OPTIMIZATION OF MODEL PREDICTIVE CONTROL USING NEURAL NETWORK AND ILQR: AN APPLICATION TO THE MOBILE ROBOT SYSTEM

CONGYAN (CRUISE) SONG
MENTOR: DAO NYGUEN

1. Introduction

Optimal control problems ask for an optimal control that minimizes the known cost function within a dynamical system. They have wide applications in engineering as well as industry, but solving such problems often presented difficulties in computations. There are mainly two categories of the control algorithms: open loop control and closed loop control. Open loop control solves the optimal control problem with a pair of specific initial and terminal conditions; therefore, one must recalculate to find the new optimal control if the conditions change. In contrast, close loop control looks for the optimal policy function that is flexible with the changes of initial conditions. Due to such flexibility, close loop control is more effective in applications compared to the open loop control, but it is also much more difficult to compute, especially in high dimension.

With the advances in the machine learning and AI, solving high-dimensional closed loop control problems become more approachable. The first attempt to apply similar techniques used in image recognition and image generations to solve optimal control problems was made in [2], and it demonstrated the feasibility to solve control problems with dimension up to 50 by applying deep learning-based algorithms.

Our goals in this project are to examine one of the most popular control algorithms called model predictive control (MPC), and to explore the advantages and challenges of deploying machine learning to empower MPC solvers. Lastly, We will define a new system called Mobile Robot System as an example of our approach.

2. Background

Definition 2.1 (Discrete-time Optimal Control Problem)**.**

$$\min_{\{x_k,u_k\}_{k=0}^{T-1}} \sum_{k=0}^{T} L_k(x_k,u_k) + M(x_T) \tag{1}$$

$$\text{s.t. } x_{k+1} = f_k(x_k,u_k)$$

where L_k is the running cost, M is the terminal cost, f_k is the dynamic system, and x_k, u_k will be the state variable and control variable, respectively. For example, if our objective is to drive a car from location A to B over some time interval T with minimum gas usage. Then at time k, x_k will denote the position and speed of the car, u_k will be the control input (could be acceleration and driving direction), the L_k is the current gas usage, and M will be the remaining gas usage to get to destination B.

Model predictive control (MPC) is used for the cases when the control problem has a large time horizon, and its idea is to approximate the solution to global optimal control problems by solving a sequence of auxiliary optimal control problems with short time intervals: "At each time step t, the MPC solvers receives a measured state and solves an optimal control problem whose state trajectory starts from the measured state at t and is defined on a shorter time interval" [1].

Obtaining a good approximation of the value function is one of the main difficulties of MPC formulation. The solution is to use so-called reference trajectory, produced by human expertise or other planning algorithms [3], which aims for a small total cost of the MPC solution in the global optimal control problem.

3. Empower MPC with Machine Learning

Then once we have a better understanding of the MPC, we will move on to applying machine learning techniques in MPC. Here we discuss three ways of empowering MPC with machine learning.

