

**Learning manipulation skills for the Franka FR3
using behavior cloning: from simulation to
real-robot deployment**

by

Aiym Issatayeva

Submitted to the Department of Robotics and Mechatronics
in partial fulfillment of the requirements for the degree of

Master of Science in Robotics

at the

NAZARBAYEV UNIVERSITY

May 2026

© Nazarbayev University 2026. All rights reserved.

Author
Department of Robotics and Mechatronics
May 12, 2026

Certified by
Almas Shintemirov
Associate Professor
Thesis Supervisor

Accepted by
Elizabeth Arkhangelsky
Dean, School of Engineering and Digital Sciences

Learning manipulation skills for the Franka FR3 using behavior cloning: from simulation to real-robot deployment

by

Aiym Issatayeva

Submitted to the Department of Robotics and Mechatronics
on May 12, 2026, in partial fulfillment of the
requirements for the degree of
Master of Science in Robotics

Abstract

This thesis presents a behavior cloning pipeline of robotic manipulation on the Franka FR3, including simulation validation, real-robot data collection, policy training, and the deployment on hardware. The superiority of temporally-aware architectures is validated by a simulation study of BC-RNN and Diffusion Policy on the Robomimic benchmark, where Diffusion Policy has 98-100% success in Lift and Pick-PlaceCan tasks compared to 78-92% in BC-RNN. Action Chunking with Transformers (ACT) is chosen to be deployed on real robots due to its capability to work with low-data regimes and single-pass inference. Three demonstration datasets were gathered through joystick teleoperation through an iterative process with each dataset being redesigned by the lessons of the previous deployment. These datasets were trained on four ACT policies with the LeRobot framework in 100,000 steps, and the training loss dropped to 0.038. The policies were executed on the physical robot through a ROS 2 network through CRISP controllers and CycloneDDS, and nine software integration issues had to be solved. The system improved in four deployment cycles, starting with no grasp attempts generated in the first deployment, and ending with the full pick-and-place task structure - approach, grasp attempt, lift, transport, and placement - generated in the last deployment. The main gap was the precision of spatial localization during the grasp phase, and reliable task completion was not achieved. This thesis not only provides a completely documented iterative pipeline but also pinpoints camera location, demonstration action allocation, and deployment time synchronization as the most important practical considerations in real-robot BC performance.

Thesis Supervisor: Almas Shintemirov
Title: Associate Professor

Acknowledgments

I would like to say that I would have not been able to work on this project without my supervisor - professor Almas Shintemirov's guidance, support, and feedback.

I am also grateful to the Department of Robotics and Mechatronics at Nazarbayev University for access to the Franka FR3 hardware and computing resources that I used in the given work.

Contents

1	Introduction	10
2	Literature Review	13
2.1	Behavior Cloning and Learning from Demonstration	13
2.2	Temporal policy architectures: BC-RNN, Diffusion Policy, and ACT .	15
2.3	BC for Robotic Manipulation with the Franka Platform and ROS 2 .	17
2.4	Simulation benchmarks for manipulation learning	18
2.5	Knowledge Gap and Motivation	19
3	Simulation experiments	21
3.1	Overview	21
3.2	Experimental Setup	21
3.2.1	Benchmark and datasets	21
3.2.2	Algorithms	22
3.3	BC-RNN results	22
3.4	Diffusion Policy results	24
3.5	Comparative analysis	25
4	Real-robot system	27
4.1	Overview	27
4.2	Hardware	27
4.2.1	Robot	27
4.2.2	Perception	28

4.2.3	Compute	28
4.3	Software	29
4.3.1	Middleware: ROS 2 Humble and CycloneDDS	29
4.3.2	Low-Level Control: CRISP	29
4.3.3	Learning framework: LeRobot 0.2.0	30
4.4	System architecture	30
4.5	Fake hardware validation	31
5	Demonstration data collection	33
5.1	Overview	33
5.2	Teleoperation setup	33
5.3	Task description	34
5.4	Observation and action space	35
5.5	Dataset statistics	36
5.5.1	Dataset 1	36
5.5.2	Dataset 2	36
5.5.3	Dataset 3	37
6	Policy training	39
6.1	Algorithm selection	39
6.2	Architecture	39
6.3	Training configuration	41
6.4	Training dynamics	42
7	Deployment and results	44
7.1	Overview	44
7.2	Deployment Pipeline	44
7.3	Engineering challenges	46
7.4	Deployment results	48
7.4.1	Run v1 — Dataset 1, both cameras	48
7.4.2	Run v2 — Dataset 2, both cameras	48

7.4.3	Run v3 — Dataset 2, stationary camera only	49
7.4.4	Run v4 — Dataset 3, stationary camera only	49
7.5	Analysis	51
8	Discussion	53
8.1	Successes	53
8.2	Failures	53
8.3	Key findings	54
8.4	Limitations	55
9	Conclusion and Future work	57
9.1	Conclusion	57
9.2	Future work	58
A	Teleoperation Script	60
B	Gripper relay node	71
C	RMW override configuration	73
D	Training configuration	74
D.1	Training command	74
D.2	Model configuration	74
E	Deployment configuration	76
E.1	Environment Configuration Files	76
E.2	Deployment command	79
F	System specifications	80

List of Figures

3-1	Training loss curves for Diffusion Policy on the Lift (dark) and Can (cyan) tasks over 173 epochs.	24
3-2	Rollout success rate for Diffusion Policy on the Lift (dark) and Can (cyan) tasks evaluated between epochs 50 and 150.	25
4-1	Two-machine system architecture. All Cartesian poses are expressed in the robot base frame, which serves as the world frame for this system.	31
4-2	RViz visualization during fake hardware testing.	32
5-1	Teleoperation setup for demonstration data collection. All end-effector motions are commanded in the robot base frame (world frame), with the z-axis pointing vertically upward.	34
5-2	Teleoperation sequence for the pick-and-place task using the joystick controller.	35
5-3	Dual-camera observation streams from a single demonstration episode shown in the CRISP_GYM data directory.	36
5-4	Episode length distributions across the three demonstration datasets.	37
6-1	The ACT policy architecture. Reproduced from [1].	41
6-2	ACT training loss for all four runs on a logarithmic scale.	43

7-1	Deployment sequence of the v4 ACT policy on the Franka FR3, extracted from the most successful trial. The robot moves from its home position (a), approaches the workspace (b), and closes the gripper at approximately the correct height (c). It then lifts upward (d), moves laterally to the target location (e), and opens the gripper as if placing the object (f). All motions are expressed in the robot base frame, with lateral transport occurring along the y-axis and the lift motion along the z-axis.	50
-----	--	----

List of Tables

2.1	Comparison of temporal policy architectures evaluated in this work. .	17
3.1	BC-RNN checkpoint evaluation results on the Lift and Can tasks. Each checkpoint was evaluated with 50 rollout episodes at a maximum horizon of 400 steps.	23
3.2	Diffusion Policy robustness evaluation on the Can task (epoch 150 checkpoint). Each seed uses 50 rollout episodes with horizon 400. . .	26
5.1	Demonstration dataset statistics. All datasets used the same task (pick-and-place), proprioceptive state space (20 dimensions), action space (7 dimensions), and storage format (LeRobot MP4 + Parquet).	38
6.1	ACT training run summary. All runs used 100,000 steps, AdamW optimizer, learning rate 1×10^{-5} , batch size 8, and the final checkpoint at step 100,000 for deployment.	42
F.1	Hardware and software specifications of the two-machine system. . . .	80

Chapter 1

Introduction

Robotic manipulation is the ability of autonomous systems to manipulate objects in the unstructured environment, and to reposition them, but also is one of the key open problems of robotics. Classical methods such as using hand-programmed controllers need a lot of domain knowledge and do not transfer well in task variations, which encourages the creation of data-driven approaches. Learning from Demonstration (LfD) provides an interesting paradigm: a robot does not need to be told the logic of control in particular; by watching human examples, it learns how to repeat desired behavior [2]. Behavior Cloning (BC), the most basic example of LfD, presents this as supervised learning - direct mapping between observed states directly into actions, and has shown to be practically viable in a variety of manipulation tasks [3].

Although being relatively straightforward conceptually, in practice BC has its issues. Compounding errors happen when the policy trained on one distribution and applied to different states ends up diverging from the observed behaviors, due to the errors that build up [4]. Temporally extended tasks are another area that standard architecture BC systems have difficulty in, as one observation is not enough to ascertain the appropriate action. More recent studies have resolved these limitations by applying temporal modeling: diffusion-based policies with multi-step action sequences are learned to be denoised [5], and specific transformer-based methods such as Action Chunking with Transformers (ACT) [1] predict action chunks and can be successfully applied to achieve smoother and more natural movement on real hardware. Another

issue here is the sim-to-real gap: the policies that are tested in simulation do not always perform well when applied to physical robots as they will experience other sensor-noise, contact dynamics, and actuation latency [6]. The effort to construct and validate a full pipeline, from demonstration collection to deployment on a real robot, including the engineering infrastructure needed to support synchronous perception, policy inference, and low-level control provided by Robot Operating System 2 [7] is in itself a non-trivial systems contribution.

This thesis talks about the algorithmic and systems parts of the issue. It uses a controlled simulation study with the benchmark to compare BC-RNN and Diffusion Policy on Lift and Can tasks. The study uses low-dimensional observations. BC-RNN gets a success rate of 92% on Lift and 78% on Can while Diffusion Policy gets 98-100% on both tasks with lower variance and faster convergence. The difference in performance is biggest on Can where temporal modeling matters most. These findings help choose ACT for real-robot experiments. ACT is chosen because it works with low-data regimes and has efficient single-pass inference. Three demonstration datasets were collected using joystick teleoperation on the Franka FR3, and each of them uses lessons from the deployment run. Four ACT policies were trained across these datasets and deployed on the physical robot via a ROS 2 network. The robot goes from making no grasp attempts in the run to doing the complete pick-and-place task in the final run.

This thesis makes the following contributions:

1. It compares BC-RNN and Diffusion Policy on the benchmark. This comparison justifies using temporally-aware policy architectures for real-robot manipulation.
2. It documents a real-robot pipeline for the Franka FR3. The pipeline uses RealSense cameras, joystick teleoperation, ACT policy training and ROS 2 deployment with CRISP. It solves nine non-trivial software integration challenges.
3. It records the deployment across four training and deployment cycles, showing how the robot goes from failure to successful execution of the full pick-and-place

task. It identifies failure modes and design changes between each cycle.

4. It finds three things that are useful for future real-robot BC work. These are the importance of camera placement the effect of demonstration action distribution and the need for timing alignment, between data collection and deployment.

This thesis addresses three research questions. First, what is the performance of temporally-aware behavior cloning architectures compared to the standard recurrent BC on standardized manipulation benchmarks in the same conditions? Second, is it possible to deploy ACT on the Franka FR3 with accessible tooling and a small demonstration dataset of less than 200 episodes? Third, what are the main practical considerations to impact the real-robot visuomotor policy performance when implementing a learned BC policy on physical hardware?

The rest of this thesis is arranged in the following way. Chapter 2 is a review of the related work on behavior cloning, temporal policy architectures, and previous applications to the Franka platform. In Chapter 3, the experiments on Robomimic simulation are shown. The integration of the hardware and software system is explained in Chapter 4. The process of data collection is described in Chapter 5. Chapter 6 covers ACT training. In Chapter 7, four deployment runs on the real robot are reported. Chapters 8 and 9.2 deal with findings, limitations and future directions. Chapter 9.1 concludes.

Chapter 2

Literature Review

2.1 Behavior Cloning and Learning from Demonstration

Learning by Demonstration (LfD) refers to the techniques by which robots learn to perform task-relevant tasks by watching expert demonstrations, eliminating the need for hand-designed controllers or even task-specific reward functions [2]. The most explicit implementation of this paradigm is Behavior Cloning (BC), a supervised learning, where a policy is trained to predict actions based on demonstration data as ground truth [3]. BC is successfully implemented in the tasks of manipulating objects, force-based manipulation tasks, and collaborative tasks between humans and robots [8, 9, 10].

BC has well documented theoretical and practical drawbacks despite its conceptual simplicity. The inherent issue here is covariate shift: a BC policy is conditioned on states that the expert demonstrator visited, but at test time its own behavior causes it to shift to a state that was not observed in training. As there is no regulation of these out-of-distribution states in the policy, even minor mistakes amplify over time and the robot will move farther from the demonstrated path [4]. This issue increases with the task horizon - the longer the manipulation sequence the more chances there are to develop little mistakes that eventually add up to failure. DAgger

(Dataset Aggregation) was suggested as an ideologically sound solution, expanded gradually with the states the trained policy has explored and asking the expert to label them correctly [4]. Though efficient, DAgger has to have continuous professional communication during the training process, which is costly and troublesome in the case of physical robot systems.

