Industry Science and Engineering Vol. 3 No. 7, 2026

Research on Object Sorting Technology of Collaborative Robots Based on Machine Vision

Yingkai Ma, Liye Zhang
School of Computer Science and Technology, Shandong University of Technology, Zibo, Shandong, China

Abstract: Collaborative robot object sorting based on machine vision integrates multiple technologies and is highly valued in industrial production and logistics automation for improving efficiency, reducing labor costs and enabling precise operations. This thesis focuses on this technology and achieves key results: dual-depth perception cameras for object coordinate positioning, inter-thread communication between the robotic arm and camera main control unit, HTTP communication between the main controller and gripper, multi-threaded operation to alleviate program congestion, and real-time system status monitoring via flag bits. These achievements realize core functions and modularize the project through standardized communication interfaces, facilitating hardware replacement.

Keywords: Machine Vision; Inter-Thread Communication; Target Detection; Mechanical Arm

1. Introduction

OpenCV is an open-source computer vision and machine learning software library that supports functions such as image and video processing, feature detection, and object recognition, and is compatible with multiple languages like C++ and Python, as well as multiple platforms such as Windows and Linux. This project utilizes its image processing tools to format the images collected by the camera, providing a foundation for subsequent recognition and positioning.

Python's multithreading technology is implemented through the threading library for concurrent programming. It enables other tasks to be executed during I/O waiting periods, thereby enhancing the program's response speed and efficiency. Inter-thread communication uses shared variables and thread queues: Shared variables are synchronized for access through threading [1]. Lock to prevent data inconsistency; Thread queues serve as thread-safe buffers, suitable for the producer-consumer model, ensuring efficient data transmission.

Hand-eye calibration is used to determine the geometric relationship between the camera and the end effector of the robot, converting visual information into position data in the robot coordinate system, and is the key to precise operation [2]. This project adopts the 6-point method to calibrate the tool coordinate system: select fixed points in the robot space, set 6 calibration points by moving the tool in different postures, and determine the X, Y, and Z axis directions of the tool coordinate system based on the right-hand rule to ensure the matching of visual positioning and the coordinates of the robotic arm operation [3]. As shown in Figure 1.

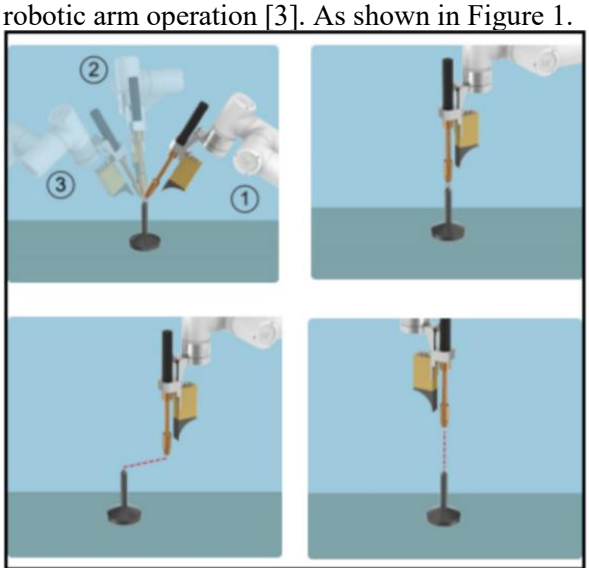

Figure 1. Schematic Diagram of the Calibration Process

2. Detailed Design

2.1 Project Structure

This project is mainly composed of five modules: the user module, the main control module, the gripper module, the robotic arm module, and the visual recognition module [4]. Their specific structure is shown in Figure 2.

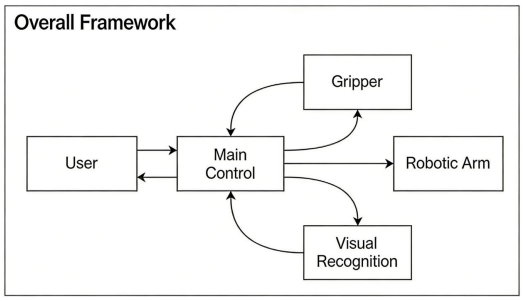

Figure 2. Project Structure Diagram

2.1.1 User interface

The user interface is built using the Tkinter library and consists of three buttons: Start, Stop, and Home. Their functions are to initialize the system, stop the system, and return to the original position, respectively. An image display area is provided, which will show the real-time image from the camera after clicking the Start button. A text output area is also included, which is used to display the system status and log information in real time [5]. As shown in Figure 3.

