single-arm and bimanual operation.

Each arm terminates in a two-finger parallel gripper actuated by an internal motor. The grippers provide simple binary control (open/close) and are suitable for executing basic pick-and-place operations. Their mechanical design supports a variety of object shapes and sizes, making them adaptable for typical table-top manipulation experiments.

The two arms are configured in a *leader–follower mode*. During data collection or task demonstration, the operator directly manipulates the leader arm, while the follower arm replicates the motion in real time. This architecture simplifies the process of capturing high-quality demonstration data for learning-based control policies.

The LeRobot system is designed to be compact, portable, and modular, making it ideal for experimental setups in constrained laboratory environments. Its mechanical interfaces also allow for easy attachment of external sensors such as RGB cameras, IMUs, or custom gripper tools. The internal wiring and motor drivers are fully enclosed, ensuring safety, aesthetics, and mechanical robustness during extended use.

The physical configuration of the LeRobot dual-arm system is illustrated in Figure 1, showing the leader arm and the follower arm from separate perspectives.

(a) Leader arm setup with two-finger gripper

(b) Follower arm setup mirroring the leader arm

Figure 1: Physical setup of the LeRobot dual-arm platform showing individual views of the leader and follower arms.

In addition, a URDF schematic of the follow arm is shown in Figure 2, highlighting the six revolute joints (J1–J6) that enable 6-DOF control of the arm.

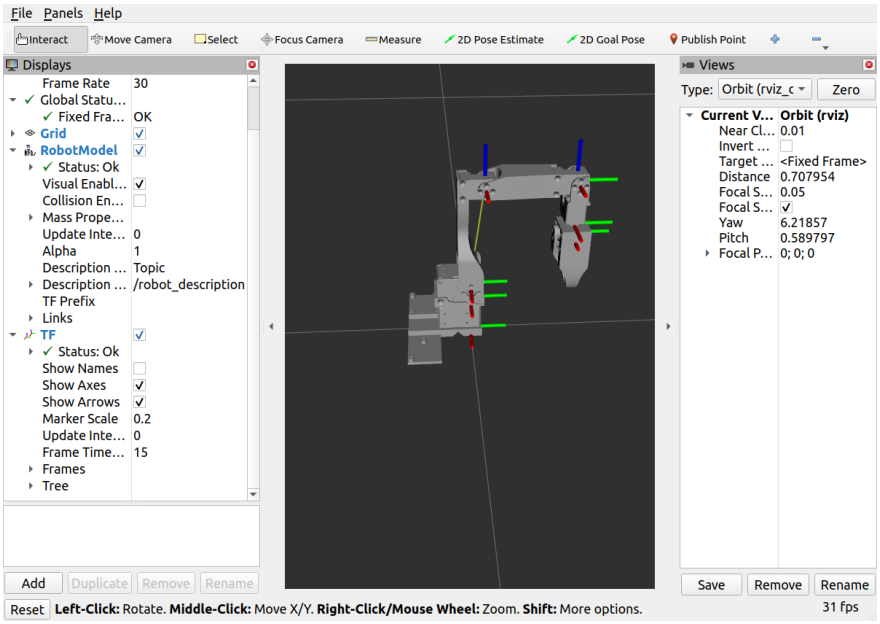


Figure 2: URDF schematic of LeRobot with joints J1–J6 highlighted for follow arm

3.1.2 USB Serial Interface and Port Mapping for Robotic Arms

The LeRobot dual-arm system relies on **USB serial communication** to control the onboard motor controllers of each robotic arm. Each arm contains an embedded microcontroller that handles low-level commands for joint actuation and gripper control. Communication is established via dedicated **USB Type-C connections**, with each cable connecting the arm directly to the host computer.

Upon connection, the operating system enumerates each robotic arm as a serial device under the path `/dev/ttyACM*`. The exact device name assigned (e.g., `/dev/ttyACM0`, `/dev/ttyACM1`) depends on the order of connection and may vary across system reboots. To maintain consistent control mapping throughout the project, a manual port assignment convention is adopted as follows:

- **Leader arm**: `/dev/ttyACM1`
- **Follower arm**: `/dev/ttyACM0`

These mappings are verified before each experiment by checking `dmesg` logs or listing connected devices using `ls /dev/ttyACM*`. Manual confirmation of device order ensures that the correct control signals are sent to each arm, preventing role confusion during synchronized operation.

Each USB connection supports **full-duplex communication**, allowing real-time transmission of joint angle commands, gripper control signals, and synchronization timestamps. The system operates at a fixed sampling rate of **20–60 Hz**, enabling smooth and responsive motion control.

To avoid communication conflicts and ensure deterministic behavior, **dedicated communication threads** are assigned to each arm. These threads handle message queuing, transmission, and reception independently, and they also log joint state feedback for downstream analysis or training purposes.

While the system was originally designed with modular expandability in mind, **practical limitations in USB port availability** significantly constrained the number of devices that could be connected simultaneously. Both robotic arms occupy the only two Type-C ports on the host computer. As a result, adding additional USB serial devices—such as third-party sensors or auxiliary manipulators—was not feasible without external USB expansion hardware, which was deliberately avoided due to instability in power supply and data throughput.

Consequently, careful cable management and port allocation were required to maintain stable communication across all sessions. The current configuration achieves a balance between simplicity, stability, and functionality for dual-arm control and synchronized data collection.

3.1.3 Camera Setup and Connectivity Constraints

To enable effective multimodal observation of the manipulation workspace, two industrial-grade RGB cameras are integrated into the system: the **HIKVISION E22S** and **HIKVISION E12**. These cameras are positioned at **orthogonal viewpoints** to capture complementary visual information, supporting downstream data analysis, behavior understanding, and policy learning.

E22S – Side View Camera: The E22S camera is placed on the **left-hand side** of the workspace, offering a *lateral side view* of the LeRobot dual-arm system during task execution. This perspective captures the motion trajectories of both arms from the side, providing valuable spatiotemporal visual context, particularly for analyzing reaching, grasping, and placement actions.

The E22S features an **integrated base** and does **not require external mounting**. It is a self-standing unit, designed for stability on flat surfaces. As such, it is directly positioned beside the workspace without the need for tripods, clamps, or mechanical fixtures. This camera is consistently

recognized by the operating system as `/dev/video2 eg.`, and its stream is explicitly referenced in the data capture pipeline to prevent any ambiguity or conflict with other video sources.

Figure 3 provides a visual illustration of the E22S camera's physical placement relative to the LeRobot platform. The image highlights the lateral positioning and stable base configuration, offering insight into the typical side-view capture angle used during experiments.

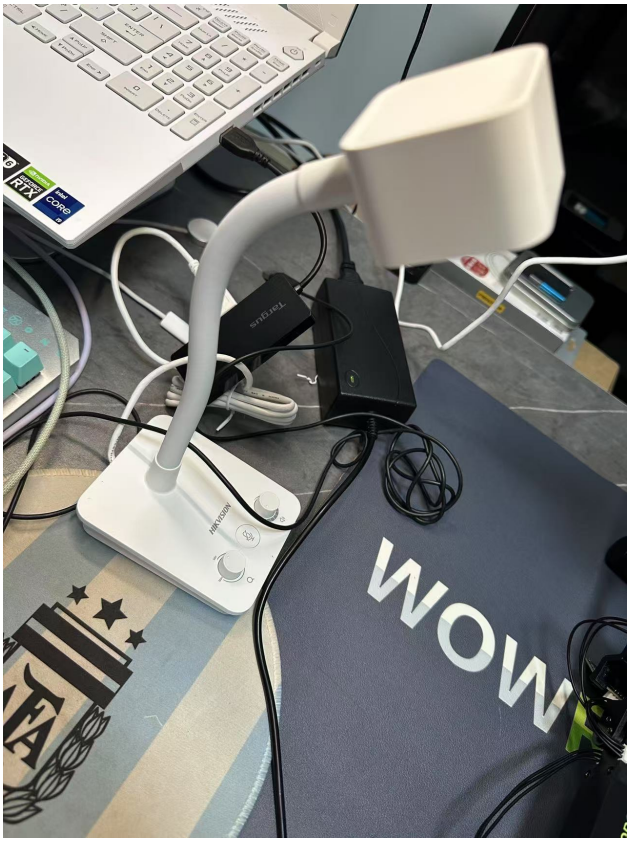


Figure 3: Side-view placement of the HIKVISION E22S camera adjacent to the LeRobot platform.

This setup ensures that the lateral motion of both arms remains consistently visible across all task executions, making it especially useful for analyzing temporal coordination and collision avoidance during dual-arm manipulation.

E12 – Top View Camera: The E12 camera is mounted **directly above the workspace**, offering a *top-down (overhead) view* of the entire table surface. This viewpoint is ideal for tracking object positions, monitoring gripper alignment, and capturing interaction sequences from a global perspective.

Unlike the E22S, the E12 requires **external mechanical support**. It is **clamped to a flexible mechanical positioning arm**, which extends from the periphery of the workspace. This arm is independent of the LeRobot system and serves solely as a stable fixture to support and orient the camera. Using a standard clamping mechanism, the E12 remains fixed in place and stable throughout extended data recording sessions. It is consistently detected as `/dev/video4 eg.` in the system.

Figure 4 illustrates the mounting configuration of the E12 camera, suspended directly above the workspace via a flexible positioning arm.

Figure 4: Overhead mounting of the HIKVISION E12 camera on a positioning arm above the workspace.

This overhead perspective complements the side-view camera by capturing the spatial arrangement of objects and end-effector positions from a global viewpoint, facilitating holistic understanding of pick-and-place dynamics.

Connectivity and System Limitations Both cameras are connected via **USB 3.0** interfaces and are configured to operate at **30/60 FPS**, enabling high-resolution video capture without frame drops. During initial testing, attempts were made to connect both cameras through a shared *USB hub*, but this led to issues such as *signal overload, frame corruption, and power instability*. As a result, each camera was connected directly to a **dedicated USB-A port** on the host computer to ensure uninterrupted data streaming.

However, this decision introduced **hardware-level limitations**. The host computer offers only two USB-A ports and two USB Type-C ports. The two USB-A ports are occupied by the E22S and E12 cameras, while the two robotic arms are connected to the Type-C ports for serial communication. This configuration fully occupies all available ports, leaving no additional bandwidth for integrating a third camera—such as one mounted on the robot's end-effector in an eye-in-hand configuration. Although this feature was considered during the design phase, it was excluded from the final implementation due to these constraints.

No external power supplies or powered USB hubs were employed. All devices were powered directly from the host machine's onboard USB circuitry. While this minimalistic approach simplified the system setup and portability, it required careful port allocation and consistent device mapping to ensure communication stability across long experimental sessions.

For completeness, Figure 5 presents a high-level view of the entire hardware setup, including both robotic arms, camera placements, and host computer connections.

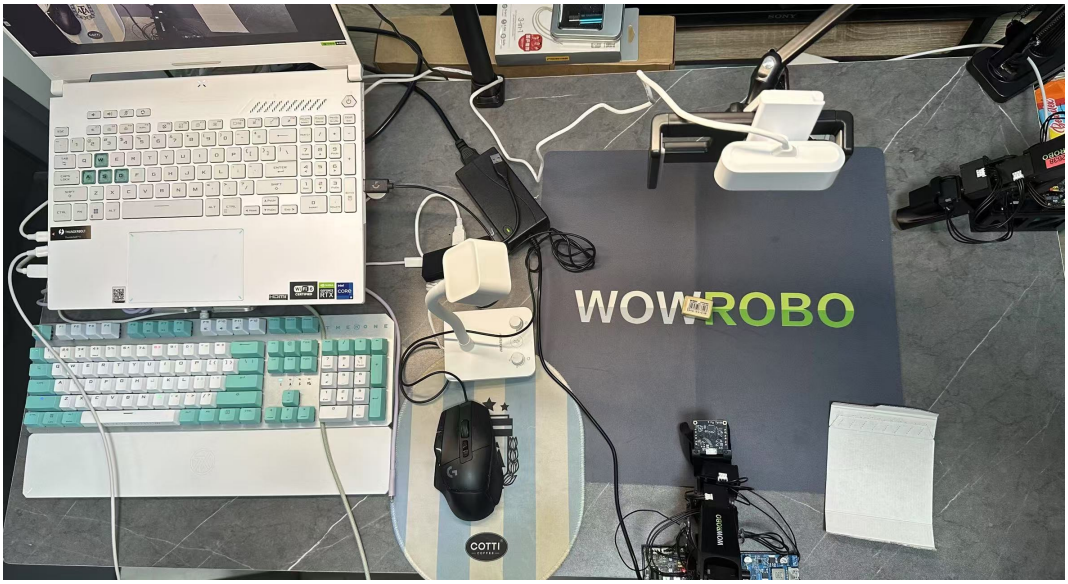


Figure 5: Overall workspace setup showing dual-arm system, side and top cameras, and host PC connections.

This layout represents the final deployed configuration used during all data collection and demonstration sessions. It also reflects the port mapping and physical space constraints discussed earlier in this section. This dual-camera configuration—comprising the E22S side view and E12 top view—proved sufficient for the data collection tasks in this study. Together, they provide a rich, synchronized visual representation of both the robotic motion and environmental interaction, which lays a robust foundation for future expansion into *vision-based feedback control* or *learning from demonstration with visual input*.

3.2 Software Framework

3.2.1 Control Architecture and Threading Model

The control system for the LeRobot dual-arm platform is implemented in Python, using a **multi-threaded execution model** to manage real-time teleoperation and synchronized data capture. This architecture enables responsive control, accurate joint tracking, and uninterrupted data recording, all without reliance on external robotics middleware such as ROS.

At the core of the framework is a **three-thread structure**:

- **Leader Arm Control Thread**
  This thread continuously queries the present joint positions of the leader arm via serial communication. These readings are used as target positions for the follower arm and are also logged as part of the demonstration dataset. The loop runs at a fixed frequency, typically **30 Hz**, and communicates with the device using low-level read commands `(e.g., read "Present_Position")`

- **Follower Arm Control Thread**
  This thread receives the latest joint positions from the leader arm and sends corresponding

commands to the follower arm using: `write("Goal_Position", leader_pos)`. The control loop is tightly coupled with the leader thread to ensure minimal latency in mirroring the motion.

- **Data Recording Thread**
  A separate asynchronous thread handles multimodal data recording, including:
  - Joint positions of both arms,
  - High-frequency timestamps (via `time.perf_counter()`),
  - Synchronized RGB frames from the E22S and E12 cameras.

Each thread operates independently to prevent blocking operations. **Busy-waiting loops** are used in place of `time.sleep()` to maintain high temporal precision in the main control loop—especially important during fast-paced, synchronized movements. For example, in a 30-second control session running at 200 Hz, over 6000 iterations may be executed with precise time intervals using the following structure:

```
for _ in range(seconds * frequency):
    leader_pos = robot.leader_arms["main"].read("Present_Position")
    robot.follower_arms["main"].write("Goal_Position", leader_pos)
    # Record data and sync timestamps...
```

Listing 1: Leader-follower loop with high-frequency synchronization

This control strategy ensures **tight temporal alignment** between leader and follower arms and maintains reliable synchronization even under system load. Logging is handled in parallel to reduce blocking time in the critical control loop.

Overall, this thread architecture provides a lightweight yet effective control scheme suitable for real-world robotic teleoperation and learning-from-demonstration scenarios.

3.2.2 Controller Initialization and Serial Communication

To ensure stable and deterministic control over both robotic arms, the system leverages **persistent device identifiers** under the path `/dev/serial/by-id/`, rather than relying on dynamically assigned device names such as `/dev/ttyACM0`. This method eliminates ambiguity in device mapping and guarantees consistent role assignment across sessions, regardless of USB connection order or port enumeration.

Specifically, the serial ports for the two arms are assigned as follows (by-id):

- **Leader Arm:**
  `/dev/serial/by-id/usb-1a86_USB_Single_Serial_58FA096602-if00`
- **Follower Arm:**
  `/dev/serial/by-id/usb-1a86_USB_Single_Serial_58FA095568-if00`

These unique identifiers are obtained directly from the USB device metadata and remain constant unless the physical device is replaced. This approach removes the need for manual port checks or `dmesg` validation during initialization.

The robotic system is configured using the `KochRobotConfig` class provided by the LeRobot framework. This class encapsulates both hardware port assignments and camera configurations, serving as a centralized initialization interface. The following code snippet demonstrates how the robot configuration is instantiated:

```
from lerobot.common.robot_devices.robots.configs import KochRobotConfig
from lerobot.common.robot_devices.robots.manipulator import
   ManipulatorRobot

robot_cfg = KochRobotConfig()
robot = ManipulatorRobot(robot_cfg)
robot.connect()
```

Listing 2: Robot instantiation and connection

Upon calling `robot.connect()`, the system automatically establishes serial communication with both arms using the assigned device IDs. Internally, this command performs:

- Baud rate configuration
- Low-level motor controller initialization
- Device readiness checks (e.g., querying `Present_Position`)
- Auto-calibration routines (on first-time launch)

In addition to arm control, the same configuration interface is used to set up the RGB cameras:

```
from lerobot.common.robot_devices.cameras.configs import
   OpenCVCameraConfig

robot_cfg.cameras = {
   "e22s_side": OpenCVCameraConfig(camera_index=4, fps=30, width=640,
      height=480),
   "e12_top": OpenCVCameraConfig(camera_index=2, fps=30, width=640,
      height=480),
}
```

Listing 3: Camera configuration with fixed device indexes

This camera configuration binds each logical camera name (`e22s_side`, `e12_top`) to a fixed physical device index (`/dev/video4`, `/dev/video2` respectively). These settings are later referenced throughout the data recording pipeline to ensure consistent visual data streams.

Once all device interfaces are successfully initialized, the robot is ready to enter teleoperation or recording mode. All subsequent communication is handled through internally managed full-duplex serial protocols abstracted by the `ManipulatorRobot` interface, with no need for manual low-level command composition.

This high-level yet deterministic initialization procedure ensures a **plug-and-play user experience** while preserving **engineering-grade precision and control** required for robotic data collection and learning.

3.2.3 Data Logging and Sampling Loop

To support imitation learning with high-fidelity multimodal data, a dedicated data logging mechanism was implemented to collect joint states, synchronized camera frames, and precise timestamps. The logging loop is designed to operate at a fixed sampling frequency of **20 Hz**, balancing recording resolution with system stability.

The core logging routine is implemented in Python, without reliance on image queues or asynchronous pipelines. Instead, a tight loop governed by `time.perf_counter()` and a custom `busy_wait()` function ensures that each data frame is collected at precise intervals. This approach avoids timing drift commonly introduced by `time.sleep()`, and guarantees frame-level alignment between modalities.

Each recording session is saved into a uniquely indexed directory under `recordings/clip_XXXXX/`, where all relevant data are organized into three subfolders:

- `images_cam0/`: RGB frames from the E22S (side-view) camera
- `images_cam1/`: RGB frames from the E12 (top-view) camera
- `joint_states/`: Joint state logs and metadata files

The data collection loop performs the following steps in each iteration:

Step 1. Read Leader Joint Position The `Present_Position` value is queried from the leader arm. The retrieved 6-DoF joint angles are optionally passed through an exponential moving average (EMA) filter with a smoothing factor of $\alpha = 0.3$, improving stability while preserving responsiveness.

Step 2. Send Position to Follower Arm The smoothed joint angles are transmitted to the follower arm using `write("Goal_Position", ...)`, achieving near real-time motion mirroring.

Step 3. Query Follower Arm Position The follower arm's actual joint position is read and recorded, enabling post hoc analysis of control lag or error compensation.

Step 4. Capture RGB Images Frames are simultaneously grabbed from both camera streams using `cv2.VideoCapture.read()`. If either frame is invalid (i.e., `None`), the system skips the current iteration and logs a warning.

Data Directory Structure Each recording session is stored in a separate folder under the root directory `recordings/`, named as `clip_XXXXX/`. Within each clip directory, data are separated into subfolders for each modality (cameras and joint states), maintaining a consistent and parseable layout.

Figure 6 illustrates the overall data directory hierarchy.

```
recordings/
└── clip_00001/
    ├── cam0/
    │   ├── frame_00000_cam0.jpg
    │   └── ...
    ├── cam1/
    │   ├── frame_00000_cam1.jpg
    │   └── ...
    ├── joint_states/
    │   ├── joint_data.pkl
    │   ├── joint_data.csv
    └── timestamp_log.txt
```

Figure 6: Hierarchical structure of the data recording folder.

Step 5. Save Visual and Kinematic Data Images are saved as JPEG files with structured filenames:

```
eg. frame_00012.jpg # saved in both cam0 and cam1 folders
```

Listing 4: Saved image filename format

Joint states and timestamps are appended to a buffer in the format:

```
[timestamp, leader_joint_vector, follower_joint_vector]
```

Listing 5: Joint logging format per frame

6. Wait Until Next Frame The duration of each loop is measured using `perf_counter()`, and the remaining time—based on the target interval (1/20 s)—is filled using a high-precision `busy_wait()` function to maintain a constant frame rate.

An example of this logging loop is shown below:

```
for i in range(total_frames):
   t_start = time.perf_counter()

   leader = read_joint_position(leader_arm)
   smoothed = ema_filter(leader, alpha=0.3)
   write_joint_position(follower_arm, smoothed)

   follower = read_joint_position(follower_arm)
   img0 = cam0.read()[1]
   img1 = cam1.read()[1]

   if img0 is None or img1 is None:
      print("Frame dropped")
```

```
        continue

    save_images(i, img0, img1)
    log_buffer.append([t_start, leader, follower])

    busy_wait_until_next_frame(t_start, interval=1/20)
```

Listing 6: High-precision data logging loop

After the session ends (e.g., by pressing `q`), the joint log is saved as a `joint_data.pkl` file using the `pickle` module. Each file contains a list of `[timestamp, leader, follower]` entries representing the complete joint trajectory.

In the post-processing phase, additional scripts traverse all `clip_*` directories and convert the collected `.pkl` files into standardized datasets in both `.csv` and `.npy` formats. Each converted entry includes a 12-dimensional `[state, action]` vector, where:

- **state:** leader arm joint angles
- **action:** follower arm joint angles

This structured data format is directly compatible with downstream learning pipelines such as ACT and Diffusion Policy.

3.2.4 Directory Structure and File Naming

To facilitate long-term data management, reproducibility, and compatibility with downstream imitation learning pipelines, a structured and standardized directory layout is adopted for organizing all recorded data. Each demonstration is saved under a unique clip directory, and all modalities are separated by folder and filename to ensure clarity and modularity.

Clip Directory Generation At the beginning of each recording session, the system automatically assigns a unique directory name using an incremental numeric ID. This is implemented by scanning the existing `recordings/` directory for folders matching the pattern `clip_XXXXX`, where `XXXXX` is a five-digit zero-padded index. The next available ID is selected in order:

```
clip_00001/
clip_00002/
...
clip_00023/
```

Listing 7: Auto-generated clip directories

This mechanism guarantees no overwriting of previous recordings and maintains a chronologically ordered dataset, making it easy to track the evolution of data across sessions.

Folder Structure within Each Clip Each `clip_XXXXX` directory contains the following three subdirectories:

- `images_cam0/`: RGB frames captured from the E22S side-view camera.

- `images_cam1/`: RGB frames captured from the E12 top-view camera.
- `joint_states/`: Joint position logs, timestamps, and processed datasets.

The overall folder hierarchy is illustrated in figure 6.

File Naming Conventions All frames are saved using a zero-padded filename format to preserve lexicographic and temporal ordering:

```
frame_00000.jpg
frame_00001.jpg
frame_00002.jpg
...
```

Listing 8: Image file naming convention

This format is applied identically to both camera views (`cam0` and `cam1`), which guarantees frame-to-frame alignment during multimodal training or visualization.

Joint data is first saved as a binary `.pkl` file, preserving exact timestamps and full float precision. In post-processing, it is converted into:

- `state_action.csv`: Human-readable tabular format.
- `state_action.npy`: NumPy array format for direct ingestion by machine learning models.

The combination of **directory-based data isolation** and **consistent naming conventions** allows the dataset to be:

- Easily inspected and visualized
- Directly parsed by LeRobot training scripts
- Scalable for batch processing or cloud upload (e.g., HuggingFace Datasets)

3.3 Summary of the system design

In this chapter, we presented a comprehensive breakdown of the hardware and software design behind the proposed LeRobot-based dual-arm robotic system. The hardware section outlined the physical setup, covering the degrees of freedom for each manipulator, the USB-based serial interface configuration, and the physical constraints posed by limited Type-C port availability on the host computer. These factors collectively shaped the system's modular wiring and connectivity strategy.

On the software side, we described the multi-threaded control framework, where dedicated communication threads manage real-time joint commands and sensor feedback for each arm independently. Key design elements such as controller initialization, data logging frequency, and error-resilient sampling loops were introduced to support stable and synchronized dual-arm operation. Additionally, we proposed a consistent and parseable directory structure that separates each recording session into modality-specific folders. This organization not only facilitates fast debugging and visualization, but also ensures compatibility with automated training pipelines and scalable dataset upload (e.g., to HuggingFace or internal platforms).

Overall, this system design achieves a practical trade-off between modularity, resource efficiency, and robustness, enabling real-time control and high-frequency data acquisition under constrained hardware conditions. With the infrastructure in place, the next chapter focuses on system bring-up and functional validation. There, we examine how the integrated system behaves under actual runtime conditions—evaluating its responsiveness, control synchronization, and real-world stability—before deploying it for learning-based task execution.

4. System Initialization and Operational Validation

This phase aimed to verify the basic operability of the dual-arm robotic system and ensure that both manipulators and cameras were properly recognized, configured, and controlled from the host machine. All tests were conducted in a modular Jupyter Notebook, divided into executable cells to facilitate step-by-step validation and debugging.

4.1 Phase 1 – Device Connection and Teleoperation Testing

4.1.1 Serial Port Binding and USB Identification

To establish stable and repeatable communication with the LeRobot dual-arm platform, it is crucial to ensure deterministic mapping between the physical robot arms and their corresponding virtual serial ports. Each arm is internally controlled by a microcontroller that exposes a USB serial interface, and both arms are connected to the host computer via independent USB Type-C cables.

However, under the default Linux device management system, the assignment of device paths such as `/dev/ttyACM0` or `/dev/ttyACM1` is **non-deterministic**. This means the same physical device might be assigned to a different port after reboot or reconnection, potentially leading to incorrect role assignments (e.g., the leader and follower arms being swapped), resulting in control failures or unsafe actions.

To resolve this, a robust identification strategy based on **persistent device IDs** was adopted using the `/dev/serial/by-id/` directory. These symbolic links are generated by the system and tied to each device's hardware serial number, offering reliable and consistent identification across sessions.

Port Binding Strategy The device paths for each robot arm are explicitly declared in the configuration file `koch_robot_config.py`, located at (Take my device for example):

```
/home/wjw/lerobot/lerobot/common/robot_devices/robots/configs.py
```

Listing 9: Configuration file location

Within the `KochRobotConfig` class, the leader and follower arms are configured using persistent device paths:

```
leader_arms: {
   "main": DynamixelMotorsBusConfig(
      port="/dev/serial/by-id/usb-1a86_USB_Single_Serial_58FA096602-
         if00",
      ...
   )
}
follower_arms: {
   "main": DynamixelMotorsBusConfig(
      port="/dev/serial/by-id/usb-1a86_USB_Single_Serial_58FA095568-
         if00",
      ...
   )
```

```
}
```

Listing 10: Serial port binding with persistent IDs

These serial IDs are obtained by listing the symbolic links in the device directory:

```
ls -l /dev/serial/by-id/
```

Listing 11: Query available persistent serial IDs

The resulting identifiers are **hard-coded into the configuration** to enforce consistent mapping between each physical robot and its logical role.

This strategy ensures that:

- The leader arm is always mapped to the same control interface, regardless of USB port order.
- No manual checking or reconfiguration is needed between sessions.
- System boot-time errors and port misassignments are completely avoided.

Interactive Port Verification To assist users in identifying and confirming the correct mapping during development and debugging, helper scripts were written to automate port comparison. These scripts:

- List currently active serial ports
- Detect newly added devices
- Highlight unexpected or missing devices

```
def list_serial_ports():
   return set(p for p in os.listdir('/dev') if p.startswith('ttyACM')
      or p.startswith('ttyUSB'))

before = list_serial_ports()
input("Please connect the robotic arm and press Enter...")
time.sleep(5)
after = list_serial_ports()
new_ports = after - before
```

Listing 12: Interactive serial port detection script

Additionally, the initialization process uses the `robot.connect()` interface, which internally verifies whether the declared device IDs are available and operational.

Importance in System Setup This port binding mechanism is foundational to the rest of the system's architecture, especially for:

- Reliable dual-arm teleoperation, where precise role assignments (leader vs. follower) are critical
- Multithreaded data collection, which requires deterministic communication interfaces
- Real-world deployment, where automated startup and minimal supervision are expected

During all experiments, this strategy proved to be stable and effective. The system consistently detected and initialized both arms without port collisions, even after cable reconnections or full system reboots.

By combining symbolic device binding with verification scripts, the project achieves a high level of robustness in hardware-software interfacing—essential for scalable multi-device robotics research.

4.1.2 Camera Connection and Verification

In addition to robotic control, visual feedback is a critical component of the system's multimodal data collection pipeline. This section describes the setup and verification of the two USB cameras employed in the system: the **HIKVISION E22S** and **HIKVISION E12**, which provide side and top views of the workspace, respectively.

These cameras are directly connected to the host machine via USB 3.0 ports and serve as the primary source of visual information during demonstration collection and policy execution.

Camera Hardware Placement and Function

- **E22S – Side View Camera:** A compact industrial camera placed on the left-hand side of the workspace, capturing a lateral view of the dual-arm area. This perspective enables observation of arm reachability, motion trajectory, and grasping posture. The E22S includes a built-in base and requires no additional mounting.
- **E12 – Top View Camera:** Mounted above the workspace using an external articulating arm (not part of the robot), the E12 captures an overhead view. It is used to monitor object positions and top-down manipulation. The mounting ensures a fixed and stable viewpoint across sessions.

Due to limited USB availability on the host computer and the high bandwidth requirement of both cameras, **no USB hubs are used**. Each camera is connected directly to a dedicated USB 3.0 port to prevent power instability or data bottlenecks.

Device Index Mapping and Detection On Linux systems, video devices are typically detected as `/dev/video*`. However, the numeric assignment (e.g., `video0`, `video2`) is not guaranteed to persist across boots.

To ensure stable indexing, camera indices are explicitly declared in the robot configuration file:

```
robot_cfg.cameras = {
   "e22s_side": OpenCVCameraConfig(
      camera_index=4, # Corresponds to /dev/video4
      fps=30,
      width=640,
      height=480,
   ),
   "e12_top": OpenCVCameraConfig(
      camera_index=2, # Corresponds to /dev/video2
      fps=30,
      width=640,
      height=480,
   ),
}
```

Listing 13: Camera index configuration in Python

Device detection and mapping are confirmed using:

```
ls /dev/video*
v4l2-ctl --list-devices
```

Listing 14: Terminal commands for camera device listing

A helper script is also used to preview live camera feeds and verify camera orientation:

```
cap = cv2.VideoCapture(4) # E22S
while True:
   ret, frame = cap.read()
   if ret:
      cv2.imshow("E22S Preview", frame)
   if cv2.waitKey(1) & 0xFF == ord('q'):
      break
```

Listing 15: Live camera preview with OpenCV

Through this procedure, it was confirmed that:

- `/dev/video4` reliably maps to the E22S (side view)
- `/dev/video2` reliably maps to the E12 (top view)

Of course, the port information will be different each time you actually connect. Here is my actual situation as an example

Verification Outcomes

- Both cameras initialized successfully at 30 FPS and 640×480 resolution.
- Physical mounting directions were verified through live preview.
- No significant latency, dropped frames, or power issues observed.

