Behavior Cloning of MPC for 3-DOF Robotic Manipulators

Theo Guegan
21229606
University of Waterloo
tguegan@uwaterloo.ca

Wen Jie Dexter Teo
21230211
University of Waterloo
d2teo@uwaterloo.ca

***Abstract*—While MPC controllers offer strong stability and robustness, they can be computationally expensive for real-time systems and resource-constrained devices. This paper investigates the application of behavior cloning to approximate Model Predictive Control (MPC) policies for real-time control of a 3-degree-of-freedom (3-DOF) robotic manipulator. We present a baseline controller combining inverse kinematics (IK) with MPC and evaluate multiple neural network architectures such as feedforward networks and recurrent neural networks (RNNs) to learn computationally efficient surrogate policies. We analyze generalization capabilities, stability considerations, and trade-offs between different architectural choices. Our empirical study relies on both online and offline evaluation to measure the performance in terms of accuracy, computational efficiency, and ability to reproduce the original MPC policy.**

I. Introduction

Model Predictive Control (MPC) has been widely used for robotic manipulation [1], offering an optimal control strategy with strong stability and robustness. However, the computational cost of MPC for solving the optimization problems limits its applicability for both real-time systems and resource-constrained devices. Neural networks may offer a promising and computationally efficient alternative for approximating MPC policies with different architectures [2].

We consider a 3-degree-of-freedom (3-DOF) robotic manipulator operating in a MuJoCo simulation environment. The simulation environment provides a realistic and controllable environment for testing and evaluating the proposed methodology. MuJoCo also handles gravity compensation and joint friction, allowing us to simplify the control problem and focus on the learning aspect. The control objective centers on driving the end-effector (EE) to reach a 3D cartesian target position within the robot's reachable workspace.

Inspired by the recent usage of imitation learning for complex controls [3], we present a complete data generation pipeline for collecting high-quality demonstrations of the desired behavior and an empirical evaluation of both feedforward and recurrent neural networks for policy learning. Our experiment focuses on minimizing the control error and testing the ability of the learned policy to generalize in the simulation environment.

II. Problem Formulation

A. System Description

We consider a 3-degree-of-freedom (3-DOF) robotic manipulator defined by generalized coordinates $q = [q_1, q_2, q_3]^T \in \mathbb{R}^3$, representing joint angles, and their time derivatives $\dot{q} \in \mathbb{R}^3$. The full observable state at discrete time step k is

$$x_k = \left[q_k^T, \dot{q}_k^T\right]^T \in \mathbb{R}^6 \tag{1}$$

The manipulator operates in a MuJoCo simulation environment (Fig. 1) governed by rigid-body dynamics with gravity compensation. The control objective is to drive the end-effector (EE) to track randomly sampled, reachable 3D Cartesian target positions $p_{\text{des}} \in \mathbb{R}^3$ within the robot's workspace $\mathcal{W} \subset \mathbb{R}^3$.

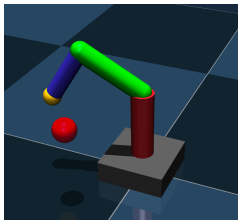

Fig. 1. 3-DOF Arm in MuJoCo

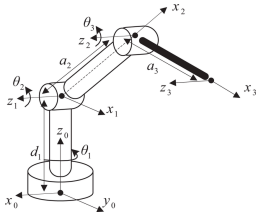

Fig. 2. 3-DOF Arm Schema [4]

III. Baseline Controller : MPC with Inverse Kinematics

Our baseline controller uses a hierarchical architecture combining an Inverse Kinematics (IK) module and a Model Predictive Control (MPC) module. The IK module computes the joint angles required to achieve the desired end-effector position, while the MPC module optimizes the joint velocities to minimize the control error.

