Simulated Kitchen Robot Solving Long-horizon Manipulation Tasks with Short Action Library

Dingkun Guo[1], Xiaofeng Guo[1], and Yunchu Zhang[1]

***Abstract*— We proposed a two stage optimization method for learning to solve long horizon manipulation tasks in a simulated kitchen environment in Mujoco. Our key innovation is to decompose the long horizon manipulation tasks into several short horizon manipulation actions and track those short actions' trajectories with Time-varying Linear Quadratic Regulator (TVLQR). Then we chain the optimal actions with transitions to get a whole reference trajectory. By utilizing nonlinear trajectory optimization – iterative Linear Quadratic Regulator (iLQR), we obtained the optimal controller and smoother trajectories of long horizon manipulation tasks. Our key ideas in this work are: (1) building up a series of short horizon primitives based on expert demonstrations such as opening a microwave, turning on a switcher, and lifting and placing the kettle, and (2) chaining short actions' trajectories with smooth transitions, then utilizing iLQR to smooth the whole trajectory to solve the long horizon manipulation tasks. In our experiments, we show that our method has smaller cost and robust performance compared to PID controller and open loop controller. Our code is available online[2].**

I. INTRODUCTION

The unprecedented advancement of artificial intelligence and automation systems has witnessed an increasing amount of popularity enjoyed by home robots when people need to do repeated housework every day. In particular, in the kitchen, we hope there could be a robot helping us prepare food and clean up plates. Unfortunately, such long-horizon manipulation tasks are still too complicated for robots to solve. In this case, we choose to investigate a divide-and-conquer method that breaks down long-horizon manipulation tasks into short actions, and apply existing methods to get optimized results. We show that by chaining those short actions, our robot can solve multiple tasks with the state-of-the-art performance.

Admittedly, with purely imitation learning or reinforcement learning frameworks, robots have shown some abilities to adapt to different environment and learn various skills. Due to the unknown intermittent contact dynamics and non-exist local minimal in multi-stage, however, those methods in general have unsatisfactory results solving long sequential trajectories. The traditional direct or indirect nonlinear optimization methods have shown their suffering in generalization ability. Additionally, in the real world, the existing learning methods also fails quickly because of insufficient data and noise and randomness in exploration and optimization, not to mention adaptive to different complicated real kitchen scenarios.

To solve complicated tasks, people usually inherently divide them into smaller tasks. Instead of trying to learn or optimize on the long sequential tasks' action commands directly from scratch, we formulate our solution as firstly building up a library of primitives for short horizon manipulation tasks and then try to chain them together and utilize the chained states and actions trajectories as Xref, Uref input to the Iterative Linear Quadratic Regulator (iLQR) algorithm for smoothing the whole trajectory and skill. And we do locally linearization around each knot points in the full trajectory for getting the dynamics jacobian matrix A and B, which will be used for iLQR's calculation.

In this work, we combine the library of primitive's [7] idea and classical iLQR [5] technology to acquire optimal and robust long sequential skills for manipulation. Through extensive qualitative and quantitative experiments, we demonstrate the effectiveness and the generalizability of the proposed framework.

Our main contributions in this project are:

1) Building up a primitive library contains a set of distinct short horizon behaviors each of which operates for a different manipulation skill.
2) Constructing a generalizable framework for learning and optimizing about the long horizon manipulation skills by chaining a series of short horizon primitives.
3) Proposing an efficient pipeline for collecting expert demonstrations and acquire short horizon skills with the help of Linear Quadratic Regulator (LQR).
4) Utilizing a numerical way to calculate Jacobian matrices in Mujoco simulator during the linearization process.

II. RELATED WORKS

Recent years, there have been a lot of successful reinforcement learning (RL) related algorithms that tries to solve complicated manipulation tasks such as grasping and door rubik's cube solving [1]. However, those applications are constrained to relatively short-horizon skills. Hierarchical reinforcement learning (HRL) [2] has been proposed to solve the long horizon task by introducing a hierarchy of subtasks such that higher level tasks' action outputs will become the subgoals of low level tasks. However, HRL methods usually have the same issue as vanilla RL methods have, such as the challenges in exploration for optimal solutions

This work the final project report of course 16-745: Optimal Control and Reinforcement Learning at Carnegie Mellon University, Pittsburgh, PA, USA in Spring 2022

[1]Authors are in alphabet order and all the authors are with Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA {dingkung, xguo2, yunchuz}@andrew.cmu.edu

[2]Our code for this project is available at: https://github.com/Xiaofeng-Guo/16745OptimalControl-project/

and data efficiency. Although there have few relative works [6] that proposed to use goal conditioned policies at multiple layers of hierarchy for RL to tackle these issues, in general current HRL methods still need a lot of online/offline data for training and are not guaranteed to converge.