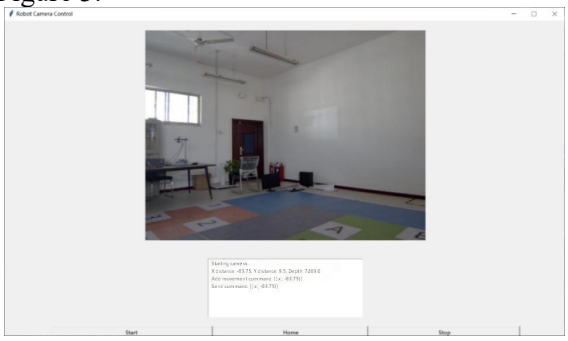

Figure 3. User Interface Diagram

2.1.2 Master control

The main control is the overall control part of the entire system. Each module is encapsulated as a class and instantiated and called in the main control, as shown in Figure 4.

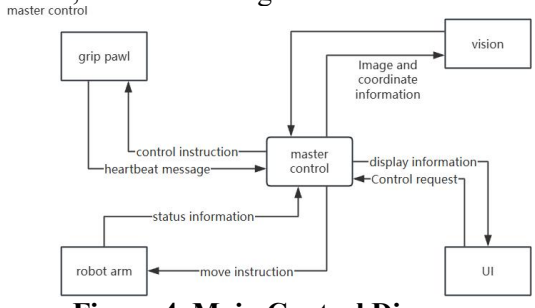

Figure 4. Main Control Diagram

The main controller is responsible for the connection and collaboration among all modules of the entire system, and it is the central part of the entire system. The main controller communicates with the gripper through HTTP requests [6]. During normal operation, the main controller will send HTTP requests to the gripper module at regular intervals to obtain the working status of the gripper module. If the gripper module responds to the main controller's request within the specified time, it can be considered that the gripper module is working normally; otherwise, the system will be paused and a prompt message will be displayed on the user interface. This design references the heartbeat detection design in microservice development, which can effectively improve the overall stability and fault tolerance of the system.

The main control module receives the item pixel coordinates processed by the Yolo v8 model in the visual module [7]. After correcting the parameters, it packages the coordinate array and depth data into a mechanical arm movement control command and sends it to the thread queue of the mechanical arm module.

The main controller is also responsible for transmitting the images obtained by the visual module to the user interface for display, and receiving control requests from the user interface module.

2.1.3 Claw module

The main components of the gripper module consist of an ESP32 microcontroller, a stepper motor driver, and a stepper motor [8]. As shown in Figure 5.

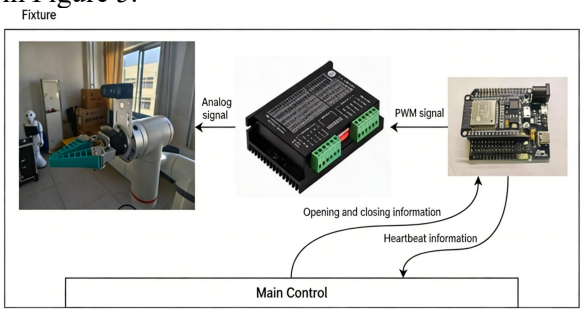

Figure 5. Claw Module Schematic Diagram

This module enables the ESP32 microcontroller to operate in the wireless terminal mode, making it a Web server. By configuring a static IP address, it can be reliably connected. By receiving HTTP requests, it controls the IO pins to achieve the function of controlling the opening and closing of the gripper [9].

The interface parameters in this module are shown in Table 1.

The control signal for the stepper motor is generated using the built-in timer of the ESP32. The PWM frequency is set to 1kHz with a duty cycle of 50%, and the current of the stepper motor driver is set to 1.5A with 32 subdivisions. The request from the main control can achieve

the opening and closing operation of the gripper at a specified angle by carrying different speed and waiting time parameters.

