Trajectory Generation for Robotic Manipulation using Large Language Models

Anindita Quraini Fal[#1], Hamas Baja Sahik Al-Jaman[#2],

Rayhan Ajie Nugraha[#3], Zinadine Zidan Alsyahana[#4]

[#]*Robotics and AI Engineering,*
Faculty of Advanced Technology and Multidiscipline,
Airlangga University
Jl. Dr. Ir. H. Soekarno, Mulyorejo, Kec. Mulyorejo, Kota SBY, Jawa Timur 60115

[1] anindita.quraini.fal-2022@ftmm.unair.ac.id

[2] hamas.baja.sahik-2022@ftmm.unair.ac.id

[3] rayhan.ajie.nugraha-2021@ftmm.unair.ac.id

[3] zinadine.zidan.alsyahana-2022@ftmm.unair.ac.id

Abstract — **Large Language Models (LLMs) have recently demonstrated potential as effective high-level planners for robotic systems, particularly when provided with a set of pre-defined low-level skills. However, it is widely assumed that LLMs lack the necessary knowledge to handle low-level trajectory generation on their own. In this work, we challenge this assumption by investigating whether an LLM, specifically GPT-4, can directly predict a dense sequence of end-effector poses for manipulation tasks, relying solely on object detection and segmentation vision models. We developed a single, task-agnostic prompt that does not incorporate in-context examples, motion primitives, or external trajectory optimization tools. Using this approach, we evaluated the model's performance across 30 real-world language-based tasks, such as "open the bottle cap" and "wipe the plate with the sponge," and analyzed which design elements of the prompt were most critical to success. Our findings expand the perceived capabilities of LLMs in robotics, revealing for the first time that LLMs possess sufficient understanding of low-level robot control to handle a variety of common tasks. Furthermore, we show that LLMs can detect task failures and autonomously re-plan trajectories as needed. Videos, prompts, and code are available at: https://www.robot-learning.uk/language-models-trajectory-generators.**

Keywords — **Large Language Models, Robot Manipulators, Trajectory Generation, ChatGPT, GPT-4**

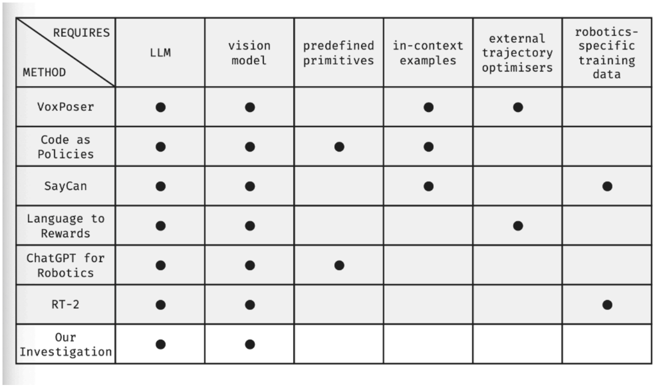
I. Background

In recent years, Large Language Models (LLMs) have garnered considerable attention and praise for their exceptional abilities to reason through common, everyday tasks [11]. This recognition has sparked interest within the robotics community, particularly in leveraging LLMs for high-level task planning [12]. However, when it comes to low-level control—such as the fine-grained manipulation of physical systems—current approaches have typically relied on additional components beyond the LLM itself (Fig. 1). These auxiliary elements include pre-trained skills, motion primitives, trajectory optimizers, and an extensive set of language-based examples provided in context. This reliance stems from the prevailing assumption that LLMs, having not been trained on physical interaction data, lack the capacity for low-level control tasks [13][14][15].

Despite this assumption, it has not been rigorously tested until now. In this paper, we challenge the notion that LLMs are inherently incapable of low-level control by exploring whether they possess sufficient understanding to generate dense trajectories for robotic manipulators in a zero-shot manner—without the need for the aforementioned auxiliary tools. Specifically, we provide an LLM (GPT-4) with access to standard object detection and segmentation models, but require that all subsequent reasoning and decision-making be handled by the LLM alone. Furthermore, we employ a task-agnostic prompt across various tasks such as "open the bottle cap" or "wipe the plate with the sponge," drawing these tasks from recent literature.

Through this investigation, we uncover key principles and strategies that enable LLMs to engage with the complexities of robot manipulation. This exploration reveals not only the potential of LLMs in low-level control but also sheds light on their ability to generalize across different tasks without relying on specialized training or additional components.

