CentraleSupélec

université PARIS-SACLAY

Big Data Research Project

Vision-based teleoperation of robotic hands through end-to-end deep neural network

Jintao Ma

Github Link:https://github.com/woshimajintao/BDRP [1]

Advisor: Guillaume Thomas - CEA guillaume.thomas@cea.fr

[1] https://github.com/woshimajintao/BDRP

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Guillaume Thomas, for his invaluable and excellent guidance, patience, and insightful feedback throughout the project. His expertise and encouragement have greatly contributed to my understanding and progress in this research.

I am also deeply grateful to Nacéra for her continuous support and constructive advice, especially the GPU sources from CentraleSupelec, which had helped me navigate through the challenges of this project with confidence.

Furthermore, I would like to extend my appreciation to Yuzhe Qin and other PhD students from top-tier universities such as UCSD for their pioneering open-source contributions. Their work has provided a solid foundation and inspiration for my research, allowing me to build upon state-of-the-art methodologies in robotic teleoperation and imitation learning.

Finally, I would like to thank my BDMA colleagues, friends, and family for their encouragement and unwavering support throughout this journey.

Abstract:

In this report, I had implemented all the Steps of my BDRP project.It built upon the completion of previous steps, which mainly involved final solution and implementation, complete experiments and evaluations. For my work, I firstly practiced on some simulation tools like Mujoco,then I accomplished the critical milestone of reproducing the methodology presented in the selected paper *AnyTeleop: A General Vision-Based Dexterous Robot Arm-Hand Teleoperation System*[14],then I focused on a smaller task and finished the hand bounding box detection task based on the Oxford Hand Dataset[10].

(i) In Chapter 1, I presented the project context, key challenges, and objectives, including paper reproduction, data collection, model implementation, and performance evaluation.

(ii) In Chapter 2, I reviewed some related work on vision-based robotic teleoperation and Objective Detection.

(iii) In Chapter 3, I provided background knowledge on vision-based teleoperation and Hand Detection.

(iv) In Chapter 4, I outlined the methodology of AnyTeleop and Hand Detection included Pipelines of Faster-RCNN and YOLO.

(v) In Chapter 5, I detailed the experimental setup on Linux, including configuration, Simulation practice, Retargeting, observation, performance testing, and analysis of results of reproduction.Then I also listed some comparison between Faster-RCNN and YOLO based on some key metrics like mAP.

(vi) In Chapter 5, I summarized all the work I had completed so far along with the timeline and analyzed the shortcomings as well as potential areas for future improvement.

At last, I uploaded the code to GitHub: BDRP Repository. The weeking meeting record and the supervisor Guillaume's feedback documents can be accessed at BDRP Meeting Record and Feedback.

Keywords: Simulation Tool,Vision-Based Teleoperation, Supervised Learning, Hand Detection,Faster-RCNN,YOLO

Contents

Chapter 1

Introduction

1.1 General Context

Over the past few years, the teleoperation of robotic systems has emerged as a vital area of study[11], particularly for scenarios demanding precise manipulation in environments that are dangerous or unreachable for humans. Vision-guided teleoperation utilizes input from visual sensors, such as cameras, to allow human operators to remotely control robots. This approach has a wide range of practical applications, including:

Medical Robotics: Enabling surgeons to perform procedures in distant or high-risk locations.

Industrial Applications: Managing intricate assembly processes in hazardous workspaces.

Space Missions: Facilitating the operation of robotic explorers on planets or other celestial bodies.

Disaster Relief: Assisting in areas impacted by nuclear incidents, chemical spills, or collapsed structures.

As robotic systems advance, supervised learning enables them to imitate human actions, enhancing adaptability and efficiency. A key component in teleoperation is hand detection, which ensures precise control of real hands or robotic hands by accurately tracking hand movements. Challenges such as occlusions, poor lighting, and real-time processing constraints necessitate robust deep learning-based hand detection methods (e.g., YOLOv8, Faster R-CNN) to improve detection precision and responsiveness.

1.1.1 Specific Sample(Underground Coal Mining)

Underground coal mining poses significant safety risks. Vision-based teleoperation enables remote control of mining equipment, reducing human exposure to hazards. However, challenges such as communication latencyand control precision hinder full automation. Stereo vision helps enhance operator control in these harsh environments.

Figure 1.1: Underground Coal Mining

1.1.1.1 Vision-Based Teleoperation for Remote Mining

Underground coal mining poses significant safety risks. Vision-based teleoperation enables remote control of mining equipment, reducing human exposure to hazards. However, challenges such as communication latency and control precision hinder full automation. Stereo vision helps enhance operator control in these harsh environments.

1.1.1.2 Imitation Learning for Repetitive Mining Tasks

Robots can automate repetitive tasks like moving stones through imitation learning, mimicking human demonstrations. However, sensor noise, task variability, and safety concerns demand adaptive models. Combining reinforcement and imitation learning enhances generalization, reducing human intervention.

1.1.1.3 Hand Detection in Mining Environments

Accurate hand detection is essential for teleoperation and imitation learning, but low visibility, occlusions, and computational constraints make it challenging. Solutions include supervised learning and deep learning-based detection (YOLOv8, Faster R-CNN) for environmental adaptability.

1.2 Problem and Challenges

1.2.1 Data Collection

In the field of robotic teleoperation, a critical question arises in Supervised Learning: *Which data should I use?* The success of supervised learning methods[19] heavily relies on the quality and quantity of labeled data. However, obtaining high-quality robotic data remains some significant challenges.

1.2.1.1 Data Quality

Producing robotic datasets requires teleoperating robots to perform various tasks while capturing visual inputs and movement commands. Ensuring consistency, diversity, and accuracy of this data is challenging, especially in complex tasks involving dexterous robotic hands.

1.2.1.2 Simulated Environments for Data Generation

Training robotic systems in real-world environments is expensive and prone to risks. Simulation environments offer a safe and cost-effective alternative, but they introduce the sim-to-real gap, where models trained in simulations may perform poorly in real-world scenarios.

1.2.1.3 Ensuring Real-Time Performance

Vision-based teleoperation systems must process visual data and produce control outputs in real-time. Achieving this level of performance with computationally heavy deep learning

models like ResNet is a challenge that must be addressed.

In addressing these challenges, this project builds upon notable prior works, including:

DexPilot (Handa et al., 2019): Vision-based teleoperation for a dexterous robotic hand-arm system.

Task-Oriented Hand Motion Retargeting (Antotsiou et al., 2018): Mapping human hand motions to robotic hands for manipulation tasks.

Robotic Telekinesis (Sivakumar et al., 2022): Training robotic hands to imitate human hand motions by watching video demonstrations.

1.2.2 Data Tranformation

In the field of robotic teleoperation hand detection, there will be also some problems related to data processing.

1.2.2.1 Data Transformation for Rotated Bounding Box Labels

Converting between formats: Transforming between different bounding box representations (e.g., COCO, Pascal VOC, YOLO formats) while maintaining precision.

Handling small angles: Small rotation angles can cause label misalignment.

Ensuring annotation consistency: Labels might not match after transformation, requiring careful re-computation.

1.2.2.2 Data Preprocessing for Different Input Formats

Standardizing annotation formats: Need to convert annotations to a unified format before training.

Rescaling issues: When resizing images, bounding boxes must be adjusted proportionally.

File structure variations: Some datasets have annotations stored in separate files, while others embed them directly into image metadata.

1.2.2.3 Data Augmentation for Geometric Transformations

Bounding box adjustments: After transformation, the new bounding box must be recalculated accurately.

Occlusion and distortion: Some transformations (e.g., cropping) may cause objects to be partially removed or stretched unnaturally.

Maintaining annotation integrity: Some augmentation techniques may create inconsistencies in the dataset if labels are not updated correctly.

1.3 Objectives

1.3.1 Familiar with simulation tools

My first objective is to become proficient in simulation tools like MuJoCo or IssacGym, gaining hands-on experience in simulating robotic environments and dynamic interac-

tions.Through this, I seek to bridge the gap between simulation and real-world applications

1.3.2 Paper Reproduction

The important objective of this step is to reproduce and extend existing research on vision-based robotic hand teleoperation using a simulated environment. I selected the paper AnyTeleop, published in the top-tier robotics conference *Robotics: Science and Systems 2023, for reproduction*[1]. First, the code for this paper has received 260 stars on GitHub[2], indicating that it is a highly popular open-source project. Additionally, the authors have been actively maintaining the repository in recent months. Furthermore, compared to papers published 6 years ago [2], its system configuration and dependencies are more compatible with current systems.

1.3.3 Data Collection

Collect teleoperation data from a simulated robotic hand system which I had reproduced. Data collection involves operating a robotic hand using visual feedback, capturing both the control commands and corresponding visual observations.

1.3.4 Hand Detection

Use some methods to finish a smaller task: Hand Bounding Box Detection based on a existing datasets.Attempt to apply single-stage objective detection and two-stage objective detection models. Evaluate different metrics like mAP and compare some approaches based on results.

[1] https://roboticsconference.org/2023/
[2] https://github.com/dexsuite/dex-retargeting

Chapter 2

Related Work

2.1 Vision-Based Teleoperation

My initial task is to find and reproduce one of these papers related to Vision-Based Teleoperation System. These papers are mostly from international top-tier conferences in robotics or computer vision, such as ICRA, ECCV, or RSS. They have a relatively high citation count and reflect the latest academic trends in recent years. Some authors have generously provided open source code to facilitate reproduction.Notably, some papers [1] from UC San Diego have received significant recognition, with their open source codes [2] gaining numerous stars on GitHub, especially AnyTelelop.At the same time, reproducing certain papers can be extremely challenging, as replication in the robotics field often requires not only hardware equipment such as robotic arms or specific cameras but also NVIDIA GPUs and nice graphics cards. This significantly increases the complexity and workload of the reproduction process. Here, I will outline the key and core contributions presented in their papers.

Currently, numerous vision-based dexterous teleoperation systems have been developed. In this study, I evaluated AnyTeleop and compared against other systems along three key dimensions and used a table2.1 to compare them:

(i) sensor requirements

(ii) robot compatibility and support

(iii) use case flexibility

Among the examined systems, AnyTeleop distinguished itself as the only framework capable of supporting multiple robot arms while also enabling collaborative teleoperation. Additionally, it was one of only two systems capable of accommodating different dexterous robotic hands, highlighting its versatility and adaptability.

[1] https://xiaolonw.github.io/publication_select.html
[2] https://github.com/yzqin

	Sensor Requirements			Robot-related Support				Use Case	
	Calibration Free	Contact Free	Depth Free	Multiple Arms	Multiple Hands	Reality	Collision Free	Remote Teleop	Collaborative Teleop
DexPilot [16]	×	✓	×	×	×	Real	✓	×	×
Holo-Dex [4]	✓	✓	✓	No Arm	×	Real	×	✓	×
DIME [5]	×	✓	✓	No Arm	×	Real	×	✓	×
TeachNet [24]	✓	✓	×	No Arm	×	Sim&Real	×	×	×
Telekinesis [54]	✓	×	✓	×	×	Real	✓	✓	×
Qin *et al.* [43]	✓	✓	×	No Arm	✓	Sim	×	✓	×
MVP-Real [45]	×	×	✓	No Arm	×	Real	×	✓	×
Transteleop [26]	×	×	×	×	×	Real	✓	✓	×
Mosbach *et al.* [37]	×	×	✓	×	×	Sim	×	✓	×
AnyTeleop	✓	✓	✓	✓	✓	Sim&Real	✓	✓	✓

Figure 2.1: Comparison of Vision-Based Teleoperation System

2.1.1 AnyTeleop

They proposed AnyTeleop as figure2.2 shown, a unified and general teleoperation system that supports diverse arms, hands, realities, and camera configurations within a single framework[14]. Despite its flexibility in integrating various simulators and real hardware, AnyTeleop maintains strong performance.

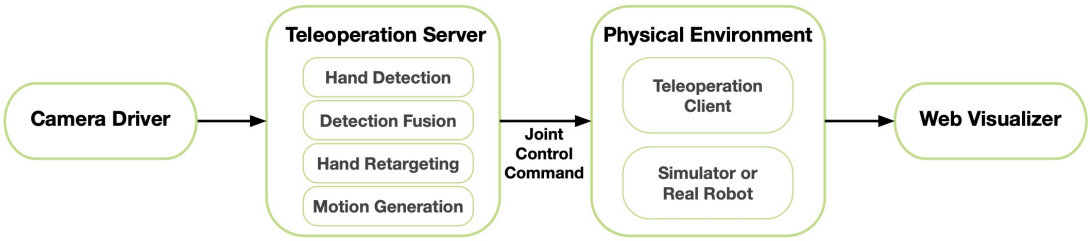


Figure 2.2: AnyTeleop

In real-world experiments, AnyTeleop achieves a higher success rate compared to previous systems designed for specific robot hardware, using the same robot. For teleoperation in simulation, AnyTeleop demonstrates superior imitation learning performance compared to systems specifically tailored for that simulator.

2.1.2 DexPilot

They proposed DexPilot[7], a vision-based teleoperation system designed to integrate three collaborative threads for dexterous robotic manipulation.

As figure 2.3 shown, the first thread, the learning thread, utilized RGB-D inputs and deep neural networks like PointNet++ to perform accurate 3D hand pose estimation and segmentation. The second thread, the tracking thread, refined these pose estimates using Dense Articulated Real-Time Tracking (DART)[3] and retargeted human hand motions to the robotic Allegro hand through kinematic optimization, ensuring alignment with the robot's constraints. The third thread, the control thread, employed Riemannian Motion

[3] https://github.com/tschmidt23/dart

Policies (RMPs)[4] to generate smooth, collision-free trajectories for the robotic arm and hand.

Together, these threads enabled real-time translation of human hand movements into robotic actions with high accuracy and low latency. By seamlessly integrating hand pose detection, tracking, and control mechanisms, the system ensured that even subtle and complex gestures were effectively captured and executed by the robot. This not only improved the efficiency of teleoperation but also enhanced the overall user experience, making DexPilot a highly effective and reliable solution for a wide range of dexterous manipulation tasks, including delicate operations and environments where precise control is crucial.

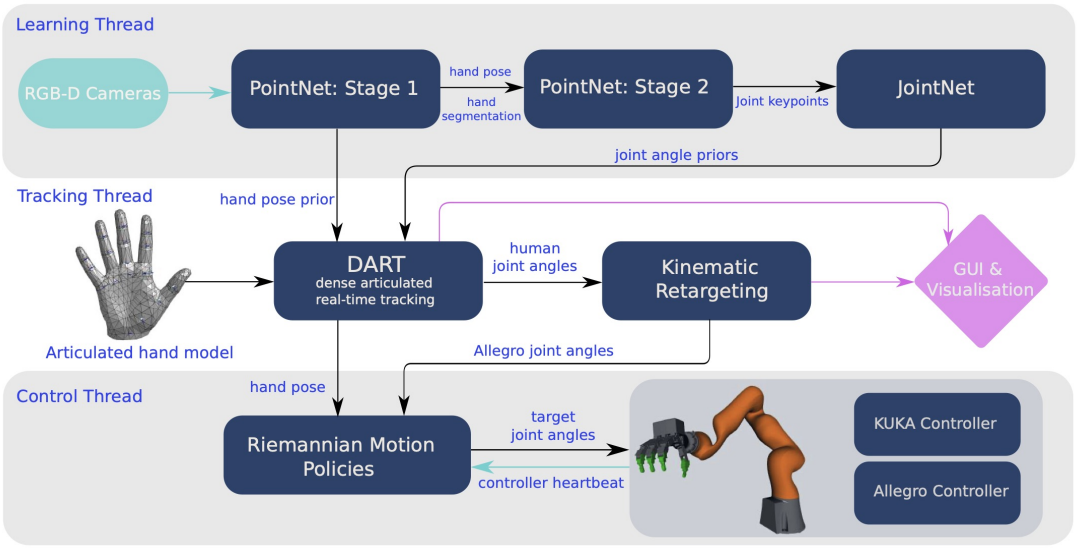


Figure 2.3: DexPilot

2.1.3 Holo-Dex

They proposed HOLO-DEX [3] teleoperation module,which used VR headsets to capture and stream human hand poses, retargeting them to robotic hands like the Allegro Hand. It ensured intuitive control and precise teleoperation through low-latency feedback, enabling compatibility with high-dimensional robotic tasks.

[4] https://research.nvidia.com/publication/2018-03_riemannian-motion-policies

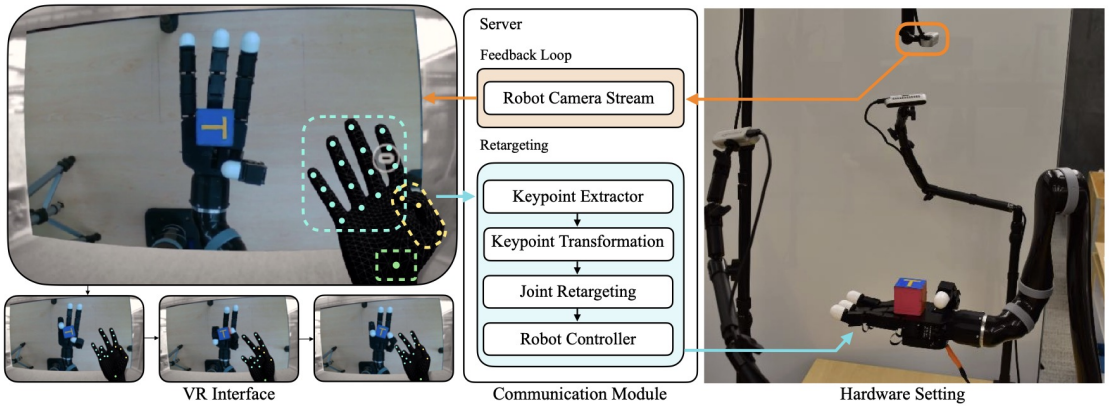


Figure 2.4: Holo-dex

2.1.4 DIME

They proposed DIME[4] teleoperation framework.It consisted of two phases: teleoperation and imitation. In the teleoperation phase, human operators controlled a robotic hand, such as the Allegro Hand, using visual input from an RGB camera. Hand pose detection estimated fingertip positions, which were retargeted to the robot via inverse kinematics, with real-time visual feedback enabling precise demonstrations. These demonstrations were later used in the imitation phase to train dexterous manipulation policies through reinforcement learning in simulation and sample-efficient non-parametric methods on real robots, ensuring adaptability across different environments.

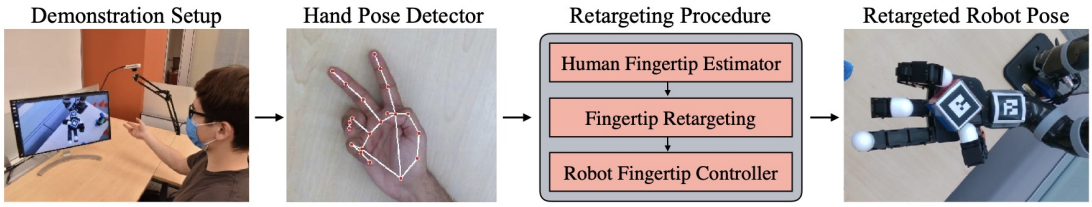


Figure 2.5: DIME

2.1.5 Telekinesis

They proposed Robotic Telekinesis2.6, a visual teleoperation system that enables real-time control of a robotic hand and arm using a single RGB camera [17].

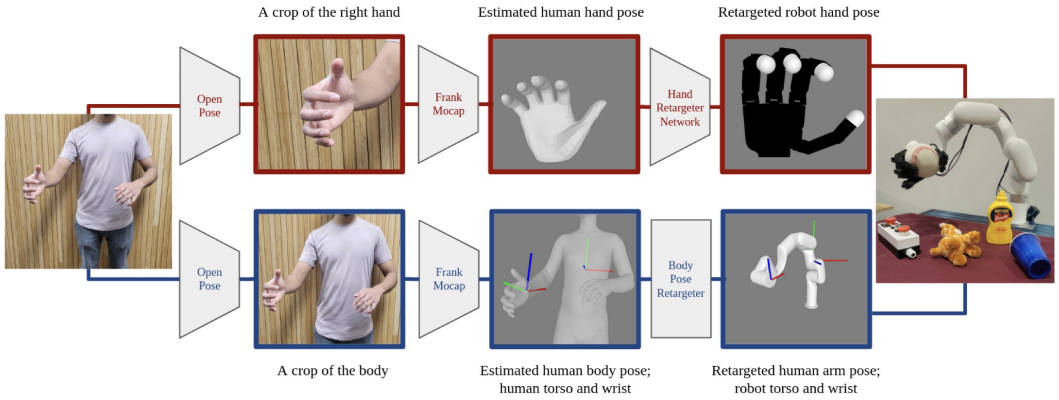


Figure 2.6: Telekinesis

For hand teleoperation, the system estimates the 3D hand pose from RGB inputs via a neural network (using FrankMocap and MANO models) and maps it to the 16-DoF Allegro hand through a retargeting module that preserves functional relationships while avoiding self-collisions using adversarial training.