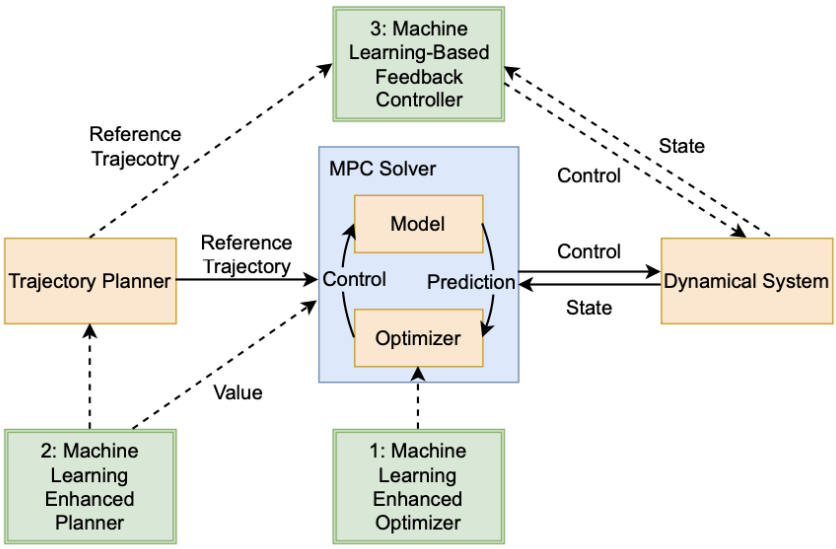


Figure 1. Three ways of enhancing MPC using ML

Firstly, machine learning techniques can enhance the optimizer in the MPC solver, therefore accelerate solving the MPC problem. Attempt has been made to apply the neural-network warm start technique in solving optimal control problems [4]. Secondly, one can use machine learning techniques to enhance the global planner, which is crucial for the MPC's performance. By applying a learning algorithm alongside dynamic programming, [5] demonstrates a way of finding the value function offline and use such approximated value function instead of the heuristic approximation. Lastly, machine learning can be used to approximate the feedback control, originally done by [2].

We will explore three open questions in applying machine learning to empower MPC: **1) How to generate the training data? 2) How to train the machine learning model efficiently? 3) How to use the approximate value or policy function?** As mentioned in [1].

4. MPC Simulator

4.1. **Quadratic Programming Solver.** Our first goal is to implement a MPC simulator using quadratic programming.

Definition 4.1 (Quadratic Programming)**.**

$$\min_z \frac{1}{2} z^T H z + q^T z + r \quad \text{subj. to } Gz \leq w \tag{2}$$

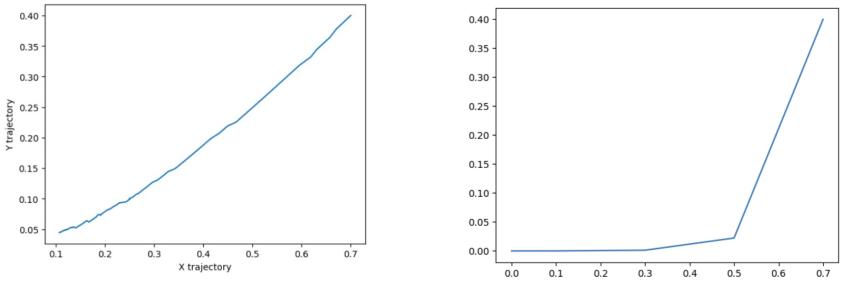
where $z \in \mathbb{R}^s$, $H \in \mathbb{R}^{s \times s}$, $q \in \mathbb{R}^s$, $G \in \mathbb{R}^{m \times s}$.

One can rewrite the equation (1) to be the form of (2), and then the problem could be solved by using quadratic programming solver. We chose to use the QP solver imported from the CVXOPT library in Python for our implementation of a simple MPC simulator. Then we designed some control inputs to test the simulator, results shown below.

4.2. **Iterative Linear Quadratic Regulator.** Although QP solver produces the exact solution, but due to the high computational cost (in some cases, $O(N^3)$), we then turn to some methods that produce approximated optimal solution, but have a faster convergence rate. In this project, we chose to examine the Iterative Linear Quadratic Regulator (iLQR) algorithm.

Definition 4.2 (Iterative Linear Quadratic Regulator (iLQR))**.** The flow of the algorithm is described as follow:

(1) Initialize with initial state x_0 and initial (could be random) control sequence $\mathbf{U}$.
(2) Do a forward pass, i.e. simulate the system using (x_0, $\mathbf{U}$) to get the trajectory .
(3) Do a backward pass, estimate the value function and dynamics for each (x, $\mathbf{u}$) in the state-space and control signal trajectories.



(A) minimal constraints (B) more constraints

FIGURE 2. Figures that shown different trajectory under different constraints. Objective is to travel from lower-left to upper-right in the graph, the results make sense as we can see adding more constraints force us to make a "detour".

(4) Calculate an updated control signal $\hat{\mathbf{U}}$ and evaluate cost of trajectory resulting from $(x_0, \hat{\mathbf{U}})$
   (a) If $|(\text{cost}(x_0, \hat{\mathbf{U}}) - \text{cost}(x_0, \mathbf{U})| <$ threshold then we've converged and exit.
   (b) If $\text{cost}(x_0, \hat{\mathbf{U}}) < \text{cost}(x_0, \mathbf{U})$, then set $\mathbf{U} = \hat{\mathbf{U}}$, and increase the update size. Go back to step (2).
   (c) If $\text{cost}(x_0, \hat{\mathbf{U}}) \geq \text{cost}(x_0, \mathbf{U})$, then decrease the update size. Go back to step (3).