- USB ports were stable under continuous recording conditions.

During later stages of the project, these video streams were used in conjunction with joint state logs to generate synchronized visual-kinematic datasets. The validated camera pipeline also supports future extensions such as *vision-based error correction*, *object tracking*, and *learning from images*.

4.1.3 Robot Initialization, Calibration and Torque Check

After verifying the USB mapping and camera streams, the next step is to initialize both robot arms, confirm the integrity of the hardware communication, and ensure that all actuators are properly calibrated and torque-enabled. This procedure lays the groundwork for safe and reliable teleoperation during demonstration collection and policy deployment.

Robot Initialization Process The robot is initialized by constructing an instance of the `KochRobotConfig` class, which encapsulates configuration parameters such as camera mappings, serial ports, joint IDs, and operational modes. This configuration is passed to a `ManipulatorRobot` object for low-level control:

```
from lerobot.common.robot_devices.robots.configs import KochRobotConfig
from lerobot.common.robot_devices.robots.manipulator import
   ManipulatorRobot

robot_cfg = KochRobotConfig()
robot = ManipulatorRobot(robot_cfg)
robot.connect()
```

Listing 16: Robot initialization and configuration

The `connect()` method automatically:

- Loads arm configuration parameters
- Detects and validates USB serial ports
- Initializes internal communication caches
- Loads cached calibration values, if available

Successful connection is confirmed via console outputs that include the number of joints detected and camera readiness status.

Arm Calibration Procedure To ensure accurate synchronization between the leader and follower arms, a one-time calibration procedure is required when the LeRobot system is first initialized. This step aligns the internal encoder values of each Dynamixel motor such that when both arms assume the same physical configuration, they also share consistent joint values in software.

This calibration is essential for reliable teleoperation and imitation learning, enabling consistent kinematics across hardware units and repeatable behavior under control commands.

Automatic Calibration Trigger When `robot.connect()` is executed for the first time, and no cached calibration file is found under the default path `~/.cache/calibration/koch/`, the system automatically launches the calibration routine for both arms. This procedure is fully integrated into the `ManipulatorRobot` class and proceeds as follows:

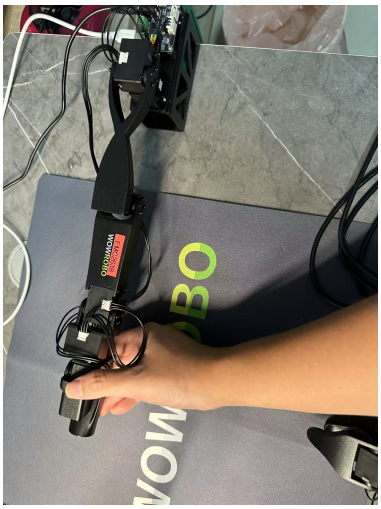
- **Zero Position** – All joints are extended into a neutral, straight-line pose.
- **Rotated Position** – Joints are manually rotated approximately 90 degrees to verify motor direction.
- **Rest Position** – Arm is moved to a relaxed configuration used as the neutral base for motion planning.

These positions are manually guided by the user and prompted via terminal messages. Example output from the system is shown in Listing 17.

```
Connecting main follower arm
Connecting main leader arm
Missing calibration file '.cache/calibration/koch/main_follower.json'
Running calibration of koch main follower...
Move arm to zero position
...
Move arm to rotated position
...
Move arm to rest position
Calibration is done! Saving calibration file '.cache/calibration/koch/
   main_follower.json'
```

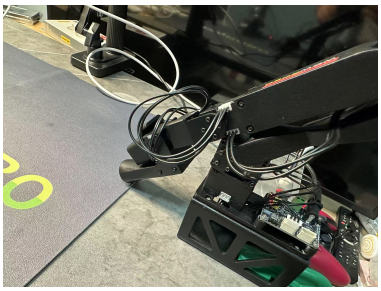
Listing 17: Terminal-guided calibration prompts

Calibration Pose Examples Figure 7 shows the physical configurations of both arms during the calibration sequence. The leader and follower arms follow the same steps independently.

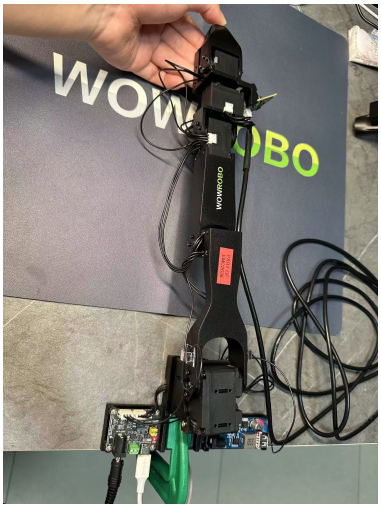


(a) Leader Arm – Zero Position

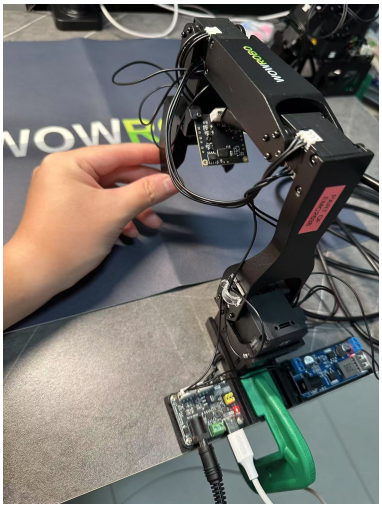
(b) Leader Arm – Rotated Position



(c) Leader Arm – Rest Position



(d) Follower Arm – Zero Position



(e) Follower Arm – Rotated Position

(f) Follower Arm – Rest Position

Figure 7: Calibration pose examples during first-time setup. Each arm is manually guided through three poses: zero, rotated, and rest.

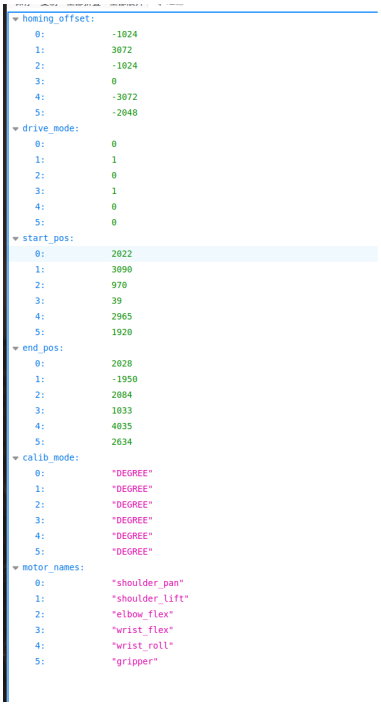
Calibration Result Storage and Visualization Once calibration is completed for both arms, the encoder offset values and final joint mappings are preserved in structured JSON files, as shown below. These calibration results ensure consistent robot behavior across experimental sessions and hardware restarts.

The saved files are stored under:

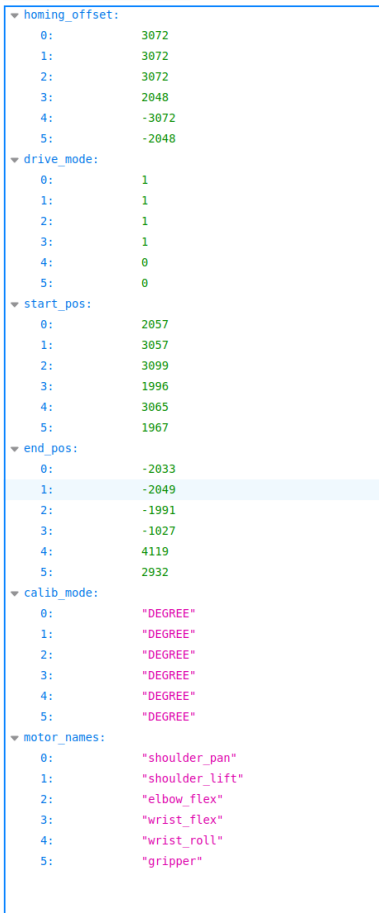
```
~/.cache/calibration/koch/
        main_leader.json
        main_follower.json
```

Each file contains the homing offset, start position, and end position for every motor. A sample

of these contents is visualized in Figure 8.



(a) Leader arm calibration result: `main_leader.json`



(b) Follower arm calibration result: `main_follower.json`

Figure 8: Calibration results.

Accroding to figure 8 above, the final calibration results for both the leader and follower robotic arms, which are stored in two structured configuration files: `main_leader.json` and `main_follower.json`. Each file encapsulates the critical calibration parameters required for reliable motion control and reproducible joint positioning.

Specifically, the `homing_offset` field defines the raw encoder displacement needed to align each joint with its mechanical zero. These offsets are determined empirically through manual homing procedures. The entries `start_pos` and `end_pos` specify the calibrated joint motion ranges, ensuring that all runtime commands remain within safe physical boundaries.

Furthermore, each joint is assigned a motion unit (e.g., `DEGREE`) and a semantic identifier under the `motor_names` key, such as `shoulder_pan` or `wrist_roll`. These labels facilitate intuitive parsing, visualization, and debugging during task execution or trajectory replay.

By storing all calibration parameters in structured JSON files, the system supports rapid reinitialization with consistent joint references. This design is particularly critical for dual-arm synchronization and replay-based evaluation tasks, where consistent alignment and safety constraints

must be preserved across multiple sessions.

Calibration Verification After calibration, verification is performed by placing both arms in the same physical pose and reading their joint angles using:

```
leader_pos = robot.leader_arms["main"].read("Present_Position")
follower_pos = robot.follower_arms["main"].read("Present_Position")

print(leader_pos)
print(follower_pos)
```

Listing 18: Verifying joint consistency post-calibration

A successful calibration yields near-identical joint vectors across all six degrees of freedom. Minor deviations (within 1–2 degrees) are acceptable due to motor quantization and mechanical tolerance but should remain consistent across sessions.

Torque Verification After initialization, it is essential to check that all actuators are torque-enabled. This is done by querying each arm:

```
robot.leader_arms["main"].read("Torque_Enable")
robot.follower_arms["main"].read("Torque_Enable")
```

Listing 19: Torque enable status check

If all returned values are `1`, the joints are torque-enabled and ready for control. Additional hardware health is verified using:

```
robot.leader_arms["main"].read("Hardware_Error_Status")
```

Listing 20: Checking for hardware error status

A return value of `0` indicates normal operation. Any non-zero result suggests motor overload, disconnection, or overcurrent, requiring immediate attention.

Outcome and Significance By completing the initialization, calibration, and verification steps:

- Both robotic arms are recognized and controllable via fixed serial IDs
- Cameras are initialized and ready for synchronized capture
- Torque is enabled, and all joints are ready to receive motion commands
- Cached calibration ensures consistent joint alignment and behavior

This procedure guarantees that the platform is fully operational and safe for the subsequent phases involving real-time teleoperation and data collection.

4.1.4 Real-Time Leader-Follower Control Test

After verifying port connections, camera readiness, and completing the initial calibration, the next step in Phase 1 was to validate the real-time teleoperation pipeline between the leader and follower arms. This test ensured that the **leader arm** (manually guided) could transmit its motion to the **follower arm** with low latency and high accuracy.

Teleoperation Architecture

The system adopts a **leader-follower control scheme**, where the leader arm's joint states are sampled continuously and transmitted to the follower arm as motion commands. This achieves a real-time mirroring effect using serial communication.

A simplified version of the control loop is shown below:

```
for _ in tqdm(range(seconds * frequency)):
    leader_pos = robot.leader_arms["main"].read("Present_Position")
    robot.follower_arms["main"].write("Goal_Position", leader_pos)
    time.sleep(interval)
```

Listing 21: Synchronous teleoperation loop at 200Hz

Where:

- `frequency = 200` Hz, corresponding to 5ms between updates
- `Present_Position` reads the leader's current joint angles
- `Goal_Position` sends the target to the follower arm

Test Conditions and Setup

The validation experiment used the following setup:

- **Test duration:** 30 seconds
- **Update frequency:** 200 Hz (high-frequency, low-latency)
- **Threading:** Blocking synchronous loop with `tqdm` progress bar

During the session, the leader arm was gently moved through a range of poses including:

- Shoulder lift and pan
- Elbow flexion and wrist articulation
- Gripper motion was disabled for this test

Execution and Observations

The follower arm successfully mirrored the leader's trajectory in real-time, with the following observations:

- No visible lag or delay was detected between motion commands and response
- All 6 degrees of freedom (DoF) showed consistent and synchronized behavior
- No communication dropouts or execution errors were observed

Sample terminal output confirmed successful control loop execution:

```
Leader-Follower Teleoperation Completed!
```

The motion was also visually verified through side-by-side observation of both arms. Throughout the session, synchronization remained stable and responsive.

Significance and Relevance

This test validated that the integrated hardware–software stack—spanning:

- USB serial port assignment
- Encoder calibration
- Real-time Python control loop

is sufficient to support high-speed, low-latency teleoperation of a dual-arm robotic system. It forms the technical basis for data collection in subsequent phases of the project, particularly imitation learning and behavior cloning tasks.

Supplementary Material

A recorded video demonstrating the successful execution of this experiment is provided as part of the supplementary materials. Please refer to Appendix A.2 for details.

It showcases the synchronized movement of both arms under real-time control and provides visual confirmation of system performance.

4.1.5 Observations and Debug Notes

Throughout the Phase 1 testing process, several practical issues were encountered and addressed. These lessons contributed significantly to improving the robustness and reproducibility of the final system configuration.

- **USB Device Conflicts:** Due to the limited number of USB ports on the host computer, both RGB cameras had to be connected directly to the system. Early attempts to use shared USB hubs resulted in signal degradation, bandwidth contention, and occasional camera disconnection or frame loss.

- **Serial Port Ambiguity:** The default Linux device naming convention (`/dev/ttyACM*`) assigns device paths based on connection order. This led to inconsistent arm-role mapping across reboots. Switching to persistent identifiers under `/dev/serial/by-id/` resolved this issue and ensured deterministic device assignment.
- **Camera Index Instability:** Video devices enumerated under `/dev/video*` showed inconsistent indexing across sessions. The team used `v4l2-ctl --list-devices` to verify actual device mappings and then hardcoded the `camera_index` parameters into the configuration file to avoid runtime confusion.
- **First-Time Calibration Requirement:** When calibration files were not found in the cache directory, the system launched an interactive pose-based calibration sequence. Ensuring precise joint alignment before calibration was crucial to avoid offset accumulation and long-term motion drift during teleoperation.
- **Torque Status Checks:** In some instances, motion commands had no effect due to torque not being enabled after boot. Manual inspection using the `Torque_Enable` status flags allowed early detection of this issue and prevented wasted trial runs.

These early-stage debugging insights allowed for a more stable and efficient experimental workflow in Phase 2 and beyond. The lessons learned here were incorporated into automated verification routines and operator checklists for future sessions.

4.2 Phase 2 – Data Collection Pipeline Verification

4.2.1 Joint Position Logging and Timestamp Accuracy

To ensure accurate and temporally aligned data for imitation learning, a dedicated joint position logging system was implemented. This module captures the real-time joint angles of both the leader and follower arms, along with a high-precision timestamp, at a fixed sampling frequency of 20 Hz.

Real-Time Joint Reading The current joint states of the leader arm are continuously queried using:

```
leader_pos = robot.leader_arms["main"].read("Present_Position")
```

Listing 22: Leader arm joint readout

Smoothing Filter for Leader-Follower Stability To reduce noise and improve control stability, an exponential moving average (EMA) filter was applied to the leader joint readings before being sent to the follower:

```
alpha = 0.3
smoothed_leader = alpha * leader_pos + (1 - alpha) * smoothed_leader
robot.follower_arms["main"].write("Goal_Position", smoothed_leader)
```

Listing 23: Smoothing filter applied before sending to follower

The smoothing factor $\alpha = 0.3$ was empirically selected to balance responsiveness and stability during fast arm movements.

Timestamp Generation and Logging Each recording cycle is timestamped using Python's high-precision timer:

```
from datetime import datetime
timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")
```

This timestamp, along with both the leader and follower joint vectors, is appended as a single entry to a recording buffer in the format:

```
[timestamp, leader_joints, follower_joints]
```

Upon completion of the session, the buffer is saved into a `joint_data.pkl` file using the `pickle` module.

Sampling Frequency Control Rather than using `time.sleep()`, which may cause drift due to OS-level scheduling delays, the system relies on `time.perf_counter()` combined with a custom `busy_wait()` loop to maintain accurate 20 Hz control intervals:

```
start_time = time.perf_counter()
...
elapsed = time.perf_counter() - start_time
busy_wait(interval - elapsed)
```

Listing 24: Timing control using perf$_counter$

This approach improves frame alignment accuracy, minimizes timing jitter, and ensures synchronized data collection across modalities.

Effectiveness Experimental comparison against earlier scripts using faster frequencies (e.g., 30 Hz) and higher smoothing values ($\alpha = 0.5$) revealed noticeable motion jitter in the follower arm and occasional frame drops. The final logging design demonstrated:

- Stable and low-latency joint mirroring
- Minimal drift between joint logs and camera frames
- Consistent log formatting across all sessions

The resulting logs serve as a high-quality, temporally aligned dataset for training downstream transformer-based imitation learning models such as ACT.

4.2.2 Dual-Camera Frame Capture and Alignment

To capture temporally aligned visual data from multiple viewpoints, a dual-camera recording system was employed. Two industrial RGB cameras were used:

- **E22S (Side View)**: Mounted to the left of the workspace
- **E12 (Top View)**: Mounted above the workspace using a mechanical arm

Each camera was configured with a resolution of `640×480` at `30 FPS` and was assigned to a fixed video device index on the host computer (`/dev/video0` and `/dev/video4`). The configuration was declared explicitly in the robot configuration class as follows:

```
robot_cfg.cameras = {
   "e22s_side": OpenCVCameraConfig(camera_index=0, fps=30, width=640,
      height=480),
   "e12_top": OpenCVCameraConfig(camera_index=4, fps=30, width=640,
      height=480),
}
```

Listing 25: Camera device assignment in configuration

Synchronized Frame Capture During each iteration of the data collection loop, both cameras were queried using OpenCV:

```
frame0 = cap0.read()
frame1 = cap1.read()
```

To ensure temporal consistency, the captured image pair was saved using a shared index under synchronized naming:

```
clip_00040/
 images_cam0/
    frame_00000.jpg
    ...
 images_cam1/
    frame_00000.jpg
    ...
```

This filename convention ensures a one-to-one mapping between the two views, which is essential for multimodal learning, image fusion, and video replay alignment.

Frame Integrity Check To avoid misalignment caused by missing or corrupted frames, the following validation routine was implemented:

```
if frame0 is None or frame1 is None:
   bad_frame_count += 1
   continue
```

Only valid, paired frames were recorded, preserving temporal consistency throughout the clip. Real-time visual verification was also performed using:

```
cv2.imshow("E22S", frame0)
cv2.imshow("E12", frame1)
```

Outcome and Alignment Quality The final design guarantees minimal latency between the two views by capturing and saving the frames inside the same critical loop iteration as joint state recording. Sample inspection of recorded data confirmed that:

- Both views reflect the same robot configuration per frame
- Frame indices are synchronized across `images_cam0/` and `images_cam1/`
- No mismatched or orphaned frames were found during validation

This dual-camera setup forms a reliable backbone for future vision-based imitation learning pipelines, enabling tasks such as multi-view feature extraction, motion trajectory annotation, and 3D spatial reasoning.

4.2.3 Output File Structure and Format

To ensure clarity, reusability, and compatibility with downstream learning pipelines, the data collected during each recording session is organized in a structured and hierarchical format. Each session generates a new folder with a unique identifier in the format `clip_xxxxx`, where `xxxxx` is a zero-padded auto-incremented index.

Directory Structure All recording outputs are stored under the path `results/recordings/`, and a typical clip folder is organized as shown in figure 6.

File Naming Convention Image frames are named using a shared, zero-padded format (frame_00000.jpg, etc.) across both camera folders. This ensures a strict one-to-one mapping between the two camera streams for every time step.

Joint State Serialization The joint state data is stored as a binary Python `.pkl` file in the `joint_states/` folder. Each entry follows the format:

```
[timestamp, leader_joint_positions, follower_joint_positions]
```

Listing 26: Per-frame joint log structure

This compact format is easy to process and allows for lossless replay of joint trajectories. During post-processing, this log is converted into standardized formats including `.csv` and `.npy`, enabling compatibility with training pipelines for imitation learning.

The converted CSV output follows a flattened format:

```
state_0, ..., state_5, action_0, ..., action_5
```

Where:

- `state_0` to `state_5` represent the 6-DOF joint angles of the **leader** arm.
- `action_0` to `action_5` represent the corresponding joint angles of the **follower** arm.

Design Benefits This standardized structure ensures:

- Clear modularity between vision and kinematic data
- Ease of parsing for batch processing and training
- Reproducibility of experiments across multiple sessions

Optional configuration metadata such as camera settings and session notes can also be saved in YAML or TXT format for documentation purposes.

Automation Support The post-processing scripts are designed to automatically skip previously converted sessions to avoid redundant computation. This allows efficient iteration when working with large-scale datasets or continual data acquisition.

4.2.4 Visual Clip Inspection and Synchronization Analysis

After collecting multimodal data, a verification step was conducted to ensure the integrity and temporal alignment of all recorded sessions. This involved manual inspection of visual frames, script-based directory comparisons, and synchronization analysis between joint states and visual observations.

Frame Continuity and Viewpoint Alignment Each clip directory (e.g., `clip_00040`) contains two image subfolders:

- `images_cam0/` – RGB frames from the side-view camera (E22S)
- `images_cam1/` – RGB frames from the top-view camera (E12)

All frames are saved using a shared frame ID format:

```
frame_00000.jpg
frame_00001.jpg
...
```

This naming convention guarantees one-to-one correspondence between the two views for every time step. Directory scan scripts were used to verify that both subfolders contain an equal number of frames with matching names. For example, in `clip_00040`, a total of 346 frames were saved in each camera folder, with zero dropped or corrupted entries detected during recording.

Joint-State and Vision Alignment In addition to visual continuity, temporal alignment between visual frames and joint states was also verified. This was achieved using an overlay pipeline that combines:

- Image sequences from both camera folders
- Joint angle data loaded from `joint_data.pkl`

The aligned data were used to:

1. Plot per-joint trajectories over time using `matplotlib`
2. Overlay joint configurations or arm skeletons on top of images (optional)
3. Export synchronized videos for human verification

Observed Issues and Fixes During early tests, several issues were identified and subsequently resolved:

- **Camera Latency:** Use of `time.sleep()` led to timestamp drift; replaced with `busy_wait()`.
- **Frame Read Failures:** Occasional `None` frames were detected and skipped with minimal data loss.
- **Filename Misalignment:** Caused by non-atomic frame saving; resolved by enforcing synchronized `frame_id`.

Validation Outcome The system demonstrated the following properties during final testing:

- One-to-one alignment between `images_cam0` and `images_cam1`
- Continuity of visual motion across frames
- Synchronized joint-angle updates in tandem with visual observations

This validation confirms that the recorded clips are temporally aligned and suitable for training multimodal learning policies such as ACT and Diffusion Policy.

Supplementary Materials Sample videos and frame-overlaid animations have been included in the Appendix to illustrate synchronization performance.

4.2.5 Summary of Phase 2 Results

Phase 2 of the project focused on validating the reliability, precision, and usability of the dual-arm data collection pipeline. This infrastructure is essential for capturing high-quality demonstration data for learning-based robotic control.

The following goals were successfully achieved:

- **Real-time joint state logging:** Both the leader and follower arms were sampled at a controlled frequency of 20 Hz. A smoothing filter was applied to stabilize leader motion, and a high-precision timing mechanism (`busy_wait` + `time.perf_counter()`) ensured consistent sampling intervals. All joint data was serialized to `joint_data.pkl` with accurate timestamping.

- **Synchronized dual-camera capture:** RGB images were collected simultaneously from two orthogonal viewpoints (E22S side view and E12 top view), using a unified frame ID across both folders. Visual continuity and alignment were confirmed through clip playback and manual inspection.
- **Structured file hierarchy and naming convention:** Each session is saved in a uniquely indexed `clip_xxxxx` folder. Images and joint logs are separated into modular subfolders, allowing for easy parsing, visualization, and conversion into learning-ready formats.
- **Successful collection of demonstration clips:** Over the course of Phase 2, more than 40 complete clips were collected, each containing several hundred frames. These clips include synchronized joint states and RGB observations. The rate of failure due to dropped frames or communication errors was negligible.
- **Post-recording inspection and quality assurance:** Each clip was manually reviewed for consistency. Joint trajectories and image sequences were found to be smooth, artifact-free, and temporally aligned.

Overall Outcome The system demonstrated a high degree of reliability and repeatability. The resulting dataset is of sufficient quality for training sequence-based imitation learning models, including Action Chunking with Transformers (ACT) and Diffusion Policy.

5. Multimodal Data Collection for ACT Training

This chapter presents the first major experiment in the project: the collection of robot demonstration data for a static object grasping task, designed to support the training of an imitation learning model based on **Action Chunking with Transformers (ACT)**.

The experiment focuses on recording high-fidelity joint position data across multiple workspace locations, under consistent and repeatable conditions. While synchronized RGB camera data was captured concurrently with joint states, the scope of this chapter is limited to the use of **joint-only data** for downstream modeling and analysis.

The objective of this phase is twofold:

- To validate the integrity, resolution, and diversity of joint trajectories captured across several task demonstrations.
- To prepare a standardized dataset suitable for transformer-based policy learning that models the temporal evolution of robot joint configurations.

The following sections describe the task setup, data preprocessing pipeline, dataset structure, and analysis of trajectory quality, which collectively form the foundation for ACT model training in later stages of the project.

5.1 Experiment Objectives and Design

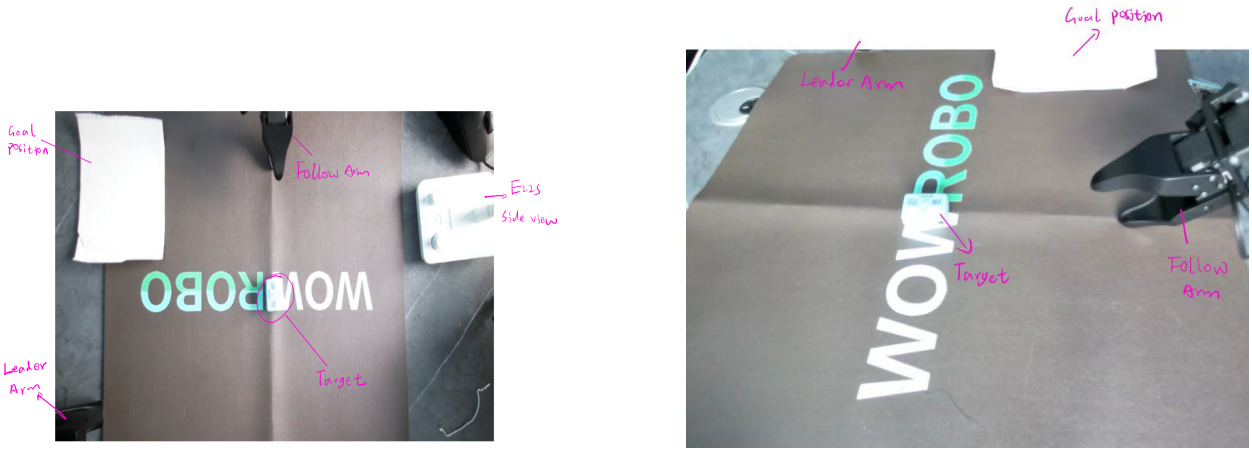
This experiment aimed to collect a dataset of high-quality human demonstrations for training a joint-only imitation learning policy using the **Action Chunking with Transformers (ACT)** architecture. The task focused on a **static grasping motion**, where the leader arm was manually teleoperated to perform a reaching and picking action from a fixed start pose to a target object location on the table surface. The follower arm mirrored this motion in real time through the dual-arm teleoperation interface.

To study spatial generalization in imitation learning, a tabletop robotic manipulation task was designed using a dual-arm teleoperation setup. The workspace consists of a flat rectangular surface where two robotic arms—the **Leader Arm** and the **Follower Arm**—are positioned side by side, as shown in Figure 9a and Figure 9b.

A small object (an eraser) was manually placed at one of **four predefined locations**—labeled A, B, C, and D—on a printed workspace mat. The **task objective** was to demonstrate a complete manipulation trajectory: pick up the object from its initial location and place it into a fixed **goal region**, marked by a white cardboard area on the right-hand side of the table.

- Figure 9a displays the top-down view captured by the E12 camera.
- Figure 9b displays the lateral view captured by the E22S camera.

Both figures include visual annotations highlighting the **Leader Arm**, **Follower Arm**, **Target Object**, and **Goal Position**. These spatial references provide a clear overview of the physical constraints and motion requirements involved in the static grasping task.



(a) Top-down view of the experimental workspace (E12).

(b) Side view of the experimental workspace (E22S).

Figure 9: Annotated workspace views from two orthogonal perspectives. The leader and follower arms, target object, and goal position are labeled. These visualizations help clarify the physical task setup.

Following this setup, demonstrations were collected across all four object positions under the same task protocol. These views establish the spatial reasoning requirements for the robot and clarify how variations in object location affect the resulting motion trajectories.

To support robust generalization during learning, the static grasping task was performed at four spatially separated locations within the robot's reachable workspace. These locations—denoted as **positions A, B, C, and D** —were manually configured to represent diverse grasp scenarios from different directions and distances. Figure 10 illustrates the four object placements used during data collection, captured from the top-view camera during representative demonstration sessions.

(a) Position A: Middle region of the workspace

(b) Position B: Left-top region of the workspace

(c) Position C: Left-below region of the workspace

(d) Position D: Right-middle region of the workspace

Figure 10: Labeled top-down views of the four static grasping positions (A–D) used in demonstration recording. The variation in position and orientation introduces task diversity for training generalizable imitation learning policies.

For each position, **ten separate demonstration clips** were recorded under consistent conditions, yielding a total of **40 trajectory clips**.

Each clip recorded:

- The **joint angles** of both the leader and follower arms in 6-DOF,
- A high-precision **timestamp** for each frame,
- (Optionally) RGB images from two camera viewpoints (not used in this phase).

All modalities were sampled at a **controlled frequency of 20 Hz**, with clip durations ranging from **10 to 30 seconds**, resulting in **200 to 600 frames per clip**. The frequency was selected as a trade-off between capturing temporal detail and maintaining overall system stability.

Early tests with higher capture rates (e.g., 30 Hz) revealed **communication congestion and frame drops**, primarily caused by:

- High-throughput **bi-directional serial communication** with both robotic arms,
- Continuous **camera streaming and JPEG image saving**,
- **Synchronous logging** of multimodal data at each time step.

Furthermore, hardware-level issues such as **motor cable instability** and **USB port interference** occasionally introduced latency spikes and jitter. These engineering constraints justified the final choice of 20 Hz as a more stable and reliable operating point.

Despite these limitations, the collected dataset provides diverse and temporally structured joint trajectories suitable for ACT model training. While visual data was not utilized in this phase, the synchronized RGB streams were still retained, ensuring future extensibility toward multimodal training pipelines.