For arm teleoperation, the pipeline estimates the operator's 3D body pose with the SMPL-X model, calculating the relative transformation between the torso and wrist, which is transferred to the robot arm using inverse kinematics. The unified, glove-free system achieves real-time performance by leveraging large-scale human video datasets, enabling dexterous manipulation tasks with high success rates.

2.1.6 TeachNet

They proposed a vision-based teleoperation architecture, TeachNet[9], provided an end-to-end framework for teleoperating the Shadow Dexterous Hand. It used depth images of a human hand as input and employed a teacher-student neural network to map human hand poses to robotic joint angles. Trained on a large dataset of paired human-robot hand images, it ensured accurate kinematic mappings while addressing joint constraints and self-collisions, enabling intuitive and real-time robotic manipulation.

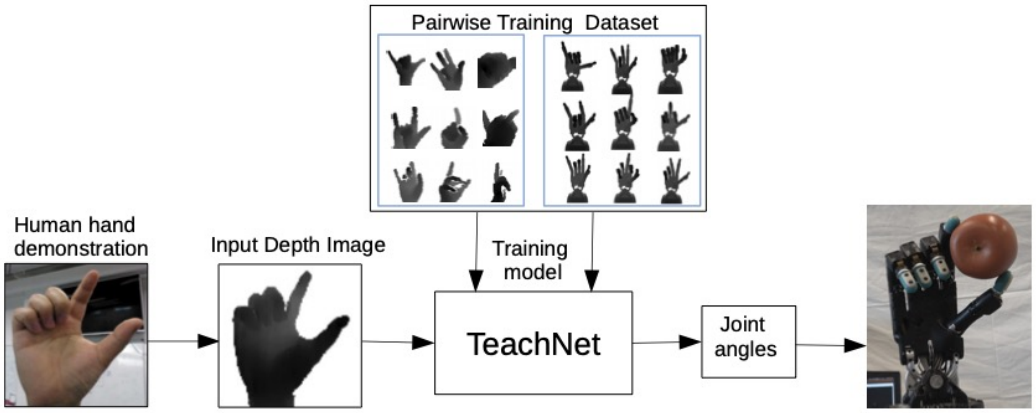


Figure 2.7: TeachNet

2.1.7 Single-camera Teleoperation

They proposed a Single-Camera Teleoperation pipeline[12], The pipeline involved capturing human hand movements with a single-camera system, simulating a customized robot hand for efficient demonstration collection, retargeting trajectories to various robot hands, and using these for demonstration-augmented policy learning. Trained policies combined imitation and reinforcement learning, enabling robust and human-like manipulation on real-world robots across diverse tasks and environments.

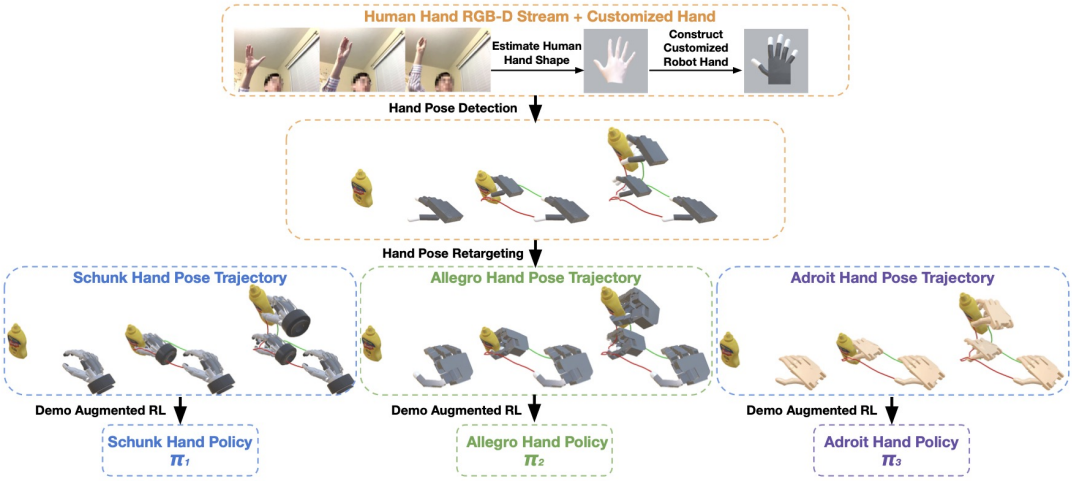


Figure 2.8: single-camera teleoperation

2.1.8 Transteleop

They proposed the Transteleop architecture[8].It utilized a vision-based teleoperation system with an image-to-image translation approach to control robotic hands. Depth images of human hands were processed through an encoder-decoder network to extract shared kinematic features and predict robot joint angles. A keypoint-based reconstruction loss enhanced fingertip accuracy, while IMU-based arm control and a custom camera holder enabled precise and natural hand-arm teleoperation for dexterous manipulation tasks.

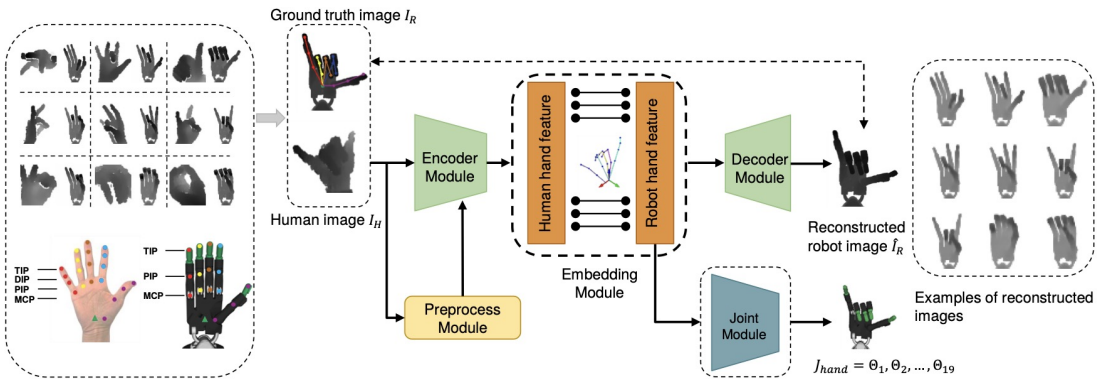


Figure 2.9: Transteleop

2.1.9 DexMV

They proposed DexMV2.10, a system for dexterous manipulation using imitation learning from human videos. The DexMV [13]platform combines a computer vision system for recording human demonstrations and a simulation environment replicating tasks for a robotic hand. Human videos are collected efficiently, and 3D hand-object pose estimation with motion retargeting translates them into robot demonstrations.

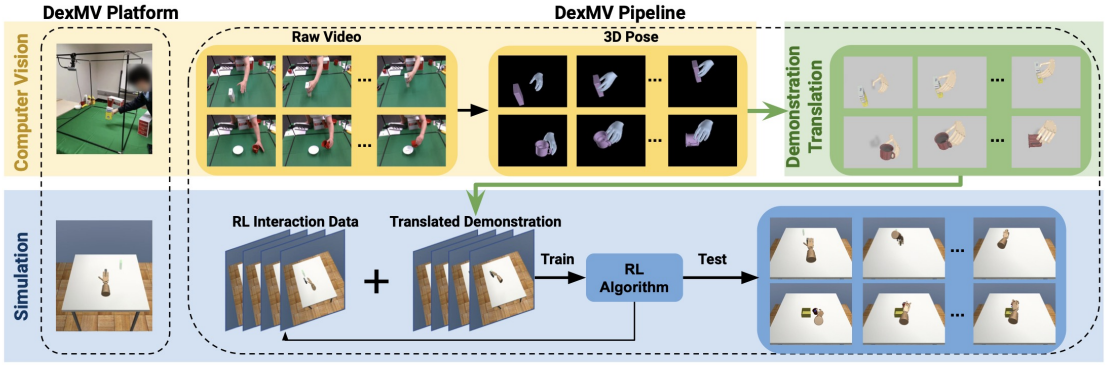


Figure 2.10: DexMV

The DexMV pipeline uses these demonstrations for policy learning, integrating imitation learning and reinforcement learning (RL) algorithms. The resulting policies generalize to varying goals and object configurations, with 3D pose estimation providing critical signals for learning natural and effective manipulation policies.

2.2 Objective Detection

My second task is focus on Hand detection. So I listed some models for objective detection below.

2.2.1 R-CNN(Region-based CNN)

The R-CNN architecture[6] introduced a region-based approach to object detection, leveraging Selective Search to generate region proposals, which were then individually processed using a CNN feature extractor. These features were fed into a Support Vector Machine (SVM) classifier for object recognition and a separate regressor for bounding box refinement. Despite its high accuracy, R-CNN suffered from slow inference speed due to the need for multiple forward passes through the CNN for each region proposal, making it impractical for real-time applications.

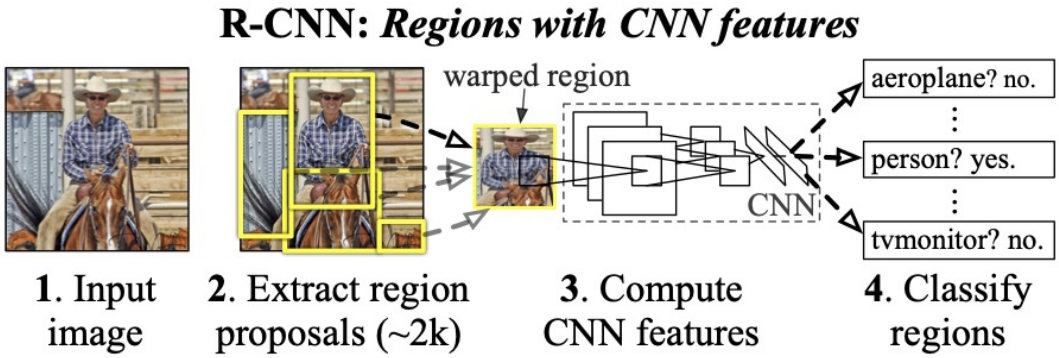


Figure 2.11: R-CNN

2.2.2 Fast R-CNN

Fast R-CNN[5] significantly improved upon R-CNN by introducing a single-stage training framework where an entire image is processed by a CNN only once to extract a feature map. A Region of Interest (RoI) pooling layer was used to generate fixed-size feature representations from different region proposals, enabling faster processing. The network was then jointly optimized for classification and bounding box regression. By eliminating redundant computations and allowing for end-to-end training, Fast R-CNN achieved superior efficiency and accuracy compared to its predecessor.

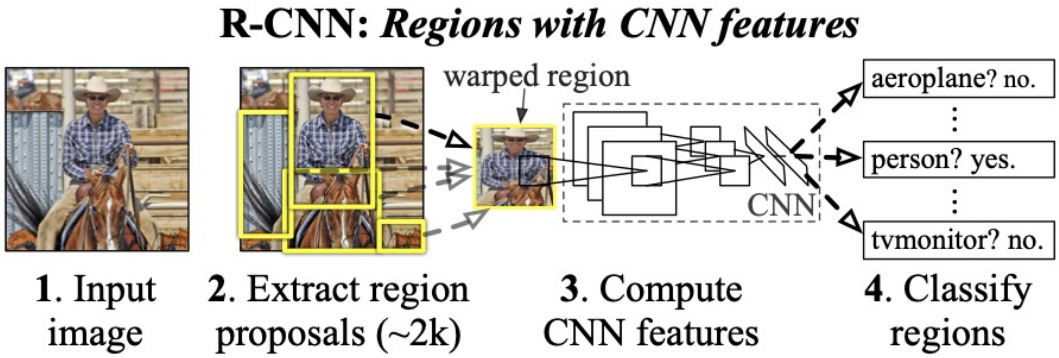


Figure 2.12: R-CNN

2.2.3 Faster R-CNN

Faster R-CNN[16] further enhanced detection speed by replacing Selective Search with a Region Proposal Network (RPN), which directly learns to generate region proposals from the feature map. The RPN and Fast R-CNN detection network share the same backbone, enabling efficient feature reuse. This innovation made two-stage object detection significantly faster while maintaining high accuracy, making Faster R-CNN one of the most widely used frameworks for object detection in research and industry.

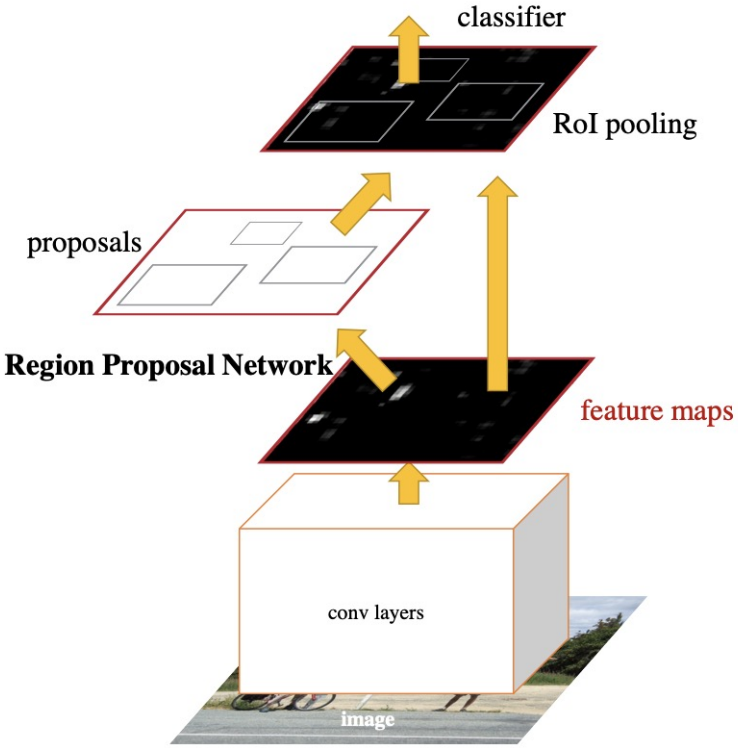


Figure 2.13: FasterR-CNN

2.2.4 YOLO

YOLO[15] introduced a one-stage detection approach, where an image is processed in a single forward pass, splitting it into a grid and predicting bounding boxes and class probabilities simultaneously. Unlike R-CNN-based models, YOLO eliminates the need for separate region proposal steps, making it extremely fast and suitable for real-time applications. However, early versions of YOLO struggled with detecting small objects and precise localization, leading to later refinements.

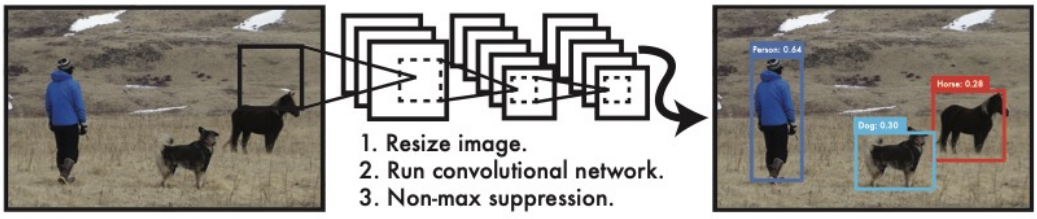


Figure 2.14: YOLO

2.2.5 YOLO V8

YOLOv8[18] had been the latest iteration in the YOLO series untill 2023, featuring enhanced network architecture, better anchor-free detection, and improved efficiency. It integrates modern techniques such as Dynamic Convolution, CSPDarkNet backbone, and a

new loss function (DFL loss) for precise bounding box regression. With superior accuracy-speed trade-offs, YOLOv8 is widely used in various real-time object detection tasks, outperforming previous versions in both precision and computational efficiency.

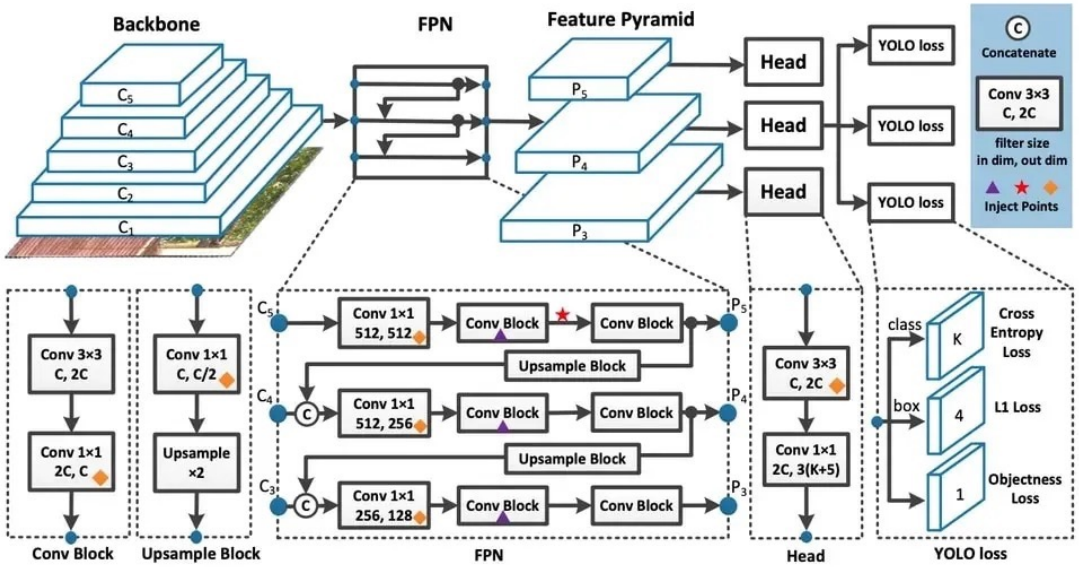


Figure 2.15: YOLO V8

Chapter 3

Background

As robotic systems become increasingly sophisticated, their applications in areas requiring precision and adaptability have expanded significantly. The integration of visual input, machine learning, and simulation environments has enabled advancements in autonomous and teleoperated robotic control. Here I provided a comprehensive overview of the two core concepts and methods relevant to this project, including vision-based teleoperation and Hand Bounding Box Detection.

3.1 Vision-Based Teleoperation

Vision-based robot teleoperation has gained significant attention in recent years, particularly in the teleoperation of dexterous robotic hands by mimicking human hand movements. This method depends on precise tracking of human hand motions and finger articulations. Unlike expensive wearable hand-tracking technologies, such as gloves, marker-based motion capture systems, inertia sensors, or VR headsets, vision-based hand tracking is especially appealing due to its affordability and minimal interference with the human operator.

Earlier research[1] in this field primarily focused on enhancing the accuracy of hand-tracking systems and developing techniques to map human hand poses to robotic hand configurations. More recent advancements have extended the application of teleoperation from individual robotic hands to complete arm-hand systems. However, many of these systems are tailored to specific robot models, such as robotic arms paired with particular robotic hands, and depend on retargeting or collision detection models specifically trained for designated hardware. This lack of flexibility makes transferring these systems to new robotic setups or environments challenging.

In contrast, Yuzhe had developed other modularized systems[14] to address these limitations. These newer solutions employ versatile hand-tracking methods compatible with various camera configurations and customizable robot hand retargeting and motion generation modules. Such modular designs enable adaptation to different robotic arm-hand combinations and diverse environments, achieving improved performance across multiple tasks compared to earlier approaches. These advancements highlight the potential for broader generalization and scalability in vision-based teleoperation systems.

3.1.1 Simulation Tools

Simulation tools are software frameworks that allow researchers and engineers to develop, test, and optimize robotic systems in a virtual environment. These tools provide realistic physics-based simulations, enabling safe and cost-effective experimentation before deploy-

ing robots in real-world scenarios. Common examples include Mujoco,Sapien,PyBullet, Gazebo, and Isaac Sim.

