Abstract

The global waste crisis, characterized by increasing of annual municipal solid waste [1] inadequate recycling rates (13.5% recycling rate) [2] and estimated shrinking labor population [3][4], demands urgent technological intervention. This project develops an automated waste-sorting robotic system integrating computer vision, embedded computing, and robotic control. The system combines YOLOv11 object detection for waste classification, monocular geometric ranging for object localization, and inverse kinematics for robotic manipulation. Leveraging the RDK X5 embedded platform with Horizon Robotics' BPU accelerator, the model achieves real-time inference through ONNX conversion and post-training quantization. Key implementation steps include:

1. Dataset adaptation from VOC2007 to YOLO format with 22 optimized waste categories;
2. WiFi-based video streaming for real-time monitoring;
3. Camera calibration and tilt angle compensation for distance estimation;
4. UART communication protocol design for robotic arm control.

The integrated system demonstrates functional deployment across hardware components (OV5640 camera, Jibot1-stm32 6-DOF arm, RDK X5 mainboard), achieving object detection, distance calculation, and grasping coordination. Practical validation confirms the feasibility of embedded AI deployment in resource-constrained scenarios, providing a modular framework for industrial automation applications.

Keywords: waste sorting; YOLOv11; embedded vision; robotic kinematics; monocular distance measurement

摘要

全球垃圾危机以年固体垃圾排放持续增长[1]，回收率不足（回收率仅为 13.5%）[2]以及世界范围内劳动人口预计[3][4]减少为主要特征，因此迫切需要技术支撑。本项目开发了一套集计算机视觉、嵌入式计算和机器人控制于一体的自动化垃圾分类机器人系统。该系统结合了用于垃圾分类的 YOLOv11 目标检测、用于物体定位的单目几何测距和用于机器人操作的逆运动学。利用 RDK X5 嵌入式平台和 Horizon Robotics 的 BPU 架构，该模型通过 ONNX 转换和训练后量化实现了实时推理。主要实施步骤包括：

1. 从 VOC2007 到 YOLO 的数据集适配，包含 22 个经过筛选的垃圾类别
2. 基于 wifi 的实时监控视频流；
3. 距离估计中的摄像机标定和倾斜角补偿
4. 机械臂控制器的 UART 通信协议设计。

集成系统实现了跨硬件部件（OV5640 摄像头、Jibot1-stm32 6 自由度机械臂、RDK X5 主板）的功能部署，实现了目标检测、距离计算和抓取协调。实践验证证实了嵌入式人工智能在资源受限场景下部署的可行性，为工业自动化应用提供了模块化框架。

關鍵詞：垃圾分类 ；YOLOv11；嵌入式视觉 ；机器人逆运动学；单目测距

Table of Contents

Abstract I

摘要 II

Table of Contents III

List of Figures VI

List of Tables X

Chapter 1 Introduction 1

1.1 Background and motivation 1

1.2 Scope of project 3

Chapter 2 Related Work 6

2.1 Vision-Based Object Recognition and Pose Estimation 6

2.2 Monocular Distance Estimation Techniques 6

2.3 Learning-Based Depth Estimation 7

2.4 Inverse Kinematics and Control Systems 7

2.5 Embedded Systems for Robotic Control 8

2.6 Grasp Pose Detection and Planning. 9

2.7 Summary 10

Chapter 3 Hardware 11

3.1 Main development board 11

3.2 Robot arm 11

3.3 Camera 12

Chapter 4 Project Design 13

Chapter 5 Methodology 14

5.1 YOLO (You Only Look Once) 14

5.1.1 Feature 14

5.1.2 Evolution 14

5.1.3 Innovation *15*
5.1.4 Details of YOLOv11 ..... *15*
5.2 Model Quantization 20
5.2.1 Optimization of the model processing flow *20*
5.2.2 Horizon Sunrise 5 algorithm toolchain *21*
5.3 Monocular distance measurement 24
5.3.1 Camera calibration *24*
5.3.2 Tilt Angle calibration and geometric ranging model *25*
5.4 Inverse kinematics 29
5.4.1 Mathematical description *29*
5.4.2 Denavit-Hartenberg parameterization method *29*
5.4.3 Inverse kinematics analysis *31*
5.5 Built-in instruction program of the robot arm 34
Chapter 6 Practices and Results 36
6.1 Model training and optimization 36
6.1.1 Dataset *36*
6.1.2 Model training *42*
6.2 Model quantization 50
6.2.1 ONNX format conversion *50*
6.2.2 Environment installation *50*
6.2.3 Model quantization using toolchain *52*
6.3 WiFi-based video streaming 59
6.3.1 Inference and video as input *59*
6.3.2 Deploy on local network port *62*
6.3.3 File migration and testing *63*
6.4 Monocular distance measurement 66
6.4.1 Camera calibration *66*
6.4.2 Tilt angle measurement *68*
6.4.3 Object height measurement *71*
6.4.4 Ranging and testing *72*
6.5 Inverse kinematics 76

6.6 UART communication 79
6.6.1 Robot arm controller side 79
6.6.2 Main development board side 83
6.7 System integration and final test 86
Chapter 7 Conclusion 89
7.1 Summary 89
7.1.1 Core achievements 89
7.1.2 Critical limitations 89
7.2 Future work 90
References 91
Appendix A. Cover Page 97
Appendix B. Project Code and User Manual 98
Resume 99
Acknowledgements 103

List of Figures

Figure 4- 1 Project Architecture 13
Figure 5-1- 1 Architecture of YOLO11 16
Figure 5-2- 1 Basic Workflow of PTQ 22
Figure 5-3- 1 Coordinate of camera 26
Figure 5-3- 2 Pinhole camera model 27
Figure 5-4- 1 Coordinate of robot arm 30
Figure 5-4- 2 Simplified model of robot arm coordinate 32
Figure 6-1- 1 File structure of VOC2007 dataset 37
Figure 6-1- 2 convert.py 37
Figure 6-1- 3 label file in VOC format 38
Figure 6-1- 4 label file in VOC format 38
Figure 6-1- 5 File structure of YOLO dataset 39
Figure 6-1- 6 divide.py 39
Figure 6-1- 7 trash.yaml 40
Figure 6-1- 8 filter.py (1) 40
Figure 6-1- 9 filter.py (2) 41
Figure 6-1- 10 filter.py (3) 41
Figure 6-1- 11 train.py for Kaggle 42
Figure 6-1- 12 train.py for HPC Fun 42
Figure 6-1- 13 yolo11.yaml 43
Figure 6-1- 14 Diverse mode size of YOLO11 43
Figure 6-1- 15 Uploaded files on Kaggle 44
Figure 6-1- 16 Notebook page on Kaggle 44
Figure 6-1- 17 log of notebook yolo11 (1) 45
Figure 6-1- 18 log of notebook yolo11 (2) 45
Figure 6-1- 19 Upload files by *WinSCP* 46
Figure 6-1- 20 Extract files on cloud VSCode 46
Figure 6-1- 21 Model training by SSH 47
Figure 6-1- 22 Labeling image using *labelImg* 48
Figure 6-1- 23 Before adjusting dataset 48
Figure 6-1- 24 After adjusting dataset 49

Figure 6-2- 1 export_onnx.py 50
Figure 6-2- 2 Ubuntu images download 50
Figure 6-2- 3 docker’ s images page 51
Figure 6-2- 4 Toolchain download 51
Figure 6-2- 5 Mount files on docker 52
Figure 6-2- 6 Architecture diagram of model deployment 52
Figure 6-2- 7 Flow diagram of model transformation 53
Figure 6-2- 8 log of model check 54
Figure 6-2- 9 Operators and allocated unit 54
Figure 6-2- 10 cal.py (transform images to required format) 55
Figure 6-2- 11 trans.yaml 55
Figure 6-2- 12 log of model transformation (1) 56
Figure 6-2- 13 log of model transformation (2) 56
Figure 6-2- 14 log of model performance validation 57
Figure 6-2- 15 I/O information of model 57
Figure 6-2- 16 I/O information of model 57
Figure 6-3- 1 wifi.py - *capture_and_detect* (1) 59
Figure 6-3- 2 wifi.py - *capture_and_detect* (2) 59
Figure 6-3- 3 wifi.py - *capture_and_detect* (3) 60
Figure 6-3- 4 wifi.py - *forward* and *c2numpy* 60
Figure 6-3- 5 wifi.py - *postprocess* (1) 61
Figure 6-3- 6 wifi.py - *postprocess* (2) 61
Figure 6-3- 7 wifi.py - *postprocess* (3) 61
Figure 6-3- 8 wifi.py - *postprocess* (4) 61
Figure 6-3- 9 wifi.py - *postprocess* (5) 61
Figure 6-3- 10 wifi.py - *postprocess* (6) 61
Figure 6-3- 11 wifi.py - Webpage layout 62
Figure 6-3- 12 wifi.py - Frame update 63
Figure 6-3- 13 Webpage deployment 63
Figure 6-3- 14 RDK Studio 64
Figure 6-3- 15 Manage files on board by file manager on PC 64
Figure 6-3- 16 Testing log 64
Figure 6-3- 17 Testing result 65

Figure 6-4- 1 Camera Calibrator in Matlab 66
Figure 6-4- 2 Camera parameters derived 67
Figure 6-4- 3 calibrate.py 67
Figure 6-4- 4 Undistorted result 68
Figure 6-4- 5 Different distances 68
Figure 6-4- 6 angle.py (1) 69
Figure 6-4- 7 angle.py (2) 69
Figure 6-4- 8 angle.py (3) 70
Figure 6-4- 9 Result of tilt angle calculation 70
Figure 6-4- 10 Height of testing object 71
Figure 6-4- 11 wifi.py – *load_object_heights* 71
Figure 6-4- 12 wifi.py – *draw_detection* 72
Figure 6-4- 13 wifi.py – *MonocularRanging* 72
Figure 6-4- 14 Theoretical result: 11.3cm 73
Figure 6-4- 15 Actual result: 16.5cm 73
Figure 6-4- 16 Theoretical result: 13.7cm 74
Figure 6-4- 17 Actual result: 15.5cm 74
Figure 6-5- 1 z_kinematics.c (1) 76
Figure 6-5- 2 z_kinematics.c (2) 76
Figure 6-5- 3 z_kinematics.c (3) 77
Figure 6-5- 4 z_kinematics.c (4) 77
Figure 6-5- 5 z_kinematics.c (5) 77
Figure 6-5- 6 z_kinematics.c (6) 78
Figure 6-5- 7 z_main.c – *kinematics_move* 78
Figure 6-6- 1 z_uart.c 79
Figure 6-6- 2 z_sensor.c – *parse_data* (1) 80
Figure 6-6- 3 z_sensor.c – *parse_data* (2) 81
Figure 6-6- 4 z_sensor.c – *ceju_jiaqu* 81
Figure 6-6- 5 z_sensor.c – *carry_wood* 82
Figure 6-6- 6 Add action groups for sorting 82
Figure 6-6- 7 control.py 83
Figure 6-6- 8 wifi.py – *ObjectTracker* (1) 84

Figure 6-6- 9 wifi.py – *ObjectTracker* (2) 84
Figure 6-6- 10 wifi.py – *ObjectTracker* (3) 85
Figure 6-6- 11 wifi.py – *send_to_serial* 85
Figure 6-7- 1 System integration 86
Figure 6-7- 2 Command line log 86
Figure 6-7- 3 Real-time streaming on web server 87
Figure Appendix- 1 Complete project code on GitHub 98
Figure Appendix- 2 Part of the user manual 98

List of Tables

Table 5-4- 1 DH parameters of robot arm 30
Table 5-5- 1 Commands' format of robot arm controller 35
Table 6-7- 1 Results of final testing..... 88

Chapter 1 Introduction

1.1 Background and motivation

The accelerating crisis of global waste management presents a critical juncture for technological innovation. As urban populations expand and consumption patterns intensify, the world generated over 2.1 billion tons of municipal solid waste in 2020—a figure projected to surge by 80% to 3.8 billion tons by 2050 [1]. This deluge of waste has outpaced traditional disposal methods, with landfills reaching capacity and incineration releasing carcinogenic dioxins into ecosystems. Compounding this environmental emergency, only 13.5% of global waste is recycled, while most of the others were accumulates in landfills and dumps or incinerated into open air, which is increasingly becoming a threat of our environment and health [2].

Amidst this ecological turmoil, shifting global demographics are reshaping labor markets in ways that demand automation. Concurrently, global labor shortages amplify the need for automated waste management solutions. The European Commission (2024) projects a 19% decline in the EU's working-age population (20–64 years) by 2050, from 264 million to 207 million, straining pension systems and public services [3]. China faces similar challenges, with the prediction results showing that from 2020 to 2030, the size of the working-age population in China will decline from 989 million to 963 million, and the labor participation rate will drop from 68.44% to 65.17%. Based on the development trends of the two indicators themselves, the scale of China's labor supply will continue to decline and reach 627 million by 2030 [4]. These challenges converge in a paradox: societies need more efficient waste management just as the labor force capable of performing this dangerous work diminishes.

Embedded automation emerges as a pivotal solution, blending precision robotics with energy-efficient computing to address both environmental and economic imperatives. Against this backdrop, embedded devices and automation technologies offer a viable solution to enhance efficiency and reduce reliance on dwindling labor resources. For example, MIT's ROCycle robot uses capacitive sensors and machine learning to sort recyclables with 85% accuracy, significantly outperforming manual sorting in contaminated streams [5]. Companies like TOMRA have developed advanced optical sorting systems, such as the AUTOSORT™, which combines AI and sensor

technologies to achieve high throughput and purity rates in waste recycling plants. These systems not only improve sorting accuracy but also reduce operational costs: TOMRA's solutions enable real-time monitoring of calorific values and contaminant levels, optimizing resource recovery and minimizing landfill waste [6].

In summary, the convergence of escalating waste volumes, plastic pollution, and labor market pressures demands scalable, technology-driven solutions. Embedded devices and automation technologies not only address the immediate challenges of waste mismanagement but also align with broader sustainability goals, such as the SDG 12 (Responsible Consumption and Production) [7]. By leveraging these innovations, cities and industries can transition to more efficient, cost-effective, and resilient waste management systems, mitigating environmental degradation while adapting to demographic realities.

It is within this context of ecological urgency and technological opportunity that the project develops a vision-guided robotic system for automated waste classification. By integrating YOLOv11's lightweight neural architecture—quantized to 8-bit precision for embedded deployment—with geometric depth estimation and adaptive motion planning, the system achieves industrial-grade sorting accuracy at lower cost than conventional alternatives. This innovation not only addresses immediate waste management gaps but also offering a scalable blueprint for low-carbon circular economies. As municipalities from Rotterdam to Shenzhen mandate stricter recycling compliance, such embedded solutions provide the missing link between policy ambition and operational reality.

1.2 Scope of project

The project scope encompasses seven key components, which covers the main work that was done in the project: 1) Training and optimization of a YOLOv11 object detection model for waste recognition, 2) Model format conversion and quantization, 3) Development of a WiFi-based video streaming subsystem, 4) Implementation of monocular distance measurement, 5) Inverse kinematics programming for robotic manipulation, 6) Design of a robust UART communication protocol, and 7) System integration and performance validation.

1. **Object detection model training**

   Considering the limited computing resources of embedded systems and the need to detect the types of garbage in real time, YOLO11 [8] is selected as the framework of object recognition model for garbage classification. The data set has a great impact on the final usability of the model. Because the YOLO framework requires labeled images for model training and has certain requirements for the format, the domestic waste data set of Huawei 2020 Cloud competition [9] is selected after screening, which contains 44 types of common domestic waste and has more accurate target object labels. Since the original dataset is in VOC2007 format, first it was required to be converted to YOLO format for later training. Then divide the training set and validation set according to a certain proportion. In the actual training, the accuracy of object recognition of some specific categories is significantly lower than the average, and considering that the weight that the robot arm can grip is relatively limited, the data set is subsequently deleted and optimized, and the model finally reached the overall accuracy of slightly above 80%.

2. **Model conversion and quantization**

   Since the selected development board RDK X5 only supports the deployment of Caffe and ONNX models, it is first necessary to export the trained waste sorting model in.pt format to ONNX format. Because RDK X5 uses a special processing unit called a BPU to accelerate model inference, the validation and final transformation of the model is performed using a toolchain developed by the company to complete model deployment on the BPU. The final result achieved object recognition throughput of 68.75fps and a latency of 14.55ms.

3. **WiFi-based video streaming**

In order to view the effect of object recognition in real time on the development computer in the same network environment as development board, it is necessary to develop a WiFi video transmission function in the LAN, which displays the camera streaming, the recognition frame of the model and the object category in the picture, and then deploy it to the local network port of the development board. In order to evaluate the performance of the model in real operation, the calculation of real-time frame rate and post-processing time were also added. After adding the object distance measurement function, the distance to objects was added to evaluate the error with the actual value.

4. **Monocular distance measurement**

   Due to the limited project funds, an ordinary monocular camera is selected, which only has the function of video transmission and does not have the function of estimating the depth of the object. Therefore, a slightly lower accuracy monocular distance measurement method was adopted for object ranging. In order to achieve more accurate ranging, the camera needs to be calibrated to minimize the error caused by image distortion, and then the tilt Angle of the camera relative to the robot arm is calibrated. Finally, the relative vertical coordinates of the object in the video frame are obtained through the object marker box returned by the YOLO model, and combining with the previous camera tilt Angle, the approximate distance between the object and the base of the robot arm can be calculated.

5. **Inverse kinematics**

   When the distance of the object is obtained, the desired position of each joint of the robot arm needs to be derived in reverse to achieve accurate clamping. To achieve this, the method of inverse kinematics needs to be used to establish a coordinate system for the whole robot arm and each joint, and then adjust the Angle of each joint according to specific formulas with the length of each joint. As a result, the mechanical claw at the end can grasp the object at the corresponding coordinate.

6. **UART communication**

   A certain communication method needs to be established between the main control development board and the controller of the robot arm in order to send the waste sorting category and distance of the object to the controller of the robot arm, and complete the subsequent clamping and classification process. Because the controller of the robot arm already has pre-built instruction receiving program of the UART

protocol, and the corresponding serial port driver has been installed on the main control board, it only needs to design a certain instruction format and modify the original program appropriately to ensure that the robot arm controller can correctly process the data in the instruction and control the robot arm to complete the expected action.

7. **System integration**

After step-by-step testing, it has been ensured that the model has a certain accuracy of recognizing the object, the error of monocular distance measurement is within the operational range, the robot arm can move to the specified distance according to the received distance to complete the clamping, and the object distance and category instructions sent by the main control board can be correctly processed by the robot arm controller, and then the final overall test needs to be completed. Different objects will be placed to cover the categories of domestic waste in the data set. After the classification of all objects, the success rate of picking and the accuracy rate of classification are calculated to evaluate the reliability of the system, and the points that need to be improved in the future project will be clarified during the process.

Chapter 2 Related Work

Recent advances in robotic manipulation have focused on integrating vision, control systems, and learning-based approaches to enable robots to interact with objects effectively. This section reviews key developments in embedded systems for robotic control, object recognition, distance estimation, inverse kinematics, and grasp planning.

2.1 Vision-Based Object Recognition and Pose Estimation

Vision systems provide robots with critical information about objects in their environment. Xiang et al. (2018) proposed PoseCNN [10], a convolutional neural network for 6D object pose estimation that decouples translation and rotation estimation, enabling improved robotic manipulation capabilities. Their approach combined a segmentation network with pose regression to achieve robust performance in cluttered scenes. Building upon this work, Wang et al. (2019) introduced DenseFusion [11], which fused RGB and depth features from different modalities at the per-pixel level, achieving more accurate pose estimation for robotic manipulation tasks. For robotic grasping applications, Mahler et al. (2019) presented Dex-Net 4.0 [12], a framework that combines synthetic data generation with deep learning to enable robust grasping of novel objects. Their system demonstrated how simulation-to-reality transfer can yield effective grasp policies for parallel-jaw and suction grippers in collaborative human-robot applications. In a similar vein, Fang et al. (2020) introduced GraspNet-1Billion [13], a large-scale benchmark for general object grasping that contained over one billion grasp poses. Their accompanying network, GraspNet, predicted grasps directly from RGB-D images and advanced the state-of-the-art in grasp detection for robotic systems.

2.2 Monocular Distance Estimation Techniques

Accurate distance estimation is crucial for robotic manipulation, particularly when depth sensors are unavailable or impractical. Geometric and learning-based approaches to monocular distance estimation have seen significant advancements in recent years. Gordon et al. (2019) presented "Depth from Videos in the Wild: Unsupervised

Monocular Depth Learning from Unknown Cameras,"[14] which uses structure-from-motion cues from video sequences to learn depth prediction without requiring camera parameters. Their approach enables robots to estimate distances in unknown environments with uncalibrated cameras.

For robotic navigation applications, Wang et al. (2018) introduced "Monocular Visual Odometry Scale Estimation Using Geometric Triangulation,"[15] a method that recovers absolute scale from monocular visual odometry by exploiting geometric constraints. This technique enables mobile robots to accurately estimate distances using only a single camera, which is particularly valuable for lightweight platforms. Similarly, Iyer et al. (2018) proposed "Geometric Consistency for Self-Supervised End-to-End Visual Odometry,"[16] a framework that improves the accuracy of visual odometry by enforcing geometric consistency constraints, providing more reliable distance measurements for robotic manipulation tasks.

2.3 Learning-Based Depth Estimation

Moving beyond geometric approaches, learning-based methods have significantly improved monocular depth estimation. Godard et al. (2019) introduced MonoDepth2 [17], an unsupervised learning framework that utilized a minimum reprojection loss and auto-masking to handle occlusions. This approach enabled training without ground truth depth data, making it particularly valuable for robotic applications in novel environments.

For robotics applications requiring real-time performance, Guizilini et al. (2020) proposed PackNet-SfM [18], a self-supervised monocular depth estimation network with 3D convolutions. Their approach demonstrated superior performance on the KITTI benchmark while maintaining computational efficiency suitable for embedded platforms. Ranftl et al. (2021) further advanced the field with their MiDaS [19] architecture for robust monocular depth estimation across diverse datasets, addressing domain generalization problems critical for robotic deployment in varied environments.

2.4 Inverse Kinematics and Control Systems

Inverse kinematics algorithms translate desired end-effector positions into joint configurations, a critical component of robotic manipulation. Traditional analytical

approaches based on the Denavit-Hartenberg (DH) convention remain fundamental to many robotic systems. Kucuk and Bingul (2014) presented "Inverse Kinematics Solutions for Industrial Robot Manipulators with Offset Wrists," [21] which provides closed-form solutions for six-degree-of-freedom manipulators using DH parameters. Their work demonstrates the efficiency of analytical approaches for specific robot geometries.

Building on classical mechanics, Chen et al. (2020) introduced "Improved Inverse Kinematics Algorithm Using Screw Theory for 6-DOF Robot Manipulators," [22] which combines screw theory with product-of-exponentials to achieve computationally efficient solutions for complex manipulators. Their method demonstrates how modern mathematical formulations can enhance traditional inverse kinematics approaches.

For iterative solutions to the inverse kinematics problem, Aristidou and Lasenby (2011) developed "FABRIK: A Fast, Iterative Solver for the Inverse Kinematics Problem," [23] which offers a geometrically-based approach that outperforms traditional methods in terms of computational efficiency and visual accuracy. Though not recent, this widely-cited work continues to influence modern robotic implementations.

Combining traditional and learning-based approaches, Csiszar et al. (2017) presented "On Solving the Inverse Kinematics Problem Using Neural Networks," [24] which utilizes deep learning to approximate inverse kinematics solutions without requiring analytical models. Their approach demonstrates how neural networks can effectively address the challenges of non-linearity and redundancy in robotic manipulators.

Lembono et al. (2020) introduced "Memory of Motion for Warm-Starting Trajectory Optimization," [25] a technique that leverages previous motion experiences to initialize optimization processes, significantly improving convergence speed for real-time inverse kinematics solutions. This approach is particularly valuable for dynamic environments where rapid adaptation is essential.

2.5 Embedded Systems for Robotic Control

The implementation of control algorithms on resource-constrained hardware presents unique challenges related to computational constraints and real-time requirements. Ohara et al. (2021) implemented a quantized neural network for grasp classification entirely on an FPGA platform [26], achieving more than 40 Hz inference rates with 90% classification accuracy for 6-DOF manipulation tasks using Xilinx Zynq hardware.

Their approach demonstrates how specialized hardware acceleration can enable complex AI-based control systems on embedded platforms without relying on high-power computing resources.

For IoT-integrated robotics, Krejčí et al. (2023) developed a system that combines low-power microcontrollers (STM32) with ARM-based IoT modules to create what they term a "Robotic Internet of Things."[27] Their framework enables control of a 5-DOF robotic arm through MQTT and ROS 2 protocols, demonstrating how modern networking approaches can extend the capabilities of embedded control systems.

Addressing real-time control needs for smaller manipulators, Ahmed et al. (2021) presented an ESP32-based embedded system for tele-operation of a 4-DOF robotic arm over Wi-Fi [28]. Their implementation handles PWM motor control and real-time sensor feedback, providing a responsive platform for remote manipulation while maintaining low power consumption and hardware costs.

These approaches demonstrate how contemporary embedded systems can implement increasingly sophisticated control algorithms directly on resource-constrained hardware, enabling fast, deterministic responses in manipulation tasks without requiring external computation resources.

2.6 Grasp Pose Detection and Planning.

Effective object manipulation requires not only accurate vision and control but also appropriate grasp pose selection. Mahler et al. (2017) pioneered data-driven approaches with "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics," [29] which uses a deep learning approach trained on synthetic data to predict grasp success probability from point clouds. Their system demonstrated reliable grasping of novel objects using only single-view depth images.

For real-time applications, Redmon and Angelova (2015) presented "Real-Time Grasp Detection Using Convolutional Neural Networks," [30] a pioneering approach that directly regresses grasp rectangles from RGB images. Their single-stage architecture achieved both high accuracy and real-time performance (13 frames per second), marking a significant improvement over earlier multi-stage methods.

Addressing the need for 6-DOF grasp pose generation, Yan et al. (2019) introduced "Learning 6-DOF Grasping Interaction with Deep Geometry-aware 3D Representations," [31] which uses variational autoencoders to generate diverse grasp

poses for arbitrary objects. Their approach learns the distribution of successful grasps directly from RGB-D data without requiring explicit object models.

For point cloud-based manipulation, Liang et al. (2019) developed "PointNetGPD: Detecting Grasp Configurations from Point Sets," [32] which uses a modified PointNet architecture to evaluate grasp candidates directly from unstructured point clouds. Their framework eliminates the need for intermediate representations, enabling more efficient grasp detection for novel objects.

More recently, Sundermeyer et al. (2021) presented "Contact-GraspNet: Efficient 6-DoF Grasp Generation in Cluttered Scenes," [33] a single-stage approach that directly generates 6-DOF grasps from partial point clouds. Their model explicitly reasons about contact points and surface normals, enabling robust grasping in cluttered environments without requiring object segmentation.

2.7 Summary

The related work demonstrates significant advances in vision-based robotic manipulation, spanning perception, control, and planning. While deep learning approaches have dramatically improved capabilities across these domains, geometric methods for problems like monocular ranging remain valuable, particularly for embedded systems with computational constraints. Integration of these techniques into cohesive systems remains challenging, requiring careful consideration of hardware limitations, real-time requirements, and application-specific constraints. The project builds upon these foundations to develop an efficient robotic arm system that balances computational requirements with performance.

