Simulating a Robotic Arm for Sequential Affordance-Based Manipulation Using Object Detection and ChatGPT

Haroon Muhammed, Dwarakesh Rajesh

Abstract – *Robotic systems must break down complex tasks into subtasks and execute them cohesively to achieve autonomy. This project focuses on enabling a robotic arm to perform multi-step manipulation tasks by integrating object detection, affordance reasoning, and task planning powered by ChatGPT. Human interaction with the robot occurs through natural language commands, such as "pick up the red ball from the drawer and place it in the cupboard." The robot interprets these commands using ChatGPT's natural language processing capabilities, detects objects in its environment through deep learning-based models, and infers object affordances (e.g., pulling a drawer handle or grasping a ball). The inferred affordances are then used to generate and execute a sequence of actions for successful task completion. The system is tested in simulated environments, focusing on real-time task sequencing and execution. Experimental results demonstrate that ChatGPT, when guided by human interaction, provides accurate and reliable support for robotic task planning and manipulation.*

Keywords**: **multi-step manipulation, object detection, affordances, natural language processing, ChatGPT, task sequencing, human-robot interaction

1. INTRODUCTION

A robot can be broadly defined as any machine capable of interacting with its environment to perform tasks autonomously or with minimal human intervention. This definition spans from simple household appliances, such as washing machines and toasters, to advanced humanoid robots. However, this project focuses specifically on modern robotics, narrowing the scope to robotic manipulators—versatile robotic arms that have revolutionized industries ranging from manufacturing to healthcare. Since the introduction of first-generation robots like UNIMATE to the more advanced fourth-generation robots such as Roomba and YuMi [1], manipulators have proven essential due to their adaptability and precision.

Despite their mechanical advancements, manipulators face critical challenges, particularly in cognition and adaptability. While significant research has focused on enhancing their dexterity and deformability—allowing them to grasp diverse objects, from fragile items like oranges to rigid steel pipes [2]—there remains a gap in their cognitive capabilities. Current manipulators often require extensive programming to adapt to new tasks, limiting their flexibility and scalability. This is where machine learning, particularly Large Language Models (LLMs) such as ChatGPT, offers transformative potential. LLMs have demonstrated unprecedented capabilities in decision-making and problem-solving across various domains, yet their integration into robotic manipulation remains underexplored.

This project investigates how LLMs can bridge the cognitive gap in robotic manipulators, enabling them to perform complex, multi-step tasks with minimal reprogramming. Specifically, it leverages ChatGPT/OpenAI to enhance a manipulator's decision-making abilities, using robot learning paradigms such as end-to-end learning, modular pipeline programming, and affordance-based reasoning. By integrating object detection, LLM reasoning, and robotic control, this project aims to address the following research questions:

i. Can ChatGPT correctly identify and predict object affordances based on detected objects and task descriptions?
ii. How reliable is the action sequence generated by ChatGPT/LLMs in ensuring the successful completion of multi-step tasks?
iii. What challenges arise when integrating object detection, LLMs, and robotic control for autonomous task completion?
iv. Which robot learning paradigm is most suitable for LLM/ChatGPT-based manipulation?

By addressing these questions, this project seeks to analyse the feasibility, efficiency, and challenges of using LLMs to provide cognition for robotic manipulation.

The remainder of this paper is organized as follows: Section 2 discusses prior work in the field; Section 3 outlines the methodology; Section 4 presents results and analysis; and Section 5 concludes with future directions.

Acknowledgements: I would like to thank Dr. Eshed Ohn Bar for providing a wonderful opportunity and supporting this endeavour.

2. RELATED WORK

Affordances, a concept grounded in psychology, have been widely studied in robotics to define the action possibilities between a robot and its environment. H. Min et al. (2016) provide a comprehensive survey on affordance-based research in developmental robotics, emphasizing its role in bridging sensorimotor interactions with high-level reasoning for adaptable learning. Their work examines various challenges and implementation strategies for affordance-based frameworks, highlighting their potential to empower robots with functional understanding of their environment [3]. This project builds on their affordance model frameworks to enable manipulation through developmental learning, allowing the robot to make informed decisions based on object affordances.

X. Yang et al. (2023) explore methodologies to infer affordances and integrate them within a reinforcement learning (RL) framework for decision-making. By formalizing affordances as state-action relationships through Markov Decision Processes, they advance affordance-based reasoning for robotics. However, their work identifies challenges in the availability of large datasets and comprehensive benchmarks [4]. Extending their approach, this project integrates multimodal data—combining depth camera input with RGB images—to enhance affordance reasoning and guide task-specific decision-making.

Large Language Models (LLMs) have emerged as transformative tools in cognitive robotics. N. Wake et al. (2023) leverage ChatGPT's few-shot learning capabilities to decompose high-level user input into granular robot tasks. Their adaptive framework incorporates a user feedback system to refine outputs and improve affordance-based action plans [5]. This project draws on their methodology by using ChatGPT to translate task descriptions into sequential actions and by evaluating the reliability of LLM-generated task sequences for robotic manipulation.

Y. Jin et al. (2024) propose a framework combining ChatGPT with reinforcement learning to create a stable policy for robotic tasks. ChatGPT serves multiple roles, including error correction, decision-making, and task evaluation, within a feedback loop that refines the robotic agent's performance over time. This project takes inspiration from their structured prompting approach and integrates a feedback system to improve task planning and execution efficiency [6]

Incorporating natural language processing with vision-based systems, H. Ahn et al. (2018) present a framework that generates position heatmaps and uncertainty maps to locate desired objects based on natural language commands. Their model uses an interactive questioning/feedback loop to resolve ambiguities, improving task understanding and execution [7]. This project adopts a simplified version of their end-to-end learning approach, leveraging affordance-based reasoning and depth data to create a small-scale system for robotic manipulation.