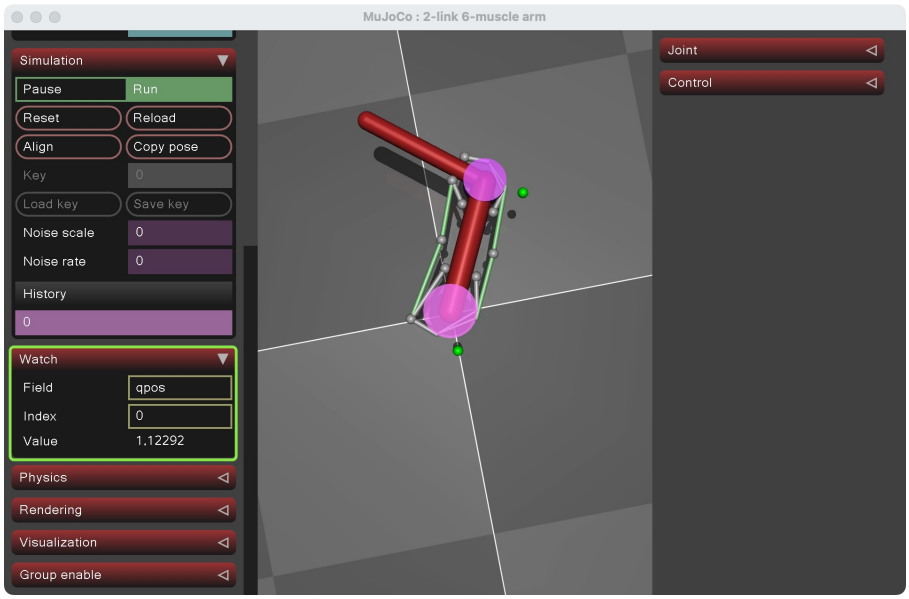


Figure 3.1: Mujoco

3.1.2 Imitation Learning

Imitation learning is a machine learning paradigm where a model learns to perform tasks by observing and mimicking human or expert demonstrations. It is widely used in robotics to teach complex behaviors without explicitly programming every action. Techniques like Behavior Cloning (BC) and Generative Adversarial Imitation Learning (GAIL) are commonly used for training robotic agents.

3.1.3 Dexterous Hands

Dexterous hands refer to robotic hand designs that replicate human-like manipulation capabilities. These hands often feature multiple articulated fingers with precise control, allowing them to grasp, manipulate, and interact with objects in a versatile manner. Examples include the Shadow Hand, Allegro Hand, and DextHand, which are used in teleoperation, prosthetics, and AI-driven robotic manipulation.

3.2 Hand Detection

Hand detection is a computer vision task that focuses on identifying and localizing human hands in images or videos. It is widely used in applications such as human-computer interaction, sign language recognition, and augmented reality.

3.2.1 Bounding Box

A bounding box is a rectangular box that encloses an object of interest in an image. It is commonly used in object detection tasks to indicate the detected object's location and

dimensions.

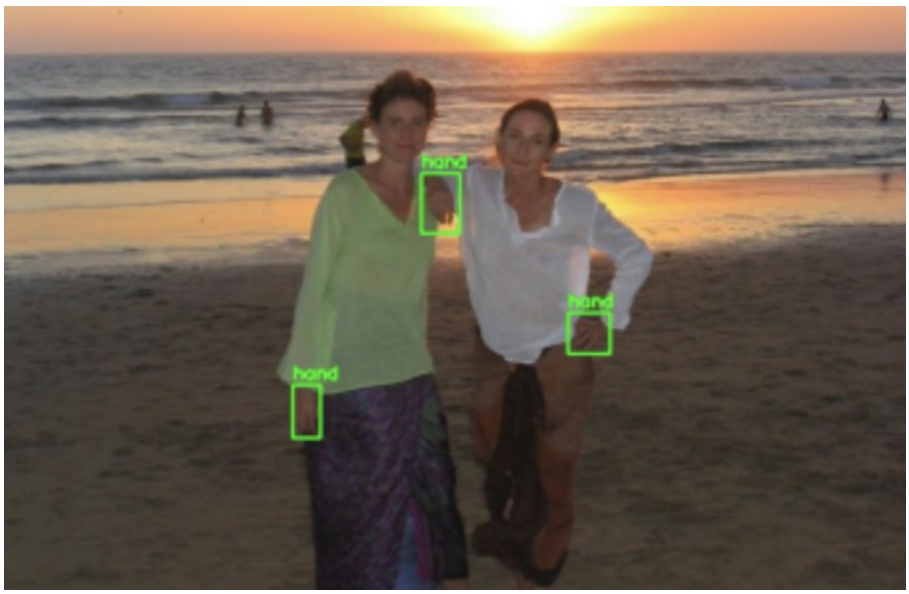


Figure 3.2: Bounding Box

3.2.2 Object Detectors Methods

Object detection methods aim to identify and locate objects within an image. These methods can be broadly categorized into one-stage and two-stage object detectors.

3.2.2.1 One-Stage Object Detectors

One-stage detectors, such as YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), directly predict object bounding boxes and class labels in a single step. They are optimized for real-time performance but may sacrifice some accuracy compared to two-stage detectors.one-stage object detectors like YOLO (You Only Look Once) are generally considered end-to-end models

3.2.2.2 Two-Stage Object Detectors

Two-stage detectors, such as Faster R-CNN, first generate region proposals and then classify and refine the object bounding boxes. These models tend to be more accurate but computationally expensive, making them less suitable for real-time applications.

3.2.3 Metrics

To evaluate object detection models, several performance metrics are used.

3.2.3.1 IoU(Intersection over Union)

IoU measures the overlap between the predicted bounding box and the ground truth bounding box. It is calculated as the ratio of the intersection area to the union area. A higher IoU indicates better localization accuracy.Usually we set the IoU threshold to 0.5.

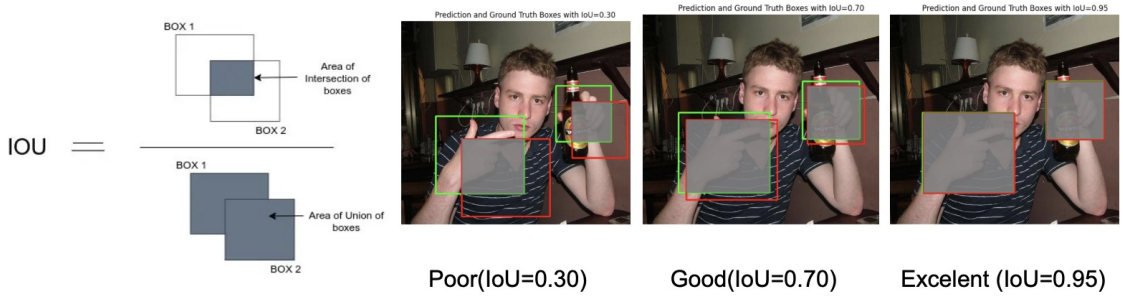


Figure 3.3: IoU

3.2.3.2 Confidence Score

The confidence score represents the probability that a detected object belongs to a specific class. It helps determine whether a prediction should be considered valid. Usually we set the Confidence threshold to 0.3.

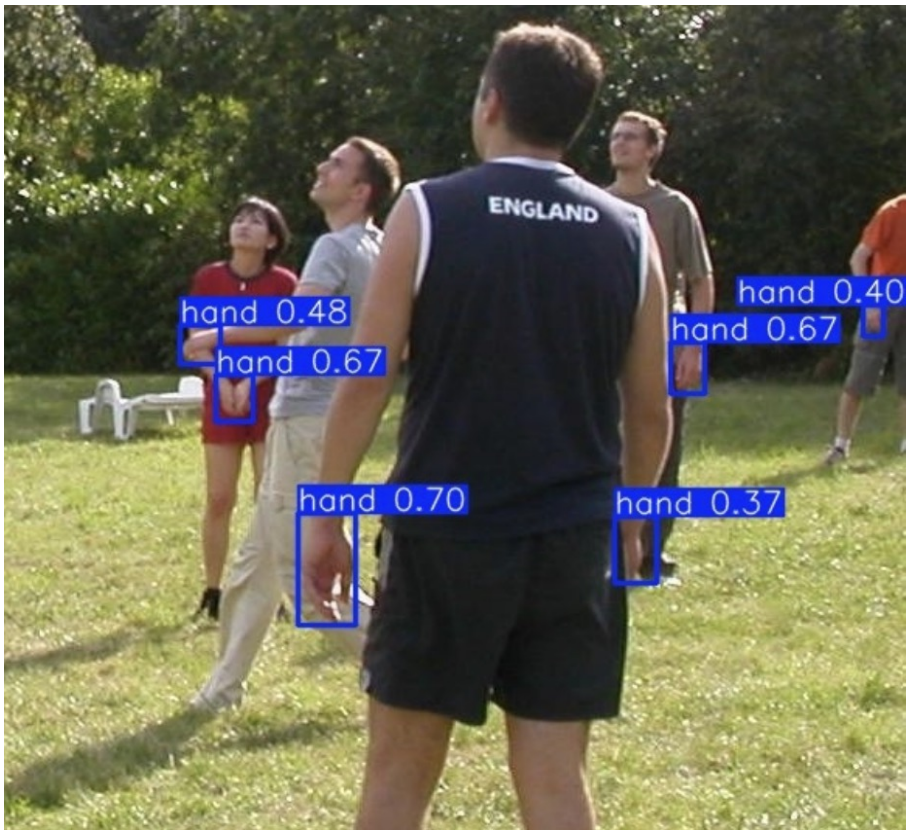


Figure 3.4: confidence score

3.2.3.3 Confusion Matrix

A confusion matrix is a table used to evaluate the performance of a classification model. It includes values such as true positives, false positives, false negatives, and true negatives, providing insight into model accuracy and error distribution.

	(Confidence Score ≥ Threshold)	(Confidence Score < Threshold)
(IoU ≥ Threshold)	**True Positive (TP)** ✅ Correct detection	**False Negative (FN)** ❌ Missed detection
(IoU < Threshold)	**False Positive (FP)** ❌ False detection (wrong object)	**True Negative (TN)** ✅ Correct rejection (no object detected)

Figure 3.5: Confusion Matrix

I also listed a table to show some important metrics like Precision,Recall and F1-score.

Metric	Formula	Purpose
Precision	$\frac{TP}{TP+FP}$	Reduces false positives (FP)
Recall	$\frac{TP}{TP+FN}$	Reduces false negatives (FN)
F1-Score	$2 \times \frac{P \times R}{P+R}$	Balances Precision and Recall

Figure 3.6: PRF Metric Table

3.2.3.4 AP(Average Precision)

AP(Average Precision) measured the model's detection performance across different recall levels.

$$AP = \int_0^1 P(r)\, dr$$

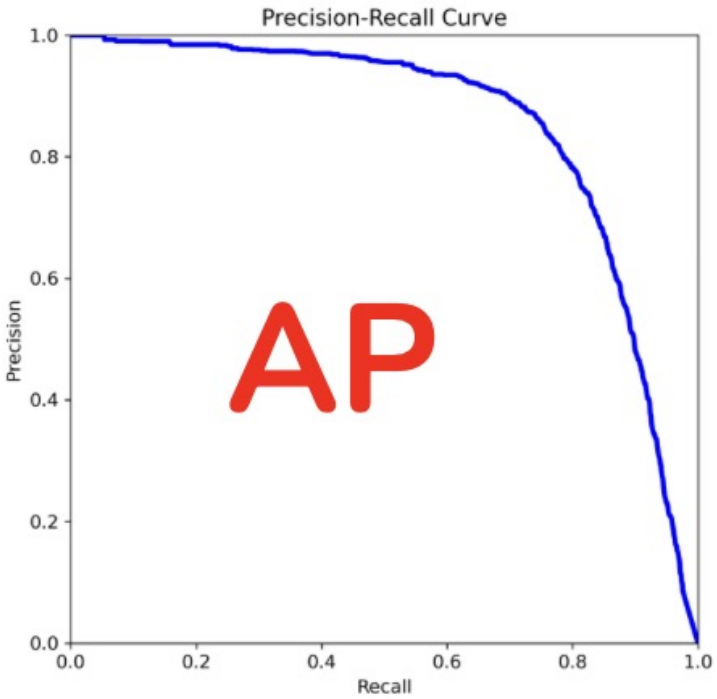


Figure 3.7: AP

3.2.3.5 mAP(mean Average Precision)

mAP is a key metric for object detection that calculates the average precision across different recall levels. It is the mean of AP values across all classes.

$$mAP = \frac{1}{N} \sum_{c=1}^{N} AP_c$$

where:

- N is the total number of classes.
- AP_c is the AP value for class c.

In the Hand bounding box Detection task, there is only one class (i.e., the class "hand").So mAP=AP here. It provides a comprehensive measure of the model's performance in terms of both classification and localization accuracy.

These fundamental concepts play a crucial role in understanding and improving object detection models, particularly in applications involving hand detection.

Chapter 4

Our Methodology and Approach

This chapter outlines the methods in paper AnyTeleop I had reproduced and two hand detection models I used.

4.1 AnyTeleop Methodology

Anyteleop is a generalized vision-based teleoperation system.

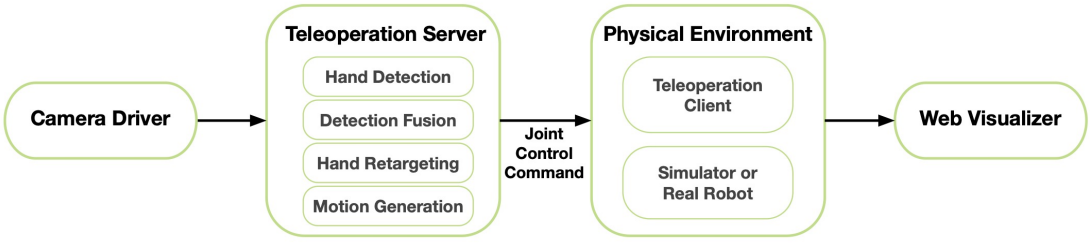
Generalized Vision-Based Teleoperation: AnyTeleop provides a flexible teleoperation system that leverages vision-based control, enabling intuitive and adaptable manipulation.

Support for Various Robotic Hands and Camera Setups: The system is designed to be compatible with multiple robotic hand models and diverse camera configurations, ensuring broad applicability.

Seamless Integration Across Simulators and Real Hardware: AnyTeleop maintains high performance in both simulated environments and real-world robotic tasks, offering a reliable and scalable solution.

4.1.1 System Architecture

The teleoperation system, as depicted in Figure2.2, consisted of a teleoperation server that processed the camera stream provided by the driver, detected the human hand pose, and translated it into joint control commands. These commands were then transmitted over a network to a client, which applied them to control either a real or simulated robot. The system was developed based on three fundamental principles: modularity, communication efficiency, and containerization.



Modularity was realized by defining clear input-output interfaces for each subsystem, enabling compatibility with a wide range of robot arms, dexterous hands, cameras, and environments. The communication-focused design facilitated remote and collaborative

teleoperation by transferring computationally intensive tasks to a dedicated server, reducing the processing load on the operator's device. Additionally, containerization streamlined the installation and deployment process, overcoming the complexities typically associated with robotics systems that rely on extensive software dependencies.

4.1.2 Teleoperation Server

The most important part of AnyTeleop is the Teleoperation Server.So I also drew this architecture as Figure 4.1, show.

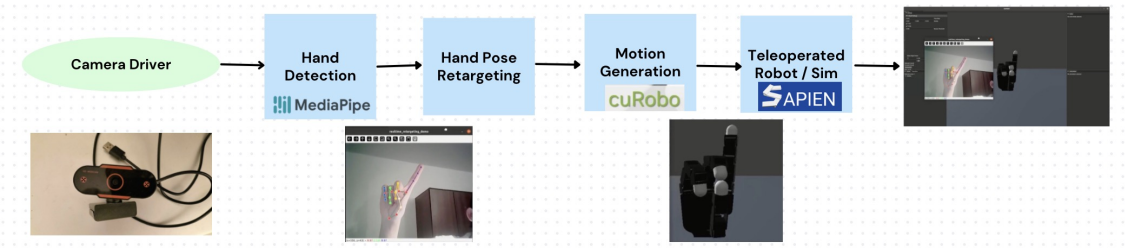


Figure 4.1: Teleoperation Server

Key Functions :

- **Hand Pose Detection**:
  - Processes visual data captured by the Camera Driver to estimate the 3D hand pose of the human operator.
  - Uses advanced algorithms to track joint angles, fingertip positions, and overall hand configuration in real time.
- **Multi-Camera Fusion**:
  - If multiple cameras are used, the server fuses input data to enhance tracking accuracy and minimize occlusion effects.
  - Aligns and calibrates inputs from different viewpoints for a comprehensive understanding of hand movements.
- **Hand Pose Retargeting**:
  - Translates human hand movements into corresponding robotic hand motions through kinematic optimization.
  - Maps the human hand's degrees of freedom (DOF) to the robot hand's specific kinematics, ensuring compatibility while respecting the robotic hardware's constraints.
- **Motion Command Generation**:
  - Generates collision-free and smooth motion trajectories for both the robotic arm and hand using algorithms such as Riemannian Motion Policies (RMPs).
  - Ensures safe and stable teleoperation by dynamically adapting to environmental constraints.

Role in the System: The Teleoperation Server serves as the computational core of the system, converting raw visual input into actionable control signals for the robotic system. It enables seamless translation of human movements into robotic actions.

4.1.3 Key Models and Algorithms in Teleoperation Server

The Teleoperation Server consists of four key modules: **Hand Pose Detection, Multi-Camera Fusion, Hand Pose Retargeting, and Motion Command Generation** . Among them,**Hand Pose Retargeting** is the core module. This section details the main algorithms and models used in each module.

Module	Main Algorithms and Models Used
Hand Pose Detection	MediaPipe, PnP Algorithm, Weak Perspective Transformation
Multi-Camera Fusion	SO(3) Rotation Calibration, SMPL-X Model, Confidence Scoring
Hand Pose Retargeting	Optimization Problem, Forward Kinematics, Finger Keypoint Mapping
Motion Command Generation	RMPs, CuRobo

Table 4.1: Modules and Their Main Algorithms/Models in the Teleoperation Server

4.1.3.1 Hand Pose Detection

MediaPipe : Detects 21 hand keypoints in 3D coordinates and 2D image coordinates[1].

PnP Algorithm (Perspective-n-Point) : Computes the 6D wrist pose (position and orientation) using RGB-D data, depth images, and camera intrinsic parameters[2].

Weak Perspective Transformation : Estimates the 3D wrist position using a scaling factor for RGB-only images, assuming the object is far from the camera.

4.1.3.2 Multi-Camera Fusion

SO(3) Rotation Calibration : Calculates relative rotations between multiple cameras using hand-relative motion as a marker.

SMPL-X Model : Uses hand shape parameters (Shape Parameters)[3] to evaluate the confidence of detection results.

Confidence Scoring : Selects the most reliable detection results by comparing shape parameter deviations from a reference value.

[1] https://chuoling.github.io/mediapipe/
[2] https://en.wikipedia.org/wiki/Perspective-n-Point
[3] https://smpl-x.is.tue.mpg.de

4.1.3.3 Hand Pose Retargeting

Optimization Problem : Minimizes the difference between human hand and robot hand keypoints while ensuring temporal smoothness.

$$\min_{q_t} \sum_{i=0}^{N} ||\alpha v_i^t - f_i(q_t)||^2 + \beta ||q_t - q_{t-1}||^2$$

where:

- q_t: Joint angles of the robot.
- α: Scaling factor to account for size differences between human and robot hands.
- β: Penalty weight for smoothness.

Forward Kinematics : Computes the 3D coordinates of robot keypoints corresponding to joint angles q_t[4].

Finger Keypoint Mapping : Manually defines the mapping between human and robot fingers for different robot morphologies (e.g., DClaw).

4.1.3.4 Motion Command Generation

RMPs (Riemannian Motion Policies) : Provides acceleration fields for real-time motion planning towards end-effector poses.

CuRobo : A GPU-accelerated collision-free motion generation library[5].

Generates smooth, high-frequency (120 Hz) trajectories in joint space to ensure safe robot motion.

Impedance Controllers : Executes the generated trajectories, ensuring precise and safe motion of the robot.

4.2 Hand Detection Methodology

4.2.1 Data Processing

The dataset used in this project is the Oxford Hand Dataset, which consists of annotated hand images for detection tasks. It includes 4,089 images with 9,163 hand instances for training, 738 images with 1,856 hand instances for validation, and 821 images with 2,031 hand instances for testing. The dataset provides diverse hand poses and environments, making it suitable for robust hand detection model training. More details can be found at Oxford Hand Dataset[6] .

