LLM Driven 3DoF Robot Arm Control in 3D space

https://github.com/CMU-18475-18675/team3

***Abstract*—The application of Large Language Model (LLM) to the robotics field has shown promising results. The LLM provides high level instructions, while robot arm needs low level commands to move successfully. Our work builds the gap between LLM and low level commands by using descriptor files for translation, and achieves convincing results in a 3D simulator built by ourselves. We also use rapidly exploring random tree (RRT) for high level trajectory planning and linear quadratic regulator (LQR) for low level control.**

***Index Terms*—Large Language Model, Robot Arm, Rapidly Exploring Random Tree, Linear Quadratic Regulator**

I. Introduction

THE use of Large Language Models (LLMs) has achieved significant attention in recent research in the field of robotics. This emerging paradigm, as an alternative to the conventional deterministic models, provides considerable advantages given the continuous advancements and accessibility of machine learning techniques. LLMs have provided solutions to a wide variety of robotics problems in many different contexts. Levine et al. (2) employed deep reinforcement learning using LLMs for grasping and manipulation tasks with a PR2 robot. A distinctive aspect was the randomized method of data collection from real-world operations which demonstrated high robustness to object variations. In another study, Kaiser et al. (3) used LLMs for world model learning for autonomous drones operating in complex, dynamic environments. They demonstrated a model that was able to plan and control a quadrotor's flight with limited sensor information.

While LLMs offer numerous advantages, there are several challenges associated with their use. One of the major challenges is the discrepancy between the real world and the simulated environments where these models are typically trained. Rusu et al. (4) proposed a method that combines real-world data and simulation for more effective learning. Another challenge lies in the interpretability of these models. In the case of robotics, operators often need to understand how the model is making decisions. While LLMs often operate as black-box models, Watter et al. (5) developed a variant of LLMs that maintained interpretability by constraining the latent space to a size that can be manually inspected.

In this project, we try to demonstrate the effectiveness of using LLM to enable robot arm manipulation. It is widely recognized high level LLM instructions cannot be directly converted into low level commands. We fill this gap by using descriptor and encoder file to successfully enable robot arm manipulation such as fetch an apple to the target position. Our environment setup and system architecture are illustrated in the subsequent sections.

II. Starter Framework Limitations

A. Limited applications

The robot arm in the starter framework is a 2DoF one in a 2D space. Though it is easy to analyze, its workspace is limited, resulting in restricted functionality.

B. Limited interaction with objects

The robot arm can only recognize several specific objects and execute commands that only involve interactions between the few objects.

C. No obstacle avoidance

The starter framework does not provide obstacles, and the robot arm has no obstacle avoidance functionality. In complex physical environment, manipulators always have to avoid collision with obstacles to get away from potential damage.

D. No delicate planner or controller

The starter framework only involves a distance based planner. It does not support fine-grained path planning or speed and acceleration control, which are critical for real life tasks.

III. Environment Setup

A. 3DoF manipulator

We employ a 3DoF articulated manipulator, which is widely applied in production and industry. Figure 1 shows its structure, parameters and work space (1). It can rotates with 2DoF around the joint on the base, and with 1DoF around the joint on the arm. We believe this manipulator can greatly leverage the applications of our framework.

B. Supported objects

Our simulation environment supports balls, cubes and boxes with unique names. The user can also define own objects by extending the *Object3D* class. Note that boxes are special cubes that can only be entered by the end-effector from the top. The environment also supports setting objects as obstacles, which will be avoided when conducting path planning. One example environment is shown in Figure 2.

IV. Individual Components

A. LLM

Converting human language into actionable commands for robots has the potential to revolutionize the field of robotics. However, the direct application of LLMs to control robotic arms using low-level commands presents significant challenges, primarily due to the scarcity of relevant training data.