5. Classical Example of Inverted Pendulum System

5.1. **System formulation.** We first attempted to apply to our methods to the quintessential inverted pendulum system. We define the state and control vectors x and u as so:

$$\mathbf{x} = \begin{bmatrix} \theta & \dot{\theta} \end{bmatrix}$$

$$\mathbf{u} = \begin{bmatrix} \tau \end{bmatrix}$$

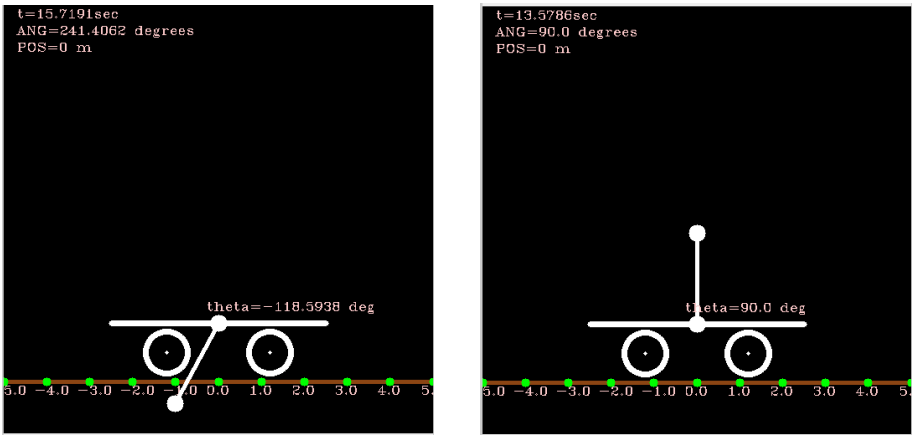
such that $\tau \in [-1, 1]$ and $\theta \in [0, 2\pi]$.

Essentially, the state of the system comprises of the angle of the the pendulum about a pivot at the base and its time derivative. We define our control input as the torque applied at the base of the pendulum (i.e. a motor response).

The goal is to keep the pendulum the upright. Perfectly balanced suggests $\dot{\theta} = 0$ and we define our coordinate system for upright to be $\theta = 0$

$$\mathbf{x}_{goal} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

We then applied the iLQR to the inverted pendulum system, and the simulations are shown below 3.



(A) With random constant control (B) With control produced by iLQR

FIGURE 3. Simulations of the inverted pendulum with different control input, objective is to stay upright.

5.2. **An alternative approach: Neural Network.**

Definition 5.1 (Neural Network)**.** Let us define the following function(unit):

$$a = \sigma(\sum_j w_j x_j + b),$$

where x_j are inputs, w_j weights, b the bias and σ an activation function. A neural network is a combination of these units.

The way we set up the Neural Network model is as followed:

- **Generate training data:**
  (1) **Discretize state space:** To get data that is representative of the state space but also computationally tractable, we create a mesh for the intervals $\theta \in [0, 2\pi]$ and $\dot{\theta} \in [-2\pi, 2\pi]$ parameterized over 20 values in each interval.
  (2) **Generate optimal controls using the iLQR Algorithm:** Using our mesh developed from (a), we run each state as an initial condition to the iLQR algorithm. From each run of the algorithm, we will obtain a list of states $\mathbf{x}_{\theta,\dot{\theta}}$ and corresponding optimal controls $\mathbf{u}_{\theta,\dot{\theta}}$ that we will then merge to form a larger data set.
  (3) **Compile results from iLQR:** From performing (a), (b) over a length of N steps of size $t_{step} = 0.02$, we developed a set of training data consisting of 3000 sets of states with respective optimal control inputs.
- **Construct and Train Neural Network:**
  (1) **Neural net specifications:** Using **PyTorch**, we constructed a Neural Network with 1 input layer, 2 hidden layers, and an