The second inherent weakness of conventional BC is its inability to work with multimodal action distributions. A feedforward policy using mean squared error loss when the same observation can rightly be associated with a variety of valid actions (such as holding a graspable object, where an action can be approached in either direction) will average modes to give an action that does not correspond to any valid behavior. It is especially an issue in contact-rich manipulation where the direction in which the choice of approach is made has a great influence on the success of the other steps. Mixture Density Networks and Gaussian Mixture Model output heads, which solve this by modeling the distribution of actions instead of a prediction, but requiring the number of modes to be specified in advance and have difficulty with high-dimensional action spaces [11]. An alternative approach to providing multimodal distributions with greater flexibility is implicit behavioral cloning that views policy learning as energy minimization over an action-conditioned energy functional [12] [13].

Lastly, although BC has been performing well in simulation, policy transfer to real hardware leads to new issues due to sensor noise, actuation latency, and the deviation between simulated and real contact dynamics, a phenomenon also referred to as the sim-to-real gap [6]. The problem of generalization is also still inherent: the standard policies of BC, which have been trained on curated data, are also prone to failure when the space of objects, lighting conditions, or workspace layout are not similar to the training space [14].

2.2 Temporal policy architectures: BC-RNN, Diffusion Policy, and ACT

Much recent research has focused on the issue of BC having temporal constraints by considering architectures that explicitly represent action sequences or time dependencies over observations instead of prediction of single actions. These techniques are driven by a fact that a lot of manipulation problems are non-Markovian in nature, meaning what is the right action at a given point in time is not necessarily just related to the current observation, but also to the history of the past states and actions. It is a common situation that a single-frame observation is ambiguous, especially when the visual occlusion or contact force is involved.

BC-RNN is an extension of standard behavioral cloning, in which a recurrent neural network is added, and the policy takes into account a hidden state over the sequence of timesteps and conditions actions based on the history of observed states, and not on a single frame. This compact memory is a small block of the last context in which the policy can correct ambiguities that would not be evident in a single observation. With a Gaussian Mixture Model output head (BC-RNN-GMM) added, the policy is capable of modeling multimodal action distributions at every step, and it does better on tasks with contact variability [11]. BC-RNN is a powerful and popular baseline in the benchmark of manipulation, which provides a useful compromise between the simplicity of feedforward BC and the expressiveness of generative models. Its major disadvantage is that the recurrent hidden state needs to encode the whole history of tasks in a fixed size vector, which may become a bottleneck to the tasks with long dependencies.

Diffusion Policy is a more expressive model, where policy learning takes the form of a conditional denoising diffusion process [5], which is a direct extension of the Denoising Diffusion Probabilistic Model (DDPM) of Ho et al. [15]. The policy does not make predictions based on a single action or a combination of Gaussians, but learns to universally denoise a sequence of actions, conditioned on current observations, with a UNet backbone. The forward process gradually introduces Gaussian

noise to the action sequences that have been demonstrated, and the reverse process, which is the learned policy, eliminates the noise step by step under condition of the current observation. The generative formulation also has a number of benefits: it automatically supports multimodal action distributions without the need to specify the modes, it is stable to train, and it can learn complex, non-Gaussian action distributions that occur in dexterous manipulation [16]. Such fundamental architectural decisions as receding horizon control (the policy anticipates a window of future behavior and acts only on some of it before re-planning), and a temporal UNet (the dependencies across the prediction horizon are modeled and run to the end) can be made. The main implementation drawback of Diffusion Policy is inference latency: the denoising procedure in the form of 100 denoising steps through the network can be incurred in real-time and needs a lot of computation which can be limited by hardware with limited parts [5].

Action Chunking with Transformers (ACT) is a focusing strategy on the temporal issue [Zhao et al., 2023]. Instead of iterative denoising, ACT can predict whole blocks of future actions in a sequence with a single forward step based on a transformer-based architecture [17] that is trained as a Conditional Variational Autoencoder (CVAE). A temporal ensemble method averages overlapping chunk predictions to smooth execution. Initially described on fine-grained bimanual manipulation problems with as few as 50 demonstrations per problem, ACT was created expressly in the low-data regime of real-robot control, which is effectively applicable in low-data settings [1]. The fact that it makes single pass, which is several times faster than diffusion-based models at deployment time, is also a significant practical consideration when using it in real-time robot control. A comparison of the main characteristics of these three architectures as analyzed in this paper is summarized in Table 2.1.

Table 2.1: Comparison of temporal policy architectures evaluated in this work.

Property	BC-RNN	Diffusion Policy	ACT
Temporal modeling	Recurrent state	Sequence denoising	Action chunking
Action output	Single step (GMM)	Full sequence	Full chunk
Inference speed	Fast	Slow (iterative)	Fast (single pass)
Multimodality	Partial (GMM)	Yes (generative)	Yes (CVAE)
Low-data regime	Moderate	Moderate	Strong
Real robot	Yes	Limited	Yes
Key reference	[11]	[5]	[1]

2.3 BC for Robotic Manipulation with the Franka Platform and ROS 2

Manipulation research has now adopted the Franka Emika Panda and its successor the Franka FR3 as a platform due to their high precision, built-in joint torque monitoring, and well-supported software interfaces [18]. By contrast to most industrial manipulators, the Franka platform is built upon compliance and force control, not just position control, making it safer by default when using physical contacts and easier to apply to work where the robot needs to engage with the object, as opposed to moving freely through the air.

Some of the publications have utilized the Franka platform as a learning-based manipulation test-bed. The demonstration-learned variable impedance control has also been shown on tasks involving contacts, where the robot can adjust its stiffness profile depending on the task phase [19]. Special nonlinear parameter identification frameworks have been formulated to the Franka Panda and more precise dynamic models can be used to control [20]. Kinesthetic teaching, which is a physical guidance of the robot for demonstrations has been explored as a type of data collection on Franka hardware, and incremental policy refinement algorithms demonstrate that

policy iterative correction can significantly improve performance when comparing to a single-round demonstration collection process [10]. As a whole, these works prove that compliance and torque sensing at the Franka platform are not only learning-based approach resources but also hardware conveniences.

The middleware infrastructure behind the majority of current real-robot learning pipelines is offered by Robot Operating System 2 (ROS 2), which supports modular sensor integration, real-time control, and structured data logging [7]. In the case of learning-based manipulation, in particular, CRISP framework (Compliant ROS 2 Controllers for Learning-Based Manipulation Policies and Teleoperation) builds on top of ROS 2 with real-time C++ controllers that can be used to deploy policies and to collect teleoperation data (joysticks) [21]. CRISP offers Cartesian impedance control and teleoperation interfaces, which make the process of demonstration collection with compliant manipulators like the FR3 simpler, and its CRISP_GYM sub-component is a Gymnasium-compatible environment wrapper that captures demonstration data as LeRobot format directly. Hugging Face created LeRobot, an open source, built-in system to train and evaluate state-of-the-art imitation learning policies, such as ACT, Diffusion Policy, and VQ-BeT, using standardized dataset formats and deployment tools [22]. CRISP and LeRobot are the two practical software components that are the foundation of the real-robot pipeline of the present thesis.

2.4 Simulation benchmarks for manipulation learning

Simulation environments are significant in the testing of learning algorithms prior to the implementation into a real-robot to secure controlled conditions, repeatable assessment and cost-effective testing. Robosuite is an open-source modular robot learning simulation platform based on the MuJoCo physics engine, which offers benchmark environments and programmatic APIs to create new tasks in manipulation using diverse robot models and objects [23]. Robomimic is an extension of Robosuite which

offers a universal benchmark on imitation learning using human demonstrations on standardized datasets operating through teleoperation across various tasks of different degrees of complexity [11]. One of the most important contributions of Robomimic is that it is the first to systematically study what is important in imitation learning such as dataset quality, observation modality, and algorithm choice and has become a standard benchmark in algorithm comparison in manipulation learning. It allows the systematic assessment of algorithms such as BC, BC-RNN, and diffusion-based approaches to the evaluation of algorithms, being the logical selection when conducting the simulation study within the framework of this thesis.

2.5 Knowledge Gap and Motivation

The literature review indicates some of the gaps that persist and form the reason behind this work. Although temporally-aware policies like Diffusion Policy and ACT have been shown to have very strong individual performance, direct empirical comparisons on standard benchmarks, specifically using BC-RNN versus Diffusion Policy in the same low-dimensional observation regime, are limited in the literature. The Robomimic benchmark offers the framework of such comparison, though published evaluations have not fully recorded the performance difference between the recurrent BC and diffusion-based approaches to the tasks of different complexity. This gap is significant in understanding the choices that researchers make in selecting algorithms to dig in real-robot systems.

Though ACT was demonstrated to be validated on bimanual low-cost hardware, its use on the Franka FR3 in a ROS 2 pipeline on modern open-source tools - namely, CRISP and LeRobot - had not been reported in the literature. The issues of integration that arise when these frameworks are combined on actual hardware are non-trivial and difficult to integrate, yet do not exist in the literature. This is a practical gap: researchers who want to recreate or expand this type of system have no documented reference implementation to start with.

Lastly, the majority of published real-robots BC rely on large demonstration

datasets, either in the hundreds or thousands of episodes, or do not cover the entire engineering pipeline, and thus there is a lack of insight into what these systems can do with limited data and available hardware. This thesis fills all three gaps through the empirical comparison of the simulations and a complete documented real-robot pipeline and practical evidence of the performance of ACT in a low-data environment.

Chapter 3

Simulation experiments

3.1 Overview

Before the deployment of real-robots, a controlled simulation experiment was performed based on the Robosuite simulation framework [23] using the Robomimic benchmark [11]. The experiment has two purposes: to provide quantitative baselines in BC-based manipulation learning and to give empirical reasons as to why a temporally-aware policy architecture was selected to be the focus of future experiments with real-robots. Two algorithms were compared -BC-RNN and Diffusion Policy, on two tasks of a progressing complexity of manipulation.

3.2 Experimental Setup

3.2.1 Benchmark and datasets

Experiments were all based on the Robomimic framework with pre-recorded expert demonstration datasets. Two tasks were evaluated. The Lift task asks the robot to pick up a small cube, and lift it above the height standard - a typical sanity check of the imitation learning pipelines with fairly simplistic contact dynamics. The PickPlaceCan is a task where the robot has to pick up a can and place it into a target bin with a higher precision in space and more complicated interaction with objects.

These two tasks utilized the low-dimensional observation setup (`low_dim_v15`) which offered robot joint states, end-effector pose, gripper state and object state in the absence of visual inputs. This option makes computation time less complex and keeps perception and temporal consistency and quality of control uncoupled. Every dataset comprises successful demonstrations of experts through teleoperation.

3.2.2 Algorithms

The default training of the Robomimic default configuration with a GMM output head (BC-RNN-GMM) was used to train BC-RNN. Both tasks were trained during around 300 epochs. The policy checkpoints were checked after every 50 epochs, and 50 rollout episodes were used to evaluate each checkpoint with a maximum horizon of 400 steps. The performance metrics that were registered on a checkpoint were success rate, return and average episode horizon.

A UNet backbone with a noise process of DDPM was used to train diffusion policy [15]. The most important parameters were: sequence length of 16, observation horizon of 2, action horizon of 8, prediction horizon of 16 and 100 diffusion timesteps of training and inference. The maximum epochs of training were 2000, but good performance was achieved much earlier. A series of 50 rollouts were considered on the best diffusion policy checkpoint on Can task to measure robustness by evaluating the model over five random seeds.

3.3 BC-RNN results

BC-RNN converged slowly, and the convergence was steady on both problems. The results of the checkpoints in Lift task and Can task are summarised in Table 3.1.

Table 3.1: BC-RNN checkpoint evaluation results on the Lift and Can tasks. Each checkpoint was evaluated with 50 rollout episodes at a maximum horizon of 400 steps.

Task	Epoch	Success rate	Num success	Avg. horizon	Notes
Lift	50	0.60	30/50	200.9	Early model
Lift	100	0.92	46/50	76.5	Strong improvement
Lift	150	0.82	41/50	117.9	Slight drop, likely noise
Lift	200	0.92	46/50	—	Best checkpoint
Can	100	0.04	2/50	389.0	Fails almost entirely
Can	150	0.16	8/50	355.9	Learning begins
Can	200	0.20	10/50	342.5	Slow improvement
Can	250	0.76	39/50	197.0	Sharp improvement
Can	300	0.78	39/50	179.7	Best checkpoint

During the Lift task, the improvement in performance was proportional to training time. The highest epoch 200 checkpoint had a success rate of 0.92 (46/50 rollouts). The decrease in horizon between 200.9 in epoch 50 and 76.5 in epoch 100 is an indication that the policy became more efficient over time to accomplish the task. The period between the performance level of epochs 100 and 200 was a plateau, and nothing significant was made after this level.

Can task was considerably more difficult. Success rate did not become significant until epoch 200, and there was a steep increase in performance between epochs 200 and 250 - the success rate increased markedly during one evaluation period, 0.20 to 0.76. This sudden change indicates that the policy has exceeded a competence level. The highest checkpoint at epoch 300 was 0.78 successful (39/50 rollouts). The average horizon at initial epochs (389 steps) was large and constant, showing that the policy

never succeeded in locating the object, whereas the decrease of the same to 179.7 steps at epoch 300 proves that the policy was able to locate the object once it became competent.

