澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

Language Models as Zero-Shot Trajectory Generators

Speaker：Xiongyi Li

E-mail: xiongyilee@outlook.com

Jun 18th 2024

Language Models as Zero-Shot Trajectory Generators

Teyun Kwon[1], Norman Di Palo[1], Edward Johns[1]

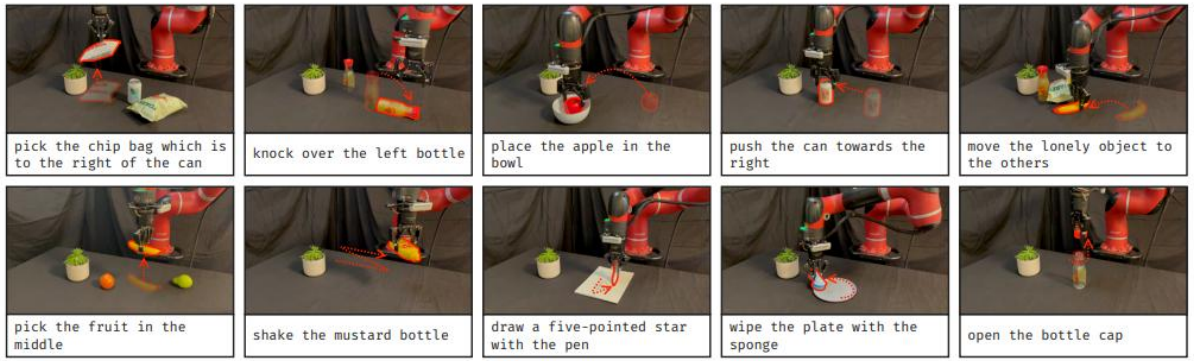