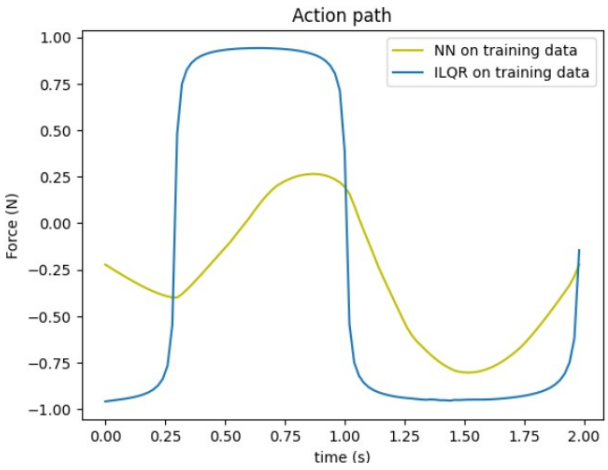


FIGURE 4. Training data

output layer. The activation function for the hidden variables were rectified linear units (ReLUs) as they are generally the recommended function of choice, as used in [2]. We used a batch size of 100 and learning rate of 0.01.

(2) **Train Neural Network:** We trained this neural network without any GPU acceleration. Passing an input $\mathbf{x}_{\theta,\dot{\theta}}$ the neural network generates an output control $\mathbf{u}_{mm}$. We defined our loss as the mean squared error between $\mathbf{u}_{mm}$ and predicted output control $\mathbf{u}_{\theta,\dot{\theta}}$ we generated from the ILQR. We iterated through 1000 epochs, (i.e. we passed the entire training set forward and backward through the NN 1000 times).

- **Benchmark Neural Network**: We evaluated performance through 2 methods:
  (1) Compare iLQR and NN control outputs when given the same initial conditions.
  (2) **Initialize iLQR with NN:** so far we have initialized ILQR through randomly sampling from a uniform distributed interval $[-1, 1]$. We see if using the NN output as an initial control policy allows for faster convergence for iLQR.

5.3. **Results.**

(1) **Comparing ILQR and NN:**
- Comparing against training data 4 and new test data 5
- We see that the NN output controls very generally coincide with iLQR output with training data while it does not align well with test data.
- This can be possibly due to oversights such as over training with the test data or not a sufficiently varied training data set.

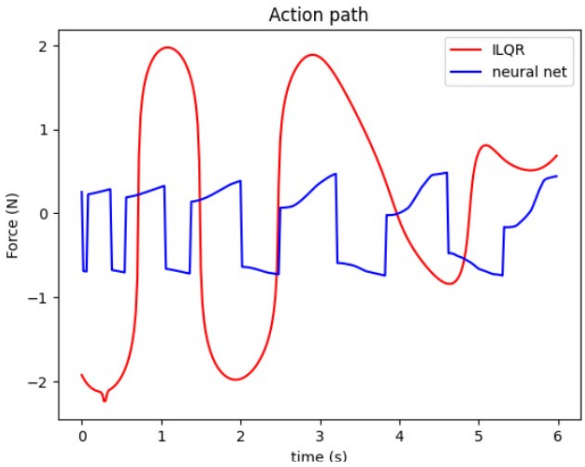


FIGURE 5. Test data

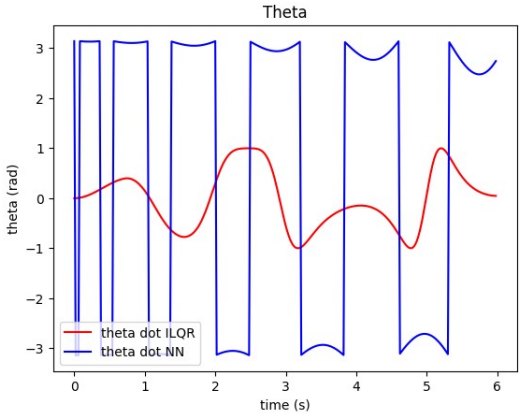


FIGURE 6. Theta

- We can compare the states realized by both the models (θ 6 and $\dot{\theta}$ 7).
- While they both are very different between models, we can see some slight similarities in $\dot{\theta}$ while it seems that θ oscillates very sharply and with little to no correspondence to the iLQR solution.

