I Can't Do It
Talking To Robots 11-851

Liwei Yang **Abhishek Warrier** **Kensuke Nakamura** **Pratik Satija** **Ashish Marisetty**
{liweiy, abhishew, kensuken, psatija, amariset}@andrew.cmu.edu

1 Introduction

For many tasks, one arm is not enough. Grabbing an object out of reach, opening water bottles, and carrying large objects are all problems that benefit from bimanual manipulation. Although an object may be out of reach by a single arm, using a different manipulator to pass the object extends the effective reach of the robotic system and bypasses the physical limitations of a single arm. However, a key challenge arises when introducing an additional manipulator to the robotic system: coordination. The additional arm may add redundancies for how a system may achieve a task, which creates additional modes of potential robot behavior (e.g., in a shared workspace, either arm can pick up an object). Navigating the high-level ambiguity, in addition to the heterogeneous physical capabilities of each robot (different operatable workspaces, different grippers, etc.), causes a core issue in the control of the robotic system. In this project, we seek to tackle this coordination issue using state-of-the-art large language models (LLMs).

2 Related Work

2.1 Language-guided single-arm manipulation

Our project extends the Code-as-policies [1] approach for bimanual manipulation. Code-as-policies were originally instantiated on single-arm manipulation systems, where an LLM leveraged predefined APIs to synthesize sophisticated behavior for robot control. This framework leverages the open-vocabulary understanding of LLMs for high-level reasoning while taking advantage of mature frameworks for low-level control. Alternative paradigms for language-enable robotic control include language-conditioned imitation learning [2], or reinforcement learning [3]. However, these paradigms either require a large amount of demonstration data or managing the sim2real gap when transferring the learned policies to hardware. In contrast, code-as-policies generates code that calls low-level functions for the robot and can be done without fine-tuning the LLM as long as function names are self-explanatory or documentation is provided.

2.2 Language-guided bimanual manipulation

Using an LLM to coordinate bimanual robotic systems has recently gained popularity [4, 5, 6, 7]. Most similar to our work is that of Zhao et al. [4], which uses multiple language model agents to coordinate a bimanual robotic system and generate dynamically feasible waypoints for each arm to follow. However, this work relies on known perception and is brittle to noise arising from real-world sensing. In [5], the authors use an LLM in a Code-as-policies [1] framework with point-cloud conditioned grasping policies learned in simulation along with trajectory optimization for motion planning. [6] offers recommendations for prompt engineering for manipulation utilizing code-as-policies and applies their method to bimanual robotic systems. Our project also follows the code-as-policies framework, however our skills are all obtained via trajectory optimization rather than learning in simulation or imitation learning [7]. Our project seeks to optimize the performance of [4] based on our experiences with the open-sourced codebase. We found that running the pre-existing code led to an exorbitant number of API calls due to the dialectic nature of their system where the LLM was responsible for both generating waypoints and ensuring dynamic feasibility.

By delegating this responsibility to MoveIt2, we believed our project would allow the LLM to focus solely on the high-level behavior.

3 System

The block diagram of the whole system is shown in Figure 1. The user prompt will be combined with the task planning prompt and sent to the task planning LLM. The task planning LLM will output plans for both robot arms. For robot interfaces, aside from the parameters being swapped and specific skills, the motion planning software stack for both robot arms is identical.