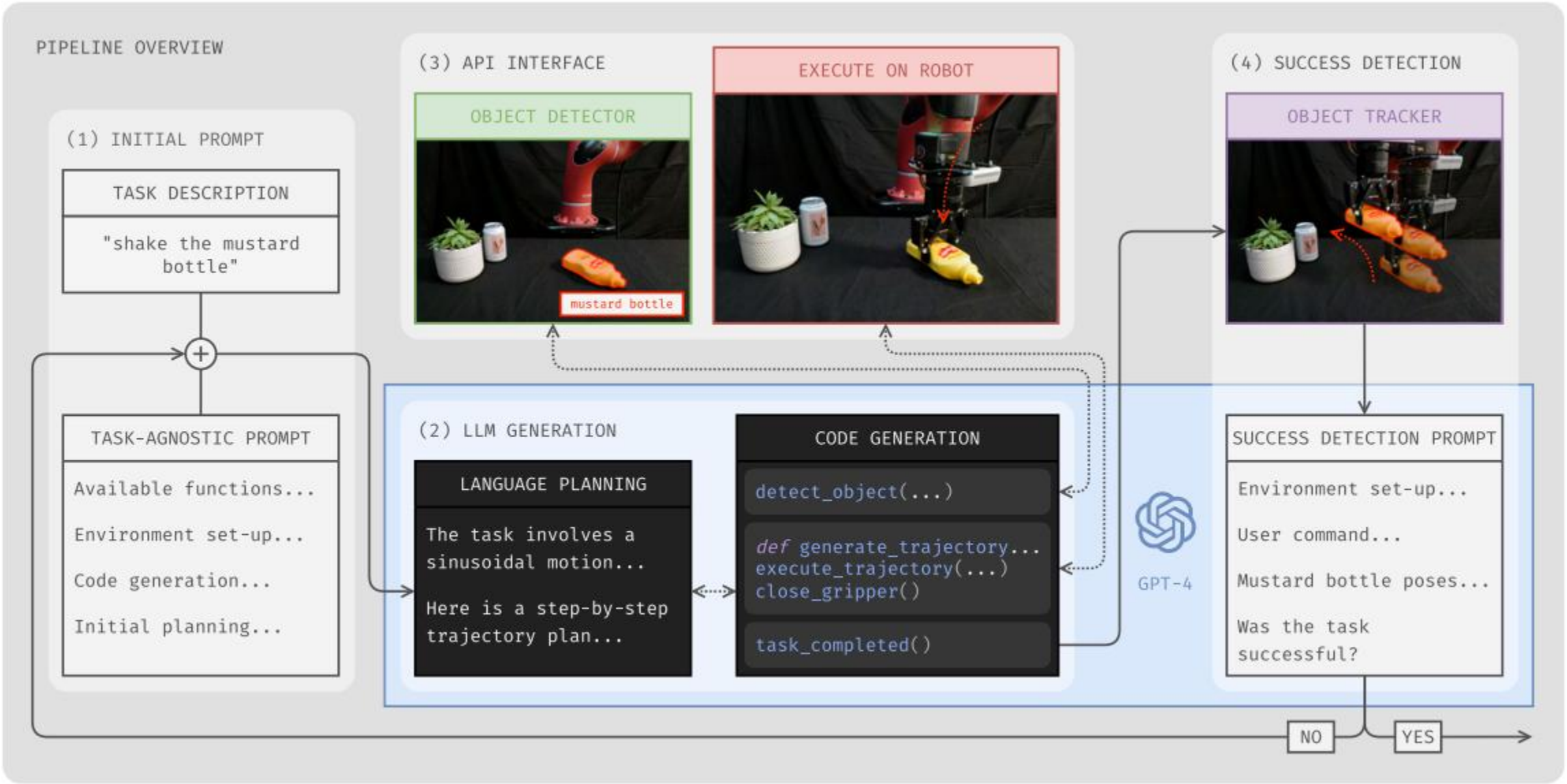
Figure 1: A selection of the tasks we use to study if a single, task-agnostic LLM prompt can generate a dense sequence of end-effector poses, when given only object detection and segmentation models, and no in-context examples, motion primitives, pre-trained skills, or external trajectory optimisers.

