Robotic Arm Development Log

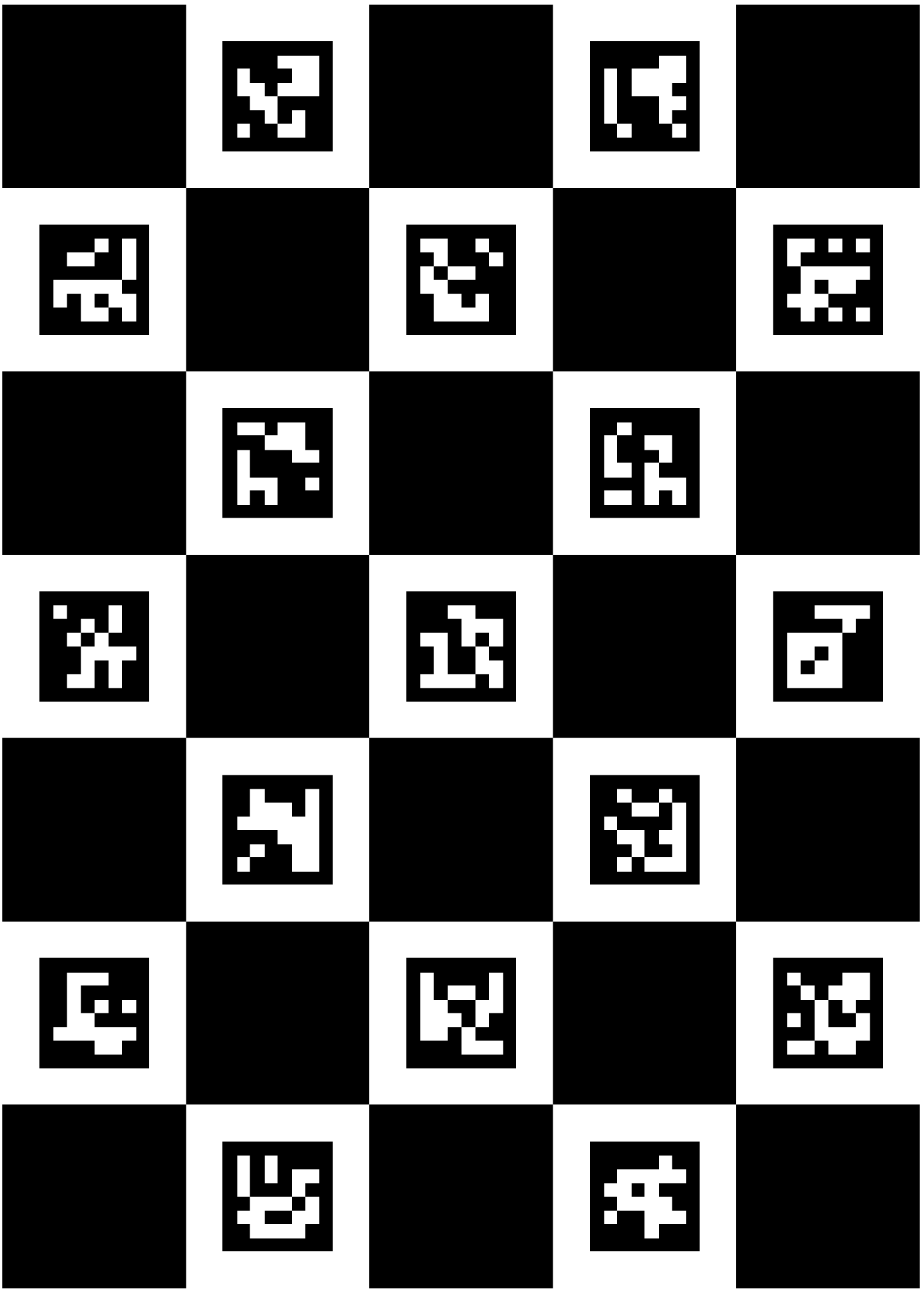
07/13

- Preliminary research on available robotic arm grasping control programs
    - AnyGrasp SDK: Requires a license application
    - Graspnet-1Billion: Can be deployed locally
- Planned the minimal viable solution
    - Use YOLO to extract the center point, and Kinpy to calculate the grasping position
    - The specific YOLO algorithm needs to be changed based on the object being grasped
- Potential issues
    - Using a 2D camera will increase the algorithm's difficulty
    - Object recognition may require special algorithms
    - Overlapping or side-by-side placement of objects will increase the grasping difficulty
    - Camera angle and position require extensive debugging
- Determined the development workflow
    - Familiarize with the robotic arm SDK/interfaces
        - Data interfaces
        - Operation interfaces
    - Perform hand-eye calibration
        - Can use OpenCV's built-in programs
    - Deploy the grasping program
        - Grasping program processing must run on the server side
        - Commands are sent via the robot SDK/API
    - Fully test the grasping functionality
        - User command -> Recognition -> Generate command -> Execute command

07/14

- Determined the usage method for Kinpy
    - Construct the robot structure file (URDF or other formats)
    - Set maximum/minimum angles
    - Generate commands using the correct coordinate system