Our contributions in this work are threefold, each addressing key challenges in the intersection of large language models (LLMs) and robotic manipulation. First, we demonstrate for the first time that a pre-trained LLM, when coupled solely with an off-the-shelf object detection and segmentation model, can successfully guide a robot manipulator in a zero-shot setting. This means the LLM can generate a dense sequence of end-effector poses—essentially the detailed movements required for precise robotic manipulation—without relying on traditional auxiliary tools such as pre-trained motor skills, motion primitives, trajectory optimizers, or in-context examples. This finding challenges conventional assumptions about the limitations of LLMs in low-level control tasks and reveals their potential to autonomously handle complex physical interactions. Second, we conduct a series of ablation studies to better understand the factors that contribute to these capabilities. By systematically altering different techniques and prompt structures, we gain valuable insights into how LLMs interpret and execute task instructions, identifying the most effective strategies for enabling robust robotic control. These studies not only deepen our understanding of LLM behavior but also provide practical guidelines for optimizing their performance in future applications. Third, we explore how LLMs can not only generate trajectories but also monitor task execution by analyzing object movement across visual inputs. By tracking the trajectory of objects during manipulation, the LLM can detect when a task has failed—such as when an object is not manipulated correctly—and re-plan an alternative trajectory to correct the error. This ability to autonomously recognize failure and adapt in real-time is crucial for making robotic systems more resilient in dynamic and unpredictable environments.



REQUIRES / METHOD	LLM	vision model	predefined primitives	in-context examples	external trajectory optimisers	robotics-specific training data
VoxPoser	●	●		●	●	
Code as Policies	●	●	●	●		
SayCan	●	●		●		●
Language to Rewards	●	●			●	
ChatGPT for Robotics	●	●	●			
RT-2	●	●				●
Our Investigation	●	●				

Fig. 1 Taxonomy of requirements of LLM-based zero-shot methods from recent literature [10].

II. Objectives of Trajectory Generation of Robotic Manipulation Using LLMs

A. Translation of Natural Language to Action

Large Language Models (LLMs) interpret human language commands and directly translate them into robotic actions, removing the need for predefined motion patterns. This feature supports more flexible and intuitive responses from robots, especially useful in unpredictable settings.

B. Smooth and Efficient Motion Paths

To create smooth and natural motion paths, techniques like quintic spline and minimum-jerk trajectory generation are used. These approaches help reduce sudden movements, which is especially important for managing fragile or irregular objects.

C. Adaptable Trajectory Generation

With feedback mechanisms in place, LLMs can identify errors and adjust motion paths in real-time. This real-time adaptability enhances the robot's effectiveness in adjusting to changes during tasks, which is crucial for reliable performance in dynamic conditions.

D. Inverse Kinematics for Precision

One of the primary goals is to apply inverse kinematics to translate end-effector paths into joint movements. This ensures that each robotic joint can be precisely controlled, resulting in accurate task execution.

III. Method Explanation

In robotic manipulation, trajectory generation is a basic task that involves calculating a continuous sequence of poses or "waypoints" that the robot's end-effector (such as a tool or gripper) should follow [1]. The first step in creating robot trajectories with large language models (LLMs) like GPT-4 is to translate a high-level instruction (such "pick up the apple and place it in the bowl") into a series of exact robot actions. In conventional robotic control, this calls for a "trajectory," or set of waypoints, that gives the end-effector a continuous path to follow inside the robot's workspace. This usually entails creating Cartesian orientations and locations for every motion step. Without the use of specialized control elements or preset motion courses, the LLM has the task of directly translating natural language commands into intricate movements [2][3][4].

A. Polynomial Interpolation for Smooth Trajectories

Polynomial interpolation, specifically the use of quintic splines, is a popular method for generating smooth trajectory. Because they can meet position, velocity, and acceleration constraints at the trajectory's beginning and ending points, quintic (5th-order) polynomials are especially helpful [5].

Given an initial position $\mathbf{p}_0$ and a final position $\mathbf{p}_f$ over a time interval $t \in [0, T]$, the trajectory can be defined as:

$$\mathbf{p}(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \qquad (1)$$

where $a_1, \ldots, a_5$ are coefficients determined by boundary conditions. These conditions are:

- Initial position $\mathbf{p}(0) = \mathbf{p}_0$
- Final position $\mathbf{p}(T) = \mathbf{p}_f$
- Initial and final velocities: $\dot{\mathbf{p}}(0) = \dot{\mathbf{p}}_0$, $\dot{\mathbf{p}}(T) = \dot{\mathbf{p}}_f$
- Initial and final accelerations: $\ddot{\mathbf{p}}(0) = \ddot{\mathbf{p}}_0$, $\ddot{\mathbf{p}}(T) = \ddot{\mathbf{p}}_f$

The polynomial coefficients a_i can be found by solving a system of equations resulting from these constraints, which guarantees smooth transitions between the starting and ending locations.

