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 Model Predictive Control (MPC) provides strong stability and robustness, it imposes a significant computational burden on real-time systems and resource-constrained devices. This paper investigates the application of Behavior Cloning to approximate MPC policies for the 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 a spectrum of neural network architectures, ranging from classical regression algorithms to complex deep learning models including Deep MLPs, RNNs, and Transformers to derive computationally efficient surrogate policies. We analyze generalization capabilities, stability considerations, and the trade-offs inherent in different architectural choices. Our empirical study employs both online and offline evaluations to assess performance regarding accuracy, computational efficiency, and fidelity to 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 feed-forward 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 (Figure 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 $\mathcal{W} \subset \mathbb{R}^3$.

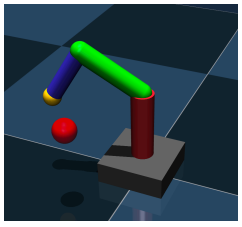

Fig. 1. 3-DOF Arm in MuJoCo

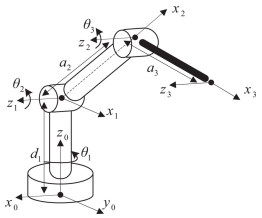

Fig. 2. 3-DOF Arm Schema [4]

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 $J(q) = \mathrm{d}(\partial p, \partial q) \in \mathbb{R}^{3\times3}$ 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 $q_{\text{des}} \in \mathbb{R}^3$ 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 :

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

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

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

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

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

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

subject to :

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

Where $x_{\text{ref}} = [q_{\text{des}}, 0^T]^T$ 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 OSQP 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 $p_{\text{des}} \in \mathbb{R}^3$ 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 $[q_1, q_2, q_3, \dot{q}_1, \dot{q}_2, \dot{q}_3] \in \mathbb{R}^6$, the target $p_{\text{des}} \in \mathbb{R}^3$, and the predicted torque $\tau_{\text{MPC}} \in \mathbb{R}^3$.
5) We step the simulation until the end of the episode or until the target is reached using $\tau = \tau_{\text{MPC}} + \tau_{\text{env}}$ (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: $(T_0 \times 6)$
    - targets: $(T_0 \times 3)$
    - actions: $(T_0 \times 3)$
  - ep_0001
    - states: $(T_1 \times 6)$
    - targets: $(T_1 \times 3)$
    - actions: $(T_1 \times 3)$
  - ...

Listing 1. Hierarchical HDF5 dataset structure

This hierarchical format preserves the temporal integrity of each trial, allowing us to process the data differently depending on the model architecture. The raw data is loaded via a custom MPCDataset class which constructs the input feature vector x by concatenating the state ($\mathbb{R}^6$) and the target ($\mathbb{R}^3$), resulting in a 9-dimensional input vector.

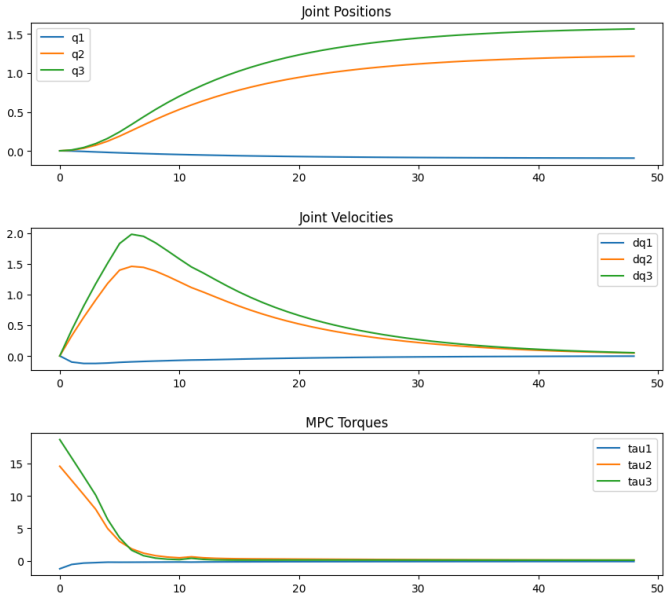

Fig. 3. Visualization of a random episode

Depending on the learning algorithm, the data is processed in two different ways regarding the model architecture.

a) *Flat Formatting:*

For non-sequential algorithms (e.g., MLPs, Random Forests), temporal dependencies are discarded to maximize sample efficiency. We treat every timestep t from every episode as an independent sample (i.i.d).

$$X_{\text{flat}} \in \mathbb{R}^{N\times 9}, \quad Y_{\text{flat}} \in \mathbb{R}^{N\times 3} \tag{10}$$

Where $N = \sum_{i=0}^{E} T_i$ is the total number of timesteps across all episodes.

b) *Sequential Formatting:*

For time-series algorithms (e.g., LSTM, GRU, Transformer), preserving the temporal dependencies is crucial. We treat every episode as a sequence of timesteps, where each timestep is a sample.

$$X_{\text{seq}} \in \mathbb{R}^{E\times T\times 9}, \quad Y_{\text{seq}} \in \mathbb{R}^{E\times T\times 3} \tag{11}$$

Where E is the number of episodes and T is the number of timesteps per episode.