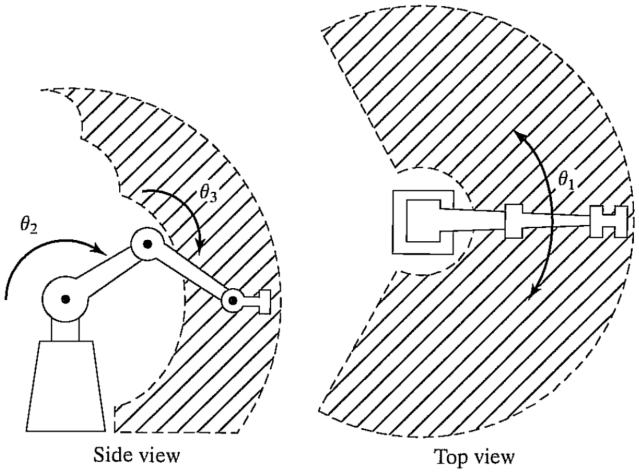


Fig. 1. 3Dof articulated manipulator

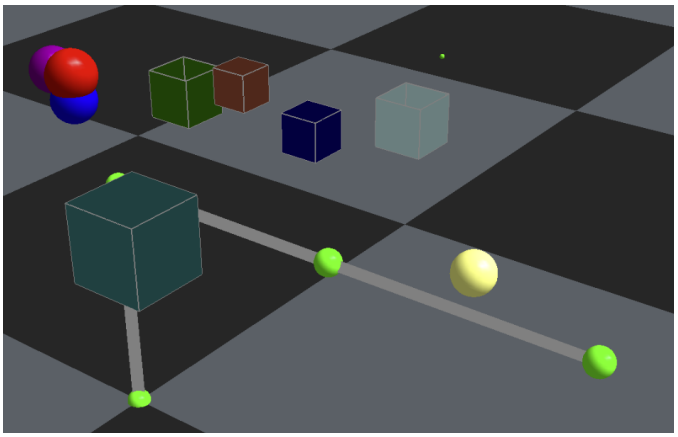


Fig. 2. Environment setup

To tackle these issues, we have developed a two-stage translation system that first interprets and fine-tunes human inputs before generating precise robotic commands. Our approach has been informed by the insights of Wenhao Yu et el. (6).

1) Stage One: Descriptor: The initial stage of our system introduces a key component named the Descriptor. This module plays a crucial role in interpreting user inputs and converting them into a structured natural language description that aligns with the desired robot motion. Initially, we present a general scenario to the LLM, detailing our task environment, requirements, and objectives. To maximize the Descriptor's efficiency, we have integrated a set of guiding rules that also act as compensatory mechanisms for our motion templates. These rules leverage the LLM's innate understanding of motion, ensuring the generation of structured and predictable outputs that enhance the stability of the entire system.

Moreover, we utilize a predefined template to maintain consistency and effectively handle complex or irregular inputs that may pose processing challenges for the LLM. Additionally, we integrate examples of test inputs alongside their expected outputs, further refining our process. This integration helps in fine-tuning the Descriptor, enabling it to consistently produce accurate and reliable outputs. This methodical approach not

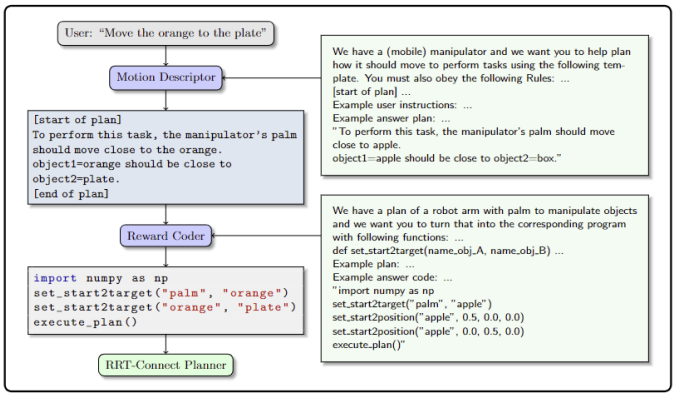


Fig. 3. The two-stage commands translator.

only improves the Descriptor's functionality but also ensures that our system can reliably translate human directives into precise robotic actions.

2) Stage Two: Coder: The second stage involves translating the motion descriptions generated by the Descriptor into specific robotic commands using another LLM, referred to as the Coder. We crafted a specialized prompt designed to guide the Coder in generating precise control commands appropriate for the given virtual situation. The prompt is composed of three elements:

- Constraints and rules that the Coder must adhere to, ensuring that the command parameters are chosen based on the Coder's understanding of the motion derived from the natural language description.
- A description of the command APIs that the LLM can use to specify parameters of the command function.
- Example responses that demonstrate the expected output, which aids the LLM in understanding the format and specificity required but does not teach it how to perform any particular task.

This structured approach allows the Coder to apply its knowledge of motion descriptions to generate accurate and functional robotic commands, thereby bridging the gap between human linguistic input and robotic action. Through this dual-stage translation system, we mitigate the limitations posed by the scarcity of direct training data for LLMs in robotic command generation, paving the way for more intuitive and interactive robot control interfaces in the future.