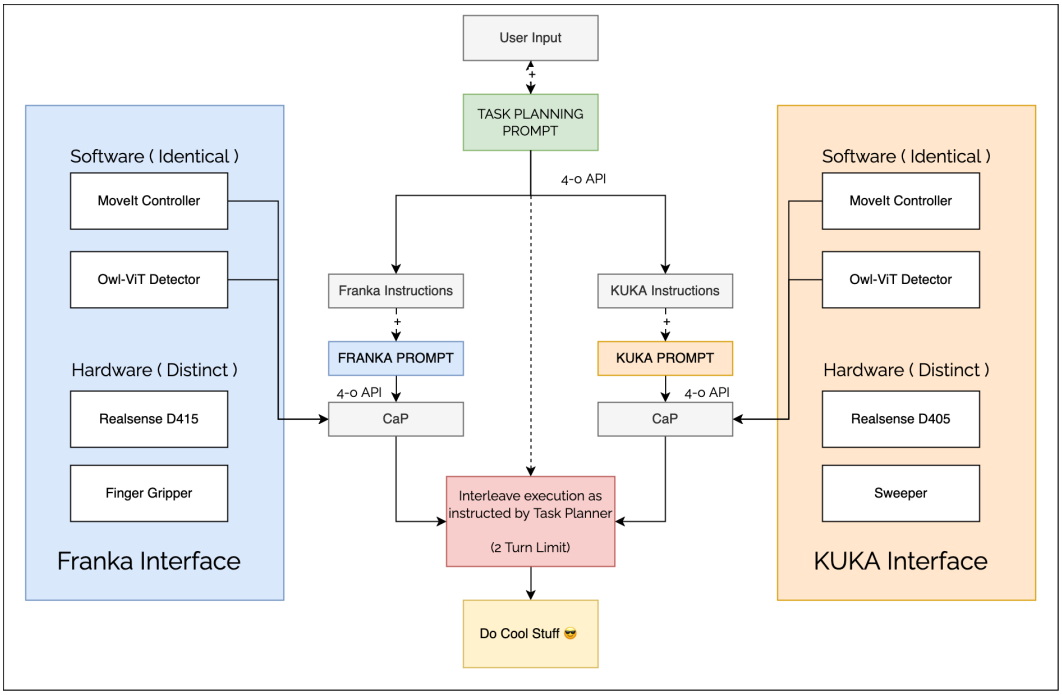


Figure 1: Block diagram of the full system.

We utilize a Franka Research 3 and an LBR Med 7 R800 arm, each with a wrist-mounted Intel RealSense camera (D415 and D405, respectively). Object detection and localization are done by using OWL-ViT [8]. Our trajectory optimization for our low-level skills is done with MoveIt2 [9]. Both the robots have different end-effectors (which we will elaborate later in the Robot Skills section). The entire system is based on ROS2 Humble and tested on Ubuntu 22.04.

We leverage OpenAI's GPT-4o for our code-as-policies generation. We also experimented with the GPT-4o-mini and Llama3.1 models, but both exhibited a greater tendency to hallucinate non-existent functions. Ultimately, we found GPT-4o to strike the best balance between accuracy and cost-efficiency. This is a turn-based bi-manual setup, where each task is constrained to be completed within two turns—one per robot. Consequently, executing a single task typically requires up to three LLM API calls: one for task planning and one for each robot, depending on its involvement in the task.

3.1 User Interface

We offered two different methods for communicating with the robot system. We support both using automatic speech recognition for verbal communication, but we also had a text-based interface where we typed commands to the robot. The automatic speech recognition was done using an open-source C/C++ implementation of OpenAI Whisper [10].

3.2 Robot Skills

Both robots possessed a shared set of skils:

- `detect_objects()`
  This function utilizes the OWL-ViT model to identify and extract bounding boxes for objects within the scene. Each robot's camera is calibrated to map the (x, y, z) coordinates of detected objects into a global coordinate frame, shared between the two robots. While this function can be part of the code-as-policies output, we usually manually call this ourselves and make the `detections` collection object available to the policy. We also generate a scene description and pass it along with our prompt.
- `move_robot(pose)`
  Commands the robot's end-effector to move to a specified pose.
- `move_home()`
  Returns the robot's end-effector to its predefined home position. This is executed at the end of each robot's turn to prevent potential collisions between robots.

These motion capabilities are enabled by the MoveIt2 framework, leveraging the `Pick_ik` inverse kinematics solver [11] and the Pilz industrial motion planner for precise and efficient movement control.

Both robots also had specialized skills tailored to their specific functionalities:

- **FR3 Robot**
  Equipped with a gripper attachment, the FR3 robot supports:
  - `open_gripper()`: Opens the gripper to release objects.
  - `close_gripper()`: Closes the gripper to securely grasp objects.
- **LBR Arm**
  Designed with a sweeping attachment, the LBR robot can:
  - `sweep_right()`: Sweeps objects to the right.
  - `sweep_left()`: Sweeps objects to the left.