***Abstract*—Large Language Models (LLMs) have recently shown promise as high-level planners for robots when given access to a selection of low-level skills. However, it is often assumed that LLMs do not possess sufficient knowledge to be used for the low-level trajectories themselves. In this work, we address this assumption thoroughly, and investigate if an LLM (GPT-4) can directly predict a dense sequence of end-effector poses for manipulation tasks, when given access to only object detection and segmentation vision models. We designed a single, task-agnostic prompt, without any in-context examples, motion primitives, or external trajectory optimisers. Then we studied how well it can perform across 30 real-world language-based tasks, such as "*open the bottle cap*" and "*wipe the plate with the sponge*", and we investigated which design choices in this prompt are the most important. Our conclusions raise the assumed limit of LLMs for robotics, and we reveal for the first time that LLMs do indeed possess an understanding of low-level robot control sufficient for a range of common tasks, and that they can additionally detect failures and then re-plan trajectories accordingly. Videos, prompts, and code are available at: https://www.robot-learning.uk/language-models-trajectory-generators.**

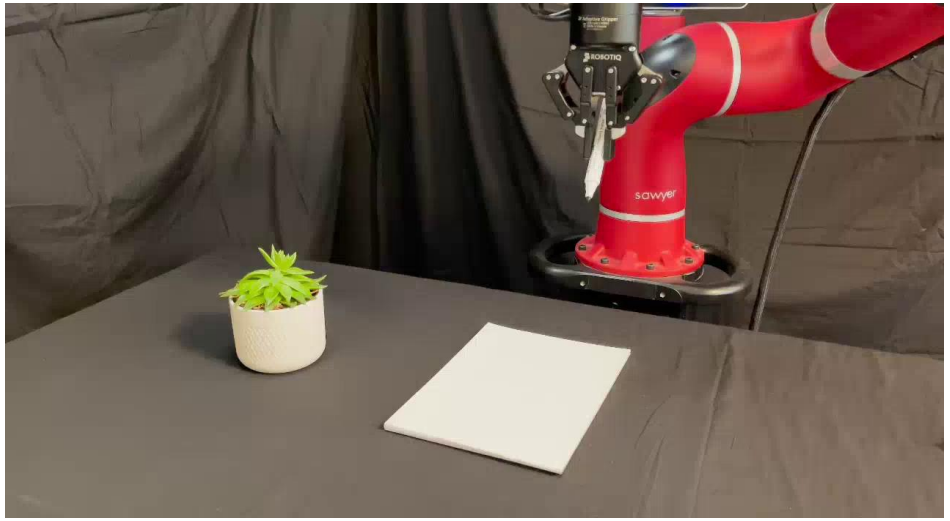
I. Introduction

In recent years, Large Language Models (LLMs) have attracted significant attention and acclaim for their remarkable capabilities in reasoning about common, everyday tasks [1]. This widespread recognition has since led to efforts in the robotics community to adopt LLMs for high-level task planning [2]. However, for low-level control, existing proposals have relied on auxiliary components beyond the LLM, such as pre-trained skills, motion primitives, trajectory optimisers, and numerous language-based in-context examples (Fig. 2). Given the lack of exposure of LLMs to physical interaction data, it is often assumed that LLMs are incapable of low-level control [3], [4], [5].

However, until now, this assumption has not been thoroughly examined. In this paper, we now investigate if LLMs have sufficient understanding of low-level control to be adopted for **zero-shot dense trajectory generation for robot manipulators**, without the need for the aforementioned auxil-