B. RRT-Connect planning

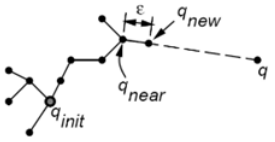
The implementation of the rapidly exploring random tree (RRT)-Connect algorithm for path planning in our robotic arm project encounters unique challenges arising from the specific requirements and constraints of our system:

- Spatial Configuration Considerations
  The necessity to convert target positions from Cartesian coordinates (x, y, z) to the configuration space required for RRT planning to ensure optimal performance.
- Obstacle Projection Complexity
  Difficulty in projecting obstacles from Cartesian coordinates to configuration space, compounded by the dynamic nature of different task environments.

```
RRT_CONNECT_PLANNER(q_init, q_goal)
 1  T_a.init(q_init); T_b.init(q_goal);
 2  for k = 1 to K do
 3      q_rand ← RANDOM_CONFIG();
 4      if not (EXTEND(T_a, q_rand) = Trapped) then
 5          if (CONNECT(T_b, q_new) = Reached) then
 6              Return PATH(T_a, T_b);
 7      SWAP(T_a, T_b);
 8  Return Failure
```

```
CONNECT(T, q)
 1  repeat
 2      S ← EXTEND(T, q);
 3  until not (S = Advanced)
 4  Return S;
```

```
EXTEND(T, q)
 1  q_near ← NEAREST_NEIGHBOR(q, T);
 2  if NEW_CONFIG(q, q_near, q_new) then
 3      T.add_vertex(q_new);
 4      T.add_edge(q_near, q_new);
 5      if q_new = q then
 6          Return Reached;
 7      else
 8          Return Advanced;
 9  Return Trapped;
```



EXTEND *operation*

Fig. 4. Pseudo code of a basic RRT algorithm

- Task Complexity
  The inherent complexity of some tasks can affect the speed and efficiency of the algorithm, necessitating adjustments to maintain performance.

To overcome these challenges, based on the classic RRT-Connect algorithm (see Figure4), we proposed the following modifications:

- Use of Inverse Kinematics for Targets
  We utilize inverse kinematics to transform target positions into a set of target points within configuration spaces. The RRT-Connect is then executed for each target, but if a target is not found within set iterations, the target tree is deleted, retaining the start tree for the next target points' search. This approach, while not as efficient as multi-target searching, is viable due to our robot arm design, which limits to at most two target configuration points, keeping costs acceptable.
- Collision Checking Enhancements
  Instead of performing inverse kinematics for every obstacle (which would be time-consuming and resource-intensive), we perform a collision check for every configuration point generated. To ensure that intermediate states between configuration points are also obstacle-free, we set the step distance (ϵ) to be very small. This meticulous setting helps in guaranteeing that collisions are virtually impossible.
- Performance Optimization
  Balancing performance, speed, and collision avoidance, we tested different ϵ values. It was determined that an ϵ value of 0.025 offers the best performance, reliably generating collision-free paths under all conditions and capable of solving multiple complex inputs in approximately 5 seconds.

These modifications ensure that the RRT-Connect algorithm is well-suited to the specific demands and conditions of our robotic arm project, providing efficient and reliable path planning even in complex environments.

C. LQR control

The LQR control is applied to ensure smooth path point following. The state space representation of the system is

$$\boldsymbol{x} = [\boldsymbol{\theta} - \boldsymbol{\theta_{ref}}, \dot{\boldsymbol{\theta}} - \dot{\boldsymbol{\theta}}_{\boldsymbol{ref}}]^T$$
$$\boldsymbol{u} = \ddot{\boldsymbol{\theta}}^T$$

where each vector is 3-dimensional, and ref means the control target:

$$\boldsymbol{\theta} := [\theta_1, \theta_2, \theta_3]$$
$$\dot{\boldsymbol{\theta}} := [\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3]$$
$$\ddot{\boldsymbol{\theta}} := [\ddot{\theta}_1, \ddot{\theta}_2, \ddot{\theta}_3]$$
$$\boldsymbol{\theta_{ref}} := [\theta_{1ref}, \theta_{2ref}, \theta_{3ref}]$$
$$\dot{\boldsymbol{\theta}}_{\boldsymbol{ref}} := [\dot{\theta}_{1ref}, \dot{\theta}_{2ref}, \dot{\theta}_{3ref}]$$