(2) **Initializing iLQR with NN:** Using the NN controls as an initialize also results in the iLQR algorithm taking much longer to converge than with a randomly sampled initialization. This may be a consequence of the the vast differences in behaviour of the 2 models as shown earlier.

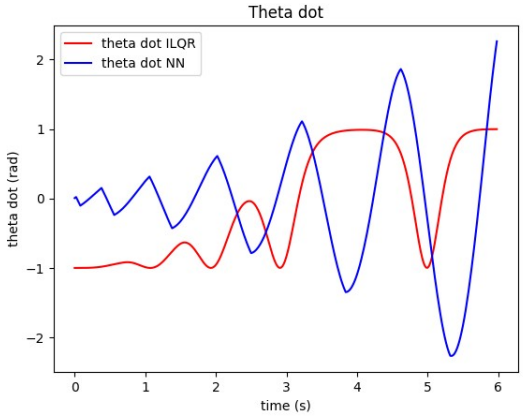


FIGURE 7. Theta dot

6. Mobile Robot System

6.1. **Motivation.** Sweeping process models describe the dynamical processes presented in elastoplasticity and related mechanical areas, providing a convenient framework for handling simulation and related issues in various applications. The mobile robot system, taken from the area of robotics, have dynamics that can be formalized as a perturbed sweeping process, allowing us to describe the controlled dynamics system with ease.

6.2. **System formulation.** We formulate a mobile robot model with obstacles which dynamics can be described as a sweeping process. This model describes n mobile robots ($n \geq 2$) of the same radius R. For each robot, the goal is to reach the target by the shortest path during a fixed time interval $[0, T]$ while avoiding the other $n-1$ robots as obstacles.

Definition 6.1 (Configuration vector)**.**

$$x = (x^1, ..., x^n) \in R^{2n},$$

where $x^i \in R^2$ is the center of ith robot with coordinates $(||x^i||cos\theta_i, ||x^i||sin\theta_i)$, and θ_i denotes the smallest positive angle between the positive x-axis and Ox^i.

Definition 6.2 (Admissible configuration set)**.** To avoid the collision between the admissible configuration set as follow:

$$Q_0 := \{x = (x^1, ..., x^n) \in R^{2n} | D_{ij}(x) \geq \text{ where } i, j \in \{1, ..., n\}\},$$

With those definitions, we then define the cost functional to be

$$\min_u J[x, u] := \frac{1}{2}||x(T)||^2,$$

with $x \in Q_0$, which implies the goal of the model to minimize the distance between the robots and the target at origin O at the terminal time T.

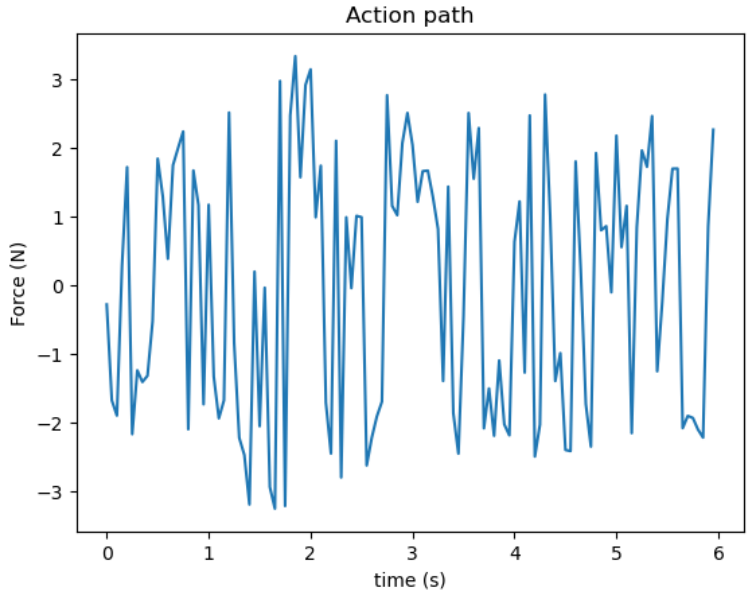


FIGURE 8. Action path for mobile robot model

Each robot was assigned with its own spontaneous speed, moving from the initial position to the target. The control, when acting on the robot of interest, can only change the speed of the robot for the purpose of minimizing the distance from terminal position to the target, with no actual physical implications. In the event of the collision, the two robots move to the target with the same speed without changing direction.