[4] https://blog.robotiq.com/how-to-calculate-a-robots-forward-kinematics-in-5-easy-steps
[5] https://curobo.org
[6] https://www.robots.ox.ac.uk/~vgg/data/hands/

4.2.1.1 Data Transformation

For the annotation files as input of Faster- RCNN and YOLO, there are two different input formats as figure 4.2 shown.

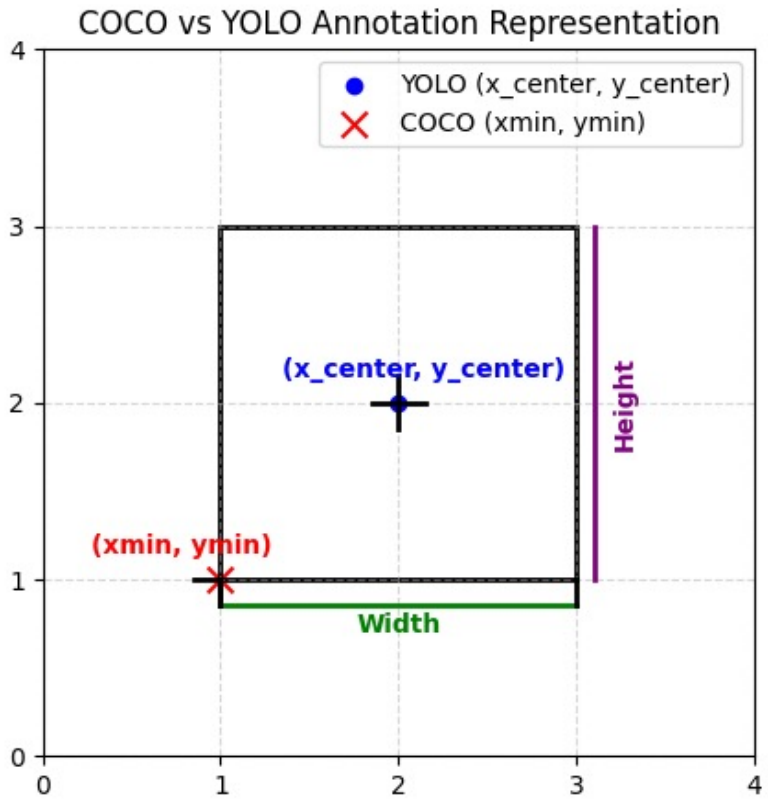


Figure 4.2: Two Annotation

YOLO vs. COCO Coordinate Formats

YOLO uses normalized coordinates with the format (c_x, c_y, w, h), where:

$$c_x = \frac{x_{\min} + \frac{w}{2}}{\text{image\_width}}, \quad c_y = \frac{y_{\min} + \frac{h}{2}}{\text{image\_height}},$$
$$w_{\text{YOLO}} = \frac{w}{\text{image\_width}}, \quad h_{\text{YOLO}} = \frac{h}{\text{image\_height}}.$$

COCO uses absolute pixel coordinates in the format $(x_{\min}, y_{\min}, w, h)$.

So we need to do some data transformation to convert the original .mat file to the COCO format.json file and the YOLO format .txt file.

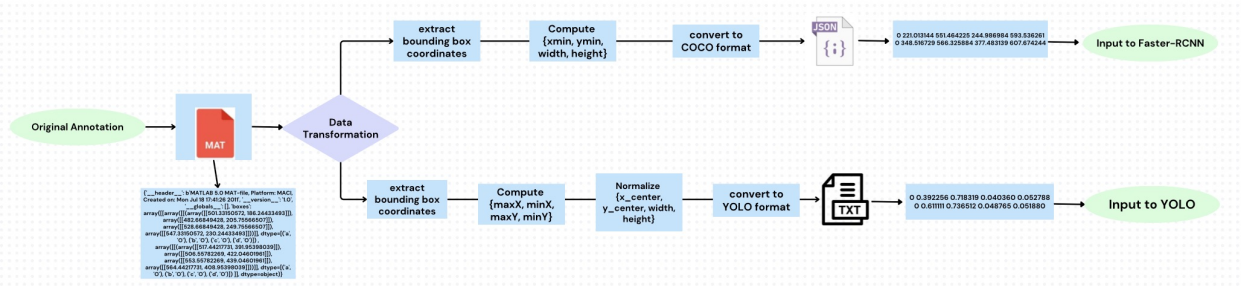


Figure 4.3: Data Processing

For Faster-RCNN: Extract bounding box coordinates from the .mat file, then compute the top-left corner (xmin, ymin) and the width and height of the bounding box. Store the annotations in COCO format and save them as a JSON file.

For YOLO: Extract bounding box coordinates from the .mat file, then determine the bounding box limits (maxX, minX, maxY, minY). Normalize the center coordinates, width, and height relative to the image dimensions. Store the annotations in YOLO format and save them as a .txt file.

4.2.1.2 Data Augmentation

Different strategies were applied to training and validation datasets to improve model generalization and avoid overfitting. Due to limited training data, extensive augmentation techniques were used to improve generalization, including geometric transformations, color and contrast adjustments, and automated methods like AutoAugment. For validation dataset, only normalization and resizing were applied to maintain evaluation stability.

Geometric Transformations: To introduce variations, rotations helped recognize different orientations, while horizontal and vertical flips improved generalization. Shift-scale-rotate operations enhanced robustness, and random cropping focused attention on different bird regions, improving classification.When performing geometric transformations, I adjusted the coordinates of the hand bounding box accordingly for each transformation to ensure precise localization.

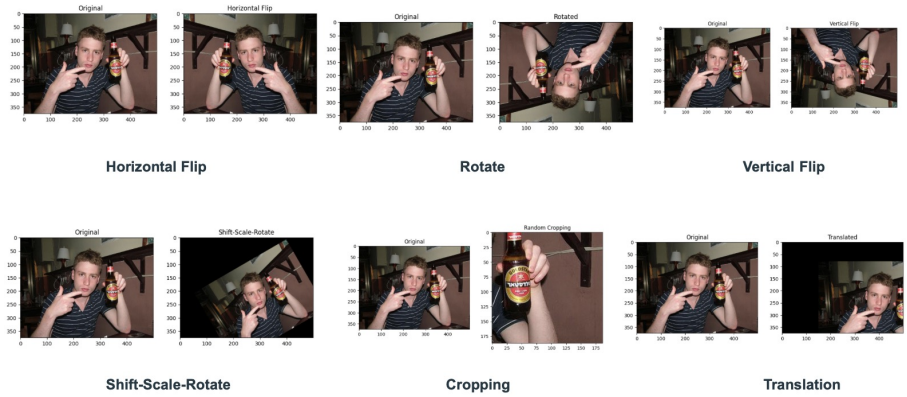


Figure 4.4: Geometric Transformations

Color and Contrast Enhancements: To simulate diverse lighting, CLAHE enhanced contrast, sharpening highlighted textures, and color jitter introduced variations.

Posterization and solarization further diversified the dataset, helping the model adapt to different environmental conditions.

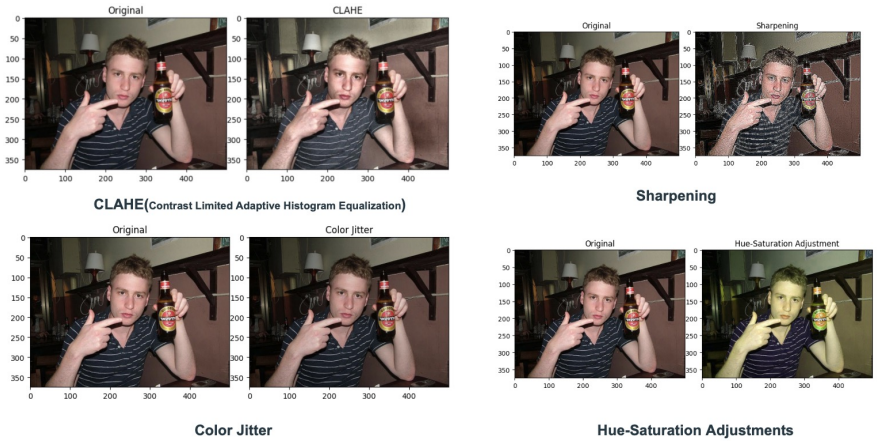


Figure 4.5: Color Contrast Adjustments

AutoAugment-Based Augmentation: AutoAugment dynamically selected optimal augmentation strategies based on pre-learned ImageNet policies, reducing manual tuning and improving generalization.

4.2.2 Faster-RCNN Pipeline

Below is a structured text-based workflow diagram explaining each stage of the Faster R-CNN pipeline when using ResNet50 as the backbone for hand bounding box detection.

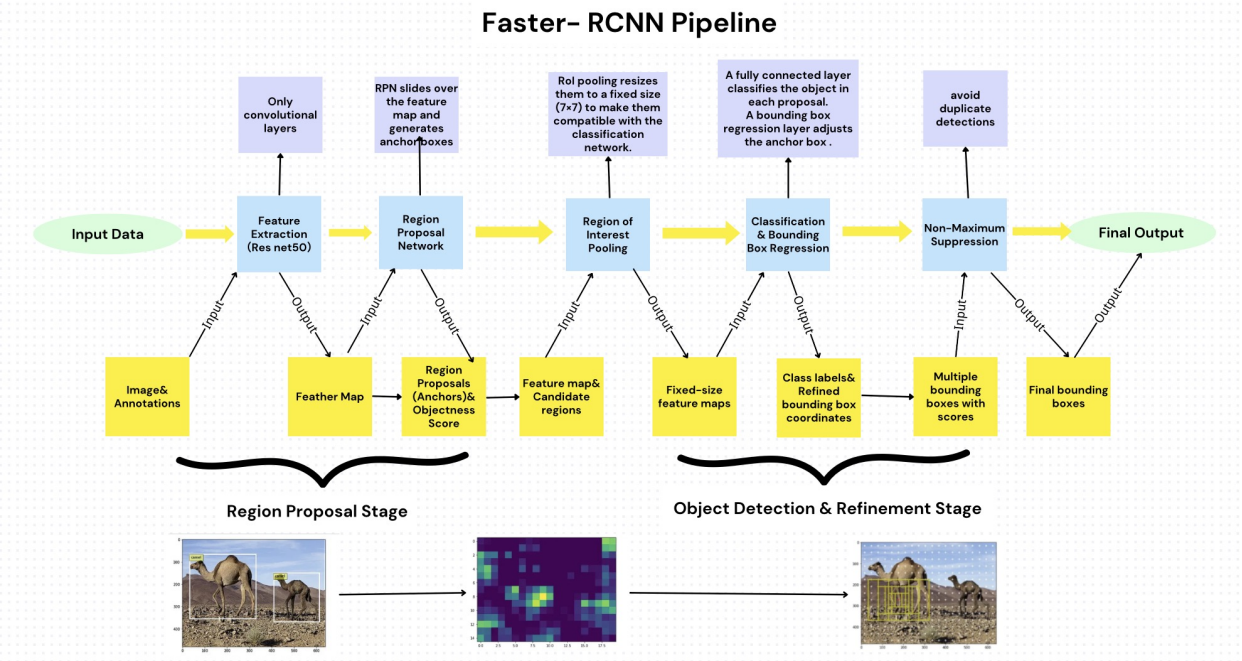


Figure 4.6: Faster-RCNN Framework

This section details the pipeline for hand detection using Faster R-CNN with ResNet50

as the backbone. The process consists of multiple stages, including feature extraction, region proposal, classification, and bounding box refinement.

4.2.2.1 Input: Image and Annotations

Input: An image (RGB format) and corresponding bounding box annotations in the form of $(class\_id, x_1, y_1, x_2, y_2)$.

Output: The image is prepared for feature extraction, and annotations serve as ground truth labels for training.

Function and Reason: The image is passed into the model for hand detection, while annotations enable supervised learning by providing reference bounding boxes.

4.2.2.2 Feature Extraction with ResNet50 (Backbone)

Input: Raw image from Step 1.

Output: A feature map of 2048 dimensions representing extracted spatial and semantic features.

Function and Reason: ResNet50 captures high-level semantic information while retaining spatial structure. Fully connected layers are removed to ensure spatial consistency, which is crucial for object detection.

4.2.2.3 Region Proposal Network (RPN)

Input: Feature map from ResNet50.

Output: Region proposals (anchors) and objectness scores indicating the likelihood of an object's presence.

Function and Reason: The RPN slides over the feature map, generating anchor boxes of different scales (16, 32, 64, 128, 256) and aspect ratios (0.5, 1.0, 2.0). It eliminates regions with low confidence, refining proposals for further processing.

4.2.2.4 RoI Pooling (Region of Interest Pooling)

Input: Feature map and region proposals from RPN.

Output: Fixed-size feature maps of 7×7 for each proposal.

Function and Reason: Since object proposals vary in size, RoI Pooling resizes them to a fixed shape, ensuring uniform input dimensions for classification and bounding box regression.

4.2.2.5 Detection Head (Classification and Bounding Box Regression)

Input: Fixed-size RoI feature maps.

Output: Class labels and refined bounding box coordinates (x_1, y_1, x_2, y_2).

Function and Reason: A fully connected network classifies each proposal as a hand or background, while a regression network fine-tunes bounding box coordinates for accurate localization.

4.2.2.6 Non-Maximum Suppression (NMS)

Input: Multiple bounding boxes with confidence scores.

Output: Final set of bounding boxes with minimal redundancy.

Function and Reason: NMS removes duplicate detections by retaining the highest-confidence bounding box for each hand, preventing overlapping predictions.

4.2.2.7 Final Output: Detected Hand with Bounding Box

Input: Refined bounding boxes post-NMS.

Output: Final hand detection results with class labels and bounding box coordinates.

Function and Reason: This final output can be utilized for downstream applications such as gesture recognition, augmented reality (AR), or sign language interpretation.

4.2.2.8 Summary

This pipeline enables efficient hand detection by:

- Extracting high-quality features via ResNet50.
- Generating and refining region proposals using the RPN.
- Performing classification and bounding box adjustment.
- Eliminating duplicate detections with NMS.

By following this structured approach, Faster R-CNN with ResNet50 provides a robust method for hand bounding box detection, ensuring accurate and reliable results.

4.2.3 YOLO Pipeline

YOLO (You Only Look Once) is a single-shot object detection framework that directly predicts object bounding boxes and class probabilities from an image in one forward pass. This section details the key steps in the YOLO detection pipeline.

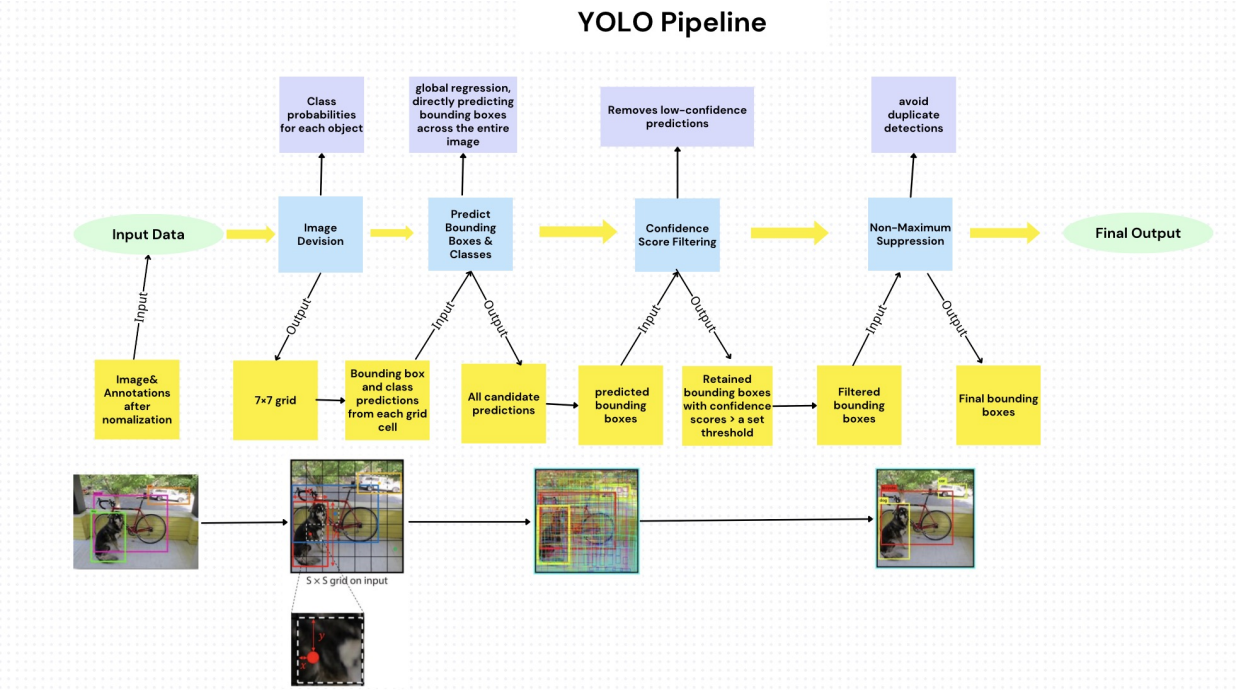


Figure 4.7: YOLO Pipeline

It divides the input image into a grid, predicts bounding boxes and class probabilities in a single forward pass, filters low-confidence detections, applies Non-Maximum Suppression (NMS) to remove duplicates, and outputs the final detected objects.

4.2.3.1 Input Data

Input: Raw image (RGB format) and annotation file with bounding box labels in the form:

$$\text{Class\_ID } x_{\text{center}} y_{\text{center}} wh \tag{4.1}$$

where $(x_{\text{center}}, y_{\text{center}})$ denotes the normalized center coordinates of the object, and (w, h) represents the normalized size of the bounding box.

Output: Preprocessed image (resized, normalized) ready for model input.

Function and Reason: The input image is resized to a fixed dimension (e.g., 416×416 or 640×640) to standardize model input. Bounding box coordinates are normalized to improve training stability.

4.2.3.2 Grid-Based Prediction

Input: Preprocessed image.

Output: An $S \times S$ grid (e.g., 7×7 or 13×13), where each grid cell predicts:

- B bounding boxes (typically $B = 3$ in YOLOv3).
- Each bounding box contains: center coordinates (x, y), width (w), height (h), an objectness score, and class probabilities.

Function and Reason: Dividing the image into a grid enables YOLO to perform object detection in a single forward pass, greatly improving computational efficiency.

4.2.3.3 Bounding Box and Class Prediction

Input: Predictions from each grid cell.

Output: Candidate predictions in the form:

$$(x, y, w, h, \text{objectness\_score}, \text{class\_probs}) \tag{4.2}$$

Function and Reason: YOLO uses a **global regression approach**, predicting bounding boxes and class scores for the entire image simultaneously rather than relying on region proposal networks (RPNs), which speeds up detection.

4.2.3.4 Confidence Score Filtering

Input: Predicted bounding boxes.

Output: Bounding boxes with confidence scores exceeding a predefined threshold (e.g., 0.5).

Function and Reason: Removing low-confidence detections improves accuracy and prevents false positives.

4.2.3.5 Non-Maximum Suppression (NMS)

Input: Filtered bounding boxes.

Output: Optimal bounding boxes.

Function and Reason: If multiple bounding boxes highly overlap (Intersection over Union (IoU) threshold > 0.5), only the one with the highest confidence score is retained, preventing duplicate detections.

4.2.3.6 Final Detection Output

Input: Bounding boxes after NMS.

Output: Final detected objects and their bounding boxes.

Function and Reason: The final output consists of accurately detected objects, ready for downstream applications such as gesture recognition, augmented reality (AR), or autonomous driving.

4.2.3.7 Summary

YOLO enables real-time object detection with high efficiency by:

- Predicting bounding boxes and class labels in a single forward pass.
- Utilizing a grid-based detection approach for direct regression.
- Applying confidence filtering and NMS to refine detections.

Its end-to-end architecture and high-speed processing make it particularly well-suited for hand detection tasks requiring low latency.

Chapter 5

Experiments and Evaluation

The process of reproducing the code was extremely challenging, primarily due to limitations in my computer's configuration and the lack of access to relevant robotic hardware. Initially, I attempted to reproduce the work using a Mac (M1 chip) and an Ubuntu system based on AMD architecture. I spent a significant amount of time on these two setups, encountering persistent issues that left me stuck for weeks. Subsequently, I tried using Google Colab with a GPU, but it lacked support for essential graphical libraries like OpenGL[1]. Ultimately, I successfully completed the reproduction of the core component of the AnyTeleop paper(dex-retargeting[2]) ,which provided open-source code for the community on a Linux system equipped with an Nvidia GPU.