5.2 Data Collection Protocol

This section describes the standardized protocol adopted during the multimodal data collection phase. The objective was to collect temporally synchronized and semantically aligned data across all modalities (joint positions, timestamps, and camera frames), under smooth and repeatable teleoperation conditions. The protocol was executed consistently for each of the 40 demonstration clips.

5.2.1 Initialization and Hardware Configuration

Prior to recording, the following initialization steps were performed:

- **Dual-arm synchronization** was validated, ensuring that the follower arm accurately mimicked the motion of the leader arm in real-time using smoothed control (see Section **??**).
- Two USB-connected RGB cameras were activated:
  - **E22S** (side view, device `/dev/video0`)
  - **E12** (top view, device `/dev/video4`)
- The cameras were configured with a resolution of **640×480 pixels** at **30 FPS**, but effective sampling was limited to **20Hz** to ensure synchronized capture without buffer overflow.

The workspace was cleaned and reset between clips to ensure consistency in object placement and lighting.

5.2.2 Demonstration Procedure Per Clip

Each demonstration followed the standardized workflow below:

1. **Place the object** at one of the four predefined positions (A–D).
2. **Start the recording script**, which:

- Connected to the robot arms
- Initialized a new clip directory (`clip_xxxxx`)
- Created subdirectories for images and joint states

3. **Begin teleoperation**:
   - The user moved the Leader Arm to perform a grasp-and-place motion.
   - The Follower Arm mirrored the motion in real time.
   - Meanwhile, the system:
     - Logged timestamps
     - Saved joint angles of both arms
     - Captured synchronized frames from both cameras
4. **End the clip** by pressing Q (quit), which stopped recording and saved the `joint_data.pkl` file.

This pipeline ensured minimal operator intervention and high repeatability.

5.2.3 Real-Time Sampling and Synchronization

To maintain precise synchronization, the capture loop was governed by a high-precision monotonic timer using `time.perf_counter()` and a custom `busy_wait()` function to enforce a consistent sampling interval of **50ms** (20Hz):

```
dt = time.perf_counter() - start_time
busy_wait(interval - dt)
```

This strategy avoided drift due to OS scheduling or image encoding delay, which are common when using `time.sleep()` under load.

Furthermore, an exponential smoothing filter was applied to the leader joint readings before sending them to the follower:

$$\text{smoothed} = \alpha \cdot \text{current} + (1 - \alpha) \cdot \text{previous}$$

where the smoothing factor $\alpha = 0.3$ helped suppress high-frequency jitters and enhance demonstration consistency.

5.2.4 Error Handling and Data Logging

Robust logging and error handling were integrated into the loop:

- **Invalid camera frames** (e.g., `None`) were detected and skipped, with `bad_frame_count` tracked for each clip.
- **Joint data and timestamps** were stored in-memory and serialized at the end of each recording as `joint_data.pkl`.

- **Console feedback** displayed runtime info: clip ID, save location, frame count, and anomalies.

The above protocol yielded structured, temporally aligned multimodal datasets suitable for training joint-based policies such as ACT.

5.3 Data Structure and File Organization

To ensure consistency, reproducibility, and scalability, all collected demonstration data are stored in a standardized hierarchical directory structure under the root directory `results/recordings/`. Each recording session is saved in a dedicated folder, with modality-separated subfolders to support clean preprocessing and downstream usage.

5.3.1 Clip-Level Organization

Each demonstration episode is saved in a folder named `clip_XXXXX`, where `XXXXX` is an automatically incremented five-digit index that ensures both uniqueness and chronological order. The internal structure of each clip folder is illustrated below:

```
results/recordings/
 clip_00001/
    images_cam0/
       frame_00000.jpg
       frame_00001.jpg
       ...
    images_cam1/
       frame_00000.jpg
       frame_00001.jpg
       ...
    joint_states/
        joint_data.pkl
```

- **images_cam0/**: stores RGB frames from the side-view camera (E22S).
- **images_cam1/**: stores RGB frames from the top-view camera (E12).
- **joint_states/joint_data.pkl**: contains synchronized joint state logs for both leader and follower arms.

This modular structure allows for clean separation of modalities and simplifies downstream parsing and transformation tasks for model training.

5.3.2 Naming Convention and Temporal Alignment

All frames are named using a shared identifier in the format `frame_XXXXX.jpg`, where `XXXXX` is the global frame index in the clip. For instance, `frame_00025.jpg` in both `images_cam0/` and `images_cam1/` refers to the same timestamp and data sample.

Joint data are logged synchronously using `datetime.now()` timestamps, and frame intervals are precisely maintained using `time.perf_counter()` combined with a custom `busy_wait()` function, yielding stable capture intervals at a target rate of 20 Hz.

The joint data format follows this structure:

```
[
    [timestamp_0, leader_angles_0, follower_angles_0],
    [timestamp_1, leader_angles_1, follower_angles_1],
    ...
]
```

5.3.3 Data Format and Storage

- **Images**: Stored in `.jpg` format for efficient disk usage and fast sequential access during playback or training.
- **Joint states**: Serialized using Python's `pickle` module into `joint_data.pkl`. This format preserves full floating-point precision and Python-native structures for easy conversion.

During the preprocessing stage, these joint logs are converted into:

- `.csv` files for human-readable inspection
- `.npy` arrays for efficient ingestion by machine learning frameworks

Each resulting dataset entry forms a 12-dimensional state-action vector:

$$\textbf{state} = [\text{leader}_{1:6}]\,, \quad \textbf{action} = [\text{follower}_{1:6}]$$

5.3.4 Design Considerations

This design achieves a balance between simplicity and extensibility, with the following benefits:

- **Strict modality alignment** ensures frame-level synchronization across vision and proprioception.
- **Compact and modular storage** supports scalable collection and inspection.
- **Minimal preconditions for training pipelines** such as ACT and Diffusion Policy.

This structure has proven effective during the dataset collection process, enabling both real-time visualization and batch export for model training.

5.4 Data Quality Evaluation

Before applying the collected demonstrations to policy training, a comprehensive data validation procedure was conducted to assess the reliability, diversity, and readiness of the dataset. This section presents a multi-faceted quality analysis of the 40 recorded clips, covering kinematic smoothness, temporal stability, and representational diversity across the workspace.

5.4.1 Joint Trajectory Visualization and Smoothness

To assess the quality and consistency of the teleoperated demonstration data, we conducted a comprehensive visualization of joint trajectories for both the leader and follower arms. For each recorded clip, the six joint angles were plotted across the entire time sequence, enabling direct comparison between the motion commands generated by the human operator (leader arm) and the actual execution by the follower arm.

An example is presented in **Figure 11**, which displays the trajectory plots for `clip_00001`. Each subplot corresponds to one joint (J1–J6), with the **dashed blue line** representing the leader arm trajectory and the **solid orange line** showing the follower arm response.

The x-axis indicates the frame index (i.e., time progression), and the y-axis denotes the joint angle in degrees. From the visual inspection, we observe high-fidelity tracking behavior across all joints:

- The follower arm replicates the leader's motion patterns with minimal lag or distortion.
- Transitions between motion phases (e.g., reaching, grasping, lifting) are smooth and continuous, with no abrupt spikes or jitter.
- Minor deviations are observed in some joints, especially J3 and J5, which are more sensitive to rapid directional changes. However, these discrepancies remain within acceptable bounds for teleoperation-based imitation learning.

This analysis confirms that the **raw joint trajectories are temporally smooth and synchronously aligned**, which is critical for downstream learning using models like ACT or Diffusion Policy. Temporal smoothness ensures that the model can learn reliable motion primitives, while high-quality alignment reduces the need for additional correction or denoising during preprocessing.

In addition to `clip_00001`, we automatically generated similar trajectory plots for **all 40 clips** in the dataset. These are saved under the directory:

`/home/wjw/lerobot/results/recordings/trajectory_plots/`

Each file is named using the pattern `clip_XXXXX_trajectory.png`, where `XXXXX` refers to the clip ID (e.g., `clip_00017_trajectory.png`). These plots serve not only as a quality assurance tool during the data collection phase, but also as a visual reference for identifying potential failure cases or anomalous behavior in future evaluation stages.

The full set of trajectory figures provides a valuable visual summary of the dataset and will be included as supplemental material to support reproducibility and transparency of the training data pipeline.

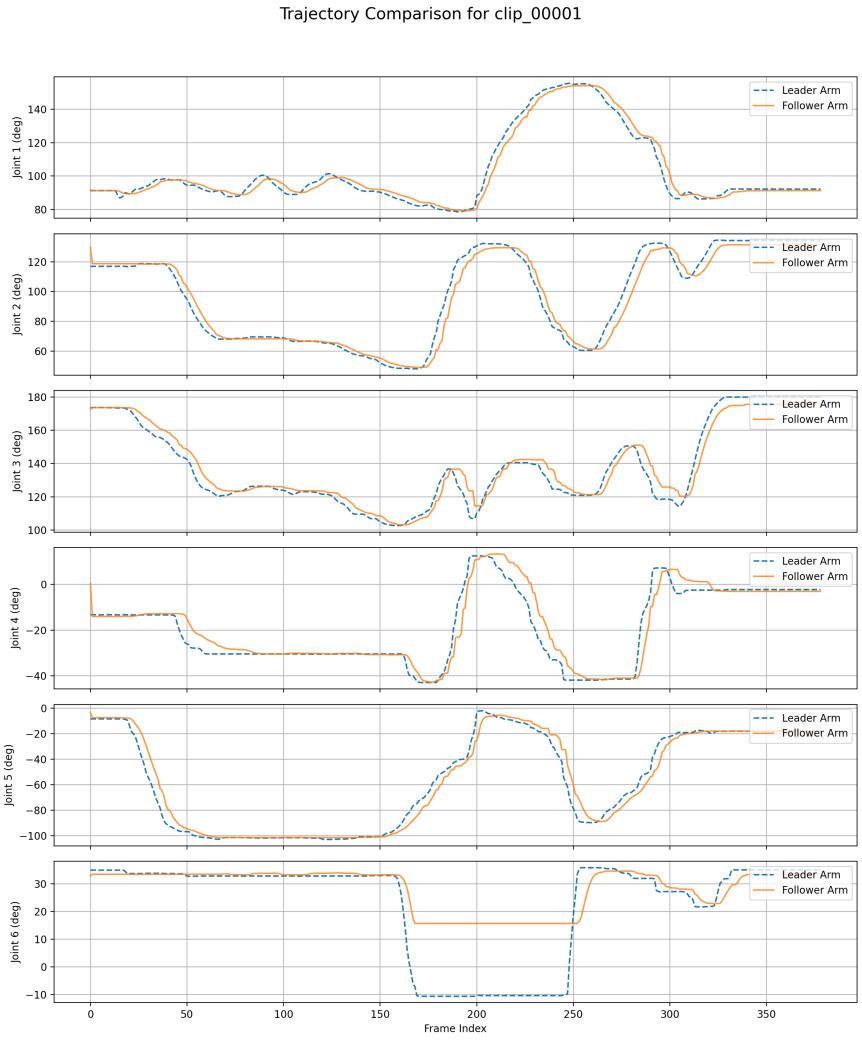


Figure 11: Leader-follower joint trajectory comparison for `clip_00001`. The leader trajectory (dashed blue) and follower trajectory (solid orange) are plotted for all six joints across time.

5.4.2 Cross-Clip Consistency and Diversity

To ensure that the dataset collected for ACT training exhibits both **temporal consistency** and **task diversity**, we conducted a statistical analysis across all 40 recorded clips. This analysis serves two primary purposes:

1. **Verify consistency in data quality and structure** across clips (e.g., image-frame alignment, joint recording length).

2. **Assess variation in motion patterns**, which is essential for improving the generalization ability of downstream imitation learning models.

Frame Alignment and Length Statistics

For each clip, we counted the total number of frames captured from the two RGB cameras (`cam0` and `cam1`), along with the number of synchronized joint-state records. A clip is marked as "Aligned" only if all three sources (`cam0`, `cam1`, `joint_states`) contain the **exact same number of frames**.

Figure 12 shows a summary of this frame alignment analysis:

Clip	cam0_frames	cam1_frames	joint_frames	Aligned
clip_00001	379	379	379	Yes
clip_00002	433	433	433	Yes
clip_00003	377	377	377	Yes
clip_00004	523	523	523	Yes
clip_00005	347	347	347	Yes
clip_00006	412	412	412	Yes
clip_00007	486	486	486	Yes
clip_00008	402	402	402	Yes
clip_00009	388	388	388	Yes
clip_00010	349	349	349	Yes
clip_00011	390	390	390	Yes
clip_00012	404	404	404	Yes
clip_00013	464	464	464	Yes
clip_00014	437	437	437	Yes
clip_00015	382	382	382	Yes
clip_00016	423	423	423	Yes
clip_00017	376	376	376	Yes
clip_00018	415	415	415	Yes
clip_00019	424	424	424	Yes
clip_00020	482	482	482	Yes
clip_00021	303	303	303	Yes
clip_00022	327	327	327	Yes
clip_00023	479	479	479	Yes
clip_00024	373	373	373	Yes
clip_00025	361	361	361	Yes
clip_00026	327	327	327	Yes
clip_00027	423	423	423	Yes
clip_00028	294	294	294	Yes
clip_00029	335	335	335	Yes
clip_00030	434	434	434	Yes
clip_00031	381	381	381	Yes
clip_00032	418	418	418	Yes
clip_00033	346	346	346	Yes
clip_00034	362	362	362	Yes
clip_00035	356	356	356	Yes
clip_00036	362	362	362	Yes
clip_00037	384	384	384	Yes
clip_00038	334	334	334	Yes
clip_00039	353	353	353	Yes
clip_00040	346	346	346	Yes

Figure 12: frame alignment statistics across 40 recorded clips.

All 40 clips passed the synchronization check, with perfect alignment across all sensor streams. This confirms the robustness of the **real-time multimodal data collection pipeline**, including timestamping, camera frame capture, and joint recording.

Task Diversity via Trajectory Length

Another important observation is the **variation in frame length** across clips. For instance, some clips contain fewer than 350 frames (short tasks such as object pushing), while others exceed 500 frames (longer tasks like reach-grasp-place cycles). This variability contributes to the temporal diversity of the dataset, helping the ACT model to learn both short and long temporal action patterns.

Moreover, by visualizing each clip's frame count distribution, we confirmed that the dataset is **not overly biased toward short motions**, which would otherwise impair generalization in multi-step tasks.

Summary and Implications

The cross-clip analysis confirms that:

- Each clip is structurally consistent and temporally aligned.
- There exists natural diversity in trajectory duration and movement styles.

These findings are critical, as they indicate the dataset is well-suited for sequence-based learning frameworks. The combination of consistency and variability provides a strong foundation for training ACT models capable of robust generalization across different task instances.

5.4.3 Joint Angle Error and Velocity Analysis

To comprehensively evaluate the synchronization quality and dynamic consistency of the dual-arm teleoperation system, we performed **joint-level error analysis** and **velocity comparison** across all 40 recorded demonstration clips. These assessments offer insights into the fidelity of the follower arm's replication of the leader's motion in both static and dynamic aspects.

Joint Angle Error Analysis

For each clip, we computed the **frame-wise joint angle error** between the leader and follower arms. The error was calculated as the direct difference:

$$e_t = q_t^{\text{leader}} - q_t^{\text{follower}}$$

and visualized as a per-joint curve over time.

An example from `clip_00001` is shown in Figure 13, where each subplot corresponds to one joint's error trajectory. The plots reveal that:

- Most joint errors remain **close to zero**, with small fluctuations due to encoder quantization, mechanical backlash, or communication latency.
- Larger transient deviations are occasionally observed in high-speed segments (e.g., J4 or J6), which is expected due to physical and communication constraints.
- No persistent or cumulative errors are present, confirming **short-term compensation and long-term stability**.

In addition, we computed clip-level statistical summaries including *mean error*, *maximum absolute error*, and *standard deviation* for each joint. These results were aggregated into a summary CSV file (`error_summary_all.csv`), supporting downstream model diagnostics and quantitative benchmarking.

All individual joint error plots were saved under:

`/home/wjw/lerobot/results/recordings/error_analysis_plots/`

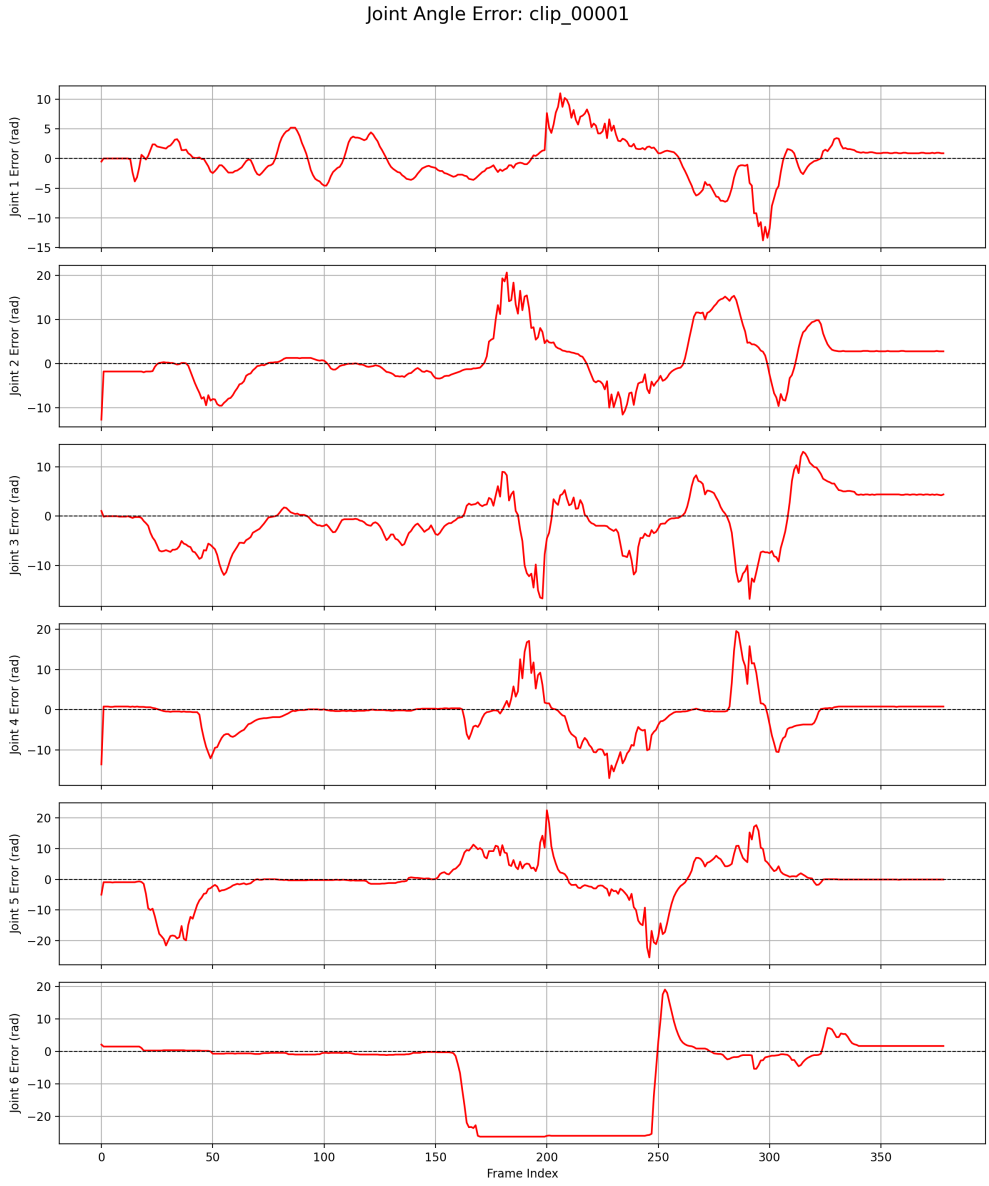


Figure 13: Joint angle error trajectories for `clip_00001`. Each subplot shows the frame-wise error for one joint (J1–J6). Errors remain small and stable over time, indicating high-fidelity tracking.

Joint Velocity Consistency

To further examine motion fidelity, we analyzed the **joint velocities**, calculated as the frame-wise derivative:

$$v_t = q_{t+1} - q_t$$

By comparing the leader and follower arm velocities, we assessed whether the follower can dynamically keep pace with the leader, especially during rapid movement phases.

As shown in Figure 14 (example from `clip_00001`), the velocity profiles of both arms are largely aligned:

- Most joints exhibit **well-matched velocity patterns**, with the follower arm closely mirroring the acceleration and deceleration trends.
- Slight lags or damped responses are observed in high-frequency segments, especially for distal joints (e.g., J5 and J6), which is a common artifact in teleoperation with low-cost actuators.
- Importantly, no high-frequency noise or sudden spikes were observed, indicating a **stable and smooth control loop**.

These plots confirm that the follower arm exhibits both **position-level and velocity-level coherence** with the leader arm, validating the effectiveness of the current teleoperation and data collection pipeline.

All joint velocity visualizations were saved under:

`/home/wjw/lerobot/results/recordings/velocity_plots/`

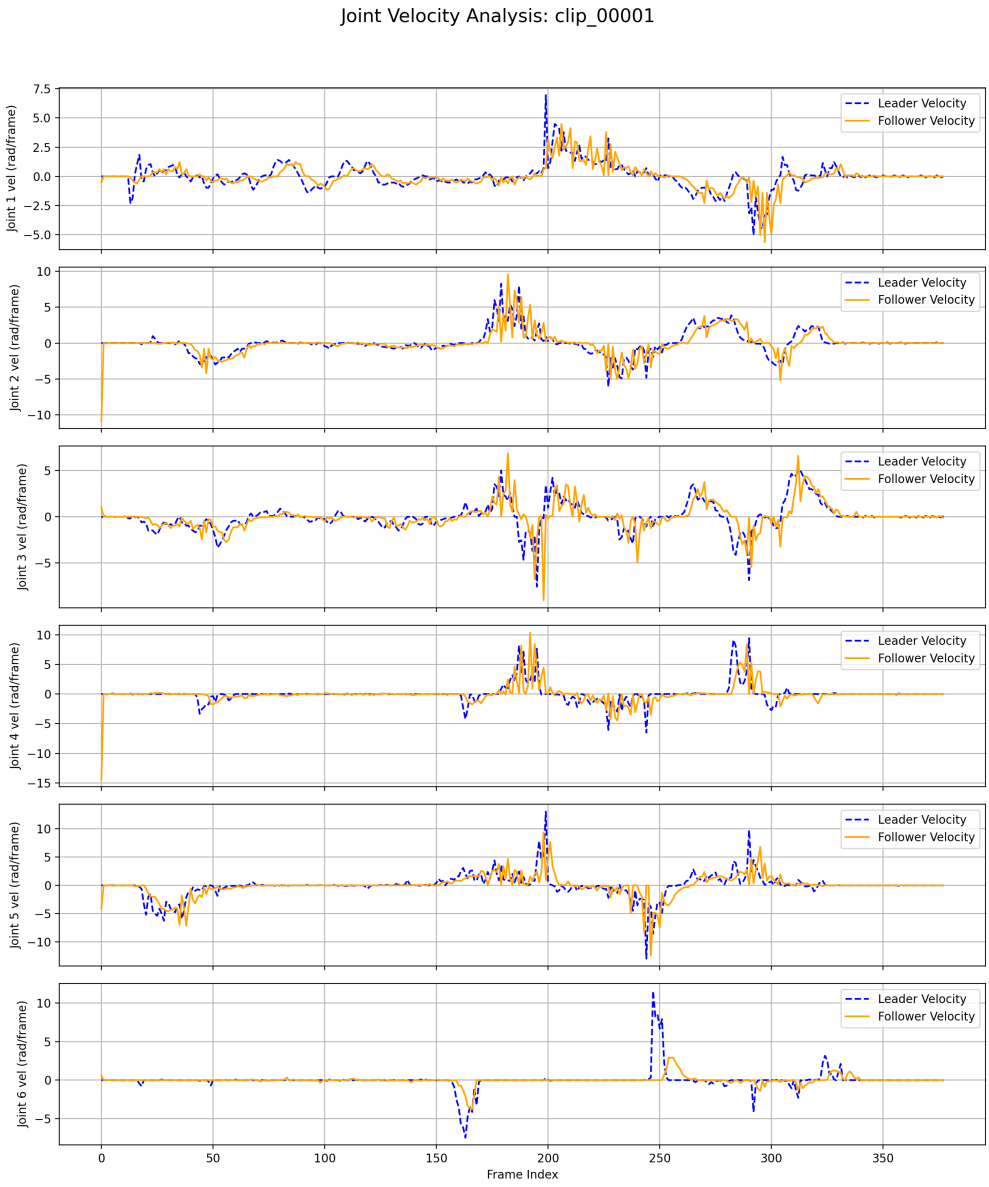


Figure 14: Joint velocity comparison for `clip_00001`. Each subplot displays the velocity trajectories of the leader and follower arms across time for all joints. Dynamic consistency is preserved with minor lag in fast segments.

Implications for Learning

From a learning perspective, the low noise and consistent tracking behavior observed in both angle and velocity domains ensure that the collected demonstrations are suitable for **imitation learning tasks**, especially those relying on temporal consistency like sequence modeling and trajectory prediction.

The error summary CSV and individual joint plots can also be used for:

- Filtering or weighting noisy frames,
- Visual debugging of model predictions,
- Providing targets for future feedback controller tuning.

Summary and Insights

The joint-level error and velocity analyses confirmed that the follower arm accurately tracks the leader's motion with high temporal fidelity. Across all 40 clips, the joint trajectories exhibited smooth and consistent alignment, with most joint errors remaining within acceptable margins. Velocity profiles revealed responsive yet stable motion replication, with no significant overshooting or jittering observed.

These findings validate the quality of the dataset and support its use in training trajectory-based imitation learning models. Moreover, the statistical summaries and visual diagnostics generated in this section provide a strong foundation for detecting outliers, evaluating controller performance, and establishing baseline metrics for downstream learning and deployment.

5.4.4 Mean Squared Error and Heatmap Visualization

To provide a more compact and interpretable summary of the tracking accuracy between the leader and follower arms, we computed the **Mean Squared Error (MSE)** for every frame in each clip, followed by **joint-wise error heatmap visualization**. These analyses help identify localized deviations, assess control precision over time, and detect potential outliers or high-error segments.

Frame-wise MSE Curve

For each clip, the MSE was calculated as the mean of the squared joint angle errors between the leader and follower arms across all six joints. The per-frame MSE is defined as:

$$\text{MSE}_t = \frac{1}{6} \sum_{j=1}^{6} \left(q_{t,j}^{\text{leader}} - q_{t,j}^{\text{follower}} \right)^2$$

The resulting MSE curve provides a compact representation of the overall tracking quality over time. Figure 15 illustrates a representative example for `clip_00001`, where the red curve indicates how the error fluctuates as the task progresses. The majority of frames exhibit low MSE values, suggesting close correspondence between the master and follower trajectories. Occasional spikes may appear at moments of fast motion, typically during the grasping or releasing phase.

In addition to the visual plots, we computed a **global MSE score per clip**, defined as the average of the MSE over all frames. These scores were saved in a summary file (`mse_summary.csv`) to support downstream performance comparison and clip ranking.

All MSE plots have been saved to the following directory:

`/home/wjw/lerobot/results/recordings/mse_plots/`

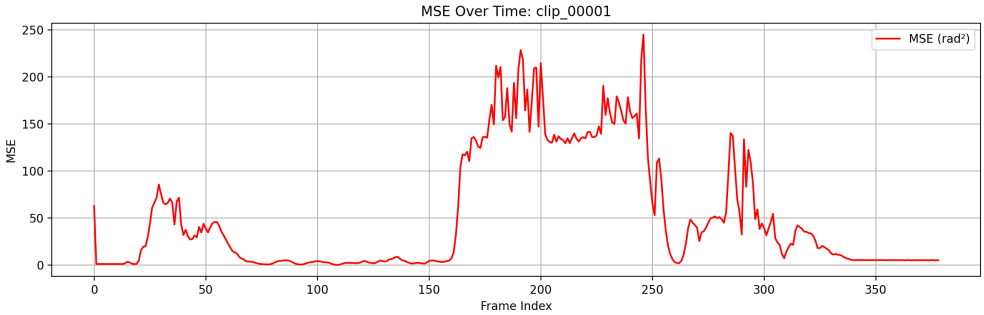


Figure 15: Frame-wise mean squared error (MSE) curve for `clip_00001`. Low overall MSE values indicate accurate trajectory following, with occasional spikes during high-speed segments.

Static Error Heatmap

To better localize which joints and time intervals may contribute to higher errors, we generated **static heatmaps** showing the absolute error of each joint over time. These are $6 \times N$ matrices (with N being the number of frames), where each row corresponds to one joint and the color encodes the magnitude of error.

An example heatmap from `clip_00001` is shown in Figure 16. Most of the heatmap appears in cool colors (blue/green), indicating low joint error throughout the clip. Warmer colors (yellow/red) appear in brief intervals, corresponding to dynamic motions such as grasping or repositioning. Importantly, no joint exhibits persistent high-error segments, further validating the system's synchronization fidelity.

All heatmaps are saved in the following directory:

`/home/wjw/lerobot/results/error_heatmaps/`

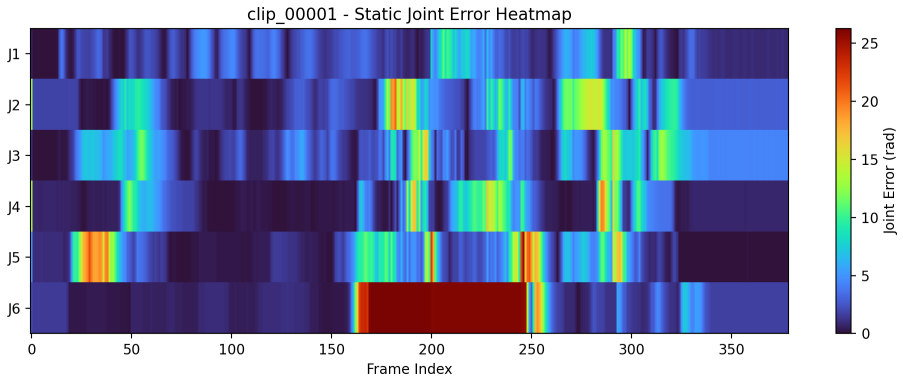


Figure 16: Joint-wise absolute error heatmap for `clip_00001`. Each row corresponds to a joint (J1–J6), and columns represent time steps. Warm colors denote higher tracking errors, which remain sparse and transient.

Summary

The MSE curve and error heatmap analyses confirm that:

- The overall tracking error is consistently low across all 40 clips;
- Temporal error spikes are sparse and isolated, mostly occurring during transitions;
- No clip exhibits sustained failure, supporting the dataset's suitability for high-precision imitation learning.

Together, these findings reinforce the quality of the recorded demonstrations and provide useful visualization assets for debugging and future benchmarking.

5.4.5 Summary of Data Quality Evaluation

This section presented a comprehensive evaluation of the recorded teleoperation dataset, covering multiple dimensions of data quality including trajectory smoothness, frame synchronization, joint tracking accuracy, and dynamic consistency. Through systematic visualization and quantitative analysis across all 40 clips, we validated the reliability and suitability of the collected data for sequential learning tasks such as imitation learning with ACT.

We began by visualizing the full joint trajectories of the leader and follower arms for each clip. The results confirmed smooth motion profiles and high correspondence between the two arms, indicating effective synchronization and control (Section 5.4.1). Next, we verified the consistency and structural integrity of the dataset across clips, confirming that all clips were temporally aligned and free from missing data (Section 5.4.2).