- Generated the calibration board required for calibration



07/15

- Reinstalled the old robotic arm chassis
- Connected the power supply and display screen to the robotic arm and vacuum pump

- Attempted to resolve the display screen issue
    - Used mystudio for burning
    - The serial port driver could not be downloaded correctly
    - The Raspberry Pi could not be recognized
    - It is highly likely that an SD card failure prevented the Raspberry Pi from booting

07/16

- Flashed the latest Ubuntu OS for the robot onto the Raspberry Pi
- Flashed the latest Atom software onto the robotic arm
- Performed a robot self-check
    - Issues occurred during the self-check; joints could not move
    - The Atom display color responded normally
- Successfully achieved robot control using a demo program
    - The baud rate must be 1000000 instead of 115600
- Successfully installed the AgileX robotic arm
- Achieved control of the robot and gripper using the operation software
- Successfully obtained robot information using the SDK
    - Because the old piper_sdk is incompatible with Windows, pyAgxArm must be used for interaction
    - The CAN interface can only support one connection at a time
- Attempted to connect the Intel RealSense camera
    - Successfully connected after using a new USB-Type C adapter cable
    - Downloaded the new firmware

07/17

- Connected the camera using the RealSense Python software library
    - The data cable needs to be manually adjusted during connection
- Attempted to operate the robot using the SDK
    - Successfully achieved movement using flange values
- Attempted to perform hand-eye calibration
    - The matrix needs to be inverted to use the OpenCV hand-eye calibration function
    - Precision is still not high; significant deviations in roll, yaw, and pitch
    - Must pay attention to robot joint position limits when generating coordinates
    - Changed the plan to first perform hand-eye calibration without the gripper

07/20

- Successfully performed hand-eye calibration
    - Must pay attention to the difference between extrinsic and intrinsic parameters
    - Image capture positions must vary significantly
- Operated the gripper using the SDK
- Attempted the grasping function
    - Because the camera is fixed, recognizing distant objects is relatively difficult
    - The robot's intrinsic joint angle settings prevent the robotic arm from performing a parallel grasp
    - The robot SDK cannot wait for tasks to finish, requiring a manual timer wait
- https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/calibrate_v3.py

07/21

- Fully implemented the process of grasping after recognizing an ArUco Marker
  - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/execute.py
- Successfully deployed the object recognition algorithm
- The grasping position can be roughly determined based on known object sizes
  - The object recognition algorithm has limited precision at low resolutions and for small objects
  - Location recognition based on annotations is not as accurate as the ArUco Marker
- Initially implemented object grasping without an ArUco Marker
  - The accuracy rate is relatively low
  - The accuracy rate can be increased later by installing a camera on the robotic arm
  - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/detection_execute.py
- Attempted camera settings with a wider viewing angle and closer distance
  - Grasping based on object recognition still has significant errors
  - ~~Possible reasons include calibration errors, robot joint precision issues, and camera and robotic arm position shifts~~
  - Verified that the error increases the further the object is from the camera
  - Highly likely to be a calibration error or the camera distance is still too far

07/22

- Successfully installed the camera bracket and camera on the robotic arm
- Performed hand-eye calibration
  - Initial calibration was unsuccessful; later resolved by changing from_euler("XYZ") to from_euler("xyz")
  - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/eye_in_hand.py
- Completed a Demo for grasping based on an ArUco Marker
  - Currently, the grasping angle still depends on the ArUco Marker's orientation; orientation markers can be used in the future to indicate the grasping direction
  - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/eye_in_hand_execute.py
- Initially implemented a Demo for grasping based on object recognition
  - Recognition may fail or cause significant errors when objects are too far or too close
  - The specific grasping posture of the robotic arm requires manual adjustment
  - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/eye_in_hand_detect_execute.py
- Attempted to fuse RGB and depth cameras
  - Depth camera readings drop directly to zero when the distance is less than 15 centimeters
  - The effective reading distance is limited

07/23

- Successfully achieved grasping without object size settings based on distance detection and visual recognition
  - The gripper obstructs part of the vision
  - Cannot obtain the distance of objects that are too far or too close

    - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/eye_in_hand_depth_display.py
- Added a process to automatically recognize the object with the largest area
- Attempted a grasping program based on YOLO World
    - The grasping angle cannot be identified due to the lack of a bounding box
- Switched to using YOLOV8N-SEG + OpenCV
    - Similarly unable to correctly confirm the grasping angle
- Created object coordinates directly using depth readings after using YOLO
    - Successfully achieved size-free calibration grasping based on YOLO
    - Recognition errors increased significantly compared to the SDK
    - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/yolo_eye_in_hand_depth_display.py
- Follow-up plans
    - [ ] ~~Find a vision model capable of recognizing specific product brands~~
    - [x] Simulate a shelf scenario for grasping
    - [x] Test grasping at larger distances and angles