5.1 Environment Setup

5.1.1 Preliminary Configuration

In November, I attempted to reproduce the work by installing VirtualBox and Ubuntu on my Mac, but the reproduction was unsuccessful. The issue arose because my Mac has an M1 chip, while the code was designed to run on an NVIDIA GPU.

```
(base) jintaoma@JintaodeMacBook-Air ~ % system_profiler SPHardwareDataType

Hardware:

    Hardware Overview:

      Model Name: MacBook Air
      Model Identifier: MacBookAir10,1
      Model Number: Z12400189TA/A
      Chip: Apple M1
```

Figure 5.1: Mac_conf

[1] https://www.opengl.org
[2] https://github.com/dexsuite/dex-retargeting

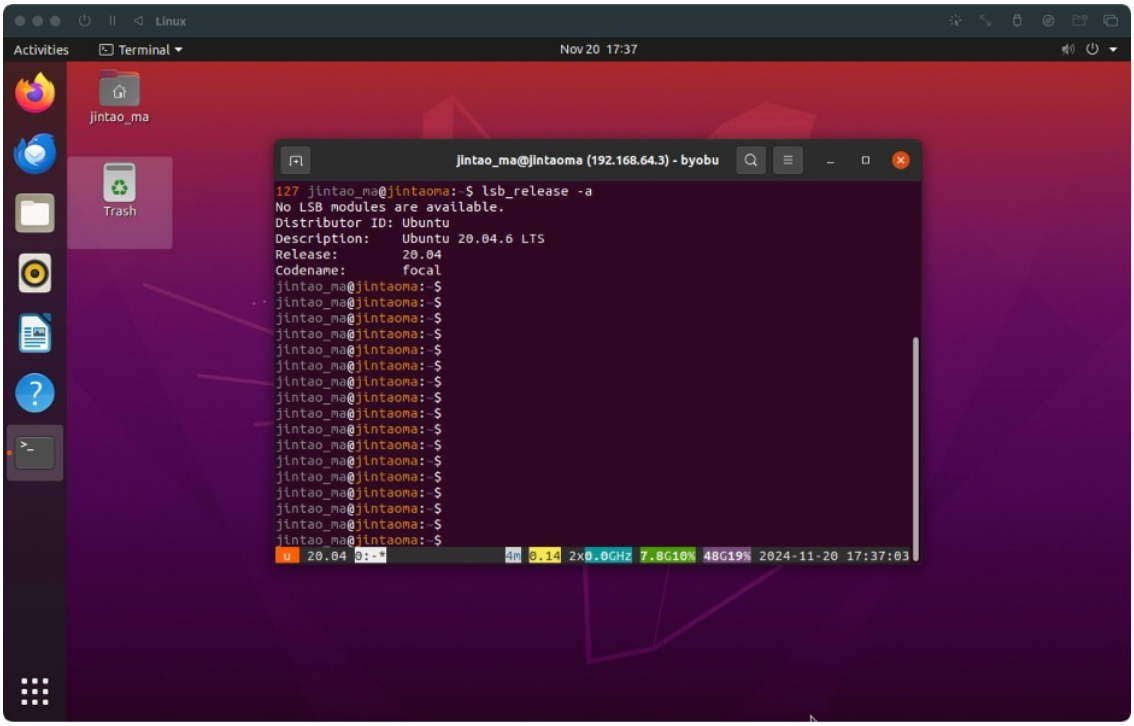


Figure 5.2: Mac_Ubuntu

5.1.2 Final Configuration

In December, I got a computer with an Nvidia GPU, and subsequently, I configured a dual-boot Linux system[3] as outlined below:

```
jintao@jintao-ROG-Zephyrus-G14-GA401QEC-GA401QEC:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.6 LTS
Release:        20.04
Codename:       focal
```

Figure 5.3: Nvidia1

My Linux kernel is running version 5.15.0. Additionally, the GPU is a GeForce RTX 3050[4] with driver version 535.183.01 and supports CUDA 12.2.The GeForce RTX 3050 is a mid-range graphics card from Nvidia, offering real-time ray tracing and AI-based performance enhancements, powered by the Ampere architecture.Therefore, this significantly aided my reproduction work.

[3] https://releases.ubuntu.com/focal/
[4] https://www.nvidia.com/fr-fr/geforce/graphics-cards/30-series/rtx-3050/

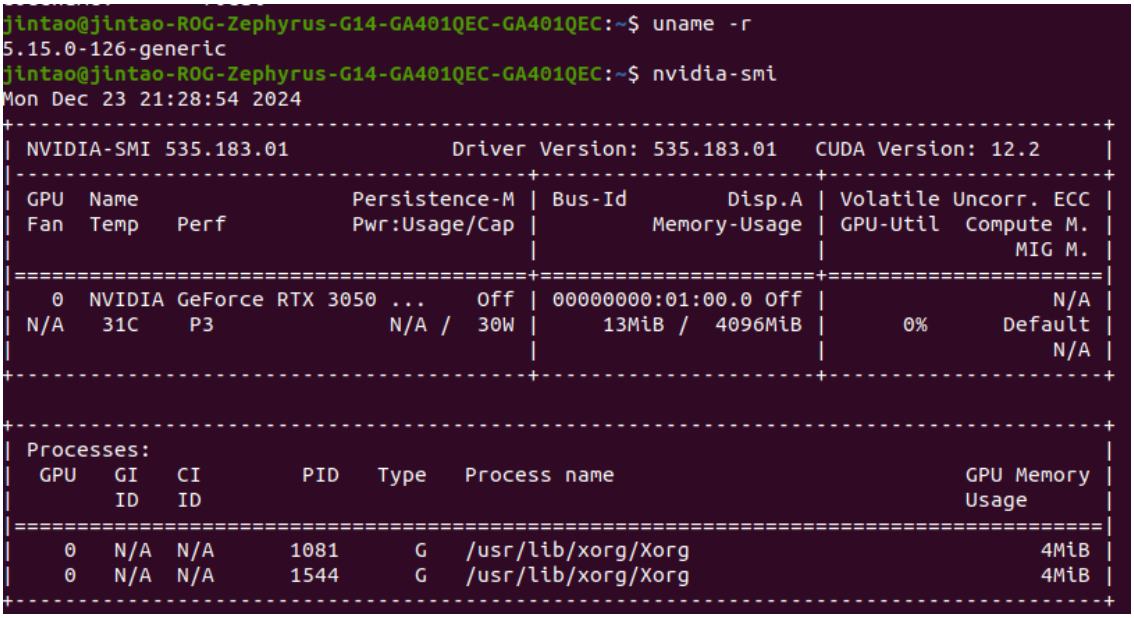


Figure 5.4: Nvidia2

5.2 Simulation Practice Process

At the beginning, my supervisor Guillaume suggested me to get fimilar with some simulation tools like Mujoco or IssacGym.Here, I attempted to use MuJoCo. MuJoCo[5] (Multi-Joint dynamics with Contact) is a physics engine designed for fast and accurate simulation of rigid body dynamics, particularly in robotics and control tasks. It is widely used for simulating complex movements, contacts, and interactions, making it ideal for research in robotics, reinforcement learning, and motion planning.

5.2.1 Exploration Process

5.2.1.1 Collection Methods

Here, I used the Franka Emika Panda robotic[6] arm to grasp a green block and place it at the red target point like figure5.5 and figure5.6. As shown in video[7],This was the initial position. After placing the block in the designated position, the system randomly resets to an initial state. This process is repeated in a loop, allowing the collection of a big dataset of postions. Then I can save it as hdf5 format.

[5]https://mujoco.org
[6]https://github.com/justagist/mujoco_panda
[7]https://drive.google.com/file/d/1MBqAYyYwkfifOJh8bGKUl1m4qfe9XVB3/view?usp=sharing

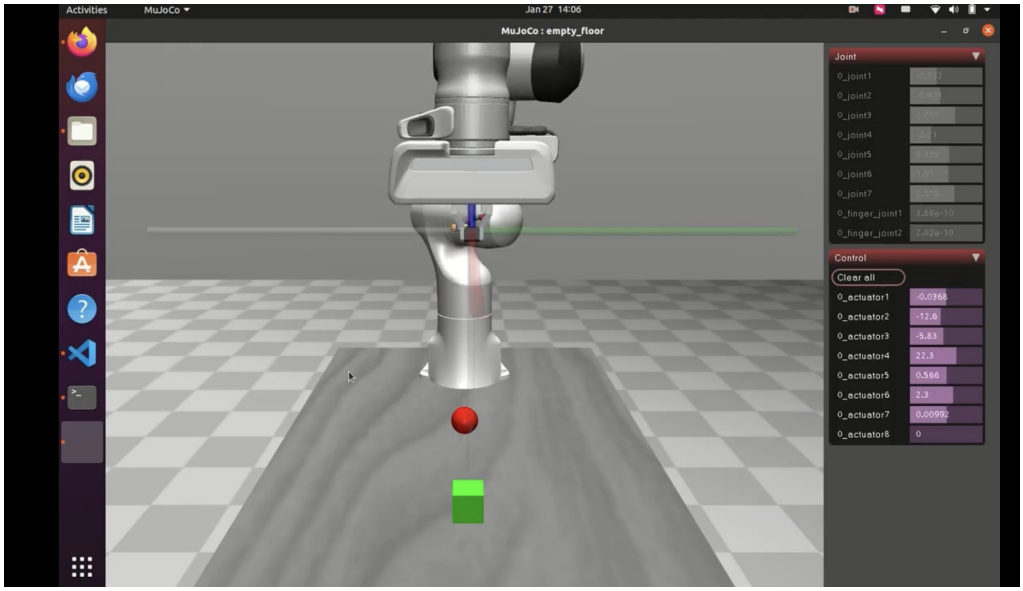


Figure 5.5: Initial position

5.2.1.2 Keyboard Control Commands

Here I used set of keyboard commands that allow control of the robotic arm's movement and operation.

Move End Effector along the x/y Axes:

Key: Arrow keys (↑ ↓ ← →)

Function: Moves the robotic arm on the horizontal plane (x/y axes).

Move End Effector along the z Axis

Key: CTRL + Arrow keys (↑ ↓)

Function: Moves the robotic arm up or down along the vertical (z) axis.

Rotate End Effector around the x/y Axes

Key: SHIFT + Arrow keys (↑ ↓ ← →)

Function: Rotates the end effector around the x or y axis.

Rotate End Effector around the z Axis

Key: CTRL + SHIFT + Arrow keys (↑ ↓)

Function: Rotates the end effector around the z axis.

Open/Close Gripper

Key: CAPSLOCK

Function: Opens or closes the robotic gripper to grasp or release objects.

Switch Control Agent (Change Active Robotic Arm)

Key: ALT

Function: Switches between different robotic arms in a multi-arm environment.

Exit Operation

Key: ESC

Function: Exits the current interactive operation and terminates the simulation.

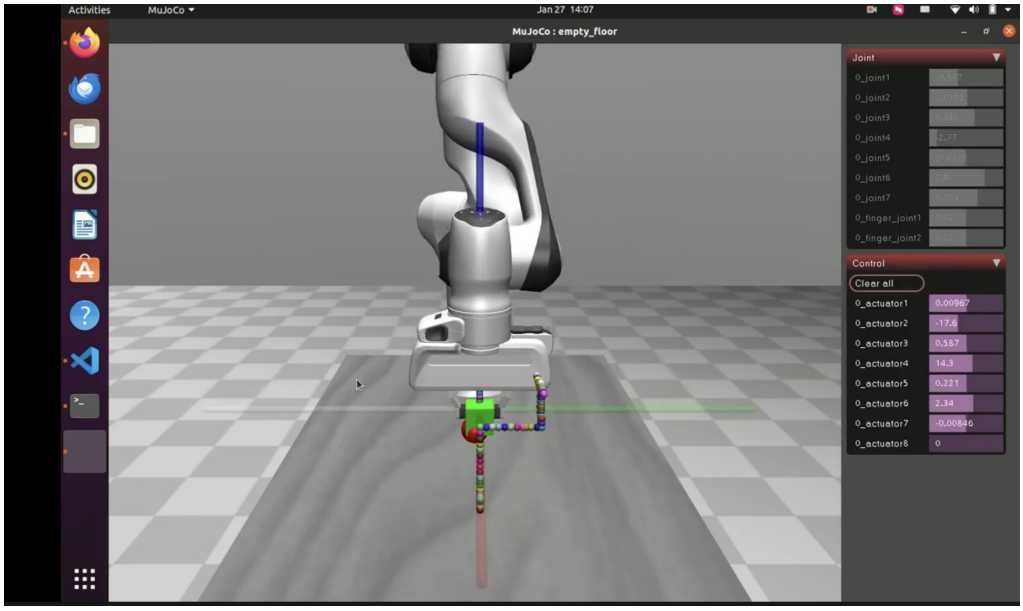


Figure 5.6: Final position

5.3 Reproduction Process

First, I set up and created a virtual environment,

```
python3 -m venv venv
```

then I activated the virtual environment.

```
source venv/bin/activate
```

After that, I installed the required dependencies for the project using the following command:

```
pip install -e ".[example]"
```

5.3.1 Simulation Tool

Here I used SAPIEN[8]. It is a high-fidelity simulation environment for robot manipulation and interaction, offering real-time physics, realistic rendering, and easy integration with machine learning algorithms.

5.3.2 Pre-recorded Video Retargeting

I ran the 'detect_from_video.py' script to detect hand poses from the video [9] which was recorded before and retarget them to the robot. This script generated the robot's joint trajectory and saved it as a pickle file.

[8] https://sapien.ucsd.edu
[9] https://drive.google.com/file/d/1C0_AoNF6GTkLimsCIIYhgZHGClV3q7Km/view?usp=sharing

```
python3 detect_from_video.py \
  --robot-name allegro \
  --video-path data/human_hand_video.mp4 \
  --retargeting-type dexpilot \
  --hand-type right \
  --output-path data/allegro_joints.pkl
```

Figure 5.7: Pre-recorded Video

I ran the ‘render_robot_hand.py‘ script to visualize the results. This script uses the pickle file generated in the previous step (‘data/allegro_joints.pkl‘) and produces a rendered video of the robot's hand (‘data/allegro.mp4‘[10]).

```
python3 render_robot_hand.py \
  --pickle-path data/allegro_joints.pkl \
  --output-video-path data/allegro.mp4 \
  --headless
```

[10] https://drive.google.com/file/d/1DE6nNW_nbn92NFw1TIzFMhz5JzwOdwah/view?usp=sharing

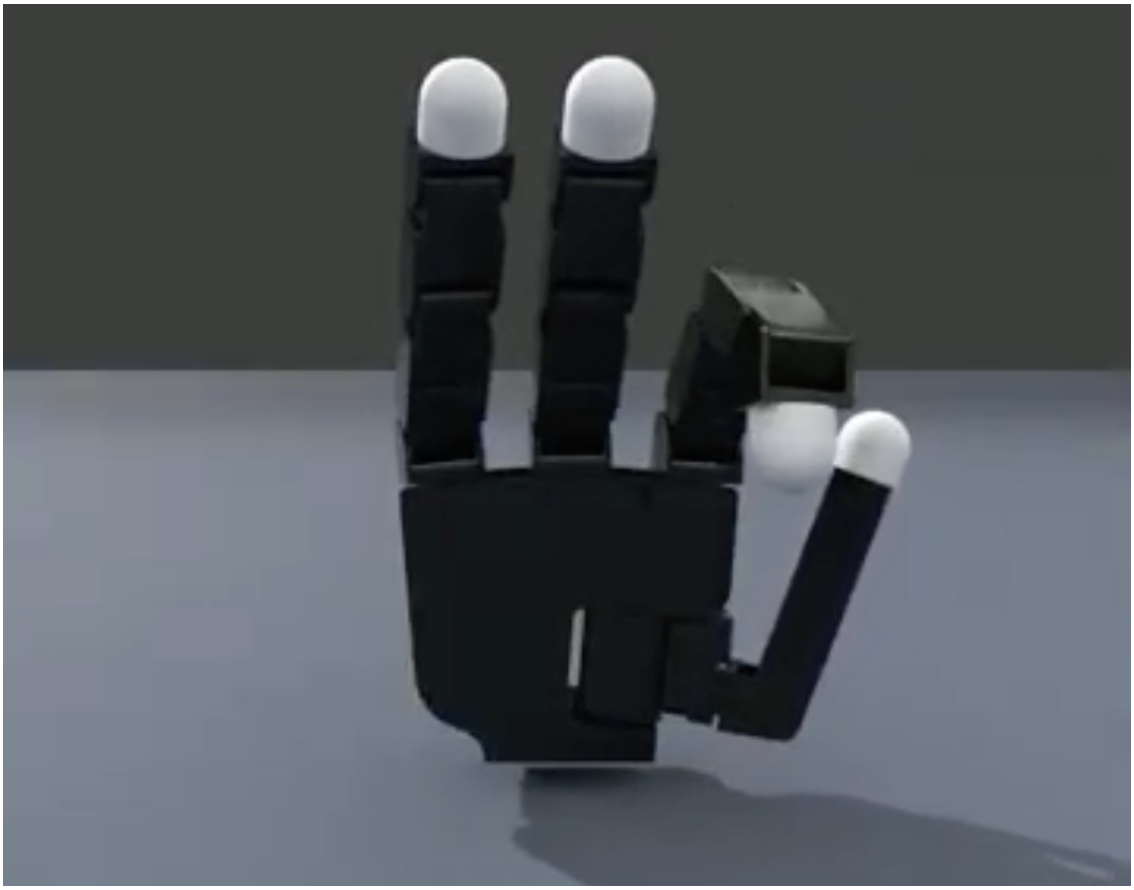


Figure 5.8: allegro

The Allegro hand[11] is a dexterous robotic hand designed with four fingers, which differs from the human hand in its design. Specifically, the thumb of the Allegro hand typically maps one-to-one with the human thumb, but the other three fingers of the Allegro hand (without a pinky) collectively perform actions similar to the four fingers of the human hand. In this case, the motion of the human pinky is often disregarded.

5.3.3 Real-time Mode Retargeting

I ran the real-time hand pose retargeting script, which captured hand movements through the network camera, redirected the hand poses to the robot in real-time, and displayed the visualized results[12] in the SAPIEN viewer.These figure presented the results of hand pose retargeting for 7 gestures. These 7 poses corresponded to the numbers 1 through 7, respectively.I aslo tried different kinds of hands liks :SVH[13] and Shadow[14] and left hand.