The proposed solution (discussed in Section III) builds on these contributions by combining affordance-based reasoning, LLM-powered decision-making and multimodal data integration to address challenges in cognitive and adaptive robotic manipulation. The proposed framework is at the intersection of developmental robotics and AI, extending prior work with a focus on object and task specific affordance prediction, task sequencing based on affordances and user input and integrated learning paradigms.

3. METHODOLOGY

The project aims to develop a simple, adaptive, and intuitive framework for robotic manipulation. This framework allows users to input a task prompt—whether simple or complex—in natural language, which the robot interprets and executes autonomously.

The simulation environment for this project was created using Webots R2023b [8] and designed to remain minimal yet functional. The setup includes a worktable equipped with a UR5e robotic arm, positioned at an offset of [0,0,0.2] from the origin. A secondary table with a soccer ball serves as the object interaction area. The robotic arm is outfitted with two end effectors: (1) an Astra RGB-D Camera by Orbbec, mounted 0.03 m above the tool slot for perception tasks, and (2) a ROBOTIQ EPick Vacuum Gripper, installed on the tool slot for manipulation. This simulation environment, shown in Figure 3.1, forms the foundation for testing and validating the proposed framework.

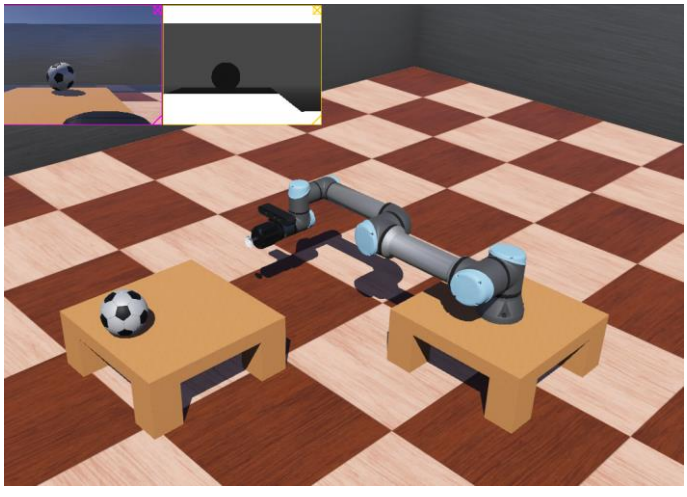

Figure 3.1: Simulation World Setup

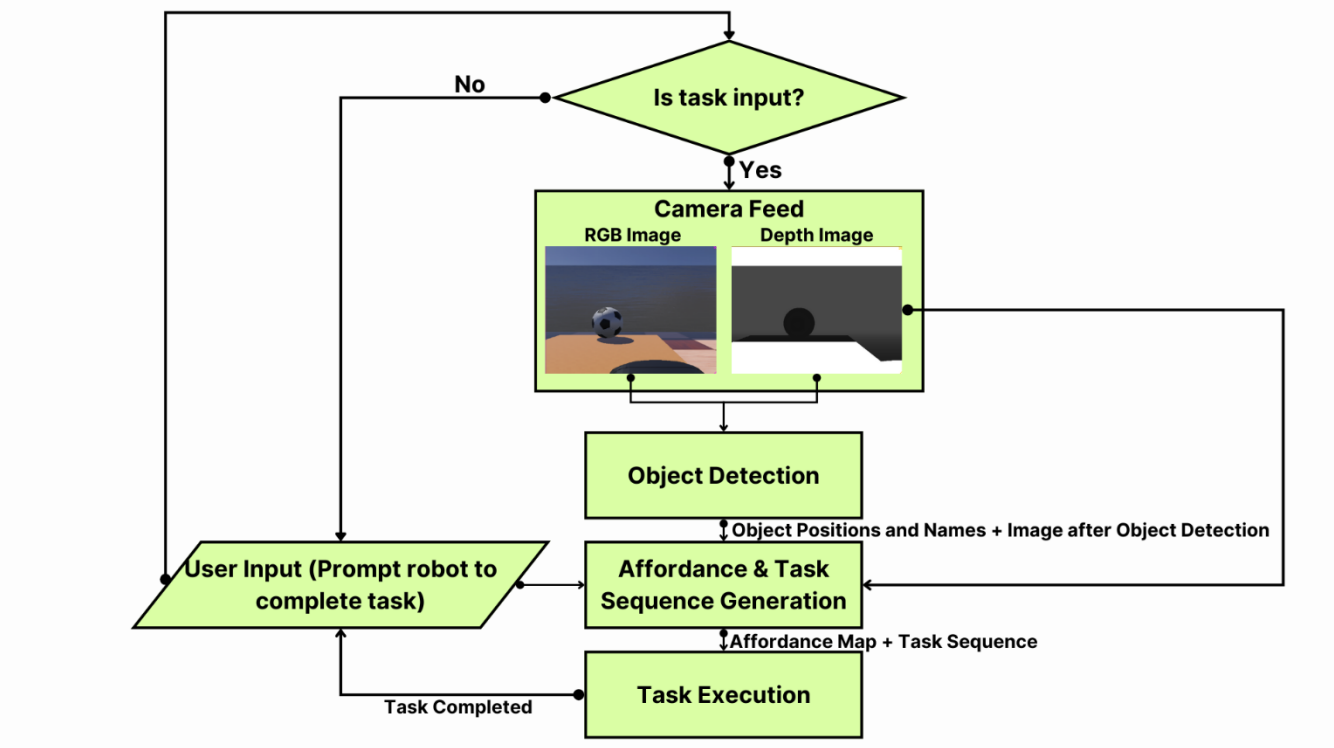

Figure 3.2: An overview of the framework

The proposed framework allows users to input tasks for the robotic arm in natural language. The system processes these inputs to generate a series of commands that can be executed to complete the specified task effectively.

As illustrated in Figure 3.2, the framework comprises three main modules:

1. Object Detection
2. Affordance and Task Sequence Generation
3. Task Execution

The workflow begins by capturing the most recent frame from the robotic arm's camera upon receiving a task input. This frame is processed by the Object Detection module, which identifies and outputs the names and positions of detected objects. These object positions, along with the user input and camera feed, are passed to the Affordance and Task Sequence Generation module. This module generates a detailed series of actions required to complete the task.