Chapter 3 Hardware

This section will introduce the hardware parameters used in the project and the reasons for selection, including the main development board, the robot arm and the camera.

3.1 Main development board

RDK X5 by D-Robotics is equipped with Sunrise 5 intelligent computing chip, which can provide BPU computing power up to 10 Tops. It is an all-around development kit for intelligent computing and robotics applications with plenty of interfaces [34]. The BPU (Brain Processing Unit) architecture proposed by Horizon is a processor architecture specially designed for artificial intelligence applications, which aims to solve the efficiency problem of traditional processors in processing large-scale parallel computing tasks, especially in the fields of image recognition, speech processing, natural language understanding and control. Through optimized hardware accelerators, BPU realizes the efficient computation of complex neural network models such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Transformer, achieving higher computational efficiency and performance with limited power consumption and cost [35]. In addition to having sufficient computing power, RDK X5 also has a number of successful deployments of common vision models and tutorials in the official documentation and forums which can help avoid unnecessary trial and error in deploying YOLO models.

In addition to the above advantages, RDK X5 also supports flash development. Only by connecting to the development computer through the Type-C line, developer can use the RDK Studio application on the computer to call a series of built-in functions such as system burning, file management, command line, etc. [34], which is convenient to transplant the developed program to the main development board for debugging in time.

3.2 Robot arm

The robot arm uses the Jibot1-stm32 6-DOF pan-tilt-head robot arm by Zhongling Technology. The body is made of full aluminum alloy material, and the maximum grasping weight is 500g. The stm32 control board equipped with STM32F103C8T6 main control chip is used as the main control of the robot arm. It supports PWM servo control, bus device control (bus servo, bus motor) and other external control methods (USB, serial port, handle, infrared, etc.). After installing the serial driver, operations like

read and write motor parameters, edit and download action groups, and send instructions can be done through the serial port connected to the host computer software developed by the manufacturer, which makes the developing and debugging procedure more convenient. In addition, the manufacturer also provides a wealth of learning materials and project code reference, so it saves the time required for the development of the underlying hardware such as motor PWM control and UART protocol, and facilitates the later development of the project.

3.3 Camera

The OV5640 series camera was selected as the camera, and a custom USB version was used considering the need for sufficient length of wire to fix the camera to the robotic arm. OV5640 sensor supports the output of up to 5-megapixel images (2592x1944 resolution, 15 fps). It supports the use of VGA timing to output image data, and the output image data format supports YUV(422/420) and MJPEG format. When the MJPEG format image is output directly, the amount of data can be greatly reduced and the network transmission is convenient. It can also compensate the acquired image, support gamma curve, white balance, saturation, chroma and other basic processing, and support auto focus function. According to different resolution configurations, the frame rate of the sensor output image data is adjustable from 15 to 30 frames, and the required power is between 150mW and 200mW when working [36]. Benefiting from the above advantages, the OV5640 series camera can meet the needs of stable transmission of real-time video signals in the project.

Chapter 4 Project Design

According to the introduction of the hardware section, the system composed by the project can be divided into four parts: the main control board, the camera, the robot arm and its controller. Their respective roles are shown in the figure:

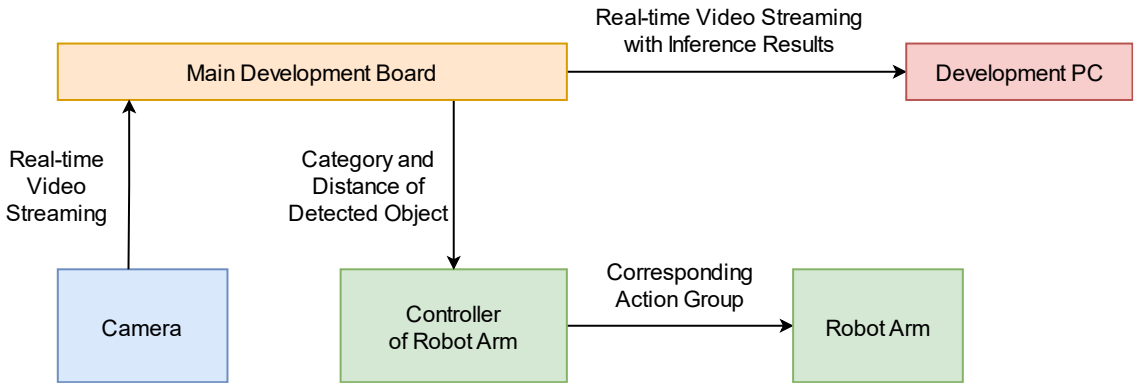

Figure 4- 1 Project Architecture

In order to realize the automatic picking and sorting of objects, the system will basically go through the following steps: 1) the camera transmits the video image to the main control board; 2) Determine the waste sorting category of the object through the object recognition model deployed on the main control board; 3) determine the distance between the robot arm and the object; 4) Transmit the object category and distance to the robot arm controller; 5) The robot arm controller operates the robot arm to complete the corresponding action group. In the original idea, the object ranging is completed by the ultrasonic sensor module, but in the actual test, it is necessary to repeatedly move the object to make the sensor detect the object and return the distance, and the object of some materials cannot be stably identified by the sensor due to the reflectance and other problems, so the monocular distance measurement method is used to estimate the distance between the robot arm and the object. After this modification, the main control board is responsible for transmitting the category and distance of the object to the robot arm controller. The specific method will be introduced in detail later.

Chapter 5 Methodology

This part will introduce the concepts involved in the project and the methods used, including the introduction of the YOLO architecture, model transformation, monocular distance measurement, inverse kinematics and built-in instruction program of the robot arm as pre-knowledge for the subsequent code interpretation part.

5.1 YOLO (You Only Look Once)

As one of the core tasks of computer vision, object detection technology evolution has always focused on the balance between accuracy and efficiency. Among the many detection frameworks, the YOLO (You Only Look Once) family stands out because of its unique One-Stage design, creating the first "end-to-end real-time detection" [36].

5.1.1 Feature

Compared with traditional two-stage detection models (such as Faster R-CNN), which need to generate candidate regions before classification and regression, YOLO directly divides the input image into grids, and each grid cell independently predicts the location and category of the object. This "single-shot forward inference" design significantly reduces computational latency, making it an absolute advantage in real-time critical scenarios (e.g., autonomous driving, drone surveillance).

5.1.2 Evolution

The development of YOLO can be seen as a continuous breakthrough in efficiency bottlenecks and accuracy defects. YOLOv1 [36] in 2016 verified the feasibility of single-stage detection for the first time, but its grid division was rough and its ability to detect small objects was insufficient. Subsequently, YOLOv2 introduced Anchor Box and Batch Normalization to improve the positioning accuracy [37]. YOLOv3 further introduces Multi-Scale Prediction, which uses feature maps of different resolutions to detect objects of different sizes, and significantly improves the detection performance of small objects [38]. However, as the model depth increases, the computational efficiency gradually becomes the bottleneck. In 2020, YOLOv4 and YOLOv5 have become the mainstream choice for industrial deployment by fusing CSP (Cross Stage Partial) structure and attention mechanism to compress the amount of parameters while

maintaining accuracy [39][40]. The YOLOv7 to YOLOv10 after 2022 focus on dynamic network design, loss function optimization and hardware adaptation, and gradually penetrate into edge computing scenarios[41][43][44].

The evolution from YOLOv1 to YOLO11 is essentially a two-way collaboration between algorithm innovation and hardware adaptation: the former improves the information extraction efficiency per unit of calculation through structural optimization, and the latter releases the computing power potential of edge devices through quantization, compilation and other technologies. This path not only consolidates the dominant position of YOLO in the field of real-time detection, but also promotes the paradigm shift of object detection technology from the cloud to the terminal, providing a solid technical base for low-power and high-real-time applications in the era of Intelligent Internet of Things (AIoT).

5.1.3 Innovation

As the latest iteration version, YOLO11 inherits the core idea of this technical vein, while deeply optimizing for the resource constraints of embedded systems. Its innovation lies in:

1. Dynamic modular design: for example, C3k2 module supports runtime switching of convolution kernel size to flexibly adapt to the computing power requirements of different scenarios [45].
2. Hardware-aware lightweight: memory access overhead is reduced by depthwise separable convolution (DWConv) and group attention mechanism (C2PSA) [45];
3. End-to-end deployment friendliness: it supports TensorRT, ONNX and other compilation tool chains, and combines INT8 quantization to achieve better compression of model size and inference speed.

5.1.4 Details of YOLOv11

You Only Look Once Version 11 (YOLO11) is a new generation of object detection model launched by the team of Ultralytics in 2024 [8]. By integrating multi-scale feature fusion, dynamic attention mechanism and lightweight module design, the detection accuracy and computational efficiency are significantly improved. Starting from the core principles of the model architecture, later paragraphs will systematically expound its complete operation process from input to output, and deeply analyzes its deployment advantages in embedded systems. As the figure shows [46], the whole

process can be divided into four stages: feature extraction (backbone) → feature fusion (neck) → target prediction (head) → post-processing. The following is a detailed description of its operation logic combined with specific modules.

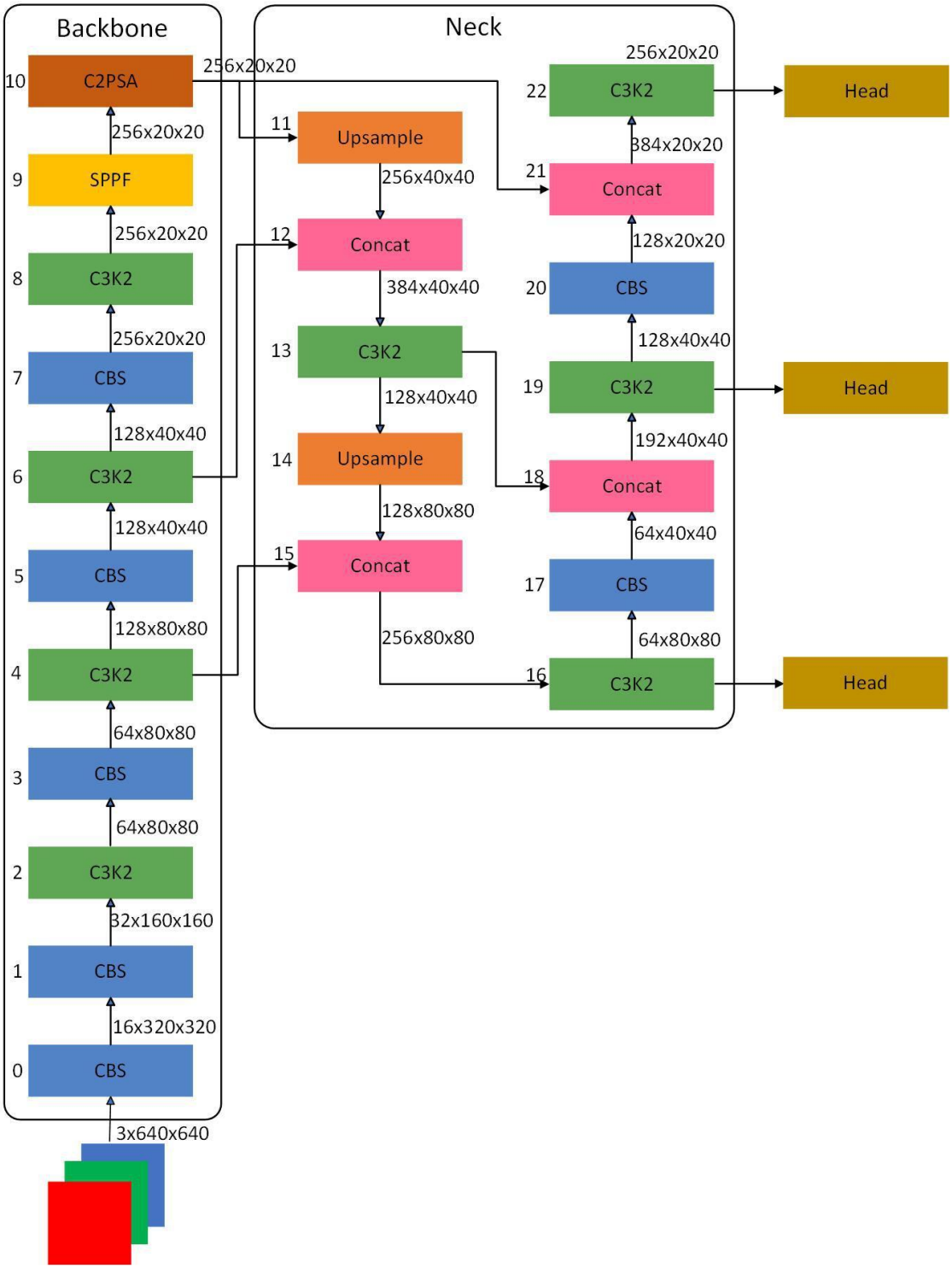

Figure 5-1- 1 Architecture of YOLO11

1. Feature Extraction: Optimal design of backbone networks

The Backbone network of YOLO11 is the core module of its feature extraction, and its core innovation is the adoption of an improved **CSP (Cross Stage Partial)** structure. This structure divides the input feature map into two parts, one part is directly passed to the subsequent layer, and the other part is merged after intensive convolution operation. The mathematical nature of this design can be described as follows:

$$X_{Out} = Concat(X_{Pass}, ConvBlock(X_{Process}))$$

Among which X_{Pass} retains the original feature information, and $X_{Process}$ extracts high-order semantic features through convolutional layers. This split-process-merge mechanism not only alleviates the vanishing gradient problem, but also improves the efficiency of feature reuse by reducing repeated calculations.

On the basis of CSP, YOLO11 further introduces the **C3k2** module, whose core idea is to achieve flexible feature extraction by **dynamically selecting the convolution kernel size**. For example, 5×5 deptwise separable convolution (DSConv) is enabled in complex scenes (e.g., dense crowd detection) to expand the receptive field to capture a wider range of context information; In simple scenes (such as industrial part detection), the 3×3 standard convolution is switched to reduce the computational overhead. As a continuation of the CSP concept, the input feature map is divided into two parts, one part retains the shallow features through the identity map, and the other part extracts the deep features through multiple Bottleneck or C3k modules (variable convolution kernel,), and finally concatenates and fuses [45]. Mathematically, it can be expressed as follows:

$$X1, X2 = split(X, dim = C)$$

$$Y = Concat(X1, F_{C3k2}(X2))$$

When the C3k module is enabled, it expands the receptive field with different convolution kernels (e.g. 3×3, 5×5).

After extracting features from multi-layer C3k2 module, the feature map goes into **Spatial Pyramid Pooling Fast (SPPF)** module. SPPF is an improvement of the traditional SPP module: traditional SPP needs multiple pooling kernels (such as 3×3, 5×5, 7×7) to fuse multi-scale information, but the computational complexity is high. SPPF concatenates the original features with a single Max pooling kernel (e.g., 5×5) to achieve efficient multi-scale fusion:

$$SPPF(F) = Conv(Concat(F, MaxPool(F, kernel = 5)))$$

By replacing traditional parallel pooling with serial Max pooling layers such as 5×5 pooling repeated three times, the computational complexity is reduced from $O(CHW \cdot \sum ki2)$（SPP）to $O(CHW \cdot max(k)2)$, preserving multi-scale features while reducing redundant calculations [48].

After SPPF output, the feature map enters **C2PSA** module (Cross Stage Partial with Pyramid Squeeze Attention), consisting of CSP structure and Position Sensitive Attention (PSA), a location-sensitive attention mechanism designed for visual tasks, which aims to enhance the perception ability of deep neural networks to spatial features. This module innovatively combines the traditional attention mechanism with the position encoding, so that the model can more effectively capture and utilize the spatial information in the image. At the same time, due to the introduction of CSP, the complex attention mechanism is only applied to some channels, which reduces the computational cost while maintaining the performance. The mathematical expression of C2PSA module can be summarized as follows [49]:

$$Y = f_{C2PSA}(X) = f_{cv2}(concat(A, f^{n}_{PSABlock}(B)))$$

This mathematical representation clearly describes how the C2PSA module enhances feature representation capabilities through channel separation and location-sensitive attention mechanisms.

2. Feature Fusion: Multi-Scale Context Enhancement

The neck Network adopts an improved PAN (Path Aggregation Network) structure, and multi-scale feature fusion is realized by up-sampling and feature stitching. The C3k2 module is introduced to replace the traditional C2f module to optimize the feature transfer efficiency through grouped convolution and channel compression. The up-sampled features are concatenated with the middle and high-level features output by the backbone network, and then processed by C3k2 to enhance the fusion of semantic information and detail information [49].

3. Target Prediction: Lightweight and multi-task adaptation

The head network continued the design of YOLOv8, but introduced depthwise separable convolution (DWConv) to further lightweight. DWConv is an efficient form of Convolution that decomposes the standard Convolution into two steps: Depthwise Convolution and Pointwise Convolution. Specifically, depthwise convolution applies a convolution kernel on each input channel independently, while pointwise convolution combines these results by 1×1 convolution to form new feature maps. This

decomposition greatly reduces the number of parameters and the amount of computation. Taking the standard convolution as an example, assuming that the number of input channels is C, the number of output channels is N, and the size of the convolution kernel is $k \times k$, then the calculation amount of the standard convolution is $C \times N \times k \times k$. However, in DWConv, the computation of depth convolution is $C \times k \times k$, and the computation of pointwise convolution is $C \times N$, and the total computation is significantly reduced. For YOLO11, this not only improves inference speed, but also reduces power consumption, allowing the model to run efficiently even on resource-constrained devices. [49]

5.2 Model Quantization

The YOLO11 model deployment to the main development board goes through the following steps: Conversion to ONNX format, setup of runtime environment, PTQ (Post-training Quantization) quantization, I/O validation, and final migration and testing. This section focuses on the model quantization process and key concepts involved, while practices and results will be covered in later sections.

5.2.1 Optimization of the model processing flow

In the process of optimizing YOLOv11 model, the main focus is how to reduce unnecessary computation and speed up inference, especially in the deployment phase. In the following, this process is parsed in detail, and each step is explained in combination with specific operations and formula derivation.

1. Filter using the monotonicity of the Sigmoid function

The YOLO model generates a large number of bounding boxes, each of which contains location information, class probabilities, and so on. In order to accurately calculate the loss function during training, it is necessary to calculate the score, category and xyxy coordinates completely for all bounding boxes. However, in the actual deployment, we only care about those qualified bounding boxes, that is, those with high confidence and satisfying the non-maximum suppression (NMS) condition. Therefore, by exploiting the monotonicity of the Sigmoid function, it is possible to filter out the potentially interesting bounding boxes before the computation, instead of performing a complete computation on all the bounding boxes.

The Sigmoid function has the form $\sigma(x) = \frac{1}{1+e^{-x}}$ and it is a strictly increasing function, meaning that if two inputs $x1$ and $x2$ satisfies $x1 < x2$, then the corresponding output will also satisfy $\sigma(x1) < \sigma(x2)$. Based on this property, we can set a threshold to filter out those bounding boxes whose prediction scores are lower than the threshold in advance, thus reducing the workload of subsequent calculation [50].

2. Speed up DFL and feature decoding using Python's numpy library

In the YOLOv11 model, DFL (Distribution Focal Loss) is used to improve the positioning accuracy of object detection. To further optimize performance, we can use Python's numpy library for advanced indexing, which will help us quickly filter out the objects of interest from a large number of bounding boxes.

This approach avoids the overhead of looping through each bounding box and greatly improves the efficiency. Similarly, a similar strategy can be adopted for the feature decoding process by screening out potential target bounding boxes in advance and then performing the necessary coordinate transformation and category recognition calculations only for these bounding boxes [50].

3. Optimization of the post-processing step

The post-processing stage consists of applying NMS to remove overlapping bounding boxes and finally determine the location and category of each detected object. According to the information provided, this part of the work can be done in about 5 ms on CPU with a single core, single frame, and single thread. This is due to the fact that the aforementioned filtering mechanism reduces the amount of data that needs to be processed, enabling efficient post-processing even in resource-constrained environments [50].

5.2.2 Horizon Sunrise 5 algorithm toolchain

The Horizon Rising Sun 5 (also known as Sunrise5, X5) algorithm tool chain is an algorithm solution developed based on the Horizon Rising Sun 5 generation processor. It contains two important components: model algorithm processing and embedded model prediction library [51].

The model algorithm deals with providing PTQ post-training quantization schemes and QAT quantization aware training schemes. The transformed fixed-point model can be executed on the Horizon computing platform after being processed by the model compilation tool. The embedded model prediction library provides a series of supporting interfaces for reasoning with fixed-point models.

Post-training Quantization (PTQ) is a common model quantization technique, which is applied after the model is trained to reduce the model size and improve the inference speed, while maintaining the performance of the model as much as possible. Post-training quantization has good results when deployed to resource-constrained devices, such as mobile devices and embedded devices. The approximate flow is shown in Figure [51]:

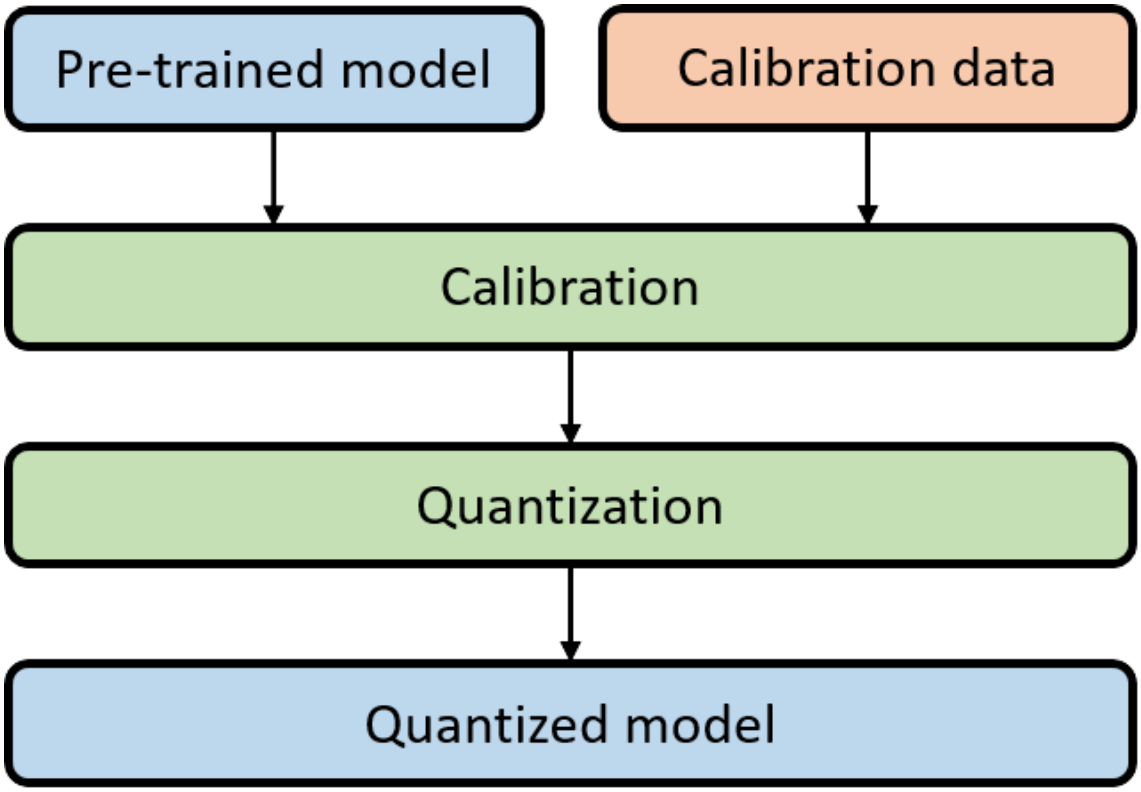

Figure 5-2- 1 Basic Workflow of PTQ

In general, the threshold, weights, etc. of the model downloaded from the official website or trained by yourself are float32 numbers, each occupying 4 bytes. However, when running on the embedded side, if the internal value can be converted from a floating-point number to a fixed-point number (int8), each number only occupies 1 byte, so the calculation amount can be greatly reduced. Therefore, converting a floating-point model to a fixed-point model with little or no loss can significantly improve its performance [51].

Model transformation is usually broken down into the following steps:

1. Check whether the model to be converted contains unsupported OP types.
2. Prepare 20-100 images used for calibration for the calibration operation in the conversion stage.
3. Convert the original floating-point model to a fixed-point model using the floating-point model conversion tool.
4. Evaluate the performance and accuracy of the transformed model to ensure that the accuracy of the fixed-point model after transformation is not much different from that before.

5. Run the model on the simulator/development board to verify the performance and accuracy of the model.

To facilitate the above steps, the toolchain provides built-in instructions. After porting the trained model to the deployed development environment and prepare the calibration dataset and configuration file. They can be accomplished by entering commands from the command-line interface. The specific operation steps will be introduced in detail in later sections.

5.3 Monocular distance measurement

In the robot grasping task, accurate target distance estimation is a key prerequisite for accurate operation. In this project, a monocular camera-based ranging method is proposed, which combines camera calibration, camera tilt Angle measurement and geometric ranging to achieve robust range estimation. This section will elaborate the core principle and implementation process of the method.

5.3.1 Camera calibration

Zhang's Camera Calibration method [52] is a widely used calibration method in the field of computer vision. Its core idea is to shoot a planar calibration board (such as a checkerboard) through multiple perspectives, and use the plane homography transformation to solve the camera internal parameters and distortion coefficients. This section will elaborate on its mathematical principle and operation process.

1. Calibration board and coordinate system definition

The plane checkerboard is used in the calibration process, and the world coordinate system F_w of its corner points is defined as the coordinate system in the plane of the calibration board:

- The plane of the calibration board is $Z_w = 0$
- The coordinates $(X_w, Y_w, 0)$ of each corner are known (based on the actual size of the checkerboard).

The relationship between the camera coordinate system F_c and the image coordinate system F_i is described by the internal reference matrix K:

$$\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Where f_x,f_y are the focal length (in pixels), (c_x, c_y) is the coordinate of the principal point, and s is the inter-axis tilt coefficient (usually 0).

2. Homography matrix and parameter solving

For each calibration plate attitude, the projection relation from the world coordinate system to the image coordinate system can be expressed as follows:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

Where r_1, r_2 are the first two columns of the rotation matrix R, t is the translation vector, and s is the scale factor. Define the homography matrix $H = K[r1\ r2\ t]$, then the projection equation is simplified as follows:

$$sm = HM$$

Where $m = [u, \quad v, \quad 1]^T$ is the image homogeneous coordinate, $M = [X_w, \quad Y_w, \quad 1]^T$ is the world homogeneous coordinate. By minimizing the reprojection error:

$$min_H \sum_i ||m_i - \hat{m}_i||^2$$

The homography matrix H corresponding to each pose can be solved. Furthermore, using the orthogonality constraint of the rotation matrix ($r_1{}^T r_2 = 0$, $||r_1|| = ||r_2|| = 1$), a closed solution equation about K was established, and the initial internal reference was solved by singular value Decomposition (SVD).