Table 1. Claw Module Interface and Parameters

interface	parameter	Description
/motor?directio	direction	Motor rotation direction
n=cw&speed=2	speed	Motor rotation speed
000&time=10	time	Waiting time

2.1.4 Mechanical arm module

The gripper module is composed of the FR5 robotic arm and the corresponding control code. The structure is shown in Figure 6.

The overall code of this module is encapsulated into the FRRobot class, which is instantiated and called in the main control [10]. When this module is instantiated, it automatically starts a new thread, configures the initial parameters of the robotic arm, establishes a RPC connection with the robotic arm, and initializes the thread queue and flags. The module controls the robotic arm's movement by reading the motion instructions stored in the thread queue [11]. The main control can directly drive the robotic arm to perform point motion or joint motion operations through the module interface, or implement combined operations through Lua scripts. The interface parameters are shown in Table 2.

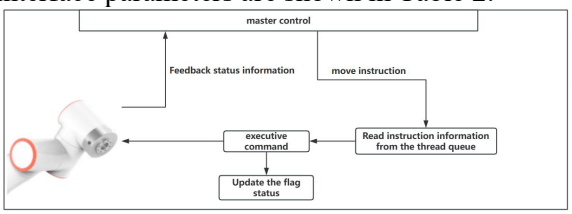

Figure 6. Schematic Diagram of the Mechanical Arm Module Structure

Table 2. Mechanical Arm Module Interface and Parameters

Interface	Parameter	Description
add_move_command	command	Add a movement command to the queue
add_catch_commands	command	Add a gripping command to the queue
run_script	script_name	Run the specified Lua script
move_x	distance	Maximum movement distance along the X-axis
	direction	Movement direction along the X-axis
move_y	distance	Maximum movement distance along the Y-axis
	direction	Movement direction along the Y-axis
move_z	distance	Maximum movement distance along the Z-axis
	direction	Movement direction along the Z-axis
open_hand	angle	Open the gripper at the specified angle
close_hand	angle	Close the gripper at the specified angle

To prevent issues such as repeated execution of instructions and interruption of instructions, several flag bits such as is_open, catch_moving, and is_arm_moving are set to indicate the current movement status of the robotic arm. Before sending the motion data to the robotic arm, this module will first check whether each flag meets the requirements. Only the states that meet the operation requirements of the instructions can be successfully sent and executed. Otherwise, the system will wait for the completion of the current operation before proceeding to the next step. The centering operation and the moving operation of placing the item at the designated position in this module are all operated by another written Lua script. By writing the joint coordinates and movement instructions in the Lua script, the atomicization of the actions is achieved, thereby preventing the actions from being accidentally interrupted and avoiding the problem of not being able to recover on its own. In the movement instructions of the robotic arm, in addition to the movement distance calculated by the main control module, there is also a soft limit composed of parameters such as initial speed configuration and movement distance calculation. The soft limit can effectively solve the problem of the robotic arm going out of bounds and ensure production safety.

2.1.5 Visual module

In the visual module, the hardware part uses the Gemini2 binocular structured light 3D camera. The module structure is shown in Figure 7.

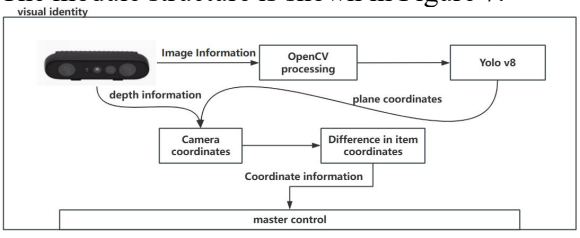

Figure 7. Schematic Diagram of the Visual Module Structure

After being instantiated, this module first initializes the camera parameters and then calls the _capture_loop method to start the data stream.

After obtaining the data frames (color and depth), it converts the color frame into an RGB image and updates the current image to the variable color_image, and updates the current depth information to the variable depth_intrinsics. The main control obtains the image and depth information by calling the get_color_image() and get_center_distance() methods instead of using a thread queue, thus ensuring the real-time nature of the data [12].