The difference between Lift and Can shows that one of the main limitations of BC-RNN is that the performance of the algorithm is highly task-dependent. BC-RNN reliably converges to high success on a simple task, but on a task of a higher spatial resolution and more intricate object interaction, converges slowly and to a lower ceiling.

3.4 Diffusion Policy results

Diffusion policy reached a convergence significantly quicker than BC-RNN in both tasks and reached higher maximum performance. The training loss curves in both tasks are shown on Figure 3-1. Both runs slowly reduce at the beginning of the training (around 0.12 at epoch 1) to around 0.03 at the end of the training, without any instabilities or divergence, which proves the existence of steady optimization. The Lift task (dark curve) reaches convergence with a slightly higher rate compared to Can (cyan curve), as it is expected by the relative difficulty of the two tasks.

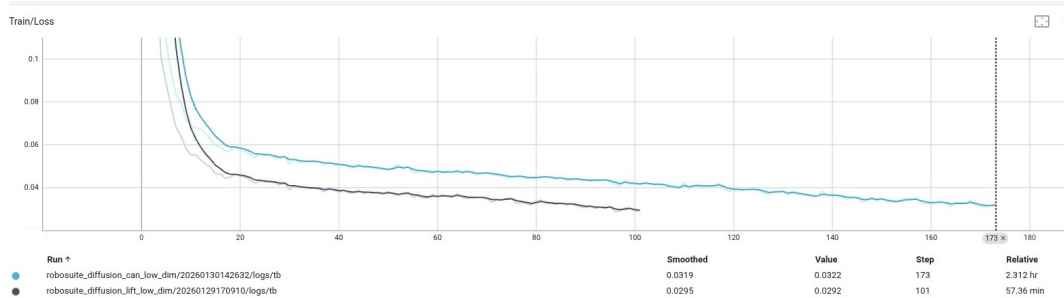


Figure 3-1: Training loss curves for Diffusion Policy on the Lift (dark) and Can (cyan) tasks over 173 epochs.

Figure 3-2 lists the rate of rollout success assessed through training. The Lift task (dark curve) achieves almost perfect performance early - the curve is virtually flat at 0.98 after approximately epoch 100 and onwards - the curve is very stable, showing

rapid convergence. The curve of the Can (cyan) has a continuously increasing curve, peaking at epoch 150 with a value of 0.96. The difference between the two curves can be directly explained by the difference in the complexity of the tasks: Lift can be solved in a short amount of time and Can needs to be trained over a variety of epochs before it can be applied to the maximum efficiency.

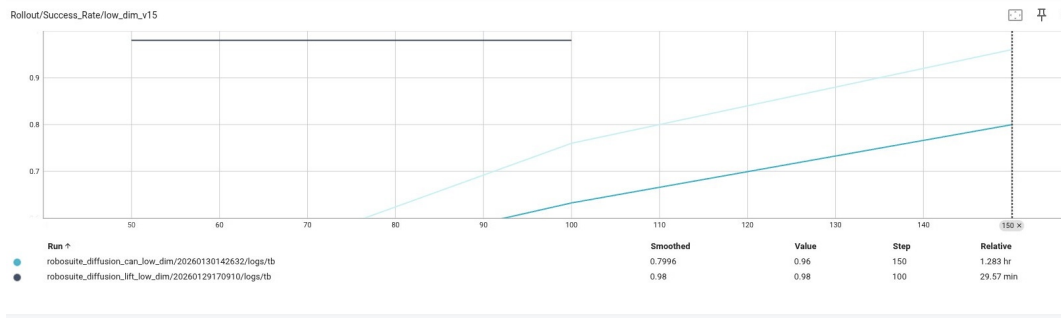


Figure 3-2: Rollout success rate for Diffusion Policy on the Lift (dark) and Can (cyan) tasks evaluated between epochs 50 and 150.

In order to test the level of robustness on the more demanding Can task, the epoch 150 checkpoint was tested on five random seeds. Table 3.2 presents the results. The average success rate of 0.976 ± 0.016 amongst five seeds shows that the outcome is consistent and is not seed-specific. Even the worst-case performance of 0.96 is significantly higher than the best of 0.78 on the same task of the BC-RNN.

3.5 Comparative analysis

Diffusion policy performs better on both tasks - by 6-8% on Lift and 18-22% on Can. The difference in performance is the greatest on the more challenging Can task on which the time constraints of the BC-RNN are the most revealed. In addition to raw performance, diffusion policy trains to near-optimal performance in less epochs, achieving convergence in both tasks at epoch 150 when compared to BC-RNN, which achieves this to 300 epochs and attains less optimal performance. Diffusion Policy also has significantly lower variance, with the seed evaluation in Table 3.2 indicating that, though BC-RNN was sensitive to training duration, and rollout instability.

Table 3.2: Diffusion Policy robustness evaluation on the Can task (epoch 150 checkpoint). Each seed uses 50 rollout episodes with horizon 400.

Seed	Success rate	Successful rollouts	Avg. horizon
0	0.98	49/50	116.9
1	0.98	49/50	118.1
2	0.96	48/50	121.6
3	0.96	48/50	122.9
4	1.00	50/50	113.0
Mean	0.976	48.8/50	118.5
Std	0.016	—	—

The increased performance disparity on Can than on Lift is in line with the theoretical insights: tasks with greater temporal coordination reveals the weaknesses of BC-RNN’s single-step GMM prediction whereas sequence-level denoising of Diffusion Policy shows no major performance differences between the two levels of difficulty.

These findings confirm two points. First, the imitation learning paradigm can be used to the manipulation tasks in question, as even BC-RNN reaches success rates of 92 on Lift, which proves that the quality of demonstration data and pipeline used to train the network is acceptable. Second, there is a transparent and consistent benefit to temporally-aware policy architecture over simple recurrent BC as the complexity of the tasks correspondingly escalates. This is the reason why these results directly inspired the choice of ACT [1], a temporal policy implemented on transformers, to be deployed in real-robots as the focus of the following chapters.

Chapter 4

Real-robot system

4.1 Overview

The experiments with real-robot were done on a Franka FR3 manipulator in a two-machine networked architecture. The system is a combination of low level compliant control, camera based perception, teleoperation to collect data, and policy inference based on GPU. This chapter outlines the hardware components, software stack, network configuration and the validation process, which was followed to ensure the pipeline is validated prior to running on physical hardware.

4.2 Hardware

4.2.1 Robot

The Franka FR3 is a 7-degree-of-freedom torque-controlled robot with inbuilt joint torque detection and a maximum load of 3 kg. It is the direct successor of the Franka Emika Panda and shares the same compliance control features that give the platform excellent aptitude to contact-rich manipulation tasks[18]. The FR3 provides an interface to a joint torque via libfranka, the hardware abstraction layer of the CRISP controllers as detailed in Section 4.3.2.

4.2.2 Perception

Visual observations are also provided to two Intel RealSense depth cameras:

1. Intel RealSense D435 - attached to the robot wrist, which gives a first-person perspective of the end-effector and the workspace right beneath the end-effector. The close-range object interactions and the gripper state are recorded in the wrist-mounted perspective.
2. Intel RealSense D455 - attached to a stationary location that gives access to a third-person perspective of the entire workspace. The larger field of view of the D455 renders it highly appropriate to the global scene comprehension as well as the object positioning.

The cameras both transmitted RGB images at 15 FPS and a resolution of 256x256 pixels. The two complementary perspectives, which are an egocentric approach and a global one, are a longstanding approach in the visuomotor policy learning of eliminating perceptual uncertainty in the manipulation process[1].

4.2.3 Compute

The system involved two workstations that were linked together with a wired Ethernet network:

1. Robot PC - operating Ubuntu 22.04 with ROS 2 Humble, which provides real-time low-level control of the FR3 through the CRISP controllers which actually run in a Docker container at 1 kHz. This machine also has CRISP_GYM which was used to collect demonstration data. This machine and the robot hardware used are connected directly to each other through Ethernet port.
2. GPU workstation - CRISP_GYM to deploy policy, LeRobot to train and infer, and the ACT policy node. This machine has an NVIDIA RTX 2080 Ti graphical card, capable of doing all deep learning computation and communicating with the robot PC through ROS 2 topics over CycloneDDS with a shared ROS_DOMAIN_ID of 100.

4.3 Software

4.3.1 Middleware: ROS 2 Humble and CycloneDDS

ROS 2 Humble offered a communication backbone to the entire system allowing the streaming of sensor data, publishing of control commands and state monitoring in both machines. The RMW (ROS 2 middleware) implementation to be used is CycloneDDS due to its dependable behavior in the networked environment of multi-machines. Regular execution of CycloneDDS on the robot PC and the workstation PC was necessary - a mismatch between the two machines in implementing RMW was one of the critical engineering issues faced in the integration of the systems, and is addressed in Chapter 7.

4.3.2 Low-Level Control: CRISP

The low-level compliant controllers that were implemented on the robot PC were given through the CRISP framework[21]. CRISP is a C++ library of Cartesian and joint-space impedance controllers, which are lightweight and can use ROS 2 control standard with any manipulator having a joint torque interface. In this piece, the Cartesian impedance control method was employed, thus enabling the robot to pursue end-effector pose goals with compliance - that is, the robot gives way to the external forces instead of rigidly following a path. This compliance is both necessary to safe teleoperation and to policy deployment where the learned policy produces discrete pose goals at low frequency (in the order of 10-15 Hz) which should be tracked smoothly by the low-level controller.

CRISP_PY offered a Python interface to the controllers, making it easier to publish commands without the use of direct access to the ROS 2 topics. CRISP_GYM offers a wrapper of a Gymnasium-compatible environment that has various purposes on each machine. CRISP_GYM was deployed on the robot PC for data collection where demonstration episodes were directly recorded in LeRobot format and communication with ROS 2 was managed. On the GPU workstation, CRISP_GYM is

used to execute the deployment script in inference where the policy environment is managed and observations and actions are redirected between the learned policy and the robot PC by using ROS 2 topics.

4.3.3 Learning framework: LeRobot 0.2.0

LeRobot[22], the system created by Hugging Face offered the single architecture of managing data sets, training policies, and deploying them. Version 0.2.0 was used throughout. LeRobot unifies the format of dataset between various robot platforms and policy architectures, and natively supports the ACT, Diffusion Policy, and VQ-BeT algorithms. The fact that it integrated with CRISP_GYM implied that demonstration episodes captured in the process of teleoperation were available at once to be used in training without any further preprocessing.

4.4 System architecture

All critical processes are executed by the robot PC: the CRISP Cartesian impedance controller (1 kHz), the camera topic publishers of both RealSense cameras, and the ROS 2 joint state broadcaster. All the learning processes are managed on GPU workstation: CRISP_GYM environment manager, LeRobot training and inference stack, and the ACT policy node. When deploying, the policy node on the GPU workstation receives the image captured by the cameras, the proprioceptive state through ROS 2 topics, calculates an action chunk, and publishes Cartesian pose targets back to the robot PC, which is then implemented by the CRISP controller at 1 kHz. All Cartesian poses are expressed in the robot base frame, which serves as the world frame throughout this system. The end-effector frame is defined at the gripper tip with the z-axis pointing along the approach direction. This architecture is depicted in Figure 4-1.

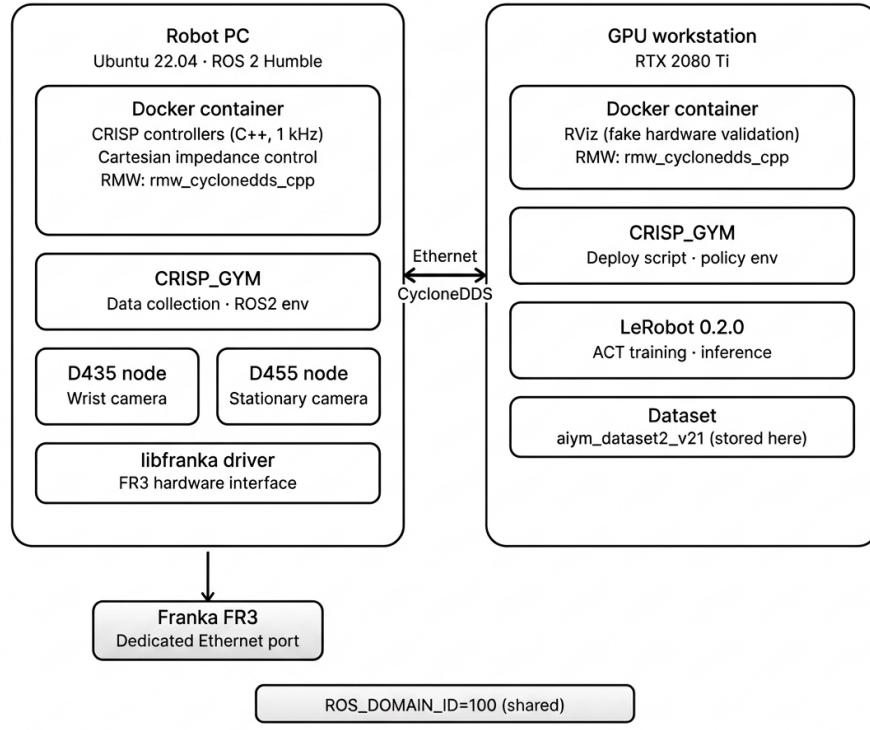


Figure 4-1: Two-machine system architecture. All Cartesian poses are expressed in the robot base frame, which serves as the world frame for this system.