[11] https://www.allegrohand.com
[12] https://drive.google.com/file/d/1tnQRBRV70f7sH1B5BF6acbjSkgz5Q01Z/view?usp=sharing
[13] https://schunk.com/de/en/gripping-systems/special-gripper/svh/c/PGR_3161
[14] https://www.shadowrobot.com/dexterous-hand-series/

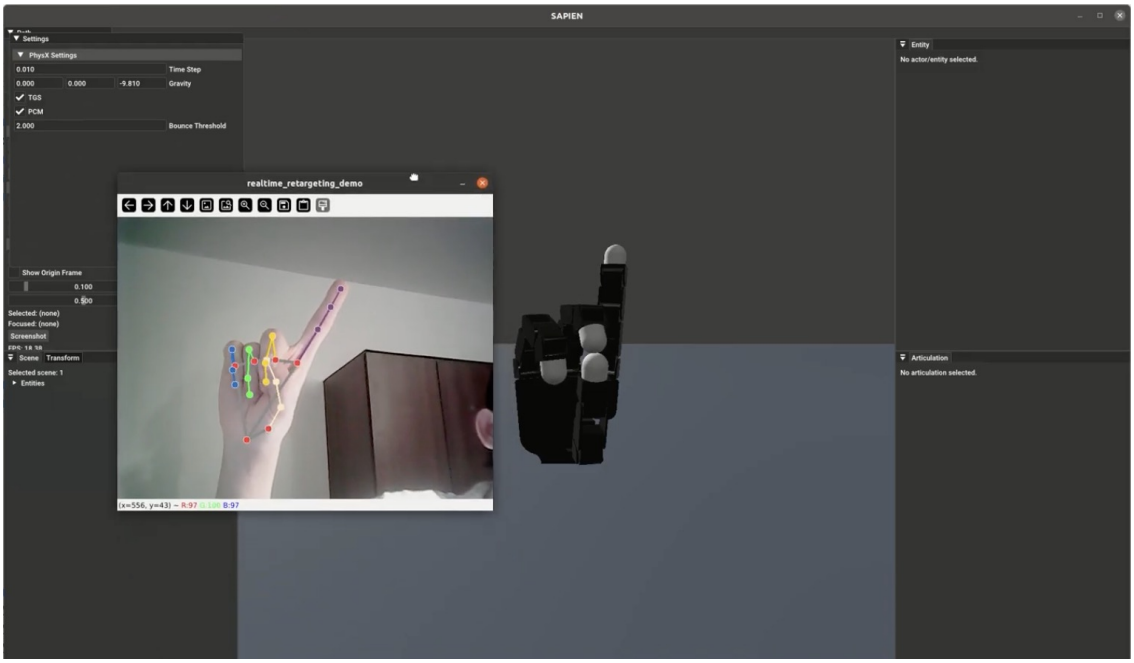


Figure 5.9: Pose1

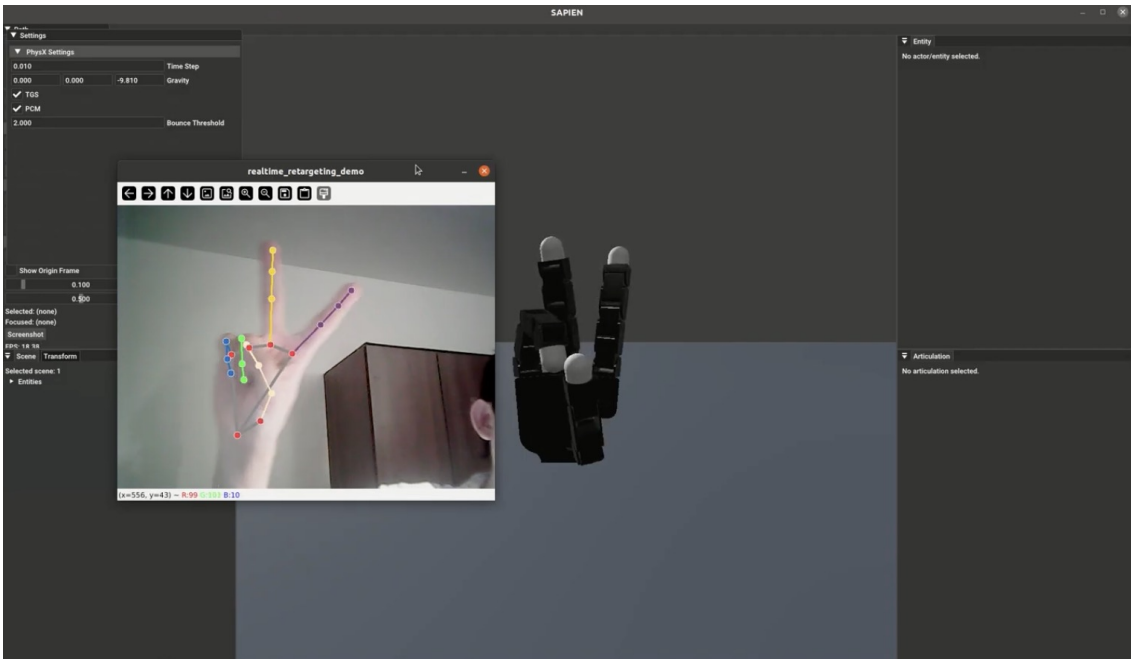


Figure 5.10: Pose2

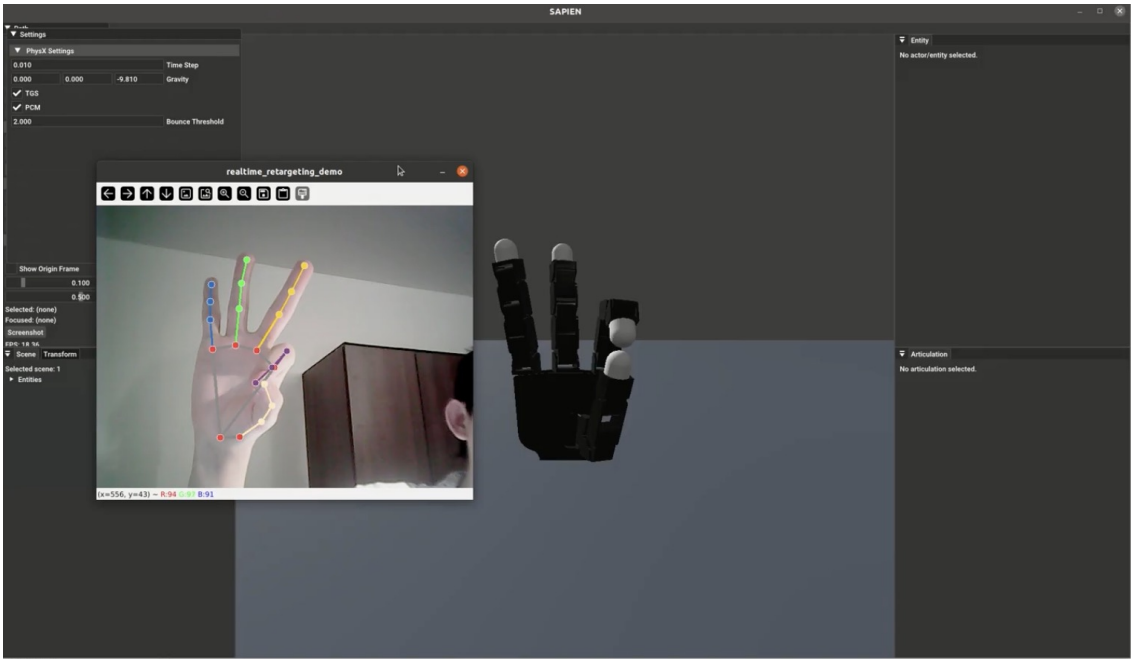


Figure 5.11: Pose3

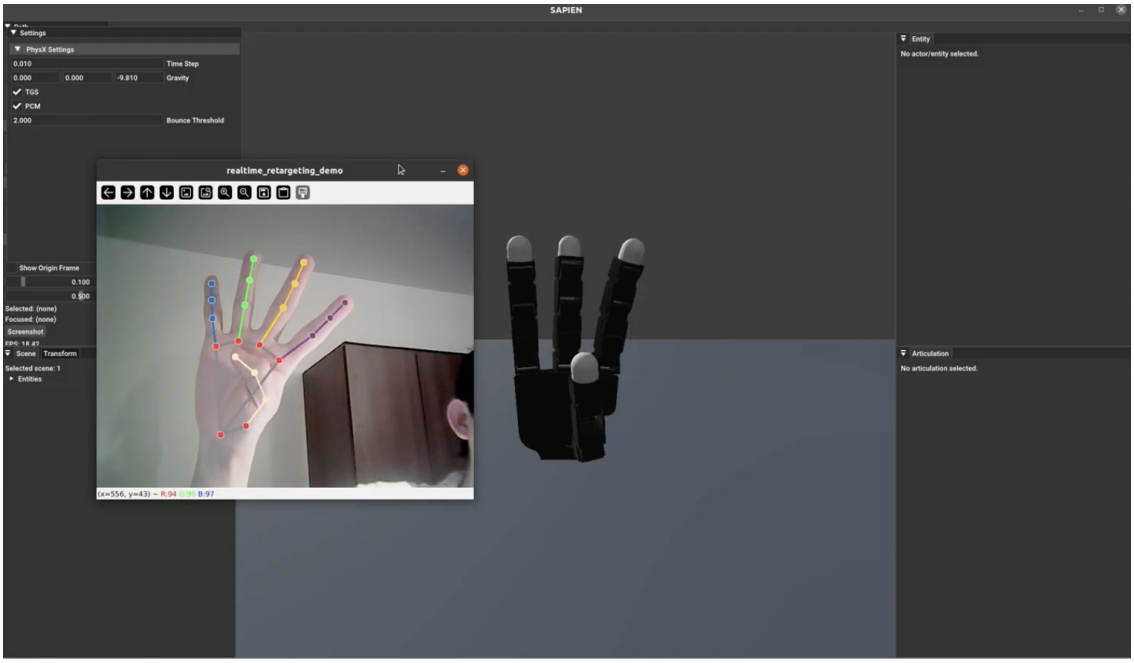


Figure 5.12: Pose4

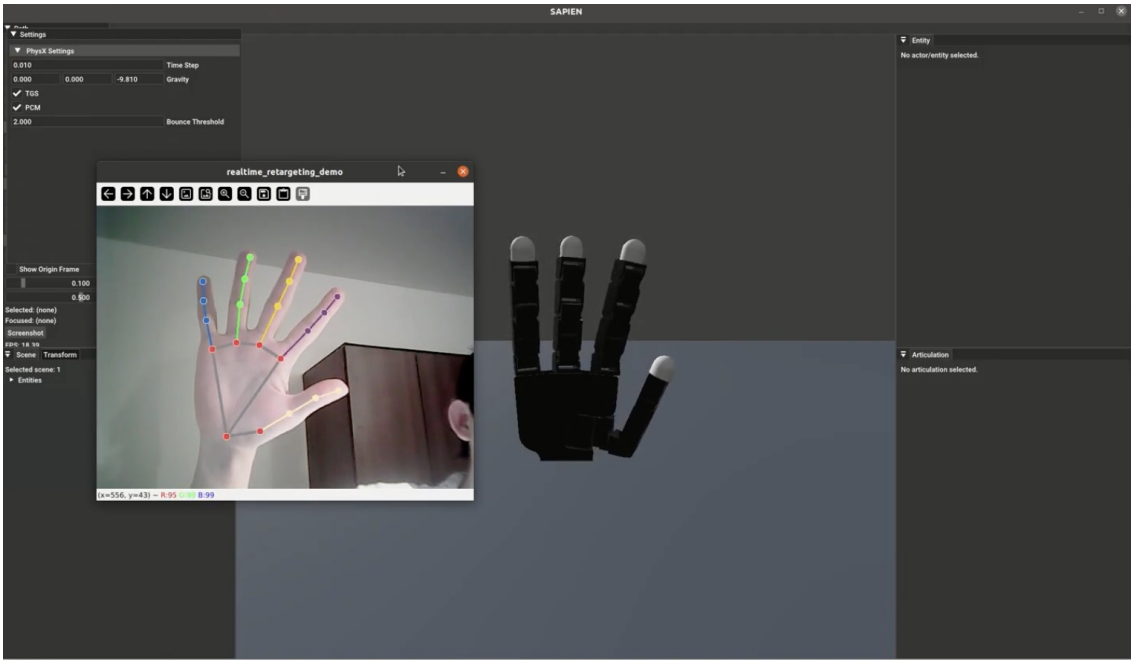


Figure 5.13: Pose5

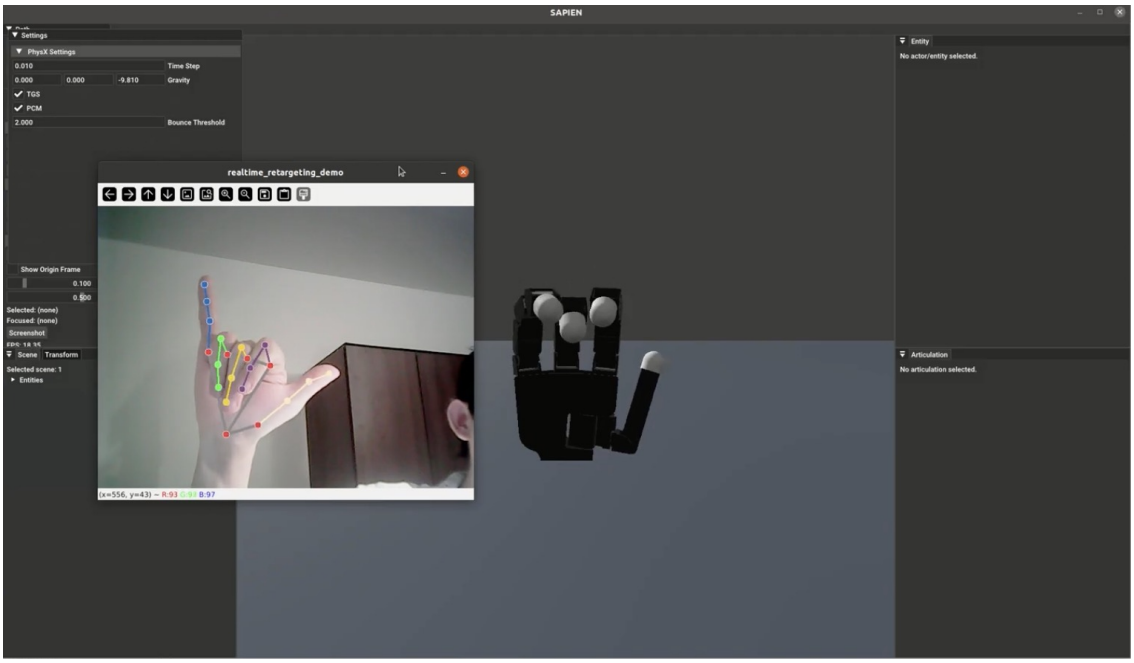


Figure 5.14: Pose6

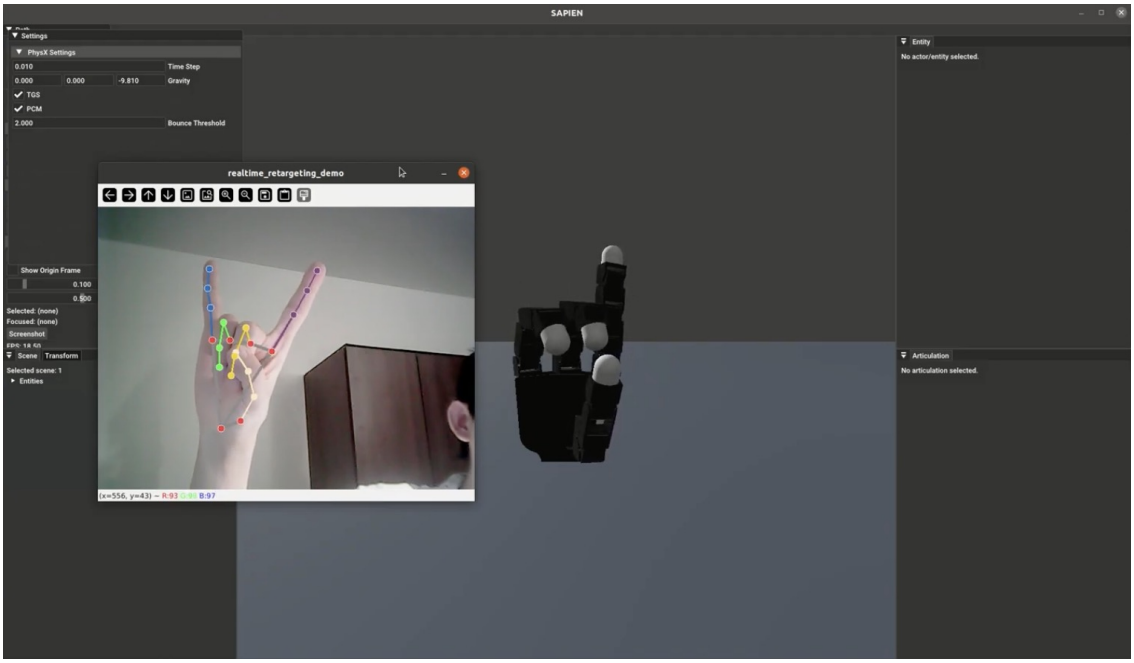


Figure 5.15: Pose7

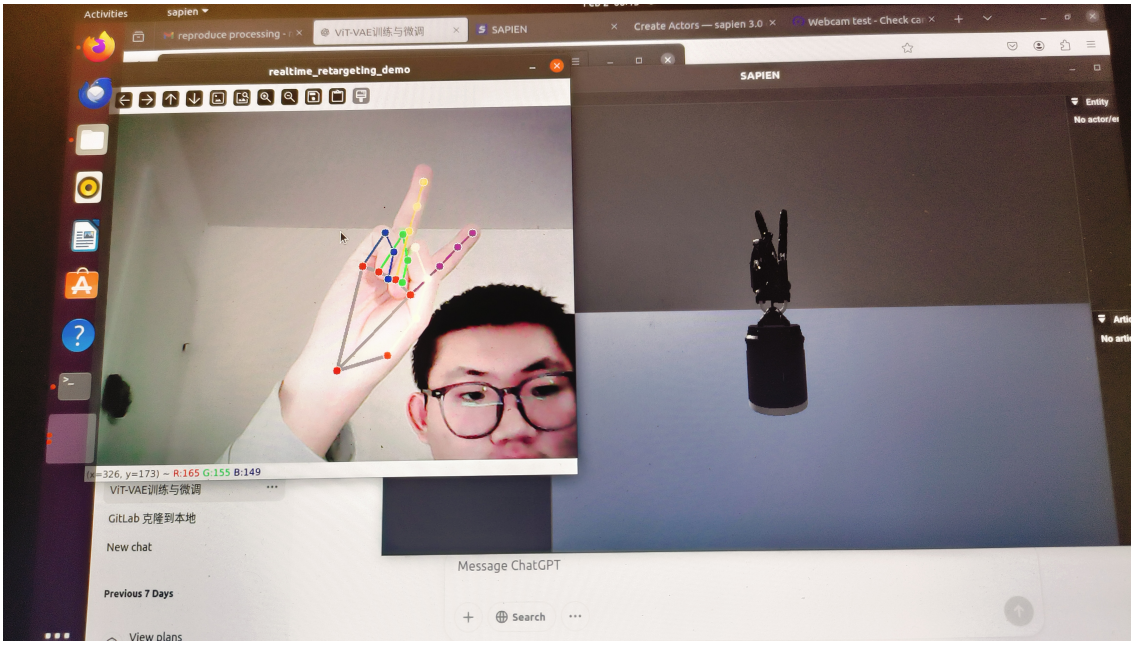


Figure 5.16: shadow

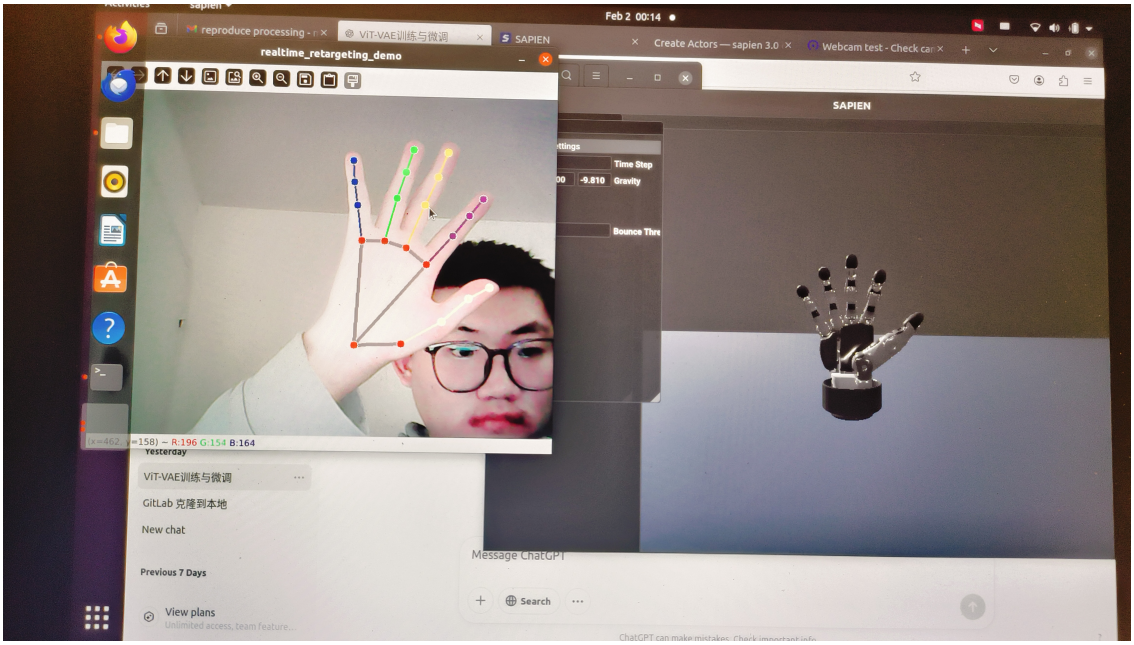


Figure 5.17: svh_right

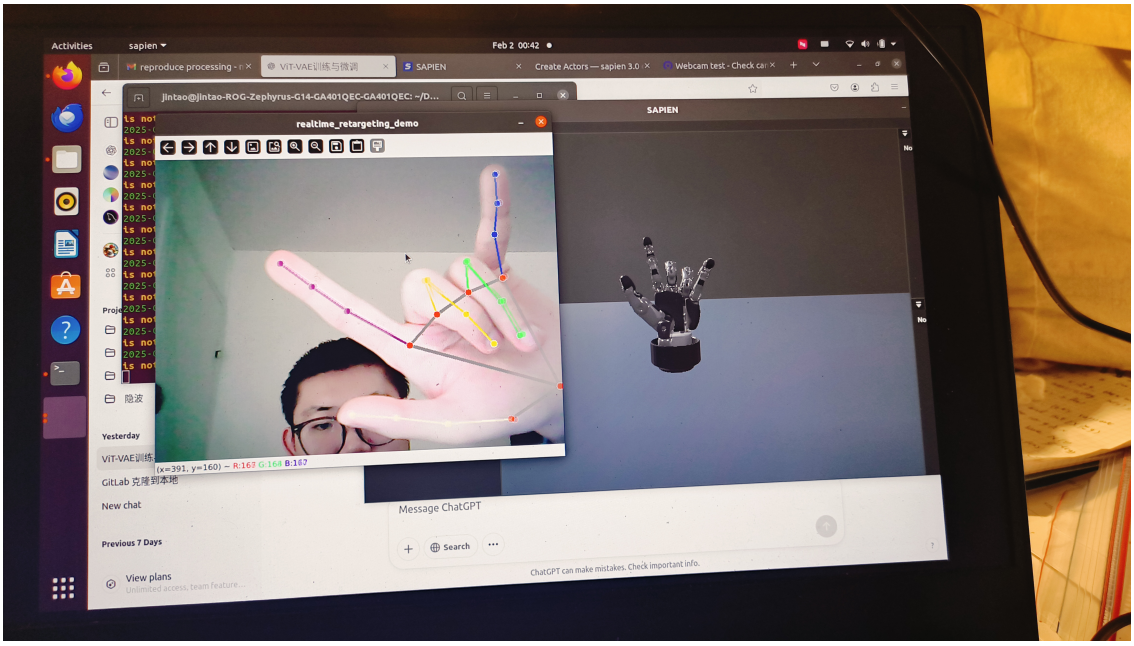


Figure 5.18: svh_left

5.3.4 Issue for data collection

The most biggest issue was system stability.My original plan was to move the green block to the red dot from different positions within the AnyTeleop system, repeatedly collecting data 100 times as Figure 5.19 shown. If I attempt retargeting multiple times or run the real-time retargeting program for an extended period (e.g., 2-3 minutes), the system eventually failed to capture images from the camera. This could be due to the fact

that the computer I rented does not have a built-in camera, and I am using an external USB camera.This is an objective limitation of the hardware infrastructure.This hardware limitation is beyond my control. I recorded a video showing the issue[15] .