The generated actions are then forwarded to the Task Execution module, which utilizes the inverse kinematics model of the UR5e robotic arm to compute the appropriate joint angles for movement. If the action requires manipulation, the system activates or deactivates the vacuum gripper based on the specific command. This modular and structured approach ensures precise task completion while maintaining flexibility for various inputs.

3.1 **Object Detection**

This module is responsible for identifying all objects within the environment at any given frame and mapping their locations relative to the camera frame. It forms the foundation of the framework by providing accurate object detection and localization.

The module takes as input the camera feed, consisting of an RGB image and a depth image. These inputs are converted into arrays using NumPy, and the RGB image is further processed using OpenCV to interface with the object detection model. The object detection model employed in this project is YOLOv4, which has been pretrained and fine-tuned on the COCO Dataset. The model's pretrained weights, configuration files, and class labels were sourced from GitHub [9].

YOLOv4 enables object detection by drawing bounding boxes around detected objects, thereby localizing them within the image. The position of each detected object is determined by calculating the centroid of its bounding box. Using the intrinsic parameters of the camera, the position of the object relative to the camera is calculated. This process provides precise object localization within the camera's frame of reference. The workflow of this module is visualized in Figure 3.1.2.

It is important to note that the extrinsic parameters of the camera were disregarded in these calculations. This simplification is valid because the robotic arm always returns to its initial position before executing a task, ensuring that the camera frame aligns with the world frame.

For task execution, however, the object's position relative to the robot base is required. This aspect is critical for inverse kinematics, is detailed in Section 3.3.

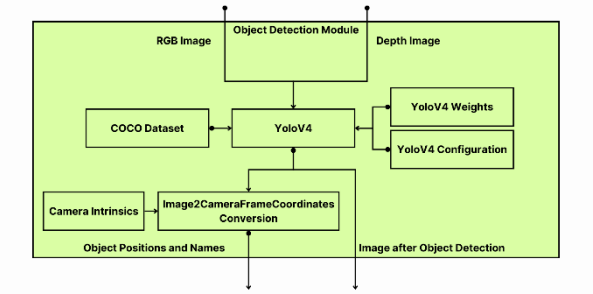

Figure 3.1.2: Detailed workflow of the Object Detection Module

Challenge 1: Duplicate Object Detections - The model frequently detected the same object multiple times, leading to numerous redundant instances of the same object throughout the simulation.

Solution: Non-Maxima Suppression (NMS) was implemented to address this challenge. This method ranks bounding boxes based on their confidence scores, selects the highest-ranked bounding box, and calculates the Intersection-over-Union (IoU) for all other boxes relative to the chosen box. Boxes with an IoU exceeding a predefined threshold are eliminated as they overlap significantly with the selected box. This process effectively reduces duplicate detections.

Challenge 2: Tuning the Threshold for NMS - Defining an appropriate IoU threshold for NMS is critical. A well-tuned threshold ensures accurate elimination of overlapping bounding boxes, but its selection becomes increasingly challenging as the environment's complexity grows. Complex environments require finer tuning of the threshold to strike a balance between eliminating duplicates and retaining valid detections.

Challenge 3: Missed Object Detections - The model occasionally failed to detect objects from certain classes, even when these objects were clearly present and part of the relevant class. This limitation highlights the reliance of the detection process on confidence levels, which can lead to inconsistencies in identifying objects in the scene.

Future Improvement - The current object detection model demonstrates limitations in handling complex environments that demand robust perception. A potential improvement involves creating and training a policy to dynamically adjust parameters, such as the IoU threshold and confidence levels, based on the environment's complexity. This adaptive approach would enhance the model's reliability and performance across diverse scenarios.

3.2 **Affordance and Task Sequence Generation**

This module focuses on visual reasoning via affordances to generate a cohesive series of sub tasks (actions) required to complete the given task. By leveraging affordances, it bridges the gap between perception and action, enabling the robot to interact effectively with its environment. The workflow of this module is represented in Figure 3.2.1.

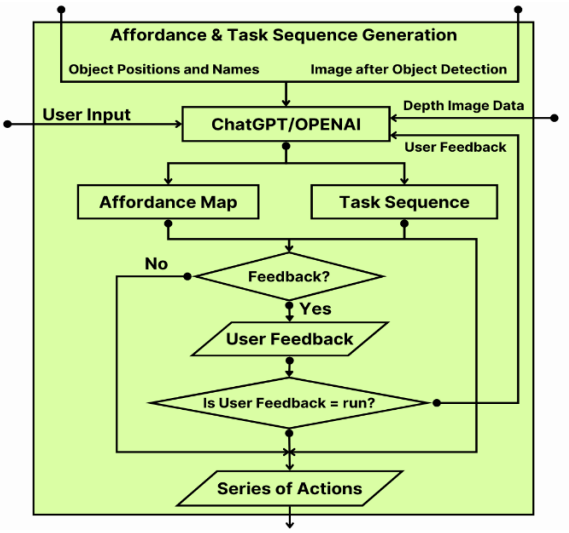

Figure 3.2.1: Detailed overview of the Affordance & Task Sequence Generation Module

The module inputs:

i. User Input – A natural language command provided by the user, prompting the robot to execute a specific task.
ii. Depth Image Data – The depth image obtained from the camera feed, formatted as an array for further processing.
iii. Image after Object Detection – The RGB image containing bounding boxes around all detected objects, each labelled with its corresponding name, as shown in Figure 3.2.2.
iv. Object Positions and Names – A structured list of dictionaries, where each dictionary includes the position and label of a detected object.
v. User Feedback – Initially null, this input allows the user to provide critical feedback to refine the output if necessary, improving task execution in subsequent iterations.