3. Nonlinear Optimization and Distortion Correction

After solving the initial internal parameters, the radial and tangential distortion models are introduced as follows:

$$\begin{cases} x_{\text{distorted}} = x(1 + k_1 r^2 + k_2 r^4) + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_{\text{distorted}} = y(1 + k_1 r^2 + k_2 r^4) + p_1(r^2 + 2y^2) + 2p_2 xy \end{cases}$$

Where (x, y) is the normalized image coordinates, $r2 = x2 + y2$, $k1, k2$ are the radial distortion coefficients, $p1, p2$ are the tangential distortion coefficients. The Levenberg-Marquardt algorithm [2] is used to jointly optimize the internal reference K and the distortion coefficient $d = [k1, k2, p1, p2]$, and the objective function is as follows:

$$\min_{K,\mathbf{d},R,\mathbf{t}} \sum_{i,j} \|\mathbf{m}_{ij} - \text{proj}(K, \mathbf{d}, R_i, \mathbf{t}_i, \mathbf{M}_j)\|^2$$

Where $proj(\cdot)$ is the projection function, R_i and t_i are the external parameters of the ith pose.

5.3.2 Tilt Angle calibration and geometric ranging model

The tilt Angle θ between the optical axis of the camera and the base plane of the robot arm is a key parameter that affects the ranging accuracy. This study proposes a dynamic calibration method based on multiple distance reference points. As shown in Figure 2, when the object is located at a distance from D_{ref}, the vertical pixel offset v of the image center satisfies the geometric relationship:

$$\theta = \arctan\left(\frac{H}{D_{ref}}\right) - \arctan\left(\frac{v}{f_y}\right)$$

Where H is the mounting height of the camera and fy is the vertical focal length. The optimal tilt Angle estimate θ^ is obtained by fitting five groups of (D_{ref}, v) data using the least square method. During camera tilt Angle calibration, the height of checkerboard calibration board is negligible. However, in actual ranging, the height of the object itself will greatly affect the accuracy of ranging. For the specific principle, refer to the figure below:

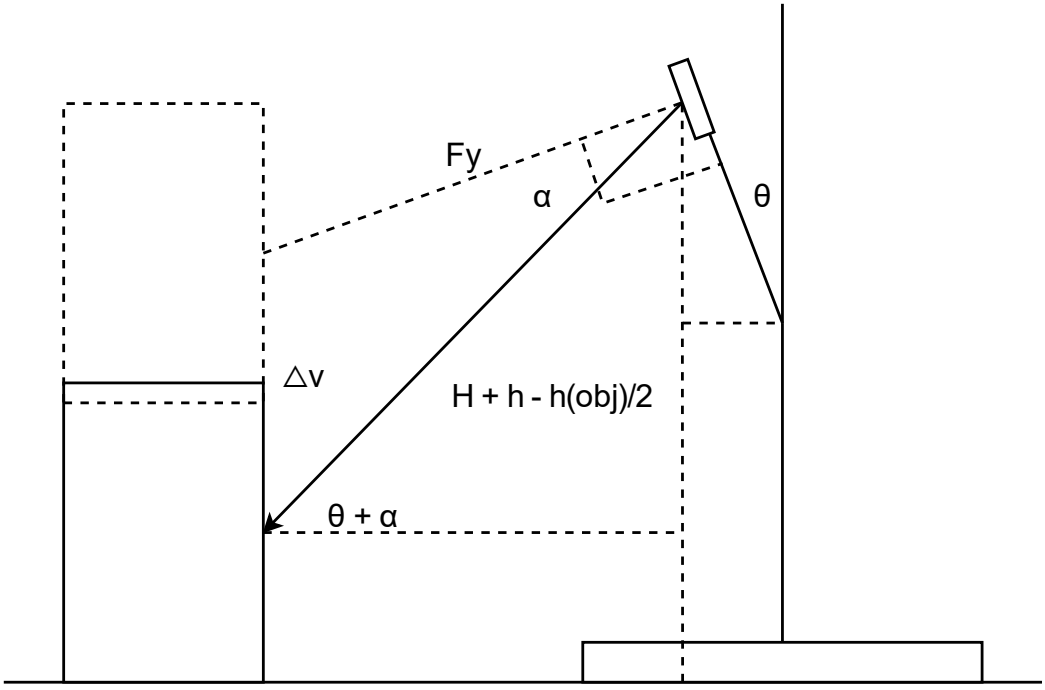

Figure 5-3- 1 Coordinate of camera

Therefore, it is necessary to measure the height of the object used in the test to form a data set, so that the measured distance is as close as possible to the actual value.
As shown in the figure, the α Angle (pitch Angle offset) reflects the Angle of the object in the vertical direction away from the optical axis of the camera, and its calculation depends on the vertical pixel offset of the object in the image (Δv) and the focal length of the camera (f). In the following pinhole camera model [53],

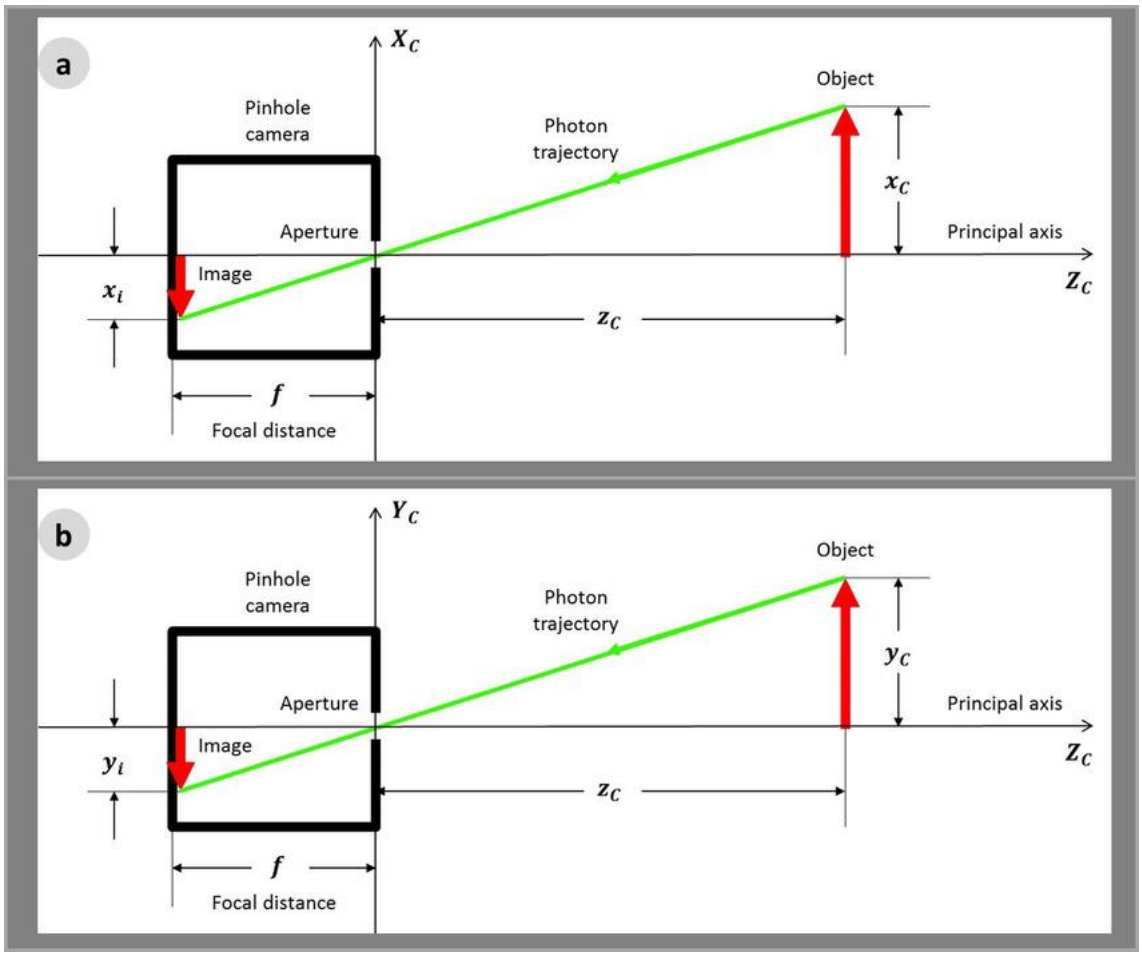

Figure 5-3- 2 Pinhole camera model

the coordinates of the 3D space point $P(X_c, Y_c, Z_c)$ projected onto the image plane are as follows.

$$u = f_x \cdot \frac{X_c}{Z_c} + c_x, \quad v = f_y \cdot \frac{Y_c}{Z_c} + c_y$$

Among them:

(u, v) are pixel coordinates, f_x, f_y are the focal length (in pixels), (c_x, c_y) is the image principal point (usually the image center). Suppose the object is located directly in front of the optical axis ($X_c = 0$), and its vertical coordinate is Y_c. The vertical pixel offset projected onto the image is:

$$\Delta v = v - c_y = f_y \cdot \frac{Y_c}{Z_c}$$

According to the geometric relation (Figure 4-2-3), the pitch Angle α satisfies:

$$\tan(\alpha) = \frac{Y_c}{Z_c}$$

Combined with the above formula, we can obtain:

$$\Delta v = f_y \cdot \tan(\alpha) \quad \Rightarrow \quad \alpha = \arctan\left(\frac{\Delta v}{f_y}\right)$$

Considering the figure 4-2-2 again, the total pitch Angle between the line of sight from the camera optical center O_c to the object point P and the base plane is $\theta + \alpha$. In triangle O_cPQ:

$$\tan(\theta + \alpha) = \frac{H_{\text{eff}}}{D}$$

Where $H_{eff} = H + h - h_{obj}/2$ is the effective vertical height from the camera to the theoretical vertical center of the object. The solution is:

$$D = \frac{H_{\text{eff}}}{\tan(\theta + \alpha)} = \frac{H + h_{\text{plane}} - \frac{h_{\text{obj}}}{2}}{\tan\left(\theta + \arctan\left(\frac{\Delta v}{f_y}\right)\right)}$$

At this point, the derivation of the core formula of monocular distance measurement is completed. Through a rigorous mathematical model and optimization strategy, this method achieves high-precision and low-cost camera calibration, which lays a foundation for subsequent vision measurement tasks.

5.4 Inverse kinematics

The inverse kinematics of robot arm is the core theoretical framework to realize the accurate control of the end effector in robotics, and its mathematical essence lies in solving the joint Angle parameters in the joint space from the end pose in Cartesian space. This section will systematically describe the mathematical modeling process of inverse kinematics, typical solution methods and their theoretical basis, and discuss the key issues in the project application.

5.4.1 Mathematical description

For a serial robot arm with n degrees of freedom, the forward kinematic model can be expressed as a nonlinear mapping as follows:

$$\mathbf{T}_n^0 = \prod_{i=1}^{n} \mathbf{T}_i^{i-1}(q_i)$$

Where $T_i - 1 \in SE(3)$ is the homogeneous transformation matrix between adjacent links, and q_i is the ith joint variable (rotation or translation). The inverse kinematics problem is to solve the equation:

$$f(\mathbf{q}) = \mathbf{T}_{desired} \quad \mathbf{q} \in \mathbb{R}^n$$

This equation is strongly nonlinear, the existence of its solution depends on the workspace of the robot arm, and the uniqueness of the solution is affected by the redundancy of the robot arm configuration [54].

5.4.2 Denavit-Hartenberg parameterization method

The Denavit-Hartenberg (DH) parameter method is used to systematically describe the link coordinate system to establish an accurate kinematic model. For any adjacent link, the relative pose is defined by four geometric parameters:

- Link length a_i (measured along the x_i axis)
- Link Angle a_i (rotation around the x_i axis)
- Link offset d_i (translation along z_{i-1} axis)
- Joint angles θ_i (rotate about z_{i-1} axis)

The corresponding homogeneous transformation matrix is as follows:

$$\mathbf{T}_i^{i-1} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The chain multiplication rule can be used to obtain the total pose matrix of the end-effector, which establishes the basis for the derivation of the inverse solution [55]. Considering the actual robot arm, the coordinate system is established for each joint, which can finally be simplified to the model shown in the following figure:

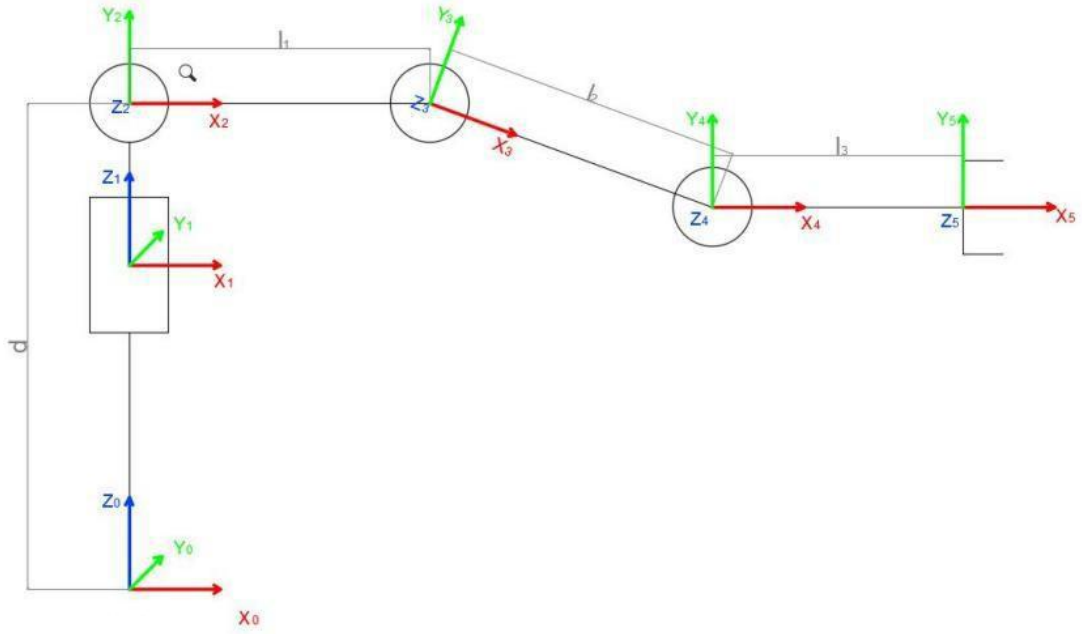

Figure 5-4- 1 Coordinate of robot arm

After the coordinate system is determined, the theoretical DH parameter table can be obtained:

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	d	θ_0
2	90	0	0	θ_1
3	0	l_1	0	θ_2
4	0	l_2	0	θ_3
5	0	l_3	0	0

Table 5-4- 1 DH parameters of robot arm

According to the formula mentioned earlier, each joint can be calculated at once, and finally the forward kinematics formula of the robot arm can be obtained:

$$ {}^{0}_{1}T = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 & 0 & 0 \\ \sin\theta_0 & \cos\theta_0 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}_{2}T = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Then, according to the rotation matrix of each joint, the coordinates of the ends can be obtained according to the following formula:

$$ {}^{2}_{3}T = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & l_1 \\ \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}_{4}T = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & l_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{4}_{5}T = \begin{bmatrix} 1 & 0 & 0 & l_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

5.4.3 Inverse kinematics analysis

The basic idea of inverse kinematics is to reverse the forward kinematics, in the case of a robot arm, which is to find the Angle of each joint given the position and orientation of the end-effector. The three-dimensional motion of the robot arm is relatively complex. In order to simplify the model, the rotation joints of the lower pan-tilt can be ignored and the kinematics analysis can be carried out on the two-dimensional plane. As shown in the following figure, where θ_0, θ_1, and θ_2 are the angles of each joint and are unknown quantities. $P(x, y)$ is the pose representation of the end-effector, x and y are the coordinates in the OXY plane, and α is the Angle between the end-effector and the horizontal plane.

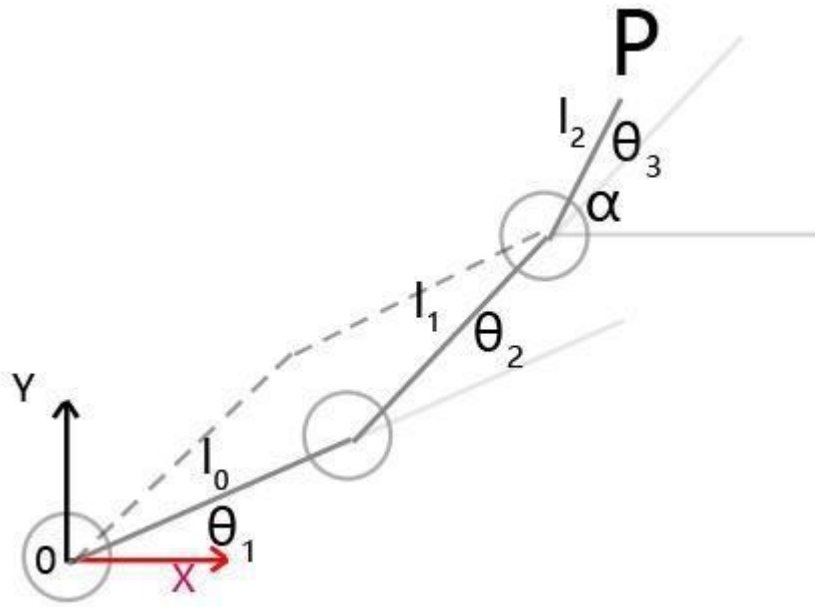

Figure 5-4- 2 Simplified model of robot arm coordinate

We use geometric analysis here:

$$x = l_0 \cos\theta_1 + l_1 \cos(\theta_1+\theta_2) + l_2 \cos(\theta_1+\theta_2+\theta_3) \quad (1)$$

$$y = l_0 \sin\theta_1 + l_1 \sin(\theta_1+\theta_2) + l_2 \sin(\theta_1+\theta_2+\theta_3) \quad (2)$$

$$\alpha = \theta_1+\theta_2+\theta_3 \quad (3)$$

In the following, the above system of equations is simplified by substituting equation (3) into (2) and (1):

$$x = l_0 \cos\theta_1 + l_1 \cos(\theta_1+\theta_2) + l_2 \cos(\alpha) \quad (4)$$

$$y = l_0 \sin\theta_1 + l_1 \sin(\theta_1+\theta_2) + l_2 \sin(\alpha) \quad (5)$$

For convenience of calculation, let:

$$m = l_2 \cos(\alpha) - x$$

$$n = l_2 \sin(\alpha) - y$$

By simplifying equations (4) and (5), the following is obtained:

$$l_1 = (l_0 \cos\theta_1 + m)^2 + (l_0 \sin\theta_1 + n)^2 \quad (6)$$

By calculating:

$$\sin\theta_1 = (-b \pm \sqrt{b^2 - 4ac}) / 2a$$

Where:

$$k = (l_0^2 - l_1^2 - m^2 - n^2)/2l_1$$

$$a = m^2 + n^2$$

$$b = -2nk$$

$$c = k^2 - m^2$$

In the same way, θ_2 can be obtained, which completes the calculation of the inverse kinematics. Obviously, there are two correct solutions at the end, which can be seen according to the dashed part of Figure 5-4- 2. Generally, the solution of the dotted part is selected, which can reduce the force on each joint.

5.5 Built-in instruction program of the robot arm

The traditional PWM servo controls the rotation of the servo by sending PWM signal through the single chip microcomputer, but the bus servo developed by the manufacturer has a main control chip inside, which has written the program controlled by PWM signal. It only needs to send string instructions through the serial port to control the servo. The chip inside the servo can also detect the working state of the servo, so the Angle of the servo can also be read through the serial port to switch the working state. The basic servo control commands are shown in the following table:

#	Servo control command	Explanation	Notation
01	#IndexPpwmTtime! For example, control the servo to 1500 position #000P15000T1000!	To control a single servo command, Index is 3 bits, pwm is 4 bits, time is 4 bits, and the insufficient is filled with "0", a total of 15 bits of data	1. All instructions are input in English format 2. Pay attention to keep the baud rate consistent when using instructions. The default baud rate of the control board is 115200
02	{#000P1500T1000!#001P0900T1000!}	To control multiple servo commands, multiple single servo commands are put together and sealed with {}	
03	$DST!	Servo stop command, **all servos** stop at the current position	

		after receiving the command	
04	$DST:x!	**Servo x** stops at current position	
05	$DGS:0!	Call action group G0000, provided the action has been stored	
06	$DGT:0-10,1!	Call the action group instruction, call the action 0-10 group to execute once, if you want to loop execution, it is changed to 0,0 represents loop execution	
07	$RST!	Reset all servos to initial position	

Table 5-5- 1 Commands' format of robot arm controller

Chapter 6 Practices and Results

This part will be described in detail according to the process of the project previously introduced in the scope of project implementation, including 1) Training and optimization of a YOLOv11 object detection model for waste recognition, 2) ONNX model format conversion and quantization, 3) Development of a WiFi-based video streaming subsystem, 4) Implementation of monocular distance measurement, 5) Inverse kinematics programming for robotic manipulation, 6) Design of a robust UART communication protocol, and 7) System integration and performance validation.

6.1 Model training and optimization

The work of training the model is mainly divided into two parts, preparing the appropriate training set and adjusting the training parameters according to the platform used for training.

6.1.1 Dataset

For the training set selection, the dataset need to first meet the requirement of the YOLO format, which can be normal YOLO format or VOC2007 format that can be converted. Secondly, the training set needs to cover common types of domestic waste to meet the practical requirements. And each kind of pictures should be in different angles, lighting and other conditions as much as possible to enhance the adaptability of the model to different environments. Finally, the labeling of the target object needs to ensure accuracy and avoid confusion. After screening, the domestic waste data set of Huawei 2020 Cloud competition is finally selected, which contains 44 types of common domestic waste, which can basically meet the above requirements.

Since the original format of the dataset is V0C2007 format, it needs to be transformed to be directly used for training. The file structure of VOC2007 is as follows [56]:

Figure 6-1- 1 File structure of VOC2007 dataset

The markup file is in.xml format, and the training set and the validation set are not divided in the initial state, so the.xml format markup file needs to be converted into.txt format first, and then the data set is divided according to a certain proportion. The key code for the format conversion is as follows:

```python
def convert(size, box):
    dw = 1. / (size[0])
    dh = 1. / (size[1])
    x = (box[0] + box[1]) / 2.0 - 1
    y = (box[2] + box[3]) / 2.0 - 1
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x * dw
    w = w * dw
    y = y * dh
    h = h * dh
    return (x, y, w, h)

解释代码 | 注释代码 | 生成单测 | ×
def convert_annotation(xmlpath, xmlname):
    with open(xmlpath, "r", encoding='utf-8') as in_file:
        txtname = xmlname[:-4] + '.txt'
        txtfile = os.path.join(txtpath, txtname)
        tree = ET.parse(in_file)
        root = tree.getroot()
        filename = root.find('filename')
        img = cv2.imdecode(np.fromfile('{}/{}.{}'.format(imgpath, xmlname[:-4], postfix), np.uint8), cv2.IMREAD_COLOR)
        h, w = img.shape[:2]
        res = []
        for obj in root.iter('object'):
            cls = obj.find('name').text
            if cls not in classes:
                classes.append(cls)
            cls_id = classes.index(cls)
            xmlbox = obj.find('bndbox')
            b = (float(xmlbox.find('xmin').text), float(xmlbox.find('xmax').text), float(xmlbox.find('ymin').text),
                 float(xmlbox.find('ymax').text))
            bb = convert((w, h), b)
            res.append(str(cls_id) + " " + " ".join([str(a) for a in bb]))
        if len(res) != 0:
            with open(txtfile, 'w+') as f:
                f.write('\n'.join(res))
```

Figure 6-1- 2 convert.py

In VOC format, the actual class name and absolute pixel coordinates are used for the top-left (xmin, ymin) and bottom-right (xmax, ymax) corners, as shown in the following figure:

Figure 6-1- 3 label file in VOC format

In the YOLO format, the category number and the coordinate of the box in terms of its center (x_center, y_center) and width, height are used to define object's category and the position in picture, as shown in the following screenshot:

```
35 0.5449074074074074 0.4996527777777778 0.5324074074074074 0.9993055555555556
24 0.4740740740740741 0.4996527777777778 0.9481481481481482 0.9993055555555556
```

Figure 6-1- 4 label file in VOC format

As a result, the coordinates need to be normalized and converted in the convert function. And for categories, it is required to build an index to change the name to corresponding number. Finally, write the changes to .txt file with the same filename.