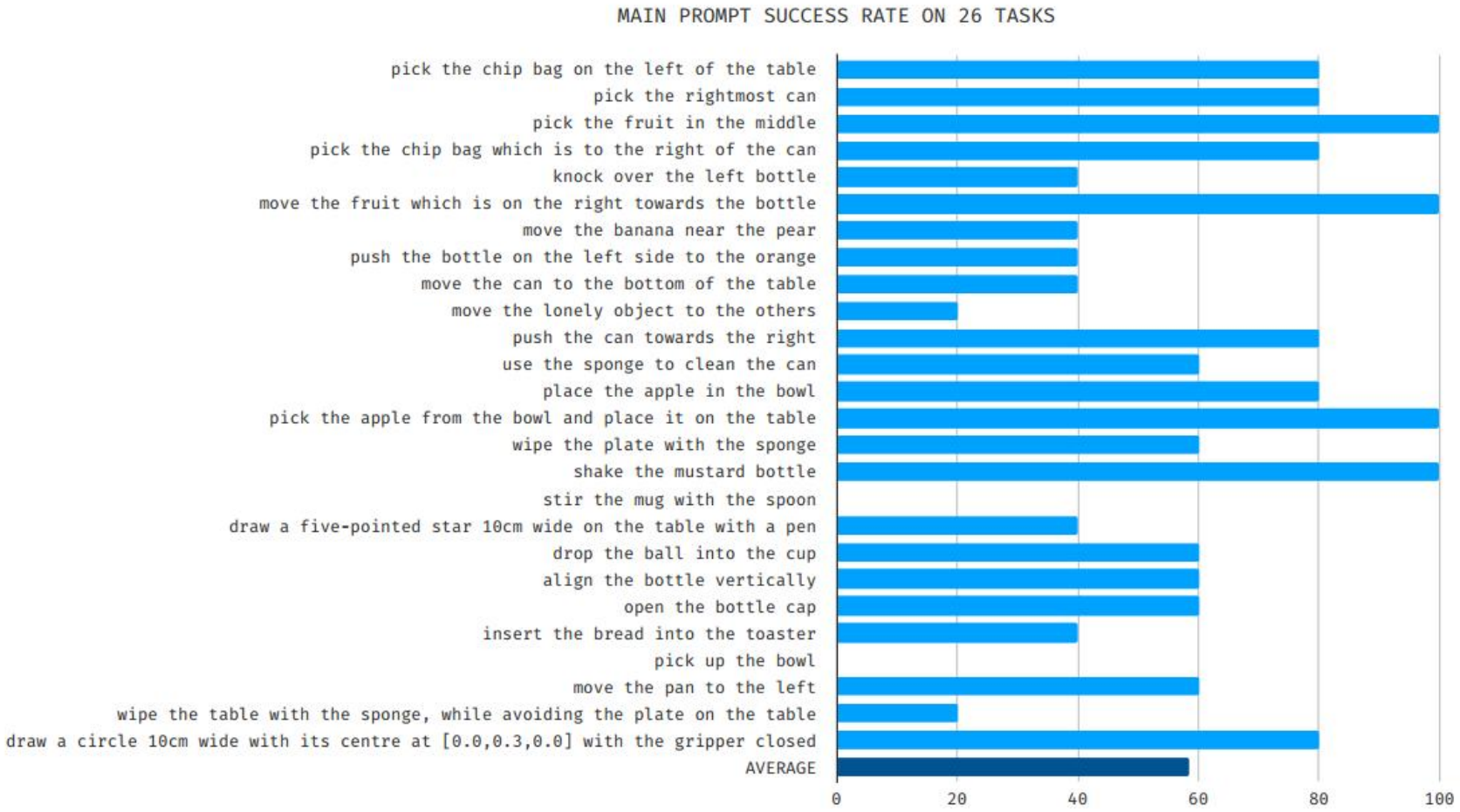

PIPELINE OVERVIEW
(1) INITIAL PROMPT
TASK DESCRIPTION
"shake the mustard bottle"
TASK-AGNOSTIC PROMPT
Available functions...
Environment set-up...
Code generation...
Initial planning...
(3) API INTERFACE
OBJECT DETECTOR
mustard bottle
EXECUTE ON ROBOT
(4) SUCCESS DETECTION
OBJECT TRACKER
(2) LLM GENERATION
LANGUAGE PLANNING
The task involves a sinusoidal motion...
Here is a step-by-step trajectory plan...
CODE GENERATION
detect_object(...)
def generate_trajectory...
execute_trajectory(...)
close_gripper()
task_completed()
GPT-4
SUCCESS DETECTION PROMPT
Environment set-up...
User command...
Mustard bottle poses...
Was the task successful?
NO
YES