6.3. **Results with iLQR.** Again, we implemented the iLQR to find the optimal control, with slight modifications: we imposed an user self-defined order on the controls, which means the iLQR is restricted to only considering one robot at a time, while treating the rest of the robots as obstacles.

It only takes 27 iterations for the iLQR to converge, with a terminal state of $[-3.02299509e{+}01, 6.73779375e{-}08, -3.02299509e{+}01, 6.73779375e{-}08]$ which corresponds to $[x, \dot{x}, y, \dot{y}]$. The convergence rate and the accuracy are both reasonably well, backed by the action path and cost-to-go below, see Firgure 8 and 9.

7. NEXT STEPS

Bounded by the time constraints, there are plenty of improvements we can make. Future works are divided into mainly two categories, one is on the methodology side of integrating iLQR and Neural Network, another is on the application side of further applying our model on the mobile robot system.

- Methodologies:

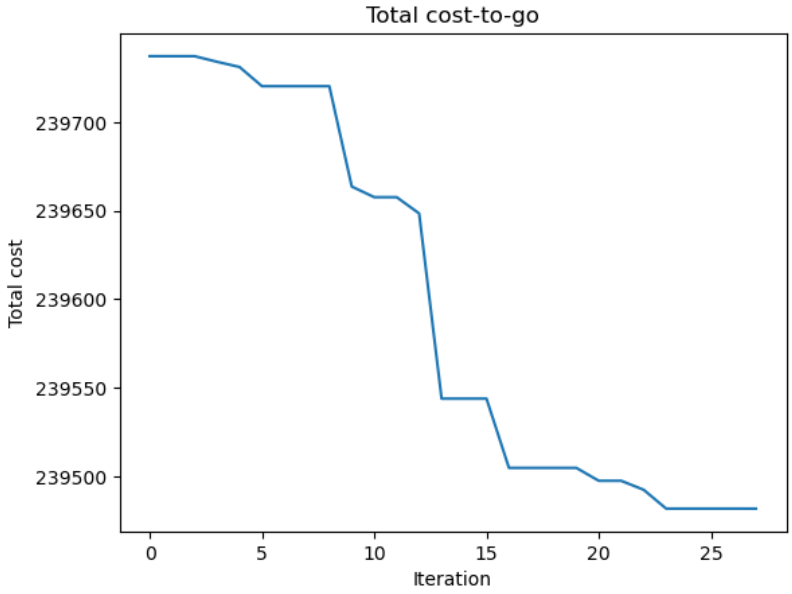


FIGURE 9. Cost-to-go for mobile robot model

(1) Investigate what is required for a sufficient training data set for NN to function feasibly as a controller. Also, determine optimal hyper parameters for NN.
(2) With a fine tuned and designed controller, apply as a solver for the local optimal control problems in MPC and compare against other algorithms used with MPC.

- Applications:
  (1) Instead of considering one robot at a time, remove the order on the controls and optimizes all robots simultaneously.
  (2) Apply the NN model trained by iLQR to the mobile robot system and investigate the convergence rate and accuracy.

REFERENCES

[1] Weinan E, Jiequn Han, and Jihao Long. Empowering optimal control with machine learning: A perspective from model predictive control, 2022.
[2] Jiequn Han and Weinan E. Deep learning approximation for stochastic control problems. `https://doi.org/10.48550/arXiv.1611.07422`, November 2016.
[3] Steven M. Lavalle. *Planning Algorithms*. Cambridge University Press, 2006.
[4] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive deep learning for high-dimensional hamilton–jacobi–bellman equations. *SIAM Journal on Scientific Computing*, 43(2):A1221–A1247, 2021.
[5] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. Value function approximation and model predictive control. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 100–107, 2013.