The interface parameters in this module are shown in Table 3.

Table 3. Visual Module Interface and Parameters

Interface	Parameter	Description
run	None	Starts the camera thread
stop	None	Terminates the camera thread
_capture_loop	None	Target function of the thread
process_depth_data	depth_frame	Processes depth data
get_color_image	None	Retrieves the RGB image
covert_pixel_to_carmera_coordinates	None	Converts pixel coordinates to coordinates in the camera coordinate system
get_center_distance	None	Retrieves the depth of the center point

The visual identification part of this project uses the Object Detection task mode in the Yolo v8 model. The performance parameters of the Yolo v8 series are shown in Table 4:

Table 4. Yolo v8 Series Performance Parameters

Model	Image Size	mAPval	Speed on A100 GPU (ms)	Params	FLOPs
Yolov8n	640	37.3	0.99	3.2	8.7
Yolov8s	640	44.9	1.20	11.2	28.6
Yolov8m	640	50.2	1.83	25.9	78.9
Yolov8l	640	52.9	2.39	43.7	165.2
Yolov8x	640	53.9	3.53	68.2	257.8

Here, mAPval represents the result obtained on the validation set (val2017) of the COCO dataset using a single model and a single scale [13]. The higher the value, the better the performance. The running speed of the A100 GPU refers to the average time required to run the model on the NVIDIA A100 GPU. The smaller the value, the better. Params indicates the number of parameters of the model, and the smaller the value, the more lightweight the model is. FLOP represents the number of floating-point operations of the model, which is used to measure the computational complexity of the model. The lower the FLOPs, the higher the computational efficiency of the model. Considering this project, Yolo v8n is used as the object recognition model.

2.2 Project Process Design

After the project is launched, the user interface will be initialized first. The visual and robotic arm threads will only be initialized and started after the user manually clicks the "Start" button.

The visual initialization operation involves instantiating the OrbbecCamera class, assigning values to the relevant parameters for the camera, calling its run method to start the camera thread, establishing a connection with the camera, and initiating the data stream, etc [14].

The initialization of the robotic arm involves instantiating the FRRobot class, setting the relevant RPC addresses, tools, workpiece coordinate system numbers, default speeds and other initialization parameters for the robotic arm, establishing a connection with the robotic arm's RPC, and performing other operations. After the robotic arm initialization is completed, the run_scrip method will be called to execute the toHome.lua script for the robotic arm's homing operation. Finally, the information of the user interface will be updated to inform the user that the initialization is complete. The initialization process is shown in Figure 8.

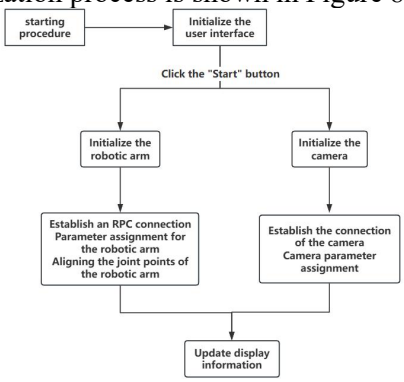

Figure 8. Update Display Information Project Initialization Flowchart

The visual recognition and object grasping process is shown in Figure 9.

The visual module continuously updates the images and depth information obtained from the camera. The main control obtains the images and depth information through the methods of get_color_image() and get_center_distance(), and compares the coordinates data obtained after object recognition by the Yolo v8 model with the preset range threshold. If the conditions for grabbing are met, a grabbing instruction will be added to the command queue of the mechanical arm module; otherwise, a moving instruction will be added. At the same time, the flag will be updated to prevent the repeated sending of instructions [15]. After the grabbing, a Lua script is called to move the object to the designated position, then open the gripper to place the object at the designated position [16]. After the placement instruction is executed, the return-to-home script will be automatically executed.