After the format conversion, the data set should be divided into a training set for model training and a validation set for accuracy verification. The approximate file structure is as follows [56]:

```
dataset
├─images
│  ├─train
│  │    ├─ flip_mirror_himg0026393.jpg
│  │    ├─ flip_mirror_himg0026394.jpg
│  │    ├─ flip_mirror_himg0026395.jpg
│  │    ├─ flip_mirror_himg0027314.jpg
│  │    ├─ flip_mirror_himg0027315.jpg
│  │    └─flip_mirror_himg0027316.jpg
│  │
│  └─val
│     ├─ flip_mirror_himg0027317.jpg
│     └─flip_mirror_himg0027318.jpg
│
└─labels
    ├─train
    │    ├─ flip_mirror_aimg0025023.txt
    │    ├─ flip_mirror_aimg0025024.txt
    │    ├─ flip_mirror_aimg0025025.txt
    │    ├─ flip_mirror_aimg0025026.txt
    │    ├─ flip_mirror_aimg0025027.txt
    │    └─ flip_mirror_aimg0025028.txt
    │
    └─val
         ├─ flip_mirror_aimg0025029.txt
         └─flip_mirror_aimg0025030.txt
```

Figure 6-1- 5 File structure of YOLO dataset

It is required to put the images and labels files in an images and labels folder, which will be split into training and validation sections. The specific process is to set the proportion of the validation set, create a folder according to the above file structure, and then shuffle the pictures to randomly divide them according to the proportion, and finally complete the file copy, as shown below:

```python
val_size = 0.2
postfix = 'jpg'
imgpath = r'd:\study\rdk_docker\dataset\trainval\VOC2007\JPEGImages'
txtpath =  r'd:\study\rdk_docker\dataset\trainval\VOC2007\Yolo_labels'

output_train_img_folder =r'd:\study\rdk_docker\dataset\trainval\VOC2007\train\images'
output_val_img_folder =  r'd:\study\rdk_docker\dataset\trainval\VOC2007\val\images'
output_train_txt_folder =  r'd:\study\rdk_docker\dataset\trainval\VOC2007\train\labels'
output_val_txt_folder =  r'd:\study\rdk_docker\dataset\trainval\VOC2007\val\labels'

os.makedirs(output_train_img_folder, exist_ok=True)
os.makedirs(output_val_img_folder, exist_ok=True)
os.makedirs(output_train_txt_folder, exist_ok=True)
os.makedirs(output_val_txt_folder, exist_ok=True)


listdir = [i for i in os.listdir(txtpath) if 'txt' in i]
train, val = train_test_split(listdir, test_size=val_size, shuffle=True, random_state=0)

#todo：需要test放开

# train, test = train_test_split(listdir, test_size=test_size, shuffle=True, random_state=0)
# train, val = train_test_split(train, test_size=val_size, shuffle=True, random_state=0)

for i in train:
    img_source_path = os.path.join(imgpath, '{}.{}'.format(i[:-4], postfix))
    txt_source_path = os.path.join(txtpath, i)

    img_destination_path = os.path.join(output_train_img_folder, '{}.{}'.format(i[:-4], postfix))
    txt_destination_path = os.path.join(output_train_txt_folder, i)

    shutil.copy(img_source_path, img_destination_path)
    shutil.copy(txt_source_path, txt_destination_path)

for i in val:
    img_source_path = os.path.join(imgpath, '{}.{}'.format(i[:-4], postfix))
    txt_source_path = os.path.join(txtpath, i)

    img_destination_path = os.path.join(output_val_img_folder, '{}.{}'.format(i[:-4], postfix))
    txt_destination_path = os.path.join(output_val_txt_folder, i)

    shutil.copy(img_source_path, img_destination_path)
```

Figure 6-1- 6 divide.py

After the data set is adjusted, a. yaml configuration file is also needed for model training, which needs to define the directories of the training set and the validation set, the number of categories and the corresponding name of each category, as shown in the following figure:

trash.yaml

```yaml
train: ./VOC2007/train/images
val: ./VOC2007/val/images
nc: 24
names: ['Disposable Fast Food Box', 'Book Paper', 'Plastic Utensils', 'Plastic Toys', 'Dry Battery', 'Express Paper Bag', 'Plug Wire', 'Can', 'Peel and Pulp', 'Stuffed Toy', 'Defiled Plastic', 'Contaminated paper', 'Toilet care products', 'Cigarette butts', 'Carton box', 'Tea residue', 'Cai Bang Cai Ye', 'Egg Shell', 'Sauce Bottle', 'Ointment', 'Expired Medicine', 'Metal Food Cans', 'edible oil drums', 'drink bottles']
```

Figure 6-1- 7 trash.yaml

Considering the limited weight and size of objects that can be gripped by the robotic arm, some categories were deleted based on the original data set. The files for category deletion screening are as follows:

filter.py > ...

```python
def filter_yolo_dataset(dataset_root, classes_to_remove, yaml_path):
    """
    筛选Yolo数据集，删除包含指定类别的标记文件及其对应的图片

    参数:
    dataset_root: 数据集根目录路径
    classes_to_remove: 需要删除的类别索引列表
    yaml_path: yaml配置文件路径
    """
    # 将classes_to_remove转换为集合，以便快速查找
    classes_to_remove_set = set(classes_to_remove)

    # 更新后的类别映射
    class_mapping = {}
    current_idx = 0
    for i in range(100):  # 假设最多100个类别
        if i not in classes_to_remove_set:
            class_mapping[i] = current_idx
            current_idx += 1

    # 处理train和val文件夹
    for split in ['train', 'val']:
        labels_dir = os.path.join(dataset_root, split, 'labels')
        images_dir = os.path.join(dataset_root, split, 'images')

        if not os.path.exists(labels_dir) or not os.path.exists(images_dir):
            print(f"警告: {labels_dir} 或 {images_dir} 不存在")
            continue

        # 遍历所有标记文件
        for label_file in os.listdir(labels_dir):
            if not label_file.endswith('.txt'):
                continue

            label_path = os.path.join(labels_dir, label_file)

            # 读取标记文件内容
            with open(label_path, 'r') as f:
                lines = f.readlines()

            # 检查是否包含需要删除的类别
            should_remove = False
```

Figure 6-1- 8 filter.py (1)

Figure 6-1- 9 filter.py (2)

As shown in the figure above, the tag file is traversed firstly, and the category that needs to be deleted corresponding to the category sequence number is marked, and then deleted uniformly after the traversal is completed. If it is a category that does not need to be deleted, the category number is updated. After the changes are complete, call the *update_yaml_config* function to update the number of categories in the. yaml file and the corresponding names of the categories:

Figure 6-1- 10 filter.py (3)

6.1.2 Model training

In order to compare and choose the best, and considering the limited budget of the project, the project selected two cloud computing platforms that can be used for free to train the model, Kaggle and HPC Fun. Kaggle provides 30 hours per week, up to 9 hours of free NVIDIA TESLA P100 GPU (16GB memory) training time per session [57]. HPC Fun offers unlimited free training time on NVIDIA Tesla P4 GPU (8G) with up to 24 hours of use per session [58].

According to different GPU memory, it is necessary to appropriately change the batch size and the number of workers (child processes) to ensure a certain efficiency without exceeding the capacity. Here are the training files for Kaggle and HPC Fun:

```python
import warnings
warnings.filterwarnings('ignore')
from ultralytics import YOLO

if __name__ == '__main__':
    model = YOLO(model=r'./ultralytics/cfg/models/11/yolo11s.yaml')  # 模型配置文件)
    model.load('yolo11s.pt') # 加载预训练权重,改进或者做对比实验时候不建议打开，因为用预训练模型整体精度没有很明显的提升
    model.train(data=r'/kaggle/input/huawei2020-trash/trash.yaml',  # 数据集路径
                imgsz=640,
                epochs=100,
                batch=64,
                workers=2,
                device='',
                optimizer='SGD',
                close_mosaic=50,
                resume=True,
                project='/kaggle/working/runs/train',
                name='exp',
                single_cls=False,
                cache=False,
                )
```

Figure 6-1- 11 train.py for Kaggle

```python
import warnings
warnings.filterwarnings('ignore')
from ultralytics import YOLO

if __name__ == '__main__':
    model = YOLO(model=r'./ultralytics/cfg/models/11/yolo11s.yaml')  # 模型配置文件)
    model.load('yolo11s.pt') # 加载预训练权重,改进或者做对比实验时候不建议打开，因为用预训练模型整体精度没有很明显的提升
    model.train(data=r'../../trainval/trash.yaml',  # 数据集路径
                imgsz=640,
                epochs=100,
                batch=16,
                workers=4,
                device='',
                optimizer='SGD',
                close_mosaic=50,
                resume=False,
                project='runs/train',
                name='exp',
                single_cls=False,
                cache=False,
                )
```

Figure 6-1- 12 train.py for HPC Fun

It can be seen that the training file of the Kaggle platform sets the batch parameter larger, because it has larger video memory, and the batch size can be appropriately increased to speed up the training.

In addition to preparing the training file, the number of classes in the .yaml configuration file in the YOLO project folder should also be set according to actual number of sets, as shown in the *nc* variable in the code below:

ultralytics > cfg > models > 11 > ! yolo11.yaml

```yaml
# Ultralytics YOLO 🚀, AGPL-3.0 license
# YOLO11 object detection model with P3-P5 outputs. For Usage examples see https://docs.ultralytics.com/tasks/detect

# Parameters
nc: 24 # number of classes
scales: # model compound scaling constants, i.e. 'model=yolo11n.yaml' will call yolo11.yaml with scale 'n'
  # [depth, width, max_channels]
  n: [0.50, 0.25, 1024] # summary: 319 layers, 2624080 parameters, 2624064 gradients, 6.6 GFLOPs
  s: [0.50, 0.50, 1024] # summary: 319 layers, 9458752 parameters, 9458736 gradients, 21.7 GFLOPs
  m: [0.50, 1.00, 512] # summary: 409 layers, 20114688 parameters, 20114672 gradients, 68.5 GFLOPs
  l: [1.00, 1.00, 512] # summary: 631 layers, 25372160 parameters, 25372144 gradients, 87.6 GFLOPs
  x: [1.00, 1.50, 512] # summary: 631 layers, 56966176 parameters, 56966160 gradients, 196.0 GFLOPs
```

Figure 6-1- 13 yolo11.yaml

From the above figure, we can see that YOLO11 provides pre-trained models with different parameters, and the detailed data provided in the official documentation is as follows [8]:

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

Figure 6-1- 14 Diverse mode size of YOLO11

Considering the limitations of training time, equipment hardware resources and other factors, YOLO11s is selected as the training framework in the project to achieve a balance between practical effects and running performance.

After preparing the training files, it is required to upload the ultralytics (YOLO framework) project folder and data sets to the cloud computing platform. The following image shows the files uploaded on the Kaggle platform:

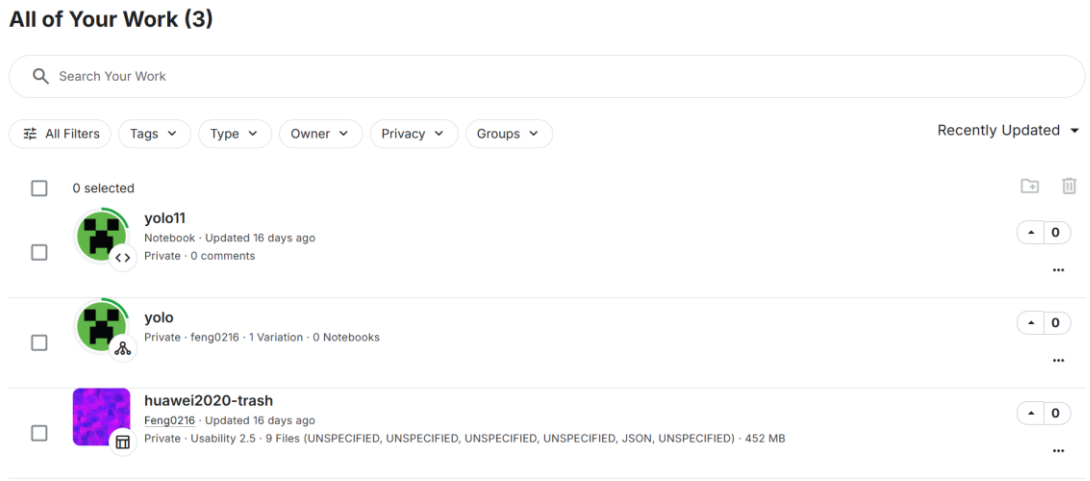

Figure 6-1- 15 Uploaded files on Kaggle

After uploading, it is necessary to create a notebook (yolo11 in the figure) for training, and the specific page is shown in the figure below:

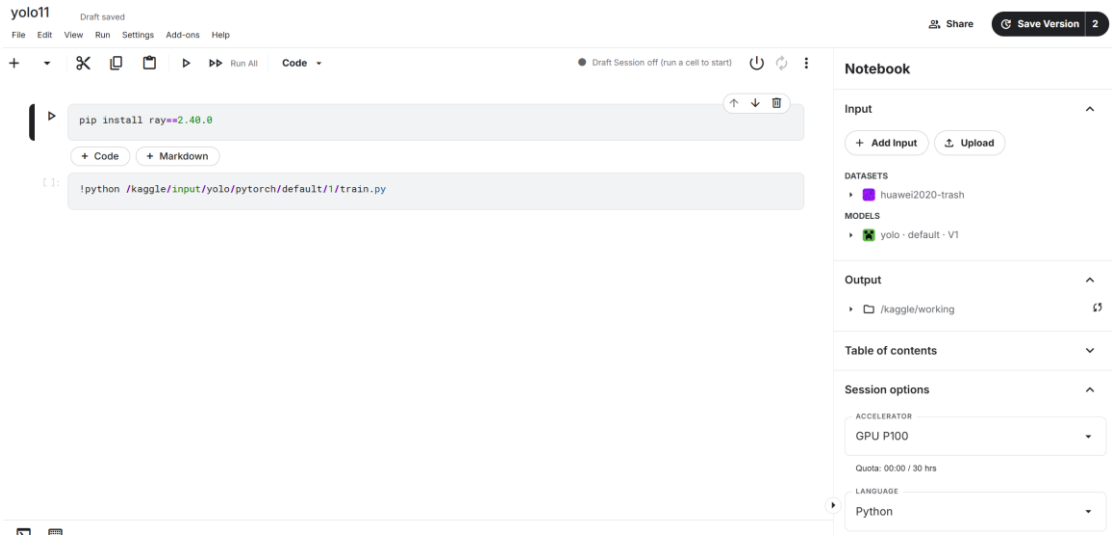

Figure 6-1- 16 Notebook page on Kaggle

In the top left is the command that will be executed on the command line. Make sure to install the required libraries and then run the training files. The input in the upper right corner needs to add the uploaded data set and project framework, and the P100 is selected as the graphics card used for training in the lower right corner. After the test runs smoothly, the operation is finished. Finally, a new Version is created by saving version in the upper right corner to ensure that the notebook can be stably connected to the server for model training in the background. The log of notebook is as follows:

Notebook Input Output Logs Comments (0)

```
In [2]: !python /kaggle/input/yolo/pytorch/default/1/train.py

Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help se
e https://docs.ultralytics.com/quickstart/#ultralytics-settings.
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11s.pt to 'yolo11s.p
t'...
100%|██████████████████████████████████████████████████| 18.4M/18.4M [00:00<00:00, 169M
B/s]
Transferred 493/499 items from pretrained weights
New https://pypi.org/project/ultralytics/8.3.97 available 😃 Update with 'pip install -U ultralytics'
Ultralytics 8.3.38 🚀 Python-3.10.12 torch-2.5.1+cu121 CUDA:0 (Tesla P100-PCIE-16GB, 16269MiB)
engine/trainer: task=detect, mode=train, model=./ultralytics/cfg/models/11/yolo11s.yaml, data=/kaggle/
input/huawei2020-trash/trash.yaml, epochs=100, time=None, patience=100, batch=64, imgsz=640, save=Tru
e, save_period=-1, cache=False, device=, workers=2, project=/kaggle/working/runs/train, name=exp, exis
t_ok=False, pretrained=yolo11s.pt, optimizer=SGD, verbose=True, seed=0, deterministic=True, single_cls
=False, rect=False, cos_lr=False, close_mosaic=50, resume=None, amp=True, fraction=1.0, profile=False,
freeze=None, multi_scale=False, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True, split=val, sav
e_json=False, save_hybrid=False, conf=None, iou=0.7, max_det=300, half=False, dnn=False, plots=True, s
ource=None, vid_stride=1, stream_buffer=False, visualize=False, augment=False, agnostic_nms=False, cla
sses=None, retina_masks=False, embed=None, show=False, save_frames=False, save_txt=False, save_conf=Fa
lse, save_crop=False, show_labels=True, show_conf=True, show_boxes=True, line_width=None, format=torch
```

Figure 6-1- 17 log of notebook yolo11 (1)

Notebook Input Output Logs Comments (0)

```
Logging results to /kaggle/working/runs/train/exp
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      16.4G     0.6884       2.97      1.241        148        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all       1723       2356      0.592       0.49      0.491      0.408

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/100      15.7G     0.6479      1.529      1.182        138        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all       1723       2356      0.435      0.337      0.304      0.216

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      3/100      15.9G     0.7823       1.71      1.274        143        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all       1723       2356      0.183      0.161     0.0848      0.045

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      4/100      16.4G     0.9416      1.964       1.39        130        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all       1723       2356      0.303      0.251      0.177     0.0981

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      5/100      16.4G     0.9185      1.877      1.369        172        640: 1
                 Class     Images  Instances      Box(P          R      mAP50  m
                   all       1723       2356      0.193      0.196     0.0947     0.0588
```

Figure 6-1- 18 log of notebook yolo11 (2)

After running the training file, the set parameters will be automatically identified for training, and the accuracy verification data of each round will also be printed in the record page.

The HPC Fun platform follows a similar operation. First, the YOLO framework folder and dataset were uploaded via *WinSCP*:

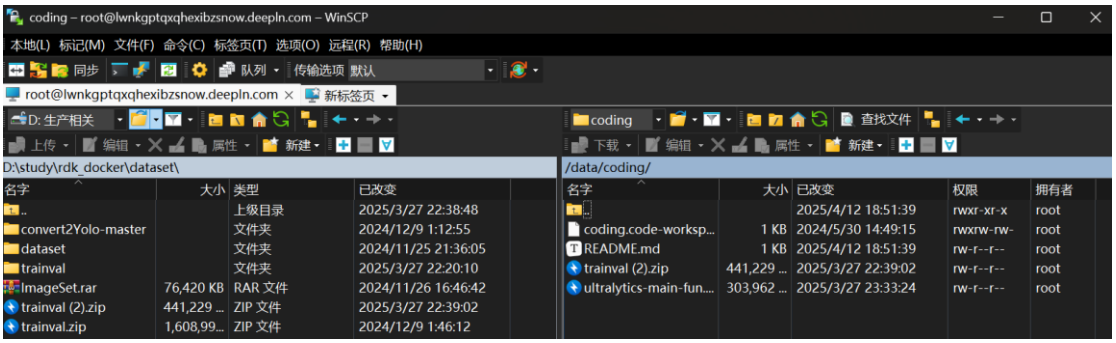

Figure 6-1- 19 Upload files by *WinSCP*

Then login to the cloud VSCode deployed by the platform to extract the file:

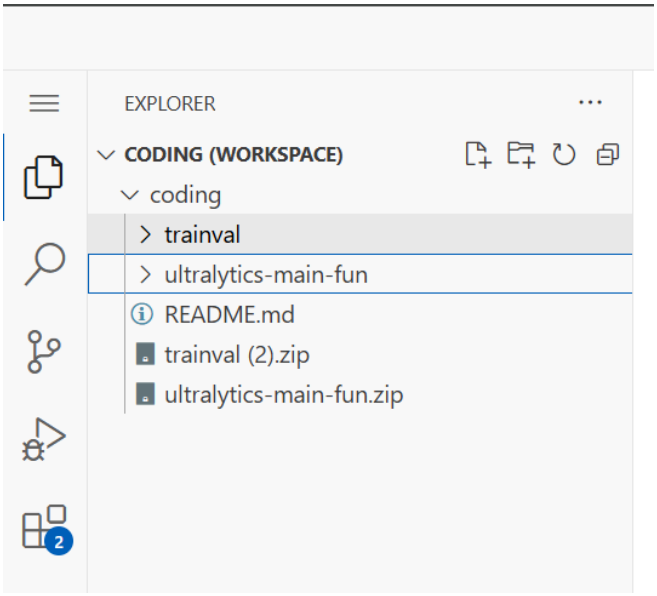

Figure 6-1- 20 Extract files on cloud VSCode

Finally, through the command line SSH login server for training, to ensure that it can run stably in the background:

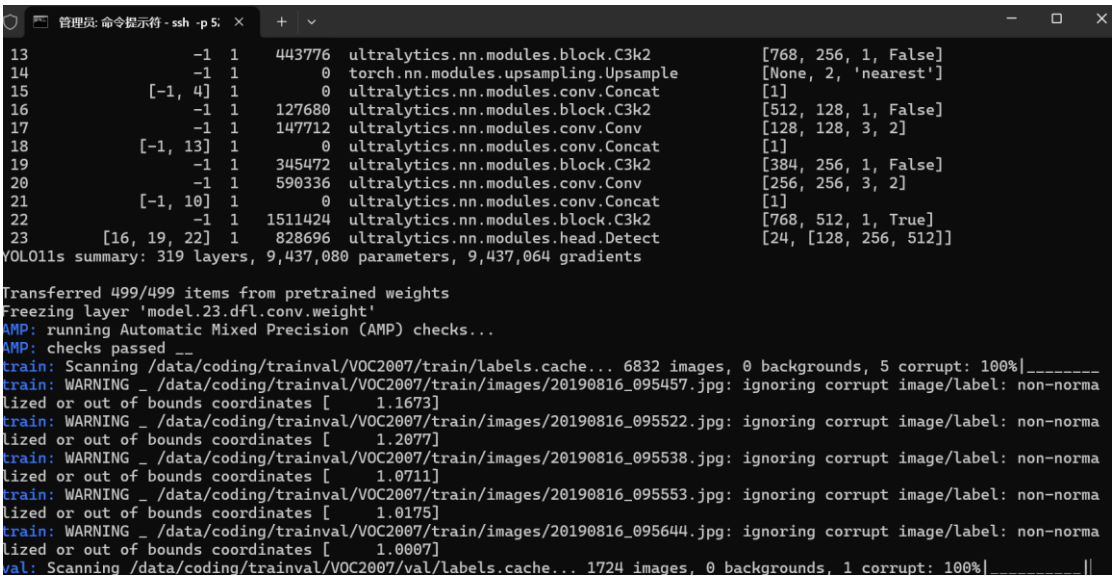

Figure 6-1- 21 Model training by SSH

After preliminary training, it was found that the accuracy of the category "Contaminated Paper" was significantly lower than the average. After checking the data set, it was found that the problem was that the original data set was not clear about the definition of this category, card paper, wrapping paper and tissue paper were all counted in this category, and the pictures used were often covered by other objects. This category was therefore specifically defined as used tissue. If the original picture only has paper other than tissue, it is deleted directly. If other labeled objects are included, only the labeled data of the objects corresponding to the "Contaminated Paper" category will be removed. Because of the size of the corresponding scripts, they are in a separate appendix and are not discussed here. After deleting, it is necessary to add some pictures of tissues with labels. At this time, the *labelImg* tool is used for data marking, as shown in the following figure:

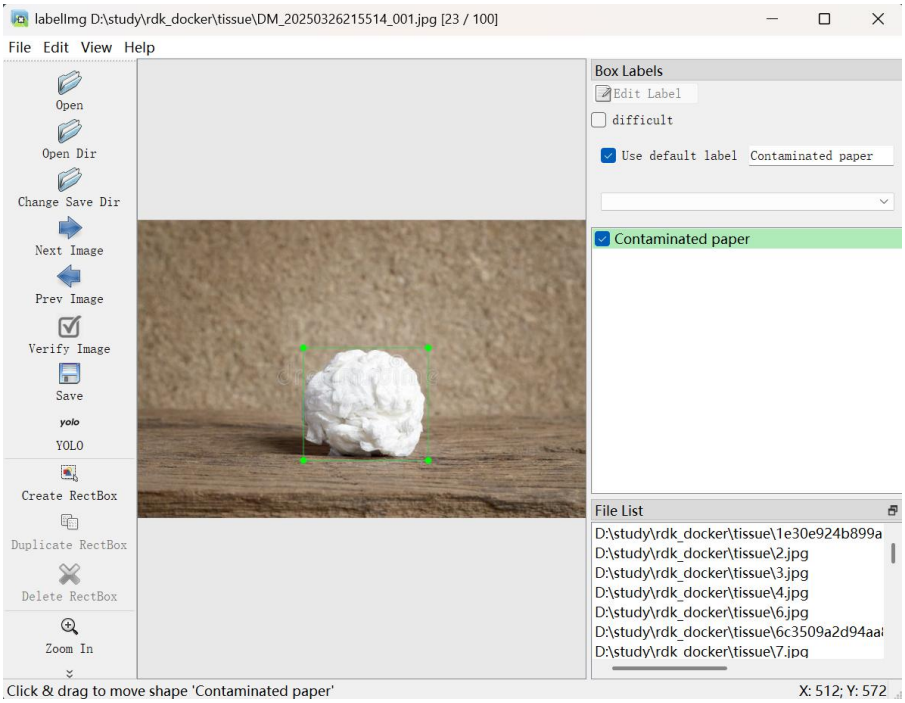

Figure 6-1- 22 Labeling image using *labelImg*

After the above dataset optimization, the mAP50 precision of the "Contaminated Paper" category increased from 0.224 to 0.758, and the mAP50 precision of the overall dataset increased from 0.783 to 0.81, under the condition that the training parameters were consistent and the final model accuracy basically converged, as shown in the following figure:

```
Validating train/exp2/weights/best.pt...
WARNING ⚠ validating an untrained model YAML will result in 0 mAP.
Ultralytics 8.3.38 🚀 Python-3.10.15 torch-2.4.1 CUDA:0 (Tesla P4, 8109MiB)
YOLO11s summary (fused): 238 layers, 9,422,088 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|__________| 27/27 [00:19<
                   all       1703       2325      0.842      0.697      0.783      0.693
Disposable Fast Food Box         42         50      0.776        0.6      0.633      0.519
           Book Paper         24         26      0.743      0.462      0.531      0.396
      Plastic Utensils        110        127      0.895      0.672      0.809      0.759
          Plastic Toys         65         76      0.836      0.763      0.879      0.795
           Dry Battery         67         87      0.947      0.825      0.884      0.755
     Express Paper Bag         35         35      0.968      0.867      0.961      0.934
             Plug Wire        149        186      0.797      0.608      0.695      0.539
                   Can         57         71      0.811      0.849      0.911      0.861
        Peel and Pulp        144        160      0.863      0.749      0.832       0.78
          Stuffed Toy        102        115      0.914      0.826      0.916      0.796
       Defiled Plastic        142        184      0.778      0.538      0.649      0.546
    Contaminated paper         10         10       0.52        0.2      0.224      0.218
  Toilet care products        150        201      0.833      0.761       0.81        0.7
      Cigarette butts         53         90      0.877      0.789      0.842      0.708
           Carton box         92        105      0.896      0.676       0.79      0.715
          Tea residue         45         47      0.955      0.915      0.939      0.903
      Cai Bang Cai Ye        118        146      0.906       0.76      0.831      0.735
            Egg Shell         53         66      0.801      0.576      0.705      0.581
         Sauce Bottle         77         89      0.822      0.708      0.801      0.711
              Ointment        70         77      0.913      0.821      0.928      0.822
     Expired Medicine        123        145      0.881      0.763       0.86      0.768
      Metal Food Cans         64         78      0.762      0.615      0.762      0.709
     edible oil drums         70         83       0.87      0.727      0.853      0.775
        drink bottles         57         71      0.854      0.657      0.735      0.605
Speed: 0.2ms preprocess, 7.3ms inference, 0.0ms loss, 0.8ms postprocess per image
Results saved to train/exp2
```

Figure 6-1- 23 Before adjusting dataset

13846.3s	606	all	1723	2356	0.832	0.739	0.81	0.719
13846.3s	607	Disposable Fast Food Box	42	50	0.765	0.6	0.67	0.56
13846.3s	608	Book Paper	24	26	0.74	0.462	0.546	0.427
13846.3s	609	Plastic Utensils	110	127	0.872	0.724	0.813	0.743
13846.3s	610	Plastic Toys	65	76	0.804	0.807	0.832	0.702
13846.3s	611	Dry Battery	67	87	0.855	0.736	0.814	0.663
13846.3s	612	Express Paper Bag	35	35	0.907	0.914	0.968	0.949
13846.3s	613	Plug Wire	149	186	0.821	0.615	0.705	0.553
13846.3s	614	Can	57	71	0.827	0.831	0.916	0.867
13846.3s	615	Peel and Pulp	144	160	0.873	0.773	0.858	0.804
13846.3s	616	Stuffed Toy	102	115	0.886	0.896	0.928	0.814
13846.3s	617	Defiled Plastic	142	184	0.805	0.576	0.67	0.593
13846.3s	618	Contaminated paper	28	41	0.713	0.727	0.758	0.636
13846.3s	619	Toilet care products	150	201	0.805	0.78	0.824	0.733
13846.3s	620	Cigarette butts	53	90	0.779	0.746	0.855	0.718
13846.3s	621	Carton box	92	105	0.913	0.697	0.796	0.725
13846.3s	622	Tea residue	45	47	0.874	0.888	0.917	0.879
13846.3s	623	Cai Bang Cai Ye	118	146	0.854	0.801	0.879	0.787
13846.3s	624	Egg Shell	53	66	0.774	0.591	0.712	0.611
13846.3s	625	Sauce Bottle	77	89	0.888	0.711	0.844	0.752
13846.3s	626	Ointment	70	77	0.897	0.87	0.927	0.856
13846.3s	627	Expired Medicine	123	145	0.866	0.803	0.874	0.778
13846.3s	628	Metal Food Cans	64	78	0.748	0.679	0.74	0.697
13846.3s	629	edible oil drums	70	83	0.886	0.843	0.879	0.795
13846.3s	630	drink bottles	57	71	0.826	0.668	0.726	0.617

Figure 6-1- 24 After adjusting dataset

6.2 Model quantization

After completing the model training, the project moves to the model deployment phase. Following the model transformation described earlier in the methodology section, the deployment of YOLO11 models to the main development board goes through the following steps: The main work completed in the project will be introduced in the following order: conversion to ONNX format, installation of the operating environment, Quantization of Post-training (PTQ), verification of input and output, and final migration and testing.