3.3 Prompts

We have a task planning prompt for the task planning LLM, as shown below. It captures the constraints and capabilities of each robot, with the final goal being to determine what each robot should do and generate 1) the sequence in which the robots should go and 2) robot specific instructions.

```
You are a reasoning robot LLM responsible for high-level planning, reasoning, and
    ↪delegating tasks to two worker LLMs: Franka and LBR. Each worker LLM represents a
    ↪robot with distinct abilities:

1. **Franka (Gripper Robot)**:
  - Can move to specific positions in the workspace.
  - Can grasp objects using a gripper.
  - Can release objects.
  - CANNOT GO THE RECYCLE ZONE.
  - Left is +ve y and forward is +ve x.
  - This robot can only move within its workspace that is,
    - x is between -0.5 and -0.8
    - y is between 0.4 and -0.4
  - IT CANNOT MOVE THE END-EFFECTOR OUTSIDE THESE WORKSPACE LIMITS DUE TO SAFETY
      ↪CONSTRAINTS. THAT IS YOU CANNOT GO TO A POSE LIKE `(-0.4, 0.0)`

2. **LBR (Sweeper Robot)**:
  - Has a sweeper as end-effector and cannot hold objects.
  - It can sweep objects in y direction.
  - Left is -ve y and forward is -ve x.
  - This robot can sweep an object if and only if the object is within its workspace that
      ↪is,
    - x is between -0.3 and -0.5
    - y is between 0.4 and -0.4
```

```
21
22  Your task is to:
23  1. Analyze the high-level objective.
24  2. Reflect on your reasoning to ensure optimal task allocation.
25  3. Reason about the abilities of Franka and LBR to break down the task into exactly TWO
        ↪subtasks to acheive the goal.
26  4. Provide prompt for each worker to give detailed explanations for each subtask.
```

Additionally, we included some rules that Franka must follow to ensure feasible grasping of objects.

```
1   **Pre-Grasp Pose**: To approach the object from above, add a positive offset to obj_pose.
        ↪pose.position.z`. For example, to move 10cm above the object, you would do:
2   ```python
3   pre_grasp_pose = copy.deepcopy(obj_pose)
4   pre_grasp_pose.pose.position.z += 0.1
5   ```
6   **Grasp Pose**: To grasp the object properly, add a negative offset of 5cm along the z-axis
        ↪ to the original obj_pose. Do not adjust from pre_grasp_pose. For example:
7   ```python
8   grasp_pose = copy.deepcopy(obj_pose)
9   grasp_pose.pose.position.z -= 0.05
10  ```
11  **Important**: Do not compute grasp_pose by modifying pre_grasp_pose. Always use a fresh
        ↪copy of obj_pose when applying offsets for
12  different poses.
13  **Important**: These rules are only for grasping. For placing, you should always move up a
        ↪certain height above the object before releasing.
```

To aid the LLM in formatting the output correctly, we also provide an example to enable in-context learning. We only provide a trivial example since we want the LLM to reason and figure out how to compose functions to achieve more complex behavior.

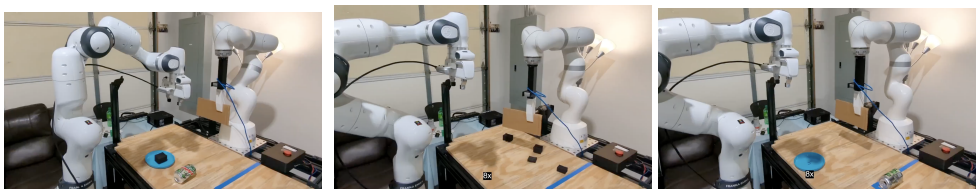
```
1   [Prompt]
2   Touch the box.
3
4   [Output]
5   Since I only need to move to the box I'm gonna set that to my target label and move to the
        ↪detected object.
6
7   ```python
8   target_label = 'box'
9   if target_label in detections:
10     detected_object = detections.find(target_label)
11     if detected_object:
12        print(f"{detected_object.label} found at {detected_object.center_3d}")
13        print(f"Moving to {detected_object.center_3d}")
14        robot_interface.move_robot(robot_interface.get_pose(detected_object))
15  else:
16     print(f"{target_label} not found.")
17  ```
```