The 1-dimensional physical expressions are

$$\theta_{t+1} = \theta_t + \dot{\theta}_t \Delta t$$
$$\dot{\theta}_{t+1} = \dot{\theta}_t + \ddot{\theta}_t \Delta t$$

Trivial transformation gives

$$\theta_{t+1} - \theta_{ref} = \theta_t - \theta_{ref} + (\dot{\theta}_t - \dot{\theta}_{ref})\Delta t + \dot{\theta}_{ref}\Delta t$$
$$\dot{\theta}_{t+1} - \dot{\theta}_{ref} = \dot{\theta}_t - \dot{\theta}_{ref} + \ddot{\theta}_t \Delta t$$

The 1-dimensional case can be applied in 3-dimensional space without loss of generality:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \begin{bmatrix} \dot{\boldsymbol{\theta}}_{\boldsymbol{ref}} \cdot \mathbf{\Delta t} \\ \mathbf{0} \end{bmatrix}$$

where

$$\mathbf{I} := Identity(3)$$
$$\mathbf{0} := Zeros(3)$$
$$\mathbf{\Delta t} := \Delta t \cdot \mathbf{I}$$
$$\mathbf{A} := \begin{bmatrix} \mathbf{I} & \mathbf{\Delta t} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$
$$\mathbf{B} := \begin{bmatrix} \mathbf{0} \\ \mathbf{\Delta t} \end{bmatrix}$$

In order to match the standard state representation for LQR, choose $\dot{\boldsymbol{\theta}}_{\boldsymbol{ref}} = 0$, current state space representation becomes

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$

Use LQR to solve for optimal control, where cost is

$$J := \sum_{k=0}^{\infty} (\mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k + 2\mathbf{x}_k^T \mathbf{N} \mathbf{u}_k)$$

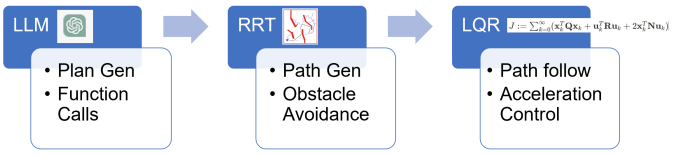


Fig. 5. Global architecture

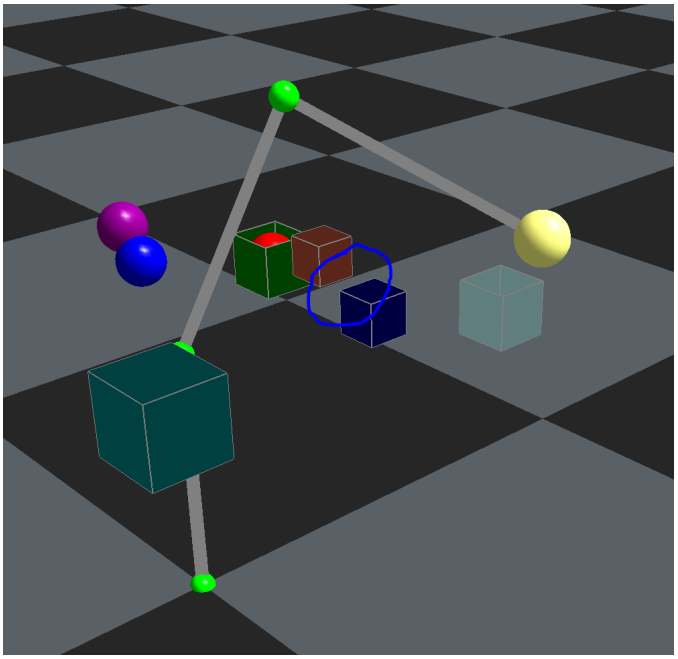


Fig. 6. Narrow gap between obstacles

V. Global Architecture

The individual components mentioned in last section are linked in a pipeline, as shown in Figure 5. The LLM is responsible for generating the manipulation plan and make corresponding function calls. For example, when user wants to *move apple to box*, the LLM will generate function calls that move the end-effector to apple, and move the apple to box respectively. The RRT-Connect takes care of generating a series of obstacle-avoiding path points in joint angle space. The LQR controller then control the manipulator to follow the path points, with the advantage of avoiding large angular speeds and accelerations. Once the manipulator is close enough in angle space to the current path point, the LQR controller will switch its target to the next path point.