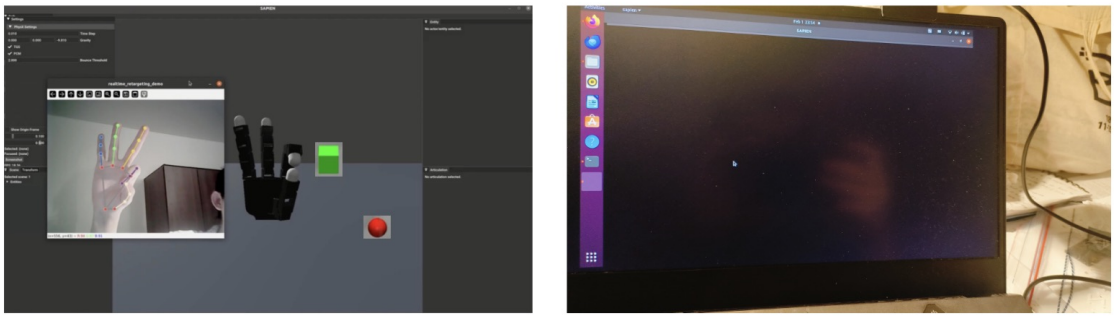


Figure 5.19: Collapsed systems

If I need to collect at least 100 data samples using a teleoperated robotic hand, the process would take a long time, with the camera failures and crashes.

5.4 Hand Bounding Box Detection

In this section, I evaluated the performance of hand bounding box detection models by comparing key metrics and analyzing their strengths and weaknesses under different conditions.

5.4.1 Comparison of metrics

5.4.1.1 IoU

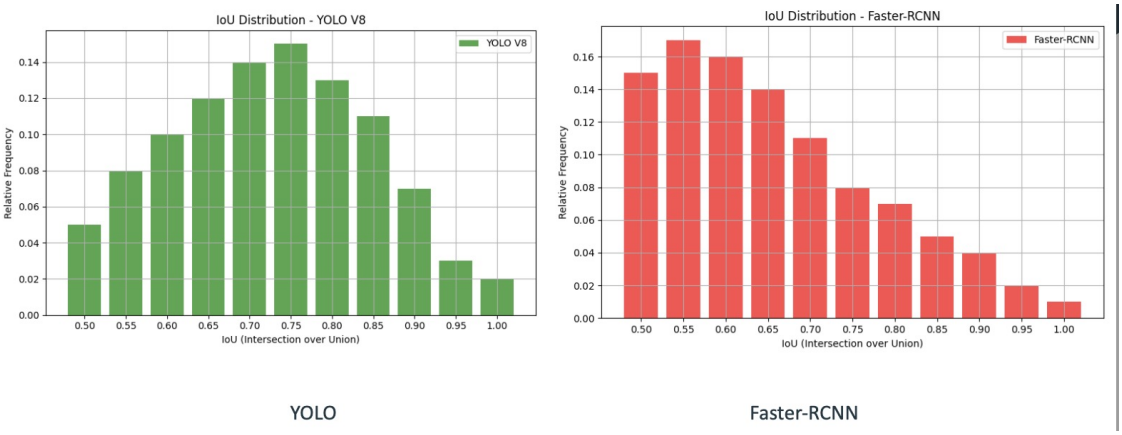


Figure 5.20: IoU Comparison

This IoU distribution graph shows the relative frequency of predicted bounding boxes across different IoU thresholds, indicating how well each model aligns with the ground

[15] https://drive.google.com/file/d/1EKH2HVkvsX4gTH2YLmW7t0AvyIKjL_1A/view?usp=sharing

truth. Faster R-CNN's distribution is more spread out, with a higher concentration in the lower IoU range (0.5-0.7), meaning its predicted boxes are often less precise and more loosely fitted. In contrast, YOLO V8's distribution is more concentrated in the higher IoU range (0.7-0.85), suggesting it produces tighter, more accurate bounding boxes. This difference implies that Faster R-CNN is more prone to false detections and missed detections, while YOLO V8 consistently generates more precise predictions.

5.4.1.2 mAP

Mean Average Precision at 50% IoU (mAP@50) measures the average precision (AP) of a model's predictions when the Intersection over Union (IoU) threshold is set to 0.50, meaning that a detected object is considered a true positive if its predicted bounding box overlaps with the ground truth by at least 50%, whereas mean Average Precision at multiple IoU thresholds (mAP@50-95) calculates the mean AP over a range of IoU thresholds from 0.50 to 0.95 in increments of 0.05, providing a more comprehensive evaluation of the model's performance across different levels of localization accuracy.

Model	mAP@50 (%)	mAP@50-95 (%)
Resnet+Faster-RCNN	63.2	31.7
YOLO V8	86.9	48.2

Table 5.1: Performance comparison of different models

In this comparison, YOLO V8 significantly outperforms ResNet+Faster R-CNN in both mAP@50 (86.9% vs. 63.2%) and mAP@50-95 (48.2% vs. 31.7%). This suggests that YOLO V8 has better overall detection accuracy, particularly in handling various Intersection over Union (IoU) thresholds, as indicated by the mAP@50-95 metric. The improvements likely stem from YOLO V8's advanced architectural enhancements, including better feature fusion, anchor-free detection, and optimized loss functions. While Faster R-CNN can still be advantageous in scenarios requiring high precision and detailed region proposals, YOLO V8's superior performance makes it a preferable choice for real-time object detection tasks.

5.4.1.3 PR curves

The Precision-Recall (PR) curve plots Recall (x-axis) against Precision (y-axis), where Recall measures the proportion of correctly detected objects among all actual objects, and Precision indicates the proportion of correctly detected objects among all detected objects; a higher area under the curve signifies better overall model performance.

It provided further insights into the performance differences between YOLO V8 and Faster R-CNN. Observing that at the same Recall (R), YOLO V8 achieves higher Precision (P) and at the same Precision (P), YOLO V8 achieves higher Recall (R) suggests that YOLO V8 consistently outperforms Faster R-CNN across different operating points.

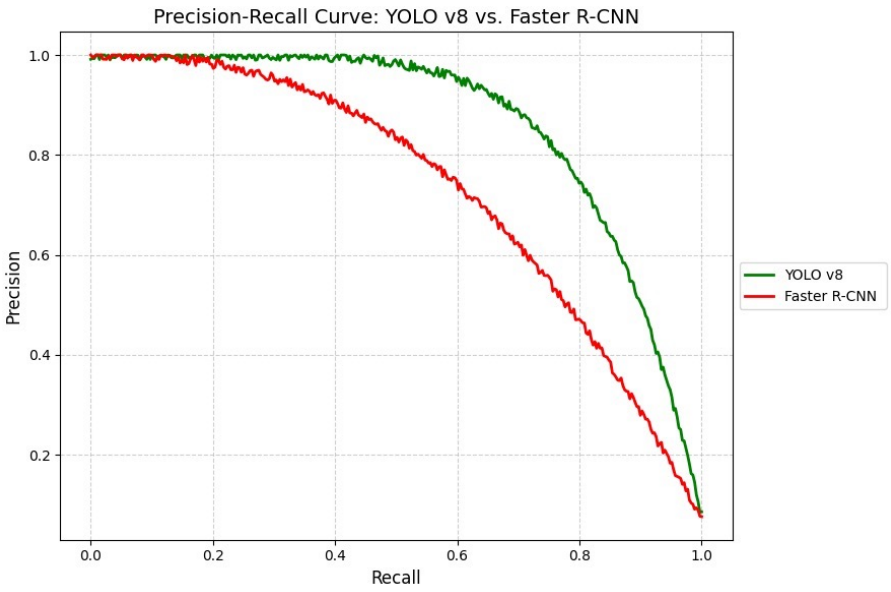


Figure 5.21: PR curves

Since the area under the PR curve corresponds to the mean Average Precision (mAP) as explained in Section 3.7, this directly explains why YOLO V8 has a significantly higher mAP score than Faster R-CNN. The superior PR curve of YOLO V8 indicates that it has a better balance between precision and recall, reducing false positives while still capturing more true positives. This reflects improvements in its detection ability, likely due to YOLO V8's enhanced feature extraction, better spatial understanding, and optimized anchor-free detection mechanism.

5.4.2 YOLO Triumphs Where Faster-Rcnn Stumble

5.4.2.1 False detection

Here I used the green box for the ground truth box and the red box for the detected bounding box after two models inference prediction.As shown in Figure 5.22, the Faster-RCNN incorrectly recognized the pattern of the little boy's blouse as a hand.

Figure 5.22: False detection

Faster R-CNN, as a two-stage detector, first generates region proposals and then classifies them, making it generally more precise than YOLO in controlled settings. However, its complex pipeline can sometimes misclassify background objects as foreground objects due to suboptimal region proposal refinement or feature misalignment. In contrast, YOLO, as a one-stage detector, is designed to minimize false positives by leveraging anchor-free mechanisms and improved feature fusion, resulting in more confident and accurate detections. This explains why, at the same recall, YOLO achieves higher precision than Faster R-CNN, indicating that it has fewer false positives.

5.4.2.2 Missed detection

Here I also used the green box for the ground truth box and the red box for the detected bounding box after two models inference prediction.As shown in Figure 5.23, the Faster-RCNN did not recognize a hand in the shadow of the rightmost man.

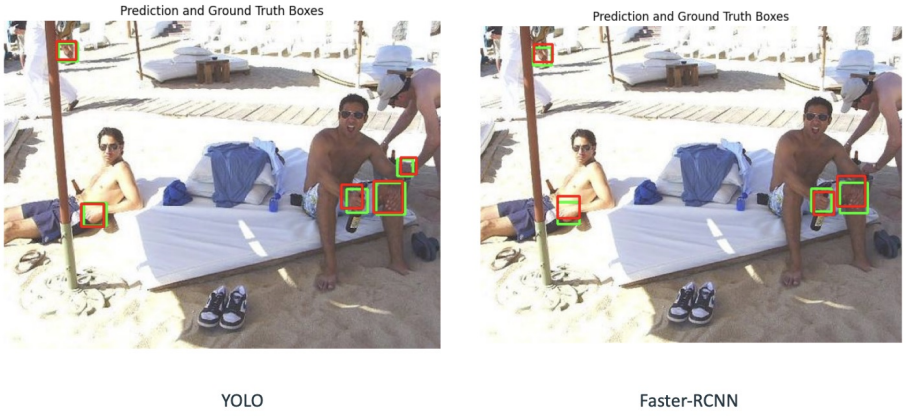


Figure 5.23: Missed detection

Faster R-CNN tends to have higher false negatives (missed detections) compared to YOLO, particularly in real-time scenarios. Since it relies on a region proposal network (RPN), it may fail to generate proposals for small or occluded objects, leading to missed detections. Additionally, its slower inference speed means it might struggle in fast-moving scenes, where objects may not be effectively captured. On the other hand, YOLO's dense prediction grid and end-to-end optimization allow it to detect objects with higher recall, which is why at the same precision, YOLO achieves higher recall than Faster R-CNN, demonstrating fewer false negatives.

5.4.3 The Impact of Data Augmentation on Model Performance

After data augmentation, I observed an improvement in the mAP of both Faster R-CNN and YOLO. Notably, the improvement was more significant for Faster R-CNN. This indicated that data augmentation had a positive impact on the performance of both models, especially given the extremely limited size of my training dataset.

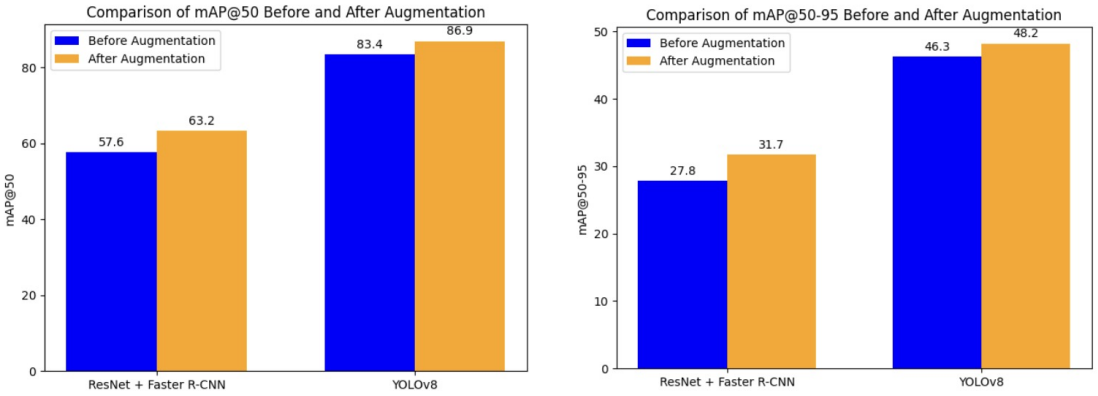


Figure 5.24: The Impact of Data Augmentation on Model Performance

Data augmentation benefits Faster R-CNN more significantly because its two-stage architecture relies on region proposals (RPN), which require diverse training samples to improve robustness. Enhancing data variety helps RPN generate better candidate boxes, leading to more accurate detections, especially for small or occluded objects. Additionally, Faster R-CNN is more prone to overfitting on small datasets, making augmentation essential for improving generalization.

YOLOv8, being a one-stage detector, is less dependent on data augmentation as it already incorporates built-in optimizations and Augmentation. Its design inherently provides strong generalization, making it effective even with minimal augmentation. While data augmentation still improves performance, the gain is smaller compared to Faster R-CNN, as YOLOv8 is already optimized for efficient feature extraction and localization.

Chapter 6

Conclusion and Perspectives

6.1 Conclusion

During the past months. I firstly got familiar with some simulation tools and succefully reproduced AnyTeleop and attempted data collection; however, teleoperation proved to be an engineer-intensive task with strong hardware dependencies. After discussion with my supervisor Guillaume, I focused on a smaller aspect—hand detection—by comparing the performance of two object detection models.

6.1.1 Work Timeline

Here I also listed the timeline of my work as figure6.1 shown.

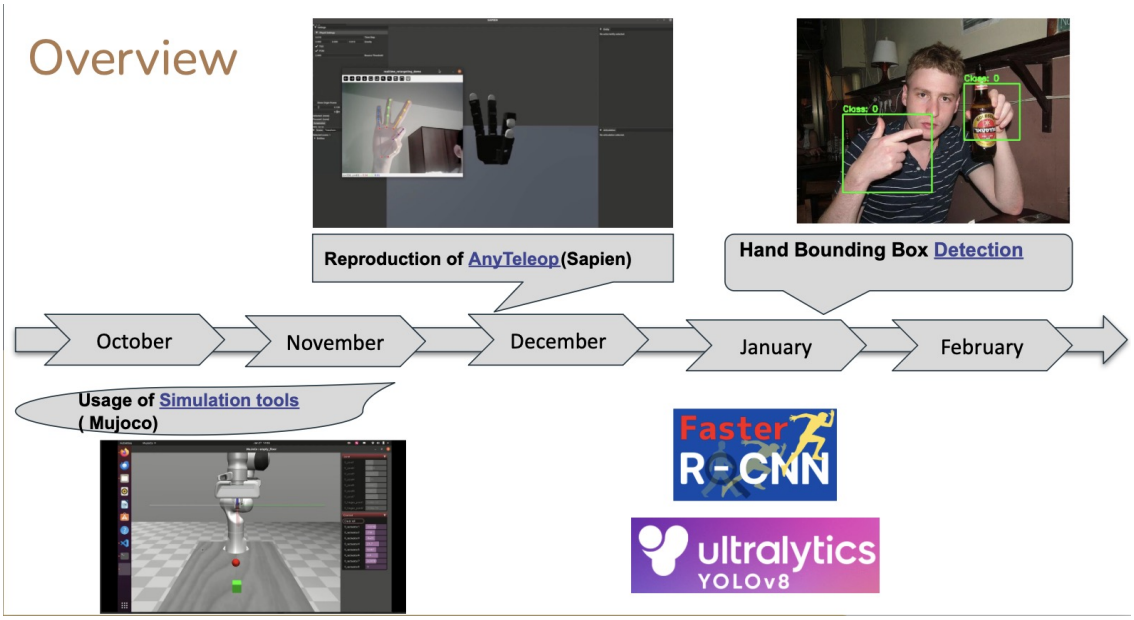


Figure 6.1: Work Timeline

6.1.2 Reproduction of AnyTeleop

The reproduction of the AnyTeleop framework was successfully carried out, with the key steps of hand pose detection, retargeting, and visualization being effectively implemented. Using an Nvidia GPU and a dual-boot Linux system with a GeForce RTX 3050, the necessary computational power was provided to run the required simulation tools and scripts. The SAPIEN simulation environment was utilized to generate realistic interactions between the human and robotic hands. By running the detect_from_video.py and render_robot_hand.py scripts, the robot's joint trajectory was successfully generated from

pre-recorded human hand videos and visualized using the rendered video output. In addition, the real-time hand pose retargeting script was executed to demonstrate real-time manipulation, with the results displayed in the SAPIEN viewer.

This process facilitated the accurate translation of human hand gestures to robotic hand motions, showcasing the potential of the AnyTeleop framework for vision-based teleoperation.

6.1.3 Hand Bounding Box Detection

Comparison	**YOLOv8**	**Faster R-CNN**
Proposed Time	2015	2023
Detection Framework	One-stage detection	Two-stage detection
Optimization Strategy	End-to-end optimization	Region proposal-based optimization
Feature Extraction	Efficient and lightweight	Relies on its backbone
Test on Small Datasets	Highly effective	Struggles due to proposal-based mechanism
Region Proposal Mechanism	No region proposals (direct regression)	Relies on region proposals
Computational Efficiency	Requires fewer computational resources	Higher computational cost

Table 6.1: Comparison of YOLOv8 and Faster R-CNN

YOLOv8 demonstrates superior performance compared to Faster R-CNN due to its one-stage detection framework, which allows direct regression of bounding boxes without requiring a separate region proposal step. This end-to-end optimization strategy makes YOLOv8 significantly faster and more computationally efficient.