When we dive into the classical optimal control context, there exists the direct or indirect nonlinear trajectory optimization algorithms. But for relatively long trajectories, it will become super hard for the solver to find the optimal solutions. Also, those methods will assume we know the system dynamics model, which will be hard to get for some black box systems.

III. PRELIMINARIES

In this section, we introduce some background knowledge in optimal control theory that helps understandings of later sections. Here we provide an overview of one of the most common used methods in control theory, Linear Quadratic Regulator (LQR), and it's derivatives, time-varying LQR (TVLQR). We then introduce a usually used nonlinear trajectory optimization method, iterative LQR (iLQR), which we use to smooth and optimize whole trajectories in later sections.

A. Linear Quadratic Regulator (LQR)

LQR is a widely used deterministic optimal control algorithm, which could be used to solve a linear or locally linear control problem without constrains. When the system dynamics is linear and objective we try to optimize is in quadratic form, we will have the closed form solution based on Riccati equation. When the extra constrains are needed in the optimization problem (such as torque limit, joint limit), the purely LQR will not work, instead, convex Model predictive control (MPC) method would be a good choice.

B. Time-varient LQR (TVLQR)

When system dynamics and cost function are different at each time step, we can use time-varying LQR to linearize around a nominal trajectory of a nonlinear system and using LQR to provide a trajectory controller. With linearized dynamics systems and the cost function formulated as the quadratic loss, we can find an optimal feedback gain K matrix, which is robust to system disturbance and error. Specifically, if we calculate a series of different K on each time step for tracking a reference trajectory due to the change of dynamics systems, and the formulation for K in discrete-time is:

$$K_k = (R + B_k^T P_{k+1} B_k)^{-1}(B_k^T P_{k+1} A_k + Q_N^T) \quad (1)$$

Then we can use this K to find the optimal controller and track the reference trajectory in real time.

C. Iterative LQR (iLQR)

To optimize a whole control sequence rather than just the current timestep's control signal, iLQR is able to plan out and optimize a long-horizon trajectory. The algorithm basically includes a main loop to iteratively minimize cost. For each step, there are a forward pass to simulate the system with control sequence U and a backward pass to estimate the value function and dynamics. Then a new $\hat{U}$ is calculated with updated gain K and set $U = \hat{U}$ if the new trajectory has less cost. Furthermore, when extra constrains are required, we need to impose the augmented Lagrangian framework and reformulate the equations to AL-iLQR as shown in [4]. For our current project, however, since we do not add any other extra constrains in our formulation other than system dynamics, we just follow the vanilla iLQR's formulation that does not have constraints.

IV. PROBLEM FORMULATION

To solve long-horizon manipulation tasks, we formulate the problem as two stages: (1) tracking short trajectory and (2) long-horizon trajectory optimization. First we predefined a list of short manipulation tasks that robot can perfectly track. Then we can combine multiple short actions into long-horizon manipulation tasks and smooth the whole trajectory.

A. Symbols

We formulate our whole manipulation process $M = (X, U, F)$ to be a finite-horizon fully observable Markov decision process (MDP), where X and U are states and action spaces, $F(X_{t+1}|X_t, U_t)$ is our system's forward dynamics model. In our problem setup, robot states X contain joint angles and joint velocities; actions U are torques applied on each joints.