6.2.1 ONNX format conversion

First, the model file in pt format needs to be converted to .onnx format. The YOLO framework supports direct export, as shown in the figure below:

export_onnx.py

```
1  from ultralytics import YOLO
2  YOLO('best.pt').export(imgsz=640, format='onnx', simplify=True, opset=11)
```

Figure 6-2- 1 export_onnx.py

6.2.2 Environment installation

Once the transformation is complete, the next step is to install the required runtime environment for the toolchain. First it is required to install the docker desktop and then download the ubuntu image [59]:

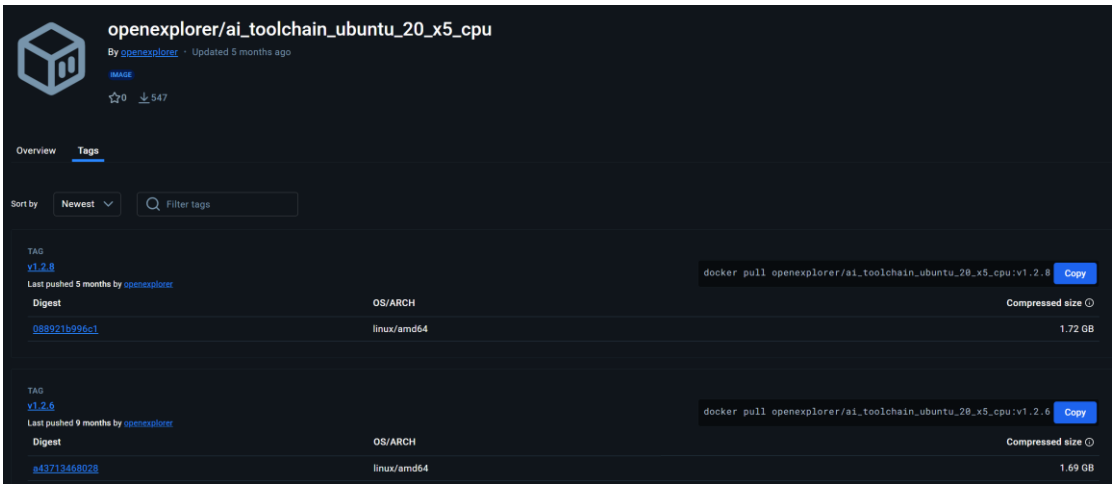

Figure 6-2- 2 Ubuntu images download

Once downloaded, the docker image can be seen in docker's images page:

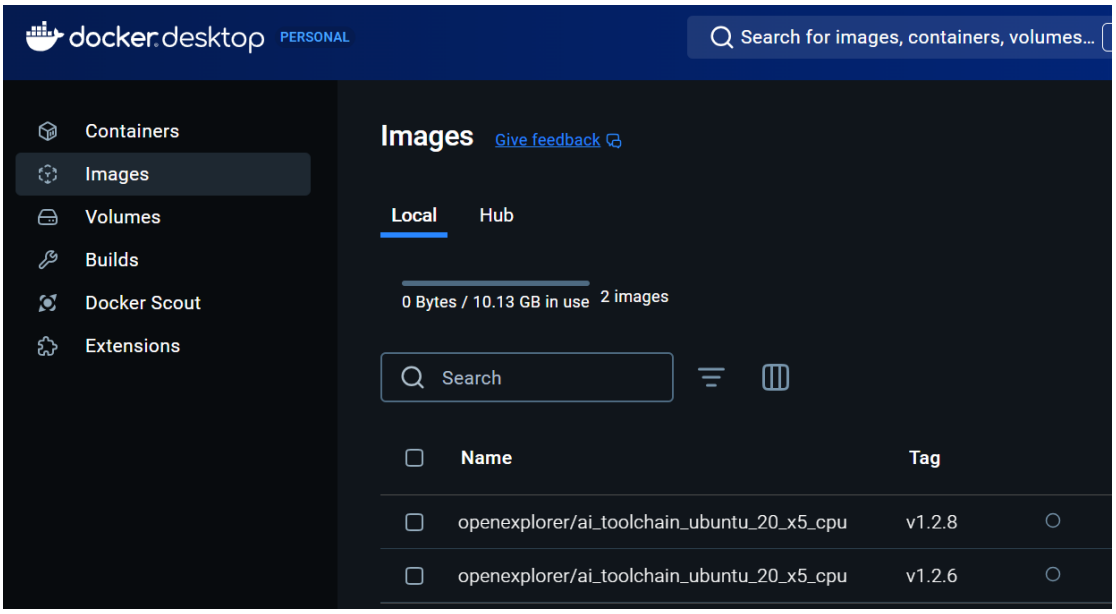

Figure 6-2- 3 docker's images page

Then it is required to download the toolchain [60]:

Figure 6-2- 4 Toolchain download

To mount the file after downloading, it is required to mount the folder where the toolchain is located and the model that needs to be converted [59]:

Figure 6-2- 5 Mount files on docker

6.2.3 Model quantization using toolchain

After the preparation of the development environment is completed, then comes the model transformation process. The architecture diagram and flow diagram are shown in Figure [51]:

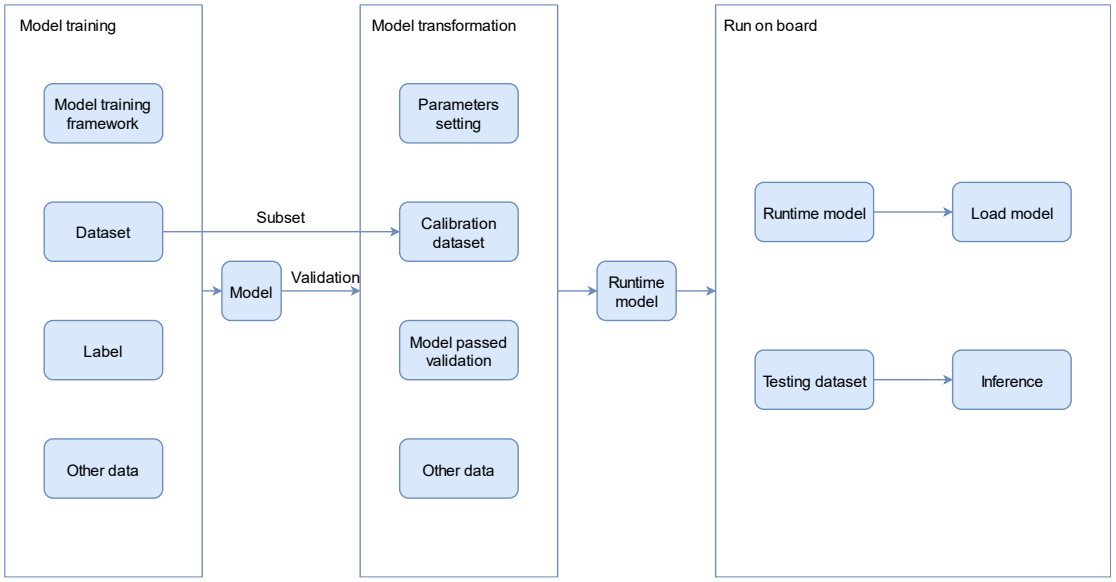

Figure 6-2- 6 Architecture diagram of model deployment

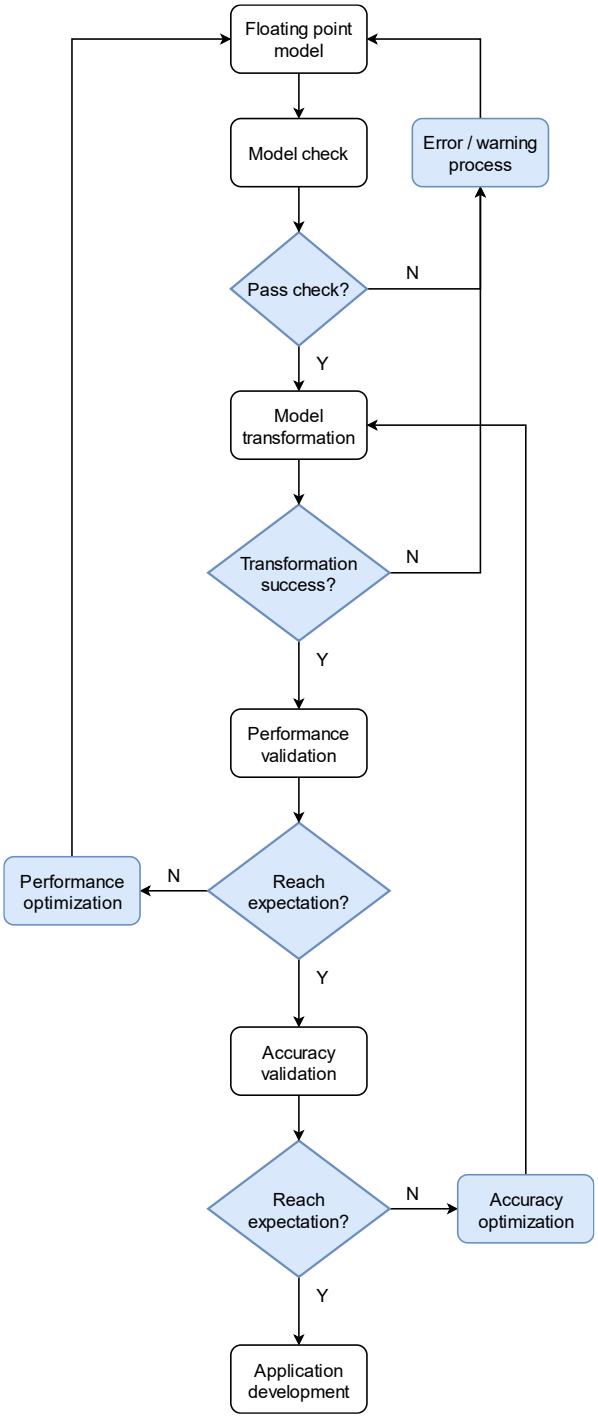

Figure 6-2- 7 Flow diagram of model transformation

The steps in the flow chart will be explained in order.

Before a model can be converted from a floating-point model to a fixed-point model, it is first checked by the *hb_mapper checker* subcommand to see if the model contains an operator that cannot be run on the Horizon hardware. If it does, the operator is not recognized at this stage. As shown in the figure, the model passes the check successfully:

```
root@ea89e0a76bd5:/data# hb_mapper checker --model-type onnx --march bayes-e --model best.onnx
/usr/local/lib/python3.10/dist-packages/paramiko/pkey.py:100: CryptographyDeprecationWarning: TripleDES has been moved t
o cryptography.hazmat.decrepit.ciphers.algorithms.TripleDES and will be removed from this module in 48.0.0.
  "cipher": algorithms.TripleDES,
/usr/local/lib/python3.10/dist-packages/paramiko/transport.py:259: CryptographyDeprecationWarning: TripleDES has been mo
ved to cryptography.hazmat.decrepit.ciphers.algorithms.TripleDES and will be removed from this module in 48.0.0.
  "class": algorithms.TripleDES,
2025-04-12 21:07:47,006 INFO log will be stored in /data/hb_mapper_checker.log
2025-04-12 21:07:47,007 INFO Start hb_mapper....
2025-04-12 21:07:47,008 INFO hbdk version 3.49.14
2025-04-12 21:07:47,009 INFO horizon_nn version 1.0.6.3
2025-04-12 21:07:47,010 INFO hb_mapper version 1.23.8
2025-04-12 21:07:47,352 INFO Model type: onnx
2025-04-12 21:07:47,356 INFO input names []
2025-04-12 21:07:47,358 INFO input shapes {}
2025-04-12 21:07:47,359 INFO Begin model checking....
2025-04-12 21:07:47,373 INFO Start to Horizon NN Model Convert.
2025-04-12 21:07:47,375 INFO Loading horizon_nn debug methods:[]
2025-04-12 21:07:47,377 INFO The specified model compilation architecture: bayes-e.
2025-04-12 21:07:47,379 INFO The specified model compilation optimization parameters: [].
2025-04-12 21:07:47,647 INFO Start to prepare the onnx model.
2025-04-12 21:07:47,649 INFO Input ONNX Model Information:
```

Figure 6-2- 8 log of model check

The Softmax operator splits the model into two BPU subgraphs:

```
/model.10/m/m.0/attn/MatMul                BPU  id(0)     HzSQuantizedMatmul          int8/int8
/model.10/m/m.0/attn/Mul                   BPU  id(0)     HzSQuantizedConv            int8/int32
/model.10/m/m.0/attn/Softmax               CPU  --        Softmax                     float/float
/model.10/m/m.0/attn/Transpose_1           BPU  id(1)     Transpose                   int8/int8
/model.10/m/m.0/attn/MatMul_1              BPU  id(1)     HzSQuantizedMatmul          int8/int8
```

Figure 6-2- 9 Operators and allocated unit

According to the architecture diagram (错误!未找到引用源。), before the formal model transformation step, the calibration data set needs to be prepared, that is, about 100 pictures are randomly selected from the original data set, and then the image data is stored according to the target size, target color (rgb or bgr, etc.), and target arrangement (CHW or HWC) according to the requirements [51]:

```python
def imequalresize(img, target_size, pad_value=127.):
    target_w, target_h = target_size
    image_h, image_w = img.shape[:2]
    img_channel = 3 if len(img.shape) > 2 else 1

    # 确定缩放尺度，确定最终目标尺寸
    scale = min(target_w * 1.0 / image_w, target_h * 1.0 / image_h)
    new_h, new_w = int(scale * image_h), int(scale * image_w)

    resize_image = cv2.resize(img, (new_w, new_h))

    # 准备待返回图像
    pad_image = np.full(shape=[target_h, target_w, img_channel], fill_value=pad_value)

    # 将图像resize_image放置在pad_image的中间
    dw, dh = (target_w - new_w) // 2, (target_h - new_h) // 2
    pad_image[dh:new_h + dh, dw:new_w + dw, :] = resize_image

    return pad_image

## 2.2 开始转换
for each_imgname in img_names:
    img_path = os.path.join(src_root, each_imgname)

    img = cv2.imread(img_path)  # BGR, HWC
    if img is None:
        print(f"Error reading image: {img_path}")
        continue
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # RGB, HWC
    img = imequalresize(img, (640, 640))
    img = np.transpose(img, (2, 0, 1))  # RGB, CHW

    # 检查图像形状
    if img.shape != (3, 640, 640):
        print(f"Unexpected image shape: {img.shape}")

    # 保存为 float32
    dst_path = os.path.join(dst_root, each_imgname + '.rgbchw')
    print("write:%s" % dst_path)
    img.astype(np.float32).tofile(dst_path)  # 保存为 float32
```

Figure 6-2- 10 cal.py (transform images to required format)

Once the calibration data is ready, we also need to prepare the .yaml configuration file required for model transformation, specifying parameters such as the format of the input model, the architecture of the development board BPU, and the format of the input image after deployment:

trans.yaml

```yaml
model_parameters:
  onnx_model: './best.onnx'
  march: "bayes-e"
  layer_out_dump: False
  working_dir: 'bin_dir/yolo11n_detect_bayese_640x640_nv12'
  output_model_file_prefix: 'yolo11n_detect_bayese_640x640_nv12'
  # YOLO11 n, s, m
  node_info: {"/model.10/m/m.0/attn/Softmax": {'ON': 'BPU','InputType': 'int16','OutputType': 'int16'}}
  # YOLO11 l, x
  # node_info: {"/model.10/m/m.0/attn/Softmax": {'ON': 'BPU','InputType': 'int16','OutputType': 'int16'},
  #             "/model.10/m/m.1/attn/Softmax": {'ON': 'BPU','InputType': 'int16','OutputType': 'int16'}}
input_parameters:
  input_name: ""
  input_type_rt: 'nv12'
  input_type_train: 'rgb'
  input_layout_train: 'NCHW'
  norm_type: 'data_scale'
  scale_value: 0.003921568627451
calibration_parameters:
  cal_data_dir: './calibration_data_rgb_f32_640'
  cal_data_type: 'float32'
  #calibration_type: 'default'
compiler_parameters:
  compile_mode: 'latency'
  debug: False
  optimize_level: 'O3'
```

Figure 6-2- 11 trans.yaml

With the configuration file ready, we can proceed with the model transformation step using *hp_mapper marketbin* command, which is partially logged as follows:

```
root@dc1e394a3cc4:/data# hb_mapper makertbin --model-type onnx --config trans.yaml
/usr/local/lib/python3.10/dist-packages/paramiko/pkey.py:100: CryptographyDeprecationWarning: TripleDES has been moved to cryptography.hazmat
.decrepit.ciphers.algorithms.TripleDES and will be removed from this module in 48.0.0.
  "cipher": algorithms.TripleDES,
/usr/local/lib/python3.10/dist-packages/paramiko/transport.py:259: CryptographyDeprecationWarning: TripleDES has been moved to cryptography.h
azmat.decrepit.ciphers.algorithms.TripleDES and will be removed from this module in 48.0.0.
  "class": algorithms.TripleDES,
2025-04-12 21:48:39,274 INFO log will be stored in /data/hb_mapper_makertbin.log
2025-04-12 21:48:39,277 INFO Start hb_mapper....
2025-04-12 21:48:39,279 INFO hbdk version 3.49.14
2025-04-12 21:48:39,281 INFO horizon_nn version 1.0.6.3
2025-04-12 21:48:39,283 INFO hb_mapper version 1.23.8
2025-04-12 21:48:39,285 INFO Start Model Convert....
2025-04-12 21:48:39,299 INFO Using onnx model file: /data/best.onnx
2025-04-12 21:48:39,595 INFO Model has 1 inputs according to model file
2025-04-12 21:48:39,605 INFO Model name not given in yaml_file, using model name from model file: ['images']
2025-04-12 21:48:39,606 INFO Model input shape not given in yaml_file, using shape from model file: [[1, 3, 640, 640]]
2025-04-12 21:48:39,613 WARNING The calibration dir name suffix is not the same as the value float32 of the parameter cal_data_type, the para
meter setting will prevail
2025-04-12 21:48:39,617 INFO custom_op does not exist, skipped
2025-04-12 21:48:39,621 WARNING Input node images's input_source not set, it will be set to ddr by default
2025-04-12 21:48:39,640 INFO ****************************************
2025-04-12 21:48:39,642 INFO First calibration picture name: DM_20250322001654_001__images.jpg.rgbchw
2025-04-12 21:48:39,643 INFO First calibration picture md5:
3a2b7351a23a14b6e6c56bdce587993f  /data/calibration_data_rgb_f32_640/DM_20250322001654_001__images.jpg.rgbchw
2025-04-12 21:48:39,737 INFO ****************************************
2025-04-12 21:48:44,044 INFO Start to Horizon NN Model Convert.
2025-04-12 21:48:44,046 INFO Loading horizon_nn debug methods:[]
2025-04-12 21:48:44,048 INFO Parsing the calibration parameter
2025-04-12 21:48:44,049 INFO There are 1 nodes designated to run on the bpu: ['/model.10/m/m.0/attn/Softmax'].
2025-04-12 21:48:44,051 INFO The specified model compilation architecture: bayes-e.
2025-04-12 21:48:44,052 INFO The specified model compilation optimization parameters: [].
2025-04-12 21:48:44,379 INFO Start to prepare the onnx model.
2025-04-12 21:48:44,381 INFO Input ONNX Model Information:
ONNX IR version:          6
Opset version:            ['ai.onnx v11', 'horizon v1']
Producer:                 pytorch v2.0.1
Domain:                   None
```

Figure 6-2- 12 log of model transformation (1)

```
2025-04-12 22:01:37,301 INFO End to Horizon NN Model Convert.
2025-04-12 22:01:37,874 INFO start convert to *.bin file....
2025-04-12 22:01:37,939 INFO ONNX model output num : 6
2025-04-12 22:01:37,949 INFO ############# model deps info #############
2025-04-12 22:01:37,951 INFO hb_mapper version   : 1.23.8
2025-04-12 22:01:37,953 INFO hbdk version        : 3.49.14
2025-04-12 22:01:37,954 INFO hbdk runtime version: 3.15.54.0
2025-04-12 22:01:37,955 INFO horizon_nn version  : 1.0.6.3
2025-04-12 22:01:37,957 INFO ############# model_parameters info #############
2025-04-12 22:01:37,958 INFO onnx_model          : /data/best.onnx
2025-04-12 22:01:37,960 INFO BPU march           : bayes-e
2025-04-12 22:01:37,962 INFO layer_out_dump      : False
2025-04-12 22:01:37,963 INFO log_level           : DEBUG
2025-04-12 22:01:37,964 INFO working dir         : /data/bin_dir/yolo11n_detect_bayese_640x640_nv12
2025-04-12 22:01:37,965 INFO output_model_file_prefix: yolo11n_detect_bayese_640x640_nv12
2025-04-12 22:01:37,967 INFO node info  : {'/model.10/m/m.0/attn/Softmax': {'ON': 'BPU', 'InputType': 'int16', 'OutputType': 'int16'}}
2025-04-12 22:01:37,968 INFO ############# input_parameters info #############
2025-04-12 22:01:37,970 INFO ----------------------------------------
2025-04-12 22:01:37,971 INFO ---------input info : images ---------
2025-04-12 22:01:37,973 INFO input_name          : images
2025-04-12 22:01:37,974 INFO input_type_rt       : rgb
2025-04-12 22:01:37,976 INFO input_space&range   : regular
2025-04-12 22:01:37,978 INFO input_layout_rt     : NCHW
2025-04-12 22:01:37,978 INFO input_type_train    : rgb
2025-04-12 22:01:37,980 INFO input_layout_train  : NCHW
2025-04-12 22:01:37,982 INFO norm_type           : data_scale
2025-04-12 22:01:37,984 INFO input_shape         : 1x3x640x640
2025-04-12 22:01:37,986 INFO scale_value         : 0.003921568627451,
2025-04-12 22:01:37,987 INFO cal_data_dir        : /data/calibration_data_rgb_f32_640
2025-04-12 22:01:37,989 INFO cal_data_type       : float32
2025-04-12 22:01:37,990 INFO ---------input info : images end -------
2025-04-12 22:01:37,991 INFO ----------------------------------------
2025-04-12 22:01:37,993 INFO ############# calibration_parameters info #############
2025-04-12 22:01:37,994 INFO preprocess_on       : False
2025-04-12 22:01:37,995 INFO calibration_type:   : default
2025-04-12 22:01:37,997 INFO per_channel         : False
2025-04-12 22:01:37,999 INFO ############# compiler_parameters info #############
2025-04-12 22:01:38,000 INFO debug               : False
2025-04-12 22:01:38,002 INFO compile_mode        : latency
```

Figure 6-2- 13 log of model transformation (2)

Once the transformation is complete, the transformed model needs to be analyzed using the *hb_perf* subcommand:

```
root@4c9f3a1382f5:/data/bin_dir/yolo11n_detect_bayese_640x640_nv12# hb_perf yolo11n_detect_bayese_640x640_nv12.bin
2025-04-12 22:26:57,615 INFO log will be stored in /data/bin_dir/yolo11n_detect_bayese_640x640_nv12/hb_perf.log
2025-04-12 22:26:57,665 INFO Start hb_perf....
2025-04-12 22:26:57,667 INFO hb_perf version 1.23.8
2025-04-12 22:26:57,764 INFO **** yolo11n_detect_bayese_640x640_nv12 perf *****
2025-04-12 22:26:57,912 INFO When the bin model input type is nv12, the BPU will convert nv12 to yuv444 internally to do
 the operation. In the drawing process, we use a BPU node to replace this input type conversion process to ensure the co
rrectness of the logical relationships expressed in the diagram, but in fact, this BPU node doesn't exist in the bin mod
el.. We name this BPU node involved in the drawing process NV12TOYUV444 with details ['1x640x640x3 / 2, NV12, UINT8'].
2025-04-12 22:26:58,166 INFO draw graph png finished.
2025-04-12 22:26:58,254 INFO get bpu model succeeded.
2025-04-12 22:26:58,629 INFO FPS=68.75, latency = 14545.8 us, DDR = 43627056 bytes   (see ./hb_perf_result/yolo11n_detec
t_bayese_640x640_nv12/torch_jit_subgraph_0.html)
2025-04-12 22:26:58,632 WARNING Load/Store with compression is used in your model. Actual latency depends on the data it
 processed.
2025-04-12 22:26:58,655 INFO get perf info succeeded.
2025-04-12 22:26:58,657 WARNING bpu model don't have per-layer perf info.
2025-04-12 22:26:58,658 WARNING if you need per-layer perf info please enable[compiler_parameters.debug:True] when use m
akertbin.
2025-04-12 22:26:58,663 INFO generating html...
2025-04-12 22:26:58,672 INFO html generation finished.
2025-04-12 22:26:58,673 INFO file stored at : /data/bin_dir/yolo11n_detect_bayese_640x640_nv12/hb_perf_result/yolo11n_de
tect_bayese_640x640_nv12/yolo11n_detect_bayese_640x640_nv12.html
```

Figure 6-2- 14 log of model performance validation

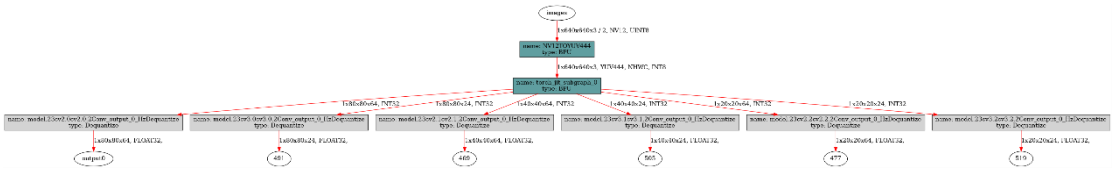

Figure 6-2- 15 I/O information of model

As can be seen from the figure above, the model has a total of 6 output headers, and each output header will have a CPU operator for inverse quantization calculation. The input has a YUV420 to YUV444 operator, which is the nv12 conversion operator automatically implemented by the compile, and NCHW-YUV444 is converted to NCHW-RGB by a convolution operator, which corresponds to the input of the .onnx model [50].

Model Perf Report

Model Performance Summary

Model Name : yolo11n_detect_bayese_640x640_nv12.bin

BPU Model Latency(ms) : 14.55

Total DDR (loaded + stored) bytes per frame(MB per frame) : 41.61

Loaded Bytes per Frame : 26739248

Stored Bytes per Frame : 16887808

Note : There are CPU nodes in the model, this part of the time consumption cannot be estimated, so only the time consumption on the BPU is shown

Details

Model Subgraph Item

Model Subgraph Name : torch_jit_subgraph_0

Model Subgraph Calculation Load (OPpf) : 21481830400

Model Subgraph DDR Occupation(Mbpf) : 41.606

Model Subgraph Latency(ms) : 14.55

Model Subgraph Info Files : torch_jit_subgraph_0 Detail

Figure 6-2- 16 I/O information of model

The above figure shows the performance of the model. It can seen that the inference frame rate of the model reaches 68.75fps, and the BPU inference takes 14.55ms.