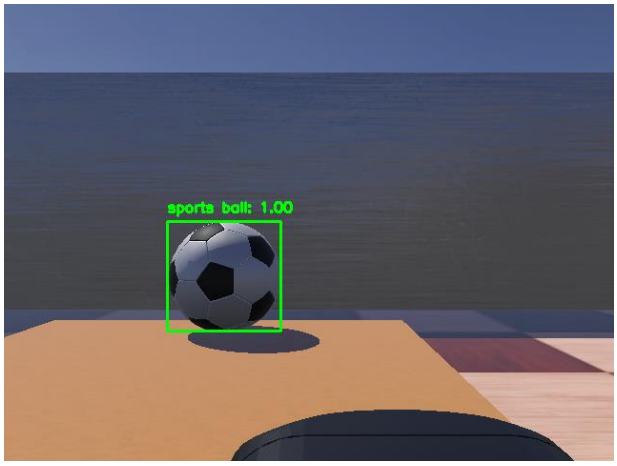

Figure 3.2.2: Image after Object Detection

These inputs are sent to the GPT-4o-mini model, along with an initial structured prompt designed to generate the required outputs. This prompt ensures that the response includes an affordance map and task sequence - lists of dictionaries containing object_name & affordances and action and its associated parameters like target_position or target_object respectively.

The initial prompt –

```
message = [
    {"role": "system", "content": "You are an assistant that generates affordance maps to aid robot learning and generates task sequences for robots."},
    {
      "role": "user",
      "content": [
      {
        "type": "text",
        "text": f"Based on the image, task in hand: {task_in} and the depth data: {depthimarr}. Generate an affordance map of each object in the scene called affordance_map and output it as a JSON format containing object_name and affordances. The output should be a JSON format only!!. Generate the task sequence too in JSON format, called task_sequence, any form of transformation would be referred to as move (MOVE actions should have a target position based on the {objposes},depth data and image) and any action related to grippers will be pick-up and place.No comments please",
      },
      {
        "type": "image_url",
        "image_url": {
        "url": f"{base64_image}"
        },
      },
      ],
    }
  ]
```

The above prompt ensures that the model outputs a consistent JSON format containing two keys – 'affordance_map' for storing the affordance map list of dictionaries describing affordances and 'task_sequence' to store the task sequence list of dictionaries describing action as seen in Figure 3.2.3


```
{
  "affordance_map": [
    {
      "object_name": "sports ball",
      "affordances": ["roll", "catch", "kick", "grab"]
    }
  ],
  "task_sequence": [
    {
      "action": "MOVE",
      "target_position": [-0.1059256541141781, 0.6651793122291565, 0.027374045445237034]
    },
    {
      "action": "pick-up",
      "target_object": "sports ball"
    },
    {
      "action": "MOVE",
      "target_position": [0.0, 0.0, 0.0]
    },
    {
      "action": "place",
      "target_object": "sports ball"
    }
  ]
}
```


Figure 3.2.3: Output from GPT

The user reviews the output and provides feedback as needed. This feedback is concatenated to the initial prompt and sent back to the model via the API to refine the response.

Challenge 1: Inconsistent Output Format - The model occasionally generates outputs that deviate from the expected structure.

Solution - Prompt engineering techniques were employed to enforce a specific output structure. This significantly reduced errors and improved consistency.

Challenge 2: Comments in the Output - The model often includes comments in the response, which interfere with JSON parsing and processing.

Solution - A RegEx-based filter was implemented to remove comments from the output, ensuring successful parsing.

Challenge 3: Unnecessary Actions in Output - The model tends to generate extraneous actions not required by the user. For instance, a user prompt like "grab the ball on the

table" resulted in additional actions such as "place," which were unnecessary for the task.

Solution - A user feedback system was introduced, allowing users to refine the model's output before execution. This ensures alignment with task expectations and eliminates unneeded actions.

Future Improvements: Automating the refinement process to reduce dependency on user feedback while maintaining task accuracy.

The results of these additions will be further discussed in Section 4.

3.3 **Task Execution**

The final module focuses on executing each action command sequentially to complete the task. An overview of this module's structure is depicted in Figure 3.3.1.

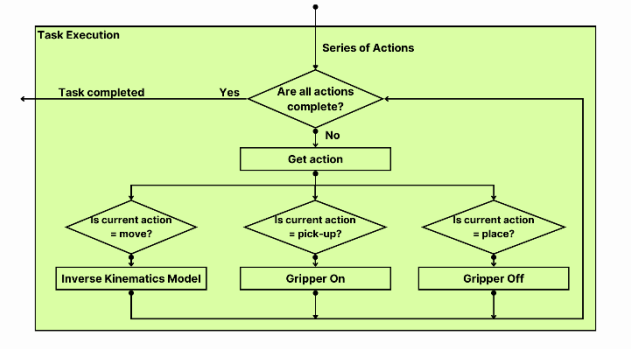

Figure 3.3.1: Structural overview of the Task Execution Module

This module takes as input a series of actions, which are the output of the Affordance and Task Sequence Generation Module discussed in Section 3.2. It ensures that all subtasks are executed cohesively in the same sequence as provided by the GPT model's output.

If the action is 'move', the target position is extracted and sent to an inverse kinematics model based on the URDF chain of the UR5e arm defined in the simulation environment. The module employs the ikpy library, which uses the URDF chain from the '.proto' file generated by Webots/Cyberbotics [10].

The position of the target object relative to the camera frame is provided by the GPT model. However, the inverse kinematics model requires the position relative to the robot's base frame. The transformation from the camera frame to the base frame is calculated using forward kinematics module in ikpy. The pose (position and orientation) of the camera is assumed to be identical to the pose of the robotic arm's end-effector at any given time. This transformation is used to accurately calculate the object's position relative to the robot's base, which is then input to the inverse kinematics model for motion planning.

Challenge 1: Inaccurate URDF File - The URDF file used in the simulation lacked precision, leading to errors in both inverse and forward kinematics calculations and resulting in incorrect robot motion.