B. Minimum-Jerk Trajectories

Minimum-jerk trajectories are frequently chosen for smooth, human-like motion because they reduce the rate at which acceleration (jerk) changes. This results in a trajectory function that is naturally smooth and minimizes sudden changes, making it perfect for manipulating delicate objects robotically [6].

A minimum-jerk trajectory between two points can be defined by [7]:

$$\mathbf{p}(t) = \mathbf{p}_0 + (\mathbf{p}_f - \mathbf{p}_0)\left(10\left(\frac{t}{T}\right)^3 - 15\left(\frac{t}{T}\right)^4 + 6\left(\frac{t}{T}\right)^5\right) \qquad (2)$$

where T is the motion's total duration. This polynomial function produces smooth, gradual motion by guaranteeing zero acceleration and velocity at the start and finish of the trajectory.

C. Inverse Kinematics for Joint Configuration

We solve an inverse kinematics issue to convert end-effector positions into joint angles [8]. We calculate the joint angles $\mathbf{q}(t)$ for every waypoint $\mathbf{p}(t)$ on the intended path so that:

$$\mathbf{f}(\mathbf{q}(t)) = \mathbf{p}(t) \tag{3}$$

where f is the forward kinematics function of the robot. Finding $\mathbf{q}(t)$ for given $\mathbf{p}(t)$ is the task of solving this equation. This can be difficult, particularly for redundant manipulators that have numerous solutions for $\mathbf{q}(t)$.

D. Generating Code and Error Detection

The trajectory is defined by the Python code snippets that the LLM produces. A function to move an object along a minimum-jerk path, for instance, might resemble this:

```python
def generate_minimum_jerk_trajectory(p0, pf, T, t):
    s = 10 * (t/T)**3 - 15 * (t/T)**4 + 6 * (t/T)**5
    return p0 + s * (pf - p0)
```

By tracking feedback on the object's position and orientation, the LLM not only creates the motion sequence for each action but also detects errors. When an error is detected, such as when an object was not grasped correctly, the LLM modifies the approach angle or recalculates the required waypoints [9].

Practical Example: Pick-and-Place Task

Using a minimum-jerk trajectory, the LLM would first define an approach path to the object (like an apple) in a pick-and-place job. Assume that the goal (apple) is at $(0.5, 0.5, 0.2)$ with $T = 1$ seconds and that the starting position is $(0, 0, 0)$. The trajectory from eq. 2 with the least amount of jerk to get to the apple would be:

$$\mathbf{p}(t) = (0,0,0) + (0.5,0.5,0.2)\left(10\left(\frac{t}{1}\right)^3 - 15\left(\frac{t}{1}\right)^4 + 6\left(\frac{t}{1}\right)^5\right)$$

After calculating $\mathbf{p}(t)$ for $t \in [0, T]$, the LLM would use inverse kinematics to convert this route into joint angles.

Once the apple was grasped, the LLM would create a trajectory to take it to a bowl, once more utilizing a smooth path, and then open the gripper to release the fruit.

IV. Algorithm

The following is the algorithm used in zero-shot trajectory generation for robotic manipulation with LLM [10].

Given: full task-agnostic prompt p, and task instruction *l*

```
1:  prompt ← p ⊕ l                                  ▷ concatenate
2:  task_completed ← False
3:  while task_completed = False do
4:      output ← LLM(prompt)
5:      prompt ← None
6:      if output contains code then               ▷ extract code
7:          try exec(output)
8:          except Exception then
9:              prompt ← error message
10:         else                                    ▷ predefined APIs
11:             if detect object(object) is called then
12:                 pos, orn, dim ← detect object(object)
13:                 prompt ← pos, orn, dim
14:             else if execute trajectory(t) is called then
15:                 move robot(t) ▷ t is list of poses
16:             else if task completed() is called then
17:                 obj_poses ← XMem(traj_frames)
18:                 task_completed ← LLM(obj_poses)
19:                 if task_completed = False then
20:                     reset environment()
21:                     summary ← LLM(obj_poses)
22:                     prompt ← p ⊕ l ⊕ summary
23:                 end if
24:             end if
25:         end try
26:     end if
27: end while
```

