NeuroGrip: Autonomous Reshelving using Vision Language Models

Yu-Hsin (Thomas) Chan[1], Boxiang (William) Fu[2], Joshua Pen[3], and Jet Situ[4]

***Abstract*— NeuroGrip is a robotic reshelving system that integrates Vision-Language Models (VLMs) to enable adaptive pick-and-place operations. Traditional warehouse robotics rely on fiducial markers and fixed structures, limiting flexibility. NeuroGrip utilizes a Franka Panda manipulator and a RealSense depth camera to interpret visual and language-based inputs for object handling. This paper reviews related work in industrial pick-and-place automation, including chain-of-thought (CoT) reasoning and vision-language-action (VLA) models, and provides a detailed overview of NeuroGrip's hardware setup. Our MVP implementation can be found here.**

I. INTRODUCTION

NeuroGrip is designed to improve robotic reshelving by reducing reliance on structured environments and predefined object locations. Traditional pick-and-place systems depend on fiducial markers, prepositioned shelving, and rigid localization techniques, limiting adaptability. By integrating Vision-Language Models (VLMs), NeuroGrip processes visual and language-based inputs to handle objects more flexibly. This paper examines related work in pick-and-place automation and vision-language integration, focusing on NeuroGrip's hardware setup, which includes a Franka Panda manipulator, a RealSense depth camera, a structured prop setup consisting of a pickup zone and shelf, and a prompt-to-location system for translating user commands into actionable placements.

II. RELATED WORK

A. Industrial Pick and Place

A primary motivation for pick and place behavior in our VLM context is the use within industrial warehouse robotics, referred to as material handling. Current solutions have a strong focus on prior organization, specialized labeling, which can then be fed into a simplified ArTag or AprilTag model that provides localization for mobile robots. In this variation, a Universal Robotics 5 cobot, a robust 6DOF mounted robot has rapidly become an industry and scientific standard, and in many cases, have been attached to a Ridgeback robot, enabling it to move autonomously in pick and place environments.

However, the existing pick and place space is strongly reliant on prepositioning components for the robots to work, including guidelines, markers, signal extenders, shelving design, walkway and railway design, etc. While effective, it results in a highly static and maladaptive configuration, losing all versatility and cross-applicability.

[1]yuhsinch@andrew.cmu.edu
[2]boxiangf@andrew.cmu.edu
[3]jpen@andrew.cmu.edu
[4]jets@andrew.cmu.edu

B. Chain-Of-Thought (CoT) Vision Language Models

A similar project to ours was published late last year by Wang and Zhang et. al [6], in which they proposed the SeeDo framework (Figure 1), whereby they provide their own methodology for the translation of VLM output into robot-parsable language.

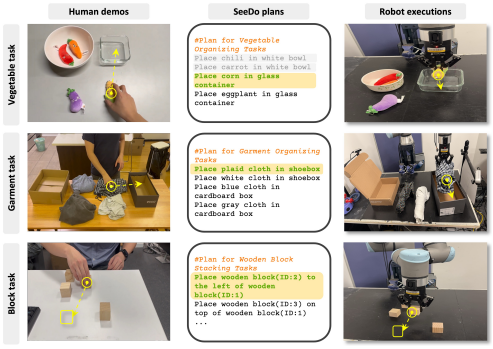

Fig. 1. SeeDo framework, as implemented by Wang and Zhang

Their variation is moderately more complex, utilizing chain-of-thought reasoning to decide what and where to perform object segmentation on a wide class of objects, thus performing generalizable tasks. From there, the CoT pipeline outputs predetermined positions, from which they can execute it with the robot arm, similar to our method of prepositioning the shelving unit and our existing calibration methods.

C. Vision-Language-Action (VLA) Models

More recent advances also propose the use of vision-language-action models. Such models forgo the translation into robot-parsable language entirely, and instead utilize an end-to-end learning model that takes in user prompts and directly outputs manipulator commands. Similar projects include the OpenVLA framework (Figure 2), TinyVLA framework, Google RT framework, and the OmniManip framework [7, 4, 1, 2].