Solution: Manual calculations for forward and inverse kinematics were attempted, but they did not yield significant improvements. A potential long-term solution involves switching to a simulation platform better suited for robotic manipulators. While WeBots is proficient at simulating mobile robots, it has limitations in handling non-mobile manipulators. Unfortunately, the dependency issues with ROS prevented the use of Gazebo and MoveIt, and time constraints limited the exploration of alternative platforms such as CoppeliaSim V-REP and ISAAC Sim. A transition to one of these platforms in future work would likely resolve the inaccuracies in simulation and kinematics.

4. RESULTS

This section provides a detailed overview of the experimental setups and outputs obtained during testing. An in-depth analysis of the results is carried out to support the proposed hypothesis.

Experiment 1: Modular Pipeline Paradigm for Robot Learning without any feedback and a vague prompt.

Prompt –

"Based on the image, task in hand: {task_in} and the depth data: {depth_im}. Generate an affordance map of each object in the scene and output it as a JSON format containing object_name and affordances. The output should be a JSON format only!!. Generate the task sequence too in JSON format any form of transformation would be referred to as move (MOVE actions should have a target position based on the {poses},depth data and image) and any action related to grippers will be pick-up and place."

Task – "Grab ball on the table"	
Trial Number	Result
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0.5

Table 4.1 – Results after running Experiment 1, 10 times

Upon analysis of Table 4.1, the experiment failed 9.5/10 times. The failures were attributed to the vague prompt and lack of feedback, leading to unstructured GPT outputs. Erroneous parsing was caused by comments included in the output, as shown in Figure 4.1. The result of 0.5 in Trial 10 indicates the GPT model produced an output in the correct structure but omitted a critical parameter, such as the target_position, as illustrated in Figure 4.2.

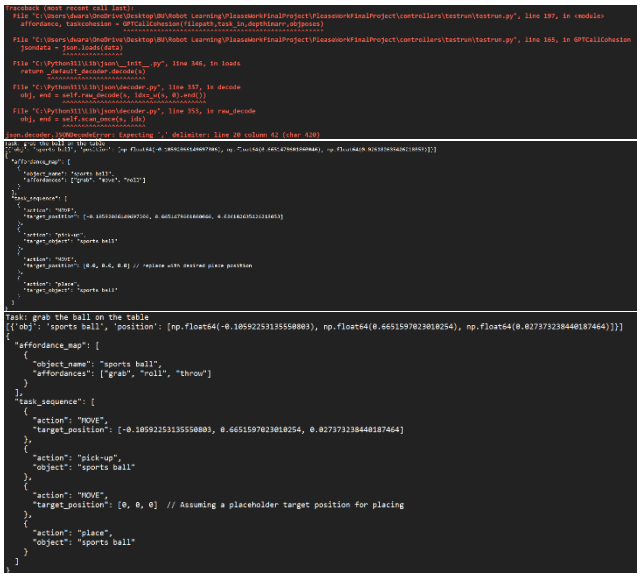

Figure 4.1: Shows the error message and the multiple GPT outputs resulting in this error that led to Result = 0.

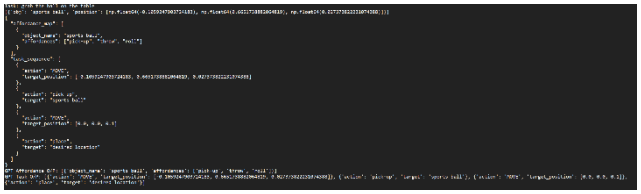

Figure 4.2: Shows the GPT output that led to Result = 0.5.

Experiment 2: Modular Pipeline Paradigm for Robot Learning without any feedback after Prompt Engineering.

Prompt after Prompt Engineering –

"Based on the image, task in hand: {task_in} and the depth data: {depth_im}. Generate an affordance map of each object in the scene and output it as a JSON format containing object_name and affordances. The output should be a JSON format only!!. Generate the task sequence too in JSON format any form of transformation would be referred to as move (MOVE actions should have a target position based on the {poses},depth data and image) and any action related to grippers will be pick-up and place. No comments please"

Task – “Grab ball on the table”	
Trial Number	Result
1	0
2	1
3	1
4	0
5	0.5
6	1
7	0
8	0
9	0.5
10	0

Table 4.2 – Results after running Experiment 2, 10 times

Figure 5.3: Shows the error message and the GPT output resulting in this error that led to Result = 0.

Figure 4.4: Shows the GPT output that led to Result = 1.

Prompt engineering improved results significantly, with a success rate of 4/10 trials, an 8-fold improvement over Experiment 1. However, parsing errors persisted due to GPT's tendency to include comments or unstructured output, as demonstrated in Figure 4.3. Figure 4.4 highlights a successful output.

Experiment 3: Modular Pipeline Paradigm for Robot Learning with a vague prompt, RegEx implementation and without any feedback.

RegEx Expression - data = re.sub(r'//.*?\n', '', data)

Prompt –

"Based on the image, task in hand: {task_in} and the depth data: {depth_im}. Generate an affordance map of each object in the scene and output it as a JSON format containing

object_name and affordances. The output should be a JSON format only!!. Generate the task sequence too in JSON format any form of transformation would be referred to as move (MOVE actions should have a target position based on the {poses},depth data and image) and any action related to grippers will be pick-up and place."

Task – "Grab ball on the table"	
Trial Number	Result
1	1
2	1
3	0
4	1
5	1
6	1
7	0.5
8	0.5
9	1
10	1

Table 4.3 – Results after running Experiment 3, 10 times

With the addition of RegEx to preprocess GPT outputs, the success rate improved to 8/10 trials. However, parsing errors persisted, as some erroneous outputs bypassed the RegEx filter (e.g., Figure 4.5). A failure case occurred when GPT provided incomplete parameters, as shown in Figure A.1.