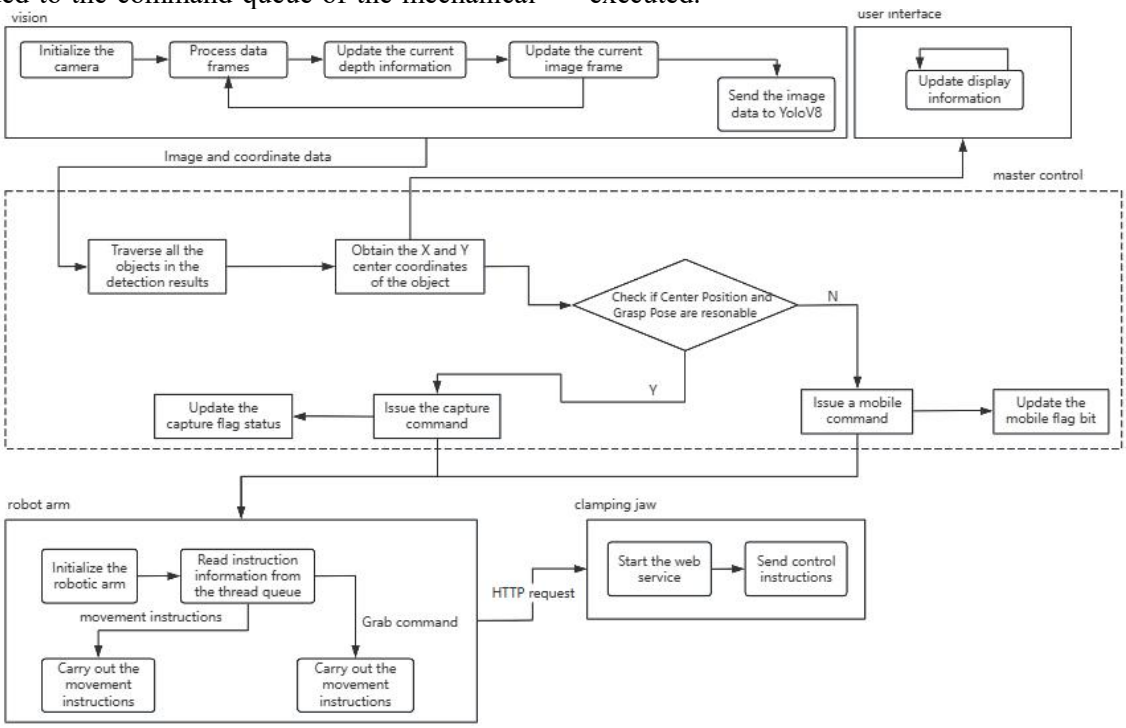

Figure 9. Flowchart of Visual Recognition and Object Grasping Process

3. Summary

This project has successfully achieved the core function of collaborative robot object sorting based on machine vision, and built a complete closed-loop workflow covering visual object recognition, spatial coordinate positioning, robotic arm grasping execution, target placement and automatic repositioning for cyclic operation. In practical scenarios, the system can continuously complete sorting tasks according to preset rules, effectively improving operation efficiency and reducing manual input in repetitive work. Through the overall modular design and unified standardized communication interfaces between functional modules, efficient data interaction and cooperative operation among all core components including the main control unit, visual recognition module, robotic arm and end gripper are fully realized. Meanwhile, the standardized interface design endows the system with excellent hardware replaceability — hardware components such as cameras and grippers of different models can be quickly replaced and adapted without modifying the core program framework, greatly reducing the cost of subsequent equipment upgrading and maintenance.

In terms of system reliability, mechanisms such as heartbeat detection and flag bit control are adopted to realize real-time monitoring of system operation status and thread working state. The heartbeat mechanism can timely perceive module disconnection or operation abnormalities, and flag bit control can flexibly schedule task execution and state switching, which effectively avoids program congestion and state confusion, and fully guarantees the operation stability and fault tolerance of the sorting system during long-term continuous operation.

However, the current system still has notable room for improvement in adaptability to complex industrial environments and algorithm generalization ability. Under harsh working

conditions such as uneven illumination, object occlusion, stacked workpieces and complex background interference, the recognition accuracy and positioning precision of the system will be affected to varying degrees. In addition, the system still relies on a fixed training dataset when facing new types of sorted objects, with insufficient rapid adaptation capacity. Therefore, there is still a certain gap between current system performance and the strict requirements of large-scale, high-intensity industrial batch production applications.