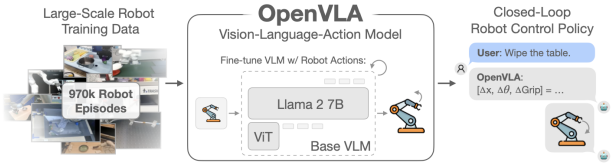

Fig. 2. OpenVLA framework

This type of model greatly generalizes the task space and can be used for a variety of end-to-end manipulation

tasks. However, the compute requirements are far higher compared to traditional methods, often requiring more than 16 gigabytes of RAM and recent NVIDIA GPUs and CUDA drivers.

III. METHODOLOGY

Our methodology involves using a prompt-to-text Large Language Model to convert the user input prompt to a pick-up item and drop-off location. Simultaneously, the RealSense on the end-effector takes a screenshot of the pickup zone. Based on the input prompt, the center pixel location of the item is calculated in the pickup zone. This is converted to an end-effector location through camera calibration. The end-effector picks up the item and moves to the drop-off location specified. Finally, the robot arm moves to its reset position ready to take another user input.

A. Hardware Setup

The entire hardware setup of our project is depicted in Figure 3. The setup consists of a Franka manipulator, shelf, pickup zone, RealSense stereo camera, and an off-location PC that runs the prompt-to-location software.

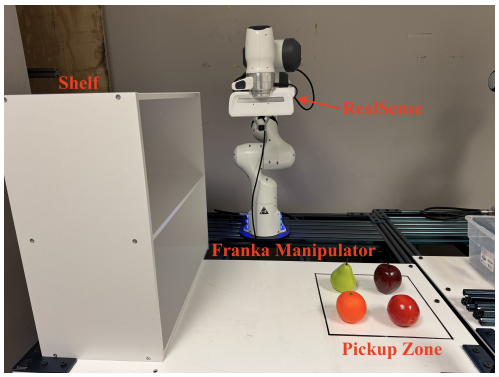

Fig. 3. Hardware setup of NeuroGrip

1) Franka Manipulator Setup: The manipulator used is a Franka Panda manipulator made available to us in the REL lab. In particular, we were allocated to the `iam-sneezy` manipulator. The robot arm has 7 degrees-of-freedom and can grab a payload of 3 kilograms. The hardware-software interface between the Franka manipulator and the control PC was already set up beforehand. The documentation for the interface setup can be found in Reference [8].

2) Shelf Setup: The shelf is placed in the left section of the world-frame table. We split the shelf into four quadrants (top-left, top-right, bottom-left, and bottom-right) for the end-effector to place items in. If the user inputs a prompt outside these four quadrants, the input will be invalidated and require the user to input a new prompt.

3) Pickup Zone Setup: The pickup zone is placed in the right section of the world-frame table. This is the location where the end-effector will attempt to pick up items and place them on the shelf. We have used dummy fruits for our setup, although the model is much more general and can recognize any object in the SAM2 database [5].

4) RealSense Setup: We installed a RealSense depth camera behind the Franka robot arm's end-effector and used it to capture images of the picking area. Figure 4 shows a clearer view of the RealSense mount location.

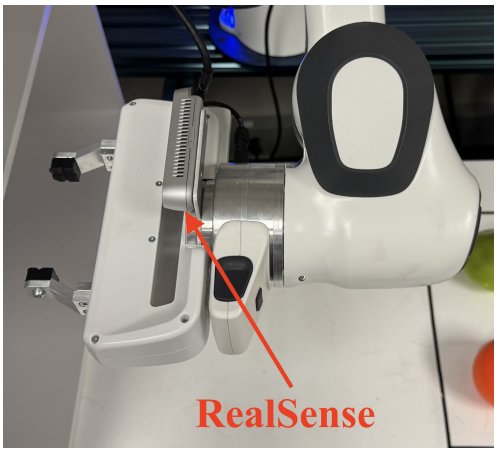

Fig. 4. RealSense setup of NeuroGrip

5) Prompt to Location Setup: Since we intend to run a visual language model in our project, we require a PC with a GPU. We discovered that the PC in the lab has an RTX3060Ti GPU installed, but the CUDA version is outdated for the model we want to run. While a software update could potentially resolve this issue, we were concerned that it might disrupt the entire environment. Consequently, we decided to deploy the model on Thomas' PC. Once a RealSense camera captures an image, we will send a request to Thomas' PC, which will process the image and provide the location of the target.