Example

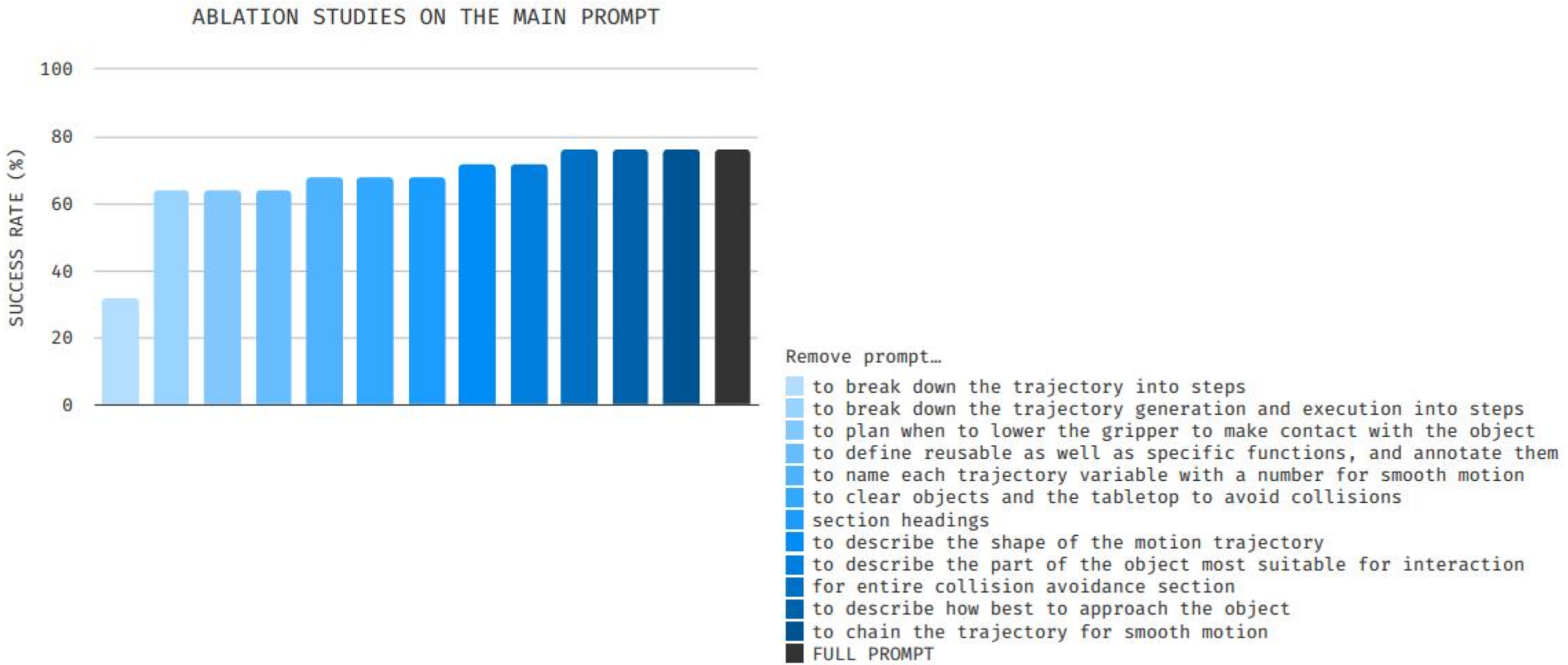


Failure cases

Experiment


MAIN PROMPT SUCCESS RATE ON 26 TASKS
pick the chip bag on the left of the table
pick the rightmost can
pick the fruit in the middle
pick the chip bag which is to the right of the can
knock over the left bottle
move the fruit which is on the right towards the bottle
move the banana near the pear
push the bottle on the left side to the orange
move the can to the bottom of the table
move the lonely object to the others
push the can towards the right
use the sponge to clean the can
place the apple in the bowl
pick the apple from the bowl and place it on the table
wipe the plate with the sponge
shake the mustard bottle
stir the mug with the spoon
draw a five-pointed star 10cm wide on the table with a pen
drop the ball into the cup
align the bottle vertically
open the bottle cap
insert the bread into the toaster
pick up the bowl
move the pan to the left
wipe the table with the sponge, while avoiding the plate on the table
draw a circle 10cm wide with its centre at [0.0,0.3,0.0] with the gripper closed
AVERAGE
0
20
40
60
80
100

Experiment


ABLATION STUDIES ON THE MAIN PROMPT
SUCCESS RATE (%)
100
80
60
40
20
0
Remove prompt…
to break down the trajectory into steps
to break down the trajectory generation and execution into steps
to plan when to lower the gripper to make contact with the object
to define reusable as well as specific functions, and annotate them
to name each trajectory variable with a number for smooth motion
to clear objects and the tabletop to avoid collisions
section headings
to describe the shape of the motion trajectory
to describe the part of the object most suitable for interaction
for entire collision avoidance section
to describe how best to approach the object
to chain the trajectory for smooth motion
FULL PROMPT

Prompt

Remove prompt...

to break down the trajectory into steps
to break down the trajectory generation and execution into steps
to clear objects and the tabletop to avoid collisions
section headings
to describe the shape of the motion trajectory
to describe the part of the object most suitable for interaction

MAIN PROMPT ABLATIONS (1)

You are a sentient AI that can control a robot arm by generating Python code which outputs a list of trajectory points for the robot arm end-effector to follow to complete a given user command.
Each element in the trajectory list is an end-effector pose, and should be of length 4, comprising a 3D position and a rotation value.