B. Short Manipulation Actions

With desired short horizon manipulation tasks' goal in the environment, we could design and calculate the positions of robot end effector in cartesian space at each time step and use inverse kinematics to find corresponding states X_{ref} we needed, which is the reference trajectory. As our system dynamics is time-varient, we decide to use TVLQR to let the robot reach desired states. To find reference control U_{ref}, we implement a coarsely tuned PID controller so the robot can roughly track X_{ref}. Then to find system dynamics Jacobian matrix A at each time step t, we set the robot to $X_{ref}(t)$ with control signal $U_{ref}(t)$ and perturb each joint to calculate Jacobian. The same perturbation method is applied on control signal $U_{ref}(t)$ to calculate control Jacobian matrix B. Lastly, we can set up TVLQR to minimize the cost:

$$\begin{aligned} J =& \frac{1}{2}(X(N) - X_{ref}(N))^T Qn(X(N) - X_{ref}(N)) \\ &+ \frac{1}{2}\sum_{k=1}^{N-1}[(X(k) - X_{ref}(k))^T Q(X(k) - X_{ref}(k)) \\ &+ (U(k) - U_{ref}(k))^T R(U(k) - U_{ref}(k))] \end{aligned} \quad (2)$$

and the solution will provide us the optimal X and U which has lower cost and more robust to noise compared to PID controller.

C. Long-Horizon Trajectory Optimization

For solving the long-horizon manipulation tasks, we formulate them as a nonlinear trajectory optimization problem. We simply concatenate all TVLQR trajectories and the transferring trajectory between two subtask in sequential order to generate the Xref, Uref. Then the iLQR solver will minimize the following cost and optimize the reference trajectory till it meets the converge criteria. The iLQR will firstly run with backward pass to update and calculate the gradient and hessian, and then it will do a forward pass with line search to move forward with the properly step size. The detailed algorithms are shown as below:

$$\begin{aligned} J =& \frac{1}{2}(X(N) - X_{ref}(N))^T Qn(X(N) - X_{ref}(N)) \\ &+ \frac{1}{2}\sum_{k=1}^{N-1}[(X(k) - X_{ref}(k))^T Q(X(k) - X_{ref}(k)) \\ &+ (U(k) - U_{ref}(k))^T R(U(k) - U_{ref}(k))] \end{aligned} \tag{3}$$

Algorithm 1: Backward pass

Input: BACKWARDPASS(X, U)
Output: K, d, ΔJ

```
p_N = Q_f(X[N] − Xref[N)
P_N = Q_f
for k = 1 to N do
    A_(n) ← R_n left singular matrix of X_(n)
end
G =← X × A^T_(1) × A^T_(2)..... × A^T_(N)
return G, A_(1), A_(2).....A_(N)
```

Algorithm 2: Forward pass

Input: HOSVD($\mathcal{X}, R_1, R_2.....R_N$)
Output: $\mathcal{G}, A_{(1)}, A_{(2)}.....A_{(N)}$

```
for k = 1 to N do
    A_(n) ← R_n left singular matrix of X_(n)
end
G =← X × A^T_(1) × A^T_(2)..... × A^T_(N)
return G, A_(1), A_(2).....A_(N)
```

V. EXPERIMENTS

In simulated environment, we demonstrate how TVLQR and iLQR can be used to help robot solve long-horizon manipulation takes. This section includes the simulated environment setup in Section V-A, our choice of short actions in Section V-B, and how to generate trajectories for experiments in Section V-D.

A. Simulation Environment

To simulate the complex environment of kitchen, we use the FrankaKitchen domain based on the Adepth environment [3], which is uses the OpenAI Gym API. Fig. 1 shows the the overview of the simulated kitchen environment. This simulated environment provides multiple interactive objects, such as the microwave, kettle, light button, etc. Based on these objects, it also offers offers a challenging manipulation problem in an unstructured environment with many possible tasks to perform.