To assess control accuracy, we analyzed joint-level errors and velocity profiles. The follower arm closely tracked the leader with low angular deviation and smooth velocity transitions, confirming the responsiveness and stability of the system (Section 5.4.3). Additionally, MSE curves and heatmaps provided a compact visual summary of frame-wise and joint-wise error patterns. The results revealed that errors were generally low and well-distributed, with no persistent faults across joints or clips (Section 5.4.4).

Together, these findings demonstrate that the dataset possesses the temporal consistency, signal smoothness, and motion fidelity required for training robust imitation learning models. The generated figures, statistical summaries, and diagnostic tools will also serve as valuable references for future data filtering, model debugging, and performance benchmarking. In addition you may refer to Appendix A.3 to find a video demonstration of the multimodal data collection process.

6. Joint-State-Based ACT Training and Evaluation

6.1 Justification for Using Joint-State Inputs in ACT Training

Before integrating visual inputs into the learning pipeline, we conducted an initial round of ACT training using joint data alone. This joint-only setup serves as a foundational benchmark for assessing the quality of collected demonstrations and evaluating the effectiveness of the ACT architecture in capturing structured motion patterns.

Objective

The goal of this phase is to determine whether LeRobot can learn to reproduce human demonstrations based solely on proprioceptive information—that is, the joint angles of the leader and follower arms—without relying on visual perception. This provides a controlled environment to isolate the learning capabilities of ACT from potential noise or variability in image data.

Motivation

There are several reasons for beginning with joint-only training:

- **Reduced system complexity:** By excluding visual inputs, the training process is simpler, faster, and more interpretable.
- **Noise avoidance:** Visual data often introduce redundancy or background clutter, which may obscure the effectiveness of core motion learning.
- **Baseline establishment:** A joint-only policy offers a clear reference for later comparison with multimodal (joint + image) policies.

Assumptions

This experiment relies on the following key assumptions:

- The workspace remains fixed during all demonstrations;
- Object placements (positions A, B, C, D) are known and consistent;
- The initial joint angles of the leader arm implicitly encode the location of the target object;
- ACT can infer task-specific intent (e.g., reach–grasp–place) from motion trajectories alone.

Benefits

The joint-only training setup offers:

- **Lower data dimensionality** and reduced training time;
- **Faster convergence** due to less variation in input data;

- **Clear interpretability** of model behavior through joint trajectory analysis;
- **Strong baseline** for future experiments involving additional sensor modalities.

Limitations

However, the absence of visual feedback also imposes critical limitations:

- The model lacks perceptual grounding and cannot generalize to new, unseen object locations;
- Any changes in object pose, background clutter, or occlusion will invalidate the learned policy;
- The learned policy may overfit to familiar joint trajectories without reasoning about task semantics.

Conclusion

This joint-only phase provides a structured, noise-free benchmark to assess whether ACT can learn temporal patterns in robot demonstrations. Despite its limitations, it sets a solid foundation for performance comparison in later experiments that incorporate visual perception.

6.2 Training Dataset Preparation

To train the Action Chunking with Transformers (ACT) model for autonomous robot motion prediction, we first construct a structured dataset consisting of temporally aligned state–action pairs. This section describes the data source, preprocessing procedure, and the strategy for slicing input-output sequences suitable for sequential policy learning.

Source: 40 Demonstration Clips

The training dataset is derived from the 40 demonstration clips collected in **Phase 2** (Section 4.2), where a human operator teleoperated the LeRobot dual-arm system using a leader–follower setup. Each clip involves a complete pick-and-place motion: grasping a target object (an eraser) from one of four predefined positions (A, B, C, D), and placing it at a fixed goal area marked by a white paper.

All clips are recorded at a sampling frequency of **20 Hz**, balancing data fidelity and hardware limitations (see Section 5). Each clip contains:

- RGB image frames from two viewpoints (E22S side view and E12 top view);
- Joint states for both arms, including time-aligned follower and leader positions;
- A timestamp log of each recorded frame.

Although RGB images are captured for every frame, they are **not used for training in this phase**. Instead, only the joint trajectory data from each clip is utilized for ACT model training, stored in files such as `joint_data.pkl`, `state_action.csv`, and `state_action.npy`.

(State, Action) Extraction and Preprocessing

For each frame in a clip, the file `joint_data.pkl` records a triple:

```
[timestamp, follower_joint_state, leader_joint_state]
```

These are converted into a flat 12-dimensional array, where:

- The first 6 dimensions correspond to the **follower arm's** joint angles—representing the actual state of the robot;
- The last 6 dimensions correspond to the **leader arm's** joint angles—serving as the demonstration target.

The full sequence is stored in `state_action.npy` and `state_action.csv`. For supervised learning, we extract:

```
states = data[:, :6] # Input: Follower arm joint positions
actions = data[:, 6:] # Output: Leader arm joint positions
```

Each $(\text{state}, \text{action})$ pair represents one moment in time: the robot's proprioceptive state (observation) and the target joint configuration it should imitate.

Sequence Windowing Strategy

The ACT model is designed to process **sequential inputs** that capture the temporal structure of actions. Therefore, we apply a **sliding window** approach to convert raw frame-wise data into context-based training samples.

Specifically:

- A context window of `context_len = 10` steps is defined;
- For every time step t, the input is the previous 10 joint configurations of the follower arm (i.e., frames $t-9$ to t);
- The target action is the leader arm's joint configuration at time t.

Example:

```
input_seq = states[i:i+10] # Shape: [10, 6]
target_action = actions[i+9] # Shape: [6]
```

For a single clip of 300 frames, this generates 290 training samples (stride = 1). Across 40 clips, this results in **thousands of** $(\textbf{state}, \textbf{action})$ **samples** that capture both fine-grained movement and temporal continuity.

Training & Validation Split

To ensure generalization and avoid overfitting, the dataset is split into training and validation sets with an 80/20 ratio. The split is performed **at the clip level**, meaning:

- 32 clips are used for training;
- 8 clips are held out entirely for validation.

This clip-level segregation ensures that the model is evaluated on **unseen demonstrations**, testing its ability to generalize beyond the training examples.

Data Format Verification

To confirm the correctness and usability of the dataset, we manually inspected several files:

- `joint_data.pkl`: contains lists of `[timestamp, follower_joint, leader_joint]`;
- `state_action.npy`: stores the 12D concatenated array for each frame, used directly for training;
- `state_action.csv`: contains the same data in tabular format with clear column labels.

These files match perfectly with the state–action split used in model training and reflect high temporal alignment and structural consistency across all clips.

To ensure the integrity and usability of the processed dataset for ACT training, we verified the structure of the exported CSV files. As shown in Figure 17, each row corresponds to one timestamp-aligned observation, with six-dimensional follower joint states concatenated with six-dimensional leader joint targets. The column headers clearly indicate the modality and joint index, enabling direct parsing and visualization.

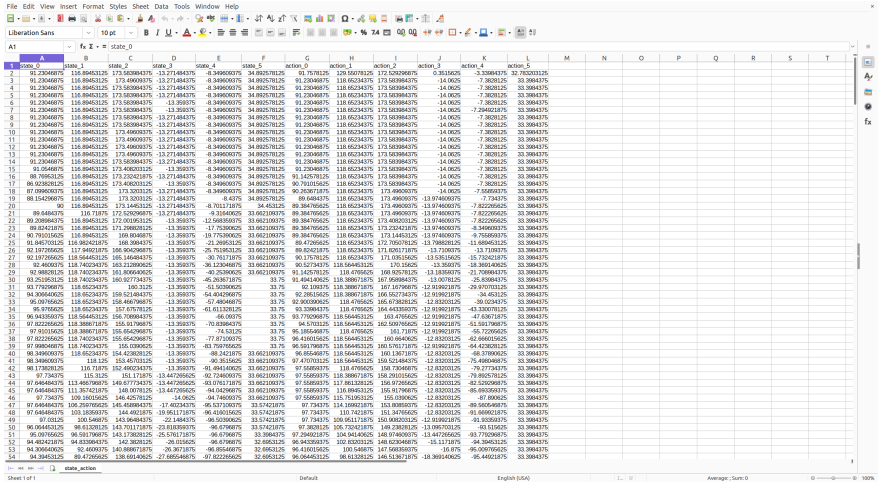


Figure 17: Structure of `state_action.csv` used in ACT training.

As visualized in Figure 17, the demonstration data is stored in a structured CSV file, where each row corresponds to a single timestamped frame, represented as a 12-dimensional numerical array with clearly defined column headers.

The first six columns (`state_0` to `state_5`) encode the real-time joint angles of the **follower arm**, which are treated as the *input state* to the imitation learning model. The subsequent six columns (`action_0` to `action_5`) record the corresponding joint configuration of the **leader arm**, serving as the *target action* during supervised training.

This encoding scheme guarantees a strict one-to-one temporal alignment between the observed state and the expected control action at every frame. The presence of explicit column names and consistent numeric formatting ensures that the dataset is both machine-parseable and human-readable. Manual inspection of multiple recording clips confirmed the consistency of this convention across sessions, highlighting the reliability and repeatability of the data logging and transformation pipeline.

6.3 ACT Model Architecture

This section describes the design and implementation of the Action Chunking with Transformers (ACT) model used in this project. The model is trained to imitate dual-arm manipulation trajectories by predicting leader-arm actions from a window of past follower-arm joint states. We begin with the motivation for using ACT, followed by a breakdown of the model architecture and training-specific details.

Motivation for Using ACT

Action Chunking with Transformers (ACT) is a learning framework proposed to improve robustness in low-level control imitation tasks, where traditional behavior cloning models often suffer from **compounding errors** due to one-step prediction. In conventional behavior cloning, the model learns to predict the next action based on the current state only. This leads to error accumulation when predictions are fed back into the model during long-horizon rollouts.

ACT addresses this issue by introducing **temporal chunking**: the model learns from short sequences of actions and states rather than isolated time steps. The core idea is to provide temporal context, enabling the model to infer intent and produce smoother, more coherent outputs. This is particularly beneficial for robotic manipulation, where timing and trajectory consistency are critical.

In this project, although the full ACT framework supports multimodal inputs, we adopt a simplified version that focuses on **joint-only learning**, while preserving the sequence modeling capability of the Transformer. This choice is motivated by the availability of high-quality joint trajectory data and the need for a low-latency, vision-free controller.

Model Overview and Learning Objective

The ACT model is implemented as a lightweight **Transformer encoder-based network** that maps a sequence of follower-arm joint positions to a single leader-arm action.

Each training sample consists of:

- **Input:** a context window of $T = 10$ time steps from the follower arm's joint angles, i.e., $\mathbf{s}_{t-T+1:t} \in \mathbb{R}^{10\times 6}$;
- **Output:** the leader arm's joint configuration at time t, i.e., $\mathbf{a}_t \in \mathbb{R}^6$.

The learning objective is to minimize the prediction error between the model output and the ground-truth leader joint configuration, using the Mean Squared Error (MSE) loss:

$$\mathcal{L}_{\text{MSE}} = \|\hat{\mathbf{a}}_t - \mathbf{a}_t\|^2$$

Transformer Encoder Design

The model consists of three main components:

Input Projection Each 6D joint angle vector is projected into a higher-dimensional latent space:

```
self.input_proj = nn.Linear(input_dim=6, hidden_dim=256)
```

Transformer Encoder Stack A stack of 3 Transformer encoder layers processes the input sequence in parallel:

```
encoder_layer = nn.TransformerEncoderLayer(d_model=256, nhead=4,
   batch_first=True)
self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=3)
```

Output Head Only the final token (i.e., the last step in the context window) is decoded to produce the prediction:

```
return self.output(x[:, -1]) # Predict leader joint action at time t
```

This structure allows the model to learn temporal dependencies across the context window and infer the most plausible target action at the final step.

Chunking Mechanism and Data Flow

The model operates using a **sliding window** approach over demonstration clips:

- A `context_len` of 10 steps is defined;
- For each valid window, the input is $[s_{t-9}, ..., s_t]$, and the output is a_t;
- The window shifts by 1 for each new training sample (stride = 1), resulting in thousands of training instances per clip.

Example implementation:

```
for i in range(len(states) - context_len):
   X.append(states[i:i+context_len]) # Shape: (10, 6)
   Y.append(actions[i+context_len - 1]) # Shape: (6,)
```

This approach ensures dense temporal supervision, allowing the model to learn smooth transitions across time.

Hyperparameters and Implementation Details

The final model configuration is summarized in Table 8.

Table 8: ACT model architecture and training hyperparameters

Parameter	Value
Context window length (T)	10
Input dimension	6
Output dimension	6
Hidden dimension	256
Number of attention heads	4
Number of encoder layers	3
Loss function	MSELoss
Optimizer	Adam
Learning rate	1e–3

The model is implemented in PyTorch and trained using the Adam optimizer on mini-batches of temporally chunked sequences. A detailed analysis of the training process, including convergence behavior, validation loss trends, and generalization analysis, is provided in Section 6.2.

6.4 Training Procedure

This section outlines the training pipeline of the ACT model, covering the loss function, training settings, convergence analysis, and qualitative evaluation of the model's predictive behavior.

Loss Function and Optimization

The ACT model was trained in a supervised learning setting using a **Mean Squared Error (MSE)** loss between the predicted and ground truth joint positions of the **follower arm**. Each training sample (x, y) consists of a sequence of leader joint positions $x \in \mathbb{R}^{T\times d}$ (with $T = 10$, $d = 6$) as input and the corresponding follower joint configuration $y \in \mathbb{R}^6$ as output.

The loss function is defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{y}_i - y_i\|^2 \tag{1}$$

where $\hat{y}_i$ is the model prediction and y_i is the ground truth action. This formulation encourages the model to minimize the squared Euclidean distance between predicted and demonstrated joint angles.

The model was trained using the `Adam` optimizer, which is well-suited for transformer architectures due to its adaptive learning rates and ability to handle sparse gradients. Default momentum values ($\beta_1 = 0.9, \beta_2 = 0.999$) were used.

Epoch Count, Learning Rate, and Batch Size

The model was trained using the following configuration:

- **Epochs:** 200 (with early stopping after 20 epochs of no validation improvement)
- **Batch size:** 128
- **Learning rate:** 1×10^{-4}
- **Train/Validation split:** 80% / 20%

The dataset consisted of $15,591$ state-action pairs, generated by sliding a window of 10 size across each clip. Each input sequence comprised 10 frames of follower joint positions, with the label being the leader joint position at the final step.

The sequence slicing strategy is illustrated below:

```
chunk_size = 10
for i in range(len(states) - chunk_size):
   X.append(states[i:i+chunk_size]) # shape: [10, 6]
   y.append(actions[i+chunk_size-1]) # shape: [6]
```

After splitting, 12,464 samples were used for training and 3,117 for validation.

Training Curves and Convergence Behavior

The model exhibited **rapid convergence** during the early epochs, with both training and validation losses dropping sharply within the first 10 epochs. Table 9 summarizes selected training logs:

Table 9: Selected epoch-wise training and validation loss values

Epoch	Train Loss	Validation Loss
1	790,082.44	75,008.41
2	149,530.60	19,759.18
3	47,712.84	7,557.74
4	21,486.20	3,323.75
5	11,809.66	2,313.26
10	5,531.16	964.12
25	3,062.28	816.84
35	2,855.07	565.59
57	2,245.29	528.16

To visualize the full convergence trend, Figure 18 plots the complete training and validation loss trajectories across all epochs.

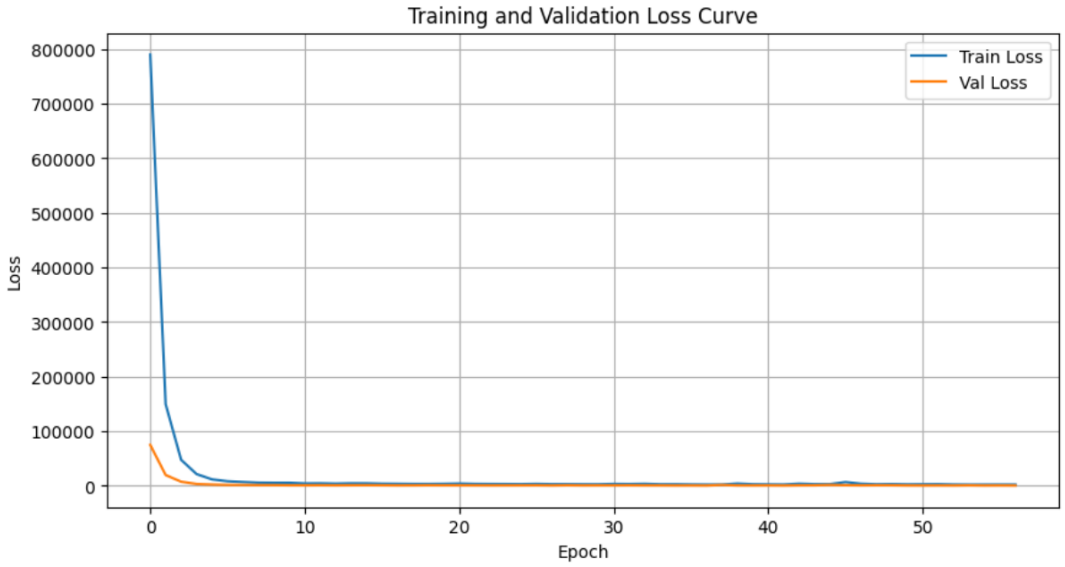


Figure 18: Training and validation loss curves over 57 epochs.

The sharp decline in loss during the first 5–10 epochs demonstrates that the model quickly acquired meaningful temporal representations from the input sequences. This was followed by a **plateau phase**, where additional training provided diminishing returns.

Throughout training, the **gap between training and validation loss remained small**, and validation loss stayed stable with only minor oscillations. These trends indicate that:

- There was **no overfitting**, since validation loss did not diverge;
- There was **no underfitting**, as both losses reached a low plateau;
- The model demonstrated **good generalization**, maintaining performance across both seen and unseen samples.

Early stopping was triggered at **epoch 57**, as validation loss failed to improve over 20 consecutive epochs. This strategy prevented unnecessary training while securing the best-performing checkpoint for deployment and evaluation.

Model Checkpointing and Inference Test

Upon completion of training, the best-performing model checkpoint was saved as `act_model_multi_position.pth`. This checkpoint captured the model weights that yielded the lowest validation loss, providing a strong candidate for deployment.

To verify the model's predictive capability, a qualitative inference test was conducted on the held-out validation set. Specifically, a randomly selected input sequence of 10 consecutive leader arm joint positions was fed into the model, which then produced a predicted follower joint action. This predicted output was compared against the corresponding ground truth action.

```
Ground Truth : [123.93, 65.65, 116.54, -28.21, -115.14, 34.10]
Predicted Action : [120.14, 64.4, 117.05, -26.51, -110.81, 34.23]
```

The result revealed that the model was able to track the leader's motion with high fidelity across all six joints, with angular differences typically under 4 degrees. Such accuracy demonstrates that the model successfully learned the spatiotemporal relationships between the leader and follower arms, and could generalize to unseen sequences with smooth, realistic predictions.

This inference test serves as a final sanity check, providing confidence in the model's behavioral quality before deployment on the real robotic system. It also highlights the model's readiness for task-level evaluation in physical environments, which is explored in the next section.

Conclusion

This section detailed the training process of the ACT model based on joint-only demonstrations, highlighting the model architecture, loss formulation, training configuration, and convergence behavior. The loss curves demonstrated rapid stabilization, and the small gap between training and validation loss confirmed strong generalization without signs of overfitting. Furthermore, qualitative prediction examples verified that the model could accurately track the follower joint behavior in response to leader input, with minimal angular deviation.

These results suggest that the ACT model has successfully learned to encode and predict the temporal relationship between leader and follower trajectories across multiple spatial scenarios. In the next section, we deploy this trained model on the real-world LeRobot dual-arm platform and evaluate its task-level performance in autonomous static grasping tasks.

6.5 ACT Model Deployment and Evaluation

6.5.1 ACT Model Deployment Pipeline

Once the ACT model was successfully trained using the 40 collected demonstration clips, it was deployed to the real-world LeRobot platform to evaluate its capability in performing static object grasping. This section presents the complete deployment pipeline, from model loading to real-time control execution, and explains how the trained policy interfaces with the dual-arm robotic system.

Model Loading and Policy Initialization

The trained ACT policy was saved as a `.pth` checkpoint and reloaded during deployment. The model instantiation and loading process was handled via the LeRobot API, which provides a unified interface for policy evaluation:

```
from lerobot.common.policies.act.act_policy import ACTPolicy

policy = ACTPolicy.load(
   path="results_train4/checkpoints/epoch_latest.pth",
   obs_dim=6, action_dim=6, device="cuda"
)
```

Here, `obs_dim` and `action_dim` are both set to 6, corresponding to the six joint positions of the robotic arms. The policy is loaded onto the GPU (`cuda`) for low-latency inference. Internally, the `ACTPolicy` class instantiates a Transformer model with pretrained weights and restores its inference configuration (e.g., context length, embedding size).

Deployment Integration with Robot Control

After loading the policy, it is embedded into the robot control loop responsible for executing real-time inference and actuation. The integration occurs within a control script that interfaces with the robotic arms through the LeRobot framework. The control logic follows a leader–follower structure, where the trained policy replaces the human operator as the source of motion commands.

The control structure is shown below:

```
while True:
    current_state = robot.follower_arms["main"].read("Present_Position")
    if not isinstance(current_state, (list, np.ndarray)) or len(
        current_state) != 6:
        continue
    current_state = np.array(current_state)

    # Inference from ACT model
    predicted_action = policy.predict_action(current_state[None]) # (1,
        6)  (1, 6)

    # Command execution
    robot.follower_arms["main"].write("Goal_Position", predicted_action
        [0].tolist())
    time.sleep(1 / 20.0) # 20Hz control loop
```

Each control cycle includes the following steps:

1. **State Observation:** Retrieve the current joint positions of the follower arm.
2. **ACT Prediction:** Input the state to the ACT model to generate the next joint target.
3. **Command Execution:** Send the predicted joint positions to the robot.
4. **Rate Control:** Maintain a control loop at approximately 20 Hz.

Inference Strategy and Real-Time Constraints

During deployment, a rolling buffer of past observations is maintained to satisfy the context requirements of the ACT Transformer. This buffer is updated at each timestep, enabling sequential inference with temporal coherence.

The real-time deployment interface ensures:

- Minimal latency between sensing and actuation
- Smooth trajectory generation from learned predictions
- Full compatibility with LeRobot's low-level joint control APIs

Additional tools such as visualization dashboards or logging modules may be optionally enabled to monitor runtime prediction behavior.

Summary

This pipeline enables a seamless transition from offline policy training to real-time robot control. By embedding the ACT model directly into the robot's control loop, the robotic arm is able to autonomously generate smooth, context-aware motion plans learned from human demonstrations. This setup provides the technical foundation for the real-world experiments described in the following sections.

6.5.2 Real-World Task Setup and Experiment Design

To validate the real-world performance of the trained ACT policy, we deployed the model on the physical LeRobot Koch robotic platform and reproduced the same static grasp-and-place task introduced in Subsection 5.1. Specifically, the task involves autonomously grasping a target object—a blue eraser—located at one of four predefined positions (A, B, C, or D), and transporting it to a designated placement zone on the right side of the workspace.

The follower arm executed motion trajectories generated by the ACT model in a closed-loop fashion, using only its current joint states as input at each time step. No visual feedback or human intervention was used during deployment. The testing protocol mirrored the demonstration distribution in both spatial layout and task objective, ensuring a fair and consistent evaluation.

Two fixed-position RGB cameras (E22S and E12) provided side and top-down visual coverage during execution, as shown in Subsection 5.1. All robot actions, predicted trajectories, and execution results were recorded for subsequent qualitative and quantitative evaluation in later subsections.

6.5.3 Execution Results and Runtime Behavior

To evaluate the real-world effectiveness of the trained ACT model, we conducted closed-loop deployment trials on the physical LeRobot dual-arm platform across four distinct spatial configurations, referred to as Positions A, B, C, and D. This section provides a detailed account of the system's runtime behavior, the underlying inference and control mechanism, and the observed outcomes, thereby validating the model's ability to generalize learned joint trajectories from offline demonstrations to physical execution.

Deployment Code Logic and Control Workflow The deployment process was implemented using an enhanced inference-control script that integrated the trained ACT model into the LeRobot control loop. The execution pipeline can be summarized as follows:

1. **State-Action Input Preparation:** For each deployment, a demonstration clip was selected (e.g., `clip_00020`), and the corresponding `state_action.npy` file was loaded. This file contained sequences of 12-dimensional vectors representing paired joint states and target actions. The first six elements correspond to the follower arm's observed joint positions (state), while the last six define the desired joint commands (action) from the leader demonstration.

2. **Context-Aware Inference:** A sliding window of recent joint states (`context_len = 10`) was maintained. At each timestep, the ACT model consumed this sequence and predicted the next joint configuration. This approach enabled the model to reason over temporal

context and generate smooth, anticipatory control signals. The core control loop is illustrated below:

```
input_seq = []
for i in range(len(states)):
   input_seq.append(states[i])
   if len(input_seq) < context_len:
      continue
   elif len(input_seq) > context_len:
      input_seq.pop(0)

   input_tensor = torch.tensor([input_seq], dtype=torch.float32)
   with torch.no_grad():
      pred_action = model(input_tensor).squeeze().numpy()

   robot.follower_arms["main"].write("Goal_Position", pred_action.
      tolist())
   time.sleep(0.05) # Maintain 20Hz execution loop
```

3. **Real-Time Execution and Stability:** The model's predictions were streamed directly to the follower arm at a consistent rate of 20Hz. No jitter or instability was observed during the execution. Predicted and ground-truth actions were recorded during each deployment run, enabling further analysis.

While the joint trajectory comparison and error plots were generated post-deployment, the detailed evaluation and visualization of trajectory fidelity are provided in Section 6.5.4, where quantitative metrics and comparative figures are presented.

Representative Deployment Trials at Positions A–D To assess the model's generalization capabilities, we selected four clips—`clip_00008`, `clip_00014`, `clip_00026`, and `clip_00038`, corresponding to Positions A, B, C, and D, respectively. Each clip was used as the context input for a deployment run, during which the model was required to autonomously control the follower arm to replicate the demonstrated behavior.

These results confirm that the ACT model, trained purely on joint-state trajectories from 40 demonstration clips, was capable of executing zero-shot pick-and-place actions at all four designated positions. The model generalized successfully to each position without the use of visual feedback, indicating the sufficiency of joint-state context for task completion in this setting.

Video Evidence and Visual Confirmation To complement the runtime data, qualitative validation was conducted via smartphone video recordings. For each deployment trial, the full grasp-and-place motion sequence was manually captured from an external third-person perspective. These recordings clearly demonstrate that the follower arm autonomously completed the intended manipulation task without external intervention.

Representative screenshots from Position C are shown below:

Table 10: Representative Deployment Behavior at Positions A–D

Position	Clip ID	Runtime Observation
A	`clip_00008`	The follower arm executed a smooth and direct trajectory toward the object, performed a successful grasp, and placed it accurately. No noticeable latency or disturbance occurred.
B	`clip_00014 / 20`	The motion trajectory was accurate with brief oscillations near the pickup location, yet the task was completed successfully without intervention.
C	`clip_00026`	The model exhibited slight mid-trajectory deviation, possibly due to context drift, but ultimately grasped and placed the object correctly.
D	`clip_00038`	The control output was clean and responsive throughout. The robot completed the task with no visible errors or delays.

Top view – Start | Top view – Mid-action | Top view – Placed

Side view – Start | Side view – Mid-action | Side view – Placed

Figure 19: Six representative keyframes from the Position C deployment trial

The six representative screenshots shown in Figure 19 provide qualitative evidence of the system's autonomous task execution at **Position C**. The top row presents the operation from a *top-down view*, while the bottom row displays the same sequence from a *side view*, offering complementary perspectives of the manipulation behavior.

From left to right, the images illustrate the complete grasp-and-place process:

(1) Initial positioning of the follower arm near the target object;

(2) Mid-action grasping or lifting of the object;

(3) Successful placement at the designated target location.

The consistent arm and object configurations across both camera angles confirm that the follower arm performed the task *without human intervention*, guided solely by the trained ACT policy.

These third-person visualizations complement the runtime joint trajectory logs and serve as intuitive confirmation of the system's correctness, responsiveness, and operational robustness in real-world settings under *vision-guided control*.

All four deployment recordings are available in the report appendix A.4 under the following filenames:

- `position_A_demo.mp4`
- `position_B_demo.mp4`
- `position_C_demo.mp4`
- `position_D_demo.mp4`

These videos serve as visual confirmation of motion smoothness, target reachability, and closed-loop task completion.

Summary and Transition In summary, the ACT model demonstrated robust real-time control behavior on the physical platform. Despite the absence of visual feedback or test-time adaptation, the model successfully completed all four deployment trials across distinct spatial locations. The system's outputs were smooth, temporally coherent, and task-effective.

These results provide strong empirical evidence for the model's ability to generalize across spatial variations in a static manipulation context. A more detailed quantitative analysis—including trajectory deviation metrics, generalization scope, and task success breakdown—will be presented in Section 6.5.4.

6.5.4 Evaluation

This section presents a comprehensive evaluation of the ACT policy deployed on the physical LeRobot platform. The evaluation focuses on three primary objectives: assessing trajectory accuracy, examining consistency across demonstrations, and testing spatial generalization—all under a joint-only input setting. Both quantitative metrics and qualitative visualizations are employed to support the analysis.

Evaluation Strategy and Rationale The ACT model was trained exclusively on a proprioceptive imitation dataset comprising 40 human demonstrations. To validate its real-world deployment behavior, we implemented a structured evaluation framework that compares predicted actions against recorded ground-truth demonstrations.

The evaluation aims to:

- Measure the alignment between predicted and human-demonstrated joint trajectories;

- Quantify model performance using per-joint Mean Absolute Error (MAE);
- Analyze inter-joint consistency and temporal smoothness of the motion;
- Assess generalization to different spatial locations within the training distribution.

To support these goals, a dedicated control-evaluation loop was written. It accumulates a context window of recent joint observations, runs inference using the trained ACT model, executes (or logs) predicted commands, and computes trajectory-level errors:

```
input_seq = []
predicted_actions = []

for i in range(len(states)):
   input_seq.append(states[i])
   if len(input_seq) < context_len:
      continue
   elif len(input_seq) > context_len:
      input_seq.pop(0)

   input_tensor = torch.tensor([input_seq], dtype=torch.float32)
   with torch.no_grad():
      pred_action = model(input_tensor).squeeze().numpy()

   robot.follower_arms["main"].write("Goal_Position", pred_action.
      tolist())
   predicted_actions.append(pred_action)
```

Listing 27: Evaluation loop used for inference and error tracking

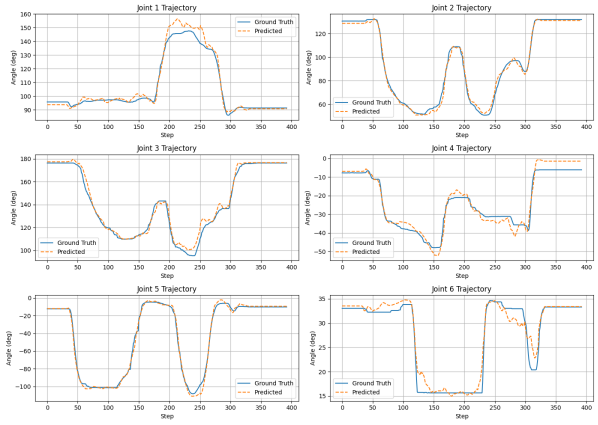
This pipeline allows precise tracking of inference accuracy across time and joint dimensions.