Future research will focus on the following directions to further optimize the system.

In terms of technical optimization, more advanced deep learning detection algorithms and high-precision kinematics calculation algorithms will be introduced. The scale and diversity of the training dataset will be expanded by adding samples of different object categories, surface materials and complex scene conditions, so as to comprehensively enhance the system's recognition accuracy and environmental adaptability.

In terms of system expansion, a multi-robot collaborative sorting system will be constructed based on the ROS system, targeting key technical issues including multi-robot path planning, dynamic task allocation and efficient inter-robot communication protocols, to unleash the efficiency advantage of multi-machine cooperation.

In terms of intelligent upgrade, the robot will be empowered with rapid learning and transfer learning capabilities, so that it can quickly adapt to new sorting objects with limited sample data. It will also optimize sorting strategies through an autonomous learning mechanism, dynamically adjust the grasping sequence, and reduce the demand for manual intervention and parameter debugging.

In terms of platform-based development, an open intelligent system and supporting cloud service platform will be built to support centralized management, real-time monitoring and unified task scheduling of distributed robot networks, so as to better meet the production requirements of large-scale industrial scenarios.

References

[1] Chen, J. F., & Zheng, M. H. (2022). A review of robot manipulation behavior research based on deep reinforcement learning. Robot, 44

[2] Zhu, C. L. (1987). Development, methods and applications of machine vision. Aerospace Precision Mechanical Engineering, (1), 62-67.

[3] Craig, J. J. (2018). Introduction to Robotics (4th ed.). Beijing: China Machine Press.

[4] Xu, H. W., & Gao, J. L. (2022). ROS2 Robot Programming in Practice: Based on Modern C++ and Python3. Beijing: China Machine Press.

[5] Ding, H. R., Deng, E. P., Zhou, W. F., Yu, Z. M., Liu, T. H., & Li, S. Y. (2024). Research on dynamic calibration method of dual-arm collaborative robot based on binocular stereo vision. Robot Technology and Application, (1), 22-27.

[6] Cheng, P. F., Liu, M. T., & Sun, K. (2024). Research on dynamic target grasping control method of food collaborative robot. Food and Machinery, 40(1), 95-100.

[7] Shao, X. J., Huang, J. C., Zhao, L. C., & Jin, M. S. (2013). Research on multi-stepping motor control based on new acceleration and deceleration curve. Automation & Instrumentation, 28(4), 53-56.

[8] Ge, S. H. (2024). Research on key technologies of test tube sorting robot for emergency and clinical scenarios. Zhejiang University of Science and Technology.

[9] Li, C. M. (2017). Research on target recognition and grasping positioning based on machine vision and deep learning. North University of China, Shanxi Province.

[10] Liu, C. (2023). Discussion on sorting technology of industrial robots based on machine vision. Popular Standardization, 23(22), 42-44.

[11] Zhang, P. W., & Li, X. W. (2023). Multi-parameter mining control technology for sorting robot grasping process under contact state perception. Machine Design and Research, 39(5), 24-28.

[12] Szeliski, Richard. Computer Vision: Algorithms and Applications. Springer, 2010.

[13] Forsyth, David A., and Ponce, Jean. Computer Vision: A Modern Approach. Prentice Hall, 2002.Szeliski, Richard. Computer Vision: Algorithms and Applications. Springer, 2010.

[14] Hartley, Richard, and Zisserman, Andrew. Multiple View Geometry in Computer Vision. Cambridge University Press, 2004.

[15] Bradski, Gary, and Kaehler, Adrian.

Learning OpenCV: Computer Vision in C++ with the OpenCV Library. O'Reilly Media, 2008.Bradski, Gary, and Kaehler, Adrian. Learning OpenCV: Computer Vision in C++ with the OpenCV Library. O'Reilly Media, 2008.

[16] Chaoyue Sun, Yajun Chen, Xiaoyang Qiu, Rongzhen Li, Longxiang You. MRD-YOLO: A Multispectral Object Detection Algorithm for Complex Road Scenes. Sensors. Volume 24, Issue 10.