B. User Prompt Input Subsystem

Our implementation uses OpenAI's GPT-4o mini API for the user prompt input subsystem. Once `main.py` is ran, terminal queries the user to enter a command. Once the command is entered and `Return` is pressed, a server query request is sent to GPT-4o mini's API. Prompt engineering is used to specify the model to take in a natural language sentence and return a dictionary with two elements (Figure 5). The first element specifies the `object` to pick up, and the second element specifies the `destination` to put down. The `object`'s value is very generic, and our setup permits the use of any reasonably sized object below 3 kilograms that exists in the SAM2 dataset [5]. The `destination`'s value permits only four values corresponding to the quadrant on the shelf. They are:

- **Top-left**: 0
- **Top-right**: 1
- **Bottom-left**: 2
- **Bottom-right**: 3

C. Perception and Segmentation Subsystem

The perception and segmentation subsystem segments the target object in the image and determines its position in the

```
def parse_sentence(self, sentence):
    prompt = f"""
    Extract the object and destination from the given command.
    Return a JSON object with "object" and "destination" fields.

    We have 4 possible destination, top-left, top-right, bottom-left, bottom-right.
    the number of top-left is 0, top-right is 1, bottom-left is 2, bottom-right is 3.

    Example:
    Input: "Pick the apple to the top-left corner."
    Output: {{"object": "apple.", "destination": 0}}

    Now, process this sentence:
    Input: "{sentence}"
    Output:
    """
```

Fig. 5. GPT prompt specification

global frame. This process involves drawing a fixed picking zone to place objects, ensuring that the Franka arm is reset to a fixed position so that the RealSense camera can capture an image of the entire area. After camera calibration, if we know the target's position in the 2D image, we can determine its position in the world frame through simple interpolation. The entire subsystem pipeline is depicted in Figure 6.

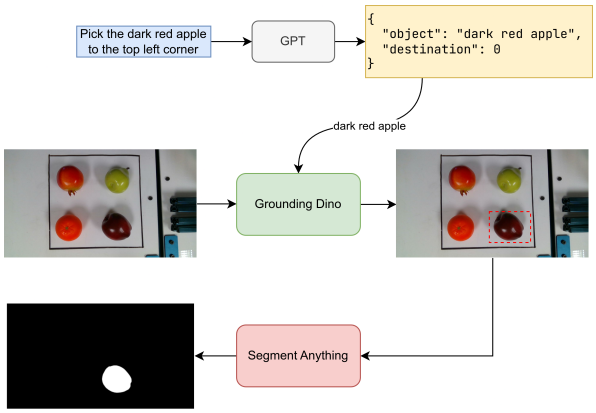

Fig. 6. Segmentation subsystem pipeline

1) Prompt to Bounding Box: Before segmenting the image, we need to obtain the bounding box of a target object. In this study, we opted to use Grounding DINO to achieve this [3]. Grounding DINO demonstrates impressive zero-shot capabilities, effectively identifying targets when provided with the correct prompt.

2) Bonding Box to Segmentation: After getting a bounding box is get, we use Segment Anything 2 to get a mask of the object [5].

3) Mask to Global Position: After we place the mask on the object, we can determine its center in the image by simply averaging the coordinates of its mask. However, this position in the image coordinate alone is insufficient for the arm to move to the target's position in the real world. Therefore, we need to perform a transformation from the image coordinate system to the world coordinate system. To achieve this, we use three points: the top-left, top-right, and bottom-left corners of the picking area. We obtain the coordinates of these three points in both the image and global coordinate systems. For future masks, we can directly use their coordinates to interpolate with the three reference points, thereby obtaining the global coordinate of the object. The procedure is shown in Figure 7. Note that we only need three reference points as our world frame is guaranteed to be parallel to the image frame.