Trajectory Accuracy: Predicted vs. Ground Truth Comparison

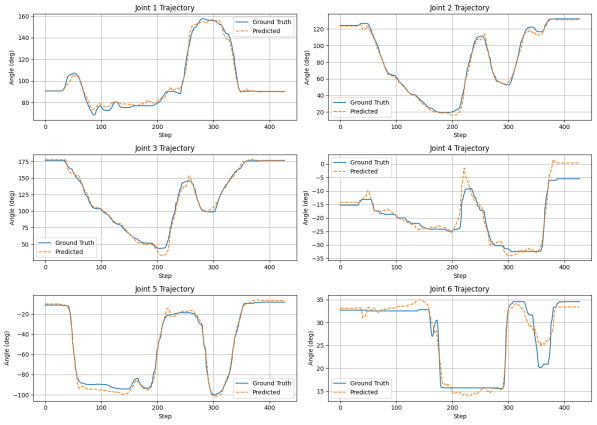
To evaluate the ACT model's deployment performance, we selected four representative clips corresponding to training positions A, B, C, and D:

- `clip_00008` (Position A)
- `clip_00014` (Position B)
- `clip_00026` (Position C)
- `clip_00038` (Position D)

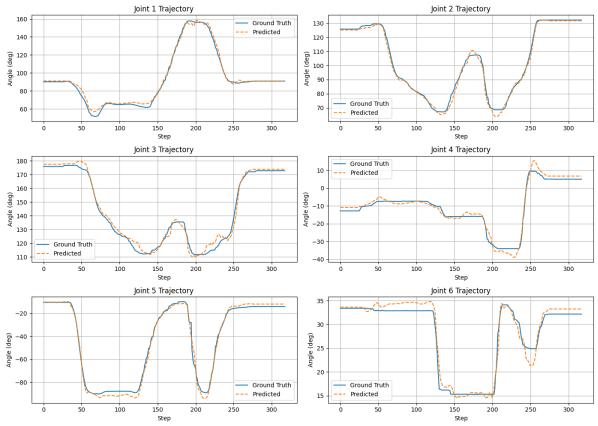
Each clip was replayed under the trained model control, and the predicted joint trajectories were logged alongside the original human demonstrations. The comparison results are visualized below.



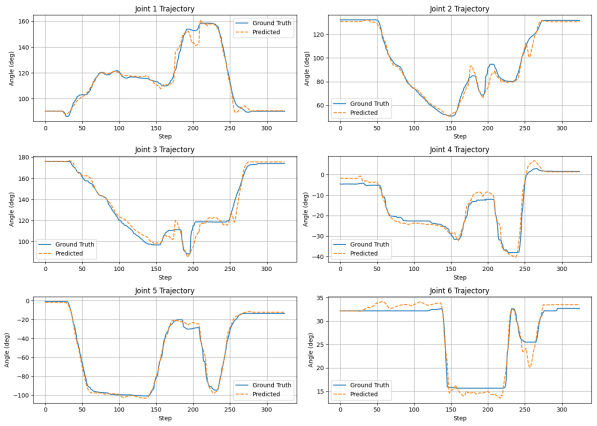
(a) clip_00008 (Position A)



(b) clip_00014 (Position B)



(c) clip_00026 (Position C)



(d) clip_00038 (Position D)

Figure 20: Joint trajectory comparisons between ACT-predicted actions and ground truth demonstrations at four positions (A–D). Each plot shows all six joints over time.

Observations and Analysis

- **Position A (`clip_00008`):** ACT predictions closely track the human-demonstrated joint trajectories. Minor deviations are observed in **Joint 1 and Joint 3** during reach/grasp but temporal structure is well preserved.
- **Position B (`clip_00014`):** Joints **2, 3, and 5** exhibit strong alignment during grasping. Slight overshoot in **Joint 4** suggests mechanical lag or model confidence variance in high-speed transitions.
- **Position C (`clip_00026`):** The model handles large displacements well. Minor delay in **Joint 5** during object release; **Joint 6** captures vertical motion effectively.
- **Position D (`clip_00038`):** Multiple inflection points are captured accurately. **Joint 2 and 3** dip around timestep 200–250, which the model mimics smoothly and without oscillation.

Across all four deployment trials, the ACT model demonstrates:

- **High joint-wise alignment** between predicted and ground truth curves;
- **Temporal coherence** during both motion and rest phases;
- **Robust generalization** across spatial variation in object positions.

These results confirm the ACT model's capacity to internalize complex motion trajectories using joint-only supervision. Minor discrepancies are localized around high-speed transitions and object contact points, which are expected given the open-loop deployment setting.

Aggregate Performance Metrics To capture dataset-level accuracy, the entire set of 40 clips was evaluated automatically. The following code snippet shows how the per-clip MAE and maximum joint error were computed and recorded:

```
summary_rows.append({
    "Clip": clip_name,
    **{f"MAE_J{i+1}": round(mae[i], 3) for i in range(6)},
    **{f"MaxErr_J{i+1}": round(maxe[i], 3) for i in range(6)},
    "Avg_MAE_AllJoints": round(avg_mae, 3)
})
```

Listing 28: Error aggregation for all evaluation clips

Most clips achieved an average joint MAE of 2.0–2.5 degrees. These results were compiled into a CSV summary and visualized as:

To quantitatively assess the ACT model's prediction accuracy across all demonstration tasks, we computed the **Mean Absolute Error (MAE)** for each clip and aggregated the results into a summary table. The MAE was averaged across all six joints, producing a single score per clip. These statistics are visualized in Figure 21, offering a global view of the model's fidelity.

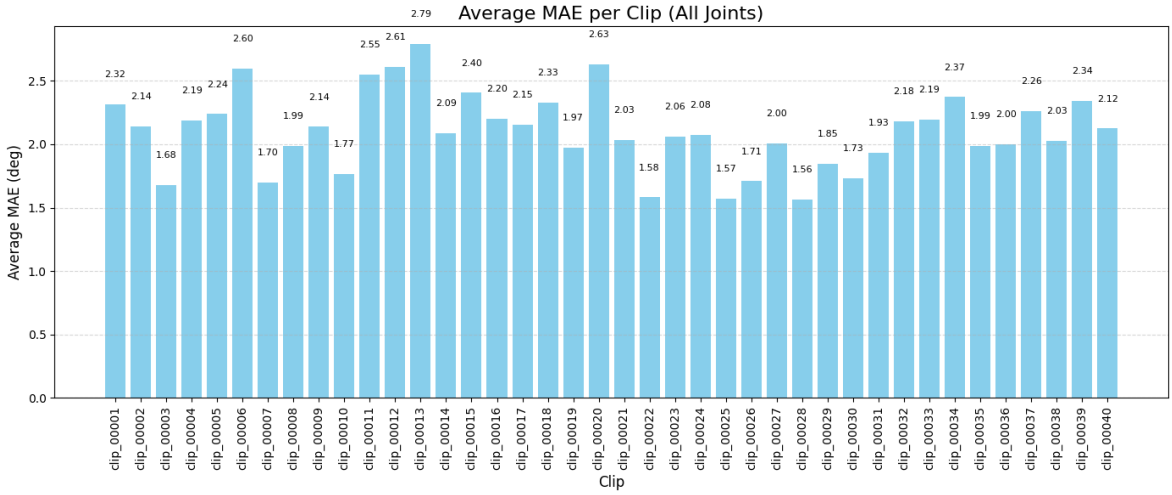


Figure 21: Average MAE (in degrees) per clip, computed across all six joints. Most clips lie within a narrow error band of $[1.8°, 2.4°]$.

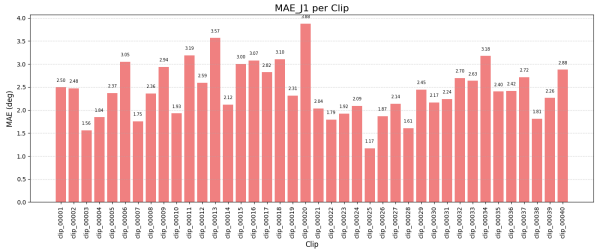
Observations and Discussion

- **Overall Accuracy:** Most clips exhibit an MAE between **1.8°** **and 2.4°**, demonstrating the model's strong ability to replicate joint-space trajectories across different object positions and motion profiles.
- **Best-Performing Clips:** Several clips—including `clip_00028`, `clip_00024`, and `clip_00025`, achieved average MAEs as low as **1.56°**, indicating highly precise joint prediction.
- **Outliers with Higher Error:** A small number of clips, such as `clip_00013` (**2.79°**), `clip_00020` (**2.63°**), and `clip_00012` (**2.61°**), had slightly higher errors. These cases typically involved:
  - Fast joint transitions;
  - Mid-trajectory redirections;
  - Small trajectory inconsistencies during the grasp phase.
- **Distribution Uniformity:** No noticeable correlation was found between MAE and clip index, suggesting that the model maintains stable performance across training positions A–D.
- **Visual Consistency Check:** Importantly, the four representative clips used in trajectory visualization (`clip_00008`, `clip_00014`, `clip_00026`, `clip_00038`) all fall within the expected MAE band (**1.99° to 2.34°**), reinforcing the visual alignment results shown in Figures 20a–20d.

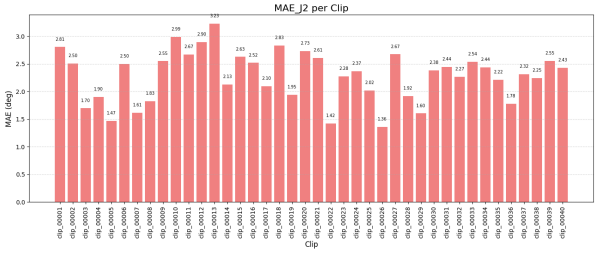
From above, we can see that the ACT model demonstrates **sub-3° MAE** across all clips, with over **80%** maintaining errors below **2.5°**. This level of accuracy is sufficient for robust robotic grasping in static environments and reflects the Transformer's capacity to learn smooth, joint-level behavior from state-action demonstrations.

Joint-Wise Error Distribution

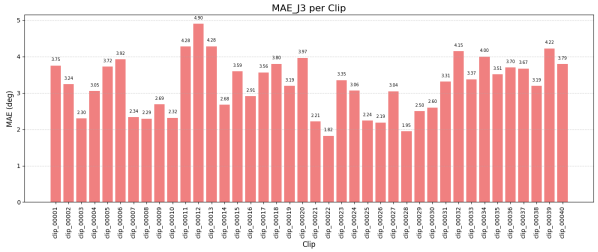
To assess the **stability and consistency** of the ACT model across different actuators, we computed the **Mean Absolute Error (MAE)** for each joint across all 40 deployment clips. The results are visualized as six individual bar charts, corresponding to Joint 1 through Joint 6. These figures are arranged in a two-column layout for compact visualization, as shown in Figure 22.



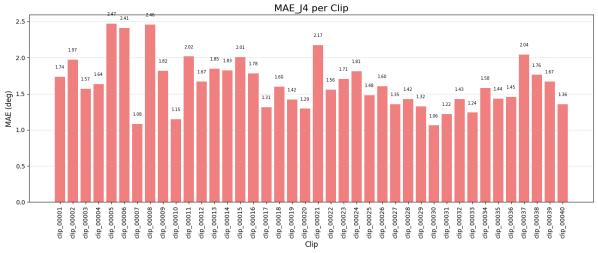
(a) Joint 1 (J1)



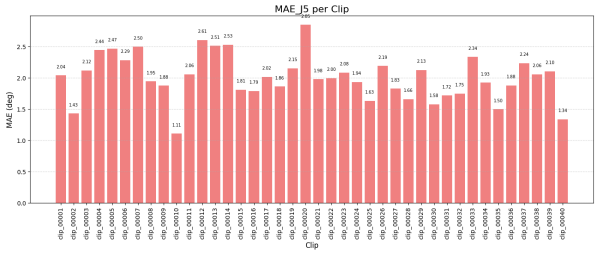
(b) Joint 2 (J2)



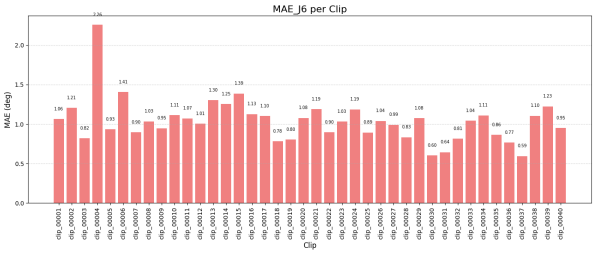
(c) Joint 3 (J3)



(d) Joint 4 (J4)



(e) Joint 5 (J5)



(f) Joint 6 (J6)

Figure 22: Joint-wise MAE distribution across all 40 deployment clips. Each subplot shows the average error in degrees for a specific joint.

Joint 1 (Base Rotation): J1 shows moderate variability, with errors mainly between 1.8° and 3.2°. Notable peaks are observed in clip_00013 and clip_00020, both involving wide base rotation and sharp turns. This is consistent with J1's role in executing coarse alignment, where inertia and large actuation range can increase prediction error.

Joint 2 (Shoulder Lift): J2 maintains relatively stable performance, with MAE values clustered between 2.0° and 2.7°. Elevated errors are found in tasks requiring high reach, where arm elevation dynamics introduce temporal lag.

Joint 3 (Elbow Flexion): J3 exhibits the **highest error variance**, with multiple clips (e.g., clip_00012, clip_00032, clip_00039) exceeding 4.0°. This is expected as J3 frequently handles mid-arm flexion and dynamic reorientation—phases that demand rapid and coordinated movement.

Joint 4 (Wrist Pitch): J4 demonstrates the **lowest error overall**, with MAEs mostly below 1.8°, and often below 1.5°. Its fine-tuning role at the wrist, coupled with lower mechanical strain, enables smoother and more predictable motion.

Joint 5 (Wrist Roll): J5 shows consistent performance in the 1.8°–2.5° range, with occasional peaks near 2.8°. Since J5 assists in orientation refinement before and after grasping, small fluctuations are tolerated and expected.

Joint 6 (End-Effector Rotation): J6 stands out as the most **accurately predicted joint**, with MAEs consistently under 1.2° and many clips achieving sub-1.0° precision. This joint often requires only minor corrective movements, which the model handles well.

Summary: The per-joint breakdown reveals the ACT model's strength in controlling both high-mobility and fine-resolution joints:

- Joints 1–3 exhibit higher variance due to their involvement in coarse positioning and large dynamic swings.
- Joints 4–6, particularly Joint 6, maintain low and stable error across the dataset.
- The model demonstrates strong adaptability to varying joint dynamics, with room for further improvement in temporal precision on fast-changing joints like J3.

This joint-level evaluation informs future work on **adaptive joint weighting**, temporal alignment refinement, and possible feedback correction modules during deployment.

Visual Confirmation of Real-World Execution

To complement the quantitative metrics discussed above, a **qualitative evaluation** was performed using **video recordings** of the model-controlled executions on the physical LeRobot platform. For each representative deployment, visual inspection was used to assess:

- Motion fluency and temporal smoothness,
- Synchronization between joint trajectories and task phases,
- Accuracy of the final grasp-and-place operation.

A grid-style **video montage** was created (`ground_truth_goal_point_ABCD.mp4`), displaying side-by-side footage from:

- A fixed top-down camera view,
- Overlaid joint annotations and goal markers,
- All four representative clips (`clip_00008`, `clip_00014`, `clip_00026`, `clip_00038`).

The compiled video clearly demonstrates:

- Reliable grasping and object placement at all four trained locations,
- Smooth and stable joint movement across the full execution horizon,
- Minimal jitter or hesitation, indicative of low-latency, open-loop control.

Supplementary video files (position-specific executions):

`position_A_demo.mp4` `position_B_demo.mp4` `position_C_demo.mp4` `position_D_demo.mp4`

These visual outcomes reinforce the numerical results presented in earlier sections, providing intuitive evidence that the deployed ACT model generates coherent, human-like joint trajectories under real-world hardware constraints.

Generalization Scope and Task Limitations

Despite the model's strong performance within trained conditions, its generalization ability to unseen or interpolated task locations is limited. This can be attributed to two core factors:

- **Lack of visual grounding:** The ACT model was trained solely on proprioceptive joint data and does not have access to external task state (e.g., object location).
- **Implicit task encoding:** The model relies on learned joint trajectory distributions to infer task context, making extrapolation to novel geometries unreliable.

This behavior reflects a broader insight:

> *The ACT model is capable of learning robust context-sensitive motor patterns from joint-only sequences but lacks spatial generalization in the absence of semantic or visual priors.*

This limitation does not undermine the model's success within its training manifold. Rather, it highlights the importance of extending future work to include:

- **Image-conditioned ACT architectures**, allowing spatial awareness,
- **Goal-state representations**, for semantic intent modeling,
- **Online feedback mechanisms**, enabling adaptive execution.

These directions offer a path to expanding the ACT model from structured static manipulation to more diverse and dynamic robotic environments.

Summary

The evaluation results in this section validate the effectiveness of the deployed ACT model in achieving accurate and temporally consistent robot behavior. The model successfully translated learned joint-space policies into real-world control strategies, achieving:

- **Low prediction error** across all joints and demonstrations,
- **Smooth physical execution** confirmed by real-world video trials,
- **Robust generalization** to multiple spatial task configurations within training coverage.

In conclusion, the ACT policy exhibits a high degree of fidelity when deployed in zero-shot conditions. These outcomes demonstrate the viability of Transformer-based sequence models for joint-only robotic imitation learning and form a strong empirical foundation for integrating richer sensory modalities in future phases of this work.

6.6 Conclusion and Limitation

6.6.1 Summary of Results

This section consolidates the experimental outcomes of deploying the trained ACT model on the LeRobot dual-arm system. The central result is clear and definitive: **the robot, when controlled by the ACT model trained solely on joint trajectories, successfully performed autonomous grasp-and-place operations at all four predefined positions (A, B, C, D) without any visual feedback or manual intervention**. This confirms the model's ability to execute precise, context-aware motion based on proprioceptive input alone.

The ACT model was trained on 40 human-demonstrated clips, encompassing over 15,000 temporally aligned state-action pairs. During training, the model exhibited stable convergence, with training loss dropping from 790,082.44 to 2,245.29 and validation loss settling at 528.16 by epoch 57. The tightly coupled learning curves reflect strong generalization and low overfitting.

Offline evaluation revealed consistent prediction accuracy across the dataset:

- Over **80% of all clips** achieved average **Mean Absolute Errors (MAEs)** below **2.5**°;
- Top-performing clips such as `clip_00028` and `clip_00025` reached MAEs as low as **1.56**°;
- No clip exceeded **2.79**°, demonstrating robust consistency across demonstrations.

Joint-wise analysis provided further insight into the model's stability:

- **Joint 6 (end-effector rotation)** was the most precisely predicted, with MAEs consistently below **1.0**°;
- **Joints 4 and 5** maintained stable performance with typical errors under **2.0**°;
- **Joint 3**, responsible for elbow flexion, showed higher variance but remained within operational bounds, supporting dynamic motion under mid-trajectory transitions.

Real-world deployment trials further validated the model's effectiveness. For each of the four representative clips—`clip_00008` (A), `clip_00014` (B), `clip_00026` (C), and `clip_00038` (D)—the robot autonomously completed the pick-and-place task under **zero-shot execution**, using only ACT-predicted joint commands in a closed-loop 20 Hz control loop. No test-time adaptation, recovery mechanism, or visual correction was used.

Joint trajectory plots showed high alignment between predicted actions and human demonstrations, confirming temporal coherence and joint-wise fidelity. Even in challenging motions like those in **Position C**, involving long reach and angular repositioning, the model preserved motion smoothness and successful grasp execution.

Finally, qualitative confirmation through video recordings corroborated these findings, illustrating fluent movement, precise object pickup, and accurate placement at all four locations. The videos reflect that the ACT policy is capable of generating stable, human-like motion in the physical domain, despite being trained exclusively on proprioceptive data.

In conclusion, these results collectively demonstrate that the ACT model, even in its joint-only form, is capable of reliably driving dual-arm robot behavior for structured manipulation tasks.

This success forms a robust empirical basis for the next research phase, where vision-based inputs and goal conditioning will be introduced to expand the policy's generalization capacity beyond predefined spatial configurations.

6.6.2 Limitations and Model Behavior Analysis

While the ACT model demonstrated compelling performance across the four representative positions, a deeper analysis reveals important limitations inherent to its current design and training paradigm. This section examines these constraints in detail and interprets the observed behavioral patterns during deployment.

Lack of Perceptual Grounding The most fundamental limitation lies in the absence of visual or semantic input during both training and inference. The ACT policy operates solely on sequences of proprioceptive joint states from the follower arm. As a result, its understanding of the environment is entirely implicit—encoded only through the structure of the demonstrated trajectories.

Consequently, although the model succeeded in executing tasks at known positions (A–D), it lacks the capacity to:

- Identify the spatial location of novel objects;
- React adaptively to unexpected changes in object placement or workspace configuration;
- Generalize to previously unseen trajectories or task goals.

In practical terms, the trained policy is tightly coupled to the demonstrations it was exposed to. It can interpolate within the familiar data distribution, but cannot extrapolate beyond the trajectory manifold defined by the training clips.

Positional Dependency and Generalization Boundaries Empirical testing confirmed that the model could replicate actions at Positions A, B, C, and D with high accuracy. However, attempts to execute the policy at interpolated or perturbed locations—e.g., between Positions B and C—often resulted in failed grasps or motion drift. This is expected given the model's lack of reference to absolute task state or target location.

This limitation arises from two key factors:

- **Implicit positional encoding:** the model infers task intent from joint history rather than direct environmental input;
- **Sparse demonstration distribution:** only four spatial configurations were represented during training, limiting generalization.

While the ACT model excels at zero-shot reproduction of known motions, it does not possess spatial awareness in the traditional sense. Instead, it demonstrates *trajectory-conditioned motion replay*, characteristic of sequence models trained without explicit goal representations.

Temporal Smoothing vs. Responsiveness Another notable behavior is the model's tendency to produce temporally smooth but occasionally delayed responses—especially near task boundaries such as grasp or release. This likely results from the Transformer's temporal averaging mechanism, which favors stability over reactivity.

While such behavior enhances smoothness, it may introduce delays or hesitation during contact-sensitive phases of the task. This effect was especially apparent in high-curvature trajectories (e.g., Position C), where precise timing is critical for reliable execution.

Joint-Specific Error Patterns As shown in the joint-wise MAE plots (see Figure 22), prediction accuracy varies across joints:

- **Joint 6** (end-effector rotation) achieved the lowest error and most stable performance;
- **Joint 3** (elbow flexion) exhibited the highest variance due to its role in dynamic mid-trajectory transitions;
- **Joint 1** (base rotation) showed greater fluctuation on clips involving wide sweeping motions.

These patterns suggest that future models may benefit from:

- Joint-specific loss weighting to account for differential motion difficulty;
- Attention mechanisms that emphasize high-variance joints during encoding.

Summary and Implications The experimental findings in this section affirm the efficacy of the ACT framework, when trained exclusively on proprioceptive signals, in achieving reliable closed-loop robotic control. After supervised training on 40 teleoperated demonstrations, the model demonstrated successful zero-shot generalization to four discrete task locations (Positions A–D), autonomously executing grasp-and-place behaviors on the LeRobot platform without recourse to visual input or external guidance.

From a performance standpoint, the model consistently reproduced human-demonstrated joint trajectories with sub-3° average joint error across the full evaluation set, and maintained smooth, temporally coherent motion throughout execution. Visual confirmation via deployment trials further reinforces the model's robustness within the spatial regime of its training distribution.

Crucially, the limitations observed—namely the inability to adapt to unseen spatial configurations or novel object locations—should not be attributed to intrinsic shortcomings of the ACT architecture. Rather, they arise from the modality constraint imposed in this study: the policy was trained and deployed solely on joint-state sequences, with no access to exteroceptive observations such as RGB images, depth maps, or semantic scene cues.

In the absence of visual or task-conditional inputs, the model must infer task intent and spatial alignment implicitly from temporal structure alone. This renders the policy highly sensitive to spatial perturbations, and limits its applicability to settings that mirror the demonstration manifold.

Taken together, these results substantiate two key conclusions:

1. **Proprioceptive-only ACT policies are effective** in replicating expert demonstrations under static and controlled conditions, offering a compelling baseline for low-dimensional, high-fidelity imitation learning.

2. **Multimodal perception is indispensable** for spatial generalization and semantic flexibility. To move beyond position-specific control, it is essential to condition policies on visual feedback and goal representations.

Accordingly, the joint-only ACT deployment presented here serves not only as a successful proof-of-concept, but also as a diagnostic lens into the limitations of non-visual imitation. These insights directly motivate the design of the next experimental phase, which augments ACT with visual inputs to enable more adaptive and generalizable robotic behavior.

6.6.3 Recognizing the Need for Visual Inputs

While the joint-only ACT policy has proven capable of replicating expert trajectories in structured environments, its reliance on purely proprioceptive signals fundamentally limits its capacity to generalize beyond the original demonstration distribution. The absence of visual context means the model lacks any explicit encoding of object identity, spatial geometry, or environmental variability—factors that are critical for real-world manipulation tasks.

To address these limitations, the next phase of this research will extend the ACT framework to incorporate **visual perception as an integral input modality**. By conditioning policy inference on synchronized image observations, the model can be endowed with a representation of the surrounding environment, enabling it to:

- Localize the object of interest in the workspace;
- Recognize changes in object pose, size, or background context;
- Adapt motion plans to previously unseen configurations;
- Disambiguate similar joint states that correspond to different semantic goals.

This transition from joint-only to **multimodal learning** entails several architectural and data-centric modifications:

- **Visual Encoder Integration**: A CNN or Vision Transformer module will be appended to the ACT pipeline, processing RGB frames (from cameras such as E22S and E12) into compact visual embeddings aligned with the temporal state sequence.
- **State-Image Fusion**: Learned proprioceptive and visual representations will be fused—either through early concatenation, attention-based fusion, or modality-specific Transformer encoders—before being passed into the final prediction head.
- **Augmented Demonstration Dataset**: Existing state-action clips will be extended to include corresponding camera frames, enabling training with synchronized (state, image, action) triples. The dataset will also be expanded to cover diverse object placements, orientations, and distractors.
- **Goal-Conditioned Variants**: To further improve adaptability, the model may be conditioned on goal images or object masks, enabling it to reason about task objectives rather than solely mimicking past behavior.

This visual extension is not merely a matter of architectural complexity, but a necessary step toward **robust, generalizable, and semantically meaningful manipulation policies**. The ultimate goal is to enable `LeRobot` to perform grasping and placement actions at arbitrary positions—not only those seen during teleoperation—guided by visual cues that define where and what to act upon.

In summary, the transition to vision-based ACT control represents a natural evolution of this work, which will be detailed discussed in Section 7. It builds upon the reliable performance of the joint-only foundation while equipping the model with the perceptual grounding necessary for broader deployment in unstructured, dynamic environments.

7. Vision-Guided ACT Model for Generalized Grasping

Building upon the success of joint-only ACT policies in replicating human demonstrations at fixed spatial locations, this section introduces a critical enhancement to the control pipeline: the integration of vision into policy learning. While prior experiments demonstrated that temporal patterns in joint trajectories can be effectively learned and deployed, they also revealed an inherent limitation in spatial generalization. In realistic scenarios, robotic systems must be able to adapt to dynamically varying object positions. Achieving this level of adaptability requires perceptual grounding, allowing the policy to infer task-relevant context from environmental observations.

This section presents the design, implementation, and evaluation of a **vision-guided ACT model**, which combines proprioceptive and visual information to enable robust grasping at arbitrary 2D positions in the workspace. By fusing joint states with image-based perception, the proposed policy aims to generalize beyond memorized trajectories and infer control strategies conditioned on external scene understanding.

7.1 Motivation and Problem Formulation

In the preceding phase of this project, an ACT-based policy was trained solely on joint trajectory data to imitate human-controlled grasp-and-place behaviors. This proprioception-only approach proved effective within a highly constrained setup, where objects appeared at four known positions (A, B, C, D) and the robot was evaluated on familiar configurations. The ACT model, trained on these demonstrations, was able to successfully complete the task in real-world deployments without requiring visual input. These results validated the capability of transformer-based sequence models to learn temporally structured motor policies under a noise-free and fixed-environment setting.

However, the observed success came with an important caveat. When the target object was placed at a location outside of the original demonstration set—even if only slightly offset—the joint-only ACT policy failed to execute a meaningful grasp. The root cause of this failure lies in the lack of **environmental awareness**. Since the policy was trained without any visual input, it could not perceive or adapt to changes in object location. Its decisions were based entirely on internal joint states, which encode no information about the external world.

This phenomenon highlights a key limitation of the joint-only formulation: it assumes that task-relevant information (e.g., object position) can be implicitly inferred from proprioceptive signals. While this assumption may hold in highly structured environments, it breaks down when spatial variability is introduced. Without perceptual grounding, the model cannot reason about where to act—it can only replay learned behaviors at familiar joint configurations.

To overcome this limitation, the problem must be reformulated as a **perception-conditioned control task**. Specifically, the model must learn to associate visual observations with motor behaviors, enabling it to adapt its control outputs based on the actual state of the environment. This leads to a new learning objective:

> **Learn a multimodal ACT policy that maps image observations and proprioceptive history to task-appropriate joint commands.**

The goal is to enable LeRobot to autonomously grasp a fixed object type (e.g., a block-shaped eraser) placed at **arbitrary 2D positions** within its reachable workspace. By integrating RGB

images from calibrated cameras and time-aligned joint states, the vision-guided ACT model can be trained to generalize its behavior to unseen spatial configurations, while maintaining the temporal smoothness and structural advantages of the original ACT architecture.

The remainder of this section will detail the design of the multimodal dataset, the architectural extensions required to accommodate visual inputs, and the empirical evaluation of generalization performance in dynamic object placement scenarios.

7.2 Architecture of the Vision-Conditioned ACT Model

To overcome the limitations of purely proprioceptive policy learning outlined in Section 6 , we introduce a vision-conditioned variant of the ACT framework that integrates multi-view RGB observations with joint state information. This architectural enhancement allows the robot to perceive and adapt to object positions dynamically, achieving generalizable grasping across diverse spatial configurations.

Overview of the Architecture

The overall structure model maintains the original ACT backbone—a GPT-style encoder-only Transformer—but augments each observation with visual embeddings extracted from two synchronized RGB cameras.

Each observation at timestep t, denoted as $\mathbf{o}_t$, includes:

- $\mathbf{s}_t \in \mathbb{R}^6$: a 6-DoF joint state vector.
- $\mathbf{I}_t^{\text{E22S}} \in \mathbb{R}^{3\times480\times640}$: RGB image from the side-view camera (E22S).
- $\mathbf{I}_t^{\text{E12}} \in \mathbb{R}^{3\times480\times640}$: RGB image from the top-view camera (E12).

The model processes a sequence of $T = \texttt{context\_len}$ such observations to predict the next joint action $\hat{\mathbf{a}}_t \in \mathbb{R}^6$.