This algorithm is designed for zero-shot trajectory generation in robotic manipulation tasks using large language models (LLMs). The algorithm begins by constructing a prompt for the LLM by combining a generic, task-agnostic prompt *p* p with task-specific instructions *l*. The goal is iteratively executed by asking the LLM to generate code for robotic manipulation until the task is complete. Initially, a flag (*task_completed*) is set to False, indicating that the task is still in progress. The main loop then runs, and the LLM generates output based on the prompt. If the output contains executable code, it is executed, with any errors detected and fed back as an updated prompt. The updated prompt ensures that the LLM learns from the previous context, allowing it to refine its subsequent responses.

Within this loop, the algorithm includes specific control statements that manage various robotic functions. For example, if the LLM calls *detect_object(object)*, it extracts the position, orientation, and dimensions of the object, feeding them back into the prompt for the updated context. Similarly, if the code generated by the LLM calls *execute_trajectory(t)*, a list of pose trajectories *t* t is passed to the robot to start the movement. Another check is for the *task_completed()* function, which uses memory frames (XMem*(traj_frames)*) to analyze whether the task is complete by verifying the object positions. If the LLM response indicates the task is still incomplete, the environment is reset, ensuring conditions are set for another round of execution. This iterative process helps the LLM generate accurate trajectory actions in real time, guided by feedback and environmental updates until the robotic task is fully completed.

A flowchart for this algorithm would begin by combining the task-agnostic prompt *p* with the task-specific instruction *l* to form an initial prompt. This input is then fed into the LLM in a loop. If the LLM's output includes code, it is executed, with any errors handled by updating the prompt. Branches represent specific robotic commands: detecting objects, executing trajectories, and checking if the task is complete. If the task remains incomplete, the environment resets, and a new prompt is generated with feedback. This loop repeats until the task is successfully completed, marking the end of the process.

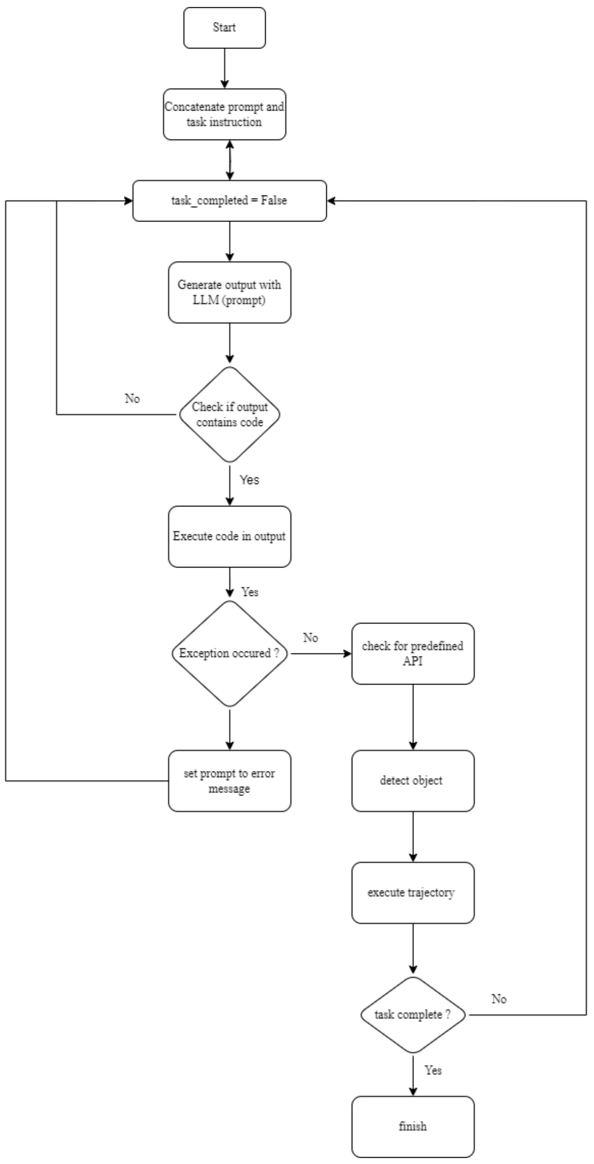


Fig. 2 Flowchart of the Zero-Shot Trajectory Generation Algorithm for Robotic Manipulation Using LLM.