4.5 Fake hardware validation

The complete software pipeline was tested in fake hardware mode on - a ROS 2 feature that emulates the joint state broadcast of the physical robot, but without physical connection to the hardware, before any experiments were run on the physical robot. This enabled the entire pipeline, such as camera topic streaming, CRISP controller configuration and policy inference to be tested and debugged on the GPU workstation alone with visual feedback in RViz.

This test process was especially useful when it came to troubleshooting and fixing the issues of software integration with no threat to the hardware. During this stage, the RMW discrepancy between machines, gripper start-up sequences, and camera topic naming patterns and conventions all were debugged. Figure 4-2 illustrates the RViz display at the time when testing fake hardware occurs and it is clear that the

FR3 model reacts appropriately to the poses when using commands without contact with a real robot.

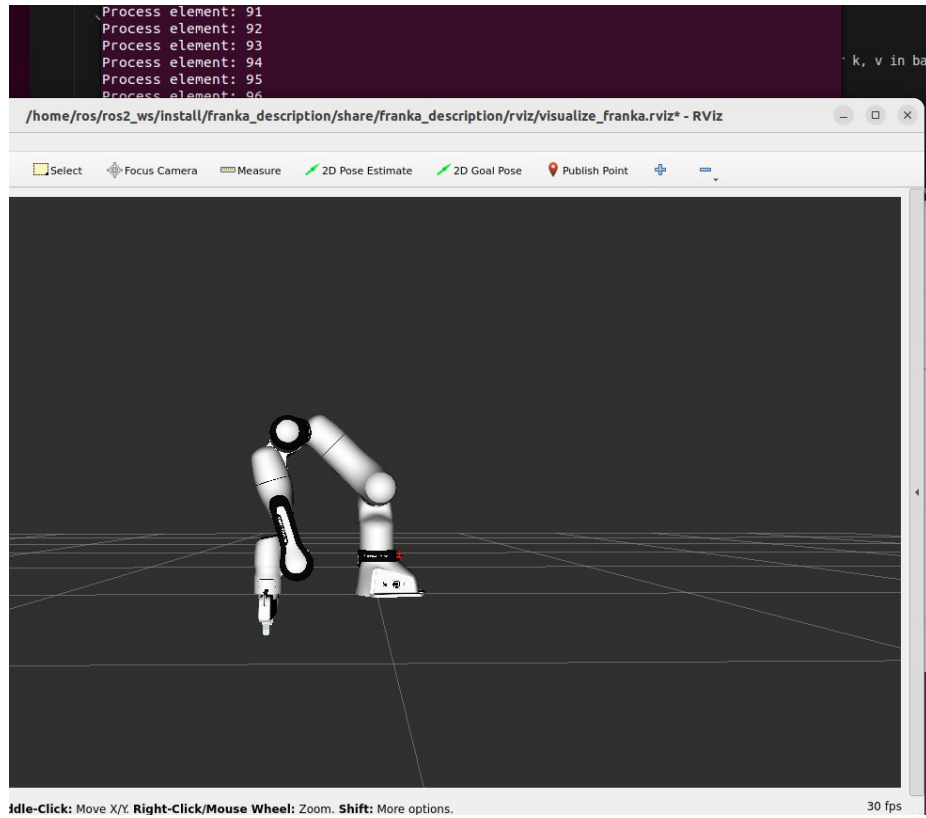


Figure 4-2: RViz visualization during fake hardware testing.

Chapter 5

Demonstration data collection

5.1 Overview

Data on the real-robot training was obtained by joystick-based teleoperation of the CRISP_GYM framework on the Franka FR3. The data collection included the control of the robot by hand and an attempt to perform a pick-and-place operation numerous times with each execution recorded in the form of a full demonstration episode. There were three demonstration datasets that were gathered throughout this project in the LeRobot format, the sole source of supervision of the training of the ACT policy. The datasets were based on lessons learned during the deployment attempts, and the changes were driven by the certain failure modes.

5.2 Teleoperation setup

The 6-axis Megatron joystick controller was used to gather demonstrations connected to the robot PC that was running CRISP_GYM. Joystick movements are remapped by the joystick interface to Cartesian end-effector velocity commands that are sent to the CRISP Cartesian impedance controller. There were joystick buttons that were mapped to gripper open and close commands. The system enabled the intuitive control of the end-effector in 3D space continuously, without any programming skills of how the joints of the robot are configured. The teleoperation interface

was used in the Gymnasium of CRISP_GYM which also logged all the observations and actions at 15 FPS in every demonstration. The data was saved directly to disk in LeRobot format, and each episode was saved as an individual MP4 video file of each camera stream and a synchronized Parquet file of proprioceptive state and action sequence information. The teleoperation configuration, as illustrated in Figure 5-1, involves the joystick controller, robot and workspace. Note that the actual recording rate was approximately 6 FPS due to control loop processing overhead, despite the metadata reporting 15 FPS — deployment runs must explicitly match this with `-fps 6`.

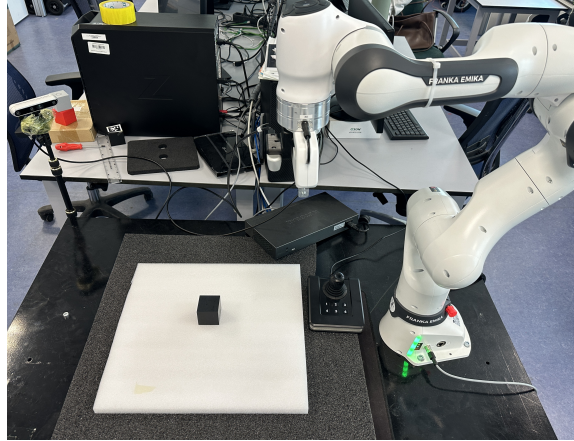


Figure 5-1: Teleoperation setup for demonstration data collection. All end-effector motions are commanded in the robot base frame (world frame), with the z-axis pointing vertically upward.

5.3 Task description

The activity presented was a pick and place task with a small black cube as the manipulation object. The work space was made on a black table that was padded with a white foam that allows the camera to see the object clearly. At the start of every episode the cube was put somewhere fixed on the foam pad. The operator then operated the joystick to work the robot through the entire task sequence; approaching the cube, descending to pick it, grasping the cube, lifting it, moving it across the workspace and placing it at a specified target position. There was a mark for the target

position across episodes but the place accuracy differed slightly among demonstrations because of the variability inherent in manual teleoperation.

Figure 5-2 depicts an example of a sequence of four frames of a teleoperation episode that illustrates the approach, grasp, lift, and place steps of the task.

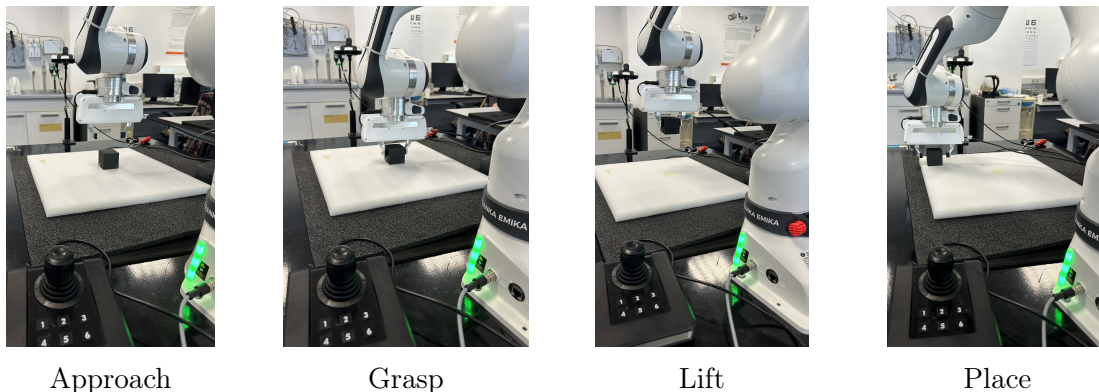


Figure 5-2: Teleoperation sequence for the pick-and-place task using the joystick controller.

5.4 Observation and action space

The observations were from two synchronized camera streams and a proprioceptive state vector. An egocentric perspective of the end-effector and object was observed with the help of a wrist-mounted Intel RealSense D435. The fixed Intel RealSense D455 allowed taking a view of the entire workspace in which the robot arm, the foam pad, and the cube could be observed. The two streams were captured at 256x256 pixels with 15 FPS. Proprioceptive state vector contained 20 dimensions, which contained joint positions, joint velocity, end-effector pose, and gripper state.

The actions were digitized as 7-dimensional Cartesian instructions that included the 3-dimensional delta of end-effector position and orientation, and 1-dimensional command of gripper. This representation of action is directly related to Cartesian impedance controller target format employed by CRISP, so that actions recorded in this format can be directly re-executed by the controller as part of policy deployment.

Figure 5-3 represents the frames of both camera streams in one episode of the demonstration, which depicts the view presented by the two-camera system.

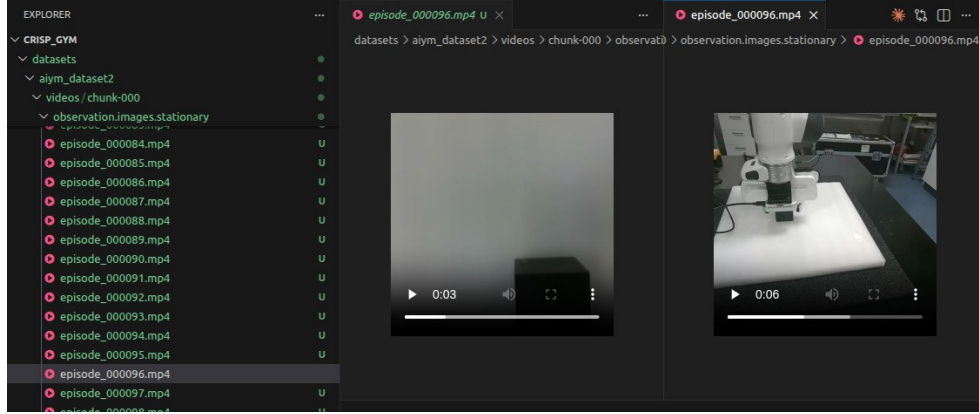


Figure 5-3: Dual-camera observation streams from a single demonstration episode shown in the CRISP_GYM data directory.

5.5 Dataset statistics

5.5.1 Dataset 1

The initial dataset was recorded with the frontal stationary camera set-up mentioned earlier with 98 full episodes. The distribution of episode length is bimodal with clumps at 100-130 frames and 240-310 frames representing the natural variation in the speed of teleoperation between recording sessions. The average length of the episodes is 206 frames.

5.5.2 Dataset 2

The second dataset was gathered with the main reason of expanding the number of demonstrations from 98 to 151 episodes, as the increased number of demonstrations was expected to enhance the performance of policies. The camera was also moved to a less obstructed position of the workspace to a diagonal elevated position to see the object during grasping clearer. The tripod position of the camera was taped so that the position remained the same between recording sessions and deployment. The average length of the episode is 216 frames.

5.5.3 Dataset 3

The third dataset was gathered following deployment analysis of the v3 policy which showed that the downward approach motion was the predominant episode frame and the policy kept on descending instead of lifting after grasping. The main alteration was an intentional change of the teleoperation strategy: once the object was grabbed, the upward lift motion was slowed down and overemphasized, becoming a significantly larger part of each episode. This rebalancing is a direct response to a failure mode of behavior cloning known: unbalanced datasets lead to the over-representation of dominant motions and under-representation of others in policies [24]. The object was changed to red cube. A total of 151 recorded episodes were of a mean length of 306 frames, which is much longer than Dataset 2 because the lift motion was slower. The comparison of episode lengths distribution throughout the 3 datasets is shown in Figure 5-4. The summary of the statistics of all three datasets is presented in Table 5.1.

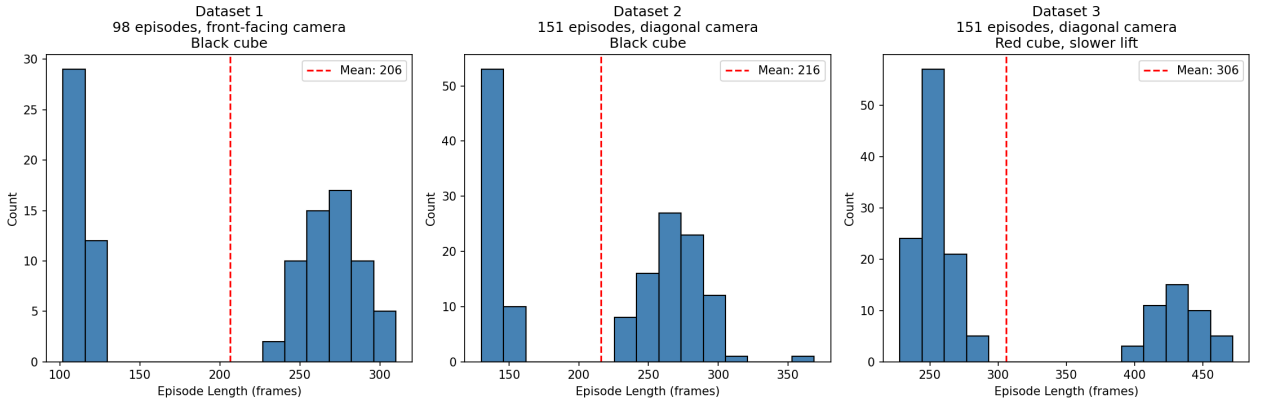


Figure 5-4: Episode length distributions across the three demonstration datasets.