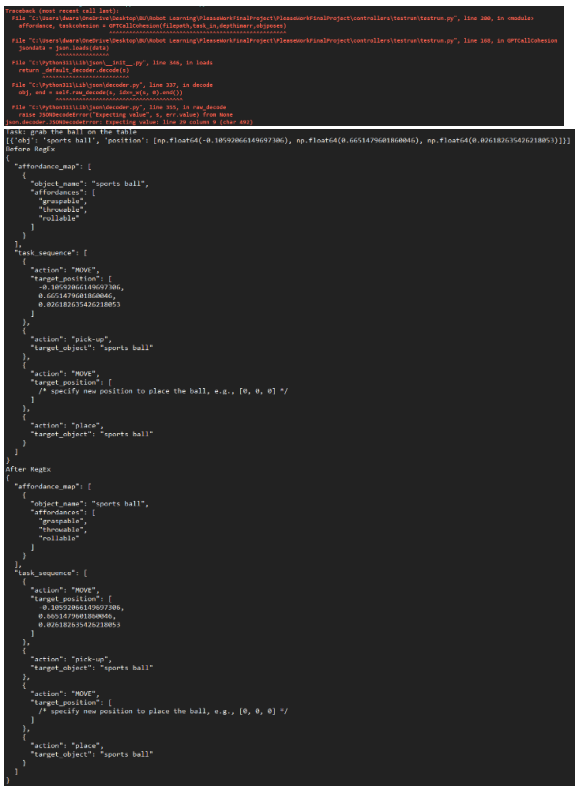

Figure 4.5: Shows the error message and the GPT output resulting in this error that led to Result = 0.

Figure A.2 shows the output from GPT that provide a successful result.

Experiment 4: Modular Pipeline Paradigm for Robot Learning with prompt engineering RegEx implementation, and without feedback.

Task – "Grab ball on the table"	
Trial Number	Result
1	1
2	1
3	1
4	0
5	1
6	1
7	1
8	1
9	0.5
10	1

Table 4.4 – Results after running Experiment 4, 10 times

The experiment demonstrated an 8.5/10 success rate. Failures stemmed from minor parsing issues or mislabelling in the output structure (e.g., "affordance_map" being labelled "affordances" as shown in Figure 4.6). Despite these challenges, this experiment was the most successful in the modular paradigm.

Figure A.4 and Figure A.5 in Appendix show the outputs that resulted in a successful outcome (Result = 1). Figure A.3 shows an error message which was encountered due to mislabelling.

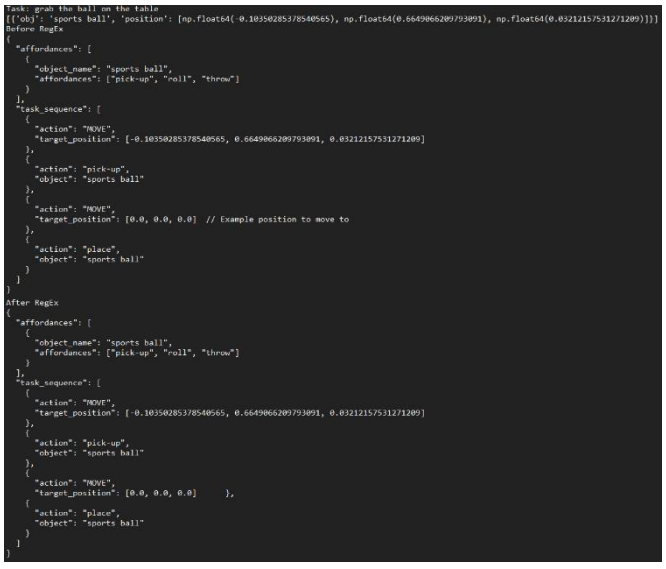

Figure 4.6: Shows the GPT output resulting that led to Result = 0.

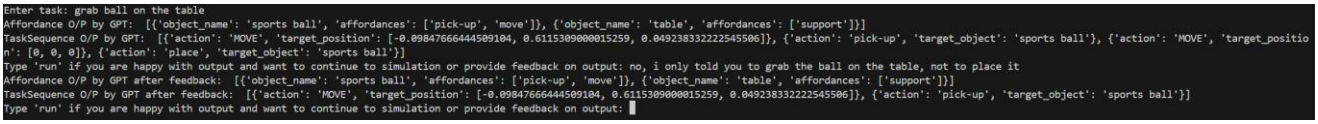

Figure 4.7: Shows the feedback communication system and the corresponding outputs based on feedback

Experiment 5: Modular Pipeline Paradigm with Feedback, prompt engineering and RegEx.

This experiment incorporated human feedback to refine GPT outputs, resulting in more robust task execution. Figure 4.7 demonstrates how user intervention improved model performance.

Experiment 6: End to End Learning Paradigm with Feedback.

Unlike the modular pipeline, this approach utilized GPT for object detection, affordance prediction, and task cohesion. However, it failed due to GPT's inability to predict object positions accurately, even when provided with camera intrinsics. The model consistently output incorrect target positions, as shown in Figure A.6, even after feedback.

Results from Inverse Kinematics Model:

```
Give me a task to do!
Task: grab the ball on the table
Sent
Command: move to [0.833, 0.223, 0.758]
Moving to [0.833, 0.223, 0.758]
Command: gripper on
turned grippper on
Command: All sub tasks complete
Waiting for next task
```

Figure 4.8: Walkthrough of the console output for any given task until completion.

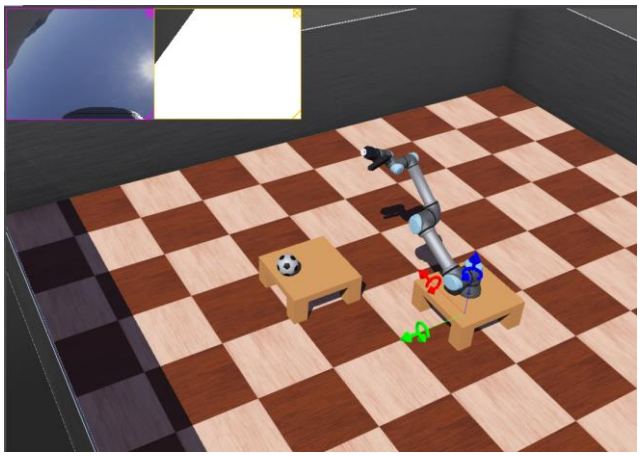

Figure 4.9: The position of the robot after command: move to [0.833,0.233,0.758] was executed

Figure 4.8 highlights the robot's workflow, while Figure 4.9 illustrates inaccuracies in the IK model that caused the robot to move to incorrect positions.