V. Conclusion

In this paper, we explored the use of Large Language Models (LLMs), specifically GPT-4, for generating robotic manipulation trajectories without relying on additional tools like pre-trained motion primitives or trajectory optimizers. By using object detection and segmentation models, we showed that GPT-4 could generate accurate end-effector poses from natural language commands, allowing robots to perform tasks such as "open the bottle cap" or "wipe the plate with the sponge." Our findings challenge the assumption that LLMs are not suitable for low-level control tasks, showing that they can handle these tasks directly. We also conducted ablation studies to identify key factors that influence the performance of LLMs in robotic control, such as prompt structure. Furthermore, we demonstrated that LLMs can monitor task execution and re-plan trajectories if needed, which helps improve their ability to adapt in real-time to changes or failures. This work opens up new possibilities for using LLMs in robotics, making systems more flexible and autonomous. Further research could focus on refining these methods and applying them to more complex tasks to fully realize the potential of LLMs in real-world robotic applications.

References

[1] R. Zhao, “Trajectory planning and control for robot manipulations,” *Université Paul Sabatier - Toulouse III*, 2015, Accessed: Nov. 02, 2024. [Online]. Available: https://theses.hal.science/tel-01285383/.

[2] J. Wang *et al.*, “Large Language Models for Robotics: Opportunities, Challenges, and Perspectives,” *arXiv preprint*, 2024, doi: https://arxiv.org/abs/2401.04334v1.

[3] H. Yao, Z. Song, Y. Zhou, T. Ao, B. Chen, and L. Liu, “MoConVQ: Unified Physics-Based Motion Control via Scalable Discrete Representations,” *ACM Transactions on Graphics*, vol. 43, no. 4, pp. 1–21, 2024, doi: https://doi.org/10.1145/3658137.

[4] Y. Mikami, A. Melnik, J. Miura, and V. Hautamäki, “Natural Language as Policies: Reasoning for Coordinate-Level Embodied Control with LLMs,” *arXiv preprint*, 2024, doi: https://arxiv.org/abs/2403.13801v2.

[5] K. Erkorkmaz and Y. Altintaş, “Quintic Spline Interpolation With Minimal Feed Fluctuation,” *Journal of Manufacturing Science and Engineering-transactions of The Asme*, vol. 127, no. 2, pp. 339–349, 2005, doi: https://doi.org/10.1115/1.1830493.

[6] Y. Fang, J. Qi, J. Hu, W. Wang, and Y. Peng, “An approach for jerk-continuous trajectory generation of robotic manipulators with kinematical constraints,” *Mechanism and Machine Theory*, vol. 153, p. 103957, 2020, doi: https://doi.org/10.1016/j.mechmachtheory.2020.103957.

[7] T. Flash and N. Hogan, “The coordination of arm movements: an experimentally confirmed mathematical model,” *The Journal of Neuroscience*, vol. 5, no. 7, pp. 1688–1703, 1985, doi: https://doi.org/10.1523/jneurosci.05-07-01688.1985.

[8] C. Schumacher, E. Knoop, and M. Bacher, “A Versatile Inverse Kinematics Formulation for Retargeting Motions Onto Robots With Kinematic Loops,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 943–950, 2021, doi: https://doi.org/10.1109/lra.2021.3056030.

[9] D. Li *et al.*, “What Foundation Models can Bring for Robot Learning in Manipulation : A Survey,” *arXiv preprint*, 2024, doi: https://doi.org/10.48550/arxiv.2404.18201.

[10] T. Kwon, N. D. Palo, and E. Johns, “Language Models as Zero-Shot Trajectory Generators,” *arXiv preprint*, 2023, doi: https://doi.org/10.48550/arxiv.2310.11604.

[11] W. X. Zhao et al., “A Survey of Large Language Models,” *arXiv preprint*, 2023, doi: https://doi.org/10.48550/arXiv.2303.18223.

[12] L. Wang et al., “A Survey on Large Language Model based Autonomous Agents,” *arXiv preprint*, 2023, doi: https://doi.org/10.1007/s11704-024-40231-1.

[13] W. Huang et al., “VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models,” *arXiv preprint*, 2023, doi: https://doi.org/10.48550/arXiv.2307.05973.

[14] W. Yu et al., “Language to Rewards for Robotic Skill Synthesis,” *arXiv preprint*, 2023, doi: https://doi.org/10.48550/arXiv.2306.08647.

[15] M. Ahn et al., “Do As I Can, Not As I Say: Grounding Language in Robotic Affordances,” *arXiv preprint*, 2022, doi: https://doi.org/10.48550/arXiv.2204.01691.