Even with the rules imposed above, the LLM still generates type errors. For example, we expect the coordinates in poses to be all in float (ROS requirement), but it sometimes generates `(1.0, 0, 2.0)` as coordinates. The `int 0` will make the code not executable. Another example is that LLM tends to use a `for` loop or `next` function on the detection result, which will also make the code not executable. We had to explicitly lay down the following rules to make the LLM generate the desired output.

```
1   **Important**: ALWAYS USE FLOATS FOR ALL POSES.
2   **Important**: When searching for a detected object in `detections`, always use the `
        ↪detections.find(label)` method.
3   Do not use loops or the `next` function.
4   Assume that `detections` has a `find(label)` method that returns the detected object with
        ↪the given label.
5   There is only one object of each type, so you can assume `detections.find(label)` will
        ↪return the correct object if it exists.
6   If there are multiple objects of the same type (let's say block in the scene description),
        ↪ignore them. Those are false positives.
7   Just use detections.find(label) to get the first one.
```

4 Experiments

We conducted a sequence of experiments for a variety of different prompts. The operating space of the two robot arms is shown in Figure 2. We describe three representative prompts and their outcomes. For each of the tasks, the LLM leverage the scene description from `detect_objects()` along with the user instruction to decide which robot arm to delegate for the task.



(a) Prompt 1 (b) Prompt 2 (c) Prompt 3

Figure 2: Experimental setup for each prompt. The area marked with blue tape is the "recycling area" accessible only to the sweeping LBR arm.

Prompt 1: *"Clean the plate by moving the block somewhere to the left and then place the can on the plate"* In this task, there was a block on top of a blue plate and a can on the side. The LLM delegated the task to the Franka arm, having it grasp the black block, place it on the table, then grab the can and place it on the plate. Although only one arm was used in this task, the LLM had access to both arms' API and automatically decided to use only one arm since the sweeping arm would not be able to grasp the object on the plate or the can.

Prompt 2: *"The table has gotten very dirty. Can you help clean it up?"* On this task, the LLM delegated all the motions to the sweeping robot. Interestingly, the robot decided to use two sweeps to get two blocks, carefully placed so that the height of each sweep only touched one block. The robot could have swept two blocks with only one sweep, but it seems that the LLM could not reason about sweeping multiple objects in one action sequence. Two of the four blocks were untouched by the arm. This reveals a limitation of our approach where it may not be able to reason about the long-term effects of the robot's actions on the environment without calling `detect_objects()` multiple times.

Prompt 3: *"A can accidentally went into the recycling area. Can you bring it out of that area and then put it on the plate?"* In this setting, the can was in the recycling area and was off-limits to the Franka arm. The LLM correctly reasoned to use the LBR's sweeping capabilities first to drag the can out of the recycling area and place it within the operating space of the Franka. Since the LBR could not pick up objects, the LLM correctly decided to send the LBR to the home position and use the Franka to then pick up the can and place it on the plate.

Failure Modes and Limitations The system cannot reason about collision-avoidance constraints at the moment. The experiments were set up so that both arms did not have to avoid any objects in the scene, and by sending each robot to the home position before allowing the next robot to move, they did not have to worry about inter-robot collisions. This was necessary to have a successful demo by the end of the semester, but it also introduces limitations on the capabilities of the bi-manual system. Specifically, any task where both arms would need to move simultaneously is not possible for the system. Furthermore, this restriction meant we do not have to perceive the other robot arm's configurations actively. A consequence of this is shown below in Figure 3.

A key limitation to progress in this space lies in the mismatch between the LLM's ability to reason in natural language and the challenge of translating spatial or visual information into text-based representations. This mismatch creates a tradeoff: how much spatial reasoning should be delegated to the LLM versus how much should be handled by underlying, specialized functions. Over-abstraction risks reducing the role of the LLM to a simple command generator, effectively negating its reasoning capabilities. Conversely, relying on the LLM to independently reason through all spatial complexities increases the likelihood of errors due to its limited understanding of spatial dynamics and its

