Behavior Cloning of MPC for 3-DOF Robotic Manipulators

Theo Guegan
21229606, University of Waterloo, Canada
tguegan@uwaterloo.ca

Dexter Teo
University of Waterloo, Canada
@uwaterloo.ca

Abstract

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—including feedforward networks, and recurrent neural networks (RNNs) to learn a surrogate policy. We analyze generalization capabilities, stability considerations, and trade-offs between different architectural choices. The proposed methodology provides a path toward deploying complex optimal control strategies on computationally constrained platforms.

1 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 on the other hand can offer a 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.

2 Problem Formulation

2.1 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 $\mathcal{W} \subset \mathbb{R}^3$.

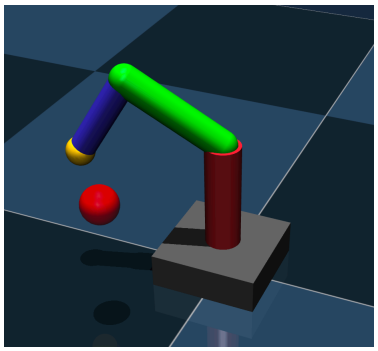

Figure 1: 3-DOF Arm in MuJoCo

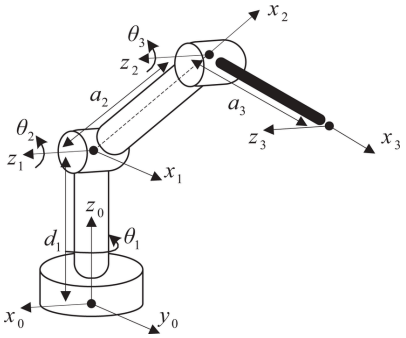

Figure 2: 3-DOF Arm Schema [4]

3 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.

3.1 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:

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

3.2 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 :

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

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 IPOPT [6] optimization solver.

4 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:

1. Target sampling: A reachable end-effector target $p_{\text{des}} \in \mathbb{R}^3$ is sampled within the workspace $\mathcal{W}$.
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 $\tau = \tau_{\text{MPC}} + \tau_{\text{env}}$ (with τ_{env} from MuJoCo bias force : `mjData.qfrc_bias`).

5 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 :

$$\min_{\theta} \quad L\Big(\pi_{\theta(X)}, \tau_{\text{MPC}}\Big) \tag{10}$$

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

We investigate two different loss functions:

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

> From this paper : For the loss metrics, we employed Mean Absolute Error (MAE) for both the regression and classification outputs. Hyperparameter tuning revealed that MAE outperformed other regression loss functions such as mean squared error.
>
> — [3]

3 main architectures to try, from this paper [7] :

- Feed forward network with 1 input state and 1 output state
- RNN with multiple input states and 1 output state (try GRU and LSTM) + FFN support at the end
- Feed forward network with multiple input states and 1 output state (RNN like)

include graph here

6 Evaluation Methodology

6.1 Data augmentation

For each batch, randomly perturb the states and/or actions with Gaussian noise.

Add noise to the input state [q1, q2, q3, $\dot{q}1$, $\dot{q}2$, $\dot{q}3$] before feeding it to your neural network.

1. Simulates sensor noise or slight inaccuracies in state observation.
2. Helps the network generalize to real-world scenarios where states are never perfectly measured.

Add Gaussian noise (e.g., N(0, 0.1)) to the output action (torque) τ_mpc before using it as the target for your network.

1. Simulates actuator noise or imperfections in the control signal.
2. Encourages the network to learn a smoother, more robust policy.

6.2 Metrics

- Computational Efficiency
- Control performance (RMSE, MAE)
- Accuracy in simulation (number of successful simulations)
- Direction accuracy with sign of direction of the torque
- explained variance ? (proportion of variance in expert action explained by the model)

$$\text{Explained Vairance} = 1 - \frac{\text{Var}\left(\tau_{\text{MPC}} - \pi_{\theta(X)}\right)}{\text{Var}(\tau_{\text{MPC}})} \tag{11}$$

7 Results

7.1 Offline evaluation

7.2 Online evaluation (MuJoCo)

8 Future Work

- improving
- extandable to more degree of freedom ? 6-DOF ?
- more complex controller (Non linear MPC for real-life scenarios)
- other methodology :
    - Transfomer or Legendre Memory Unit (LMU) [8]
    - Inverse reinforcement learning (IRL) [3]

9 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