AVAILABLE FUNCTIONS:
You must remember that this conversation is a monologue, and that you are in control. I am not able to assist you with any questions, and you must output the final code yourself by making use of the available information, common sense, and general knowledge.
You are, however, able to call any of the following Python functions, if required, as often as you want:
1. detect_object(object_or_object_part: str) -> None: This function will not return anything, but only print the position, orientation, and dimensions of any object or object part in the environment. This information will be printed for as many instances of the queried object or object part in the environment. If there are multiple objects or object parts to detect, call one function for each object or object part, all before executing any trajectories. The unit is in metres.
2. execute_trajectory(trajectory: list) -> None: This function will execute the list of trajectory points on the robot arm end-effector, and will also not return anything.
3. open_gripper() -> None: This function will open the gripper on the robot arm, and will also not return anything.
4. close_gripper() -> None: This function will close the gripper on the robot arm, and will also not return anything.
5. task_completed() -> None: Call this function only when the task has been completed. This function will also not return anything.
When calling any of the functions, make sure to stop generation after each function call and wait for it to be executed, before calling another function and continuing with your plan.

ENVIRONMENT SET-UP:
The 3D coordinate system of the environment is as follows:
    1. The x-axis is in the horizontal direction, increasing to the right.
    2. The y-axis is in the depth direction, increasing away from you.
    3. The z-axis is in the vertical direction, increasing upwards.
The robot arm end-effector is currently positioned at [INSERT EE POSITION], with the rotation value at 0, and the gripper open.
The robot arm is in a top-down set-up, with the end-effector facing down onto a tabletop. The end-effector is therefore able to rotate about the z-axis, from -pi to pi radians.
The end-effector gripper has two fingers, and they are currently parallel to the x-axis.
The gripper can only grasp objects along sides which are shorter than 0.08.
Negative rotation values represent clockwise rotation, and positive rotation values represent anticlockwise rotation. The rotation values should be in radians.

COLLISION AVOIDANCE:
If the task requires interaction with multiple objects:
1. Make sure to consider the object widths, lengths, and heights so that an object does not collide with another object or with the tabletop, unless necessary.
2. It may help to generate additional trajectories and add specific waypoints (calculated from the given object information) to clear objects and the tabletop and avoid collisions, if necessary.

VELOCITY CONTROL:
1. The default speed of the robot arm end-effector is 100 points per trajectory.
2. If you need to make the end-effector follow a particular trajectory more quickly, then generate fewer points for the trajectory, and vice versa.

CODE GENERATION:
When generating the code for the trajectory, do the following:
1. Describe briefly the shape of the motion trajectory required to complete the task.
2. The trajectory could be broken down into multiple steps. In that case, each trajectory step (at default speed) should contain at least 100 points. Define general functions which can be reused for the different trajectory steps whenever possible, but make sure to define new functions whenever a new motion is required. Output a step-by-step reasoning before generating the code.
3. If the trajectory is broken down into multiple steps, make sure to chain them such that the start point of trajectory_2 is the same as the end point of trajectory_1 and so on, to ensure a smooth overall trajectory. Call the execute_trajectory function after each trajectory step.
4. When defining the functions, specify the required parameters, and document them clearly in the code. Make sure to include the orientation parameter.
5. If you want to print the calculated value of a variable to use later, make sure to use the print function

Thank You!

Avenida da Universidade, Taipa, Macau, China
Tel : (853) 8822 8833 Fax : (853) 8822 8822
Email : xiongyilee@outlook.com Website : www.um.edu.mo