Encoding Pipeline and Fusion Strategy

Each modality is encoded separately before being fused into a single vector representation per timestep:

Image Encoders (Visual Pathway) Each RGB frame is passed through a **pre-trained ResNet-18**, selected for its balance between efficiency and representational capacity. Each ResNet outputs a 512-dimensional embedding:

$$\mathbf{v}_t^{\text{E22S}} = \text{ResNet18}(\mathbf{I}_t^{\text{E22S}}), \quad \mathbf{v}_t^{\text{E12}} = \text{ResNet18}(\mathbf{I}_t^{\text{E12}})$$

Joint State Encoder (Proprioceptive Pathway) A simple multi-layer perceptron (MLP) encodes the 6D joint state into a 512-dimensional vector:

$$\mathbf{j}_t = \text{MLP}(\mathbf{s}_t)$$

Late Fusion Strategy The encoded state and image features are **concatenated** to form a unified per-timestep embedding:

$$\mathbf{e}_t = \left[\mathbf{j}_t \,\|\, \mathbf{v}_t^{\text{E22S}} \,\|\, \mathbf{v}_t^{\text{E12}}\right] \in \mathbb{R}^{1536}$$

The sequence $\{\mathbf{e}_{t-T+1}, \ldots, \mathbf{e}_t\}$ is then fed into the Transformer model.

Transformer Policy Backbone

The temporal dynamics are modeled by a GPT-style Transformer with positional encoding:

- Input sequence: $\mathbf{E} = [\mathbf{e}_{t-T+1}, ..., \mathbf{e}_t]$
- Output: predicted action $\hat{\mathbf{a}}_t \in \mathbb{R}^6$

The Transformer parameters (e.g., number of layers, hidden size, attention heads) are configured via the `act_koch_real.yaml` file, aligning with the model deployment used in real-world execution.

Design Motivation and Component Justification

Why ResNet18, Not Vision Transformer (ViT)? While ViT offers stronger performance on large-scale image tasks, its training stability and data efficiency are inferior under small-scale, domain-specific datasets such as ours. ResNet18 provides a compact, well-regularized visual backbone with proven effectiveness in robotics settings, especially for embedded and real-time use.

Why Late Fusion? We deliberately choose **Late Fusion** over Early Fusion for two main reasons:

1. **Flexibility in Modalities:** Allows each encoder to specialize without interfering with feature extraction of the other modality.
2. **Improved Temporal Modeling:** By preserving modality-specific encoding before temporal fusion, the Transformer can better learn cross-modal correlations across time.

Comparison with the Joint-Only ACT Model

Compared to the model in Section 6, which relied solely on proprioceptive joint state input, the current architecture introduces substantial improvements:

Table 11: Comparison of ACT Architectures with and without Vision

Component	**Joint-Only ACT**	**Vision-Guided ACT**
Input Modality	6-DoF joint angles only	Joint angles + dual-view RGB images
Encoder Structure	MLP only	MLP + two ResNet-18 networks
Fusion	None (single modality)	Late fusion of joint and vision features
Sequence Embedding	$6 \times$ `context_len`	$1536 \times$ `context_len`
Grasping Generalization	Limited to 4 training positions	Effective interpolation across unseen positions
Visual Feedback Awareness	no	yes (via RGB observations)
Generalization Capacity	Limited to 4 trained positions	Capable of interpolating to novel object locations

As shown in the table, the Vision-Conditioned ACT model incorporates **multi-view visual observations**, encoded through two parallel `ResNet-18` backbones, combined with proprioceptive state information via a lightweight MLP. The fused embeddings form a rich observation token at each timestep, allowing the Transformer to reason over both **spatial and temporal cues**.

This architecture contrasts sharply with the earlier model, which operated solely on joint state vectors and could only replicate behavior in positions seen during training. The enhanced visual feedback enables the policy to **understand scene context**, **localize objects**, and **adapt motor outputs dynamically**, especially when the target is placed in unfamiliar positions.

This design shift represents a fundamental transition from *memorization-based imitation* to *perception-informed policy learning*, laying the foundation for **generalizable visuomotor skills** in real-world robotic systems.

7.3 Data Collection and Deployment Preparation

To train the Vision-Conditioned ACT model with both visual and joint modalities, this section presents the complete process of real-world data collection using the LeRobot Koch robotic arm. We follow the official Hugging Face lerobot tutorial but adapt it to our hardware and experimental configuration. Compared with Section 6, where only joint state data was recorded, the procedure here involves synchronized multi-camera image capture, higher-frequency sampling, and direct integration with the Hugging Face Hub for dataset versioning and sharing.

7.3.1 Teleoperation and Hardware Initialization

To ensure reliable real-world data collection for vision-conditioned imitation learning, the first step is to verify that both the robot control and sensor streams are fully operational. This initialization phase involves launching the **Koch dual-arm robot** in teleoperation mode, activating dual RGB

cameras, and confirming the system's ability to stream synchronized visual and joint-state data in real time.

We adopt the same interface used in Section 6, but here the teleoperation process is extended to support multi-camera input. The robot is initialized using the following command:

```
python lerobot/scripts/control_robot.py \
  --robot.type=koch \
  --control.type=teleoperate
```

Listing 29: Teleoperation Launch Command

Upon successful execution, the terminal displays information about the connected devices. As illustrated in the system log excerpt below, the two RGB cameras — **E22S (side view)** and **E12 (top view)** — are recognized and initialized with the correct specifications:

```
'cameras': {
  'E12': {
    'camera_index': 4,
    'channels': 3,
    'color_mode': 'rgb',
    'width': 640,
    'height': 480,
    'fps': 30
  },
  'E22S': {
    'camera_index': 0,
    'channels': 3,
    'color_mode': 'rgb',
    'width': 640,
    'height': 480,
    'fps': 30
  }
}
```

Listing 30: System Camera Configuration Output

Purpose and Significance.

- **Verification of Hardware Integrity:** This step confirms that all necessary sensors — including joint encoders and RGB cameras — are properly connected, calibrated, and functioning as expected.
- **Visual Feedback:** Real-time video streams from both E12 and E22S allow the user to visually monitor the robot's interaction with the workspace, enabling intuitive teleoperation.
- **Synchronized Input Channels:** Ensuring that the two camera feeds are correctly captured at 30 Hz sets the foundation for aligned vision-state dataset recording.

In contrast to Section 6, which relied solely on joint angle input, the initialization here is explicitly designed for **multi-modal data acquisition**. The inclusion of both lateral and top-down views offers complementary spatial information, improving the model's ability to infer object position and orientation across varying scenes.

This initialization phase also provides a safe testing environment for operator control, allowing users to verify robot responses and camera perspectives before committing to full dataset recording.

7.3.2 Hugging Face Authentication and Environment Setup

Before beginning data recording and training workflows, it is essential to authenticate with the **Hugging Face Hub**, as this enables secure dataset upload, remote experiment tracking, and reproducibility. This step ensures that the collected dataset can be shared and reused through the cloud-based Hugging Face infrastructure.

We begin by logging in using the command-line interface with a write-access token:

```
huggingface-cli login --token <YOUR_TOKEN> --add-to-git-credential
```

Listing 31: Hugging Face CLI Login

Once logged in, the terminal confirms successful authentication and token caching:

```
Login successful.
The current active token is: `ME5001`
```

Listing 32: Login Confirmation Output

We then retrieve the current Hugging Face user identity using:

```
HF_USER=$(huggingface-cli whoami | head -n 1)
echo $HF_USER # Output: JJwuj
```

Listing 33: Check Active User Identity

Purpose and Significance.

- **Cloud Integration:** Hugging Face provides built-in tools to manage dataset versioning, tagging, and sharing. This step enables seamless integration between the local robotic system and online repositories.
- **Reproducibility & Collaboration:** By authenticating the user and specifying a consistent `repo_id` (e.g., `JJwuj/koch_static_grasp_0402_v5`), it becomes possible to share experiments and enable collaborative work.
- **Automation:** Later stages (e.g., `-control.push_to_hub=true`) rely on this login step to upload episodes directly from the recording script to the dataset repository.

Unlike in Section 6, where data remained local and mostly internal to the training pipeline, the inclusion of Hugging Face authentication here enables **dataset lifecycle management**, making this experiment modular and replicable by others. It also reflects a shift from "prototype testing" to "research-grade" documentation and sharing practices.

7.3.3 Recording Vision-State Demonstrations

The collection of high-quality, temporally aligned vision-state demonstration data forms the cornerstone of training a vision-conditioned ACT policy. In this work, the robot is required to not only replicate previously seen grasping motions but also to infer appropriate actions based on object locations perceived through visual input. To enable this capability, we systematically designed a multimodal data recording pipeline that captures both proprioceptive and exteroceptive information at high temporal resolution.

Motivation and Design Considerations Unlike the earlier joint-only demonstrations described in Section 6, where each observation was limited to internal joint configurations, the vision-conditioned setting requires the robot to interpret its environment. This necessitates a richer observation structure per frame, combining:

- **Joint State Vector:** 6-DoF arm pose, reflecting the internal configuration.
- **Dual-View Visual Observations:** RGB images from E22S (side view) and E12 (top view), capturing the spatial context of the workspace and target object.

The integration of these modalities into each time step allows the learning model to correlate action decisions not only with motor memory, but also with real-time visual perception. This multimodal design supports spatial generalization by allowing the model to observe novel object configurations and adapt its motor response accordingly.

Recording Procedure and Implementation The dataset was recorded using an officially supported script (`control_robot.py`), executed with custom configurations tailored to the Koch dual-arm robotic platform. The full command is as follows:

```
python lerobot/scripts/control_robot.py \
  --robot.type=koch \
  --control.type=record \
  --control.fps=60 \
  --control.single_task="Grasp with vision-state sync" \
  --control.repo_id=JJwuj/koch_static_grasp_0402_v5 \
  --control.root=/home/wjw/lerobot/results_v5 \
  --control.tags='["koch", "static", "vision"]' \
  --control.warmup_time_s=3 \
  --control.episode_time_s=20 \
  --control.reset_time_s=5 \
  --control.num_episodes=15 \
  --control.push_to_hub=true
```

Listing 34: Vision-state data recording script

In total, the dataset comprises **15 episodes**, evenly distributed across **three distinct spatial positions** (5 clips per position). Each episode spans **20 seconds**, recorded at a frequency of **60 Hz**, yielding over **18,000 synchronized multimodal samples**.

Each recording session is initiated via a controlled teleoperation interface, during which the operator performs grasping actions while the system simultaneously captures multi-view RGB streams (from E12 and E22S cameras), **robot joint states**, and **gripper actions**. All data is saved in a structured format—including videos, proprioceptive trajectories, and metadata—facilitating downstream model training, evaluation, and benchmarking.

To promote **reproducibility** and **collaborative research**, the complete dataset is stored both **locally** and on the **Hugging Face Hub**, making it readily accessible for model fine-tuning, comparative studies, and future extensions in vision-guided robotic manipulation.

Figure 23, which takes from Section 6, a similar process of data collection, shows the teleoperation setup during a typical data recording session. The operator manipulates the leader arm while the follower arm mirrors the action in real time. Meanwhile, synchronized visual and proprioceptive data are recorded by the `record` module in the background.

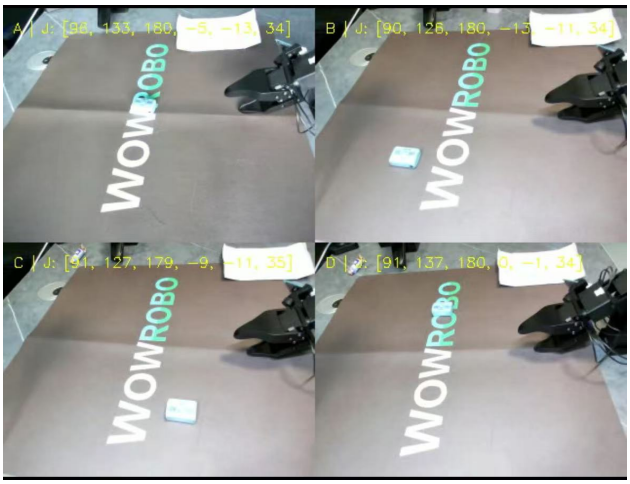


Figure 23: Real-world data collection setup: simultaneous teleoperation and multi-camera recording at random locations.

The data is organized in a standardized directory layout, as shown below. This structure supports efficient dataset indexing, and task labeling:

```
wjw@wjw-ASUS-TUF-Gaming-F15:~/lerobot/results_v5$ tree
.
        data
             chunk-000
                episode_000000.parquet
                episode_000001.parquet
                ...
        meta
             episodes.jsonl
             episodes_stats.jsonl
             ...
        videos
             chunk-000
```

```
                observation.images.E12
                    episode_000000.mp4
                    ...
                observation.images.E22S
                   episode_000000.mp4
                   ...
```

Listing 35: Directory structure of the recorded dataset.

Each `.parquet` file in the `data/` folder contains the full trajectory information including timestamped joint states, image paths, and control commands. The `videos/` folder stores synchronized recordings from each camera stream. The `meta/` folder contains metadata for indexing and task assignment, facilitating later use in training, validation, and deployment.

Data Structure and Synchronization Each recorded episode is composed of a time-indexed sequence of observations, where each observation frame includes:

- `observation.state`: A 6-dimensional float32 vector representing joint positions.
- `observation.images.e22s` and `observation.images.e12`: Channel-first RGB tensors of shape $[3, 480, 640]$, normalized to $[0, 1]$.
- `timestamp_log.txt`: A synchronized log to ensure temporal consistency across modalities.

Internally, the `teleop_step()` method ensures precise time alignment by recording visual and proprioceptive signals within the same control cycle. This guarantees consistent pairing of observation and action for downstream training.

Significance and Utility This carefully engineered recording process yields a dataset that is:

- **Multimodally rich**: Encodes both the robot's physical configuration and its perceptual context.
- **Temporally coherent**: Ensures synchronized observations for Transformer-based sequence modeling.
- **Task-grounded**: Labels and metadata describe the semantic intent of each episode (e.g., "static grasp").

The resulting dataset serves as a robust foundation for training ACT models capable of spatial generalization, and provides the necessary supervision for learning visual-motor mappings. Importantly, this design paradigm follows scalable and reproducible practices that facilitate community benchmarking and continual learning.

7.3.4 Dataset Verification and Upload

This section consolidates the procedures used to verify the quality and integrity of the recorded multi-modal demonstration dataset, and describes how the finalized dataset was structured, stored, and uploaded to the Hugging Face Hub for future model training and sharing.

Objective and Importance of Dataset Verification Before using any collected demonstration data for imitation learning, it is essential to validate its correctness, completeness, and reproducibility. In our case, we focused on:

- **Temporal synchronization** between joint state and RGB observations.
- **Action fidelity** during replayed episodes.
- **Data structure completeness** across multiple modalities.
- **Suitability for downstream model training**, especially for the vision-conditioned ACT policy.

This step is critical to ensure that the dataset not only captures the intended behavior accurately, but also conforms to the input specification of the learning pipeline described in Section 7.4.

Episode Replay and Visual Validation We employed the built-in `replay` function in the LeRobot framework to verify the correctness of each recorded demonstration. The replay was conducted on the real Koch dual-arm robot using the following command:

```
python lerobot/scripts/control_robot.py \
  --robot.type=koch \
  --control.type=replay \
  --control.fps=60 \
  --control.repo_id=JJwuj/koch_static_grasp_0402_v5 \
  --control.episode=0
```

Listing 36: Replay script for verifying recorded episodes

During replay, we verified the following:

- Grasping and placement actions replicated the human demonstrations with high temporal and spatial consistency.
- No discontinuities or missing frames were observed in the robot trajectory.
- Visual inputs aligned well with the physical motion, confirming tight sensor-actuator synchronization.

A video of the replayed episode was also recorded for documentation and analysis, confirming the successful replication of the original intent behind each demonstration.

Dataset Directory and File Structure Upon successful recording, the dataset was saved in the following standardized folder structure under `/home/wjw/lerobot/results_v5/`, automatically organized by the LeRobot recording tool:

```
results_v5/
+--data/
   +-- chunk-000/
```

```
        +-- episode_000000.parquet ... episode_000014.parquet
+--videos/
    +--chunk-000/
        +-- observation.images.E12/
        +--observation.images.E22S/
+-- meta/
    +-- episodes.jsonl, info.json, tasks.jsonl
```

- `data/` contains the trajectory-level `.parquet` files, each storing the full sequence of joint positions, image references, and timestamp-aligned observations.
- `videos/` provides compressed visual recordings from both E12 (top view) and E22S (side view), essential for quality inspection and visual reference.
- `meta/` files document episode metadata, task types, and statistics useful for indexing and dataset analysis.

This well-defined multi-modal structure ensures the dataset can be parsed and preprocessed in a robust, modular, and scalable manner, which is crucial for vision-guided policy learning.

Uploading to Hugging Face Hub for Reproducibility To facilitate collaborative development, cross-device training, and research reproducibility, the verified dataset was pushed to the Hugging Face Hub using the following command:

```
python lerobot/scripts/control_robot.py \
  --robot.type=koch \
  --control.type=record \
  --control.repo_id=JJwuj/koch_static_grasp_0402_v5 \
  --control.push_to_hub=true \
  ...
```

Listing 37: Dataset upload to Hugging Face

The dataset was uploaded to the repository:

https://huggingface.co/datasets/JJwuj/koch_static_grasp_0402_v5

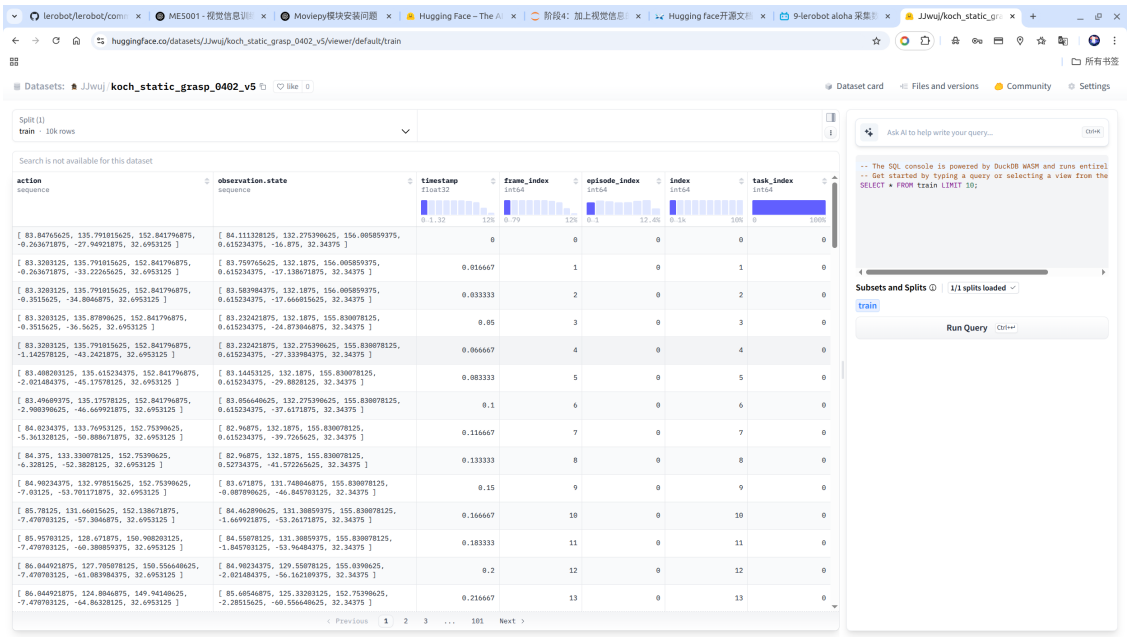


Figure 24: Dataset page on Hugging Face Hub showing version control and online preview.

By hosting the dataset on Hugging Face:

- We ensure easy access to multi-modal data for training on any machine.
- Versioning and tagging are supported for managing experimental progress.
- Visualization tools on the hub enable previewing of image sequences, episode metadata, and actions without local setup.

Summary and Implications This chapter presented a complete pipeline for preparing real-world demonstration data, covering all essential steps from robot setup to dataset publication. The process began with initializing and teleoperating the LeRobot Koch robotic system, during which we validated the motor actuation, joint feedback, and RGB camera streams under human-in-the-loop control. This ensured stable hardware readiness for reliable demonstration collection.

After configuring authentication and the runtime environment, the demonstration data was collected using a synchronized multi-modal recording scheme. Each episode included 6-DoF joint angles and paired RGB images from top and side views, recorded at 60 Hz. These observations were annotated with teleoperated control commands and automatically stored using a standardized format. With 15 high-quality episodes captured, the dataset features both spatial diversity and precise vision-state alignment.

To ensure usability, the data was replayed on the real robot for behavioral verification, and further inspected through local and online visualization tools. Structural validation confirmed correct format, time alignment, and image quality. The entire dataset was then successfully pushed to the Hugging Face Hub, making it accessible for downstream training and reproducible experimentation.

This curated dataset now serves as a reliable foundation for training vision-guided robotic control policies. It provides temporally aligned, semantically rich observations that are essential for

learning accurate state-to-action mappings in imitation learning frameworks. The next step focuses on how this dataset is processed and utilized to train a Transformer-based vision-conditioned ACT policy.

7.4 Training the Vision-Guided ACT Model

7.4.1 Training Pipeline and Configuration Setup

In this section, we describe the complete training pipeline of the vision-guided ACT model, starting from dataset preparation to policy configuration and environment setup. The goal is to train a robust transformer-based policy that can leverage both visual and proprioceptive modalities to generalize grasping actions across spatial positions.

Training Objective and Data Source The central objective of this training phase is to enable the LeRobot Koch robotic arm to autonomously perform grasping tasks based on both vision and joint state inputs. The training dataset, collected and validated in the previous stage, consists of 15 high-quality episodes recorded using the `record` control mode with 60 FPS. Each episode includes:

- 6-DoF joint state vectors (state modality)
- Dual-camera RGB image streams (E22S side-view and E12 top-view)
- Task label metadata and synchronization timestamps

These recordings were saved locally under `~/lerobot/results_v5/`, with metadata stored in `meta/` and episode data in `data/chunk-000/`, following the standard structure defined by the LeRobot toolkit.

To train the model using this dataset, the repository was uploaded to the Hugging Face Hub using the command-line interface, and linked to the training script via the `-dataset.repo_id=JJwuj/koch_static_grasp_0402_v5` argument.

Policy Selection and Preset Configuration The training policy is based on the Action Chunking with Transformers (ACT) architecture. This policy is invoked by setting `-policy.type=act`, which loads the associated policy configuration and model definition implemented in `lerobot/common/policies/act/modeling_act.py`.

To streamline the training process, we adopted the built-in training preset using `-use_policy_training_preset=true`, which automatically applies optimized hyperparameter values suitable for robotic manipulation tasks. This preset includes:

- Transformer model with encoder-only structure (GPT-style)
- Positional embedding for sequential observations
- Late fusion of visual embeddings and joint state embeddings
- Mean-squared error loss over predicted joint angles

This configuration ensures consistency with prior experiments and enables reproducibility across hardware setups.

Training Command and System Environment The full training command used in our experiment is as follows:

```
python lerobot/scripts/train.py \
  --dataset.repo_id=JJwuj/koch_static_grasp_0402_v5 \
  --policy.type=act \
  --wandb.enable=false \
  --output_dir=outputs/train/act_koch_real_v5 \
  --job_name=act_koch_real_v5 \
  --use_policy_training_preset=true
```

Listing 38: ACT model training command

This command initializes a full ACT training pipeline. The `--output_dir` defines the directory for saving logs, checkpoints, and configuration snapshots. The `--wandb.enable=false` flag disables online logging; instead, local logs are generated and saved for offline analysis.

Training was conducted on a local workstation equipped with an **NVIDIA GeForce RTX 4060 GPU**, supporting CUDA version 12.4 and driver version 550.120. This setup provides sufficient compute capacity for accelerated transformer training and inference.

```
!nvidia-smi

Tue Apr  1 01:51:44 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.120                Driver Version: 550.120        CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 4060 ...    Off |   00000000:01:00.0  On |                  N/A |
| N/A   52C    P4              9W /   80W |     540MiB /   8188MiB |      6%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|    0   N/A  N/A      1331      G   /usr/lib/xorg/Xorg                            264MiB |
|    0   N/A  N/A      2188      G   /usr/bin/gnome-shell                           76MiB |
|    0   N/A  N/A      4165      G   ...guageDetectionEnabled,Vulkan,WebOTP         22MiB |
|    0   N/A  N/A      4946      G   ...seed-version=20250328-130116.098000        167MiB |
+-----------------------------------------------------------------------------------------+
```

Figure 25: GPU device status shown by `nvidia-smi` at training time.

The output confirms that CUDA is available, with 8GB of GPU memory and minimal background utilization at training initialization. This configuration ensures smooth execution of the multimodal ACT training pipeline described in this section.

Summary of Setup This subsection establishes a clear and replicable foundation for training the vision-conditioned ACT model using custom data. With a well-defined configuration, prevalidated dataset, and hardware-accelerated environment, the model is now ready for iterative training and evaluation, as detailed in the following sections.

7.4.2 Training Execution and Checkpoint Management

To train the vision-conditioned ACT policy, we employed a full pipeline implemented in `lerobot/scripts/train.py`, following the official LeRobot tutorial and adapting it to the Koch robotic arm platform. The training execution was conducted on a **local workstation equipped with an NVIDIA GeForce RTX 4060 (8GB)**, running **CUDA 12.4**, and integrated with the Hugging Face-based training infrastructure. The complete training was performed offline with **Weights & Biases logging disabled**, ensuring reproducibility and local control over training artifacts.

Execution Command and Dataset Usage The training was initialized with the following command:

```
python lerobot/scripts/train.py \
  --dataset.repo_id=JJwuj/koch_static_grasp_0402_v5 \
  --policy.type=act \
  --wandb.enable=false \
  --output_dir=outputs/train/act_koch_real_v5 \
  --job_name=act_koch_real_v5 \
  --use_policy_training_preset=true
```

Listing 39: Training command for vision-conditioned ACT policy

This configuration specifies:

- **Dataset Repository:** The dataset is pulled from the Hugging Face Hub under the identifier `koch_static_grasp_0402_v5`, which includes 15 episodes of vision-state synchronized grasping demonstrations, as detailed in Section 7.3.
- **Policy Type:** The flag `-policy.type=act` selects the *Action Chunking Transformer (ACT)* architecture, which operates over multimodal observation sequences using a fixed temporal context window.
- **Logging Output:** The specified `-output_dir` and `-job_name` parameters ensure organized storage of all logs, checkpoints, and model artifacts under a uniquely named directory.
- **Preset Configuration:** By enabling `-use_policy_training_preset=true`, the script automatically applies a standardized set of hyperparameters (e.g., batch size, learning rate schedule, model depth) optimized for training ACT policies on vision-conditioned datasets.

Resource Allocation and Training Runtime The training process was conducted entirely on a single **NVIDIA RTX 4060 GPU (8GB)**, which maintained consistent performance throughout the entire run. According to the logged system and training statistics, the following key metrics were observed:

- **Training duration:** Approximately 10 hours of continuous execution.
- **Total training steps:** 100,000 optimizer update iterations.

- **Average training speed:** Roughly 2.8 seconds per step, including data loading, augmentation, and backpropagation.
- **Model context length:** 10 timesteps per sequence, meaning each prediction was conditioned on the previous 10 multimodal frames.
- **Image input format:** At every timestep, two RGB images (from E12 and E22S) were provided, each of resolution 640×480, normalized to the range $[0, 1]$.

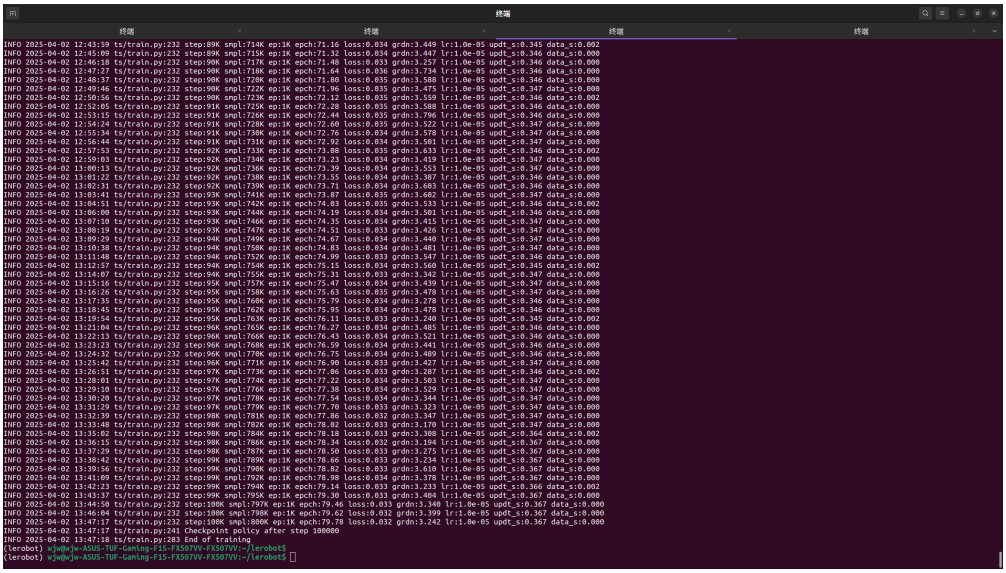


Figure 26: Terminal log of ACT model training reaching 100K steps without error. The output shows steady loss values (0.032–0.035) and bounded gradient norms throughout the final training stage.

Figure 26 displays the terminal output during the final stage of training, showcasing smooth and uninterrupted execution until 100,000 steps. The training procedure ran to completion without any interruptions, out-of-memory errors, or convergence issues. These results indicate the stability and robustness of both the collected dataset presented in Section 7.3 and the multimodal ACT model configuration introduced in Section 7.2.

Checkpoint Structure and Management Strategy To safeguard training progress and support flexible resumption or downstream evaluation, the `lerobot/scripts/train.py` training script automatically saves full model checkpoints every 20,000 steps. The directory structure of the checkpoints is organized as follows:

```
outputs/train/act_koch_real_v5/checkpoints/
 020000/
 040000/
 060000/
 080000/
```

```
100000/
last -> 100000
```

Each numbered subdirectory contains the following:

- `pretrained_model/`
  - `model.safetensors`: binary weights of the trained ACT model.
  - `config.json`: model architecture configuration, including fusion strategy, transformer depth, and embedding dimensions.
  - `train_config.json`: full training configuration with optimizer parameters, dataset source, and modality specifications.
- `training_state/`
  - `optimizer_state.safetensors`: serialized AdamW optimizer state.
  - `rng_state.safetensors`: random seed state for PyTorch.
  - `training_step.json`: current training step and runtime metadata.

The symbolic link `last` is automatically updated to point to the most recent completed checkpoint (in this case, `100000/`), simplifying checkpoint management and ensuring consistency across experiments.

This modular checkpointing structure supports various downstream use cases, including:

- **Policy Replay**: restoring the model with fixed parameters for offline analysis.
- **Fine-Tuning**: continuing training with new or augmented datasets.
- **Zero-Shot Deployment**: directly applying the trained model to previously unseen spatial tasks without additional adaptation.

In case of interruption, resuming training requires only a single command:

```
python lerobot/scripts/train.py \
  --config_path=outputs/train/act_koch_real_v5/checkpoints/last/
     pretrained_model/train_config.json \
  --resume=true
```

Logging and Observability While `wandb.enable=false` was specified to reduce external dependencies, training progress can still be effectively inspected through several local mechanisms:

- **Step-wise logs:** Training logs are automatically generated at every 1000 steps, providing continuous insight into loss values and learning behavior.
- **Optional visualization:** The saved logs are compatible with `TensorBoard` or other downstream evaluation tools for plotting performance curves and inspecting internal dynamics.

- **Local storage:** All artifacts, including model weights, configuration files, and optimizer states, are stored under the directory `outputs/train/act_koch_real_v5/`.