Table 5.1: Demonstration dataset statistics. All datasets used the same task (pick-and-place), proprioceptive state space (20 dimensions), action space (7 dimensions), and storage format (LeRobot MP4 + Parquet).

Parameter	Dataset 1	Dataset 2	Dataset 3
Episodes	98	151	151
Cameras	Wrist + stationary	Wrist + stationary	Stationary only
Camera position	Frontal	Diagonal elevated	Diagonal elevated
Object	Black cube	Black cube	Red cube
Mean episode length	206 frames	216 frames	306 frames
Actual FPS	~ 6	~ 6	~ 6
Metadata FPS	15	15	15

Chapter 6

Policy training

6.1 Algorithm selection

The simulation experiments outlined in Chapter 3 proved that temporally-sensitive policy architectures have a distinct benefit on top of standard recurrent BC especially when dealing with tasks where object interaction is required to be precise. For deployment of real robots, Action Chunking with Transformers (ACT) is chosen as the policy architecture[1]. There were two practical reasons that led to choosing ACT as opposed to diffusion policy in the real-robot context. First, ACT was created specifically to be used in low-data regimes, and it has been initially tested with as few as 50 demonstrations per task. Second, ACT does single-pass inference, where a full action chunk is predicted in a single forward pass through the network, which is not as costly as iterative denoising of diffusion policy and is more compatible with real-time deployment in a single GPU.

6.2 Architecture

The ACT policy architecture is constructed based on a Conditional Variational Autoencoder (CVAE) that is transformer-based [25, 26, 17], illustrated in Figure 6-1. The training goal is a composite of an L1 action reconstruction loss and a KL

divergence regularization:

$$\mathcal{L} = \mathbb{E} \left[\sum_{t=1}^k \|a_t - \hat{a}_t\|_1 \right] + \beta \cdot D_{\text{KL}}(\mathcal{N}(\mu_z, \sigma_z) \parallel \mathcal{N}(0, I)) \quad (6.1)$$

where $\hat{a}_t$ are the predicted actions over a chunk of length $k = 100$, μ_z and σ_z are the style variable outputs of the CVAE encoder, and $\beta = 10.0$ is the KL weight [1]. During training the encoder combines the demonstration action sequence along with the current joint observation and compresses into a style variable z , that captures demonstration variability in execution. The transformer decoder takes the current visual observations, proprioceptive state, and z as input and predicts the entire action chunk in one forward pass. At test time, z is sampled from the standard Gaussian prior $z \sim \mathcal{N}(0, I)$.

One stream per active camera stream visual observations are processed by a ResNet18 backbone [27] trained on ImageNet, generating feature embeddings that are fed to the transformer. The transformer encoder has 4 layers and the decoder has 1 layer, dimension of the model: 512, feedforward dimension: 3200, 8 attention heads, and dropout of 0.1. A temporal ensemble strategy is used in deployment to smooth the implemented trajectory. Overlapping action chunk the predictions of a series of inference steps are averaged as follows:

$$\hat{a}_t = \frac{1}{|\mathcal{W}_t|} \sum_{i \in \mathcal{W}_t} a_t^{(i)} \quad (6.2)$$

where $\mathcal{W}_t$ is the set of overlapping chunks that predict the action at timestep t . This minimizes jitter at boundaries of chunks and generates smoother Cartesian trajectories during real-robot execution.

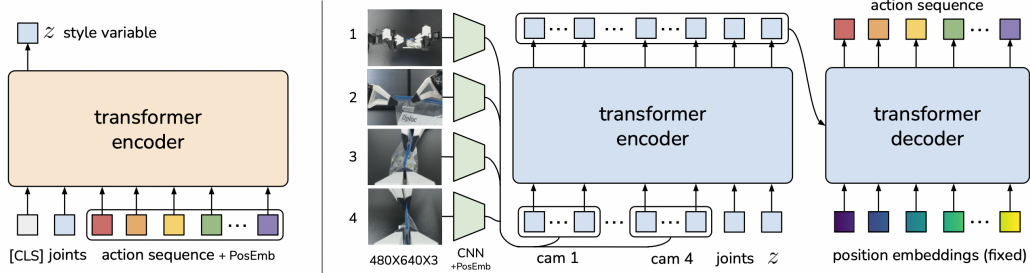


Figure 6-1: The ACT policy architecture. Reproduced from [1].

6.3 Training configuration

The training was performed on the LeRobot version 0.2.0 on the GPU workstation for 100,000 steps in four runs. Hyperparameters were the same in all runs. The AdamW optimizer was used with a fixed learning rate of 1×10^{-5} , beta parameters $(\beta_1, \beta_2) = (0.9, 0.999)$, epsilon $\epsilon = 1 \times 10^{-8}$, and weight decay $\lambda = 1 \times 10^{-4}$. The gradient clipping was used with a maximum norm of 10.0. There was no learning rate schedule applied - it was constant during training. The checkpoints were saved after every 20,000 steps and the last checkpoint at step 100,000 was applied to all deployment experiments.

All active camera streams were augmented with images to enhance generalization to minor changes in lighting and appearance during training. The augmentation pipeline used was the random ColorJitter transformations, perturbing the brightness, contrast, hue, and saturation of each image randomly, and SharpnessJitter, which perturb the image sharpness randomly. The two augmentations were used with independent random parameters per image per training step. These augmentations make the policy to concentrate on the task-relevant visual features as opposed to overfitting to the particular lighting or color-specific conditions that were observed in the data collection process.

Pre-training, action and state observations were normalized based on the mean and the standard deviation calculated on the training data. All four training runs are summarized in Table 6.1. The values of the training losses are not monotonically decreasing: v3 has the same final loss as v1 even though it has more data on its side,

because the removal of the wrist camera reduced the information that the policy sees. v4, which was trained on the rebalanced Dataset 3 using only stationary camera, had the lowest final loss of 0.040, and the lowest observed loss of 0.038 at an intermediate checkpoint.

Table 6.1: ACT training run summary. All runs used 100,000 steps, AdamW optimizer, learning rate 1×10^{-5} , batch size 8, and the final checkpoint at step 100,000 for deployment.

Run	Dataset	Cameras	Episodes	Final loss	Min. loss
v1	Dataset 1	Wrist + stationary	98	0.044	0.044
v2	Dataset 2	Wrist + stationary	151	0.042	0.041
v3	Dataset 2	Stationary only	151	0.044	0.041
v4	Dataset 3	Stationary only	151	0.040	0.038

6.4 Training dynamics

Figure 6-2 plots training loss of the four runs on 100,000 steps on a logarithmic scale. The pattern of convergence of all runs is the same: a steep decrease from about 5.6 at the 200th step to lower than 1.0 in the first 5,000 steps, then a gradual steady decrease up to 100,000 steps. The v1 curve (green dashed) is an approximation, and is built up by checkpoint measurements and not by continuous logging.

The v2 and v3 curves (blue and orange) come very close to each other during training and reach a final loss of 0.042 and 0.044 respectively. This resemblance validates the fact that the training loss did not change significantly when the wrist camera was removed in v3 - the policy fits the data equally well with or without the wrist camera. This difference in behavior between v2 and v3 deployments is thus a measure of the quality of visual information during inference time and not a difference in training convergence.

The v4 curve (red) has the lowest final loss 0.040, and the lowest point is 0.038 in an intermediate checkpoint. This enhancement is due to the increased number of episodes as well as a better distribution of actions in Dataset 3, where the upward lift motion is more accurately captured. The steady reduction in final loss among datasets -0.044 to 0.042 to 0.040- is associated with the positive change in deployment behavior described Chapter 7.

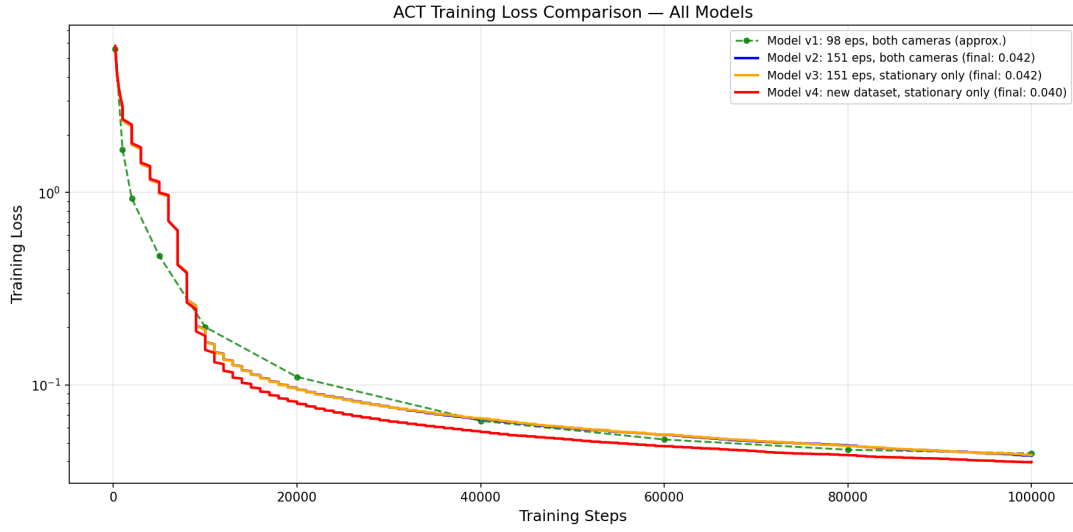


Figure 6-2: ACT training loss for all four runs on a logarithmic scale.

Chapter 7

Deployment and results

7.1 Overview

After training the ACT policy with four configurations, these learned models then were rolled out one at a time on the real-robot Franka FR3. The various deployment executions exposed certain failure modes which inspired the modifications of the next training setup. This chapter outlines the deployment pipeline, the engineering issues faced and solved in all the runs and how the robot behaved in each deployment. The four runs collectively constitute an iterative experimental record, where one design decision is directly informed by the following result. The experiments of deployment described in this chapter directly answer the second and third research questions in Chapter 1. The four experiments of deployment run on the FR3 with a small dataset find out whether ACT can be effectively deployed on the robot, and the iterative failure analysis reveals the practical factors that have the most significant impact on real-robot policy performance.

7.2 Deployment Pipeline

The deployment pipeline is based on the two-machine scheme outlined in Chapter 4. 100,000 steps trained ACT policy checkpoint and it was loaded on the GPU workstation and run by the CRISP_GYM deployment script. Each step of the in-

ference process takes the current camera image of the active camera stream and the current proprioceptive state vector, predicts a 100 step block of future Cartesian actions in a single forward pass, and publishes these as successive pose targets to the CRISP Cartesian impedance controller on the robot PC through ROS 2 topics over CycloneDDS. The controller steps the targets received at 1kHz, and compliantly tracks the commanded end-effector.

It was performed asynchronously in its policy execution, which was set with `1_async_default_config`. In this mode, the policy inference executes on the GPU workstation in a parallel thread as the robot takes action on the action chunk that has been predicted. Since ACT stores 100 actions in a single inference call and only re-executes inference after executing a chunk, the effective frequency of inference is low and the overhead of latency on control is insignificant. The initial call took about 1,266 ms to make the inference on the CPU, which verified that the deployment of the system in real time was only possible with the GPUs. Subsequent calls were cache benefited by action chunks and were about 1 ms.

An important parameter to all deployment runs was the `-fps 6` flag, directly specifying the deployment frequency. In the absence of this flag the deployment script would default to 15 FPS, resulting in the robot performing actions 2.5 times faster than it was during the demonstrations. This was also found to be one of the causes of the failures in runs v1 to v3 and it was fixed in v4.

The full deployment command was:

Listing 7.1: ACT policy deployment command on the GPU workstation.

```
cd ~/crisp_gym && source scripts/set_env.sh
export LD_PRELOAD=/home/robot/crisp_gym/.pixi/envs/\
humble-lerobot/lib/libstdc++.so.6

pixi run -e humble-lerobot python crisp_gym/scripts/deploy_policy.py \
--path outputs/checkpoints/aiym_act_v4_no_wrist/100000/pretrained_model
\
--env-config right_aloha_franka_no_wrist \
```

```
--env-namespace "" \  
--repo-id aiym_act_deploy_v4 \  
--no-push-to-hub \  
--num-episodes 1 \  
--policy-config 1_async_default_config  
--fps 6
```

7.3 Engineering challenges

It took a total of 9 software integration problems to be solved in the four deployment runs. These are recorded here in full and as a working guide to future work involving similar pipelines.