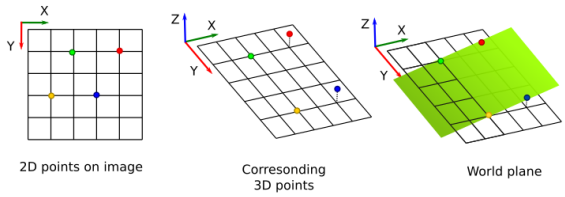

Fig. 7. Pixel coordinate to world frame transformation

D. Manipulator Control Subsystem

A finite state machine (FSM) is initialized to track the progression of the manipulator control task. The progression is a set of Cartesian way-points, and depends on the user prompt input. The manipulator control to reach these Cartesian way-points uses the `FrankaArm.goto_pose` method provided by the `Frankapy` package [8]. In essence, we transform the desired goal position into a homogeneous transform matrix and pass this as an argument to the method. The states of the FSM is discussed below in turn:

1) RESET state (Figure 8): This is the state the manipulator will move to once `main.py` is ran. This state is approximately 30 centimeters directly above the center of the pickup zone. Once the manipulator reaches this position, the terminal then queries the user for a command input. When a query is received and the valid parsed output is returned from OpenAI's API, the FSM transitions onto the next state.

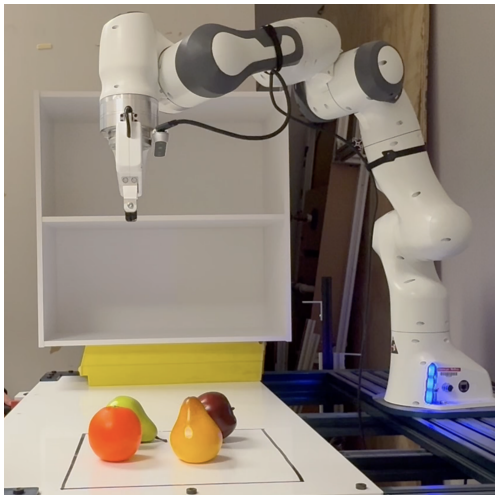

Fig. 8. RESET state

The reason we set this as the reset position is because it provides a direct view for the RealSense camera to view the pickup zone below. This allows us to calibrate the Cartesian x- and y-coordinates of any pixel in the picture taken by the RealSense camera. If at any time a state transition could not be completed, the manipulator returns to the RESET position and asks the user for a new prompt input.

2) PRE_PICKUP state (Figure 9): Once the Cartesian x- and y-coordinates of the desired pickup item is obtained from the perception & segmentation subsystem, the end-effector moves to approximately 10 centimeters above the center of the desired pickup item. This state acts as an intermediate step to prevent the end-effector from colliding with the item. Once this state is successfully reached, PICKUP is initiated.

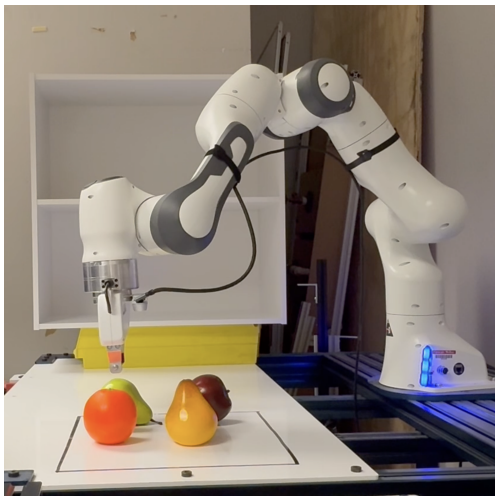

Fig. 9. PRE_PICKUP state

3) PICKUP state (Figure 10): This state attempts to pick up the item by bringing down the end-effector and closing the gripper. Once this is successful, the manipulator is returned to the RESET pose as an intermediate pose. Note that we do not return to the RESET state, but rather only its pose. The reason for this extra step is to prevent cumulative errors from stacking up. Once the RESET pose is reached, the FSM initiates a state transition.

4) PRE_DROPOFF state (Figure 11): Depending on the `destination` value from the API callback, the manipulator will move to one of the four predetermined poses that correspond to the desired drop-off quadrant on the shelf. Each pose is approximately 5 centimeters in-front of the shelf to prevent colliding with the shelf itself. The end-effector is also rotated so that the gripper is horizontal (see Figure 4). This is to allow the end-effector to move further into the shelf while holding an object without colliding with the shelf walls.