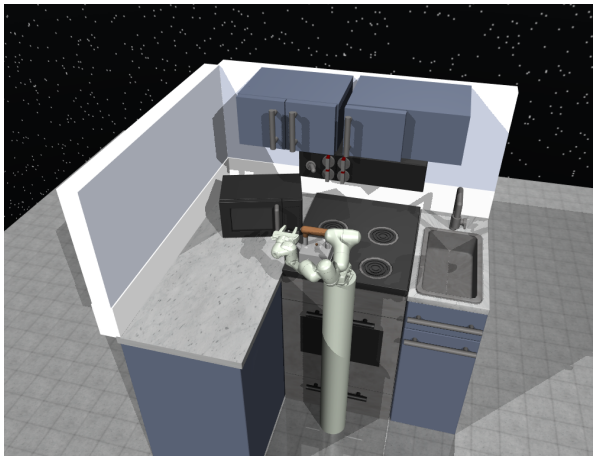
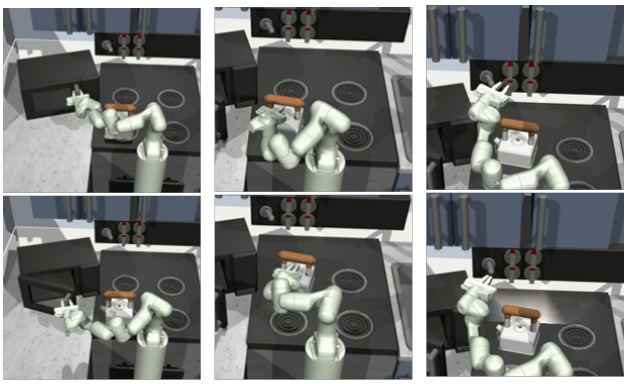


Fig. 1. The FrankaKitchen Simulation Environment. The FrankaKitchen domain is based on the Adept environment. This domain offers a challenging manipulation problem in an unstructured environment with many possible tasks to perform.



(a) (b) (c)

Fig. 2. A set of predefined actions. Examples of actions include (a) opening the microwave, (b) lifting and placing the kettle, and (c) turning on the oven light. In our demonstration, the robot completes these three actions in order to form a long-horizon manipulation task.

B. Action Definition

As a set of predefined actions that can be combined into a long-horizon manipulation task, we choose some common activities in kitchen: 1) open/close microwave, 2) lift/place kettle, and 3) turn on/off switch. These actions can be performed in the simulated kitchen environment and the illustration of these actions is shown in Fig. 2.

C. Franka Robot

In the simulation, we use the Franka Emika Panda robot arm, which has 7 degrees-of-freedom (DoF). There is also a 2-DoF gripper attached to it. We can only control the torque at each joint and get the observation of joint angle from simulation. In this case, the states X contain 18 states: seven joint angles, two gripper positions and seven joint velocities plus two gripper velocities; actions U are torques applied on each joints and gripper.

In addition, to better simulate the real environment, there is noise in the observations of joint angles and velocities of

the robot. Therefore, simple control strategies that are not robust to noisy feedback cannot be employed here.

D. Trajectory Generation

As mentioned in Section IV-B, we design and calculate the positions of robot end effector and use inverse kinematics to find joint angles of the robot. In our experiment, to ensure fair comparison with the reinforcement learning method, d4rl [3], we break down one of long-horizon manipulation tasks used in their paper into three actions (opening microwave, lifting and placing kettle, and turning on oven light). We then apply our methods to track and optimize the trajectory, and we show our results comparison in Section VII.

VI. RESULTS

A. PID vs TVLQR

The cost using TVLQR is around 1518.91, while using PID is around 1662.86.