RMW middleware mismatch. The Docker container running the Franka controllers on the robot PC used the default `rmw_fastrtps_cpp` middleware, while the CRISP pixi environment on the GPU workstation used `rmw_cyclonedds_cpp`. This prevented the deployment script from discovering ROS 2 topics published by the controllers. The fix required adding a `docker-compose.override.yaml` on the robot PC to force `RMW_IMPLEMENTATION=rmw_cyclonedds_cpp` inside the container, and configuring `CYCLONEDDS_URI` with the correct network interface on both machines (`eno1` on the GPU workstation, `enp1s0` on the robot PC).

Gripper initialization failure. The CRISP gripper node required `/gripper/gripper_state_broadcaster/joint_states` to be published before it would initialize, but this topic was not bridged from the Franka gripper driver. A ROS 2 relay node was written to republish `/franka_gripper/joint_states` to the expected topic name, resolving the initialization timeout.

Hardcoded joint name prefix mismatch. The `joint_control.yaml` configuration referenced joints as `right_fr3_joint1` through `right_fr3_joint7`, while the actual controller parameters used `fr3_joint1` through `fr3_joint7`. This caused a parameter server error on initialization and was resolved by removing the `right_` prefix from the configuration file.

Hardcoded camera topic mismatch. The `RightAlohaFrankaEnvConfig` dataclass had hardcoded camera topic names that did not match the topics used during data collection. Additionally, the subscriber expected compressed images while the camera nodes published raw images. The fix required updating the dataclass defaults to match the recording configuration and enabling compressed image publishing from both camera nodes.

ACTConfig attribute incompatibility. The async policy worker accessed `policy_config.horizon`, which does not exist in LeRobot 0.2.0's `ACTConfig`. The equivalent attribute is `chunk_size`. This was resolved using `getattr(policy_config, 'horizon', policy_config.chunk_size)` for backwards compatibility.

Inference worker API mismatch. The async inference worker attempted to access `policy._queues` and call `populate_queues`, neither of which exist in LeRobot 0.2.0. The inference block was rewritten to call `policy.predict_action_chunk(batch)` directly, which is the correct API for this version.

FPS mismatch between recording and deployment. The robot control loop maintained about 6 FPS when collecting data but the metadata of the dataset was always 15 FPS. The script defaulted to 15 FPS during the deployment and made the robot act about 2.5 times faster than shown and miss the grasp point. The solution was to pass the deployment command with the explicit option of passing the `-fps 6` flag.

Wrist camera occlusion at grasp point. The camera mounted on the wrist (D435) was unable to view the object during the grasping stage, as the fingers of the gripper were in front of the camera. This eliminated the visual cue required by the policy to notice closeness to the object and stop descending. The solution was to re-train with the stationary camera only, which could get a clear view of the entire motion of the grasp.

7.4 Deployment results

7.4.1 Run v1 — Dataset 1, both cameras

The initial deployment utilized the v1 policy that was trained on 98-episode dataset with both stationary and wrist cameras. In all runs, the robot went down to the workspace and proceeded to go down the object and touch the workspace surface, which activated the safety stop. The gripper failed to actuate, no trial included a grasp. The movement was utterly barely similar to pick-and-place: there was no attempt to grasp, no lifting.

This was due to two reasons. The mismatch of the FPS made the robot move at a rate 2.5 times that of the demonstrated movements, resulting in the robot overshooting the grasp point before any grasp signal was received. Also, it was believed that 98 episodes might be insufficient for a successful trial.

7.4.2 Run v2 — Dataset 2, both cameras

The second deployment utilized the v2 policy that was trained on the bigger 151-episode Dataset 2, and still used both cameras. The action was similar to v1 - the robot kept descending and activated the safety stop without making any attempt to grasp. No trial would result in the gripper actuation.

This result was informative despite being a failure. It showed that the issue of robot descending till the floor could not be overcome by increasing the size of the dataset to 151 episodes. It was not the amount of data that caused the problem but the quality of visual information to the policy when the critical grasp moment occurred. The wrist camera did not give any helpful visual feedback at the point of grasping because of the gripper occlusion, and the policy did not get any visual input to cease descent.

7.4.3 Run v3 — Dataset 2, stationary camera only

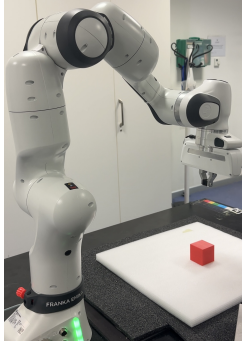
The third deployment was v3 policy, retrained on the same Dataset 2, but with only stationary camera, and the FPS error fixed. This resulted in the first qualitatively significant outcome in all deployments. The robot went towards the workspace following a similar path as the one shown and lowered itself to a height of about the right one above the object. The gripper closed, for the first time, meaning that the policy had learned to relate the visual state of the object to a grasp action. Nonetheless, once the gripper was closed, the robot proceeded to descend rather than lift, which made it touch the workspace surface and activate the safety stop. This is in line with the imbalance of action distribution in Dataset 2: the downwards approach movement took almost 70% of the episode frame, thus the policy was strongly biased downwards even after the gripper had been closed.

7.4.4 Run v4 — Dataset 3, stationary camera only

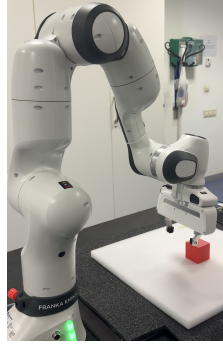
The fourth deployment utilized the v4 policy that was trained on the redesigned 151-episode Dataset 3 with action distribution that was intentionally rebalanced. This gave the best results in all the deployment runs. Figure 7-1 presents the series of frames of the most successful trial.

The robot started at its home position and directed its end-effector towards the workspace along a smooth approach trajectory. It went down to the object and shut the gripper at about the right height - the initial attempt to grasp the object. On this initial try the gripper missed the object and the robot raised a little and then dropped again to make a second attempt, which was slightly different horizontally. The robot went up after a third failed grasp attempt, the first motion of lift that was seen in all deployment runs, and turned to the right to the target location, and opened the gripper as though it were placing the object, finishing the entire kinematic pick-and-place cycle.

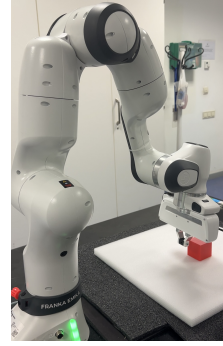
All movements in v4 were smooth and continuous with no sudden jerks or oscillations, which indicated that the Cartesian impedance controller was following the



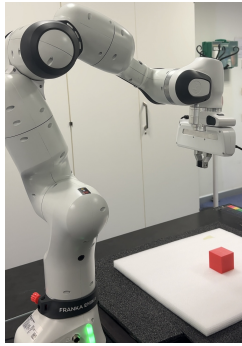
(a) Home position



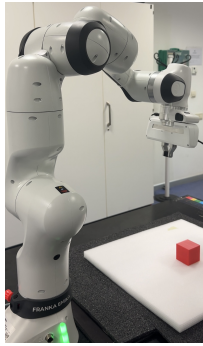
(b) Approach



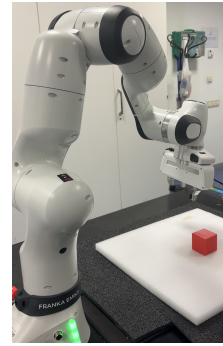
(c) Grasp attempt



(d) Lift



(e) Transport to target



(f) Placement

Figure 7-1: Deployment sequence of the v4 ACT policy on the Franka FR3, extracted from the most successful trial. The robot moves from its home position (a), approaches the workspace (b), and closes the gripper at approximately the correct height (c). It then lifts upward (d), moves laterally to the target location (e), and opens the gripper as if placing the object (f). All motions are expressed in the robot base frame, with lateral transport occurring along the y-axis and the lift motion along the z-axis.

policy commands correctly, and that the temporal ensemble strategy was operating as intended. It is worth to mention that in these deployment runs, formal trial tracking was not used, results were assessed qualitatively, not with an automated measure of success, which is seen as a methodological limitation due to time constraints and not negligence, and is explained further in Chapter 8.

7.5 Analysis

The four deployment runs are a natural progression, which shows the contribution of every design change. Comparison between v1 and v2 isolates the effect of dataset size - the outcome was the same, and it indicates that adding 98 to 151 episodes did not resolve the underlying failure mode. Comparison between v2 and v3 isolates the influence of eliminating the wrist camera - this was the first successful grasp attempt, showing that the most significant barrier to grasping behavior was the occlusion of the wrist camera. Comparison between v3 and v4 isolates the impact of rebalancing action distribution - this generated the first lift motion and full task structure, and at the same time demonstrated that the down-bias in the initial dataset was preventing the policy to learn the post-grasp behavior.

The main failure mode still present in v4 is inaccurate localization of space during the grasp phase. The robot moves down to a height which is likely to be the right one but it has a horizontal error that makes the gripper close in the air instead of closing around the object. It is probably due to two factors. First, 151 demonstrations might not be enough to teach the policy accurate localization of objects in a single stationary camera perspective. Second, visuomotor policies have been shown to be sensitive to minor changes in camera position and scene appearance during training and deployment, leading to systematic spatial errors in predicted actions [28]. The tripod position of the camera was also taped to guarantee uniformity, yet sub-centimeter differences in the tripod position between recording and deployment runs might still result in significant offsets at grasp distance.

The repetitive grasp-attempt behavior seen in v4 - three attempts followed by the

placement attempt indicates that the policy recoded some implicit retry behavior of the demonstrations, with the operator occasionally making several descents before getting a clean grasp. This is a property of the demonstration data and not an explicit design choice, and is an example of how much the BC policies capture the full distribution of demonstrator behavior with hesitations and corrections.

The ACT is an open-loop policy, which implies that the policy lacks a mechanism that would indicate whether the grasp was successful and then move on to the placement stage. This is a basic weakness of standard BC: the policy will follow the learned action sequence no matter what the task state is [29]. The v4 score is thus a ceiling of what standard BC can obtain on this task using this dataset. To bridge this gap, it would be necessary to gather demonstrations in large quantities, add explicit grasp success detection or to go beyond standard BC to a closed-loop feedback method.

Chapter 8

Discussion

8.1 Successes

The main contribution of this thesis is an end-to-end functional pipeline on the manipulation of the Franka FR3 based on learning. It was found, during the simulation study, that Diffusion Policy significantly outperforms BC-RNN on both tasks, which supports the selection of temporally-aware architecture in the application of real-robot deployment. The physical robot underwent four complete training and deployment cycles where each cycle recognized certain modes of failure and inspired specific improvements to the next setup. The gradual reduction of training loss by run e.g., 0.044 in v1 and 0.038 in v4 is associated with the performance in deployment behavior, indicating that both quality and size of the dataset helped to achieve improved policy convergence. The last deployment architecture achieved qualitatively correct pick-and-place behavior - approach, grasp attempt, lift, lateral transport, and placement, and indicates that the pipeline can learn the complete task structure based on human demonstrations.

8.2 Failures

The full task completion was not accomplished in any of the four deployment runs. In both runs v1 and v2 the gripper did not actuate at all - the robot went directly

to the workspace surface and did not attempt to grasp. In run v3 the gripper first closed and then the robot went on descending instead of lifting after grasping. In run v4 that yielded the best results, the gripper failed to pick up the object during most of the trials because there was a consistent spatial offset during the grasp. The main failure mode that was not addressed during the current project was the imprecise localization of the space at the point of grasp.

8.3 Key findings

The iterative deployment process produced three tangible lessons that extend the particular findings of this thesis and apply to the future research of similar pipelines. Visuomotor policy performance is highly dependent on the placement of the camera. The occlusion of the wrist camera at the grasp time completely interfered successful grasping in v1 and v2, and removal of the occlusion was immediately followed by a significant improvement in v3. This observation aligns with the previous studies indicating that end-to-end visuomotor policies are camera viewpoint and occlusion sensitive [28].

The distribution of action in demonstrations is a direct influence on learned policy behavior. The dominance of downward motion in Datasets 1 and 2 resulted in the policy moving downward without executing any lift movement after closing the gripper. Intentionally exaggerating and slowing the lift motion in Dataset 3 solved this and yielded the first measured lift of all the runs. It is an immediate implication of the BC learning objective: the policy learns to recreate the most frequent behaviors in the training data [24, 11]. The datasets of pick-and-place problems should be designed to contain enough demonstrations of every stage of the task approach, grasp, lift, transport, and place, and the coverage should be roughly proportional.

It is imperative to have timing between deployment and data collection match. The FPS mismatch (the actual recording rate was around 6 FPS, but the metadata was reported as 15 FPS) resulted in the robot performing tasks 2.5 times faster in runs v1 through v3, and systematically missed the grasp point. It was an invisible

infrastructure problem, which was experienced in every run of the deployment until it was detected and fixed. To ensure the use of CRISP_GYM or other pipelines in the future, it should be a pre-deployment requirement to ensure that the deployment FPS is equal to the actual recording rate.