A. Inverse Kinematics Formulation

The IK module translates desired end-effector positions into feasible joint-space configurations. Let $p(q) : \mathbb{R}^3 \to \mathbb{R}^3$ denote the forward kinematics mapping. The Cartesian error is defined as :

$$e = p_{\text{des}} - p(q) \tag{2}$$

We solve the IK problem using the Jacobian transpose method with Damped Least Squares (DLS) for numerical stability near singularities. The iterative update rule is

$$\Delta q = J^T \left(JJ^T + \lambda^2 I\right)^{-1} e \tag{3}$$

With $J(q) = \mathrm{d}(\partial p, \partial q) \in \mathbb{R}^{3\times 3}$ is geometric Jacobian and λ is the damping factor.

To prevent overshooting or divergence, the joint update is clamped to a maximum norm relative to the step size $\alpha \in [0, 1]$:

$$\Delta q = \begin{cases} \Delta q \text{ if } \|\Delta q\| \leq \alpha \\ \Delta q * \frac{\alpha}{\|\Delta q\|} \text{ otherwise} \end{cases} \tag{4}$$

Finally, joint angles are wrapped to avoid numerical drift:

$$q_i \leftarrow \mathrm{atan2}(\sin(q_i), \cos(q_i)) \tag{5}$$

B. Model Predictive Control Formulation

The MPC modules is given the desired joint angles $q_{\mathrm{des}} \in \mathbb{R}^3$ from the IK module and computes optimal control torques τ_{MPC} with a specified prediction horizon. We can simplify our system dynamics and represent it as a simplified double-integrator model as MuJoCo is used to compensate dynamics including gravity or joint friction.

Our simplified dynamic system can be defined as :

$$\ddot{q} = \tau_{\mathrm{MPC}} \tag{6}$$

With $x = [q, \dot{q}]^T \in \mathbb{R}^6$ and discrete-time dynamics :

$$x_{k+1} = x_k + \Delta t * \begin{bmatrix} \dot{q}_k \\ \tau_{\mathrm{MPC,k}} \end{bmatrix} = f(x_k, \tau_k) \tag{7}$$

where $x = [q, \dot{q}]^T \in \mathbb{R}^6$

The MPC solves a finite-horizon optimal control problem with quadratic cost function :

$$\min_{\tau_{0:N-1}} \quad \sum_{k=0}^{N-1} \left(\|x_k - x_{\mathrm{ref}}\|_{\mathrm{Q}}^2 + \|\tau_k\|_{\mathrm{R}}^2 \right) + \|x_N - x_{\mathrm{ref}}\|_{\mathrm{Q_N^2}} \tag{8}$$

subject to :

$$x_{\mathrm{k+1}} = f(x_k, \tau_k), \quad x_0 = x(t), \quad \tau_{\min} \leq \tau_k \leq \tau_{\max} \tag{9}$$

Where $x_{\mathrm{ref}} = \left[q_{\mathrm{des}}, 0^T\right]^T$ is the target state, and $\boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{Q_N}$ are positive definite matrices. The optimization problem is solved using CasADI [5] with IPOPT [6] optimization solver.

IV. Data Generation Pipeline

To enable behavior-cloning from the expert IK-MPC controller, we generate a dataset of joint angles, joint velocities, target, and predicted torques from the closed-loop MuJoCo simulation. The process for each episode is as follows:

A. Collection process

1) Target sampling: A reachable end-effector target $p_{\mathrm{des}} \in \mathbb{R}^3$ is sampled within the workspace $\mathcal{W}$, for this purpose we use cylindrical coordinates to sample a radius r and height z uniformly within the maximum workspace dimensions.
2) The IK solver computes the corresponding joint-space reference q_{des}.
3) The MPC controller generates torque commands τ_{MPC} to achieve the desired joint angles and velocities, given the current state x_k and a specified prediction horizon N.
4) For each time step k, we record the current state $[q_1, q_2, q_3, \dot{q}_1, \dot{q}_2, \dot{q}_3]$, the target p_{des}, and the predicted torque τ_{MPC}.
5) We step the simulation until the end of the episode or until the target is reached using $\tau = \tau_{\mathrm{MPC}} + \tau_{\mathrm{env}}$ (with τ_{env} from MuJoCo bias force : mjData.qfrc_bias).