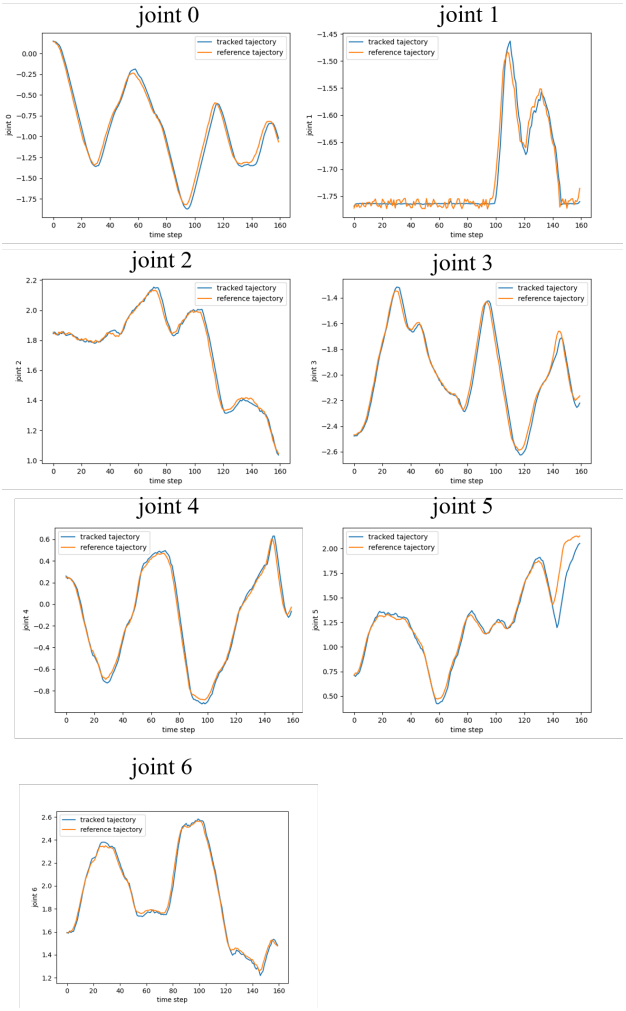


Fig. 3. Caption

B. TVLQR vs iLQR

The cost using iLQR is around 1419.75.

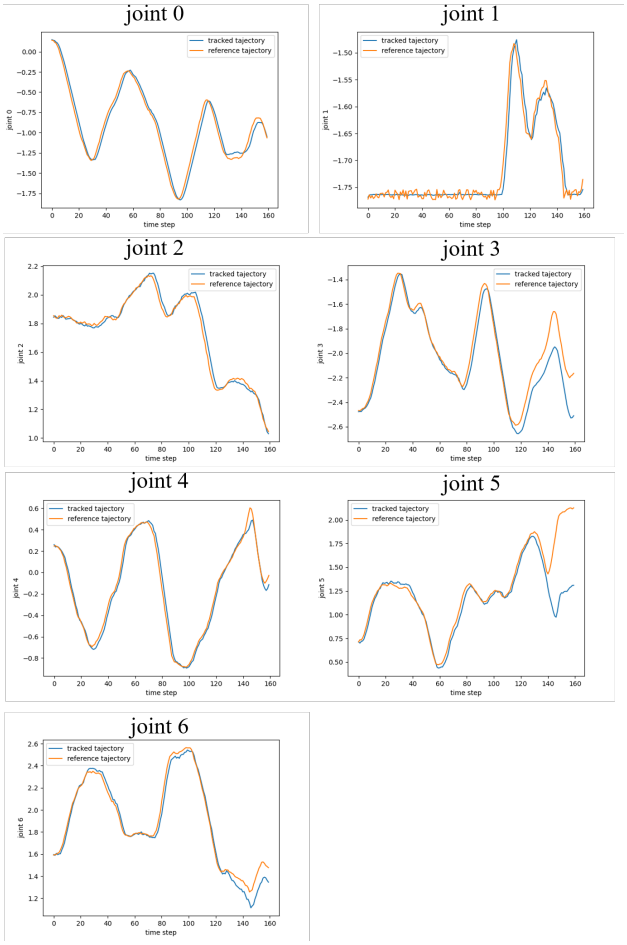


Fig. 4. Caption

VII. DISCUSSION AND FUTURE WORK

A. our subtasks vs RL

B. our iLQR vs demo trajectory

C. Future Work

1) Motion Planning: Although currently we could obtain the transferring trajectory between two subtasks by trimming the super long demonstrations from d4rl dataset, we will still need to get it with a more general motion planner which could output a reasonable transferring trajectory given the start and end configurations in the joint space followed by some constraints. Rapidly-exploring Random Tree (RRT) is potentially a good planner that meets all the requirements we described above, and it could efficiently search in any high-dimensional spaces by randomly filling valid tree nodes and connect those nodes till reaching the goal configuration.

2) More diverse long horizon tasks: We will firstly enrich our library of primitives with more short horizon tasks' controllers and then try to solve more self-constructed long horizon tasks which could be obtained by randomly sampling few subtasks from the library we have. Then we will utilize the planner we mentioned above to obtain transferring trajectories and concatenate those trajectories with primitives' trajectories together as our iLQR's solver input.