VI. Simulation & Results

Our simulation program can constantly deal with user inputs in one run, and allows simulation with or without LQR control, governed by the *–enbale-lqr* command line argument.

A. Supported inputs

From simulation, our program features moving objects to another object or by certain translation, and it can process multiple tasks in one user input. It supports user inputs like:

- move apple to box

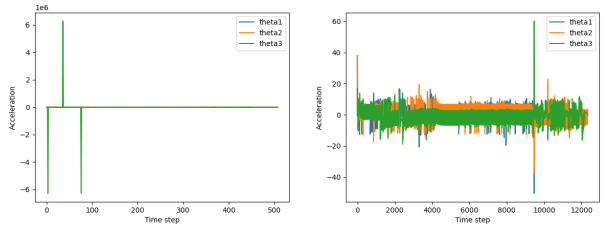


Fig. 7. Accelerations with (right) / without (left) LQR

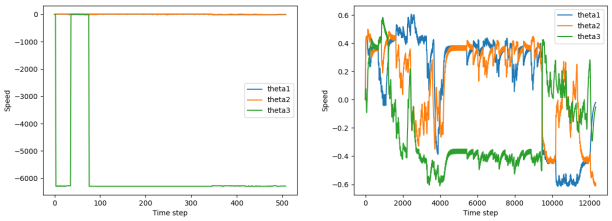


Fig. 8. Speeds with (right) / without (left) LQR

- move banana to target
- move apple up by 0.5
- move apple to target, banana to box

B. Object avoidance

When executing any task, the whole robot arm can void collision with any obstacle (the arm is assumed with no width, while the arm shown in the environment has width, for better visibility). It can even pass through narrow gaps between obstacles, as shown in Figure 6.

C. With / Without LQR

LQR has played an important role in constraining angular speeds and accelerations. Angular accelerations will explode to million without LQR, and angular speeds to -6000, as depicted in Figure 7 and Figure 8. The only side effect of LQR control is that the manipulator needs to take longer to reach its target, because its accelerations and speeds are constrained. Since the manipulator always follows RRT-Connect planned path, the angles with or without LQR demonstrates similar patterns, as shown in Figure 9.

VII. Conclusion

In conclusion, the deployment of LLMs in robotics opens up exciting possibilities and research avenues. This report has demonstrated its applicability and effectiveness by successfully bridging the gap between high-level instructions provided by the LLM and the low-level commands required for physical control. The combination of RRT-connect and LQR further augmented this process, presenting a comprehensive approach from planning to execution. Future work can explore refining these algorithms, expanding the range of tasks, and potentially moving from the simulated environment to real-world applications.

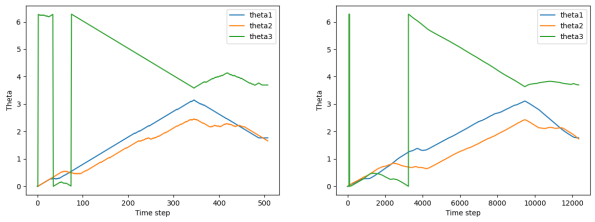


Fig. 9. Angles with (right) / without (left) LQR

REFERENCES

[1] John J. Craig. *Introduction to Robotics: Mechanics and Control*, Third Edition. Pearson Education International (2005).

[2] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J. and Quillen, D. *Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection*, The International Journal of Robotics Research,37(4-5), 421-436 (2018).

[3] Kaiser, L., Babuschkin, I., Milosevic, M., Osindero, S., Sepassi, R., Heess, N. and Hadsell, R. *Model-Based Reinforcement Learning for Atari*, ArXiv, abs/1903.00374 (2019).

[4] Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R. and Hadsell, R. *Sim-to-Real Robot Learning from Pixels with Progressive Nets*, ArXiv, abs/1610.04286 (2017).

[5] Watter, M., Springenberg, J. T., Boedecker, J., and Riedmiller, M. *Embed to control: A locally linear latent dynamics model for control from raw images*, In Advances in neural information processing systems (2015).

[6] Wenhao, Y. et al. *Language to rewards for robotic skill synthesis* arXiv preprint arXiv:2306.08647 (2023).

[7] Saraf, Prathamesh and R N, Ponnalagu. *Modeling and Simulation of a Point to Point Spherical Articulated Manipulator using Optimal Control*, In 7th International Conference on Automation, Robotics and Applications (ICARA) (2021).