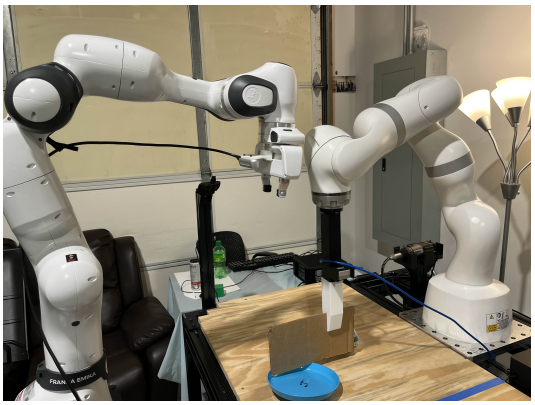


Figure 3: Safety-critical Failure : The `move_home` command failed due to an IK issue, causing the other arm to almost collide into it.

lack of direct perceptual feedback. We believe balancing this tradeoff is critical and it remains a fundamental challenge for building systems that integrate language with robotics.

5 Conclusion

This project demonstrates the capabilities of Large Language Models for coordinating two robotic arms without explicit assignments for which arm should do what at deployment time. Although we scoped the project to turn-based bi-manual manipulation, we hypothesize that with more complex (potentially learned) skills that utilize both arms, the LLM could efficiently coordinate both arms to perform tasks such as object handover and jointly lifting larger objects. We identified that by allowing the LLM to only focus on high-level reasoning, rather than outputting trajectory waypoints as in [4], we have significantly fewer API calls and have consistent performance between runs. We view that LLMs have extremely useful capabilities in reasoning over open-vocabulary and open-object settings. However, for tasks such as generating dynamically feasible trajectories, leveraging mature trajectory optimization frameworks offer a more stable solution for problems well understood by roboticists. Our view after this project is that although LLMs are useful high-level thinkers, thoughtful integration with pre-existing robotics methods will be crucial to having a reliable system. This work does not address safety constraints, such as collision avoidance (potentially solvable with existing robotics frameworks) or semantic constraints addressable by large pre-trained models (i.e., carefully handling sharp/delicate objects). Adding such safety guardrails to our framework is left for future work.

References

[1] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[2] M. Shridhar, L. Manuelli, and D. Fox. Cliport: What and where pathways for robotic manipulation, 2021. URL https://arxiv.org/abs/2109.12098.

[3] Y. Jiang, S. S. Gu, K. P. Murphy, and C. Finn. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[4] Z. Mandi, S. Jain, and S. Song. Roco: Dialectic multi-robot collaboration with large language models, 2023. URL https://arxiv.org/abs/2307.04738.

[5] J. Varley, S. Singh, D. Jain, K. Choromanski, A. Zeng, S. B. R. Chowdhury, A. Dubey, and V. Sindhwani. Embodied ai with two arms: Zero-shot learning, safety and modularity, 2024. URL https://arxiv.org/abs/2404.03570.

[6] M. G. Arenas, T. Xiao, S. Singh, V. Jain, A. Ren, Q. Vuong, J. Varley, A. Herzog, I. Leal, S. Kirmani, M. Prats, D. Sadigh, V. Sindhwani, K. Rao, J. Liang, and A. Zeng. How to prompt your robot: A promptbook for manipulation skills with code as policies. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4340–4348, 2024. doi:10.1109/ICRA57147.2024.10610784.

[7] L. X. Shi, Z. Hu, T. Z. Zhao, A. Sharma, K. Pertsch, J. Luo, S. Levine, and C. Finn. Yell at your robot: Improving on-the-fly from language corrections, 2024. URL https://arxiv.org/abs/2403.12910.

[8] M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn, A. Dosovitskiy, A. Mahendran, A. Arnab, M. Dehghani, Z. Shen, X. Wang, X. Zhai, T. Kipf, and N. Houlsby. Simple open-vocabulary object detection with vision transformers, 2022. URL https://arxiv.org/abs/2205.06230.

[9] M. Görner, R. Haschke, H. Ritter, and J. Zhang. Moveit! task constructor for task-level motion planning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 190–196, 2019. doi:10.1109/ICRA.2019.8793898.

[10] ggerganov. whisper.cpp. https://github.com/ggerganov/whisper.cpp.

[11] S. Starke, N. Hendrich, and J. Zhang. Memetic evolution for generic full-body inverse kinematics in robotics and animation. *IEEE Transactions on Evolutionary Computation*, 23(3): 406–420, 2019. doi:10.1109/TEVC.2018.2867601.