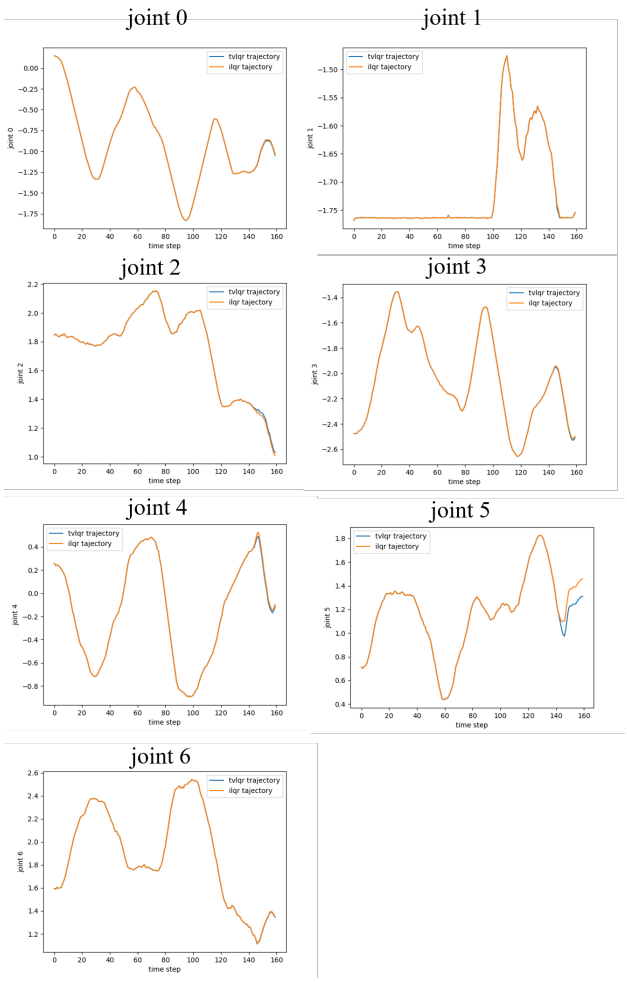
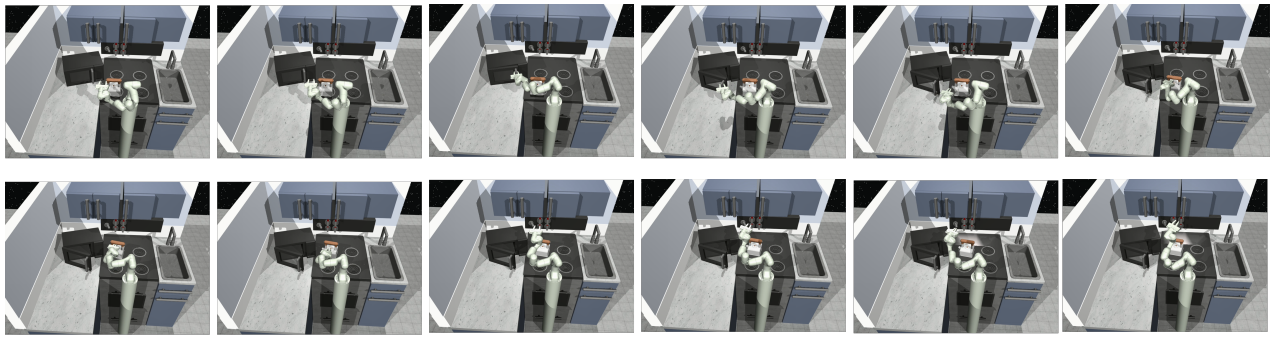
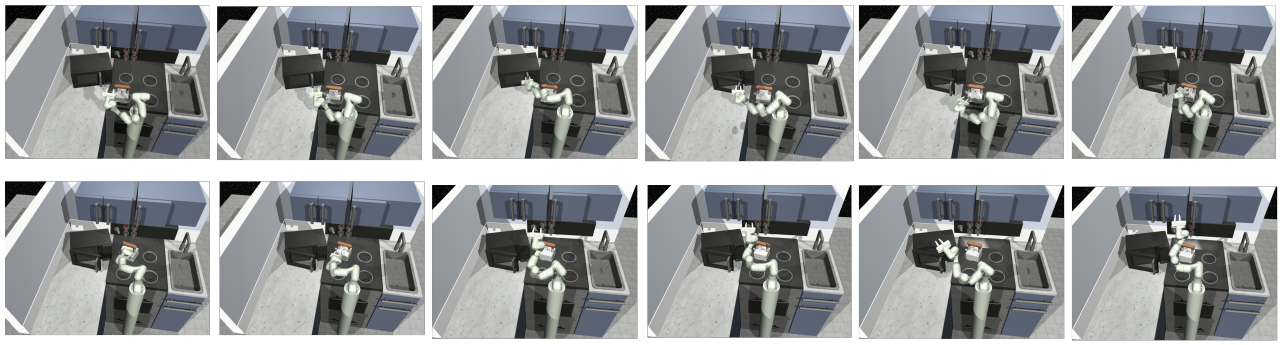