In contrast, Faster R-CNN adopts a two-stage detection approach, where the Region Proposal Network (RPN) first generates candidate regions before classification and box refinement. While this method can improve accuracy in well-annotated and large-scale datasets, it struggles with small datasets because optimizing the candidate boxes requires a substantial amount of training data.

Additionally, YOLOv8 employs an efficient and lightweight feature extraction mechanism, enabling real-time inference with fewer computational resources. Faster R-CNN, on the other hand, heavily depends on its backbone for feature extraction, leading to higher computational costs. The absence of region proposal mechanisms in YOLOv8 further enhances its efficiency, making it a preferred choice for real-time applications such as autonomous driving and surveillance systems.

6.2 Limitations

Despite the success of the reproduction and detection work, there were several limitations encountered during the process:

6.2.1 Hardware Constraints:

Due to the inherent limitations of the computer hardware and camera equipment, I was unable to collect satisfactory data to do the imitation learning task, which is quite regrettable.

6.2.2 Lack of Exploration in Data Augmentation:

While data augmentation is a crucial technique for improving model generalization and robustness, this work did not thoroughly investigate its impact. I did not analyze which augmentation strategies were beneficial and which might have been detrimental to performance. This limitation means that potential improvements in model accuracy and robustness were not fully explored.

6.2.3 Prolonged Training Time:

I found that training Faster R-CNN took at least 25 minutes per epoch on average, which was extremely time-consuming. In contrast, YOLO required only about 50 seconds per epoch. The training process for Faster R-CNN still needs further optimization.

6.2.4 Lack of Transformer-Based Models:

The Faster R-CNN and YOLO models used here are both based on CNN architectures. Due to time constraints and the fact that I worked alone, I did not compare them with Transformer-based models.

6.3 Perspectives

While the limitations encountered in this work posed challenges, they also highlight opportunities for future improvements and optimizations:

6.3.1 Improved Hardware Setup:

Upgrading the computing hardware and camera equipment would allow for better data collection, reducing constraints related to image quality and processing capabilities. Future work should consider leveraging high-performance GPUs and higher-resolution cameras to improve the dataset quality.

6.3.2 Systematic Evaluation of Data Augmentation:

Future work should include a systematic study of different data augmentation techniques to assess their impact on model performance. Experiments should be conducted to determine which augmentations enhance detection accuracy and which may introduce noise

or degrade performance. Techniques such as rotation, scaling, brightness adjustment, and synthetic data generation could be evaluated to establish a more effective preprocessing pipeline.

6.3.3 Optimizing Training Efficiency:

To address the prolonged training time of Faster R-CNN, future efforts should explore techniques such as model pruning, mixed-precision training, or using more efficient backbone networks. Additionally, hyperparameter tuning and distributed training strategies could be employed to improve training speed without sacrificing accuracy.

6.3.4 Incorporating Transformer-Based Models:

To gain a more comprehensive understanding of model performance, future work should include comparisons with Transformer-based models, such as DETR or Swin Transformer, which have demonstrated strong object detection capabilities. This would provide insights into whether modern architectures offer advantages over traditional CNN-based models in the given task.Some hybrid architectures may also improve feature extraction and localization performance

Appendix A

Appendix

For more details on each week's progress, the documents can be accessed at **Jintao's Weekly Progress Record (with Supervisor's Feedback)** on Teams'BDRP_Ma_Araujo' File Folder[1] and my GitHub repository[2].

A.1 Simulation Tool Practice

This appendix described the key components of the **human demonstration collection** script using `robopal`. The script initializes a robotic environment, enables human teleoperation via a Gamepad, and collects demonstrations for imitation learning.

A.1.1 Environment Setup

The environment is created using `robopal.make()`, which initializes a **Pick-and-Place** task with the **Panda robotic arm**. Various parameters define control methods, rendering options, and randomization settings.

Listing A.1: Environment Setup

```
env = robopal.make(
    "PickAndPlace-v1",
    robot="PandaPickAndPlace",
    render_mode='human',
    control_freq=20,
    controller='CARTIK',
    camera_in_window="frontview",
    is_render_camera_offscreen=True,
    is_randomize_end=False,
    is_randomize_object=True,
)
```

Key Configurations:

- `"PickAndPlace-v1"`: Selects the **pick-and-place** task.
- `"PandaPickAndPlace"`: Uses the **Franka Emika Panda** robot.
- `control_freq=20`: Controls frequency of commands.
- `controller='CARTIK'`: Uses **Cartesian Impedance Control (CARTIK)**.
- `is_randomize_object=True`: Randomizes object placement in each episode.

[1] https://drive.google.com/file/d/1yS3ZDzt_QWBXg7xbEHecqk0ybkTOYvET/view?usp=sharing
[2] https://github.com/woshimajintao/BDRP

A.1.2 Human Demonstration Wrapper

The `HumanDemonstrationWrapper` enables human input via a **Gamepad**, allowing teleoperation for collecting demonstrations.

Listing A.2: Human Demonstration Wrapper

```
env = HumanDemonstrationWrapper(
    env,
    device=Gamepad,
    collect_freq=4,
    saved_action_type="position",
    is_render_actions=True,
)
env.device.start()
```

Key Parameters:

- `device=Gamepad`: Uses a **Gamepad** for human input.
- `collect_freq=4`: Collects action data every **4 timesteps**.
- `saved_action_type="position"`: Stores **position-based** actions.
- `is_render_actions=True`: Displays actions on the screen.

A.1.3 Main Loop for Demonstration Collection

The script runs a loop to **continuously collect human demonstrations**, execute actions, and reset when the task is completed.

Listing A.3: Human Demonstration Collection Loop

```
for t in range(int(1e6)):
    action = env.get_action()
    next_obs, reward, termination, truncation, info = env.step(action)

    # Reset the environment if the task is completed
    if env.task_completion_hold_count == 0 or env.device._reset_flag:
        env.reset()

    # Track consecutive success steps
    if info["is_success"]:
        if env.task_completion_hold_count > 0:
            env.task_completion_hold_count -= 1
        else:
            env.task_completion_hold_count = 10
    else:
        env.task_completion_hold_count = -1
```

Key Features:

- `env.get_action()`: **Captures human input** from the Gamepad.
- `env.step(action)`: Executes the action and receives **next state, reward, and termination flags**.

- `env.reset()`: **Resets the environment** when:
  - The task is completed.
  - The user manually resets via the **Gamepad**.
- **Task Completion Check**:
  - Tracks **10 consecutive successful timesteps** to confirm task success.
  - Resets the counter if no success is detected.

A.1.4 Closing the Environment

At the end, `env.close()` ensures proper cleanup.

Listing A.4: Closing the Environment

```
env.close()
```

Purpose:

- Properly **closes the environment**.
- Prevents **memory leaks or rendering issues**.

A.1.5 Summary

This script enables **human teleoperation** for collecting robot demonstrations using a **Gamepad**. The key functionalities include:

- **Setting up the pick-and-place environment** using `robopal.make()`.
- **Enabling human input with a Gamepad** via `HumanDemonstrationWrapper`.
- **Collecting human demonstrations** while tracking task success.
- **Resetting the environment** based on completion conditions.

This approach is useful for **imitation learning**, where collected human demonstrations can train models for robotic manipulation tasks.

A.2 Dex-Retargeting

I also explained my most important codes of reproduction paper here:

A.2.1 Hand Pose Detection and Retargeting

The `start_retargeting` function initializes the SAPIEN simulation environment and loads the robot model. It performs real-time hand pose detection and translates human hand movements to the robotic hand.

Listing A.5: Hand Pose Detection and Retargeting

```
def start_retargeting(queue: multiprocessing.Queue, robot_dir: str,
    config_path: str):
    RetargetingConfig.set_default_urdf_dir(str(robot_dir))
    retargeting = RetargetingConfig.load_from_file(config_path).build()
    detector = SingleHandDetector(hand_type="Right", selfie=False)

    # Set up SAPIEN environment
    scene = sapien.Scene()
    scene.add_ground()

    # Load robot
    loader = scene.create_urdf_loader()
    robot = loader.load(Path(config.urdf_path))

    while True:
        try:
            bgr = queue.get(timeout=5)
            rgb = cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB)
        except Empty:
            return

        _, joint_pos, keypoint_2d, _ = detector.detect(rgb)
        bgr = detector.draw_skeleton_on_image(bgr, keypoint_2d, style="
            default")
        cv2.imshow("realtime_retargeting_demo", bgr)

        if joint_pos is not None:
            qpos = retargeting.retarget(joint_pos)
            robot.set_qpos(qpos)
```

A.2.2 Frame Producer (Video Capture)

The `produce_frame` function captures frames from a video source (e.g., webcam) and sends them to a multiprocessing queue.

Listing A.6: Frame Producer

```
def produce_frame(queue: multiprocessing.Queue, camera_path: Optional[str
    ] = None):
    cap = cv2.VideoCapture(0 if camera_path is None else camera_path)

    while cap.isOpened():
        success, image = cap.read()
        if success:
            queue.put(image)
```

A.2.3 Main Function and Process Management

The `main` function manages multiprocessing, launching both the frame producer and the retargeting system.

Listing A.7: Main Function

```
def main(robot_name: RobotName, retargeting_type: RetargetingType,
    hand_type: HandType, camera_path: Optional[str] = None):
    queue = multiprocessing.Queue(maxsize=1000)
    producer_process = multiprocessing.Process(target=produce_frame, args
        =(queue, camera_path))
    consumer_process = multiprocessing.Process(target=start_retargeting,
        args=(queue, str(robot_dir), str(config_path)))

    producer_process.start()
    consumer_process.start()
    producer_process.join()
    consumer_process.join()
```

A.3 Hand Bounding Box Detection

Due to limited space, I have only included images from the YOLO training process here.In particular, I showed plots of loss curves, Precision and Recall trends and maP trends during training.

A.3.1 Loss Curves

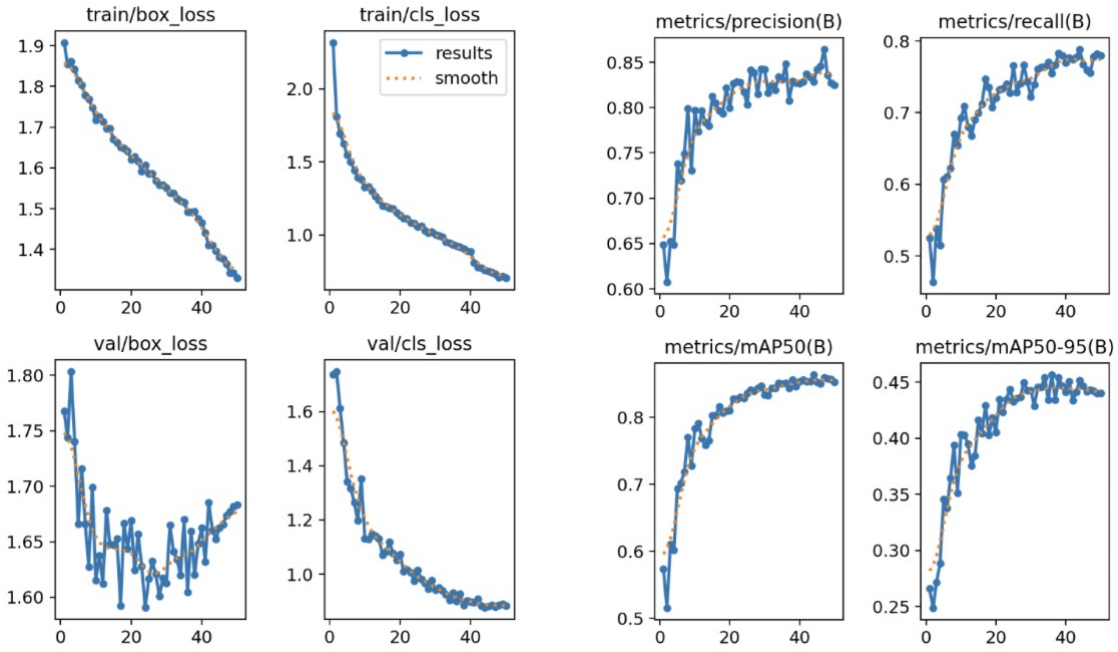


Figure A.1: Loss Curves

A.3.2 P&R Trends

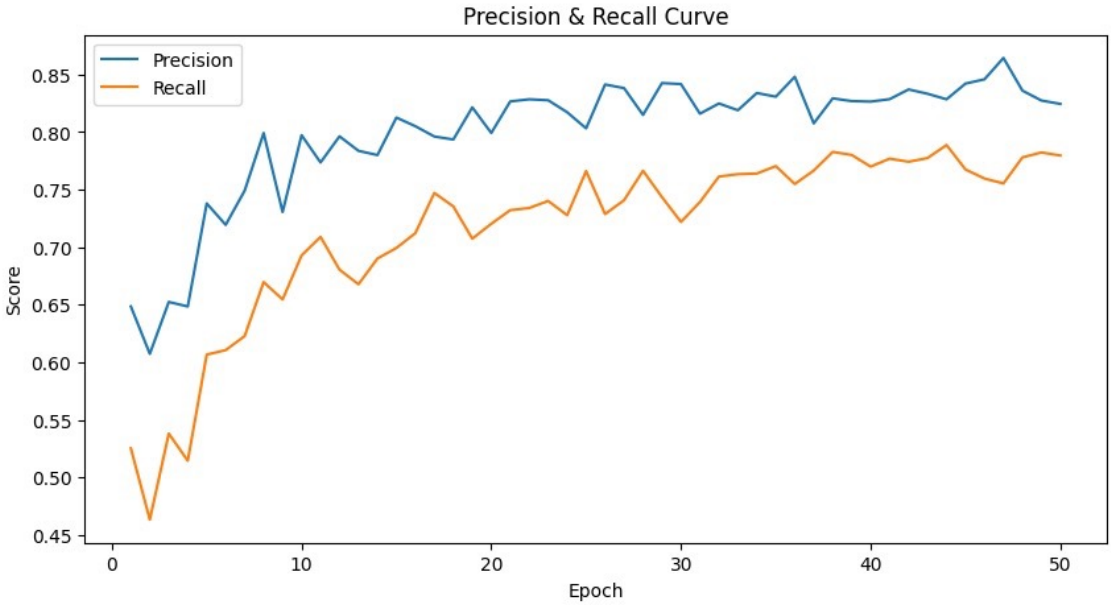


Figure A.2: P&R Trends

A.3.3 maP Trends

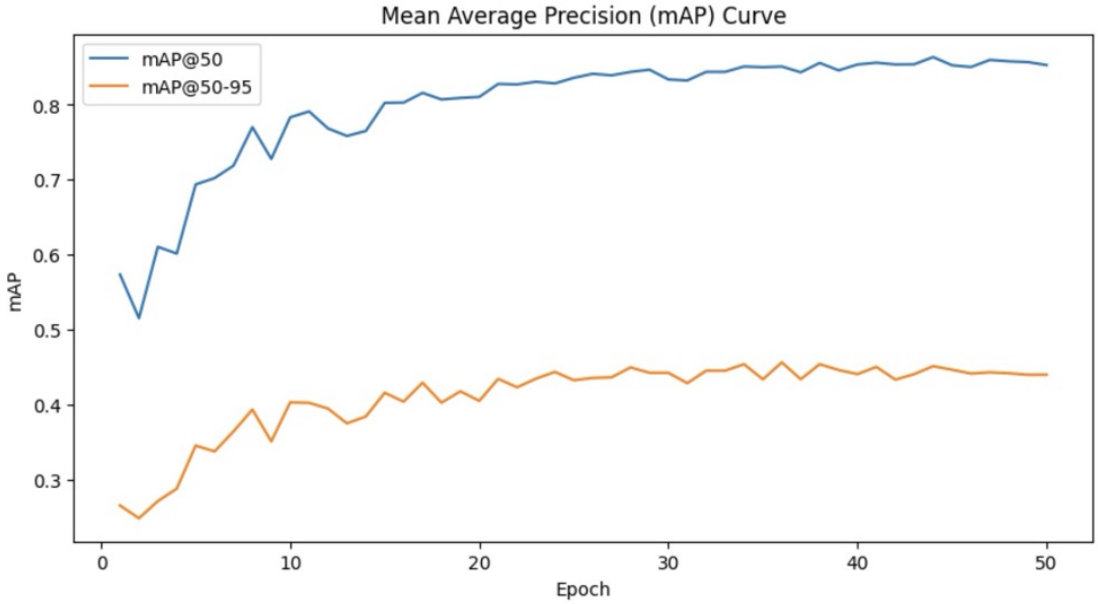


Figure A.3: maP Trends

Bibliography

[1] Insaf Ajili, Malik Mallem, and Jean-Yves Didier. Gesture recognition for humanoid robot teleoperation. In 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pages 1115–1120. IEEE, 2017. (Cited on page 15.)

[2] D. Antotsiou et al. Task-oriented hand motion retargeting for dexterous manipulation imitation. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops, 2018. Accessed: 2024-06-03. (Cited on page 4.)

[3] Sridhar Pandian Arunachalam, Irmak Güzey, Soumith Chintala, and Lerrel Pinto. Holo-dex: Teaching dexterity with immersive mixed reality. arXiv preprint arXiv:2210.06463, 2022. (Cited on page 7.)

[4] Sridhar Pandian Arunachalam, Sneha Silwal, Ben Evans, and Lerrel Pinto. Dexterous imitation made easy: A learning-based framework for efficient dexterous manipulation. arXiv preprint arXiv:2203.13251, 2022. (Cited on page 8.)

[5] Ross Girshick. Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (ICCV), pages 1440–1448, 2015. (Cited on page 12.)

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pages 580–587, 2014. (Cited on page 11.)

[7] A. Handa et al. Dexpilot: Vision based teleoperation of dexterous robotic hand-arm system. Oct 2019. Accessed: 2024-06-03. (Cited on page 6.)

[8] Shuang Li, Norman Hendrich, Hongzhuo Liang, Philipp Ruppel, Changshui Zhang, and Jianwei Zhang. A dexterous hand-arm teleoperation system based on hand pose estimation and active vision. IEEE Transactions on Cybernetics, 2022. (Cited on page 10.)

[9] Shuang Li, Xiaojian Ma, Hongzhuo Liang, Michael Görner, Philipp Ruppel, Bin Fang, Fuchun Sun, and Jianwei Zhang. Vision-based teleoperation of shadow dexterous hand using end-to-end deep neural network. ICRA, 2019. (Cited on page 9.)

[10] A. Mittal, A. Zisserman, and P. H. S. Torr. Hand detection using multiple proposals. In British Machine Vision Conference, 2011. (Cited on page 1.)

[11] MD Moniruzzaman, Alexander Rassau, Douglas Chai, and Syed Mohammed Shamsul Islam. Teleoperation methods and enhancement techniques for mobile robots: A comprehensive survey. Robotics and Autonomous Systems, 154:104103, 2022. Accessed: 2024-06-03. (Cited on page 1.)

[12] Yuzhe Qin, Hao Su, and Xiaolong Wang. From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation. RA-L, 7(4):10873–10881, 2022. (Cited on page 10.)

[13] Yuzhe Qin, Yueh-Hua Wu, Shaowei Liu, Hanwen Jiang, Ruihan Yang, Yang Fu, and Xiaolong Wang. Dexmv: Imitation learning for dexterous manipulation from human videos, 2021. (Cited on page 11.)

[14] Yuzhe Qin, Wei Yang, Binghao Huang, Karl Van Wyk, Hao Su, Xiaolong Wang, Yu-Wei Chao, and Dieter Fox. Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system. In Robotics: Science and Systems, 2023. (Cited on pages 1, 6 and 15.)

[15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pages 779–788, 2016. (Cited on page 13.)

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (NeurIPS), pages 91–99, 2015. (Cited on page 12.)

[17] A. Sivakumar, K. Shaw, and D. Pathak. Robotic telekinesis: Learning a robotic hand imitator by watching humans on youtube. Jul 2022. Accessed: 2024-06-03. (Cited on page 8.)

[18] Ultralytics. YOLOv8: Real-Time Object Detection and Segmentation, 2023. Available at https://docs.ultralytics.com/. (Cited on page 13.)

[19] Tao Zhang. Toward automated vehicle teleoperation: Vision, opportunities, and challenges. IEEE, 2022. Accessed: 2024-06-03. (Cited on page 2.)