During this process, if either the MPC controller or the IK solver fails to converge, we discard the data for that time step to keep only high-quality data.

B. Dataset Structure

After generation, the dataset is stored in an episode-based format within a HDF5 file.

episodes/
- ep_0000/
  - states : (T_0 × 6)
  - targets : (T_0 × 3)
  - actions : (T_0 × 3)
- ep_0001/
  - states : (T_1 × 6)
  - targets : (T_1 × 3)
  - actions : (T_1 × 3)

This format allows us to easily process an episode at a time (fittable for both MLP and RNN achitecture), and to split the dataset into training and validation sets regardless of the episode length.

C. Data Preprocessing

Our goal is to develop a robust and reliable controller which can handle uncertainties and disturbances in the system. For this purpose, we introduce small gaussian noise to both the input state $[q_1, q_2, q_3, \dot{q}_1, \dot{q}_2, \dot{q}_3]$ and the output action τ_{MPC}. This noise helps to simulate real-world conditions, such as sensor noise, actuator noise, and environmental disturbances.

Because our data generation pipeline allows us to generate as many samples as needed, we can easily collect a large dataset for training our neural network. Therefore, for the training process we can increase the number of samples until we reach a plateau in the validation loss or a computational limit. For the splitting of the dataset, we use a 90/10 split, where 90% of the data is used for training and 10% for the validation as done in this paper [3].

V. Neural Network Architecture

We formulate the learning problem as a regression task, where the goal is to predict the torque τ_{MPC} given the current state x_k and the target p_{des}. We want to minimize the error between the neural network policy π_θ and the expert MPC actions :

$$\min_{\theta} \quad L\left(\pi_{\theta(X)}, \tau_{\mathrm{MPC}}\right) \tag{10}$$

where L is a loss function that measures the difference between the predicted torque and the expert MPC torque.

Loss function investigation: A key hyperparameter in our study is the choice of the loss function L. We will conduct a comparative analysis of two primary candidates:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)

MSE heavily penalizes large errors, which can lead to smoother policies but may make the model sensitive to outliers. MAE is more robust to outliers and may lead to more stable training. The final selection will be based on which loss function yields the best offline and online performance across our evaluation metrics.

Inspired by the work of Pon Kumar et al. [7], we investigate 3 primary neural network (NN) architectures to understand the trade-offs between model complexity, temporal awareness, and performance:

1) Feedforward Network (NN-only): This architecture serves as our baseline. It is a memory-less controller which "captures the MPC response based on current control actions and current outputs by discarding the past" [7]. For our problem, the input is the concatenated vector $x_k = [q_k, \dot{q}_k, q_{\text{des}}]$, which is mapped directly to torque through multiple fully-connected layers. This vector incorporates the current state and target.
2) Recurrent Neural Network (LSTM-only): To capture the temporal dependencies inherent in the robotic system's dynamics, we employ a recurrent architecture based on Long Short-Term Memory (LTSM) units. This controller "captures the dependency of u_{t+1} on the past inputs, outputs and set-points" [7]. The network takes a sequence of these state-target vectors as input and maps the final hidden state to the action space.
3) LSTM-Supported Feedforward Network (LSTMSNN): This is a hybrid architecture which combines the benefits of the previous two. As described in [7], its "output is a weighted combination of an LSTM-only controller output and the NN-only controller output". This will allow us to implement a neural network which can "effectively learn an optimal control action given past and current inputs, outputs and set-points". We adapt this structure by using the same input definitions as above for the two pathways.