8.4 Limitations

There are four limitations to the inferences that can be made out of the findings of this thesis.

First, there was no formal measure of success that was being monitored in trials of deployments. Qualitative results were documented on the outcomes of trials and the approximate success rate of one in eight trials in v4 is just an estimate and not a statistically correct measure. Unless the results of all the trials are systematically tracked, that is, documented to have the outcome of all the trials with a consistent success criterion, then one cannot make confident quantitative claims about the performance of the policy.

Second, 151 demonstrations might not be enough to teach the policy how to localize space with a single stationary camera view with high accuracy. Reported performance in the original ACT paper is high with only 50 demonstrations, but they were gathered using standardized hardware and well-calibrated cameras in controlled conditions [1]. The camera used in this work was a consumer-grade tripod-mounted camera with sub-centimeter variability in positioning between sessions, and the workspace was not depth sensing to address this variability.

Third, several variables varied all at once across dataset versions - camera position, object color, and action distribution - and it was impossible to determine the individual contribution of each of these changes to the increase in deployment behavior. An adequate ablation study would involve varying one variable at a time between distinct data collection and training runs, which were not viable in the constraints of this project.

Fourth, testing was done under one fixed-position condition and one type of ob-

ject. It has not been tested on generalization of the learned policy to diverse object position, to diverse objects, or to workspace layout changes [14]. The Standard BC policies are also sensitive to changes in distribution between training and deployment conditions and the results reported here cannot be expected to be applicable to other configurations.

Chapter 9

Conclusion and Future work

9.1 Conclusion

This thesis introduced a pipeline of robotic manipulation behavior cloning on the Franka FR3, which involves simulation validation, real-robot data collection, policy training and hardware deployment through four training and deployment cycles. Simulation experiments with the Robomimic benchmark proved that temporally-aware policy architectures significantly outperform canonical recursive BC - Diffusion Policy with 98-100% success, justifying the algorithmic direction of real-robot deployment. A trained ACT policy was then deployed to four configurations on three demonstration datasets gathered through joystick teleoperation, and on the real robot with a ROS 2 implementation based on CRISP and LeRobot.

The deployment process was iterative and gave successively better results. There were no grasp attempts on the first two runs. The third run created the initial successful gripper closure. The fourth trial, which was trained on a rebalanced dataset with conscious focus on the lift motion, generated the entire pick-and-place task structure approach, grasp attempt, lift, lateral transport, and placement, in the first run. Reliable task performance was not completed and the major failure mode was inaccurate spatial localization during the grasp stage.

The main contribution of this work is a full-documented end-to-end pipeline with the solution of nine non-trivial software integration problems, and an empirical his-

tory of four successive deployment cycles with explicit failure-mode identification and design change between successive deployments. Three tangible results were obtained: camera positioning is a crucial element in visuomotor grasping, distribution of actions in demonstrations directly influences learned behavior, and timing alignment between data-gathering and deployment is crucial. These results can be directly applied to future research on such systems. The last gap - understanding of precision with the existing camera configuration - can be comprehended and addressed with the refinements found further in Section 9.2.

9.2 Future work

The following four directions could be the most direct ways of enhancing the results of this thesis.

The most prioritized direction is the reliable grasping by calibrated sensing for grasp localization. The existing configuration has a single stationary camera with no depth sensing, limiting the spatial resolution for precise localization of objects at grasping distance. The replacement of the existing camera with a depth camera or the inclusion of a camera calibration between recording and deployment steps would directly solve the offset in space that causes most of the grasp failures. Furthermore, using the native joint torque sensing of the FR3 to detect and verify the object in the gripper would make the policy try again after failed grasp attempt and prevent the policy to proceed to the placement phase with an empty gripper.

To compare deployment trials quantitatively in terms of success it is necessary to have a formal way to monitor them. The RealSense depth could monitor the force exerted by the gripper, or the position of the object, allowing for the automatic calculation of an episode’s success and a statistically defensible success rate, instead of the qualitative assessments that have been reported in this thesis.

It would help to determine which factors are most important if a controlled ablation experiment was conducted, separating the impact of the camera viewpoint and action distribution on deployment performance. The current experimental record had

multiple changes across the versions of the datasets. Separate datasets, with different positions of the camera, different appearances of the object, and different distributions of the actions — but with the other conditions the same — would yield clean scientific conclusions.

Lastly, training Diffusion Policy with the same datasets and running it on the actual FR3 would be a great addition to simulation comparison from Chapter 3, offering a direct empirical comparison of ACT and Diffusion Policy under the same real-world conditions.

Appendix A

Teleoperation Script

This script connects joystick input to the CRISP_GYM teleoperation interface. It subscribes to the `/joy` topic published by the ROS 2 and publishes Cartesian end-effector displacement commands to `/phone_pose` and gripper commands to `/phone_gripper`, which are processed by the CRISP_GYM `-use-streamed-teleop` recording pipeline. Joystick axis 0 controls lateral (Y) motion, axis 1 controls forward/backward (X) motion, and dedicated buttons control Z motion and gripper open/close. A home button triggers a smooth return to the origin pose.

Listing A.1: Joystick teleoperation bridge script (`joy_to_teleop_stream.py`).

```
"""ROS2 bridge: joystick (/joy)  TeleopStreamedPose topics.

Reads joystick input from /joy and publishes:

  /phone_pose      (geometry_msgs/PoseStamped) - accumulated absolute pose
  /phone_gripper (std_msgs/Float32)           - gripper value [0.0=closed,
  1.0=open]

Default mapping for a 1-stick / 6-button joystick (MEGATRON RSDESKTOP):

Stick X  (axis 0) : move Y  (left / right)
Stick Y  (axis 1) : move X  (forward / back)
Button 2 (index 1): return to home (reset accumulated pose)
```

```
Button 3 (index 2): move Z  up
Button 4 (index 3): move Z  down
Button 5 (index 4): close gripper
Button 6 (index 5): open gripper
```

Run `--print-joy` first to verify your stick axes and button indices:

```
ros2 run joy joy_node &
pixi run -e humble-lerobot crisp-joy-teleop --print-joy
```

Then in three terminals:

```
Terminal 1 (joy node):
ros2 run joy joy_node
```

```
Terminal 2 (this bridge):
pixi run -e humble-lerobot crisp-joy-teleop
```

```
Terminal 3 (record):
pixi run -e humble-lerobot crisp-record-leader-follower \\\
  --repo-id <account>/<dataset> \\\
  --follower-config right_aloha_franka \\\
  --follower-namespace right \\\
  --use-streamed-teleop
```

```
"""
```

```
from __future__ import annotations
```

```
import argparse
```

```
import rclpy
```

```
from geometry_msgs.msg import PoseStamped
```

```

from rclpy.node import Node
from rclpy.qos import qos_profile_sensor_data
from sensor_msgs.msg import Joy
from std_msgs.msg import Float32

class JoyToTeleopStream(Node):
    """Bridge node that converts /joy messages to TeleopStreamedPose topics.
    """

    def __init__(
        self,
        xy_scale: float = 0.10,
        z_step: float = 0.06,
        gripper_step: float = 0.05,
        deadzone: float = 0.1,
        publish_hz: float = 30.0,
        axis_x: int = 1,
        axis_y: int = 0,
        btn_home: int = 1,
        btn_z_up: int = 2,
        btn_z_down: int = 3,
        btn_gripper_open: int = 5,
        btn_gripper_close: int = 4,
        namespace: str = "",
    ) -> None:
        super().__init__("joy_to_teleop_stream")

        self._xy_scale = xy_scale
        self._z_step = z_step
        self._dt = 1.0 / publish_hz
        self._gripper_step = gripper_step

```

```

self._deadzone = deadzone

self._axis_x = axis_x
self._axis_y = axis_y
self._btn_home = btn_home
self._btn_z_up = btn_z_up
self._btn_z_down = btn_z_down
self._btn_gripper_open = btn_gripper_open
self._btn_gripper_close = btn_gripper_close

# Accumulated pose
self._px = 0.0
self._py = 0.0
self._pz = 0.0

# Gripper: 1.0 = open, 0.0 = closed (starts open, matches
joy_teleop_test.py)
self._gripper = 1.0
self._grip_open = True

self._prev_home = 0
self._homing = False
self._latest_joy: Joy | None = None

prefix = f"{namespace}_" if namespace else ""
pose_topic = f"/{prefix}phone_pose"
gripper_topic = f"/phone_gripper"

self.create_subscription(Joy, "/joy", self._joy_cb,
qos_profile_sensor_data)

self._pose_pub = self.create_publisher(PoseStamped, pose_topic, 10)
self._gripper_pub = self.create_publisher(Float32, gripper_topic,

```

10)

```
self.create_timer(1.0 / publish_hz, self._publish)

self.get_logger().info(
    f"Publishing pose {pose_topic}, gripper {gripper_topic} at {
publish_hz:.0f} Hz"
)

self.get_logger().info(
    f"Stick: axis{axis_x}=X axis{axis_y}=Y | "
    f"Buttons: {btn_home}=home {btn_z_up}=Z {btn_z_down}=Z "
    f"{btn_gripper_open}=open {btn_gripper_close}=close"
)

# -----
def _joy_cb(self, msg: Joy) -> None:
    self._latest_joy = msg

def _axis(self, axes: list, idx: int) -> float:
    """Return axis value with deadzone applied."""
    if idx < 0 or idx >= len(axes):
        return 0.0
    v = axes[idx]
    return 0.0 if abs(v) < self._deadzone else v

def _btn(self, buttons: list, idx: int) -> bool:
    return 0 <= idx < len(buttons) and bool(buttons[idx])

@staticmethod
def _step_toward_zero(v: float, step: float) -> float:
    """Move v toward 0 by at most step."""
    return 0.0 if abs(v) <= step else v - (step if v > 0 else -step)
```

```

def _publish(self) -> None:
    if self._latest_joy is not None:
        axes = list(self._latest_joy.axes)
        buttons = list(self._latest_joy.buttons)

        # Home (edge-triggered): start smooth return to origin
        home_now = int(self._btn(buttons, self._btn_home))
        if home_now and not self._prev_home:
            self._homing = True
            self.get_logger().info("Home pressed returning to origin.")
            self._prev_home = home_now

        if self._homing:
            # Step each axis toward 0 at normal joystick rate no sudden
            jump
            self._px = self._step_toward_zero(self._px, self._xy_scale *
self._dt)
            self._py = self._step_toward_zero(self._py, self._xy_scale *
self._dt)
            self._pz = self._step_toward_zero(self._pz, self._z_step *
self._dt)
            if self._px == 0.0 and self._py == 0.0 and self._pz == 0.0:
                self._homing = False
                self.get_logger().info("Home reached origin.")
            else:
                # X/Y from stick velocity-based, clamped per tick
                dx = self._axis(axes, self._axis_x) * self._xy_scale * self.
_dt
                dy = self._axis(axes, self._axis_y) * self._xy_scale * self.
_dt
                dx = max(-0.02, min(0.02, dx))

```

```

        dy = max(-0.02, min(0.02, dy))
        self._px += dx
        self._py += dy

        # Z from buttons velocity-based, clamped per tick
        dz = 0.0
        if self._btn(buttons, self._btn_z_up):
            dz = self._z_step * self._dt
        if self._btn(buttons, self._btn_z_down):
            dz = -self._z_step * self._dt
        dz = max(-0.02, min(0.02, dz))
        self._pz += dz

        # ---- Gripper: edge-triggered, matches joy_teleop_test.py
exactly ----
        open_now = self._btn(buttons, self._btn_gripper_open)
        close_now = self._btn(buttons, self._btn_gripper_close)
        if open_now and not self._grip_open:
            self._grip_open = True
            self._gripper = 1.0
            self.get_logger().info("Gripper open")
        elif close_now and self._grip_open:
            self._grip_open = False
            self._gripper = 0.0
            self.get_logger().info("Gripper close")

        # Always publish so TeleopStreamedPose doesn't time out on startup
        now = self.get_clock().now().to_msg()

        pose_msg = PoseStamped()
        pose_msg.header.stamp = now
        pose_msg.header.frame_id = "world"

```

```

        pose_msg.pose.position.x = self._px
        pose_msg.pose.position.y = self._py
        pose_msg.pose.position.z = self._pz
        pose_msg.pose.orientation.x = 0.0
        pose_msg.pose.orientation.y = 0.0
        pose_msg.pose.orientation.z = 0.0
        pose_msg.pose.orientation.w = 1.0

        gripper_msg = Float32()
        gripper_msg.data = float(self._gripper)

        self._pose_pub.publish(pose_msg)
        self._gripper_pub.publish(gripper_msg)

def main() -> None:
    """Entry point for the joy-to-teleop bridge."""
    parser = argparse.ArgumentParser(
        description="Bridge /joy to TeleopStreamedPose topics (1-stick / 6-
        button layout)"
    )
    parser.add_argument("--speed", type=float, default=1.0,
                        help="Global speed multiplier applied to all motion
        (default: 1.0)")
    parser.add_argument("--xy-scale", type=float, default=0.10,
                        help="XY speed in m/s at full stick deflection (
        default: 0.10)")
    parser.add_argument("--z-step", type=float, default=0.06,
                        help="Z speed in m/s when Z button is held (default:
        0.06)")
    parser.add_argument("--gripper-step", type=float, default=0.05,
                        help="Gripper change per tick when button held (