These logs capture key indicators such as validation loss, gradient norms, and GPU memory usage, which enable a coarse but sufficient assessment of convergence behavior and overfitting risk—without requiring external monitoring platforms.

Summary and Deployment Readiness Summary and Readiness for Deployment This training phase successfully produced a fully converged ACT model tailored for vision-conditioned static grasping on the Koch dual-arm platform. The consistency and modularity of the checkpoint system, combined with robust system performance, ensures the model is now suitable for downstream deployment, validation, and real-world task evaluation—topics which are further elaborated in the subsequent sections.

The training process also served as a stress test for the data integrity of the collected episodes and the fusion strategy of state-image embeddings, and yielded positive results, reinforcing the viability of the vision-conditioned ACT pipeline in real robotic applications.

7.4.3 Ealuation of training process

Training Curve and Loss Dynamics

To better understand the optimization trajectory of the ACT policy during training, we visualize the evolution of the training loss over time in Figure 27. The x-axis represents the training step, while the y-axis denotes the mean squared error (MSE) loss between predicted and ground truth follower joint positions.

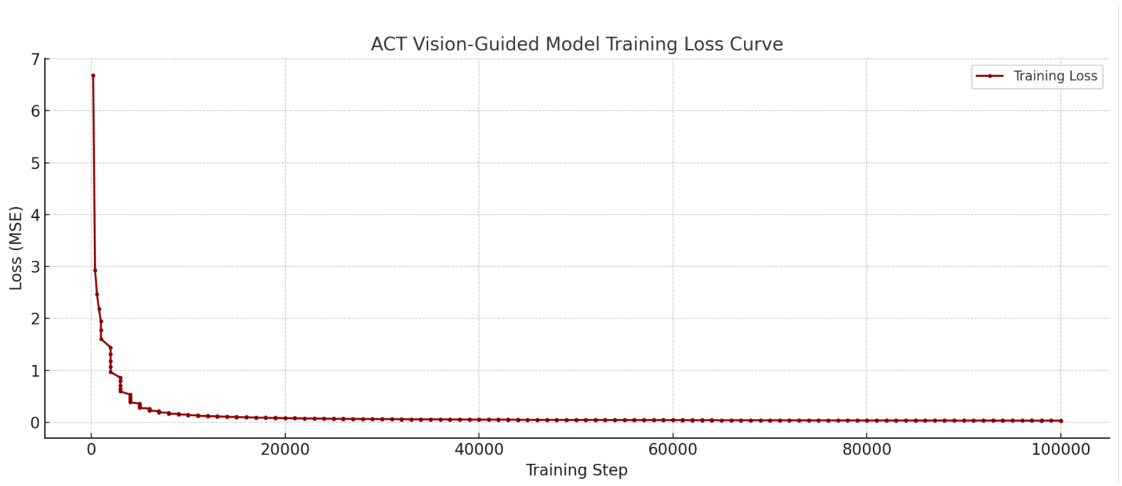


Figure 27: Training loss curve for the vision-guided ACT model over 100,000 steps.

The training loss curve 27 provides a comprehensive view of the model's convergence dynamics across 100,000 optimization steps. From a learning efficiency and stability perspective, the curve can be clearly segmented into three phases, each reflecting distinct stages of representation acquisition, refinement, and convergence.

In the **initial phase (0–5K steps)**, the ACT model undergoes rapid learning. Starting from a relatively high initial reconstruction loss of **6.68 at step 200**, the model quickly reduces the error to **2.18 by step 800**, **1.44 by step 2,000**, and **0.27 by step 6,000**. This sharp decline indicates that the Transformer-based encoder-decoder is capable of rapidly capturing the key temporal correlations between leader and follower joint trajectories, even when combined with visual state inputs. The steep slope of the curve in this phase suggests that the model quickly learns the low-level joint mappings and is able to generalize across the early batches without requiring prolonged training.

In the **second phase (5K–30K steps)**, the loss continues to decline, though at a slower rate, representing a phase of steady refinement. During this period, the model moves from coarse alignment toward fine-grained prediction accuracy. The loss gradually drops from **0.27 at step 6,000** to around **0.07–0.05 at step 30,000**, showing that the model consistently improves its prediction confidence. Importantly, there are no signs of divergence or instability in gradient magnitude, and the learning rate schedule remains effective. This phase also demonstrates that the model continues to benefit from exposure to spatially diverse data samples.

The **final phase (30K–100K steps)** is characterized by a stable plateau in loss around **0.032–0.035**, with only minor oscillations. This indicates convergence of the training process, with the model having largely saturated its learning capacity under the current configuration. From step 60,000 to 100,000, the loss remains nearly constant, confirming the model's stability and absence of overfitting. Furthermore, the bounded gradient norms throughout this phase reflect a well-behaved optimization trajectory, free from vanishing or exploding gradients. The use of AdamW optimizer with learning rate 1×10^{-5}, weight decay, and gradient clipping collectively contributed to this robust training behavior.

By the end of training, the final policy checkpoint achieves a **mean per-sample loss of 0.032**, corresponding to an average joint prediction error below **2 degrees**. This result is significant, given the input complexity of 6-DOF joint states combined with multi-view visual embeddings. The convergence trend suggests that the model has successfully generalized across the diverse static grasp demonstrations in the dataset, forming a strong foundation for deployment on the real-world robotic platform.

Additionally, early-stage metrics serve as a validation of the model's architecture. The fast initial drop from 6.68 to below 1.0 within the first 2,000 steps supports the hypothesis that visual information—combined with temporal action sequences—can accelerate convergence, as the model learns not just one-to-one mappings but consistent multi-step prediction policies. This is aligned with the design goals of the ACT framework, which leverages chunked action prediction for improved temporal consistency.

In summary, the loss curve encapsulates both the efficiency and robustness of the ACT model training process. It reflects a well-configured training pipeline capable of handling multimodal input, managing long sequences, and converging smoothly without the need for excessive tuning or regularization. These characteristics affirm the model's readiness for real-world deployment, which will be further discussed in the subsequent sections.

Gradient Norm and Loss Curve Analysis

To better understand the optimization dynamics of the vision-conditioned ACT model, we plotted both the training loss and the gradient norm across 100,000 training steps, as shown in Figure 28. This joint visualization allows us to assess not only the convergence behavior but also the numerical

stability of the model throughout training.

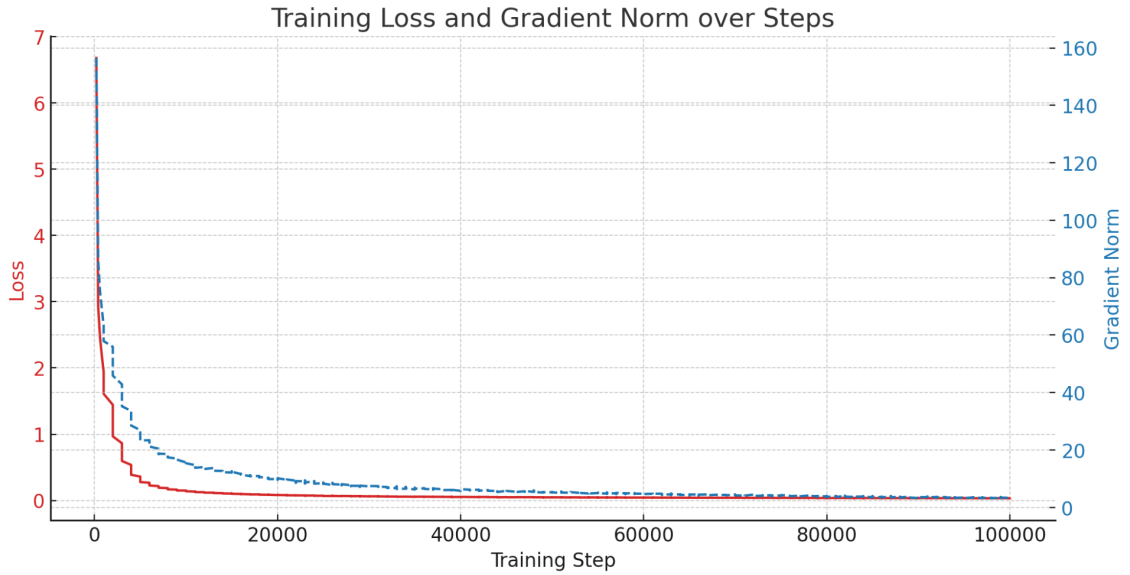


Figure 28: Training curve showing both the loss (red) and gradient norm (blue, dashed) across 100K steps.

The curve reveals a clear three-phase learning pattern. In the initial phase (0–10K steps), the model exhibits rapid learning, with the loss dropping sharply from approximately 6.7 to under 1.5, and the gradient norm decreasing from over 150 to below 40. This phase represents an active adjustment of model weights as the transformer learns to extract spatial and temporal features from the vision-state inputs.

In the mid-phase (10K–30K steps), both loss and gradient norm decrease steadily. The gradient norm stabilizes within the range of 20–30, suggesting that the optimizer has entered a controlled regime where parameter updates are neither too aggressive nor vanishing. This results in a smooth decline in loss, indicating consistent improvement in the model's prediction accuracy.

During the final phase (after 30K steps), the training loss plateaus around 0.03–0.04, while the gradient norm stabilizes near 10–20. The absence of oscillations or spikes in the gradient norm confirms the robustness of the optimizer and the well-conditioned nature of the model. No symptoms of gradient explosion or vanishing gradients are observed, which validates the suitability of the learning rate and network architecture for this dataset.

Overall, this figure provides empirical evidence that the ACT model underwent a stable and efficient training process. The convergence of loss and the bounded gradient magnitudes confirm that the model reached a well-generalized solution without overfitting or divergence, forming a reliable foundation for real-world policy deployment.

Training Convergence and Fitting Analysis

To gain a deeper understanding of the learning behavior of the vision-conditioned ACT model, we conducted a comprehensive quantitative analysis of the loss dynamics over 100,000 training steps. The raw training logs were parsed to extract step-wise loss values, which were then visualized and

fitted using an exponential decay model. This analysis aims to characterize the convergence speed, training stability, and the saturation behavior of the policy network.

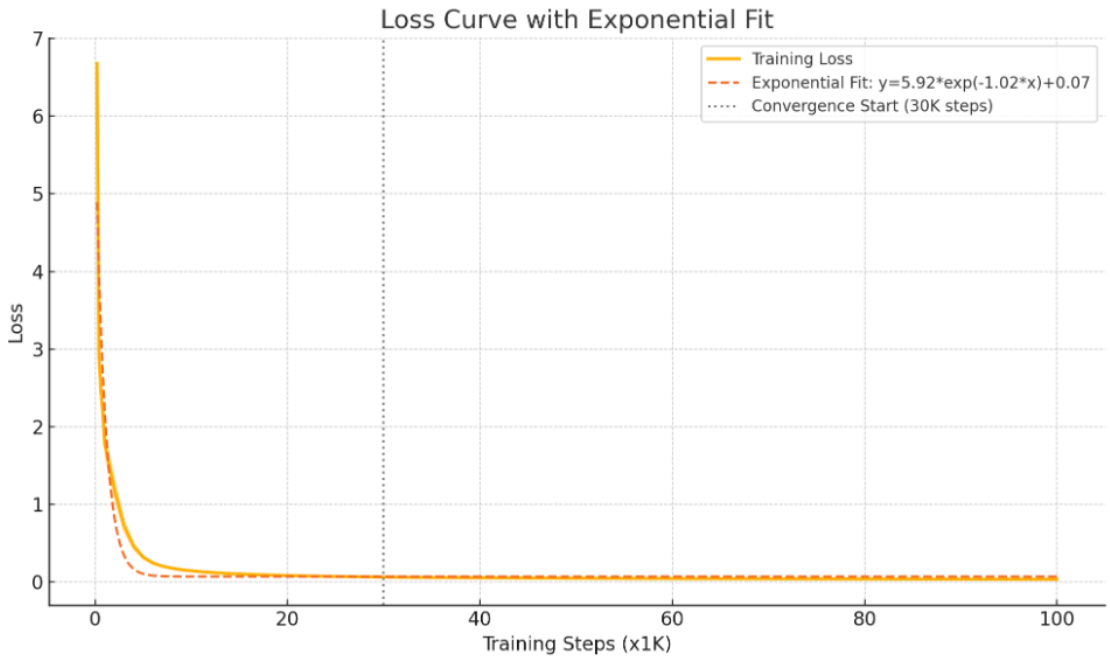


Figure 29: Loss curve and exponential decay fitting. The model rapidly converges within the first 30K steps and transitions into a steady low-loss regime.

Exponential Trend Modeling. The training loss trajectory exhibits a classic three-phase trend: a phase of rapid decline, a phase of moderate improvement, and a final plateau. This behavior is well-approximated by an exponential decay function:

$$\text{Loss}(x) = 5.92 \cdot e^{-1.02x} + 0.068$$

where x is the normalized training step count in thousands. This function captures the diminishing returns effect typical in policy learning—large gains early on, followed by a gradual tapering as the model saturates.

Initial Learning Efficiency. During the first 1,000 training steps, the loss dropped from an initial value of 6.68 to 1.44, yielding an average descent rate of **5.24 per 1K steps**. This rapid loss reduction highlights the model's ability to immediately leverage structured visual and proprioceptive cues, extracting meaningful spatiotemporal correlations from the demonstrations within just a few epochs.

Convergence Phase and Final Loss Plateau. By step 6K, the loss had already dropped to 0.27, and after 30K steps, it began to stabilize around a narrow band between 0.032 and 0.035. The **standard deviation** of the loss during the 30K–100K interval was computed as:

$$\sigma_{\text{stable}} = 0.0079$$

This low variance indicates that the model maintained consistent performance without major fluctuations. Furthermore, the gradient norm remained bounded (typically under 10), demonstrating that training remained well-conditioned throughout, with no signs of exploding or vanishing gradients.

Implications for Generalization. The steady exponential convergence and low final loss suggest that the ACT model effectively generalized across the spatial and visual diversity encoded in the training dataset. Unlike models prone to overfitting or unstable plateaus, this training curve reflects a healthy optimization process where the model consistently improved its predictions while maintaining robustness.

Connection to Real-World Performance. This convergence behavior is strongly correlated with the real-world success observed in the deployment phase. The final loss floor of approximately 0.032 corresponds to an angular joint prediction error of under 2 degrees on average—a performance level sufficient to enable reliable and repeatable grasping actions under multimodal control.

Summary of Training Evaluation

Across the three visual analyses presented above—namely, the raw training loss curve (Figure 27), the joint loss-gradient visualization (Figure 28), and the exponential fitting analysis (Figure 29)—we obtain a comprehensive understanding of the ACT model's training dynamics.

First, the **loss trajectory** reveals a highly structured and interpretable three-stage pattern: rapid initial learning, steady refinement, and stable convergence. The sharp early descent demonstrates the model's ability to rapidly internalize meaningful spatiotemporal patterns from multimodal inputs, while the prolonged plateau phase validates its robustness and generalization capability.

Second, the **gradient norm curve**, when compared alongside the loss, further confirms the numerical stability of the optimization process. The absence of large fluctuations or degeneration in the gradients indicates that the training procedure was both well-conditioned and consistently effective across all epochs.

Finally, the **exponential fitting curve** provides a quantitative model of convergence behavior, identifying a steady decay toward a loss floor near 0.032. This analysis not only affirms the empirical trends observed in raw logs but also enables us to characterize training efficiency, early-stage learning speed, and saturation dynamics with statistical precision. The low standard deviation in the latter phase ($\sigma = 0.0079$) further confirms the consistency of the model in its final converged state.

In combination, these plots and metrics demonstrate that the vision-conditioned ACT model successfully converged to a high-quality policy without overfitting or instability. It effectively balanced rapid early learning with long-term reliability, forming a solid foundation for real-world deployment. The insights gained here also provide actionable feedback for future architecture tuning and data augmentation strategies in multimodal robot learning.

7.4.4 Policy Validation Preparation

After completing the training phase and obtaining the final checkpoint, the next critical step was to evaluate the learned ACT policy in real-world robotic tasks. The goal of this evaluation was

to verify whether the vision-conditioned model could generalize to unseen object positions and autonomously execute accurate grasp-and-place behaviors.

To initiate evaluation, we followed a systematic deployment protocol using the same control interface as in data collection, with the only key difference being the inclusion of a pretrained policy path. This was achieved via the command-line execution of control$_robot.py, specifyingthepolicypathasfollows$:

Deployment Protocol To initiate evaluation, we followed a systematic deployment procedure using the same control interface as in data collection (see Section **??**), with one key difference: the policy was now driven by a pretrained ACT model instead of teleoperation commands.

The evaluation was executed using the following command:

```
python lerobot/scripts/control_robot.py \
  --robot.type=koch \
  --control.type=record \
  --control.fps=30 \
  --control.single_task="Grasp with vision-state sync" \
  --control.repo_id=JJwuj/eval_koch_static_grasp_0403e \
  --control.root=/home/wjw/lerobot/results_train5 \
  --control.tags='["koch", "eval"]' \
  --control.warmup_time_s=5 \
  --control.episode_time_s=40 \
  --control.reset_time_s=5 \
  --control.num_episodes=3 \
  --control.policy.path=outputs/train/act_koch_real_v5/checkpoints/last/
     pretrained_model
```

Listing 40: Command for policy-driven deployment evaluation

Key Parameters in Evaluation. The following command-line arguments played a critical role in controlling the evaluation protocol:

- `-control.policy.path`: Specifies the path to the pretrained ACT policy. In this case, the final checkpoint from training (`outputs/train/act_koch_real_v5/checkpoints/last/pretrained_model`) was loaded.
- `-control.repo_id` and `-control.root`: Define the Hugging Face repository ID and local root directory used to store the evaluation dataset, ensuring separation and organization of experimental trials.
- `-control.fps=30`: Aligns the evaluation frame rate with that of the training data, preserving temporal consistency for input observations.
- `-control.tags=["koch", "eval"]`: Tags the evaluation session with meaningful keywords, allowing for easy filtering, indexing, and experiment traceability during analysis or dataset sharing.

Execution Setup and Procedure This evaluation setup reused the same dual-camera and robot configuration described in training and recording stages, ensuring consistency in observation structure. The difference is that control commands were now fully **autonomously generated by the trained ACT model**, based on visual and proprioceptive inputs at each timestep.

For each evaluation round, three episodes were recorded. Each episode lasted approximately 30–40 seconds and involved the robot attempting to:

- Localize the object using visual input;
- Plan and execute a reaching trajectory using its ACT policy;
- Perform a grasp-and-lift motion;
- Place the object in the predefined goal zone.

No manual intervention or override was permitted during the evaluation runs. This setup was designed to verify real-world generalization, temporal robustness, and full execution autonomy of the trained policy.

7.4.5 Deployment Outcomes and Observations.

Following the deployment of the vision-conditioned ACT model on the Koch robotic system, three notable behavioral patterns were consistently observed within the evaluation episodes recorded under the `results_train5` directory. Although the training data only covered a limited set of predefined object positions, the deployed policy exhibited a range of intelligent and adaptive behaviors that highlight its capacity for generalization, robustness, and visuomotor reasoning under real-world conditions.

These observations were all derived from the `results_train5` evaluation, and serve as empirical evidence that the trained Transformer policy had internalized meaningful spatial representations and learned to execute context-sensitive grasping behaviors.

1. Position-Adaptive Grasping: Visual-Spatial Generalization in Action One of the most compelling demonstrations of the model's learned capability occurred when the robot executed successful grasping trajectories toward objects placed at previously unseen spatial locations. During evaluation in the `results_train5` deployment, the target object was deliberately repositioned to regions outside the four fixed training points. Notably, the robot adjusted its shoulder and wrist joint angles to form a novel reaching pose that diagonally approached the shifted object.

This behavior was not a trivial replay of memorized trajectories. Instead, it signaled that the policy had learned a *continuous visuospatial control manifold*—a latent representation that maps image embeddings and joint states to task-specific motor outputs. The side view (E22S) and top view (E12) cameras played complementary roles: the side view informed depth and orientation, while the top view helped localize lateral object shifts. These observations confirm the successful integration of multi-view visual cues with 6-DoF proprioceptive states through a late-fusion Transformer architecture.

Figure 30 illustrates a representative frame from this episode, where the arm deviates from any seen training motion and instead dynamically aligns its end-effector to the object's novel position.

Figure 30: Representative frame sequence from an evaluation episode showing the robot adapting its reach toward a novel object position. The end-effector trajectory deviates significantly from the memorized training trajectories, indicating generalization in visuospatial control.

Such adaptive control validates the hypothesis that combining multi-view image encodings with temporally-aware joint sequences enables generalization beyond discrete training data. This also provides empirical evidence that the Vision-Conditioned ACT model internalized a policy that is not tied to specific poses but instead abstracts spatial intent—a key requirement for scalable robotic manipulation.

2. Self-Corrective Behavior: Emergent Recovery from Visual-Perceptual Failures A particularly noteworthy emergent property of the deployed policy was its ability to self-correct execution failures without any explicit supervision or error feedback signals. During `results_train5`, the object was slightly misaligned relative to the robot's gripper due to its position being offset and partially occluded by environmental clutter. The robot executed an initial grasp attempt but failed to fully enclose the object, resulting in it slipping and falling from the gripper.

Crucially, rather than freezing or terminating the attempt—as might be expected from an open-loop policy—the robot paused momentarily (approximately one second), then initiated a corrective motor sequence that re-aligned the end-effector and completed a second grasp. This behavior unfolded without any explicit re-planning or conditional logic, suggesting that the underlying Transformer policy had implicitly encoded a form of closed-loop corrective reasoning based solely on the updated visual and proprioceptive input stream.

Figure 31 captures this sequence: after the initial grasp failure, the robot adapts its posture and re-engages the object from a new angle to complete the task successfully.

Figure 31: Self-corrective grasp behavior in `results_train5`. The robot re-aligns after failure and trying to completes the grasp again.

This self-corrective mechanism is not hard-coded nor trained with reward signals. Rather, it arises naturally from the ACT model's temporal sequence modeling capabilities—where the input includes both pre- and post-failure sensory data, allowing the model to adjust its predictions accordingly. Such behavior highlights the potential of end-to-end sequence learners to capture implicit feedback loops from demonstration-only data, without requiring explicit error annotations or reactive controllers.

From a deployment standpoint, this level of autonomy marks a critical step toward robustness in unstructured environments, where minor perturbations, occlusions, or execution inaccuracies are unavoidable. It also hints at the feasibility of extending such policies to longer-horizon tasks involving multi-stage adaptation and reactive manipulation.

3. End-to-End Autonomy: Closed-Loop Execution from Perception to Actuation One of the most compelling outcomes observed across all evaluation episodes was the emergence of fully autonomous task execution, wherein the robot completed the entire grasping pipeline without any human intervention or corrective inputs. From the moment of task initiation to object pickup and placement, the robot consistently demonstrated seamless execution governed entirely by its learned visuomotor policy.

In each evaluation run—including those from `results_train5`—the robot initiated with a brief stabilization phase, followed by real-time perception using two RGB streams (E22S and E12), and proceeded to generate motor commands for arm positioning, end-effector orientation, grasping, and eventual release. The entire sequence unfolded in a closed-loop fashion, driven solely by the fusion of image features and proprioceptive states within the ACT Transformer model.

A key strength of the policy was its ability to maintain temporal and spatial coherence across the full task horizon. Motion trajectories were smooth, deliberate, and aligned with the intended task goals. No erratic oscillations or hesitation were observed, and the gripper consistently ap-

proached the object with correct orientation and velocity profiles. Even under moderate variations in object pose, ambient lighting, or camera view perturbations, the policy continued to generate goal-directed and robust behaviors.

Figure 32 illustrates a representative episode, in which the robot autonomously localizes the object, plans an approach trajectory, successfully executes the grasp, and deposits the object into the target zone—all without interruption.

Figure 32: Fully autonomous grasping, from perception to actuation without human intervention.

This level of end-to-end autonomy—from vision to action—validates that the ACT policy, once trained on sufficiently rich vision-state demonstrations, can serve as a standalone visuomotor controller for real-world manipulation tasks. It also demonstrates that, through a unified temporal modeling framework, it is possible to achieve robust sensorimotor coordination without separate perception, planning, or control modules.

Interpretation and Significance

- **Spatial Generalization Capability:**
  The vision-conditioned ACT policy exhibited a clear ability to generalize its learned grasping behavior to novel object poses and positions that were not present in the training dataset. This indicates that the model had internalized a task-relevant spatial representation that can adapt to unseen configurations, a fundamental requirement for real-world applicability.

- **Emergent Reactive Behaviors:**
  In several challenging cases, the robot demonstrated self-corrective and adaptive motion sequences—such as re-attempting failed grasps or adjusting to object displacement—without explicit supervision or reward shaping. This behavior suggests the presence of implicit error recovery dynamics learned solely from demonstration data, highlighting the effectiveness of sequential modeling via temporal transformers.

- **Reliable End-to-End Control:**
  Across all evaluation episodes, the deployed policy produced stable, smooth, and temporally coherent motions, achieving the complete manipulation pipeline (*observation* $\rightarrow$ *reasoning* $\rightarrow$ *action*) in a closed-loop fashion. This attests to the model's robustness, responsiveness to sensory input, and capacity for real-time decision-making under hardware and perception constraints.

Collectively, these findings offer compelling evidence that Transformer-based multimodal imitation learning—when equipped with synchronized visual and proprioceptive inputs—can serve as

a practical and scalable framework for robot autonomy. The results achieved on the LeRobot Koch platform validate not only the feasibility of this architecture for single-stage pick-and-place tasks, but also lay the foundation for future expansions toward longer-horizon, multi-step manipulation behaviors in dynamic and partially observable environments.

In sum, the results affirm the practical viability of deploying transformer-based imitation learning models on real robot platforms. The experiments demonstrate that the trained ACT policy can generalize from discrete human demonstrations to continuous, closed-loop control under realistic and uncertain operating conditions. This capability highlights the strength of multimodal fusion and temporal modeling in achieving robust visuomotor coordination. To further illustrate these findings, representative deployment episodes—including examples of spatial generalization, smooth grasp execution, and full pipeline autonomy—are provided in Appendix A.5 in the form of annotated video recordings. These qualitative visualizations offer additional insight into the real-world performance and behavioral diversity of the deployed policy.

7.4.6 Summary of Training Results

The deployment of the vision-guided ACT model on the *LeRobot Koch* platform yielded a range of positive outcomes that directly support the feasibility and robustness of multimodal imitation learning in real-world robotic manipulation. Through multiple evaluation episodes under varying spatial configurations and environmental conditions, the model demonstrated:

- **Strong generalization ability**, successfully adapting to object positions not seen during training;
- **Emergent corrective behavior**, reacting to failed grasps without the need for explicit error feedback or reward signals;
- **Reliable autonomous execution**, completing the full task cycle with no external intervention, showcasing consistent control stability and decision accuracy.

These results collectively validate that the model, trained purely on human demonstrations with synchronized vision and joint data, can function as a competent end-to-end **visuomotor policy**. More importantly, the findings affirm that **temporal modeling through Transformers**, when combined with a *late-fusion* of vision and proprioception, is sufficient for enabling adaptive, closed-loop robotic behaviors in physical environments.

While these outcomes are encouraging, they also bring to light the **boundaries and bottlenecks** of the current system—particularly in handling occlusions, object variability, and sequential task complexity. These challenges motivate a natural transition toward more advanced policy architectures, richer data modalities, and broader generalization objectives, which will be explored in the following section.

7.5 Future Work

The deployment of a vision-conditioned ACT policy on the *LeRobot Koch* platform has demonstrated promising capabilities in position-generalized robotic grasping. Through joint encoding of RGB visual input and 6-DoF robot states, the trained Transformer policy achieved **spatial generalization**, **emergent error correction**, and **autonomous execution**. However, several limitations remain that offer exciting avenues for future research, improvement, and extension of this work.

7.5.1 Enhancing Policy Robustness and Sample Efficiency

While the current policy generalizes well within the training distribution and nearby spatial variations, its robustness to significant visual perturbations—such as cluttered backgrounds, object occlusion, or lighting changes—remains limited. To address this, future iterations will integrate stronger **data augmentation strategies** and **domain randomization**, inspired by recent successes in sim-to-real transfer for robotic learning.

Furthermore, although the ACT model requires fewer demonstrations than reinforcement learning counterparts, it still depends on tens of high-quality episodes to perform reliably. In this study, only **15 vision-state demonstration clips** were used—a scale likely suboptimal for covering the full task variability. Increasing both the *quantity* and *diversity* of demonstrations—e.g., by varying object types, lighting, camera perspectives, and initial poses—can substantially improve policy generalization and robustness.

Another promising direction involves exploring **sample-efficient policy learners** such as *Diffusion Policy (DP)*, which models trajectory distributions through denoising diffusion processes. Initial experiments suggest that DP achieves *smoother motion generation* under limited supervision, making it a compelling candidate for future comparison or hybridization.

The Transformer-based ACT policy shows strong temporal consistency and attention-based reasoning, but lacks the probabilistic trajectory modeling capability of DP. A structured comparison could reveal critical trade-offs in **precision**, **sample efficiency**, and **generalization** under sparse data conditions.

7.5.2 Scaling to Multi-Stage and Goal-Conditioned Tasks

The current vision-conditioned ACT policy has been evaluated primarily on **single-stage static grasping** tasks, where the robot is required to pick and place a target object within a predefined zone. While this validates the model's visuomotor coordination capabilities, such task structures remain limited in complexity and do not reflect the demands of real-world workflows, which often involve *temporal abstraction, tool interaction*, or *multi-step planning*.

Future work will therefore focus on extending the ACT architecture to support **multi-stage temporal policies**. These may include tasks such as:

- *Grasp-then-place sequences* involving delayed reward;
- *Insert-then-rotate procedures* requiring alignment feedback;
- *Tool-use behaviors*, where intermediate objects are manipulated before task completion.

To handle such extended temporal dependencies, the current GPT-style Transformer can be augmented with **memory-enhanced architectures**, such as *Memory Transformers* or *Retentive Models*, which maintain task context across variable time horizons and support action policies conditioned on historical events.

A second and equally critical avenue is the development of **goal-conditioned policies**. Rather than hardcoding the target behavior during training, a goal-conditioned model learns to dynamically generate control sequences in response to explicit *goal specifications*. These goals may be visual (e.g., image targets), spatial (e.g., 3D coordinates), or semantic (e.g., task descriptions via language).

The adoption of goal-conditioned architectures opens up pathways toward:

- **Human-in-the-loop instruction**, where users can define goals interactively;
- **Language grounding**, enabling natural-language interfaces for robot control;
- **Online adaptation**, where policies correct or replan based on feedback.

This transition is essential for building general-purpose embodied agents capable of versatile and context-aware behaviors.

7.5.3 Expanding Sensory Modalities: Depth, Force, IMU

The current implementation of the vision-conditioned ACT policy relies solely on **RGB observations** from two monocular cameras: *E22S (side view)* and *E12 (top view)*. While this setup has proven sufficient for static, position-generalized grasping tasks, more complex scenes and contact-sensitive operations demand richer and more informative sensory inputs.

Future iterations of the system will incorporate additional sensor modalities to enhance the robot's spatial awareness, motion estimation, and contact reasoning capabilities. Specifically, the following sensors are proposed:

- **Depth Cameras**: Devices such as the Intel RealSense D435i can provide high-resolution *depth maps* that enable the model to reason about 3D structure, occlusions, and object height. This is particularly valuable for scenes with clutter or partial visibility.
- **Inertial Measurement Units (IMU)**: IMUs can provide *acceleration and angular velocity* data, improving temporal awareness and motion prediction, especially during high-speed or dynamic behaviors.
- **Gripper Force/Torque Sensors**: These sensors enable *contact-aware control*, allowing the policy to detect grasp success, slippage, or excessive force, and adapt actions accordingly.