Results from YOLOv4 Object Detection:

Figure 4.10 and Figure 4.11 compare object detection before and after Non-Maxima Suppression (NMS), showcasing improved object localization.

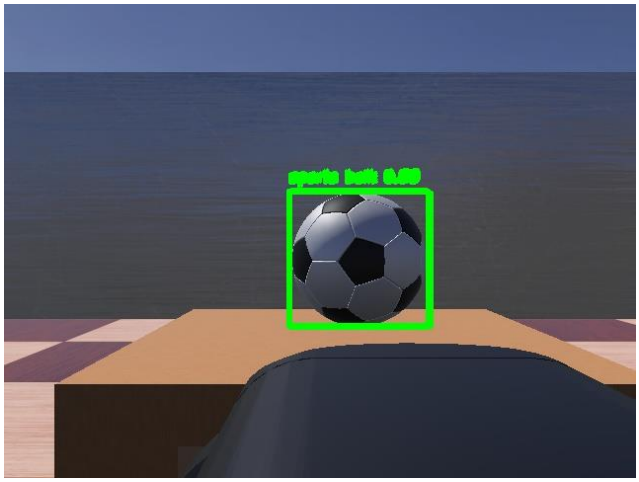

Figure 4.10: Output before NMS implementation

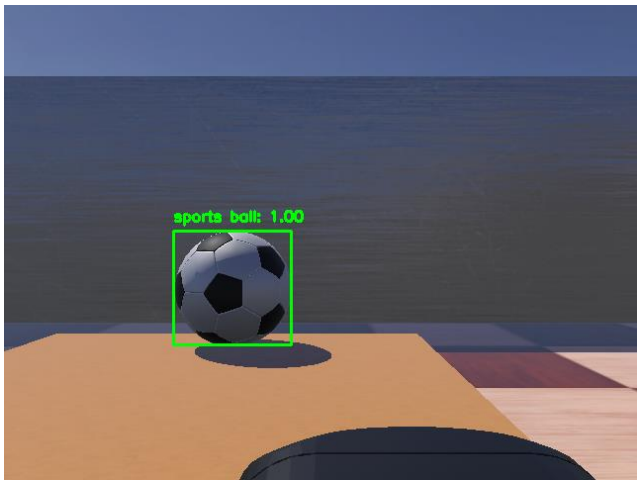

Figure 4.11: Output after NMS implementation

4. CONCLUSION

After thorough consideration of the different paradigms and analysis of experimental results, this study demonstrated that the modular pipeline paradigm with user feedback proved to be the most effective approach for robotic task execution. The inclusion of feedback significantly improved GPT's output reliability, highlighting the critical importance of consistency in the framework's success. The success rate of the model increased exponentially, rising from 5% in initial setups to 85% in the refined modular pipeline implementation.

Despite these advancements, challenges remain. The inaccuracies in the inverse kinematics model, coupled with the need for better hyperparameter fine-tuning, indicate areas where further investigation is required. Transitioning to a more robust simulation environment and improving kinematic modelling could further enhance the precision and reliability of the system. The complexity of the environment plays a very big role too.

This work demonstrates the transformative potential of integrating affordance-based reasoning with advanced language models, such as GPT, to enhance robotic autonomy. By bridging the gap between perception, cognition, and action, the proposed framework lays the groundwork for more adaptable and intelligent robotic systems. Future research should focus on refining the framework's adaptability, automating feedback loops, and exploring end-to-end learning paradigms to enable seamless human-robot interaction and scalable task execution.

5. REFERENCES

1. Zamalloa, R. Kojcev, A. Hernández, I. Muguruza, L. Usategui, A. Bilbao, and V. Mayoral, "Dissecting robotics-historical overview and future perspectives," arXiv preprint arXiv:1704.08617, 2017.

2. Aude Billard, Danica Kragic, "Trends and challenges in robot manipulation." Science364,eaat8414(2019).DOI:10.1126/science.aat8414

3. H. Min, C. Yi, R. Luo, J. Zhu and S. Bi, "Affordance Research in Developmental Robotics: A Survey," in IEEE Transactions on Cognitive and Developmental Systems, vol. 8, no. 4, pp. 237-255, Dec. 2016, doi: 10.1109/TCDS.2016.2614992.

4. X. Yang, Z. Ji, J. Wu and Y. -K. Lai, "Recent Advances of Deep Robotic Affordance Learning: A Reinforcement Learning Perspective," in IEEE Transactions on Cognitive and Developmental Systems, vol. 15, no. 3, pp. 1139-1149, Sept. 2023, doi: 10.1109/TCDS.2023.3277288.

5. N. Wake, A. Kanehira, K. Sasabuchi, J. Takamatsu and K. Ikeuchi, "ChatGPT Empowered Long-Step Robot Control in Various Environments: A Case Application," in IEEE Access, vol. 11, pp. 95060-95078, 2023, doi: 10.1109/ACCESS.2023.3310935.

6. Y. Jin et al., "RobotGPT: Robot Manipulation Learning From ChatGPT," in IEEE Robotics and Automation Letters, vol. 9, no. 3, pp. 2543-2550, March 2024, doi: 10.1109/LRA.2024.3357432.

7. H. Ahn, S. Choi, N. Kim, G. Cha and S. Oh, "Interactive Text2Pickup Networks for Natural Language-Based Human–Robot Collaboration," in IEEE Robotics and Automation Letters, vol. 3, no. 4, pp. 3308-3315, Oct. 2018, doi: 10.1109/LRA.2018.2852786.