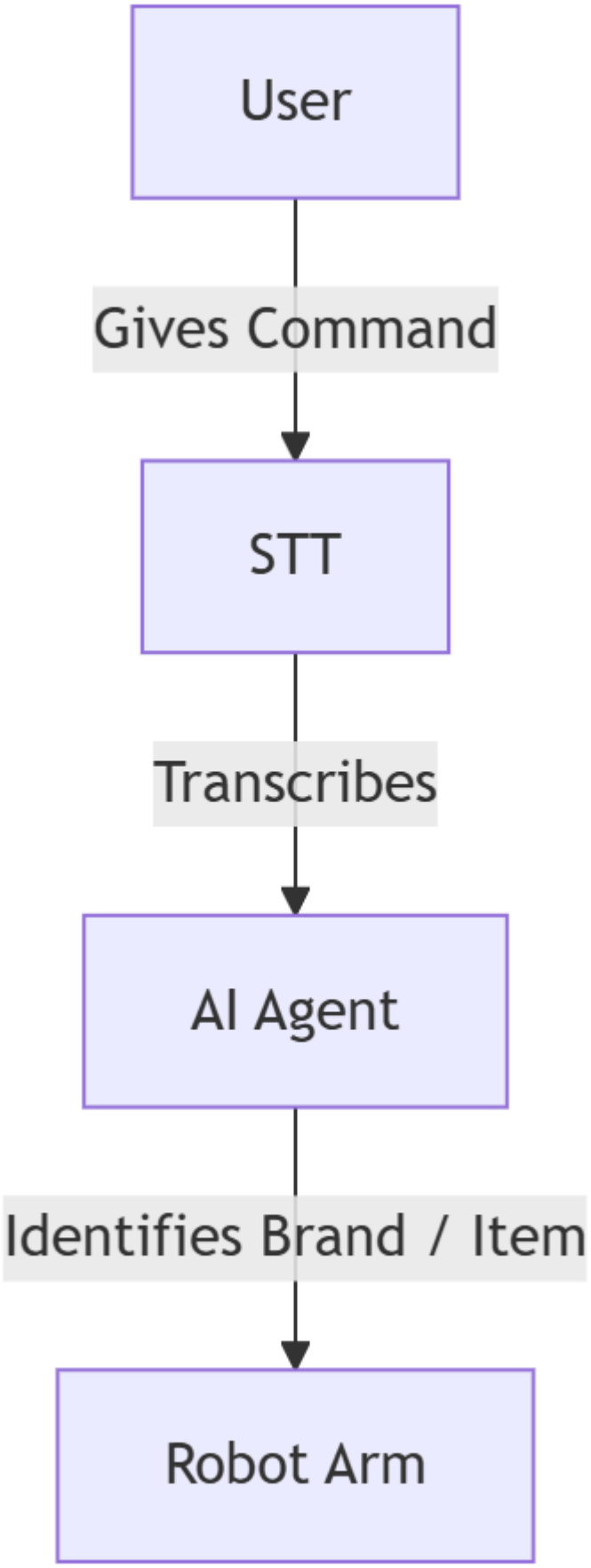
07/24

- Attempted to use the company's SDK vision model for image recognition
    - Unable to accurately return object coordinates
- Most vision models cannot output accurate brand recognition and object coordinates simultaneously
    - Future use should directly integrate the company's vision model
- Achieved grasping at further distances and larger angles using two-stage recognition
    - Secondary target recognition is performed after the robot approaches the target
    - Grasping is performed directly if the target is close enough
    - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/look_twice.py
- YOLOV8 still has recognition issues in complex situations
    - Changed the model size from Nano to Small; performance showed no significant changes
- Attempted to deploy the open-source VLA model
    - The OpenVLA primary training set uses overhead cameras; deploying it on an eye-in-hand setup requires fine-tuning the model yourself
    - The LingBot-VLA-V2 training set also uses a fixed perspective; deployment requires post-training
- Searched for other eye-in-hand based models
    - https://alibaba-damo-academy.github.io/CamVLA/ (Code not yet published)

07/25

- Reviewed the Patch Policy architecture and usage methods
    - The source code has not yet been published; requires development after future publication
- Fully integrated the company's SDK into the two-stage recognition grasping
    - There is still room for improvement in the success rate
        - Depth camera readings are sometimes still inaccurate (possibly due to being too close)
        - In some recognitions, excessive errors lead to unsuccessful grasping
        - The robotic arm gripper does not open automatically in subsequent grasping attempts

- Follow-up plans
  - [x] Add a grasping function based on specific item/brand types
  - [x] Integrate robotic arm actions into the human-machine interaction loop (STT portion can be temporarily skipped)
  - [x] Modularize and encapsulate the code



07/28

- Integrated the company's brand recognition SDK into the grasping program
  - Sometimes recognition results may have deviations, with a probability of causing the grasping program to not execute
- Used the LLM SDK for brand keyword extraction and brand-based grasping
  - The LLM sometimes responds slowly, requiring a manual wait
  - https://git.aimall-tech.com/shengzhen.wang/robot-arm-research/-/blob/main/box_look_twice.py

07/29

- Modularized the code
  - constants.py
    - Stores various constants in the project
  - vision_models.py
    - Initializes recognition models
  - llm.py
    - Initializes the LLM API
  - vision.py
    - Functions for visual processing
  - main.py
    - The main program