```

```

default: 0.05)")
parser.add_argument("--deadzone", type=float, default=0.1,
                    help="Stick deadzone (default: 0.1)")
parser.add_argument("--hz", type=float, default=30.0,
                    help="Publish frequency in Hz (default: 30)")
parser.add_argument("--namespace", type=str, default="",
                    help="Robot namespace prefix for topics (default:
empty)")
# Axis overrides
parser.add_argument("--axis-x", type=int, default=1,
                    help="Axis index for X translation (default: 1 =
stick Y)")
parser.add_argument("--axis-y", type=int, default=0,
                    help="Axis index for Y translation (default: 0 =
stick X)")
# Button overrides
parser.add_argument("--btn-home", type=int, default=1,
                    help="Button index: return to home / reset pose (
default: 1)")
parser.add_argument("--btn-z-up", type=int, default=2,
                    help="Button index: move Z up (default: 2)")
parser.add_argument("--btn-z-down", type=int, default=3,
                    help="Button index: move Z down (default: 3)")
parser.add_argument("--btn-gripper-open", type=int, default=5,
                    help="Button index: open gripper (default: 5)")
parser.add_argument("--btn-gripper-close", type=int, default=4,
                    help="Button index: close gripper (default: 4)")
parser.add_argument("--print-joy", action="store_true",
                    help="Diagnostic mode: print raw /joy values to
identify axis/button indices")
args = parser.parse_args()

```

```

rclpy.init()

if args.print_joy:
    node = rclpy.create_node("joy_diagnostic")

    def _cb(msg: Joy) -> None:
        axes_str = " ".join(f"a{i}={v:.2f}" for i, v in enumerate(msg.
axes))

        btns_str = " ".join(f"b{i}={v}" for i, v in enumerate(msg.
buttons))

        node.get_logger().info(f"AXES: {axes_str}")
        node.get_logger().info(f"BTNS: {btns_str}")

    node.create_subscription(Joy, "/joy", _cb, qos_profile_sensor_data)
    node.get_logger().info(
        "Diagnostic mode: move the stick and press buttons to find their
indices. Ctrl+C to exit."
    )
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()
    return

node = JoyToTeleopStream(
    xy_scale=args.xy_scale * args.speed,
    z_step=args.z_step * args.speed,
    gripper_step=args.gripper_step,
    deadzone=args.deadzone,

```

```

    publish_hz=args.hz,
    axis_x=args.axis_x,    # fixed: was -args.axis_x (caused negative
index bug)
    axis_y=args.axis_y,
    btn_home=args.btn_home,
    btn_z_up=args.btn_z_up,
    btn_z_down=args.btn_z_down,
    btn_gripper_open=args.btn_gripper_open,
    btn_gripper_close=args.btn_gripper_close,
    namespace=args.namespace,
)

try:
    rclpy.spin(node)
except KeyboardInterrupt:
    pass
finally:
    node.destroy_node()
    if rclpy.ok():
        rclpy.shutdown()

if __name__ == "__main__":
    main()

```

Appendix B

Gripper relay node

This node was written to resolve the gripper initialization failure described in Section 7.3. It republishes `/franka_gripper/joint_states` to the topic name expected by the CRISP gripper node.

Listing B.1: Gripper relay node.

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import JointState

rclpy.init()
node = Node('gripper_relay')
pub = node.create_publisher(
    JointState,
    '/gripper/gripper_state_broadcaster/joint_states',
    10)

def cb(msg):
    pub.publish(msg)

sub = node.create_subscription(
    JointState,
```

```
    '/franka_gripper/joint_states',  
    cb, 10)  
  
print('Gripper relay running...')  
rclpy.spin(node)
```

Appendix C

RMW override configuration

The `docker-compose.override.yaml` was added to the robot PC to force `rmw_cyclonedds_cpp` inside the CRISP Docker container, resolving the middleware mismatch described in Section 7.3.

Listing C.1: Docker compose override for RMW configuration.

```
services:
  launch_franka:
    environment:
      - RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
      - ROS_DOMAIN_ID=100
      - ROS_NETWORK_INTERFACE=enp1s0
```

Appendix D

Training configuration

Section D.1 provides the training command used for all four runs. Section D.2 provides the full model configuration saved in the checkpoint `config.json`.

D.1 Training command

Listing D.1: LeRobot ACT training command (run v4).

```
python3 -m lerobot.scripts.train \  
  --policy.type=act \  
  --dataset.repo_id=aiym_data_no_wrist \  
  --dataset.root=/home/robot/datasets/aiym_data_no_wrist \  
  --dataset.video_backend=pyav \  
  --batch_size=8 \  
  --steps=100000 \  
  --output_dir=outputs/train/aiym_act_v4 \  
  --policy.push_to_hub=false  
2>&1 | tee ~/training_log_v3.txt
```

D.2 Model configuration

Listing D.2: ACT policy model configuration (`config.json`).

```
{
  "type": "act",
  "vision_backbone": "resnet18",
  "dim_model": 512,
  "n_heads": 8,
  "chunk_size": 100,
  "n_encoder_layers": 4,
  "n_decoder_layers": 1,
  "use_vae": true,
  "latent_dim": 32,
  "kl_weight": 10.0,
  "optimizer_lr": 1e-5
}
```

Appendix E

Deployment configuration

Section E.1 provides the environment configuration file used for stationary-camera-only deployment in runs v3 and v4. Section E.2 provides the full deployment command for run v4.

E.1 Environment Configuration Files

Two environment configuration files were used across the four deployment runs. The original configuration (`right_aloha_franka.yaml`) was used for runs v1 and v2 with both cameras. The stationary-camera-only configuration (`right_aloha_franka_no_wrist.yaml`) was created for runs v3 and v4. Note that the `cartesian_control_param_com` field is present only in the no-wrist version — its absence in the original file caused the initialization failure described in Section 7.3 when the no-wrist config was first created without it.

Listing E.1: Original environment configuration used for runs v1 and v2 (`right_aloha_franka.yaml`).

```
# right_aloha_franka.yaml (runs v1 and v2)
control_frequency: 30.0
gripper_mode: "absolute_continuous"
max_episode_steps: 1000
observations_to_include_to_state:
```

```

- observation.state.cartesian
- observation.state.gripper
- observation.state.joints
- observation.state.target
joint_control_param_config: null
robot_config:
  robot_type: "franka"
  time_to_home: 2.0
  publish_frequency: 50.0
  home_config:
    - -1.73960110e-02
    - 9.55319758e-02
    - 8.09703053e-04
    - -1.94272034
    - -4.01435784e-03
    - 2.06584183
    - 7.97426445e-01
gripper_config:
  from_yaml: "grippers/gripper_right_v2.yaml"
camera_configs:
- camera_name: "primary"
  camera_frame: "camera_color_optical_frame"
  resolution: [256, 256]
  camera_color_image_topic: /wrist/wrist_camera/color/image_raw
  camera_color_info_topic: /wrist/wrist_camera/color/camera_info
- camera_name: "stationary"
  camera_frame: "stationary_camera_link"
  resolution: [256, 256]
  camera_color_image_topic: /stationary/stationary_camera/color/image_raw
  camera_color_info_topic: /stationary/stationary_camera/color/camera_info

```

Listing E.2: Stationary-camera-only environment configuration used for runs v3 and v4 (`right_aloha_franka_no_wrist.yaml`). The `cartesian_control_param_config` field is present and set to null — its absence caused initialization failure when this file was first created.

```
# right_aloha_franka_no_wrist.yaml (runs v3 and v4)
control_frequency: 30.0
gripper_mode: "absolute_continuous"
max_episode_steps: 1000
observations_to_include_to_state:
  - observation.state.cartesian
  - observation.state.gripper
  - observation.state.joints
  - observation.state.target
joint_control_param_config: null
cartesian_control_param_config: null
robot_config:
  robot_type: "franka"
  time_to_home: 2.0
  publish_frequency: 50.0
  home_config:
    - -1.73960110e-02
    - 9.55319758e-02
    - 8.09703053e-04
    - -1.94272034
    - -4.01435784e-03
    - 2.06584183
    - 7.97426445e-01
gripper_config:
  from_yaml: "grippers/gripper_right_v2.yaml"
camera_configs:
  - camera_name: "stationary"
```

```

camera_frame: "stationary_camera_link"
resolution: [256, 256]
camera_color_image_topic: /stationary/stationary_camera/color/image_raw
camera_color_info_topic: /stationary/stationary_camera/color/camera_info
- camera_name: "wrist"
  camera_frame: "wrist_camera_link"
  resolution: [256, 256]
  camera_color_image_topic: /wrist/wrist_camera/color/image_raw
  camera_color_info_topic: /wrist/wrist_camera/color/camera_info

```

E.2 Deployment command

Listing E.3: Full deployment command for run v4.

```

pixi run -e humble-lerobot python crisp_gym/scripts/deploy_policy.py --path
  outputs/checkpoints/aiym_act_v4/100000/pretrained_model --env-config
  right_aloha_franka_no_wrist --env-namespace "" --repo-id
  aiym_act_deploy_v4 --no-push-to-hub --num-episodes 1 --fps 6 --policy-
  config 1_async_default_config

```

Appendix F

System specifications

Table F.1 summarizes the hardware and software specifications of the two machines used in this work.

Table F.1: Hardware and software specifications of the two-machine system.

Parameter	Robot PC	GPU workstation
OS	Ubuntu 22.04	Ubuntu 22.04
Hostname	frankaubuntu22	robot-HP-Z4-G4
IP address	192.168.1.160	192.168.1.146
GPU	None	NVIDIA RTX 2080 Ti (11 GB)
RAM	32 GB	32 GB
ROS version	ROS 2 Humble	ROS 2 Humble
Primary role	Robot control, cameras	Training, inference

Bibliography

- [1] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.
- [2] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.
- [3] Dylan J. Foster, Adam Block, and Dipendra Misra. Is behavior cloning all you need? Understanding horizon in imitation learning. In *Advances in Neural Information Processing Systems*, volume 37, pages 120602–120666, 2024.
- [4] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 627–635, 2011.
- [5] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.
- [6] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access*, 9:153171–153188, 2021.
- [7] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.
- [8] Leonel Roza. *Robot learning from demonstration of force-based manipulation tasks*. PhD thesis, Universitat Politècnica de Catalunya, 2013.
- [9] Siddharth Mehta and Dylan P. Losey. Unified learning from demonstrations, corrections, and preferences during physical human-robot interaction. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2022.
- [10] Martin Simonič, Tadej Petrič, Aleš Ude, and Bojan Nemec. Analysis of methods for incremental policy refinement by kinesthetic guidance. *Journal of Intelligent & Robotic Systems*, 102(1):5, 2021.

- [11] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2021.
- [12] Pete Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning (CoRL)*, pages 158–168, 2021.
- [13] Shuang Bai et al. Rethinking latent representations in behavior cloning: An information bottleneck approach for robot manipulation. arXiv preprint, 2025.
- [14] Wenbo Liang, Jian Xie, Zhe Wang, Jian Tan, and Xu Ma. Constrained behavior cloning for robotic learning. arXiv preprint arXiv:2408.10568, 2024.
- [15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851, 2020.
- [16] Haonan Wang et al. Diffusion model-augmented behavioral cloning. arXiv preprint, 2023.
- [17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.
- [18] Sami Haddadin. The Franka Emika robot: A standard platform in robotics research. *IEEE Robotics & Automation Magazine*, 2024.
- [19] Zhimin Zhou, Xin Yang, and Xue Zhang. Variable impedance control on contact-rich manipulation of a collaborative industrial mobile manipulator: An imitation learning approach. *Robotics and Computer-Integrated Manufacturing*, 92:102896, 2025.
- [20] Gerardo Soto et al. Nonlinear parameter identification of the Franka Emika PANDA robot. arXiv preprint, 2024.
- [21] Daniel San José Pro, Oliver Hausdörfer, Ralf Römer, Maximilian Dösch, Martin Schuck, and Angela P. Schöllig. CRISP: Compliant ROS2 controllers for learning-based manipulation policies and teleoperation. arXiv preprint arXiv:2509.06819, 2025.
- [22] Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Thomas Wolf, et al. LeRobot: State-of-the-art machine learning for real-world robotics in PyTorch. <https://github.com/huggingface/lerobot>, 2024.

- [23] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. arXiv preprint arXiv:2009.12293, 2020.
- [24] Donald Hejna et al. Towards balanced behavior cloning from imbalanced datasets. *Autonomous Robots*, 2026.
- [25] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, volume 28, 2015.
- [26] Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.
- [27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [28] Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. VIOLA: Imitation learning for vision-based manipulation with object-centric representations. In *Conference on Robot Learning (CoRL)*, 2022.
- [29] Yixiang Hu, Mingshan Jia, Yeneng Lin, et al. Closed-loop visuomotor control with generative expectation for robotic manipulation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.