5) DROPOFF state (Figure 12): Finally, once the PRE_DROPOFF location is reached, the state transitions to initiate DROPOFF. During this state, the end-effector is extended 15 centimeters linearly into the shelf. Once this is reached, the gripper is opened and the object is dropped onto the shelf. The manipulator then reverses its extension

Fig. 10. PICKUP state

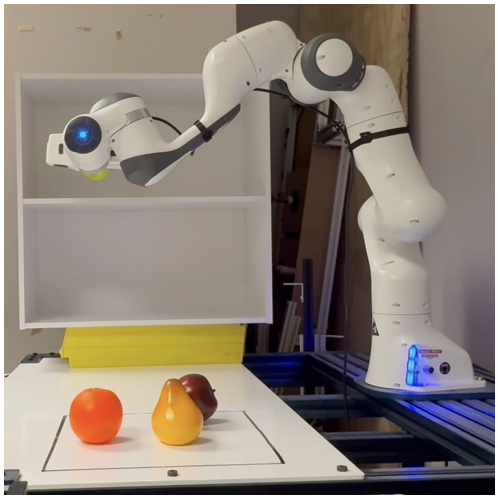

Fig. 11. PRE_DROPOFF state

and returns to the PRE_DROPOFF pose as an intermediate step. It then returns to the RESET state and pose ready to take in another user command.

IV. IMPLEMENTATION

Our MVP implementation can be found here.

V. REFERENCE

[1] Anthony Brohan et al. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*. 2023. arXiv: 2307.15818 [cs.RO]. URL: https://arxiv.org/abs/2307.15818.

[2] Moo Jin Kim et al. *OpenVLA: An Open-Source Vision-Language-Action Model*. 2024. arXiv: 2406.09246 [cs.RO]. URL: https://arxiv.org/abs/2406.09246.

[3] Shilong Liu et al. *Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection*. 2024. arXiv: 2303.05499 [cs.CV]. URL: https://arxiv.org/abs/2303.05499.

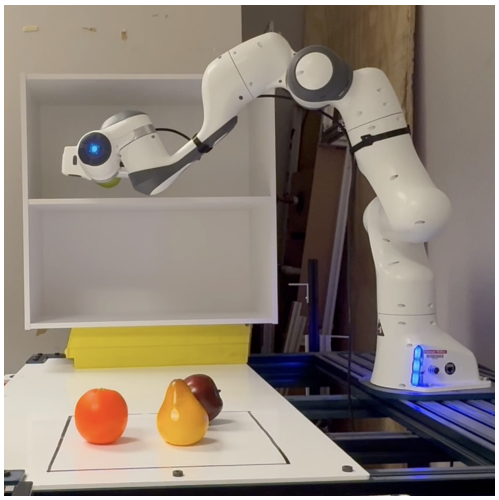

Fig. 12. DROPOFF state

[4] Mingjie Pan et al. *OmniManip: Towards General Robotic Manipulation via Object-Centric Interaction Primitives as Spatial Constraints*. 2025. arXiv: 2501.03841 [cs.RO]. URL: https://arxiv.org/abs/2501.03841.

[5] Nikhila Ravi et al. "SAM 2: Segment Anything in Images and Videos". In: *arXiv preprint arXiv:2408.00714* (2024). URL: https://arxiv.org/abs/2408.00714.

[6] Beichen Wang et al. *VLM See, Robot Do: Human Demo Video to Robot Action Plan via Vision Language Model*. 2024. arXiv: 2410.08792 [cs.RO]. URL: https://arxiv.org/abs/2410.08792.

[7] Junjie Wen et al. *TinyVLA: Towards Fast, Data-Efficient Vision-Language-Action Models for Robotic Manipulation*. 2024. arXiv: 2409.12514 [cs.RO]. URL: https://arxiv.org/abs/2409.12514.

[8] Kevin Zhang et al. "A modular robotic arm control stack for research: Franka-interface and frankapy". In: *arXiv preprint arXiv:2011.02398* (2020).