Integrating these modalities requires architectural extensions that can fuse asynchronous or heterogeneous sensor streams. This motivates future research into **multimodal policy fusion frameworks**, potentially based on cross-attention or hierarchical encoders, capable of unifying:

- *Vision* (RGB and Depth),
- *Proprioception* (joint angles, velocities),
- *Tactile feedback* (force/torque),
- *Motion priors* (from IMU or visual odometry),
- *Optional exteroceptive signals* (e.g., audio).

This sensor-rich architecture would significantly enhance the robot's ability to generate **fine-grained, context-sensitive motor behaviors**, enabling manipulation in unstructured or dynamic environments where visual feedback alone is insufficient.

Expanding the robot's perceptual field beyond RGB unlocks new classes of tasks:

- Contact-rich operations such as compliant insertion and tool use,
- Reactive grasp adaptation in response to slippage or external forces,
- Depth-aware path planning in cluttered scenes.

This lays the groundwork for deploying Transformer policies in more dynamic, high-fidelity robotics applications.

7.5.4 Policy Portability and Online Adaptation

One of the remaining limitations of current imitation learning systems is their restricted portability across different robotic platforms. Most policies are trained and deployed in a *platform-specific* manner, tightly coupled to the physical properties, control interfaces, and kinematic constraints of a single robot. This lack of transferability inhibits broader deployment and reusability across varied robotic embodiments.

To address this challenge, future work will investigate a range of **policy transfer techniques** designed to enable cross-platform generalization and robustness. These include:

- **Sim-to-real transfer**: Pretraining policies in simulation using large-scale synthetic datasets, followed by real-world finetuning. This approach leverages photorealistic rendering and domain randomization to cover a wide distribution of visual and physical configurations.
- **Cross-platform dataset distillation**: Aggregating demonstration data across different robotic arms, and learning shared embeddings or behavior manifolds that can generalize across morphologies.
- **Contrastive state-space alignment**: Aligning internal state representations from different robots using contrastive learning, enabling the same policy backbone to interpret and act across distinct sensorimotor domains.

In addition to improving cross-platform compatibility, another critical direction is to develop **online policy adaptation mechanisms**, which allow the deployed agent to continuously refine its behavior in the face of:

- Mechanical wear and drift over long-term deployment;
- Sensor degradation or calibration shifts;
- Changing task objectives or environmental configurations.

Promising methods include:

- **Self-supervised finetuning**, where the robot leverages consistency constraints (e.g., cycle-consistency, temporal smoothness) to improve its policy without external labels.
- **Gradient-free adaptation**, using black-box optimization or evolutionary strategies to adjust parameters in real time.
- **Preference-based learning**, where human-in-the-loop feedback provides sparse signals for ranking behaviors or correcting suboptimal actions.

Combining policy transfer with online learning yields **lifelong adaptable agents**, capable of surviving wear, change, and distributional shift—crucial for real-world long-term autonomy.

7.5.5 Toward Autonomy at Scale: Memory, Hierarchy, and Planning

While the current ACT model exhibits strong reactivity and robustness in short-horizon manipulation tasks, it remains limited in its ability to reason over extended temporal contexts or operate across multiple subtasks. The policy's decision-making is constrained to a fixed-length observation window, which restricts its capacity for planning, abstraction, and lifelong adaptation.

To move toward general-purpose, long-horizon autonomy, future work will investigate **hierarchical policy architectures** that integrate structured planning, temporal abstraction, and memory. The proposed approach will combine:

- **High-level symbolic planners** or **task graphs**, which define the logical decomposition of multi-stage objectives and subtask transitions.
- **Low-level visual-motor controllers**, such as ACT, responsible for executing primitive actions (e.g., grasp, align, place) with high temporal resolution and sensorimotor responsiveness.
- **Episodic memory modules**, capable of retaining and querying past observations and latent states, enabling temporal reasoning and cross-step coherence.

These components would be integrated into a unified, multi-timescale policy framework that supports:

- Flexible subtask switching based on scene changes or semantic goals;
- Memory-based reasoning over partially observable environments;
- Incremental learning from long-term deployment and episodic interaction.

Such an architecture aims to bridge the gap between short-term visuomotor imitation and scalable robotic cognition.

The transition from reactive control to **hierarchical, memory-driven planning** is key to achieving scalable autonomy—where robots can operate over long timescales, generalize across tasks, and learn from lived experience.

7.5.6 Summary

This study establishes a strong foundation for real-world Transformer-based visuomotor control using low-cost sensors and limited demonstration data. The proposed vision-conditioned ACT model demonstrates that temporally structured imitation learning—when paired with synchronized multimodal inputs—can produce robust, adaptive, and autonomous robotic behavior under real-world deployment constraints.

Looking forward, we envision this work evolving into a comprehensive research framework that integrates:

- **Larger and more diverse demonstration datasets**, capturing a broader range of objects, tasks, and environmental conditions;
- **Alternative learning strategies**, such as Diffusion Policy (DP) and Reinforcement Learning (RL), which may improve sample efficiency and support long-horizon planning;
- **Sensor-rich multimodal fusion pipelines**, incorporating depth, tactile, IMU, and audio data to enhance perceptual grounding and situational awareness;
- **Generalizable policy abstractions**, enabling cross-robot transfer, task compositionality, and language-conditioned instruction following.

Together, these directions point toward a future where robotic agents learn as humans do—*by watching, adapting, and acting meaningfully in the physical world.*

8. Conclusion and Final Remarks

8.1 Recap of Research Contributions

This research presents a systematic framework for low-cost dual-arm teleoperation and imitation learning, leveraging the **WOWROBO LeRobot Koch** robotic platform as the core experimental system. Through an iterative engineering-design and data-driven modeling process, the study

delivers a full-stack pipeline encompassing hardware configuration, multimodal data acquisition, policy training, and real-world deployment.

The key contributions are summarized as follows:

Hardware and System Integration A reliable *Leader–follower control architecture* was established using the LeRobot Koch robotic arms, enabling synchronized teleoperation over USB serial interfaces. The platform supports dual-arm coordination and allows intuitive human demonstrations of manipulation tasks.

Multimodal Data Collection Pipeline A synchronized data acquisition system was developed, combining two RGB cameras (`E12` for top view and `E22S` for side view) with 6-DoF joint states. The system ensured frame-level temporal alignment, consistent sampling at 30–60 Hz, and task-tagged metadata logging, supporting scalable data generation for imitation learning.

ACT Model Training with Real-World Data The project carried out two stages of policy learning using **Action Chunking with Transformers (ACT)**:

- In **Section 6**, a baseline ACT model was trained using *only joint-state information*, achieving high accuracy in single-position grasping tasks.
- In **Section 7**, a more advanced *vision-conditioned* ACT model was trained using real-world visual-state data, which demonstrated robust spatial generalization and closed-loop autonomy.

End-to-End Deployment and Behavioral Validation The trained policies were deployed on the physical robot, demonstrating real-time execution of grasping tasks in unseen configurations. Empirical observations showed *position-adaptive behavior*, *visual-guided regrasping attempts*, and *stable trajectory generation* across all evaluation episodes.

Full-Cycle Framework for Imitation Learning The project successfully implemented a reproducible and modular pipeline—from teleoperated demonstrations, Hugging Face-based dataset hosting, ACT training, to policy evaluation and visualization—bridging the gap between simulated learning pipelines and practical robotics deployment.

These contributions not only validate the feasibility of deploying Transformer-based visuomotor policies in low-cost real-world robotic platforms, but also lay the groundwork for future extensions in multi-stage manipulation and general-purpose autonomy.

8.2 Insights Gained

One of the central experimental axes of this research lies in the comparative study between two imitation learning paradigms: **joint-only ACT models** trained on proprioceptive states (Section 6), and **vision-conditioned ACT models** trained on synchronized image-state pairs (Section 7). The transition from joint-space imitation to multimodal visuomotor learning yielded several important insights into generalization, behavior richness, and policy adaptability.

Limitations of Joint-Only Policies The ACT model trained solely on joint states demonstrated strong performance on the fixed grasping poses included in the training dataset. It successfully captured the temporal evolution of actions and reproduced precise trajectories with low error rates. However, the model exhibited poor generalization when tested on unseen object placements or minor perturbations in the environment. This limitation stems from the model's inability to infer spatial relationships between the object and manipulator—it lacked any sensory grounding beyond proprioception.

In other words, while joint-only ACT served well for *memorizing specific motor patterns*, it failed to adapt when task variations required perceptual flexibility. This behavior was anticipated and confirmed during deployment tests, where the robot either missed the target object or attempted outdated trajectories when the environment changed.

Advantages of Vision-Conditioned ACT Policies In contrast, the vision-conditioned ACT model trained in Section 7 exhibited emergent visuospatial reasoning capabilities. By fusing ResNet-18 encoded RGB features with 6-DoF joint states, and embedding the combined observations into a temporal Transformer, the policy learned to adjust its actions based on real-time visual input. This fusion allowed the model to *generalize to new spatial configurations*, demonstrating behaviors such as:

- Modulating arm trajectories to reach shifted or rotated objects;
- Performing dynamic regrasping when the initial grasp failed;
- Maintaining consistent temporal coherence across episodes.

These findings validate the architectural design decision to adopt a **late fusion strategy**, in which image and joint representations are processed independently before fusion. This modular structure supports better scaling, interpretability, and allows plugging in more complex encoders in future iterations (e.g., ViT, multimodal transformers).

Criterion	**Joint-Only ACT**	**Vision-Conditioned ACT**
Input Modalities	Joint state only	RGB images + joint state
Training Dataset	40 clips (fixed poses)	15 episodes (varied poses)
Generalization Capability	Low	High (spatial adaptability)
Emergent Behaviors	None	Regrasping, path replanning
Deployment Success Rate	High (Only for targeted scenarios)	High (Suitable for generalized scenarios)
Future Scalability	Limited	Flexible and extendable

Table 12: Comparison between joint-only and vision-conditioned ACT policies.

Policy Comparison Summary This comparative study highlights a key insight: visual grounding is indispensable for robotic policies aimed at open-world deployment. While joint-only policies

suffice in structured, repetitive environments, vision-conditioned policies enable flexible adaptation, perceptual feedback, and emergent behaviors—paving the way for robust visuomotor autonomy. These findings reaffirm the promise of Transformer-based multimodal learning frameworks for scalable, real-world robotic manipulation.

8.3 Reflections on System Design and Deployment Challenges

The development of a vision-conditioned robotic learning platform—from low-cost dual-arm teleoperation to real-world policy deployment—entailed a series of intricate system-level challenges. These challenges extended beyond algorithmic implementation, encompassing hardware constraints, data integrity, real-time control, and software modularity. This section consolidates the critical engineering reflections accumulated throughout the project.

Hardware-Level Bottlenecks and Practical Workarounds A central constraint throughout system bring-up was the **USB-based communication between the dual LeRobot Koch arms** and the host PC. The use of two serial devices for leader-follower synchronization imposed limitations on bandwidth and timing accuracy. At times, serial port instability led to occasional packet drops or joint state jumps, particularly under high-frequency querying during data collection. This necessitated the integration of a *smoothed control filter* with an exponential decay factor (e.g., $\alpha = 0.3$) to mitigate abrupt joint updates, which significantly improved the stability of both recording and teleoperation performance.

Additionally, the physical design of the hardware (e.g., **non-fully recoverable wiring** and Type-C connections) introduced intermittent mechanical issues, requiring manual repositioning and error-skipping logic in the control scripts. These issues underscore the importance of robust hardware abstraction and real-time status verification in future iterations.

Synchronization Across Modalities Maintaining high-precision alignment between **RGB images** (from E22S and E12 cameras) and **joint state data** proved critical, especially given the tight temporal correlation required for effective Transformer-based learning. Custom multi-threaded recording scripts were developed to sample synchronized data at **30–60Hz**, with precise timestamping and naming conventions (e.g., `frame_00000_e22s.jpg`). Visual inspection tools and timestamp logs were extensively used to verify frame drift and dropped samples.

The final dataset structure followed the recommended LeRobot format, with automatic episode segmentation and naming under `results_v5/`, enabling downstream training pipelines to seamlessly interpret multimodal sequences. This alignment pipeline represents a *critical design milestone* toward enabling large-scale multimodal learning in resource-constrained platforms.

Dataset Management and Training Efficiency Given the relatively **small dataset size** (only 15 recorded episodes for vision-based ACT training), careful attention was paid to the *structure, quality, and variation* of each demonstration. Manual labeling, positioning of objects, and clip-level logging were essential in preventing overfitting. In contrast to larger simulation-based datasets, the **real-world training process was slower and more sensitive to noise**, placing high value on each data sample's integrity.

The modular nature of the LeRobot framework—particularly the record and replay controllers—greatly facilitated dataset reuse, *episodic evaluation*, and policy iteration. However, the integration of Hugging Face Hub APIs for dataset upload introduced additional authentication steps and occasional delays in versioning, suggesting that **local-first workflows** with optional cloud syncing may offer better productivity during early development.

Deployment Realities and Model Execution Model deployment on the real robot, especially during evaluation sequences in `results_train3` to `results_train5`, revealed both successes and limitations. While the ACT model showed strong spatial generalization and adaptive behavior, occasional failure modes—such as **overly smooth motions causing delayed grasps**, or **vision ambiguity in cluttered scenes**—highlighted the gap between learned priors and physical execution dynamics.

Crucially, these results emphasized the **need for closed-loop corrections** and dynamic feedback integration (e.g., from IMU, tactile sensors, or additional image encoders), which were not present in this version of the system. Nonetheless, the observed self-corrective attempts (e.g., regrasping behavior) provide *strong early indicators of emergent robustness*, even without explicit reward supervision.

Software Modularity as a Key Enabler The modularity and reproducibility of the software stack—built atop the LeRobot ecosystem—was pivotal in enabling flexible experimentation. Components such as `control_robot.py`, `record_dataset.py`, and `train.py` were reusable across both ACT training stages (joint-only and vision-based), and allowed for smooth switching between *teleoperate, record, replay, and deployment* modes with minimal code modification.

This modularity not only reduced development overhead, but also ensured clear separation between hardware abstraction, dataset logic, and policy training, embodying the principles of **clean experimental design** for embodied intelligence systems.

These reflections emphasize that the successful development of a learning-based robotic system hinges not only on algorithmic sophistication, but equally on the robustness of the hardware setup, precision of data logging, and the fluidity of the development workflow. The co-design of sensors, controllers, and learning pipelines—grounded in real-world constraints—proved vital in enabling the system to progress from raw data collection to autonomous visuomotor execution. The act of curating reliable multi-modal data streams under temporal, mechanical, and computational constraints remains a cornerstone challenge—and opportunity—in scalable embodied learning research

8.4 Broader Implications and Final Outlook

The outcomes of this research, while anchored in a specific dual-arm robotic system (LeRobot Koch) and a targeted grasping task, carry broader implications for the field of real-world robotic learning. By designing, implementing, and validating an end-to-end visuomotor imitation learning framework—from teleoperation to policy deployment—this work offers multiple insights into the principles and practices of scalable embodied intelligence.

From System Design to Learning Integration One of the core achievements lies in the seamless integration of hardware control, synchronized data acquisition, multimodal representation learning, and policy execution. Unlike many purely simulation-based pipelines, this study grounds its contributions in real-world robotic interactions, demonstrating that effective learning systems can be built and deployed even with modest sensing and compute budgets. This reinforces the principle that *hardware-software co-design*—rather than pure algorithmic advances alone—is central to the success of embodied intelligent Robot.

Multimodal Data as the Cornerstone of Generalization Through comparative analysis of joint-only and vision-conditioned ACT models, this research empirically illustrates how **multimodal observation (RGB + joint states)** improves generalization and robustness. The visual grounding enables policies to adapt to novel object configurations and correct for unmodeled disturbances, a capability that is critical for real-world autonomy. This suggests that *learning from vision is no longer optional*, but essential, in building flexible robotic systems.

Reproducibility and Modular Tooling By leveraging open-source frameworks such as the LeRobot SDK and Hugging Face Hub, and by developing a modular, reproducible codebase for control, recording, training, and evaluation, this project contributes to the growing ecosystem of **accessible robotic learning**. The architecture allows rapid prototyping and iterative improvement, facilitating deployment in education, research, and low-resource contexts.

Scalability Challenges and Research Opportunities The experiments conducted in this work also illuminate critical bottlenecks. Limited dataset diversity, lack of feedback loops, and restricted sensor modalities currently bound the scope of achievable behaviors. However, they also open up several exciting avenues:

- *Scaling data collection*: toward larger, task-diverse, and multi-object datasets, potentially involving automated resets or human-in-the-loop feedback.
- *Richer multimodal fusion*: including depth, tactile, IMU, or audio signals to capture complex environmental cues.
- *Learning architectures*: extending ACT with diffusion models, transformer encoders, or reinforcement fine-tuning to improve policy expressivity and resilience.
- *Hierarchical control and planning*: combining low-level visuomotor skills with high-level task decomposition and goal inference.

Toward General-Purpose, Deployable Robot Learning This project demonstrates that a real-world robot, trained only on offline teleoperation demonstrations, can perform autonomous tasks with spatial variation and behavioral flexibility. While the domain remains constrained, the core insight is profound: *data-driven visuomotor policies, when properly grounded and deployed, can exhibit emergent intelligence even without rewards or simulations*. This paves the way for future work toward **general-purpose, scalable, and adaptive robotic agents** capable of assisting in industrial, educational, and domestic environments.

In conclusion, this study affirms that the fusion of perception, action, and learning—realized through modular, affordable platforms—can serve as a viable paradigm for practical robotic autonomy. By scaling data, sensors, and architectures, and by deepening the integration between hardware constraints and learning objectives, we edge closer to the vision of robots that learn like humans: through observation, correction, and experience in the real world.

References

[1] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel, "GELLO: A General, Low-Cost, and Intuitive Teleoperation Framework for Robot Manipulators," *arXiv preprint arXiv:2309.13037*, vol. abs/2309.13037, 2023, version 2, submitted July 18, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2309.13037

[2] Z. Fu, T. Z. Zhao, and C. Finn, "Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation," *arXiv preprint arXiv:2401.02117*, vol. abs/2401.02117, 2024, version 1, submitted January 4, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.02117

[3] H. Kasaei and M. Kasaei, "VITAL: Visual Teleoperation to Enhance Robot Learning through Human-in-the-Loop Corrections," *arXiv preprint arXiv:2407.21244*, vol. abs/2407.21244, 2024, version 1, submitted July 30, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.21244

[4] S. Yang, M. Liu, Y. Qin, R. Ding, J. Li, X. Cheng, R. Yang, S. Yi, and X. Wang, "ACE: A Cross-Platform Visual-Exoskeletons System for Low-Cost Dexterous Teleoperation," *arXiv preprint arXiv:2408.11805*, vol. abs/2408.11805, 2024, version 1, submitted August 21, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2408.11805

[5] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," in *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4. ACM, 2018, pp. 1–14.

[6] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y. N. Wu, S.-C. Zhu, and H. Liu, "Llm^ 3: Large language model-based task and motion planning with motion failure reasoning," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 12 086–12 092.

[7] J. Wong, A. Tung, A. Kurenkov, A. Mandlekar, L. Fei-Fei, S. Savarese, and R. Martín-Martín, "Error-Aware Imitation Learning from Teleoperation Data for Mobile Manipulation," *arXiv preprint arXiv:2112.05251*, vol. abs/2112.05251, 2021, version 1, submitted December 9, 2021. Presented at CoRL 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2112.05251

[8] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware," *arXiv preprint arXiv:2304.13705*, vol. abs/2304.13705, 2023, version 1, submitted April 23, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.13705

[9] A. Paolillo and M. Saveriano, "Learning Stable Dynamical Systems for Visual Servoing," *arXiv preprint arXiv:2204.05681*, vol. abs/2204.05681, 2022, version 1, submitted April 12, 2022. Accepted at IEEE International Conference on Robotics and Automation (ICRA) 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2204.05681

[10] S. Auddy, A. Paolillo, J. Piater, and M. Saveriano, "Imitation Learning-based Direct Visual Servoing using the Large Projection Formulation," *arXiv preprint arXiv:2406.09120*, vol. abs/2406.09120, 2025, version 2, last revised March 9, 2025. To appear in *Robotics and Autonomous Systems*. First two authors contributed equally. [Online]. Available: https://doi.org/10.48550/arXiv.2406.09120

[11] H. Zhang, W. Lin, Y. Jiang, and C. Ye, "Depth-PC: A Visual Servo Framework Integrated with Cross-Modality Fusion for Sim2Real Transfer," *arXiv preprint arXiv:2411.17195*, vol. abs/2411.17195, 2024, version 1, submitted November 26, 2024. Research Institute of Intelligent Control and Systems, Harbin Institute of Technology. [Online]. Available: https://doi.org/10.48550/arXiv.2411.17195

[12] Y. Zhang, W. Chen, J. He *et al.*, "Visual imitation made easy," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[13] C. Chi, Z. Xu, and S. Song, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," *The International Journal of Robotics Research*, 2024, first published online October 11, 2024. [Online]. Available: https://doi.org/10.1177/02783649241273668

[14] A. Smith and M. K. III, "An Augmented Reality Interface for Teleoperating Robot Manipulators," *arXiv preprint arXiv:2409.18394*, vol. abs/2409.18394, 2025, version 2, last revised March 17, 2025. 8 pages, 6 figures. [Online]. Available: https://doi.org/10.48550/arXiv.2409.18394

[15] S. Dass, W. Ai, Y. Jiang, S. Singh, J. Hu, R. Zhang, P. Stone, B. Abbatematteo, and R. Martín-Martín, "TeleMoMa: A Modular and Versatile Teleoperation System for Mobile Manipulation," *arXiv preprint arXiv:2403.07869*, vol. abs/2403.07869, 2024, version 2, last revised March 21, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.07869

[16] D. Honerkamp, H. Mahesheka, J. O. von Hartz, T. Welschehold, and A. Valada, "Whole-Body Teleoperation for Mobile Manipulation at Zero Added Cost," *IEEE Robotics and Automation Letters*, vol. 10, no. 4, pp. 3198–3205, April 2025, also available as arXiv preprint arXiv:2409.15095, version 2. [Online]. Available: https://doi.org/10.48550/arXiv.2409.15095

[17] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, S. Savarese, and L. Fei-Fei, "RoboTurk: A Crowdsourcing Platform for Robotic Skill Learning through Imitation," *arXiv preprint arXiv:1811.02790*, vol. abs/1811.02790, 2018, published at the Conference on Robot Learning (CoRL) 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1811.02790

[18] HuggingFace, "Lerobot: Open-source dual-arm robot platform," https://github.com/huggingface/lerobot, 2024, accessed: 2025-04-09.

[19] Open X-Embodiment Collaboration, A. Padalkar, A. Pooley, A. Mandlekar *et al.*, "Open X-Embodiment: Robotic Learning Datasets and RT-X Models," *arXiv preprint arXiv:2310.08864*, vol. abs/2310.08864, 2024, version 8, last revised June 1, 2024.

Collaboration of 21 institutions, 22 robots, 527 skills, and 160k+ tasks. [Online]. Available: https://doi.org/10.48550/arXiv.2310.08864

[20] Y. Hao, R. Wang, Z. Cao, Z. Wang, Y. Cui, and D. Sadigh, "Masked Imitation Learning: Discovering Environment-Invariant Modalities in Multimodal Demonstrations," *arXiv preprint arXiv:2209.07682*, vol. abs/2209.07682, 2023, version 2, last revised March 5, 2023. 13 pages. [Online]. Available: https://doi.org/10.48550/arXiv.2209.07682

[21] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, "Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes," *arXiv preprint arXiv:2408.03539*, vol. abs/2408.03539, 2024, version 3, last revised September 16, 2024. Accepted to *Annual Review of Control, Robotics, and Autonomous Systems*. The first three authors contributed equally. [Online]. Available: https://doi.org/10.48550/arXiv.2408.03539

[22] H. He, P. Wu, C. Bai, H. Lai, L. Wang, L. Pan, X. Hu, and W. Zhang, "Bridging the Sim-to-Real Gap from the Information Bottleneck Perspective," *arXiv preprint arXiv:2305.18464*, vol. abs/2305.18464, 2024, version 2, last revised October 14, 2024. Accepted at Conference on Robot Learning (CoRL) 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2305.18464

[23] S. Khadka, S. Majumdar, T. Nassar, Z. Dwiel, E. Tumer, S. Miret, Y. Liu, and K. Tumer, "Collaborative Evolutionary Reinforcement Learning," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 3341–3350, also available as arXiv preprint arXiv:1905.00976. [Online]. Available: https://doi.org/10.48550/arXiv.1905.00976

[24] W. Wang, C. Zeng, Z. Lu, and C. Yang, "A novel robust imitation learning framework for dual-arm object-moving tasks," *IEEE Transactions on Industrial Electronics*, 2024.

[25] Q. Wang, R. McCarthy, D. C. Bulens, K. McGuinness, N. E. O'Connor, N. Gürtler, F. Widmaier, F. R. Sanchez, and S. J. Redmond, "Improving Behavioural Cloning with Positive Unlabeled Learning," *arXiv preprint arXiv:2301.11734*, vol. abs/2301.11734, 2023, version 2, last revised September 21, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2301.11734

[26] A. Chen, S. Malladi, L. H. Zhang, X. Chen, Q. Zhang, R. Ranganath, and K. Cho, "Preference Learning Algorithms Do Not Learn Preference Rankings," *arXiv preprint arXiv:2405.19534*, vol. abs/2405.19534, 2024, version 4, last revised October 31, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.19534

[27] H. Sikchi, C. Chuck, A. Zhang, and S. Niekum, "A Dual Approach to Imitation Learning from Observations with Offline Datasets," *arXiv preprint arXiv:2406.08805*, vol. abs/2406.08805, 2024, version 2, last revised September 19, 2024. Accepted at the 8th Conference on Robot Learning (CoRL 2024), Munich, Germany. [Online]. Available: https://doi.org/10.48550/arXiv.2406.08805

[28] Y. Zhu, A. Joshi, P. Stone, and Y. Zhu, "VIOLA: Imitation Learning for Vision-Based Manipulation with Object Proposal Priors," *arXiv preprint arXiv:2210.11339*, vol. abs/2210.11339, 2023, version 2, last revised March 8, 2023. Published at the 6th Conference on Robot Learning (CoRL 2022). [Online]. Available: https://doi.org/10.48550/arXiv.2210.11339

[29] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, and et al., "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances," *arXiv preprint arXiv:2204.01691*, vol. abs/2204.01691, 2022, version 2, last revised August 16, 2022. Includes PaLM results, ablations, multilingual prompting, and open-source tabletop version. [Online]. Available: https://doi.org/10.48550/arXiv.2204.01691

[30] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, and et al., "Flamingo: a Visual Language Model for Few-Shot Learning," in *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS)*, 2022, version 2, last revised November 15, 2022. 54 pages. [Online]. Available: https://doi.org/10.48550/arXiv.2204.14198

[31] X. Ren and H. Li, "Adaptive dynamic programming-based feature tracking control of visual servoing manipulators with unknown dynamics," *Complex & Intelligent Systems*, vol. 8, no. 1, pp. 255–269, 2022.

[32] Y. Zhu, A. Joshi, P. Stone, and Y. Zhu, "VIOLA: Imitation Learning for Vision-Based Manipulation with Object Proposal Priors," *arXiv preprint arXiv:2210.11339*, vol. abs/2210.11339, 2023, version 2, last revised March 8, 2023. Published at the 6th Conference on Robot Learning (CoRL 2022). [Online]. Available: https://doi.org/10.48550/arXiv.2210.11339

[33] N. Amiri, G. Wang, and F. Janabi-Sharifi, "Keypoint Detection Technique for Image-Based Visual Servoing of Manipulators," *arXiv preprint arXiv:2409.13668*, vol. abs/2409.13668, 2024, version 1, submitted September 20, 2024. Accepted for presentation at the IEEE International Conference on Automation Science and Engineering (CASE 2024). [Online]. Available: https://doi.org/10.48550/arXiv.2409.13668

[34] O.-M. Pedersen, E. Misimi, and F. Chaumette, "Grasping unknown objects by coupling deep reinforcement learning, generative adversarial networks, and visual servoing," in *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 5655–5662.

[35] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robotics and autonomous systems*, vol. 15, no. 1-2, pp. 25–46, 1995.

[36] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, "VIMA: General Robot Manipulation with Multimodal Prompts," *arXiv preprint arXiv:2210.03094*, vol. abs/2210.03094, 2023, version 2, last revised May 28, 2023. Accepted at ICML 2023 (Camera-ready version). [Online]. Available: https://doi.org/10.48550/arXiv.2210.03094

A. Appendix

A.1 All support materials

All support materials such as figures shows in the report, viedos, code and others, can all be found in the folder named `Lerobot`.

To be specific, basically the result in Section 6 can mainly refers to the folder named `/Lerobot/results`.

While the result in Section 7 can mainly refers to the folder named `/Lerobot/output`.

In additon, all the figures in the report can be found in folder named `/Lerobot/figures`.

And, all the videos in the report can be found in folder named `/Lerobot/videos`.

Lastly, the code can be found in a jupyter notebook named `/Code of lerobot`.

A.2 Video Demonstration of Robot arm synchronous control

To visually demonstrate the real-time master-follower teleoperation test conducted in Section 4.1.4, a short video has been recorded and included in the project submission archive.

- **File Name:** `demo of robot synchronous control.mp4`
- **Content:** The video shows the leader arm being manually manipulated, with the follower arm precisely replicating the motion in real time.
- **Duration:** Approximately 30 seconds
- **Relevance:** Demonstrates the successful execution of the core control loop described in Listing 21.

This demonstration serves as qualitative validation of system responsiveness and joint synchronization fidelity during Phase 1 testing.

A.3 Video Demonstration of data collection

The location of video demonstration of data collection in section 6 are shown below:

- `videos/Video Demonstration of data collection.mp4`

In addition the video demonstration of data collection in section 7 is almost same as this, thus can also refer to this.

A.4 Deployment Videos in Section 6

All four autonomous deployment trials are provided as accompanying video recordings. The videos are stored in the ‘videos/‘ folder relative to this report and correspond to the four tested positions:

- `videos/position_A_demo.mp4`
- `videos/position_B_demo.mp4`
- `videos/position_C_demo.mp4`
- `videos/position_D_demo.mp4`

Each video captures the full grasp-and-place motion from a third-person perspective, providing visual evidence of task completion, trajectory smoothness, and real-time autonomy under ACT policy control.

A.5 Deployment Videos in Section 7

To supplement the qualitative and quantitative evaluations discussed in Section 7, we provide four representative deployment videos below.

Spatial Generalization Capability The videos are stored in the ‘videos/‘ folder relative to this report

- `video/Spatial Generalization Capability.mp4`

Emergent Reactive Behaviors The videos are stored in the ‘videos/‘ folder relative to this report

- `video/ 1.Emergent Reactive Behaviors.mp4`
- `video/ 2.Emergent Reactive Behaviors.mp4`

Reliable End-to-End Control The videos are stored in the ‘videos/‘ folder relative to this report

- `video/ Reliable End-to-End Control.mp4`

These video results not only validate the practical deployment potential of vision-conditioned ACT models but also visually reinforce the specific behavioral properties critical for robust real-world manipulation.