6.3 WiFi-based video streaming

After the transformation of the model, it is necessary to obtain the video from the camera, perform real-time reasoning, draw the object label box and category information on the screen, and finally deploy the screen to the local network port to view the real-time recognition effect on the development computer. The following will be elaborated in combination with the code:

6.3.1 Inference and video as input

```python
def capture_and_detect(model, camera_id, ranging=None):
    """从摄像头捕获视频并进行实时目标检测"""
    global global_frame

    # 打开摄像头
    cap = cv2.VideoCapture(camera_id)
    if not cap.isOpened():
        logger.error(f"无法打开摄像头 {camera_id}")
        return

    # 设置摄像头分辨率
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)

    # 帧率统计变量
    frame_count = 0
    fps = 0
    fps_timer = time()

    # 初始化物体跟踪器
    tracker = ObjectTracker(stability_threshold=10, distance_threshold=5.0, position_threshold=20)

    logger.info("成功打开摄像头，开始实时检测")

    try:
        while True:
            # 读取一帧
            ret, frame = cap.read()
            if not ret:
                logger.error("无法获取摄像头画面")
                break

            # 计时器开始
            process_start_time = time()

            # 准备输入数据
            input_tensor = model.bgr2nv12(frame)

            # 推理
            outputs = model.c2numpy(model.forward(input_tensor))
```

Figure 6-3- 1 wifi.py - *capture_and_detect* (1)

```python
try:
    ids, scores, bboxes = model.postProcess(outputs)
```

Figure 6-3- 2 wifi.py - *capture_and_detect* (2)

```
# 计算并更新FPS
frame_count += 1
current_time = time()

# 每秒更新一次FPS值
if current_time - fps_timer >= 1.0:
    fps = frame_count / (current_time - fps_timer)
    frame_count = 0
    fps_timer = current_time

# 计算处理单帧的时间
process_time = time() - process_start_time

# 在画面上显示性能信息
cv2.putText(frame, f"FPS: {fps:.1f}", (10, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
cv2.putText(frame, f"Process Time: {process_time*1000:.1f} ms", (10, 60),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

# 更新全局帧
_, buffer = cv2.imencode('.jpg', frame, [cv2.IMWRITE_JPEG_QUALITY, 100])
global_frame = buffer.tobytes()
```

Figure 6-3- 3 wifi.py - *capture_and_detect* (3)

The *Capture_and_detect* function is mainly responsible for setting the camera parameters, preparing the input image, calling the model inference, and printing the computed performance parameters on the screen.

```
解释代码 | 注释代码 | 生成单测 | ×
def forward(self, input_tensor: np.array) -> list[dnn.pyDNNTensor]:
    begin_time = time()
    quantize_outputs = self.quantize_model[0].forward(input_tensor)
    return quantize_outputs

解释代码 | 注释代码 | 生成单测 | ×
def c2numpy(self, outputs) -> list[np.array]:
    begin_time = time()
    outputs = [dnnTensor.buffer for dnnTensor in outputs]
    return outputs
```

Figure 6-3- 4 wifi.py - *forward* and *c2numpy*

The *forward* method passes the preprocessed input tensor to the quantization model to obtain the original output tensor. The *c2numpy* method converts the specialized tensor format of the model output to a standard *NumPy* array for subsequent processing.

```
class YOLO11_Detect(BaseModel):
    def postProcess(self, outputs: list[np.ndarray]) -> tuple[list]:
        begin_time = time()
        try:
            # reshape
            s_bboxes = outputs[0].reshape(-1, 64)
            m_bboxes = outputs[1].reshape(-1, 64)
            l_bboxes = outputs[2].reshape(-1, 64)
            s_clses = outputs[3].reshape(-1, 24)
            m_clses = outputs[4].reshape(-1, 24)
            l_clses = outputs[5].reshape(-1, 24)
```

Figure 6-3- 5 wifi.py - *postprocess* (1)

Transforms the raw output into a tractable shape.

```
# classify: 利用numpy向量化操作完成阈值筛选(优化版 2.0)
s_max_scores = np.max(s_clses, axis=1)
s_valid_indices = np.flatnonzero(s_max_scores >= self.conf_inverse)  # 得到大于阈值分数的索引，此时为小数字
s_ids = np.argmax(s_clses[s_valid_indices, : ], axis=1)
s_scores = s_max_scores[s_valid_indices]
```

Figure 6-3- 6 wifi.py - *postprocess* (2)

A confidence threshold is used to filter out low-quality predictions.

```
# 3个Bounding Box分支：dist2bbox (ltrb2xyxy)
s_ltrb_indices = np.sum(softmax(s_bboxes_float32.reshape(-1, 4, 16), axis=2) * self.weights_static, axis=2)
s_anchor_indices = self.s_anchor[s_valid_indices, :]
s_x1y1 = s_anchor_indices - s_ltrb_indices[:, 0:2]
s_x2y2 = s_anchor_indices + s_ltrb_indices[:, 2:4]
s_dbboxes = np.hstack([s_x1y1, s_x2y2])*8
```

Figure 6-3- 7 wifi.py - *postprocess* (3)

Decode the model output into the actual bounding box coordinates and categories.

```
# 大中小特征层阈值筛选结果拼接
dbboxes = np.concatenate((s_dbboxes, m_dbboxes, l_dbboxes), axis=0)
scores = np.concatenate((s_scores, m_scores, l_scores), axis=0)
ids = np.concatenate((s_ids, m_ids, l_ids), axis=0)
```

Figure 6-3- 8 wifi.py - *postprocess* (4)

Integrate detections from three feature layers (bounding box, confidence, class number).

```
# nms
indices = cv2.dnn.NMSBoxes(dbboxes, scores, self.conf, self.iou)
```

Figure 6-3- 9 wifi.py - *postprocess* (5)

Non-maximum suppression (NMS) was used to remove overlapping detections.

```
# 还原到原始的img尺度
bboxes = dbboxes[indices] * np.array([self.x_scale, self.y_scale, self.x_scale, self.y_scale])
bboxes = bboxes.astype(np.int32)
```

Figure 6-3- 10 wifi.py - *postprocess* (6)

The detection results are mapped back to the original image dimensions.

6.3.2 *Deploy on local network port*

```
def start_server(port):
    """启动Flask Web服务器"""
    app.run(host='0.0.0.0', port=port, threaded=True)

@app.route('/')
解释代码 | 注释代码 | 生成单测 | ×
def index():
    """网页主页"""
    html_template = """
    <!DOCTYPE html>
    <html>
    <head>
        <title>RDK Realtime Object Detection Streaming</title>
        <style>
            body {
                font-family: Arial, sans-serif;
                margin: 0;
                padding: 20px;
                text-align: center;
                background-color: #f5f5f5;
            }
            h1 {
                color: #333;
            }
            .video-container {
                margin: 20px auto;
                max-width: 800px;
            }
            img {
                max-width: 100%;
                border: 2px solid #333;
                border-radius: 5px;
            }
        </style>
    </head>
    <body>
        <h1>RDK Realtime Object Detection Streaming</h1>
        <div class="video-container">
            <img src="{{ url_for('video_feed') }}" alt="视频流">
        </div>
```

Figure 6-3- 11 wifi.py - Webpage layout

When the browser loads the <img> tag, it makes a request to the server and continuously receives and displays images from the MJPEG stream.

```python
@app.route('/video_feed')
解释代码 | 注释代码 | 生成单测 | ×
def video_feed():
    """提供视频流"""
    return Response(generate_frames(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

解释代码 | 注释代码 | 生成单测 | ×
def generate_frames():
    """生成实时视频帧"""
    global global_frame
    while True:
        if global_frame is not None:
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + global_frame + b'\r\n')
        else:
            # 如果没有可用帧，生成一个空白帧
            blank_frame = np.ones((480, 640, 3), dtype=np.uint8) * 255
            cv2.putText(blank_frame, "等待摄像头画面...", (150, 240),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2)
            _, buffer = cv2.imencode('.jpg', blank_frame)
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + buffer.tobytes() + b'\r\n')
```

Figure 6-3- 12 wifi.py - Frame update

Use the *multipart/x-mixed-replace* content type, which allows the server to continuously replace the same resource Streaming via Python *yield*, without waiting for the entire video to be ready.

```python
def main():
    # 获取本机IP地址
    ip_address = get_ip_address()
    logger.info(f"本机IP地址: {ip_address}")
    logger.info(f"打开浏览器访问: http://{ip_address}:{opt.port} 查看实时检测结果")

    # 实例化模型
    model = YOLO11_Detect(opt.model_path, opt.conf_thres, opt.iou_thres)

    # 启动Web服务器线程
    server_thread = threading.Thread(target=start_server, args=(opt.port,))
    server_thread.daemon = True
    server_thread.start()
```

Figure 6-3- 13 Webpage deployment

Deploy the web page to a local network port.

6.3.3 File migration and testing

After the transformation of the model and the development of the wifi graph transfer program, the files need to be transferred to the main development board. The development board can be connected to the development computer through the type-c

line, so that the RDK Studio application on the development computer can be directly used to manage the files of the development board and run from the command line:

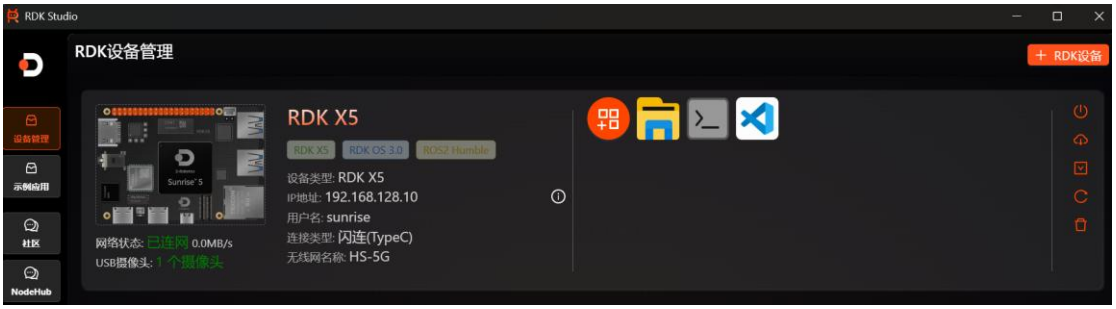

Figure 6-3- 14 RDK Studio

After entering the file manager, you can directly drag the file to copy the model-related files on the development computer to the development board, which greatly saves the time of porting files:

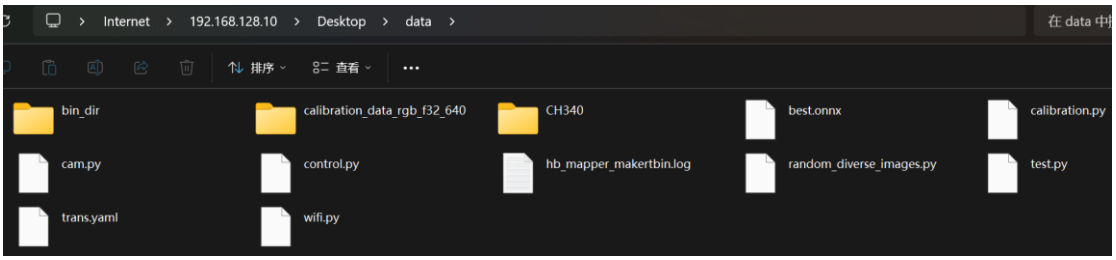

Figure 6-3- 15 Manage files on board by file manager on PC

In addition, the command line can be directly called through the development computer, which is convenient to complete the test:

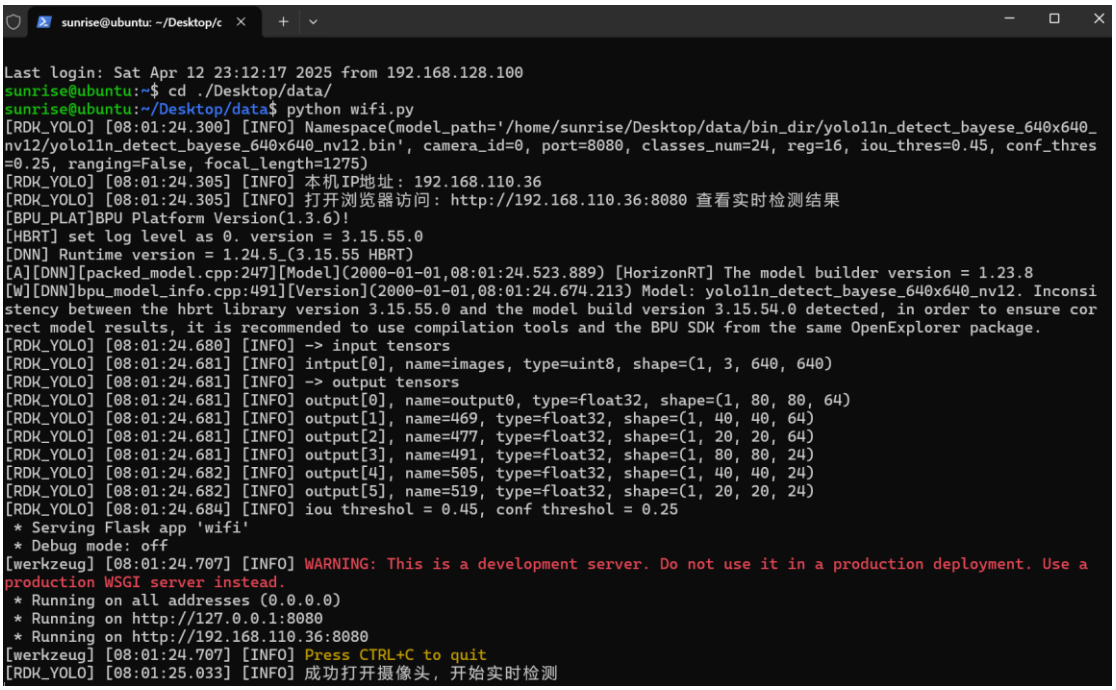

Figure 6-3- 16 Testing log

The actual running effect is shown as follows:

Figure 6-3- 17 Testing result

6.4 Monocular distance measurement

According to the previous methodology introduction, the following steps are required to achieve monocular distance measurement: camera calibration, tilt Angle measurement, measuring the height of the test object, and finally writing a program to solve the formula.

6.4.1 Camera calibration

Camera calibration can be done with the Camera Calibrator application in Matlab. First, upload pictures of the calibration board taken at different angles, and the app will automatically identify the corners of the checkerboard. After the initial calibration, samples with large reprojection errors are deleted and calibrated again. Finally, the parameters of the camera are derived:

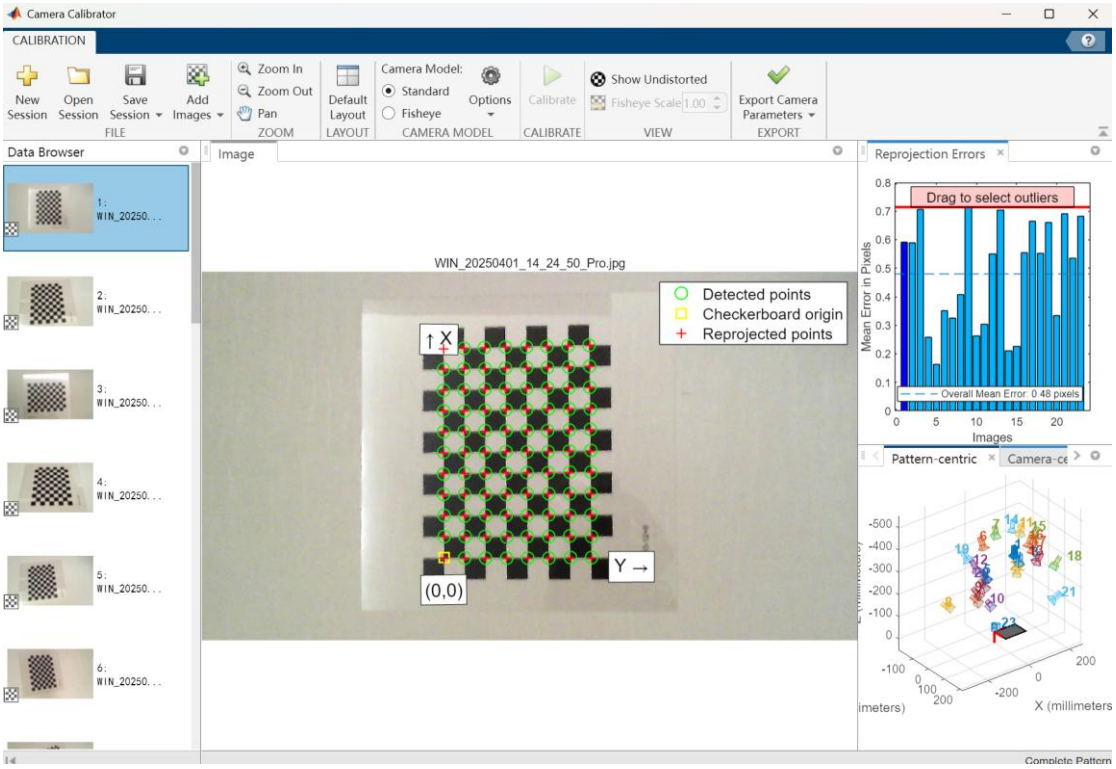

Figure 6-4- 1 Camera Calibrator in Matlab

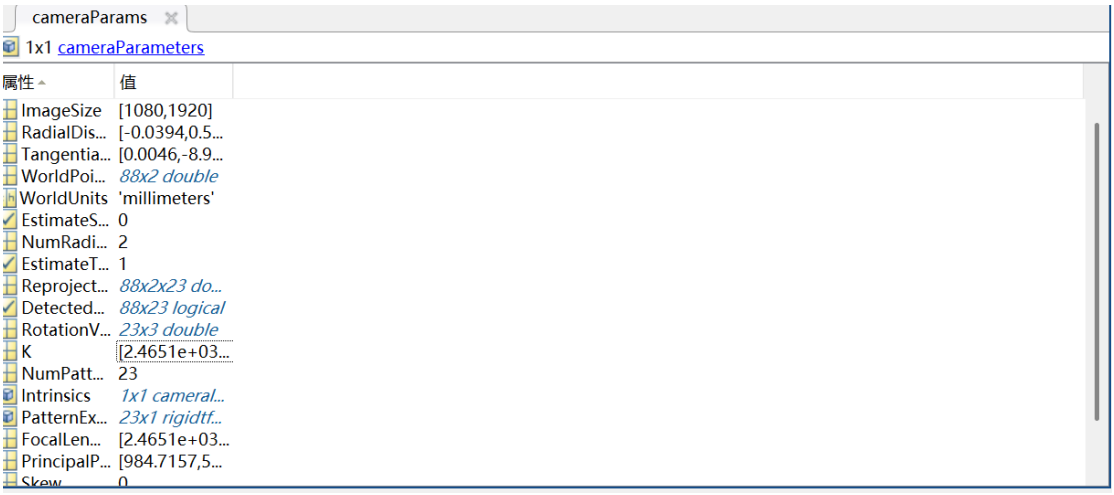

Figure 6-4- 2 Camera parameters derived

After getting the parameters, fill them in according to the previous camera internal parameter matrix and error term, and test them:

calibrate.py > undistort_video

```python
#IntrinsicMatrix
fx,cx,fy,cy=2465.128986762485,984.7157440488494,2465.442899090040,540.7101182076190
#TangentialDistortion
p1,p2=0.004625970045023,-0.0008902630420674212
#RadialDistortion
k1,k2,k3=-0.039376100278451,0.563886970534354,0

解释代码 | 注释代码 | 生成单测 | ×
def undistort_video(camera_matrix, dist_coeffs, video_source=1):
    # 打开视频流
    cap = cv2.VideoCapture(video_source)

    # 读取第一帧以获取其尺寸
    ret, frame = cap.read()
    if not ret:
        print("无法打开视频流或文件")
        return

    h, w = frame.shape[:2]

    # 创建一个窗口用于显示校正前的视频
    cv2.namedWindow('Original Video', cv2.WINDOW_AUTOSIZE)

    # 创建一个窗口用于显示校正后的视频
    cv2.namedWindow('Undistorted Video', cv2.WINDOW_AUTOSIZE)

    while True:
        # 读取视频帧
        ret, frame = cap.read()
        if not ret:
            break

        # 校正图像
        undistorted_frame = cv2.undistort(frame, camera_matrix, dist_coeffs, None, camera_matrix)

        # 显示校正前和校正后的视频
        cv2.imshow('Original Video', frame)
        cv2.imshow('Undistorted Video', undistorted_frame)

        # 按'q'键退出
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

Figure 6-4- 3 calibrate.py

Here is a comparison of the original and undistorted images:

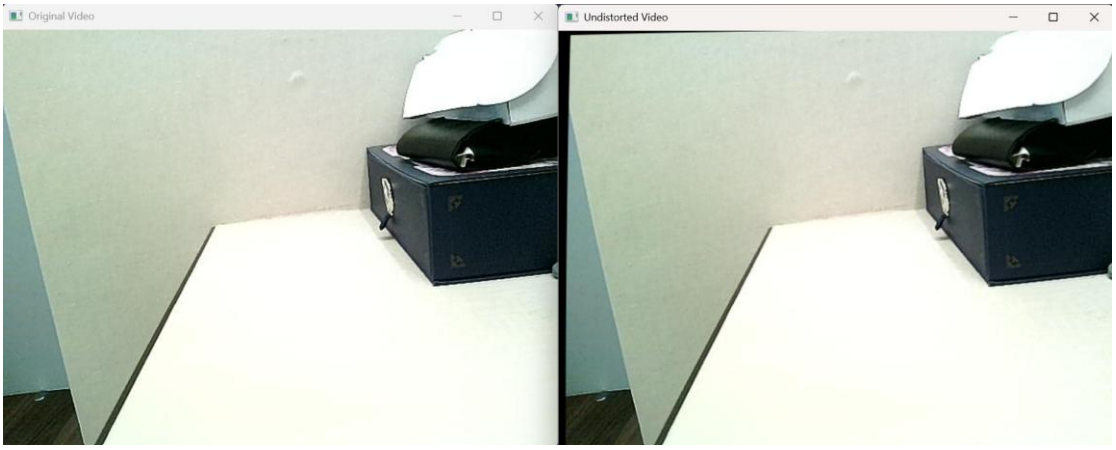

Figure 6-4- 4 Undistorted result

6.4.2 Tilt angle measurement

Next, it is necessary to measure the tilt Angle. First, it is necessary to place the calibration plate on the horizontal table and measure the distance from the center point to the camera (take pictures from different distances on the same axis):

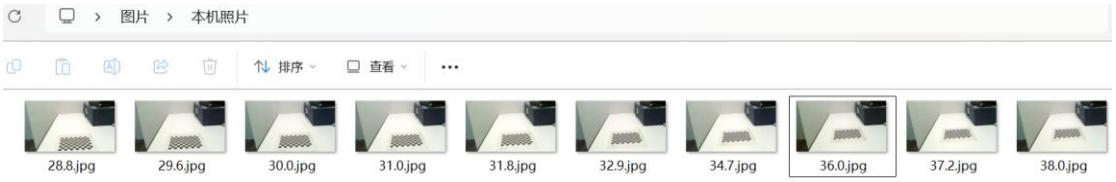

Figure 6-4- 5 Different distances

The following is the procedure for measuring the tilt Angle. After reading the image, the checkerboard corners are detected:

```python
angle.py > ...
class TiltCalibrator:
    def detect_board_center(self, img_path, pattern_size=(11,8)):
        # 输出图像信息
        print(f"处理图像: {img_path}, 尺寸: {img.shape}")

        # 转换为灰度图
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # 提高对比度（可选）
        # gray = cv2.equalizeHist(gray)

        # 创建调试目录
        debug_dir = "debug_images"
        os.makedirs(debug_dir, exist_ok=True)

        # 保存灰度图用于调试
        cv2.imwrite(f"{debug_dir}/gray_{os.path.basename(img_path)}", gray)

        # 尝试使用多种标志组合检测棋盘格角点
        flags = cv2.CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_NORMALIZE_IMAGE + cv2.CALIB_CB_FAST_CHECK
        ret, corners = cv2.findChessboardCorners(gray, pattern_size, flags)

        # 如果检测失败，尝试其他尺寸
        if not ret:
            print(f"在标准尺寸下检测失败: {pattern_size}")
            # 尝试其他可能的棋盘格尺寸
            alternative_sizes = [(12, 9), (10, 7), (9, 6), (8, 5), (7, 6), (6, 9)]

            for alt_size in alternative_sizes:
                print(f"尝试替代尺寸: {alt_size}")
                ret, corners = cv2.findChessboardCorners(gray, alt_size, flags)
                if ret:
                    pattern_size = alt_size
                    print(f"成功检测到棋盘格，使用尺寸: {pattern_size}")
                    break
```

Figure 6-4- 6 angle.py (1)

After that, detect the center of the checkerboard and calculate the vertical offset:

```python
angle.py > ...
class TiltCalibrator:
    def detect_board_center(self, img_path, pattern_size=(11,8)):
        # 绘制检测到的角点并保存（用于调试）
        img_with_corners = img.copy()
        cv2.drawChessboardCorners(img_with_corners, pattern_size, corners, ret)
        cv2.imwrite(f"{debug_dir}/corners_{os.path.basename(img_path)}", img_with_corners)

        # 亚像素精细化
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
        corners = cv2.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)

        # 计算中心点（图像坐标系原点在左上角）
        center_pixel = np.mean(corners, axis=0)[0]

        # 转换为以图像中心为原点 (u, v)
        h, w = img.shape[:2]
        u = center_pixel[0] - w/2
        v = h/2 - center_pixel[1]  # 注意Y轴方向

        # 在图像上标记中心点并保存
        cv2.circle(img_with_corners, (int(center_pixel[0]), int(center_pixel[1])),
                   5, (0, 0, 255), -1)
        cv2.imwrite(f"{debug_dir}/center_{os.path.basename(img_path)}", img_with_corners)

        print(f"成功检测棋盘格中心点: u={u:.1f}, v={v:.1f}")
        return u, v
```

Figure 6-4- 7 angle.py (2)

Put into the formula to calculate the tilt Angle:

```python
angle.py > ...
class TiltCalibrator:
    def theta_residuals(self, theta, D_ref_list, v_pixel_list):
        """最小二乘残差计算"""
        residuals = []
        for D_ref, v in zip(D_ref_list, v_pixel_list):
            alpha = np.arctan(v / self.f)
            H_eff = self.H + self.h  # 计算有效垂直距离
            pred_D = H_eff / np.tan(theta[0] + alpha)
            residuals.append(pred_D - D_ref)
        return np.array(residuals)

    解释代码 | 注释代码 | 生成单测 | ×
    def calibrate(self, img_paths, D_ref_list):
        """主标定函数"""
        # 1. 检测所有图像的标定板中心坐标
        v_pixel_list = []
        for path in img_paths:
            _, v = self.detect_board_center(path)
            v_pixel_list.append(v)

        # 2. 非线性最小二乘优化
        res = least_squares(
            self.theta_residuals,
            x0=np.radians(30),  # 初始猜测角度（弧度）
            args=(D_ref_list, v_pixel_list),
            bounds=(0, np.pi/2) # 角度在0-90度之间
        )

        self.theta = res.x[0]
        return np.degrees(self.theta)
```

Figure 6-4- 8 angle.py (3)

The result is as follows:

```
[Running] set PYTHONIOENCODING=utf8 && D:\Anaconda\envs\pytorch\python.exe -u "d:\study\rdk_docker\data\angle.py"
处理图像: C:/Users/Feng/Pictures/Camera Roll/28.8.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=80.1, v=-238.9
处理图像: C:/Users/Feng/Pictures/Camera Roll/29.6.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=73.0, v=-222.6
处理图像: C:/Users/Feng/Pictures/Camera Roll/30.0.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=68.1, v=-207.0
处理图像: C:/Users/Feng/Pictures/Camera Roll/31.0.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=81.5, v=-197.9
处理图像: C:/Users/Feng/Pictures/Camera Roll/31.8.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=82.2, v=-184.1
处理图像: C:/Users/Feng/Pictures/Camera Roll/32.9.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=76.4, v=-165.4
处理图像: C:/Users/Feng/Pictures/Camera Roll/34.7.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=76.7, v=-142.4
处理图像: C:/Users/Feng/Pictures/Camera Roll/36.0.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=86.6, v=-106.5
处理图像: C:/Users/Feng/Pictures/Camera Roll/37.2.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=84.2, v=-102.4
处理图像: C:/Users/Feng/Pictures/Camera Roll/38.0.jpg, 尺寸: (720, 1280, 3)
成功检测棋盘格中心点: u=78.5, v=-94.7
标定结果: 倾斜角θ = 31.41°
```

Figure 6-4- 9 Result of tilt angle calculation

6.4.3 Object height measurement

After calculating the tilt Angle of the camera, it is necessary to measure the height of the test object to ensure the accuracy of the distance measurement. The following is part of the measurement results. If there is no number at the end of each line, the height is ignored or not added to the test:

```
一次性快餐盒 -- 其他 汤碗 5.5
书籍纸张 -- 可回收
塑料器皿 -- 可回收
塑料玩具 -- 可回收 扭蛋 5.5
干电池 -- 有害 4.7
快递纸袋 -- 可回收
插头电线 -- 可回收
易拉罐 -- 可回收 9.0
果皮果肉 --厨余 小苹果 5.6
毛绒玩具 -- 可回收 10.0
污损塑料 -- 其他 8.2
污损用纸 -- 其他 5.0
洗护用品 -- 可回收 无比滴 9.7
烟蒂 -- 其他
纸盒纸箱 -- 可回收 10.0
茶叶渣 -- 厨余
菜帮菜叶 --厨余
蛋壳 --厨余 4.0
调料瓶 --可回收 13.1
软膏 -- 有害
过期药物 --有害 止痛散 5.5
金属食品罐 -- 可回收 午餐肉 10.0
食用油桶 --可回收
饮料瓶 -- 可回收 16.0
```

Figure 6-4- 10 Height of testing object

Load the height of objects:

```python
def load_object_heights():
    """从label.txt文件中加载物体高度信息"""
    object_heights = [0] * len(coco_names)  # 默认高度为0
    try:
        with open('label.txt', 'r', encoding='utf-8') as f:
            for i, line in enumerate(f):
                line = line.strip()
                if not line:
                    continue

                # 尝试提取行末尾的数字（物体高度）
                parts = line.split()
                if len(parts) >= 2:
                    try:
                        # 最后一个部分可能是高度数字
                        height = float(parts[-1])
                        if i < len(object_heights):
                            object_heights[i] = height
                    except ValueError:
                        # 如果不是数字，则忽略
                        pass
    except Exception as e:
        logger.error(f"读取label.txt文件时出错: {e}")

    logger.info(f"已加载物体高度信息: {object_heights}")
    return object_heights
```

Figure 6-4- 11 wifi.py – *load_object_heights*

6.4.4 Ranging and testing

Next, the core code of the ranging function will be explained. First, the coordinates of the bounding box are extracted and the vertical center of the object is calculated:

```python
def draw_detection(img: np.array,
                   bbox: tuple[int, int, int, int],
                   score: float,
                   class_id: int,
                   ranging=None,
                   object_height=0) -> tuple:
    """
    绘制检测框并计算距离（如果提供了测距对象）

    Parameters:
        img (np.array): 输入图像
        bbox (tuple[int, int, int, int]): 边界框坐标 (x1, y1, x2, y2)
        score (float): 检测分数
        class_id (int): 类别ID
        ranging (MonocularRanging, optional): 单目测距对象

    Returns:
        tuple: (center_x, center_y, distance) - 检测框中心坐标和距离
    """
    x1, y1, x2, y2 = bbox
    # 检查边界框坐标是否超出图像范围
    h, w = img.shape[:2]
    x1, y1 = max(0, x1), max(0, y1)
    x2, y2 = min(w-1, x2), min(h-1, y2)

    # 计算中心点坐标
    center_x = (x1 + x2) / 2
    center_y = (y1 + y2) / 2

    # 绘制中心点
    cv2.circle(img, (int(center_x), int(center_y)), 3, (0, 255, 255), -1)

    # 测距计算
    distance = None
    if ranging is not None:
        distance = ranging.calculate_distance(center_y, object_height)
```

Figure 6-4- 12 wifi.py – *draw_detection*

Estimated distance is then calculated according to the formula previously presented in the methodology section:

```python
class MonocularRanging:
    解释代码 | 注释代码 | 生成单测 | ×
    def __init__(self, H, theta, h=0, f=800, img_size=(640,480)):
        self.H = H
        self.theta = theta  # 倾斜角（弧度）
        self.h = h
        self.f = f
        self.img_center_y = img_size[1] / 2  # 仅需垂直中心

    解释代码 | 注释代码 | 生成单测 | ×
    def pixel_to_alpha(self, y_pixel):
        """根据垂直像素坐标计算角度偏移"""
        v = y_pixel - self.img_center_y  # 计算垂直偏移
        alpha = np.arctan(v / self.f)
        return alpha

    解释代码 | 注释代码 | 生成单测 | ×
    def calculate_distance(self, y_pixel, object_height=0):
        alpha = self.pixel_to_alpha(y_pixel)
        # 根据物体高度调整有效高度
        H_eff = self.H + self.h - object_height/2
        D = H_eff / np.tan(self.theta + alpha) - 7.0 #减去相机到基座前端的距离
        return D
```

Figure 6-4- 13 wifi.py – *MonocularRanging*

Some measured results are as follows:

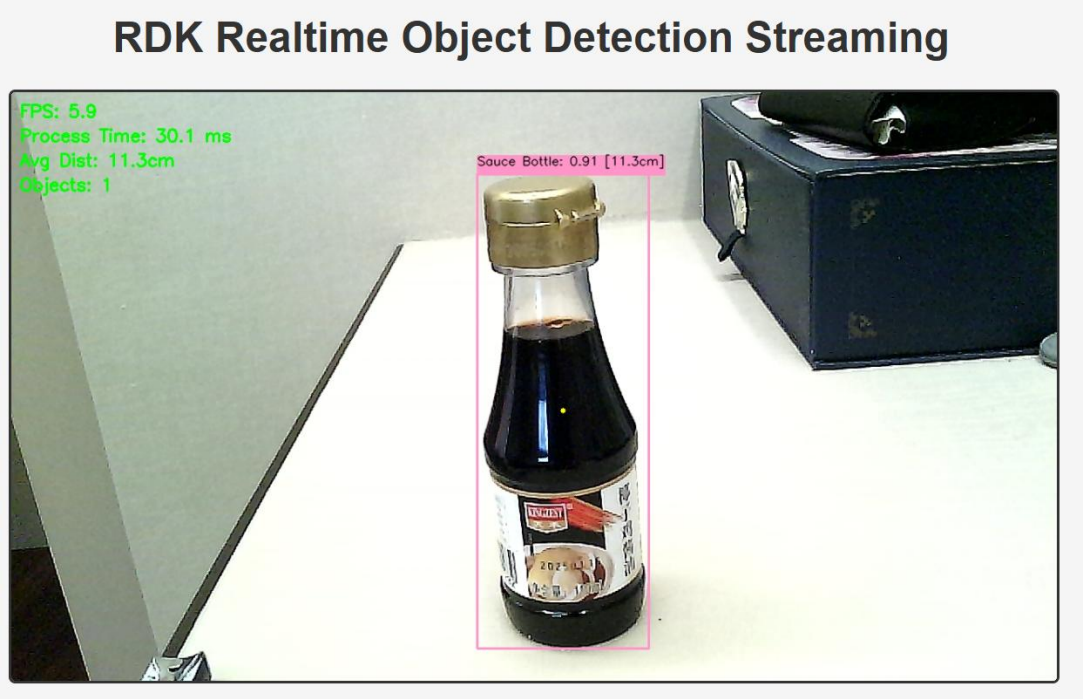

Figure 6-4- 14 Theoretical result: 11.3cm

Figure 6-4- 15 Actual result: 16.5cm

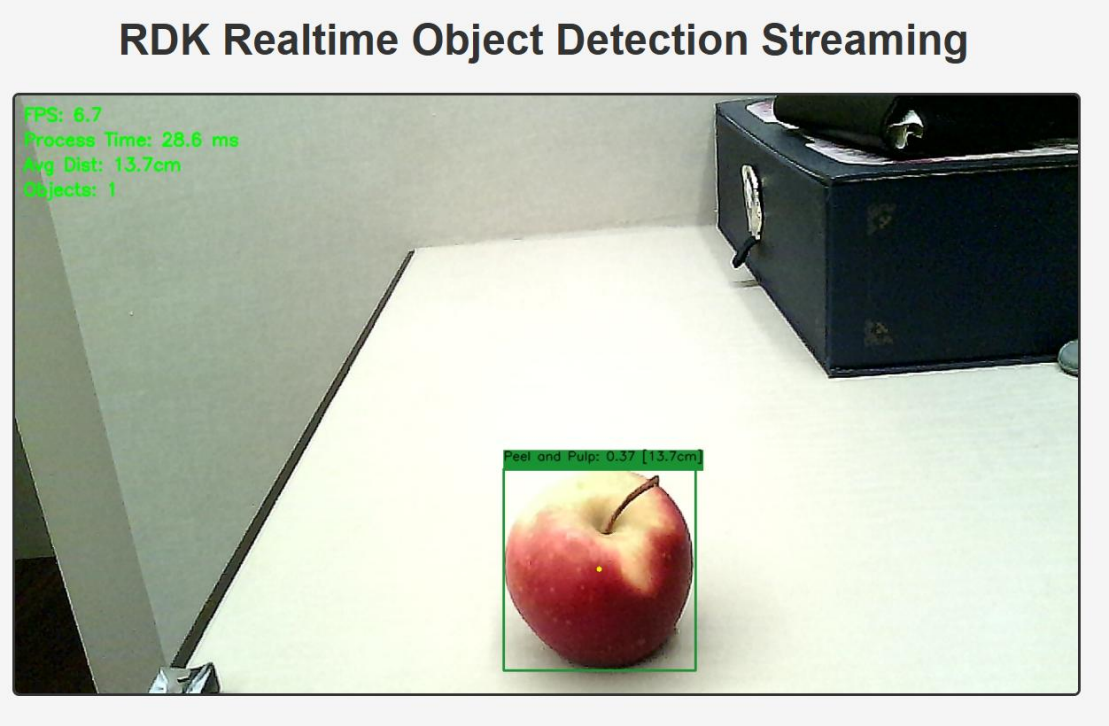

Figure 6-4- 16 Theoretical result: 13.7cm

Figure 6-4- 17 Actual result: 15.5cm

It can be seen that the ranging error is large, so an offset is added manually by the experience of the test data for the time being, and then the method of ranging needs to be improved to reduce the error.

6.5 Inverse kinematics

The code for inverse kinematics will be described next.

First, set the joints' length and angles between them:

```c
/*
    设置四个关节的长度
    单位1mm
*/

解释代码 | 注释代码 | 生成单测 | ×
void setup_kinematics(float L0, float L1, float L2, float L3, kinematics_t *kinematics) {
    //放大10倍
    kinematics->L0 = L0*10;
    kinematics->L1 = L1*10;
    kinematics->L2 = L2*10;
    kinematics->L3 = L3*10;
}

/*
    x,y 为映射到平面的坐标
    z为距离地面的距离
    Alpha 为爪子和平面的夹角 -25~-65范围比较好
*/

解释代码 | 注释代码 | 生成单测 | ×
int kinematics_analysis(float x, float y, float z, float Alpha, kinematics_t *kinematics) {
    float theta3, theta4, theta5, theta6;
    float l0, l1, l2, l3;
    float aaa, bbb, ccc, zf_flag;

    //放大10倍
    x = x*10;
    y = y*10;
    z = z*10;


    l0 = kinematics->L0;
    l1 = kinematics->L1;
    l2 = kinematics->L2;
    l3 = kinematics->L3;
```

Figure 6-5- 1 z_kinematics.c (1)

When x=0, only the y axis is considered, that is, the axis of the front and rear of the robot arm, otherwise the rotation Angle of the base needs to be considered. Due to the limited time, only the single axis of the project is temporarily realized, and the x axis will be added later:

```c
int kinematics_analysis(float x, float y, float z, float Alpha, kinematics_t *kinematics) {
    if(x == 0) {
        theta6 = 0.0;
    } else {
        theta6 = atan(x/y)*180.0/pi;
    }
```

Figure 6-5- 2 z_kinematics.c (2)

Next, the distance that the robot arm needs to reach forward along the Y-axis and descend along the z-axis (the height of the robot arm at the height of the frame) is calculated:

```
y = sqrt(x*x + y*y);           // 计算水平距离
y = y-l3*cos(Alpha*pi/180.0); // 考虑夹爪角度对水平距离的影响
z = z-l0-l3*sin(Alpha*pi/180.0); // 考虑基座高度和夹爪角度对垂直距离的影响
```

Figure 6-5- 3 z_kinematics.c (3)

Check if the location is within reach:

```
if(z < -l0) return 1;          // 目标低于基座范围
if(sqrt(y*y + z*z) > (l1+l2)) return 2; // 超出机械臂最大伸展范围
```

Figure 6-5- 4 z_kinematics.c (4)

Use the geometric solution of the law of cosines to calculate the angles between the joints and make sure they are within the allowable angles:

```
C z_kinematics.c > ...
int kinematics_analysis(float x, float y, float z, float Alpha, kinematics_t *kinematics) {

    ccc = acos(y / sqrt(y * y + z * z));
    bbb = (y*y+z*z+l1*l1-l2*l2)/(2*l1*sqrt(y*y+z*z));
    if(bbb > 1 || bbb < -1) {
        return 5;
    }
    if (z < 0) {
        zf_flag = -1;
    } else {
        zf_flag = 1;
    }
    theta5 = ccc * zf_flag + acos(bbb);
    theta5 = theta5 * 180.0 / pi;
    if(theta5 > 180.0 || theta5 < 0.0) {
        return 6;
    }

    aaa = -(y*y+z*z-l1*l1-l2*l2)/(2*l1*l2);
    if (aaa > 1 || aaa < -1) {
        return 3;
    }
    theta4 = acos(aaa);
    theta4 = 180.0 - theta4 * 180.0 / pi ;
    if (theta4 > 135.0 || theta4 < -135.0) {
        return 4;
    }

    theta3 = Alpha - theta5 + theta4;
    if(theta3 > 90.0 || theta3 < -90.0) {
        return 7;
    }
```

Figure 6-5- 5 z_kinematics.c (5)

Convert the angle to the motor PWM value:

```
kinematics->servo_angle[0] = theta6;
kinematics->servo_angle[1] = theta5-90;
kinematics->servo_angle[2] = theta4;
kinematics->servo_angle[3] = theta3;

kinematics->servo_pwm[0] = (int)(1500-2000.0 * kinematics->servo_angle[0] / 270.0);
kinematics->servo_pwm[1] = (int)(1500+2000.0 * kinematics->servo_angle[1] / 270.0);
kinematics->servo_pwm[2] = (int)(1500+2000.0 * kinematics->servo_angle[2] / 270.0);
kinematics->servo_pwm[3] = (int)(1500+2000.0 * kinematics->servo_angle[3] / 270.0);

return 0;
```

Figure 6-5- 6 z_kinematics.c (6)

Pass the coordination to *kinematics_analysis* and pick at the best position:

```
int kinematics_move(float x, float y, float z, int time) {
    int i,j, min = 0, flag = 0;

    if(y < 0)return 0;

    //寻找最佳角度
    flag = 0;
    for(i=0;i>=-135;i--) {
        if(0 == kinematics_analysis(x,y,z,i,&kinematics)){
            if(i<min)min = i;
            flag = 1;
        }
    }

    //用3号舵机与水平最大的夹角作为最佳值
    if(flag) {
        kinematics_analysis(x,y,z,min,&kinematics);
        for(j=0;j<4;j++) {
            set_servo(j, kinematics.servo_pwm[j], time);
        }
        return 1;
    }

    return 0;
}
```

Figure 6-5- 7 z_main.c – *kinematics_move*

6.6 UART communication

After the development of the inverse kinematics program, a communication mechanism needs to be established between the main development board and the robot arm controller. After the main development transmits the category and distance of the object to the robot arm controller, the controller analyzes the instructions and controls the robot arm to complete the corresponding action and place the object in the corresponding position.

6.6.1 Robot arm controller side

First, it is required to modify the original UART instruction receiver program of the robot arm controller, because the original program can only receive instructions in a specific format:

z_usart.c > USART1_IRQHandler(void)

```
int USART1_IRQHandler(void) {
    u8 sbuf_bak;
    static u16 buf_index = 0;

    if(USART_GetFlagStatus(USART1,USART_IT_RXNE)==SET) {
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
        sbuf_bak = USART_ReceiveData(USART1);
        //uart1_send_byte(sbuf_bak);
        if(uart1_get_ok)return 0;
        if(sbuf_bak == '<') {
            uart1_mode = 4;
            buf_index = 0;
        }else if(uart1_mode == 0) {
            if(sbuf_bak == '$') {           //命令模式 $XXX!
                uart1_mode = 1;
            } else if(sbuf_bak == '#') {    //单舵机模式    #000P1500T1000! 类似这种命令
                uart1_mode = 2;
            } else if(sbuf_bak == '{') {    //多舵机模式    {#000P1500T1000!#001P1500T1000!} 多个单舵机命
                uart1_mode = 3;
            } else if(sbuf_bak == '<') {    //保存动作组模式    <G0000#000P1500T1000!#001P1500T1000!B000!
                uart1_mode = 4;
            } else if(sbuf_bak == '%') {    //识别物体距离和类别    %D15.0C13
                uart1_mode = 5;
            }
            buf_index = 0;
        }

        uart_receive_buf[buf_index++] = sbuf_bak;

        if((uart1_mode == 4) && (sbuf_bak == '>')){
            uart_receive_buf[buf_index] = '\0';
            uart1_get_ok = 1;
        } else if((uart1_mode == 1) && (sbuf_bak == '!')){
            uart_receive_buf[buf_index] = '\0';
            uart1_get_ok = 1;
        } else if((uart1_mode == 2) && (sbuf_bak == '!')){
            uart_receive_buf[buf_index] = '\0';
            uart1_get_ok = 1;
        } else if((uart1_mode == 3) && (sbuf_bak == '}')){
            uart_receive_buf[buf_index] = '\0';
            uart1_get_ok = 1;
        } else if((uart1_mode == 5) && (sbuf_bak == '*')){
```

Figure 6-6- 1 z_uart.c

As can be seen from the above code, the program will only process instructions with a specific start symbol and end symbol, so it is necessary to design a special format packaging data, that is, "%" as the start character, "*" as the end character, the middle contains two data transmission distance and category, so D and C are used as delimiters to avoid confusion. The resulting format is %D15.0C0*, where 15.0 is a floating-point number representing the distance; 0 is an integer representing the class, and the specific data parsing code is as follows:

```
int parse_data(float *distance, int *category) {
    char *d_ptr = NULL;
    char *c_ptr = NULL;
    char *end_ptr = NULL;
    float temp_dist = 0.0;
    int temp_cat = -1;

    // 初始化输出参数
    *distance = 0.0;
    *category = -1;

    sprintf((char*)cmd_return, "Raw input: %s\r\n", uart_receive_buf);
    uart1_send_str(cmd_return);

    // 查找格式标识
    if(strstr((char*)uart_receive_buf, "%D") != NULL) {
        d_ptr = strstr((char*)uart_receive_buf, "%D");
    } else {
        d_ptr = strstr((char*)uart_receive_buf, "D");
    }

    if(d_ptr == NULL) {
        return 0; // 未找到距离标识
    }

    c_ptr = strstr(d_ptr, "C");
    if(c_ptr == NULL) {
        return 0; // 未找到类别标识
    }

    end_ptr = strstr(c_ptr, "*");
    if(end_ptr == NULL) {
        // 如果没有找到星号，尝试继续解析（兼容旧格式）
    }

    // 解析格式：%D15.0C13* - D后面是距离，C后面是类别，*是结束符
    if(d_ptr[0] == '%') {
        if(sscanf(d_ptr, "%%D%f", &temp_dist) != 1) {
            return 0; // 距离解析失败
        }
    } else {
        if(sscanf(d_ptr, "D%f", &temp_dist) != 1) {
```

Figure 6-6- 2 z_sensor.c – *parse_data* (1)

```c
int parse_data(float *distance, int *category) {
    // 解析格式：%D15.0C13* - D后面是距离，C后面是类别，*是结束符
    if(d_ptr[0] == '%') {
        if(sscanf(d_ptr, "%%D%f", &temp_dist) != 1) {
            return 0; // 距离解析失败
        }
    } else {
        if(sscanf(d_ptr, "D%f", &temp_dist) != 1) {
            return 0; // 距离解析失败
        }
    }

    if(sscanf(c_ptr, "C%d", &temp_cat) != 1) {
        return 0; // 类别解析失败
    }

    // 验证解析结果有效性
    if(temp_dist <= 0.0 || temp_cat < 0) {
        return 0; // 解析值不合理
    }

    // 解析成功，设置输出参数
    *distance = temp_dist;
    *category = temp_cat;

    return 1; // 解析成功
}
```

Figure 6-6- 3 z_sensor.c – *parse_data* (2)

After extracting the data, the executability of the data needs to be judged. If it is within the specified range, the action flow function of grabbing the object and placing it in the specified position is called. Similar to the finite state machine, the global variable for the number of steps is updated after each step is complete. If the number of steps has changed, it is called again to execute the next step. A time interval between each step is also set to ensure that the action completes properly:

```c
void ceju_jiaqu(void) {
    static float dis = 0.0;
    static u32 last_state_change = 0;

    if(group_do_ok == 0) return; //有动作执行，直接返回

    // 检查串口是否接收到数据
    if(uart1_get_ok) {
        float distance = 0.0;
        int cat = -1;

        if(parse_data(&distance, &cat)) {
            dis = distance;
            category = cat;

            // 根据接收到的数据触发动作
            if(dis != 0.0 && category != -1) {
                kms_y = dis*10 + 120;
                sprintf((char*)cmd_return, "dis=%f cm, category=%d, kms_y=%d\r\n", dis, category, kms_y);
                uart1_send_str(cmd_return); // 发送目标位置和类别信息
                // 重置状态机，开始自动执行
                carry_step = 1;
                last_state_change = millis();
            }
        }

        uart1_get_ok = 0; // 清除接收标志
    }

    // 自动执行状态机的下一步
    if(carry_step != 0 && millis() - last_state_change >= 2500) { // 防抖
        carry_wood();
        last_state_change = millis();
    }
}
```

Figure 6-6- 4 z_sensor.c –*ceju_jiaqu*

```
void carry_wood(void) {
    //张开爪子
    if(carry_step == 1){
        set_servo(5,1200,1000);
        //mdelay(500);
        carry_step = 2;
        uart1_send_str("carry_wood step 1");
    //运行到目标位置
    }else if(carry_step == 2){
        uart1_send_str("carry_wood step 2");
        if(kinematics_move(0,kms_y,15,1500)){
            uart1_send_str("move test");
            beep_on_times(1,100);
            //mdelay(2000);
            carry_step = 3;
            uart1_send_str("carry_wood step 2");
        }else{
            carry_step = 0;
            return;
        }
    //夹取
    }else if(carry_step == 3){
        uart1_send_str("pick test");
        set_servo(5,1800,1000);
        //mdelay(500);
        carry_step = 4;
    //分类
    }else if(carry_step == 4){
        uart1_send_str("up test");
        set_servo(1,1200,1500);
        carry_step = 5;
    }else if(carry_step == 5){
        uart1_send_str("classify test");
        switch(category){
            case 0:
                //可回收
                do_group_once(39);
                break;
            case 1:
                //厨余垃圾
                do_group_once(40);
                break;
```

```
void carry_wood(void) {
    }else if(carry_step == 5){
        switch(category){
                break;
            case 1:
                //厨余垃圾
                do_group_once(40);
                break;
            case 2:
                //有害垃圾
                do_group_once(41);
                break;
            case 3:
                //其他垃圾
                do_group_once(42);
                break;
        }
        carry_step = 6;
    }else if(carry_step == 6){
        uart1_send_str("drop test");
        set_servo(5,1200,1000);
        carry_step = 7;
    }else if(carry_step == 7){
        parse_group_cmd("$RST!");
        if(group_do_ok == 1){
            carry_step = 0;
        }
    }
}
```

Figure 6-6- 5 z_sensor.c –*carry_wood*

Add a garbage sorting action group to the debugging software provided by the manufacturer:

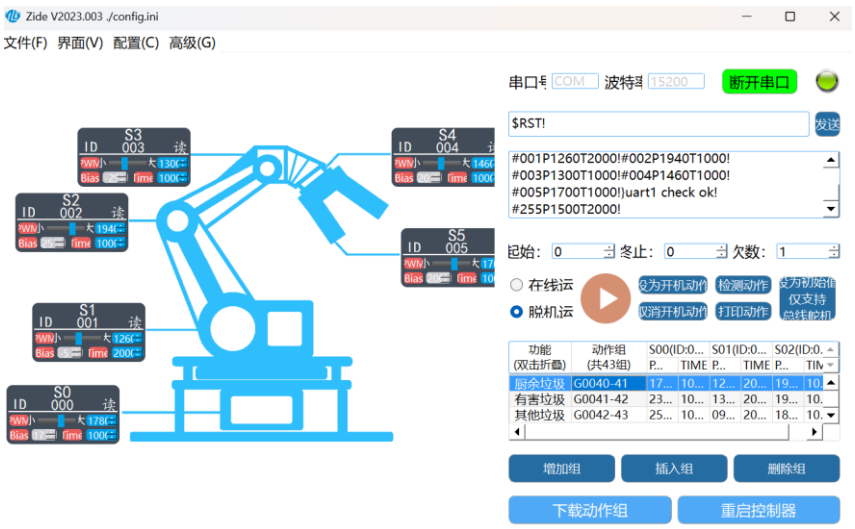

Figure 6-6- 6 Add action groups for sorting

6.6.2 *Main development board side*

On the main development board, a serial port sending script needs to be implemented in order to transmit data to the controller. The key code segment is as follows:

```
def send_data_to_serial(port, data, baud_rate=115200, timeout=1, hex_mode=False, keep_open=True):
        else:
            # 普通字符串，编码为bytes
            send_bytes = data.encode('utf-8')

        # 显示要发送的数据
        logger.info(f"发送数据: '{data}'")
        logger.info(f"十六进制形式: {' '.join([f'0x{b:02X}' for b in send_bytes])} ({len(send_bytes)} 字节)")

        # 发送数据
        bytes_sent = ser.write(send_bytes)
        logger.info(f"已发送 {bytes_sent} 字节")

        # 等待数据发送完成
        ser.flush()

        # 尝试读取回复(如果有)
        time.sleep(0.5)  # 给设备一点响应时间
        response_bytes = bytearray()

        start_time = time.time()
        while time.time() - start_time < timeout:
            if ser.in_waiting:
                response_bytes.extend(ser.read(ser.in_waiting))
                time.sleep(0.1)  # 短暂等待更多数据
            else:
                break

        if response_bytes:
            logger.info(f"收到回复: {' '.join([f'0x{b:02X}' for b in response_bytes])} ({len(response_bytes)} 字节)")