Fig. 5. Caption

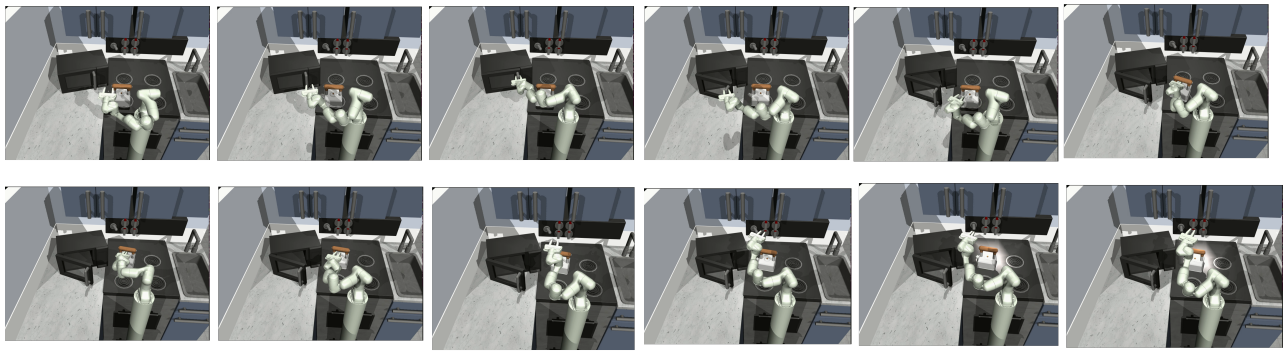
VIII. CONCLUSIONS



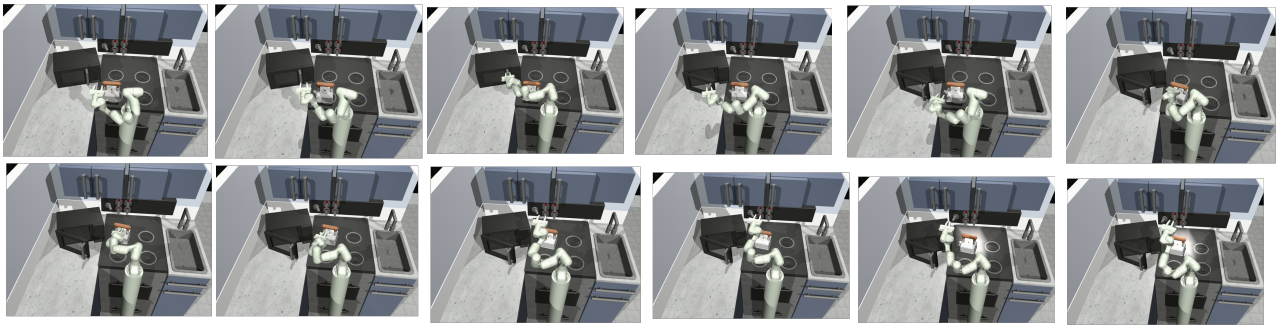
demonstration



tracking by PID



tracking by TVLQR



optimized by iLQR

Fig. 6. Caption

REFERENCES

[1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[2] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, no. 1, pp. 41–77, 2003.

[3] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arXiv preprint arXiv:2004.07219*, 2020.

[4] B. E. Jackson, "Al-ilqr tutorial." [Online]. Available: https://bjack205.github.io/papers/AL_iLQR_Tutorial.pdf

[5] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *ICINCO (1)*. Citeseer, 2004, pp. 222–229.

[6] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," *Advances in neural information processing systems*, vol. 31, 2018.

[7] J. Yang, H.-Y. Tung, Y. Zhang, G. Pathak, A. Pokle, C. G. Atkeson, and K. Fragkiadaki, "Visually-grounded library of behaviors for manipulating diverse objects across diverse configurations and views," in *5th Annual Conference on Robot Learning*, 2021.