VI. Evaluation Methodology

A. Metrics

We evaluate the learned policies using a combination of offline and online metrics:

1) Offline Metrics (on the test dataset):
   - Mean Absolute Error (MAE) & Root Mean Squared Error (RMSE): Measure the average deviation of the predicted torques from the expert torques.

$$\text{Formula} \tag{11}$$

   - Explained Variance Score: Measures the proportion of variance in the expert's action that is explained by our model. A score of 1.0 indicates perfect prediction.

$$\text{Explained Variance} = 1 - \frac{\text{Var}\left(\tau_{\text{MPC}} - \pi_{\theta(X)}\right)}{\text{Var}(\tau_{\text{MPC}})} \tag{12}$$

   - Direction Accuracy: The percentage of predictions where the sign of each torque component matches the expert's. This assesses whether the model correctly identifies the direction of joint acceleration.

$$\text{Formula} \tag{13}$$

2) Online Metrics (in MuJoCo simulation):
   - Success Rate: The percentage of simulation episodes where the end-effector reaches within an acceptable threshold of the target position.
   - Average Position Error: The mean Euclidean distance between the end-effector and the target throughout the episode. This confirms if the model is behaving in the right way to minimize the error.
   - Computational Time Efficiency: The average inference time of the policy, measured against the baseline MPC to confirm real-time feasibility.
   - Computational Cost: CPU Usage

VII. Results

A. Offline evaluation

B. Online evaluation (MuJoCo)

VIII. Future Work

This work establishes a foundation for behavior cloning of MPC on 3-DOF manipulators, which can be extended in several directions. Firstly, the scalability of the approach should be evaluated on robotic manipulators with higher degrees of freedom (e.g., 6-DOF). This is to assess how the method handles increased state and action space dimensionality. Second, to advance towards real-world deployment, the methodology should be extended to handle more complex control scenarios. It could be interesting to investigate the cloning of a non-linear MPC which is capable of handling complex dynamics.

From a methodological perspective, exploring advanced neural network architectures represents a promising direction. Transformer models, with their self-attention mechanisms, could be investigated for their ability to capture complex, long-range dependencies. Furthermore, the Legendre Memory Unit (LMU) [8], developed at the University of Waterloo, offers a complementary, principled approach to continuous time memory, which may prove to be well-suited for the robotic system's underlying dynamics. Inverse reinforcement learning [3] may prove to be an efficient alternative to learn the underlying MPC cost function.

Acknowledgments

This project is a final project for the course "Foundations of Artificial Intelligence" - SYDE522. We would like to thank our instructor Terry Stewart for his guidance and support.

References

[1] Z. Zhou, Y. Zhang, and Y. Li, "Model Predictive Control Design of a 3-DOF Robot Arm Based on Recognition of Spatial Coordinates." [Online]. Available: https://arxiv.org/abs/2209.01706

[2] C. Gonzalez, H. Asadi, L. Kooijman, and C. P. Lim, "Neural Networks for Fast Optimisation in Model Predictive Control: A Review." [Online]. Available: https://arxiv.org/abs/2309.02668

[3] V. G. de A. Porto, D. C. Melo, M. R. Maximo, and R. J. Afonso, "Imitation learning of a model predictive controller for real-time humanoid robot walking," *Engineering Applications of Artificial Intelligence*, vol. 143, p. 109919, Mar. 2025, doi: 10.1016/j.engappai.2024.109919.

[4] N. Ngoc Son, H. P. H. Anh, and N. Thanh Nam, "Robot manipulator identification based on adaptive multiple-input and multiple-output neural model optimized by advanced differential evolution algorithm," *International Journal of Advanced Robotic Systems*, vol. 14, no. 1, Dec. 2016, doi: 10.1177/1729881416677695.

[5] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, 2018.

[6] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Apr. 2005, doi: 10.1007/s10107-004-0559-y.

[7] S. S. Pon Kumar, A. Tulsyan, B. Gopaluni, and P. Loewen, "A Deep Learning Architecture for Predictive Control," *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 512–517, 2018, doi: 10.1016/j.ifacol.2018.09.373.

[8] A. Voelker, I. Kajić, and C. Eliasmith, "Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, p. . [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/952285b9b7e7a1be5aa7849f32ffff05-Paper.pdf