            # 尝试以不同编码解码响应
            try:
                # 尝试UTF-8解码
                utf8_text = response_bytes.decode('utf-8', errors='replace')
                logger.info(f"UTF-8解码: {utf8_text}")
            except UnicodeDecodeError:
                pass
```

Figure 6-6- 7 control.py

The program mainly uses the serial library to send a string to the specified serial port and receives the reply from the other end.

The main program of the development board also needs to realize the data transmission function. The approximate process is that when the category, bounding box and distance of the object are stable in a certain period of time (reach a certain threshold), their category and distance are sent to the controller by calling the serial port transmission script:

```python
class ObjectTracker:
    解释代码 | 注释代码 | 生成单测 | ×
    def __init__(self, stability_threshold=10, distance_threshold=5.0, position_threshold=20):
        """
        初始化物体跟踪器

        参数:
            stability_threshold: 判定物体稳定所需的连续帧数
            distance_threshold: 判定距离稳定的阈值(cm)
            position_threshold: 判定位置稳定的阈值(像素)
        """
        self.tracked_objects = {}  # 跟踪的物体 {id: {positions:[], distances:[], class_id:, stable_count:, sent:}}
        self.stability_threshold = stability_threshold
        self.distance_threshold = distance_threshold
        self.position_threshold = position_threshold

    解释代码 | 注释代码 | 生成单测 | ×
    def update(self, detections):
        """
        更新跟踪的物体

        参数:
            detections: 列表，每个元素是 (center_x, center_y, distance, class_id, score)

        返回:
            stable_objects: 刚刚变得稳定的物体列表，每个元素是 (avg_distance, class_id)
        """
        global last_command_time
        stable_objects = []

        # 检查是否可以发送新指令
        current_time = time()
        can_send_new_command = (current_time - last_command_time >= 20)

        # 如果不能发送新指令，就不需要检测新的稳定物体
        if not can_send_new_command:
            return []

        # 标记所有当前物体为未更新
        for obj_id in self.tracked_objects:
            self.tracked_objects[obj_id]['updated'] = False
```

Figure 6-6- 8 wifi.py – *ObjectTracker* (1)

```python
class ObjectTracker:
    def update(self, detections):

        # 为每个检测匹配或创建跟踪对象
        for center_x, center_y, distance, class_id, score in detections:
            if distance is None:
                continue

            # 尝试匹配到现有物体
            matched = False
            for obj_id, obj_data in self.tracked_objects.items():
                if obj_data['class_id'] == class_id and not obj_data['updated']:
                    # 检查位置距离是否接近
                    last_x, last_y = obj_data['positions'][-1]
                    pos_distance = np.sqrt((center_x - last_x)**2 + (center_y - last_y)**2)

                    if pos_distance < self.position_threshold:
                        # 匹配成功，更新数据
                        obj_data['positions'].append((center_x, center_y))
                        obj_data['distances'].append(distance)
                        obj_data['updated'] = True

                        # 保持最近的10个数据点
                        if len(obj_data['positions']) > 10:
                            obj_data['positions'].pop(0)
                            obj_data['distances'].pop(0)

                        # 判断物体是否稳定
                        if not obj_data['sent']:
                            # 计算距离方差
                            if len(obj_data['distances']) >= self.stability_threshold:
                                recent_distances = obj_data['distances'][-self.stability_threshold:]
                                distance_std = np.std(recent_distances)

                                recent_positions = obj_data['positions'][-self.stability_threshold:]
                                x_std = np.std([p[0] for p in recent_positions])
                                y_std = np.std([p[1] for p in recent_positions])

                                # 如果位置和距离都稳定
                                if distance_std < self.distance_threshold and x_std < self.position_threshold/2 and
                                    obj_data['stable_count'] += 1
                                else:
```

Figure 6-6- 9 wifi.py – *ObjectTracker* (2)

```
class ObjectTracker:
    def update(self, detections):
                                    obj_data['stable_count'] = 0

                                # 如果连续几帧都稳定，则认为物体稳定
                                if obj_data['stable_count'] >= 3:
                                    avg_distance = np.mean(recent_distances)
                                    stable_objects.append((avg_distance, class_id))
                                    obj_data['sent'] = True

                            matched = True
                            break

            # 如果没有匹配到现有物体，创建新物体
            if not matched:
                new_id = len(self.tracked_objects) + 1
                self.tracked_objects[new_id] = {
                    'positions': [(center_x, center_y)],
                    'distances': [distance],
                    'class_id': class_id,
                    'stable_count': 0,
                    'updated': True,
                    'sent': False
                }

        # 移除长时间未更新的物体
        ids_to_remove = []
        for obj_id, obj_data in self.tracked_objects.items():
            if not obj_data['updated']:
                ids_to_remove.append(obj_id)

        for obj_id in ids_to_remove:
            del self.tracked_objects[obj_id]

        return stable_objects
```

Figure 6-6- 10 wifi.py – *ObjectTracker* (3)

The data is sent to the controller through the serial port transmission script according to the previously specified format, and the distance limit is added to avoid invalid transmission and a certain time interval limit is added to avoid frequent transmission:

```
def send_to_serial(distance, category):

    global last_command_time
    current_time = time()

    # 检查是否满足发送时间间隔要求
    if current_time - last_command_time < 20:
        time_to_wait = 20 - (current_time - last_command_time)
        logger.info(f"上次发送指令距现在只有 {current_time - last_command_time:.1f} 秒，"
                    f"需等待 {time_to_wait:.1f} 秒")
        return

    # 距离限制：大于20cm不发送
    if distance > 20:
        logger.info(f"距离 {distance:.1f}cm 超过20cm限制，不发送指令")
        return

    command = f"%D{distance:.1f}C{category}*"
    logger.info(f"发送指令: {command}")

    # 构建调用命令
    import subprocess

    # 发送命令
    result = subprocess.run(['python', 'control.py', '--data', command, '--keep-open'],
                            capture_output=True, text=True)

    # 更新最后发送时间
    last_command_time = time()

    logger.info(f"发送结果: {result.stdout}")
    if result.stderr:
        logger.error(f"发送错误: {result.stderr}")
```

Figure 6-6- 11 wifi.py – *send_to_serial*

6.7 System integration and final test

The complete robotic arm is shown in the picture. The camera is fixed at the first link of the robotic arm. The main development board connects the controller of the robotic arm through the USB port, and the development computer transmits and runs files through the type-c port of the main development board:

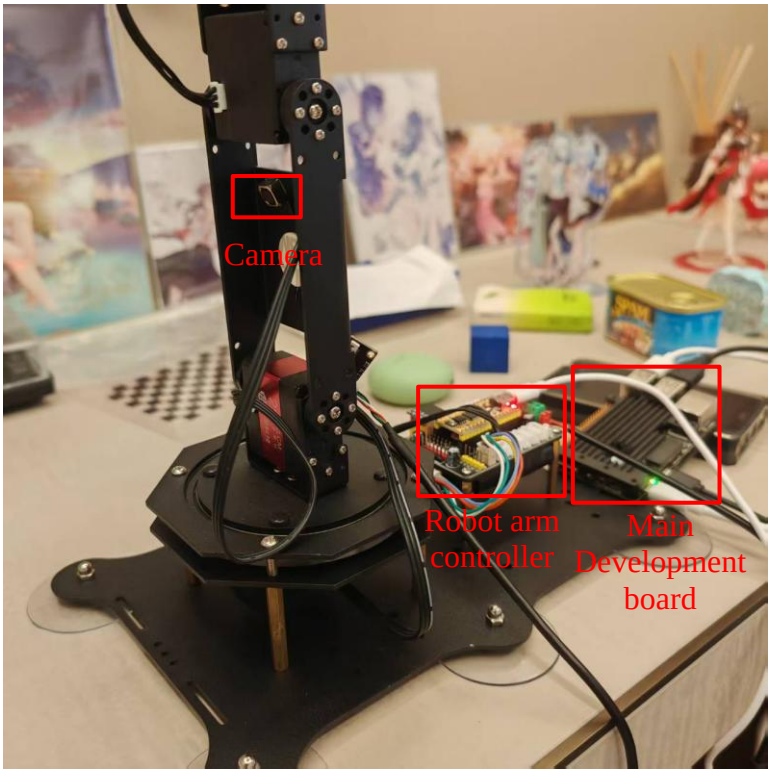

Figure 6-7- 1 System integration

Followings are the log of testing:

```
363 updates can be applied immediately.
120 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

117 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Sun Apr 13 20:20:38 2025 from 192.168.128.100
sunrise@ubuntu:~$ cd ./Desktop/data/
sunrise@ubuntu:~/Desktop/data$ python wifi.py --ranging
[RDK_YOLO] [14:42:29.795] [INFO] 已加载物体高度信息: [5.5, 0, 0, 5.5, 4.7, 0, 0, 9.0, 5.6, 10.0, 8.2, 5.0, 9.7, 0, 10.0, 0, 0, 4.0, 13.1, 0, 5.5, 10.0, 0, 16.0]
[RDK_YOLO] [14:42:29.800] [INFO] Namespace(model_path='/home/sunrise/Desktop/data/bin_dir/yolo11n_detect_bayese_640x640_nv12/yolo11n_detect_bayese_640x640_nv12.bin', camera_id=0,
port=8080, classes_num=24, reg=16, iou_thres=0.45, conf_thres=0.25, ranging=True, focal_length=2465)
[RDK_YOLO] [14:42:29.804] [INFO] 本机IP地址: 192.168.110.36
[RDK_YOLO] [14:42:29.804] [INFO] 打开浏览器访问: http://192.168.110.36:8080 查看实时检测结果
[BPU_PLAT]BPU Platform Version(1.3.6)!
[HBRT] set log level as 0. version = 3.15.55.0
[DNN] Runtime version = 1.24.5_(3.15.55 HBRT)
[A][DNN][packed_model.cpp:247][Model](2025-04-18,14:42:30.24.728) [HorizonRT] The model builder version = 1.23.8
[W][DNN]bpu_model_info.cpp:491][Version](2025-04-18,14:42:30.141.921) Model: yolo11n_detect_bayese_640x640_nv12. Inconsistency between the hbrt library version 3.15.55.0 and the m
odel build version 3.15.54.0 detected, in order to ensure correct model results, it is recommended to use compilation tools and the BPU SDK from the same OpenExplorer package.
[RDK_YOLO] [14:42:30.148] [INFO] -> input tensors
[RDK_YOLO] [14:42:30.149] [INFO] intput[0], name=images, type=uint8, shape=(1, 3, 640, 640)
[RDK_YOLO] [14:42:30.149] [INFO] -> output tensors
[RDK_YOLO] [14:42:30.149] [INFO] output[0], name=output0, type=float32, shape=(1, 80, 80, 64)
[RDK_YOLO] [14:42:30.149] [INFO] output[1], name=469, type=float32, shape=(1, 40, 40, 64)
[RDK_YOLO] [14:42:30.150] [INFO] output[2], name=477, type=float32, shape=(1, 20, 20, 64)
[RDK_YOLO] [14:42:30.150] [INFO] output[3], name=491, type=float32, shape=(1, 80, 80, 24)
[RDK_YOLO] [14:42:30.150] [INFO] output[4], name=505, type=float32, shape=(1, 40, 40, 24)
[RDK_YOLO] [14:42:30.150] [INFO] output[5], name=519, type=float32, shape=(1, 20, 20, 24)
[RDK_YOLO] [14:42:30.152] [INFO] iou threshol = 0.45, conf threshol = 0.25
[RDK_YOLO] [14:42:30.153] [INFO] 已启用距离测量功能 - 相机高度: 17.0cm, 倾斜角: 31.4°
 * Serving Flask app 'wifi'
 * Debug mode: off
[werkzeug] [14:42:30.175] [INFO] WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://192.168.110.36:8080
[werkzeug] [14:42:30.175] [INFO] Press CTRL+C to quit
[RDK_YOLO] [14:42:30.333] [INFO] 成功打开摄像头，开始实时检测
[RDK_YOLO] [14:42:34.193] [INFO] 检测到稳定物体: 类别=Contaminated paper(11), 距离=22.6cm, 垃圾分类=3
[RDK_YOLO] [14:42:34.193] [INFO] 距离 22.6cm 超过20cm限制，不发送指令
```

Figure 6-7- 2 Command line log

Figure 6-7- 3 Real-time streaming on web server

After all the program development work, the test cases and evaluation indicators need to be designed. Since the distance of the object needs to be measured according to the height of the object, only one object of each category can be selected. The performance evaluation link requires the recognition and classification accuracy of the statistical model, as well as the picking and placement success rate of the robot arm. The complete test data is as follows, with the missing class being the untested class:

	Identification Correct	Pick success	Sorting correcct	Posit Correct
Disposable Fast Food Box	0	0	0	0
Book Paper	1	0	1	0
Plastic Utensils	0	0	1	0
Plastic Toys	0	0	1	0
Dry Battery	1	0	1	0
Express Paper Bag	0	0	1	0
Plug Wire	1	1	1	1
Can	1	0	1	0
Peel and Pulp	1	0	1	0
Stuffed Toy	1	0	1	0
Defiled Plastic	1	1	1	1
Contaminated paper	1	1	1	1
Toilet care products	1	0	1	0
Cigarette butts				
Carton box	1	1	1	1
Tea residue	1	1	1	1
Cai Bang Cai Ye	1	1	1	1
Egg Shell	1	0	1	0
Sauce Bottle	1	0	1	0
Ointment	1	1	1	1
Expired Medicine	0	1	0	1
Metal Food Cans	0	0	1	0
edible oil drums				
drink bottles	1	1	1	0
Summary	18/22	9/22	20/22	8/22

Table 6-7- 1 Results of final testing

It can be seen that the picking success rate of the robot arm is relatively low, and the subsequent optimization of ranging and picking position is needed.

Chapter 7 Conclusion

7.1 Summary

This project establishes a functional automated waste sorting system through multidisciplinary integration, validating the viability of lightweight edge AI solutions in sustainable waste management, offering a replicable template for communities lacking advanced recycling infrastructure.

7.1.1 Core achievements

1. **Cross-domain synergy**: Successfully combined YOLOv11 detection (81% mAP50), geometric monocular ranging, and DH-parameterized inverse kinematics into an embedded deployment (RDK X5 + OV5640 camera + Jibot1-stm32 arm).
2. **Practical validation**: Handling objects within 0-15cm height range through tilt-compensated distance estimation, the robot arm achieved a certain success rate of grasping and placing, regardless of objects that were too large in size or did not have suitable grasping points.
3. **Scalability**: Since the main development board and the robot arm controller have sufficient external interfaces, the project supports modular development, and subsequent project optimization and further expansion can be carried out.

7.1.2 Critical limitations

- Dataset coverage limited to 22 categories, and does not cover enough environmental conditions, such as lighting, background color, etc., which leads to large external influences when running inference.
- Fixed y-axis assumption restricts horizontal grasping range.
- Fixed picking angle affects the success rate of picking objects.
- Obvious ranging errors (e.g., 15.5cm measured vs. 13.7cm theoretical).
- Requiring heights of testing objects results in poor practicability of the system.
- The grasping direction is limited, so there are strict requirements for the direction of object placement, resulting in a temporary lack of satisfying adaptability of the project.

7.2 Future work

In view of the limitations of the current system, it is necessary to improve the practicality and stability of the system in real application scenarios in the following aspects：

1. **Dataset expansion**: Include 50+ common household waste types through synthetic data generation.
2. **Full 3D manipulation**: Add x-axis rotation to enable 360° horizontal grasping.
3. **Enhanced ranging and grasping accuracy**: Implement MiDaS [19] monocular depth estimation combined with GraspNet [13] for adaptive grasp poses.
4. **Mobile platform**: Integrate autonomous navigation chassis for area coverage.

According to the idea in the early stage of the project, an intelligent waste sorting vehicle should be finally realized, which has functions such as automatic cruise, obstacle avoidance and automatic search for nearby waste for picking and sorting. However, due to the cost and time constraints, the expected effect has not been achieved for the time being. I hope that the initial idea can be realized after learning more advanced knowledge and methods.

References

[1] UNEP. (2024, February 24). Global Waste Management Outlook 2024. https://www.unep.org/resources/global-waste-management-outlook-2024

[2] Kaza, S., et al. (2021, April 27). What a waste 2.0: A global snapshot of solid waste management to 2050. World Bank. https://documents.worldbank.org/en/publication/documents-reports/documentdetail/697271544470229584/what-a-waste-2-0-a-global-snapshot-of-solid-waste-management-to-2050

[3] Directorate-General for Economic and Financial Affairs. (2024, April 18). 2024 Ageing Report. Economic and Budgetary Projections for the EU Member States (2022-2070). https://economy-finance.ec.europa.eu/publications/2024-ageing-report-economic-and-budgetary-projections-eu-member-states-2022-2070_en

[4] Development Research Center of The State Council. (2020, October 15). Analysis of the Supply and Demand Trend of China's Labor Force in the Next Ten Years. https://www.drc.gov.cn/DocView.aspx?chnid=379&leafid=1338&docid=2901905

[5] Will Douglas Heaven. (2019, April 11). This Robot Can Sort Recycling by Giving It a Squeeze. MIT Technology Review. https://www.technologyreview.com/2019/04/11/1141/this-robot-can-sort-recycling-by-giving-it-a-squeeze/

[6] TOMRA. (n.d.). AUTOSORT™: Advanced Multifunctional Sorting System. https://www.tomra.com/en/waste-metal-recycling/products/machines/autosort

[7] UN Environment Programme. (n.d.). GOAL 12: Responsible Consumption and Production. https://sdgs.unep.org/goal-12

[8] Ultralytics. (2024). Ultralytics yolov11. https://docs.ultralytics.com/models/yolo11

[9] Huawei Cloud Cup 2020 Shenzhen Open Data Application Innovation Competition - Household Waste Image Classification (Object Detection). (2020, June 26). https://blog.csdn.net/qq_38410428/article/details/106974147

[10] Xiang, Y., Schmidt, T., Narayanan, V., & Fox, D. (2017). POseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1711.00199

[11] Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., & Savarese, S. (2019). DenseFusion: 6D object pose estimation by iterative dense fusion. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1901.04780

[12] Mahler, J., Matl, M., Satish, V., Danielczuk, M., DeRose, B., McKinley, S., & Goldberg, K. (2019). Learning ambidextrous robot grasping policies. Science Robotics, 4(26). https://doi.org/10.1126/scirobotics.aau4984

[13] Fang, H., Wang, C., Gou, M., & Lu, C. (2020). GRAspNet-1Billion: a Large-Scale Benchmark for General Object Grasping. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 11441–11450. https://doi.org/10.1109/cvpr42600.2020.01146

[14] Gordon, A., Li, H., Jonschkowski, R., & Angelova, A. (2019). Depth from videos in the Wild: Unsupervised monocular depth learning from unknown cameras. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). https://doi.org/10.1109/iccv.2019.00907

[15] Monocular Visual odometry scale recovery using geometrical constraint. (2018, May 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/8462902

[16] Iyer, G., Murthy, J. K., Gupta, G., Krishna, K. M., & Paull, L. (2018). Geometric consistency for Self-Supervised End-to-End visual odometry. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). https://doi.org/10.1109/cvprw.2018.00064

[17] Godard, C., Mac Aodha, O., Firman, M., & Brostow, G. (2019b). Digging into Self-Supervised Monocular depth estimation. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). https://doi.org/10.1109/iccv.2019.00393

[18] Guizilini, V., Ambrus, R., Pillai, S., Raventos, A., & Gaidon, A. (2020). 3D packing for Self-Supervised Monocular depth Estimation. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2482–2491. https://doi.org/10.1109/cvpr42600.2020.00256

[19] Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., & Koltun, V. (2019, July 2). Towards robust monocular depth estimation: mixing datasets for zero-shot cross-dataset transfer. arXiv.org. https://arxiv.org/abs/1907.01341v3

[20] Watson, J., Firman, M., Brostow, G., & Turmukhambetov, D. (2019). Self-Supervised monocular depth hints. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). https://doi.org/10.1109/iccv.2019.00225

[21] Kucuk, S., & Bingul, Z. (2013). Inverse kinematics solutions for industrial robot manipulators with offset wrists. Applied Mathematical Modelling, 38(7–8), 1983–1999. https://doi.org/10.1016/j.apm.2013.10.014

[22] Chen, Q., Zhu, S., & Zhang, X. (2015). Improved inverse kinematics algorithm using screw theory for a Six-DOF robot manipulator. International Journal of Advanced Robotic Systems, 12(10). https://doi.org/10.5772/60834

[23] Aristidou, A., & Lasenby, J. (2011). FABRIK: A fast, iterative solver for the Inverse Kinematics problem. Graphical Models, 73(5), 243–260. https://doi.org/10.1016/j.gmod.2011.05.003

[24] Csiszar, A., Eilers, J., & Verl, A. (2017b). On solving the inverse kinematics problem using neural networks. 2021 27th International Conference on Mechatronics and Machine Vision in Practice (M2VIP). https://doi.org/10.1109/m2vip.2017.8211457

[25] Lembono, T. S., Paolillo, A., Pignat, E., & Calinon, S. (2020). Memory of motion for Warm-Starting trajectory optimization. IEEE Robotics and Automation Letters, 5(2), 2594–2601. https://doi.org/10.1109/lra.2020.2972893

[26] Ohara, S., Ogata, T., & Awano, H. (2021). Binary Neural network in robotic manipulation: flexible object manipulation for humanoid robot using partially binarized Auto-Encoder on FPGA. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 6010–6015. https://doi.org/10.1109/iros51168.2021.9636825

[27] Krejčí, J., Babiuch, M., Babjak, J., Suder, J., & Wierbica, R. (2022). Implementation of an Embedded System into the Internet of Robotic Things. Micromachines, 14(1), 113. https://doi.org/10.3390/mi14010113

[28] Ahmed, A. S., Marzog, H. A., & Abdul-Rahaim, L. A. (2021). Design and implement of robotic arm and control of moving via IoT with Arduino ESP32. International Journal of Power Electronics and Drive Systems/International Journal of Electrical and Computer Engineering, 11(5), 3924. https://doi.org/10.11591/ijece.v11i5.pp3924-3933

[29] Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., & Goldberg, K. (2017). DEX-Net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1703.09312

[30] Real-time grasp detection using convolutional neural networks. (2015). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/7139361

[31] Yan, X., Hsu, J., Khansari, M., Bai, Y., Pathak, A., Gupta, A., Davidson, J., & Lee, H. (2017). Learning 6-DOF grasping interaction via deep geometry-aware 3D representations. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1708.07303

[32] Liang, H., Ma, X., Li, S., Gorner, M., Tang, S., Fang, B., Sun, F., & Zhang, J. (2019). PointNetGPD: Detecting Grasp Configurations from Point Sets. 2022 International Conference on Robotics and Automation (ICRA). https://doi.org/10.1109/icra.2019.8794435

[33] Sundermeyer, M., Mousavian, A., Triebel, R., & Fox, D. (2021). Contact-GraspNet: Efficient 6-DOF grasp generation in cluttered scenes. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2103.14127

[34] D-Robotics. (2024). RDK X5 robot developer kit [Hardware development platform]. D-Robotics Developer. https://developer.d-robotics.cc/rdkx5

[35] Horizon Robotics. (2024, April 22). Horizon Robotics submits IPO application and launches SuperDrive solution [Press release]. Horizon Auto News. https://www.horizon.auto/news/press/224

[36] Embedfire. (2021, January 12). OV5640 camera and HDMI interface design based on Altera EP4CE10 Mini development board [Technical documentation]. Embedfire Documentation. https://doc.embedfire.com/fpga/altera/ep4ce10_mini/zh/latest/fpga/OV5640_HDMI.html

[37] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015, June 8). You only look once: Unified, Real-Time Object Detection. arXiv.org. https://arxiv.org/abs/1506.02640

[38] Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, faster, stronger. arXiv. https://arxiv.org/abs/1612.08242

[39] Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv. https://arxiv.org/abs/1804.02767

[40] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv. https://arxiv.org/abs/2004.10934

[41] Ultralytics. (2020). YOLOv5 documentation. GitHub. https://github.com/ultralytics/yolov5

[42] Wang, C., Bochkovskiy, A., & Liao, H. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2207.02696

[43] Wang, C., Yeh, I., & Liao, H. M. (2024). YOLOV9: Learning what you want to learn using programmable gradient information. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2402.13616

[44] Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., & Ding, G. (2024). YOLOV10: Real-Time End-to-End Object Detection. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2405.14458

[45] CoovallyAIHub. (2025, February 12). Behind the performance leap of YOLO11: from C3k2 to C2PSA, the technical details are fully analyzed. Tencent Cloud. https://cloud.tencent.com/developer/article/2495888

[46] YOLO11 immersive explanation of YOLOV11 network structure and code analysis. (2025, April 1). CSDN. https://blog.csdn.net/qq_64693987/article/details/142668985

[47] YOLO11 is coming | A detailed interpretation of YOLO11 improvements. (2024, October, 21). CSDN. https://blog.csdn.net/qq_40716944/article/details/142993902

[48] Ultralytics latest YOLO11 algorithm principle analysis - including the latest detailed - structure diagram, as well as the detailed structure diagram and code analysis of each part of YOLO11. (2024. November 16). CSDN. https://blog.csdn.net/qq_38668236/article/details/143820170

[49] YOLO Full series modules explained (continuously updated). (2024, October 12). CSDN. https://blog.csdn.net/Alex_Tlover/article/details/142764338

[50] D-robotics Chao. (2024, October 1). YOLOv11, D-Robotics RDK X5 development board, TROS end-to-end 140FPS. D-Robotics Developer Community. https://developer.d-robotics.cc/forumDetail/253775338902237209

[51] D-Robotics. (n.d.). *RDK X5 developer kit documentation (Version 1.26)* [Technical documentation]. D-Robotics Developer. https://developer.d-robotics.cc/api/v1/fileData/x5_doc-v126cn/index.html

[52] Zhang, Z. (2000). A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11), 1330–1334. https://doi.org/10.1109/34.888718

[53] Correia, C. a. M., Andrade, F. a. A., Sivertsen, A., Guedes, I. P., Pinto, M. F., Manhães, A. G., & Haddad, D. B. (2022). Comprehensive direct georeferencing of aerial images for unmanned aerial systems applications. Sensors, 22(2), 604. https://doi.org/10.3390/s22020604

[54] Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). Robotics: modelling, planning and control. Choice Reviews Online, 46(11), 46–6226. https://doi.org/10.5860/choice.46-6226

[55] Craig, J.J. (2005). Introduction to Robotics: Mechanics and Control. Pearson. (pp. 96-118)

[56] Common dataset formats used in target detection tasks (voc, coco, yolo). (2023, June 3). CSDN. https://blog.csdn.net/weixin_45277161/article/details/130331788

[57] Kaggle. (n.d.). *Efficient GPU usage* [Technical documentation]. Kaggle Documentation. https://www.kaggle.com/docs/efficient-gpu-usage

[58] FunHPC. (n.d.). *User Manual for High-Performance Computing Platform* [Technical documentation]. FunHPC Documentation. https://funhpc.com/#/documentation/UserManual

[59] [BPU Deployment Tutorial] An article to take you out of the model deployment novice village easily. (2022, August 6). CSDN. https://blog.csdn.net/Zhaoxi_Li/article/details/125516265?spm=1001.2014.3001.5502

[60] D-robotics Chao. (2024, October 11). D-Robotics X5 algorithm toolchain version released. D-Robotics Developer Community. https://developer.d-robotics.cc/forumDetail/251934919646096384