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 80/20 split, where 80% of the data is used for training and 20% for the validation.

V. Neural Network Architecture

We first 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 :

$$\min_{\theta} \quad L\big(\pi_{\theta(X)}, \tau_{\text{MPC}}\big) \tag{12}$$

where L is a loss function that measures the difference between the predicted torque and the expert MPC torque. We investigate a range of models, from traditional machine learning to deep learning architecture, to understand their effectiveness in approximating the MPC. For this task we compare the performance with 2 different loss functions :

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

A. Regression Baselines

As baseline comparisons, we evaluate several models from the Scikit-Learn library. To establish a performance benchmark, we utilize standard implementations from scikit-learn. These models operate on the flat dataset, treating each timestep as independent. First, Random Forest Regressor and Gradient Boosting Regressor which are tree-based algorithms. And second, a shallow MLP Regressor as baseline to compare against deeper custom architectures.

B. Custom Multi-Layer Perceptron (MLP)

We implement a custom feedforward network to explore the impact of model capacity on cloning accuracy. This model processes the flat input vector through a series of fully connected linear layers with ReLU activations. We conduct an architectural search by varying:

- Depth: Number of hidden layers.
- Width: Number of neurons per layer

This memory-less architecture captures and learns directly the mapping from the current state and target to the required control action.

C. Time Series Models

To leverage the temporal structure of our system, we also employ sequence architectures. Unlike the flat models, these architectures maintain a history of past inputs, outputs to predict the current torque [6].

a) *Recurrent Neural Networks (RNNs):*

We evaluate both Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) networks. These models maintain a hidden state h_t that summarizes the history of the episode up to time $t-1$. For all the recurrent models, the final hidden state is passed through a linear output layer to produce the predicted torque $\pi_{\theta(X)}$. Here again, we can vary the number of hidden units per layer and the number of layers.

$$h_t = \text{RNN}(x_t, h_{t-1}), \quad \pi_{\theta(X)} = \text{Linear}(h_t) \tag{13}$$

D. Transformer Architecture

If time permits, we will investigate a Transformer model. Unlike RNNs, the multi-head self-attention mechanisms used to weigh the importance of different past timesteps may result in better performance due to their ability to capture long-range dependencies.

VI. Evaluation Methodology

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

A. Offline Metrics

We used Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to measure the average deviation of the predicted torques from the expert torques.

$$\text{MAE}_j = \frac{1}{N}\sum_{i=0}^{n}\Big|\tau_{\text{MPC,i,j}} - \pi_\theta(X_i)_j\Big| \tag{14}$$

$$\text{RMSE}_j = \frac{1}{N}\sum_{i=0}^{n}\Big\|\tau_{\text{MPC,i,j}} - \pi_\theta(X_i)_j\Big\|_2^2 \tag{15}$$

We can also evaluate the percentage of predictions where the sign of each torque component matches the expert's using Direction Accuracy (DA). This assesses whether the model correctly identifies the direction of joint acceleration.

$$\mathrm{DA} = \frac{1}{3N} \sum_{i=0}^{N} \sum_{j=1}^{3} \mathbb{I}\Big(\mathrm{sign}\big(\tau_{\mathrm{MPC,i,j}}\big) = \mathrm{sign}\Big(\pi_\theta(X_i)_j\Big)\Big) \quad (16)$$

Finally, we measure the proportion of variance in the expert's action that is explained by our model, using explained variance. It's a normalized, scale-invariant metric for comparing performance defined as follows.

$$\text{Explained Variance} = 1 - \frac{\mathrm{Var}\big(\tau_{\mathrm{MPC}} - \pi_{\theta(X)}\big)}{\mathrm{Var}(\tau_{\mathrm{MPC}})} \quad (17)$$

B. Online Metrics

To assess the closed-loop performance, we deploy the trained policies in our simulation environment and evaluate them on the following criteria:

- Success Rate: The percentage of episodes where the end-effector's final position converges within a specified tolerance ε of the target
$$\left\| p_{\text{final}} - p_{\text{target}} \right\|_2 < \varepsilon \quad (18)$$
- Average Position Error: The mean Euclidean distance between the end-effector and the target across the entire trajectory. This verifies that the model actively minimizes error rather than just drifting near the goal.
- Computational Time Efficiency: The average inference latency per control step. We compare this against the baseline MPC solution time to confirm that the neural networks achieve higher control frequencies.
- Computational Cost: Average CPU utilization during operation, ensuring the surrogate model is sufficiently lightweight for potential embedded deployment.

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 more 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) [7], 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] 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.

[7] 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