8. Webots. http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software.

9. https://github.com/kiyoshiiriemon/yolov4_darknet

10. https://github.com/cyberbotics/webots/blob/master/projects/robots/universal_robots/protos/UR5e.proto

APPENDIX

Figure A.1 – Experiment 3, Result = 0.5

Figure A.2 – Experiment 3, Result = 1

```
Traceback (most recent call last):
  File "C:\Users\dwara\OneDrive\Desktop\BU\Robot Learning\PleaseWorkFinalProject\PleaseWorkFinalProject\controllers\testrun\testrun.py", line 200, in <module>
    affordance, taskcohesion = GPTCallCohesion(filepath,task_in,depthimarr,objposes)
                               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\dwara\OneDrive\Desktop\BU\Robot Learning\PleaseWorkFinalProject\PleaseWorkFinalProject\controllers\testrun\testrun.py", line 169, in GPTCallCohesion
    affordancedata = jsondata['affordance_map']
                     ~~~~~~~~^^^^^^^^^^^^^^^^^^
KeyError: 'affordance_map'
```

Figure A.3 – Experiment 4, Result = 0

```
Task: grab the ball on the table
[{'obj': 'sports ball', 'position': [np.float64(-0.1059247903724183), np.float64(0.6651738882064819), np.float64(0.027373822231074388)]}]
Before RegEx
{
  "affordance_map": [
    {
      "object_name": "sports ball",
      "affordances": [
        "pick-up",
        "roll",
        "throw"
      ]
    }
  ],
  "task_sequence": [
    {
      "action": "MOVE",
      "target_position": [-0.1059247903724183, 0.6651738882064819, 0.027373822231074388]
    },
    {
      "action": "pick-up",
      "object": "sports ball"
    },
    {
      "action": "MOVE",
      "target_position": [0.0, 0.0, 0.1]
    },
    {
      "action": "place",
      "object": "sports ball"
    }
  ]
}
After RegEx
{
  "affordance_map": [
    {
      "object_name": "sports ball",
      "affordances": [
        "pick-up",
        "roll",
        "throw"
      ]
    }
  ],
  "task_sequence": [
    {
      "action": "MOVE",
      "target_position": [-0.1059247903724183, 0.6651738882064819, 0.027373822231074388]
    },
    {
      "action": "pick-up",
      "object": "sports ball"
    },
    {
      "action": "MOVE",
      "target_position": [0.0, 0.0, 0.1]
    },
    {
      "action": "place",
      "object": "sports ball"
    }
  ]
}
GPT Affordance O/P: [{'object_name': 'sports ball', 'affordances': ['pick-up', 'roll', 'throw']}]
GPT Task O/P: [{'action': 'MOVE', 'target_position': [-0.1059247903724183, 0.6651738882064819, 0.027373822231074388]}, {'action': 'pick-up', 'object': 'sports ball'}, {'action': 'MOVE', 'target_position': [0.0, 0.0, 0.1]}, {'action': 'place', 'object': 'sports ball'}]
```

Figure A.4 – Experiment 4, Result = 1 because of prompt engineering

```
Task: grab the ball on the table
[{'obj': 'sports ball', 'position': [np.float64(-0.10592231304715116), np.float64(0.6651583313941956), np.float64(0.02737318202342108)]}]
Before RegEx
{
  "affordance_map": [
    {
      "object_name": "sports ball",
      "affordances": ["pick-up", "move"]
    }
  ],
  "task_sequence": [
    {
      "action": "MOVE",
      "target_position": [-0.10592231304715116, 0.6651583313941956, 0.02737318202342108]
    },
    {
      "action": "pick-up",
      "object": "sports ball"
    },
    {
      "action": "MOVE",
      "target_position": [0, 0, 0]  // Specify the target position for placement if required
    },
    {
      "action": "place",
      "object": "sports ball"
    }
  ]
}
After RegEx
{
  "affordance_map": [
    {
      "object_name": "sports ball",
      "affordances": ["pick-up", "move"]
    }
  ],
  "task_sequence": [
    {
      "action": "MOVE",
      "target_position": [-0.10592231304715116, 0.6651583313941956, 0.02737318202342108]
    },
    {
      "action": "pick-up",
      "object": "sports ball"
    },
    {
      "action": "MOVE",
      "target_position": [0, 0, 0]     },
    {
      "action": "place",
      "object": "sports ball"
    }
  ]
}
GPT Affordance O/P: [{'object_name': 'sports ball', 'affordances': ['pick-up', 'move']}]
GPT Task O/P: [{'action': 'MOVE', 'target_position': [-0.10592231304715116, 0.6651583313941956, 0.02737318202342108]}, {'action': 'pick-up', 'object': 'sports ball'}, {'action': 'MOVE', 'target_position': [0, 0, 0]}, {'action': 'place', 'object': 'sports ball'}]
```

Figure A.5 – Experiment 4, Result = 1 because of RegEx Implementation

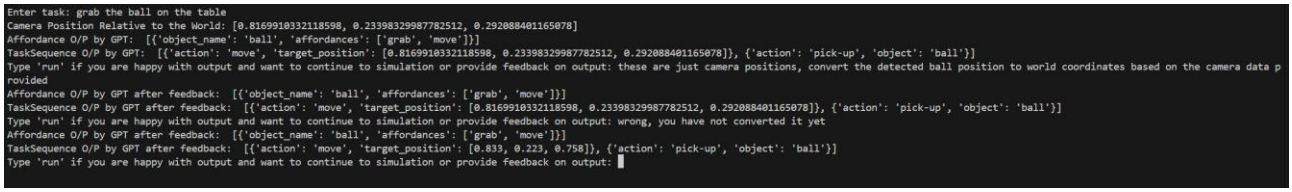

Figure A.6 – Experiment 6, Failure Case